

République Algérienne Démocratique et Populaire
Ministère de l'enseignement Supérieur et de la recherche Scientifique
Université de Constantine2
Faculté des Technologies de l'Information et de la Communication
Département d'Informatique Fondamentale et ses Applications

Année : 2013

N° d'ordre :/2013

N° de série :/2013

Thèse

Pour l'obtention du diplôme de Doctorat
3ème cycle LMD
Spécialité : Informatique

MÉTHODES DE RÉOLUTION DE PROBLÈMES
DIFFICILES ACADÉMIQUES

Amira Gherboudj

Soutenue le : ../../2013 devant le jury composé de:

Pr. ALLAOUA CHAOUI, Président, Université de Constantine
Pr. SALIM CHIKHI, Directeur de thèse, Université de Constantine
Dr. RAMDANE MAAMERI, Examineur, Université de Constantine
Dr. BAGHDAD ATMANI, Examineur, Université d'Oran
Dr. FODIL CHERIF, Examineur, Université de Biskra

Année universitaire 2012-2013

Remerciements

Je tiens à exprimer en tout premier lieu ma profonde gratitude à mon directeur de thèse, Professeur Salim Chikhi pour m'avoir encadré, orienté et guidé afin de réaliser ce travail. Je le remercie pour ses encouragements continuels qui ne cessaient de me remonter le moral pendant les moments difficiles et pour l'énorme soutien scientifique et moral qu'il a su m'accorder pendant ces trois années.

Je remercie Professeur Allaoua Chaoui pour avoir accepté de présider mon jury. Je remercie également les Docteurs: Ramdane Maameri, Baghdad Atmani et Fodil Cherif, membres du jury d'avoir accepté l'examination et l'évaluation de ce travail.

J'adresse aussi mes vifs remerciements à mon cher père et à ma chère mère pour leur encouragement et le soutien affectif et matériel qu'ils m'ont apporté tout au long de mon existence.

Je remercie aussi mes frères, mes sœurs, mes chers, mes collègues, mes enseignants et tout le personnel du laboratoire MISC, ainsi que toutes les personnes qui m'ont apporté un soutien moral de loin ou de près.

Abstract

The objective of this thesis is to propose methods for solving difficult academic optimization problems. In the aim to achieve our goal, we proposed three key contributions. In the first contribution we proposed four classes of particle swarm optimization algorithms. Each class contains four algorithms inspired from variants proposed in the literature. In the second contribution we proposed a hybrid algorithm based on some principles of the particle swarm optimization algorithm and the crossover operation of the genetic algorithm. The objective of this hybridization is twofold: first, to propose a standard algorithm for solving optimization problems of all types (continuous, discrete or binary), unlike the original algorithm of particle swarm optimization designed for solving continuous optimization problems. Secondly, freeing the user from the phase of selection and adaptation of algorithm parameters by minimizing the number of the essential parameters for the functioning of the algorithm in a single parameter (the size of the population). In the third contribution we thought to fill the gap of the new metaheuristic called "cuckoo search" by proposing its binary version to cope with a wide range of problems that can be modeled by binary structures.

To test the performance of the algorithms that we have proposed, we tried to solve three NP-Hard academic optimization problems as models of difficult optimization problems. The resolved problems are: the one-dimensional knapsack problem (KP), the multidimensional knapsack problem (MKP) and the traveling salesman problem (TSP).

Keywords: Optimization, NP-hard Optimization Problem, Metaheuristic, Particle Swarm Optimization Algorithm (PSO), Cuckoo Search (CS), Hybridization, Knapsack Problem (KP), Multidimensional Knapsack Problem (MKP), Traveling Salesman Problem (TSP).

Résumé

L'objectif de cette thèse est de proposer des méthodes pour la résolution de problèmes d'optimisation académiques difficiles. Dans le but de réaliser notre objectif, nous avons proposé trois contributions essentielles. Notre première contribution consiste à proposer quatre classes d'algorithmes d'optimisation par essaim de particules. Chaque classe contient quatre algorithmes inspirés des variantes proposées dans la littérature. Notre deuxième contribution consiste à proposer un algorithme hybride basé sur quelques principes de l'algorithme d'optimisation par essaim de particules et l'opération du croisement de l'algorithme génétique. L'objectif de cette hybridation est double: premièrement, proposer un algorithme standard permettant la résolution de problèmes d'optimisation de tous types (continu, discret ou binaire), contrairement à l'algorithme original de l'optimisation par essaim de particules conçu pour la résolution de problèmes d'optimisation continus. Deuxièmement, affranchir l'utilisateur de la phase du choix et d'adaptation de paramètres de l'algorithme en minimisant le nombre de paramètres essentiels pour le fonctionnement de l'algorithme en un seul paramètre (la taille de la population). Notre troisième contribution consiste à combler la lacune de la nouvelle métaheuristique nommée « la recherche coucou » par la proposition de sa version binaire pour faire face à une grande gamme de problèmes pouvant être modélisés par des structures binaires.

Afin de tester la performance des algorithmes que nous avons proposés, nous avons essayé de résoudre trois problèmes d'optimisation académiques NP-Difficiles, comme modèles de problèmes d'optimisation académiques difficiles. Les problèmes résolus sont: le problème du sac à dos unidimensionnel, le problème du sac à dos multidimensionnel et le problème du voyageur de commerce.

Mots clés: Optimisation, Problème d'optimisation NP-Difficile, Métaheuristique, Algorithme d'optimisation par essaim de particules, La recherche coucou, Hybridation, Le problème du sac à dos, Le problème du sac à dos multidimensionnel, Le problème du voyageur de commerce.

ملخص

هدفنا من هذه المذكرة هو إقتراح طرق لحل مشاكل التحسين الأكاديمية الصعبة بغية تحقيق ذلك إقتراحنا ثلاث مشاركات علمية أساسية: قمنا في المشاركة الاولى بإقتراح أربعة مجموعات من خوارزمية التحسين باستعمال سرب من الجسيمات، في كل مجموعة أربعة خوارزميات استوحيناها من جملة من الخوارزميات المتركة على خوارزمية التحسين باستعمال سرب من الجسيمات. في المشاركة الثانية قمنا بإقتراح خوارزمية مهجنة مزجنا فيها بعض المبادئ المستوحاة من هذه خوارزمية التحسين باستعمال سرب من الجسيمات وعملية التهجين المستعملة في طريقة التحسين التهجينية. الهدف من هذه الخوارزمية المهجنة مزدوج أولاً: إقتراح خوارزمية عامة تسمح بمعالجة كل أنواع مشاكل التحسين (المستمرة، المتقطعة و الثنائية)، على عكس خوارزمية التحسين الأصلية باستعمال سرب من الجسيمات التي إقترحت لحل مشاكل التحسين المستمرة. ثانياً: إعفاء المستعمل من مرحلة إختيار وضبط معلمات الخوارزمية بتقليص عددها إلى معلم واحد (حجم السرب). أما عن المشاركة الثالثة فقد قمنا من خلالها بإقتراح نسخة ثنائية للخوارزمية الحديثة المسماة "البحث الوقائي". بغية تمكين هذه الأخيرة (البحث الوقائي) من مواجهة وحل جملة من مشاكل التحسين التي يمكن تمثيلها ببنى ثنائية.

بغرض تجريب كفاءة و فعالية الخوارزميات المقترحة قمنا باستعمالها لحل 3 مشاكل كأمتلة عن مشاكل تحسين أكاديمية صعبة. المشاكل التي حاولنا حلها هي مشكل حقيبة الظهر احادية الأبعاد، مشكل حقيبة الظهر متعددة الأبعاد ومشكل التاجر المسافر.

الكلمات المفتاحية: تحسين، مشاكل التحسين الصعبة ن ب، الطرق التقريبية العامة، خوارزمية التحسين باستعمال سرب من الجسيمات، البحث الوقائي، التهجين، مشكل حقيبة الظهر احادية الأبعاد، مشكل حقيبة الظهر متعددة الأبعاد ومشكل التاجر المسافر.

Table des matières	1
Liste des figures	5
Liste des tableaux	7
Liste des algorithmes	8
Liste des abréviations	9
Introduction générale	10
Chapitre 1 : Introduction aux problèmes d'optimisation	16
1.1. Introduction.....	17
1.2. Notions de base en optimisation	18
1.3. La complexité et la théorie de la complexité	22
1.3.1. Les problèmes de la classe P	23
1.3.2. Les problèmes de la classe NP	23
1.3.2.1 Les problèmes NP-Complets	24
1.3.2.2 Les problèmes NP-Difficiles	24
1.3.3. Les problèmes NP-Complets versus les problèmes NP-Difficiles	24
1.3.4. La relation entre les problèmes P et NP	25
1.4. Exemples de problèmes d'optimisation	26
1.4.1. Les problèmes de satisfiabilité booléenne SAT et MAX-SAT	27
1.4.2. Les problèmes d'ordonnancement	27
1.4.2.1. Le problème Job-Shop	28
1.4.2.2. Le problème Flow-Shop	29
1.4.2.3. Le problème Open-Shop	29
1.4.3. La famille du problème du sac à dos	29
1.4.2.1. Le problème du sac à dos	29
1.4.2.2. Le problème du sac à dos multidimensionnel	31
1.4.2.3. Le problème du sac à dos à choix multiple	32
1.4.2.4. Le problème du sac à dos Quadratique	33
1.4.2.5. Le problème du sac à dos Quadratique Multidimensionnel	33
1.4.4. Le problème de tournées de véhicules	34
1.4.5. Le problème du voyageur de commerce	36
1.5. Conclusion	38
Chapitre 2 : Les méthodes de résolution de problèmes d'optimisation	40
2.1. Introduction.....	41

Table des matières

2.2. Les méthodes exactes	45
2.2.1. L'algorithme de retour arrière (Backtracking)	45
2.2.2. La méthode Branch and Bound (B&B)	46
2.3. Les métaheuristiques	47
2.3.1. Les métaheuristiques à base de solution unique	48
2.3.1.1. La recherche locale simple (la descente)	48
2.3.1.2. Le recuit simulé	50
2.3.1.3. La recherche tabou	52
2.3.1.4. La recherche à voisinage variable	54
2.3.2. Les métaheuristiques à base de population de solutions	55
2.3.2.1. Les algorithmes évolutionnaires	56
2.3.2.2. L'algorithme génétique	58
2.3.2.2.1. Le codage des individus	62
2.3.2.2.2. La sélection	64
2.3.2.2.3. Le croisement	65
2.3.2.2.4. La mutation	66
2.3.2.2.5. Le remplacement	66
2.3.2.3. La recherche par harmonies	67
2.3.2.4. L'intelligence par essaim	68
2.3.2.4.1. L'optimisation par essaim de particules	69
2.3.2.4.2. Le système immunitaire artificiel	70
2.3.2.4.2.1. L'algorithme de la sélection clonale	71
2.3.2.4.2.3. L'algorithme de colonies de fourmis	71
2.3.2.4.2.4. L'optimisation par colonies d'abeilles	72
2.3.2.4.5. La recherche coucou	74
2.3.2.4.6. L'algorithme de l'optimisation par coucou	75
2.4. Conclusion	76
Chapitre 3 : Ensemble d'algorithmes d'optimisation par essaim de particules.....	78
3.1. Introduction	79
3.2. L'origine de l'idée de l'optimisation par essaim de particules	80
3.3. L'optimisation par essaim de particules (PSO)	81
3.4. Les variantes de l'algorithme PSO	85
3.5. L'algorithme discret/binaire de l'optimisation par essaim de particules (BPSO)..	87

Table des matières

3.6. L'algorithme quantique de l'optimisation par essaim de particules (QPSO).....	89
3.7. Le PSO avec paramètres adaptatifs	90
3.8. Les applications de l'algorithme PSO	94
3.9. Les classes des algorithmes proposés	98
3.9.1. La représentation de solutions	98
3.9.2. La mise à jour des vitesses et des positions des particules	99
3.9.3. Les algorithmes proposés	100
3.9.3.1. Les algorithmes de la première classe	101
3.9.3.2. Les algorithmes de La deuxième classe	102
3.9.3.3. Les algorithmes de la troisième classe	102
3.9.3.4. Les algorithmes de la quatrième classe	103
3.10. Comparaison et résultats expérimentaux	104
3.11. Conclusion	113
Chapitre 4: Un algorithme hybride basé sur l'optimisation par essaim de particule et l'opération du croisement	116
4.1. Introduction	117
4.2. Le PSO et l'hybridation	118
4.2.1. Le PSO et l'algorithme génétique	118
4.2.2. Le PSO et l'optimisation par colonie de fourmis	119
4.2.3. Le PSO et le recuit simulé	120
4.2.4. Le PSO et la recherche tabou	122
4.2.5. Le PSO et la recherche à voisinage variable	122
4.2.6. Le PSO et le système immunitaire	122
4.2.7. Le PSO et la méthode simplexe	123
4.2.8. Le PSO et la méthode GRASP	124
4.2.9. Autres hybridations	124
4.3. L'algorithme proposé	128
4.4. Application du MHP SO sur le problème du sac à dos multidimensionnel	130
4.4.1. Représentation	130
4.4.2. L'opérateur de croisement	131
4.4.3. L'algorithme de Réparation des Particules (PRA)	132
4.4.4. L'algorithme "Check and Repair Operator (CRO)"	133
4.4.5. Les algorithmes proposés	134

Table des matières

4.4.5.1. Le premier algorithme (MHPSO1)	134
4.4.5.2. Le deuxième algorithme (MHPSO2)	135
4.4.6. Les résultats expérimentaux	137
4.5. Application du MHPSO sur le problème du voyageur de commerce	143
4.5.1. L'algorithme 2-opt	143
4.5.2. Représentation	143
4.5.3. L'opérateur de croisement.....	144
4.5.4. L'algorithme proposé.....	146
4.5.5. Les résultats expérimentaux	147
4.6. Conclusion	151
Chapitre 5 : L'algorithme binaire de la recherche coucou.....	154
5.1. Introduction.....	155
5.2. Le comportement et la reproduction des Coucous	157
5.2.1. L'habitat des coucous	159
5.2.2. La migration des coucous	159
5.3. Les métaheuristiques inspirées Coucou	160
5.3.1. La recherche Coucou (CS)	160
5.3.1.1. Le vol de lévy	161
5.3.1.2. Le principe et les étapes de la recherche coucou	162
5.3.2. L'algorithme d'optimisation par Coucou (COA).....	164
5.3.2.1. Le principe et les étapes du COA	165
5.4. Discussion et comparaison	168
5.5. La méthode proposée: Une version binaire de la recherche coucou	171
5.5.1. L'architecture générale du BCS	173
5.5.2. La représentation binaire de la solution	174
5.5.3. L'algorithme proposé	177
5.6. Les résultats expérimentaux	178
5.7. Discussion.....	184
5.8. Conclusion.....	185
Conclusion générale	187
Bibliographie	190

Liste des figures

Figure 1.1. Courbe représentant les optimums locaux et les optimums globaux.

Figure 1.2. La relation entre les problèmes P, NP et NP-Complets.

Figure 1.3. Exemple de tournées avec 4 véhicules.

Figure 1.4. Exemple de solutions d'un problème de voyageur de commerce.

Figure 2.1. Classification des méthodes de résolution des problèmes d'optimisation.

Figure 2.2. Un schéma d'évolution d'une recherche locale simple.

Figure 2.3. Organigramme d'un algorithme évolutionnaire.

Figure 2.4. Les types des algorithmes évolutionnaires.

Figure 2.5. Démarche d'un algorithme génétique.

Figure 2.6. Codage binaire d'un chromosome.

Figure 2.7. Codage réel d'un chromosome.

Figure 2.8. Codage entier d'un chromosome.

Figure 2.9. Codage à caractère d'un chromosome.

Figure 2.10. Codage arborescent d'un chromosome.

Figure 2.11. Croisement 1-point.

Figure 2.12. Croisement 2-points.

Figure 2.13. Croisement uniforme.

Figure 2.14. La structure de la mémoire harmonique.

Figure 2.15. Organigramme de l'algorithme d'optimisation par essaim de particules.

Figure 2.16. Organigramme de l'algorithme de la recherche coucou.

Figure 2.17. Organigramme de l'algorithme de l'optimisation par la recherche coucou.

Figure 3.1. Déplacement d'une particule.

Figure 3.2. Les algorithmes proposés.

Figure 3.3. Le résultat du test de Friedman entre les algorithmes proposés, le standard BPSO et le standard PSO2006 sur des instances KP.

Figure 3.4. Moyenne de temps d'exécution des algorithmes proposés, le Standard PSO2006 et le Standard BPSO.

Figure 4.1. Un exemple de croisement de deux particules.

Figure 4.2. Organigramme de l'algorithme MHPSO1.

Figure 4.3. Organigramme de l'algorithme MHPSO2.

Figure 4.4. Le résultat du test de Friedman entre l'algorithme MHPSO1 et le PSO-P.

Liste des figures

Figure 4.5. Le résultat du test de Friedman entre le MHPSO1, le MHPSO2, le QICSA et la solution exacte.

Figure 4.6. Exemple d'application de 2-opt.

Figure 4.7. Un exemple du croisement MECH.

Figure 4.8. L'organigramme de l'algorithme MHPSO appliqué sur le problème du voyageur de commerce.

Figure 4.9. Le résultat du test de Friedman entre le MHPSO et la solution optimale.

Figure 4.10. Applications de l'algorithme MHPSO.

Figure 5.1. Ponte aléatoire d'un œuf dans l'ELR du coucou, l'étoile centrale rouge représente l'habitat initial d'un coucou avec 5 œufs, les étoiles roses représentent les nouveaux nids des œufs.

Figure 5.2. Migration d'un coucou vers le meilleur habitat.

Figure 5.3. Les métaheuristiques inspirées coucou

Figure 5.4. L'architecture du BCS.

Figure 5.5. Exemple d'exécution de l'algorithme BSR.

Figure 5.6. Le résultat du test de Friedman entre le BCS et le BPSO sur des instances KP.

Figure 5.7. Le résultat du test de Friedman entre la solution exacte, le BCS, le PSO-P et le QICSA sur des instances MKP.

Liste des tableaux

Tableau 1.1. Nombre de trajets possibles et temps de calcul estimé.

Tableau 3.1. Les applications des algorithmes PSO (Partie 1).

Tableau 3.1. Les applications des algorithmes PSO (Partie 2).

Tableau 3.2. L'étude expérimentale des résultats des algorithmes de la 1^{ère} classe.

Tableau 3.3. L'étude expérimentale des résultats des algorithmes de la 2^{ème} classe.

Tableau 3.4. L'étude expérimentale des résultats des algorithmes de la 3^{ème} classe.

Tableau 3.5. L'étude expérimentale des résultats des algorithmes de la 4^{ème} class.

Tableau 3.6. Comparaison des résultats obtenus par les algorithmes des classes 3 et 4.

Tableau 3.7. Comparaison des résultats obtenus par les algorithmes des classes 3 et 4.

Tableau 3.8. Comparaison de meilleures solutions obtenues par les algorithmes proposés, le Standard PSO2006 et le Standard BPSO.

Tableau 4.1. Les travaux d'hybridations à base de l'algorithme PSO.

Tableau 4.2. Les résultats expérimentaux avec quelques instances tirées de la littérature.

Tableau 4.3. Comparaison des résultats obtenus par le MHPSO1, le MHPSO2 et le PSO-P avec quelques instances de petites tailles.

Tableau 4.4. Les résultats expérimentaux avec mknpcb1 et mknpcb 4 obtenus par NHBPSO et PSO-P.

Tableau 4.5. Les résultats expérimentaux avec mknpcb1 et mknpcb 4 obtenus par NHBPSO et QICSA.

Tableau 4.6. Résultats du MHPSO.

Tableau 4.7. Résultats obtenus par DPSO.

Tableau 4.8. Résultats obtenus par CDPSO.

Tableau 4.9. Résultats obtenus par MHPSO.

Tableau 4.10. Résultats comparatifs entre MHPSO, DPSO et CDPSO.

Tableau 5. 1. Comparaison entre le CS et le COA.

Tableau 5. 2. Les résultats expérimentaux du BCS et NGHS avec quelques instances KP.

Tableau 5. 3. Les résultats expérimentaux du BCS et BPSO avec quelques instances KP.

Tableau 5. 4. Les résultants expérimentaux sur des instances MPK de l'instance mknpcb1.

Tableau 5. 5. Les résultants expérimentaux sur des instances MPK des instances mknpcb1 et mknpcb4.

Liste des algorithmes

- Algorithme 2.1. La recherche locale simple (la descente).
- Algorithme 2.2. Le recuit simulé.
- Algorithme 2.3. La recherche tabou.
- Algorithme 2.4. La recherche à voisinage variable.
- Algorithme 2.5. L'algorithme génétique.
- Algorithme 2.6. L'algorithme général de la recherche par harmonies.
- Algorithme 2.7. L'algorithme de la sélection clonale.
- Algorithme 2.8. L'algorithme de colonies de fourmis pour le TSP.
- Algorithme 2.9. L'algorithme de colonies d'abeilles.
- Algorithme 3.1. L'optimisation par essaim de particules.
- Algorithme 4.1. Un schéma général de l'algorithme MHPSO.
- Algorithme 4.2. Algorithme de Croisement.
- Algorithme 4.3. Algorithme de Réparation d'une Particule (PRA).
- Algorithme 4.4. Algorithme de réparation de solution.
- Algorithme 4.5. Algorithme d'amélioration de la solution.
- Algorithme 4.6. L'algorithme MECH.
- Algorithme 5.1. L'Algorithme de la recherche coucou (CS).
- Algorithme 5.2. L'Algorithme d'Optimisation par Coucou.
- Algorithme 5.3. L'algorithme BSR.
- Algorithme 5.4. L'algorithme proposé (BCS).

Liste des abréviations

ACO: L'optimisation par colonies de fourmis (de l'anglais: Ant Colony Optimization)

BPSO: L'algorithme binaire d'optimisation par essaim de particules (de l'anglais: Binary Particle Swarm Optimization)

COA: L'algorithme d'optimisation par coucou (de l'anglais: Cuckoo Optimization Algorithm)

CS: La recherche coucou (de l'anglais: Cuckoo Search)

FL: La logique floue (de l'anglais: Fuzzy Logic)

GA: L'algorithme génétique (de l'anglais: Genetic Algorithm)

GP: La programmation génétique (de l'anglais: Genetic Programming)

IS: Le système immunitaire (de l'anglais: Immun System)

KP: Le problème du sac à dos (de l'anglais: Knapsack Problem)

LS: La recherche locale (de l'anglais: Local Search)

MA: L'algorithme mimétique (de l'anglais: Memetic Algorithm)

MAM: Le modèle multi agents (de l'anglais: Multi Agent Model)

MKP: Le problème du sac à dos multidimensionnel (de l'anglais: Multidimensional Knapsack Problem)

NMS: La méthode simplexe de Nelder-Mead (de l'anglais: Nelder-Mead Simplex)

PSO: L'optimisation par essaim de particules (de l'anglais: Particle Swarm Optimization)

SA: Le recuit simulé (de l'anglais: Simulated Annealing)

TS: La recherche tabou (de l'anglais: Tabu Search)

TSP: Le problème du voyageur de commerce (de l'anglais: Travelling Salesman Problem)

VNS: La recherche à voisinage variable (de l'anglais: Variable Neighborhood Search)

ILS: la recherche locale réitérée (de l'anglais: Iterated Local Search)

GLS: la recherche locale guidée (de l'anglais: Guided Local Search)

Introduction générale

Quelque soit son domaine, l'être humain est confronté à différents problèmes dans toutes les sphères de la société. Un problème donné peut être défini par l'ensemble des propriétés que doivent vérifier ses solutions. Il peut être un problème de décision ou un problème d'optimisation. Un problème de décision peut se ramener à un problème d'existence de solution. Il consiste à répondre à la question « Est-ce qu'il existe une solution basée sur un ensemble d'énoncés et qui satisfait un ensemble de contraintes ? » par une des réponses: « *Oui il existe* » ou bien « *Non il n'existe pas* ». Par contre, un problème d'optimisation peut se ramener à un problème d'existence de solution de bonne qualité. Il consiste à rechercher une solution de qualité suffisante au regard d'un (des) critère(s) donné(s) et des objectifs à satisfaire. Les problèmes d'optimisation peuvent être partagés en deux catégories: des problèmes académiques et des problèmes industriels. En fait, les problèmes industriels sont des projections de problèmes académiques sur le plan de la pratique.

Un problème d'optimisation combinatoire consiste à parcourir l'espace de recherche (l'ensemble de combinaisons pouvant être prises par les variables du problème) afin d'en extraire une solution optimale parmi un ensemble fini de solutions, d'une taille souvent très grande telle que son énumération exhaustive est une tâche fastidieuse. La résolution d'un problème d'optimisation combinatoire nécessite l'utilisation d'un procédé algorithmique permettant la maximisation ou la minimisation d'une ou de plusieurs fonctions « objectif » en respectant les contraintes posées par le problème. Le nombre de fonctions « objectif » du problème à traiter permet de le classer en problème mono objectif ou problème multi objectifs. En fait, un problème mono objectif consiste à optimiser une seule fonction « objectif ». Tandis qu'un problème multi objectifs consiste à optimiser deux ou plusieurs fonctions « objectif » souvent contradictoires. La résolution de ce type de problèmes (les problèmes multi objectifs) consiste à trouver un compromis entre les différents objectifs imposés.

La complexité d'un problème donné est liée à deux facteurs: le temps de calcul nécessaire pour sa résolution et l'espace mémoire requis. Ces derniers (temps et mémoire) augmentent avec la taille du problème traité. La théorie de la complexité permet de classer les problèmes selon leur complexité en deux classes essentielles: les problèmes P (de l'anglais: Polynomial time) et des problèmes NP (de l'anglais: Non-deterministic Polynomial time). Un problème P est un problème facile à résoudre. Il possède une solution qui peut être trouvée dans un temps

Introduction générale

polynomial avec la taille de l'instance traitée. Tandis qu'un problème NP est un problème difficile à résoudre. Il possède une solution qui peut être trouvée dans un temps exponentiel ou pire qu'exponentiel par rapport à la taille de l'instance traitée.

La résolution de différentes sortes de problèmes a poussé les chercheurs à investiguer dans le domaine de la résolution de problèmes. Par conséquent, de nombreuses méthodes de résolution de différents problèmes ont été proposées dans la littérature. Puisque la performance de méthodes de résolution de problèmes est mesurée en termes de deux notions souvent antagonistes: la rapidité de la recherche et la qualité des solutions fournies, les méthodes de résolution de problèmes ont été classées en deux catégories: les méthodes exactes et les méthodes approchées. Les méthodes exactes sont connues par le fait qu'elles garantissent l'optimalité de la solution mais elles demandent des coûts de recherche (temps de calcul et espace mémoire) souvent prohibitifs qui augmentent avec la taille de l'instance du problème traité. Tandis que, les méthodes approchées sont connues par le fait qu'elles ne garantissent pas l'optimalité de la solution (elles fournissent des solutions de bonne qualité et des fois optimales) mais elles nécessitent des coûts de recherche raisonnables. Par conséquent, les méthodes exactes sont plus pratiques pour la résolution de problèmes faciles. En revanche, les méthodes approchées sont plus pratiques dans le cas où on cherche une solution de bonne qualité dans un bref délai. Particulièrement, pour la résolution de problèmes difficiles dont la résolution exacte est très coûteuse.

Les méthodes approchées sont classées en deux catégories: les heuristiques et les métaheuristiques. Une heuristique est une méthode approchée spécifique à un problème donné. Elle forme un ensemble de règles empiriques ou des stratégies qui fonctionnent, comme des règles d'estimation [Solso, 1979]. Les métaheuristiques sont des méthodes approchées polyvalentes, elles peuvent être appliquées sur de nombreux problèmes. De leur part, les métaheuristiques peuvent être classées en deux catégories: les métaheuristiques à base de solution unique et les métaheuristiques à base de population de solutions. Une métaheuristique à base de solution unique manipule une seule solution durant la procédure de recherche. Elle lance la recherche avec une seule solution et essaye d'améliorer sa qualité au cours des itérations. Cependant, une métaheuristique à base de population de solutions manipule un ensemble de solutions parallèlement. Elle lance la recherche avec une population de solutions et essaye d'améliorer leurs qualités au cours des itérations dans le but de fournir la ou les meilleures solutions trouvées. La majorité de métaheuristiques sont inspirées de systèmes naturels, nous pouvons citer à titre d'exemple: le recuit simulé qui est inspiré d'un

Introduction générale

processus métallurgique, les algorithmes génétiques qui sont inspirées des principes de l'évolution Darwinienne et de la biologie, la recherche tabou qui s'inspire de la mémoire des êtres humains, les algorithmes basés sur l'intelligence par essaim comme l'algorithme d'optimisation par essaim de particules, l'algorithme de colonies de fourmis, l'algorithme de colonies d'abeilles, la recherche coucou et l'algorithme d'optimisation par coucou qui s'inspirent du comportement social de certaines espèces évoluant en groupe.

Les méthodes d'optimisation basées sur l'intelligence par essaim ont construit une tendance très active cette dernière décennie. Elles s'inspirent généralement du comportement collectif de certaines espèces dans la résolution de leurs problèmes. Ces espèces se regroupent en essaim pour construire une force collective qui leur permet de surpasser leurs capacités individuelles très limitées. Comme le cas des oiseaux et des poissons qui se regroupent en essaim pour faire face aux prédateurs ou pour rechercher la nourriture, les fourmis qui se regroupent pour construire leurs nids, rechercher la nourriture et élever les larves, les abeilles qui vivent en colonies dont les membres œuvrent pour le bien du groupe.

La métaheuristique d'optimisation par essaim de particules (de l'anglais PSO: Particle Swarm Optimization) est la métaheuristique principale dans la classe des métaheuristicues basées sur l'intelligence par essaim. Elle a été proposée en 1995 par Kennedy et Eberhart [Kennedy et Eberhart, 1995] pour la résolution de problèmes continus (Nous allons nous étendre sur cette métaheuristique dans le chapitre 3).

L'idée de l'optimisation par essaim de particules, l'étude et l'observation du comportement d'une espèce particulière d'oiseaux ont donné naissance à deux nouvelles métaheuristicues nommées respectivement: la recherche coucou et l'algorithme d'optimisation par coucou. Ces nouvelles métaheuristicues ont enrichi le nombre de métaheuristicues basées sur l'intelligence par essaim. La recherche coucou (de l'anglais CS: Cuckoo Search) a été proposée en 2009 par Yang et Deb [Yang et Deb, 2009] et l'algorithme d'optimisation par coucou (de l'anglais COA: Cuckoo Optimisation Algorithm) a été proposé en 2011 par Rajabioun [Rajabioun, 2011]. Ces chercheurs se sont inspirés du comportement d'oiseaux parasites appelés « Coucous » pour proposer leurs nouvelles métaheuristicues (Nous allons nous étendre sur ces deux métaheuristicues dans le chapitre 5).

L'objectif de cette thèse est de proposer des méthodes de résolution de problèmes d'optimisation académiques difficiles à résoudre. Dans le but de réaliser notre objectif, nous avons commencé nos recherches par une phase de récolte d'informations et de connaissances

Introduction générale

dans le domaine de l'optimisation. Nous avons établi un état de l'art sur les problèmes d'optimisation ce qui nous a permis d'acquérir quelques notions de base dans le domaine. Puis, nous avons essayé d'établir un état de l'art sur les différentes méthodes de résolution de problèmes d'optimisation. Cette étude nous a permis de s'orienter vers la méthode d'optimisation par essaim de particules, qui a prouvé sa simplicité et sa performance. Ainsi, nous avons proposé quatre classes d'algorithmes d'optimisation par essaim de particules, chaque classe englobe quatre algorithmes. Nous avons appliqués l'ensemble des algorithmes proposés sur le problème d'optimisation combinatoire académique NP-difficile du sac à dos unidimensionnel. Ensuite, nous avons proposé un algorithme hybride basé sur quelques principes de l'algorithme d'optimisation par essaim de particules et l'opération du croisement de l'algorithme génétique et nous avons appliqué l'algorithme proposé sur deux problèmes d'optimisation combinatoires académiques NP-difficiles: le problème du sac à dos multidimensionnel et le problème du voyageur de commerce. Enfin, nous nous sommes orientés vers la nouvelle métaheuristique nommée « la recherche coucou » et nous avons proposé sa version binaire pour faire face aux problèmes d'optimisation binaires. Nous avons appliqué notre version sur le problème du sac à dos unidimensionnel et sur le problème du sac à dos multidimensionnel binaires.

Nous présentons notre travail de thèse en 5 chapitres. Les deux premiers chapitres présentent un état de l'art sur les problèmes d'optimisation et les méthodes de résolution de ces problèmes. Les trois autres chapitres présentent nos contributions.

Dans le premier chapitre, nous essayons d'aborder quelques notions de base en optimisation. Nous présentons des définitions de notions confrontées souvent dans le domaine de l'optimisation en allant de la définition d'un problème d'optimisation, passant par les types de problèmes d'optimisation (minimisation mono-objectif, minimisation multi-objectifs, maximisation mono-objectif, maximisation multi-objectifs), jusqu'à la définition d'une solution optimale d'un problème donné. Ensuite, nous dévoilons le rôle de la théorie de la complexité dans la classification de différents problèmes selon leurs complexités en problèmes P et problèmes NP. Nous donnons une vue générale sur les problèmes des deux classes et nous essayons d'éclaircir la relation entre les deux classes. Enfin, nous tentons d'élucider le principe de quelques problèmes d'optimisation académique difficiles à résoudre en donnant une importance particulière aux problèmes du sac à dos et au problème du voyageur de commerce car ils ont été utilisés dans le test de méthodes que nous avons proposées dans le cadre de notre thèse.

Introduction générale

Dans le deuxième chapitre, nous présentons les méthodes de résolution (exactes et approchées) de problèmes d'optimisation proposées dans la littérature.

Dans le troisième chapitre, nous présentons notre première contribution qui consiste à proposer quatre classes d'algorithmes d'optimisation par essaim de particules, chacune contient quatre algorithmes d'optimisation par essaim de particules. Nous présentons d'abord le principe de l'algorithme d'optimisation par essaim de particules, ses variantes et ses applications. Ensuite, nous présentons nos algorithmes inspirés des variantes de l'algorithme d'optimisation par essaim de particules proposées dans la littérature. Enfin, nous présentons les résultats expérimentaux de l'application des algorithmes proposés sur le problème du sac à dos.

Dans le quatrième chapitre, nous présentons notre deuxième contribution qui consiste à proposer un algorithme hybride basé sur quelques principes de l'algorithme d'optimisation par essaim de particules et l'opération du croisement de l'algorithme génétique. L'objectif de cette hybridation est double: premièrement, proposer un algorithme standard permettant la résolution de problèmes d'optimisation combinatoires de tous types (continu, discret ou binaire), contrairement à l'algorithme originel de l'optimisation par essaim de particules conçu pour la résolution de problèmes d'optimisation continus. Deuxièmement, affranchir l'utilisateur de la phase du choix et d'adaptation de paramètres de l'algorithme en minimisant le nombre de paramètres essentiels pour le fonctionnement de l'algorithme en un seul paramètre (la taille de la population). Dans ce chapitre, nous présentons d'abord quelques hybridations basées sur l'optimisation par essaim de particules. Ensuite, nous présentons notre hybridation et son application sur les deux problèmes d'optimisation académiques difficiles: le problème du sac à dos multidimensionnel (en tant qu'exemple de problèmes binaires) et le problème du voyageur de commerce (en tant qu'exemple de problèmes discrets).

Dans le cinquième chapitre, nous présentons notre troisième contribution qui consiste à proposer une version binaire pour la nouvelle métaheuristique « la recherche coucou » qui a été conçue pour la résolution de problèmes d'optimisation continus. Nous commençons par la présentation de l'idée de la métaheuristique « la recherche coucou » et de son principe, ainsi la présentation d'une autre métaheuristique nommée « algorithme d'optimisation par coucou » inspirée du comportement des coucous et qui a été proposée après deux ans de l'apparition de la métaheuristique « la recherche coucou ». Nous établissons une petite comparaison entre les deux métaheuristicues (i.e. la recherche coucou et l'algorithme d'optimisation par coucou). Puis, nous présentons notre version binaire et son application sur les problèmes d'optimisation

Introduction générale

académiques difficiles: le problème du sac à dos unidimensionnel et le problème du sac à dos multidimensionnel. Enfin, nous présentons et nous discutons nos résultats expérimentaux.

A la fin de cette thèse, nous présentons une conclusion générale sur les contributions proposées et des perspectives de nouvelles directions de travaux de recherche futurs.

NB : Dans la suite du document et pour plus de clarté:

La fonction de coût, aussi appelée fonction « objectif » ou fonction « fitness » sera simplement notée: fonction objectif.

CHAPITRE 1

Introduction aux problèmes d'optimisation

Sommaire

1.1	Introduction.....	17
1.2	Notions de base en optimisation.....	18
1.3	La complexité et la théorie de la complexité.....	22
1.4	Exemples de problèmes d'optimisation.....	26
1.5	Conclusion	38

1.1. Introduction

Dans la vie quotidienne, l'hôte humain est souvent confronté à des problèmes dans des domaines différents y compris le transport, la médecine, l'économie, l'industrie, l'énergie, l'éducation, la télécommunication...etc. Quelque soit le domaine du problème confronté, ce dernier doit surement avoir une solution permettant d'en résoudre ou d'en remédier dans le pire des cas. La phase de formalisation du problème est une phase très essentielle elle se positionne en amont des autres phases de résolution de n'importe quel problème. En fait, avant de se pencher dans la résolution d'un problème donné, il faut d'abord penser à en définir en tenant compte de données disponibles et de contraintes posées. Un problème P peut être défini par l'ensemble des propriétés que doivent vérifier ses solutions. Plus formellement, un problème P est défini comme une relation binaire sur un ensemble E d'instances du problème et un ensemble S des solutions [Cormen et al, 1994]. Où, l'instance I d'un problème P est une spécification des valeurs particulières pour tous les paramètres du problème. Deux types de problèmes peuvent être distingués: des problèmes de décision et des problèmes d'optimisation. Un problème de décision peut être transformé en un ensemble de problèmes d'existence. Tandis qu'un problème d'optimisation peut être transformé en un ensemble d'existence d'une solution de qualité suffisante [Garey et Johnson, 1979]. En effet, un problème de décision est une classe d'énoncés auxquels on doit répondre par oui ou par non [Xuong, 1992]. Par contre, un problème d'optimisation consiste à rechercher une solution de bonne qualité au regard d'un (des) critère(s) donné(s) et des objectifs à satisfaire.

Le domaine de l'optimisation combinatoire est un domaine très important, il est situé au carrefour de la recherche opérationnelle, mathématique et informatique. Son importance s'explique par la grande difficulté posée par les problèmes d'optimisation [Papadimitriou et Steiglitz, 1982] d'une part, et par le nombre important des applications pratiques pouvant être formulées sous forme de problèmes d'optimisation combinatoires [Ribeiro et Maculan, 1994] d'autre part. L'optimisation combinatoire consiste à parcourir l'espace de recherche (l'ensemble de combinaisons pouvant être prises par les variables du problème) afin d'en extraire une solution optimale parmi un ensemble fini de solutions, d'une taille souvent très grande telle que son énumération exhaustive est une tâche fastidieuse. En fait, un problème d'optimisation combinatoire se ramène à résoudre une de ses instances par un procédé algorithmique permettant la maximisation (en cas de problème de maximisation) ou la minimisation (en cas de problème de minimisation) d'une (ou de plusieurs) fonction(s) objectif(s) en respectant certaines contraintes rendant infaisable une partie de l'espace de

recherche. Malgré que la définition de problèmes d'optimisation soit une tâche souvent facile, la résolution exacte de ce type de problème est une tâche souvent difficile et prohibitive en termes de temps de calcul et de l'espace mémoire requis. Ces derniers, augmentent avec la taille du problème traité. Plus elle est grande, plus l'algorithme de résolution est coûteux.

La théorie de la complexité [Garey et Johnson, 1979] permet de mesurer la complexité des problèmes en considérant le temps de calcul nécessaire et l'espace mémoire requis pour en résoudre. En effet, elle classe les problèmes en deux classes essentielles: la classe P (Polynomial time) et la classe NP (Non-deterministic Polynomial time). Par conséquent, elle permet d'aider au choix général de l'algorithme de résolution d'un problème donné en fonction de la difficulté intrinsèque du problème, qui selon son degré permet de partager les problèmes en deux types: faciles et difficiles à résoudre. Un problème est considéré facile s'il possède une solution polynomiale avec la taille de l'instance traitée, sinon il est difficile dans le cas où il possède une solution exponentielle ou pire qu'exponentielle par rapport à la taille de l'instance traitée. En plus du classement des problèmes imposé par leur complexité, les problèmes d'optimisation peuvent aussi être classés en deux autres classes en fonction du nombre de critères à satisfaire (fonction « objectif »). On parle de problèmes mono-objectifs et de problèmes multi-objectifs. Dans le premier cas (i.e. les problèmes mono objectif), on se trouve devant un nombre de critères à satisfaire qui ne dépasse pas le 1. Tandis que, dans le deuxième cas (i.e. les problèmes multi objectifs), on se trouve devant un nombre supérieur ou égal à deux critères (souvent contradictoires) à satisfaire simultanément. Dans ce dernier cas, la solution construit généralement un compromis entre les différents objectifs imposés. Outre la classification selon la complexité et selon le nombre de critère (fonction « objectif ») à satisfaire, les problèmes d'optimisation peuvent aussi être classés en problèmes industriels ou en problèmes académiques. En fait, les problèmes industriels sont des projections des problèmes académiques dans le plan de la pratique (i.e. la vie réelle).

1.2. Notions de base en optimisation

Deux types de problèmes d'optimisation sont distingués: des problèmes de minimisation et des problèmes de maximisation. Un problème d'optimisation (de minimisation ou de maximisation) est défini par un ensemble de données et un ensemble de contraintes. Un ensemble de solutions S est associé au problème d'optimisation. Parmi les solutions S , un sous-ensemble $X \subseteq S$ représente des solutions réalisables respectant les contraintes C du problème, à chaque solution s est associée une valeur $f(s)$ qui représente sa qualité. La

résolution du problème d'optimisation consiste à trouver une solution $s^* \in X$ qui minimise ou maximise la valeur $f(s)$.

Quelque soit le type du problème d'optimisation, ce dernier est défini par le 6-uplet $\langle D, C, S, X, f, mode \rangle$. Où D représente les données du problème, C les contraintes que doit satisfaire une solution afin d'être admissible, S l'ensemble des solutions possibles du problème traité, X un sous-ensemble de S représentant les solutions réalisables (admissibles), f une fonction du coût (aussi appelée fonction « objectif » ou fonction fitness) qui associe à chaque solution s une valeur numérique $f(s)$ (nombre réel ou entier) représentant la qualité de s , $mode$ indique le type du problème, il permet de savoir est ce qu'on doit minimiser ou maximiser les valeurs des solutions de X . Dans ce qui suit nous présentons quelques définitions tirées de la littérature liées aux problèmes d'optimisation en général.

Définition 1.1. Inspirée de [Papadimitriou et Steiglitz, 1982]: *Une instance d'un problème de minimisation (maximisation) est un couple (X, f) . Où $X \subseteq S$ est un ensemble fini de solutions potentielles admissibles et f est une fonction du coût (fonction objectif) à minimiser (à maximiser), $f : X \rightarrow \mathbb{R}$. L'objectif est de trouver $s^* \in X$ tel que $f(s^*) \leq f(s)$ (Au cas de maximisation: $f(s^*) \geq f(s)$) pour n'importe quelle solution $s \in X$.*

Définition 1.2. Inspirée de [Sakarovitch, 1984]: *Un problème d'optimisation combinatoire est défini par l'ensemble S des solutions possibles d'un problème. $X \subseteq S$ est l'ensemble des solutions réalisables. $f : X \rightarrow \mathbb{R}$, une fonction que l'on nomme fonction « objectif ». La résolution du problème consiste à minimiser (maximiser) la valeur $f(s)$, où $s \in X$.*

Définition 1.3. Inspirée de [Papadimitriou et Steiglitz, 1982]: *Un problème mono-objectif est décrit par la formule 1.1*

$$\begin{aligned}
 & \text{Mode}_x f(s) \tag{1.1} \\
 \text{Tel que } & \begin{cases} C_i(s) \Delta 0, i=1 \dots m \\ H_j(s) = 0, j=1 \dots p \\ s \in S \subset \mathbb{R}^n \end{cases} \\
 \text{Où } \text{Mode} = & \begin{cases} \text{Min au cas de problème de minimisation} \\ \text{Max au cas de problème de maximisation} \end{cases} \\
 \Delta \text{ est remplacé par } & \begin{cases} \leq \text{ au cas de problème de minimisation} \\ \geq \text{ au cas de problème de maximisation} \end{cases}
 \end{aligned}$$

f est la fonction du coût (la fonction «objectif») à minimiser (ou à maximiser). C_i sont des contraintes d'inégalité et H_i sont des contraintes d'égalité. S est l'ensemble de solutions possibles (l'espace de recherche). s est une solution admissible, elle appartient à S et respecte les contraintes du problème.

Définition 1.4. Inspirée de [Fonseca et Fleming, 1993]: *Un problème multi-objectif* est décrit par la formule 1.2

$$\begin{aligned} & \text{Mode}_x \vec{f}(s) & (1.2) \\ \text{Tel que } & \begin{cases} \vec{C}_i(s) \Delta 0, i=1 \dots m \\ \vec{H}_i(s) = 0, j=1 \dots p \\ s \in S \subset \mathfrak{R}^n \end{cases} \\ \text{Où } & \text{Mode} = \begin{cases} \text{Min au cas de problème de minimisation} \\ \text{Max au cas de problème de maximisation} \end{cases} \\ \Delta \text{ est remplacé par } & \begin{cases} \leq \text{ au cas de problème de minimisation} \\ \geq \text{ au cas de problème de maximisation} \end{cases} \end{aligned}$$

Dans le cas de problèmes d'optimisation multi-objectifs, la fonction «objectif» est représentée par un vecteur \vec{f} regroupant N fonctions «objectif». Aussi les contraintes sont représentées par des vecteurs regroupant les contraintes de chaque fonction «objectif».

Définition 1.5. *Une solution d'un problème d'optimisation* est un ensemble de quantités souvent numériques pour lesquelles des valeurs sont à choisir. L'ensemble de ces valeurs est généralement regroupé dans un vecteur représentant une solution. Supposant un problème de taille n , le vecteur représentant la solution s est représenté par:

$$\vec{s} = [s_1, s_2, s_3, \dots, s_n] \quad (1.3)$$

Les différentes valeurs prises par les variables $(s_1, s_2, s_3, \dots, s_n)$ constituent l'ensemble des solutions envisageables. Selon le type des variables $(s_1, s_2, s_3, \dots, s_n)$, le type du problème d'optimisation peut être reconnu. Par conséquent, on distingue deux classes de problèmes d'optimisation: des problèmes continus et des problèmes discrets. Dans la première classe (i.e. la classe des problèmes continus), les variables composant une solution donnée sont de type réel. Tandis que, dans les problèmes de la deuxième classe (i.e. la classe des problèmes discrets), les variables composant une solution donnée peuvent être de type entier, naturel ou binaire.

Définition 1.6. Une *contrainte* d'un problème est une restriction imposée par la nature et les caractéristiques du problème sur les solutions proposées.

Définition 1.7. L'*espace de recherche* S d'un problème est composé de l'ensemble de valeurs pouvant être prises par les variables $(s_1, s_2, s_3, \dots, s_n)$ qui construisent la solution s .

Définition 1.8. Le *voisinage* $V(s)$ d'une solution s est un sous ensemble de S , dont les membres sont des solutions proches (voisines) de la solution s . En effet, on dit qu'une solution « s' » est une voisine de s , si elle peut être obtenue en modifiant légèrement la solution s .

Définition 1.9. L'*optimum local* est la meilleure solution appartenant à un voisinage $V(s)$ de la solution s . Une solution s' (appartenant à S) est un optimum local de la structure du voisinage $V(s)$ de la solution s si elle vérifie la condition suivante (1.4):

$$f(s') \leq f(s) \quad \forall s \in V(s) \quad (1.4)$$

Pour un problème de maximisation l'inégalité est inversée, c-à-d. la condition (1.4) sera remplacée par la condition (1.5):

$$f(s') \geq f(s) \quad \forall s \in V(s) \quad (1.5)$$

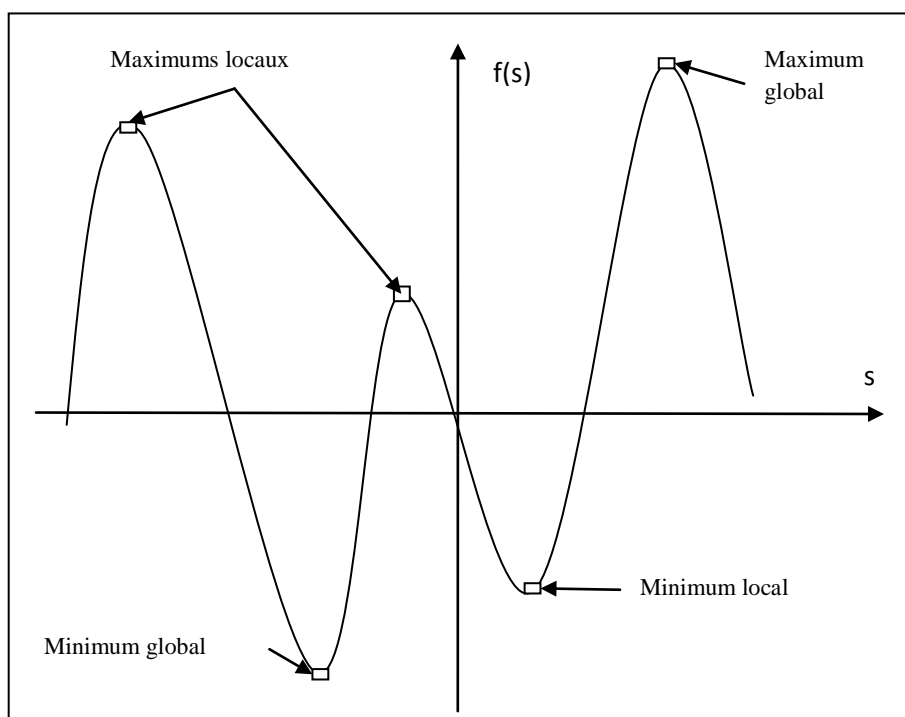


Figure1.1. Courbe représentant les optimums locaux et les optimums globaux

Il est à noter que l'optimum local est aussi nommé maximum local au cas de problème de maximisation et minimum local au cas de problème de minimisation.

Définition 1.10. Une solution optimale (L'optimum global): Une solution s^* est dite optimale (ou optimum global) si les deux contraintes suivantes sont vérifiées:

1. Elle est réalisable: tirée de l'ensemble de solutions possibles (l'espace de recherche S) et respecte toutes les contraintes du problème posé.
2.
$$\begin{cases} f(s^*) = \max_{s \in X} \{f(s)\} & \text{En cas de problème de maximisation} \\ f(s^*) = \min_{s \in X} \{f(s)\} & \text{En cas de problème de minimisation} \end{cases}$$
 Où X est l'ensemble de solutions réalisables.

Autrement dit, l'optimum global est le meilleur optimum local. Ainsi, une solution est dite optimum global si:

$$\begin{cases} f(s^*) \geq f(s) \forall s \in E(S) & \text{au cas de problème de maximisation} \\ f(s^*) \leq f(s) \forall s \in E(S) & \text{au cas de problème de minimisation} \end{cases}$$

Où $E(S)$ est l'ensemble d'optimums locaux. Il est à noter que l'optimum global est aussi nommé maximum global au cas de problème de maximisation et minimum global au cas de problème de minimisation.

La figure 1.1 représente une courbe représentant les optimums locaux et les optimums globaux d'une fonction d'évaluation.

1.3. La complexité et la théorie de la complexité

Afin de mesurer la difficulté d'un problème donné et la comparer avec celles d'autres problèmes pour pouvoir dire qu'un tel problème est plus facile à résoudre que l'autre, nous pouvons calculer la complexité algorithmique de chacun d'entre eux. La complexité d'un problème donné est discutée sur deux cotés: coté temporel (complexité temporelle) et coté spatial (complexité spatiale). La complexité temporelle consiste à évaluer le temps de calcul nécessaire pour résoudre un problème donné. Tandis que la complexité spatiale permet d'estimer les besoins en mémoire (l'espace mémoire requis) pour la résolution d'un problème donné. La complexité d'un problème donné est estimée en fonction du nombre d'instructions permettant d'aboutir à la solution du problème posé. Elle est influencée par la taille du problème en question. En effet, elle exprime un rapport entre la taille du problème, le temps de calcul nécessaire et l'espace mémoire requis.

La théorie de la complexité s'intéresse à l'évaluation de la difficulté des problèmes via l'étude de la complexité de solutions algorithmiques proposées. Elle associe une fonction de complexité à chaque algorithme résolvant un problème donné et mesure les ressources

nécessaires pour la résolution d'un problème posé. En effet, elle classe l'ensemble des problèmes selon deux critères qui sont le temps de calcul nécessaire et l'espace mémoire requis.

Bien que la théorie de la complexité se concentre sur des problèmes de décision, elle peut être étendue aux problèmes d'optimisations [Garey et Johnson, 1979]. Elle classe les problèmes selon leur complexité en deux classes principales: la classe P (Polynomial time) et la classe NP (Non deterministic Polynomial time). En outre, elle partage les problèmes de la classe NP en deux sous classes: NP-Complet et NP-Difficile.

1.3.1. Les problèmes de la classe P

Les problèmes appartenant à la classe P sont des problèmes pouvant être résolus par une machine de Turing¹ déterministe en temps de calcul polynomial par rapport à la taille de l'instance du problème à traiter. Ils sont souvent faciles à résoudre par des algorithmes efficaces dont le nombre d'instructions nécessaire pour la résolution d'un problème donné est borné par une fonction polynomiale par rapport à la taille du problème [Sakarovitch, 1984]. Plus formellement, les problèmes de cette classe sont définis comme suit [Alliot et Schiex, 1994]:

$$P = \{PL / PL \subseteq X^*, \exists MTD, PL = PL(MTD), \exists Ply, \forall w \in PL, |w| = n, T_{MTD}(w) < Ply(n)\} \quad (1.6)$$

Où Ply est une fonction polynomiale, MTD une machine de Turing déterministe, PL un problème et T_{MTD} le temps de reconnaissance du problème.

Le calcul du plus grand diviseur commun est un exemple d'un problème appartenant à la classe P.

1.3.2. Les problèmes de la classe NP

La classe des problèmes NP regroupe l'ensemble de problèmes qui peuvent être résolus avec une machine de Turing non-déterministe² et admettent un algorithme polynomial pour tester la validité d'une solution du problème traité (i.e. il est possible de vérifier que la solution proposée est correcte en un temps polynomial par rapport à la taille du problème).

¹ Une machine abstraite représentant un modèle abstrait du fonctionnement de l'ordinateur et de sa mémoire. Elle a été proposée par le mathématicien Alan Mathisan Turing en 1936 en vue de donner une définition précise au concept d'algorithme.

² Le terme non- déterministe désigne un pouvoir qu'on incorpore à un algorithme pour qu'il puisse aboutir à la bonne solution.

Le problème du sac à dos, le problème du voyageur de commerce, les problèmes d'ordonnancement, le problème de coloration de graphes sont des exemples de problèmes appartenant à la classe NP.

1.3.2.1. Les problèmes NP-Complets

La classe NP-Complet regroupe les problèmes de décision pour lesquels il n'existe pas d'algorithme permettant leur résolution en un temps polynomial. Selon Garey et Johnson, un problème X est un problème NP-Complet s'il appartient à la classe NP, et si quelque soit le problème X' qui appartient aussi à la classe NP, on peut le réduire au problème X en un temps polynomial [Garey et Johnson, 1979]. Selon Ferro et ces collègues, un problème X est un problème NP-Complet s'il appartient à la classe NP, et si on peut le réduire à un problème connu dans NP-Complet [Ferro et al, 2005]. Par conséquent, la NP-Complétude est un problème décisionnel. Le premier problème qui a été démontré comme étant NP-Complet a été celui de la satisfiabilité d'une formule logique binaire (SAT). Cela a été établi en 1971 par Stephen Cook dans son théorème publié dans [Cook, 1971].

1.3.2.2. Les problèmes NP-Difficiles

La classe de problèmes NP-Difficiles englobe les problèmes de décision et les problèmes d'optimisation. Les problèmes NP-Difficiles sont aussi difficiles que les problèmes NP-Complets. Si un problème de décision associé à un problème d'optimisation P est NP-Complet alors P est un NP-Difficile [Charon et al, 1996]. Par conséquent, afin de prouver qu'un problème d'optimisation est NP-Difficile, il suffit de montrer que le problème de décision associé à P est NP-Complet [Layeb, 2010]. Il est à noter que jusqu'à maintenant, aucun algorithme polynomial n'est connu pour résoudre ce type de problèmes (i.e. NP-Difficiles).

1.3.3. Les problèmes NP-Complets versus les problèmes NP-Difficiles

Les problèmes NP-Complets et les problèmes NP-Difficiles sont deux types de problèmes difficiles. Aucun algorithme polynomial permettant leur résolution n'a été trouvé. En fait, les algorithmes proposés dans la littérature sont de complexité exponentielle ou pire qu'exponentielle. La différence entre les problèmes de ces deux types de problèmes réside dans le type de la solution attendue. Les problèmes NP-Complets sont des problèmes décisionnels dont la réponse attendue pourra être « *oui* ou *non* ». Comme l'exemple du problème du cycle hamiltonien qui consiste à répondre à la question: Est-ce qu'il existe un cycle permettant de parcourir tous les sommets d'un graphe non orienté en passant par chacun

une seule fois ? Ici on s'attend à une des deux réponses: *oui il existe* **ou** *non il n'existe pas*. Par contre, les problèmes NP-Difficiles concernent beaucoup plus les problèmes d'optimisation ou on cherche la solution *optimale*. Comme le cas du problème du voyageur de commerce dont on cherche le plus court chemin permettant de parcourir un ensemble de sommets (villes) en passant par chacun une seule fois.

1.3.4. La relation entre les problèmes P et NP

La question « $P=NP$? » est une des questions les plus importantes qui n'ont pas encore été résolues en informatique théorique. En fait, la réponse à cette question a construit un champ de recherche ouvert. La réponse à la question « $P=NP$? » par « oui », revient à montrer que tous les problèmes de la classe NP sont dans la classe P. Autrement dit, la réponse à cette question revient à répondre à la question: Peut-on trouver en temps polynomial ce qu'on peut prouver en temps polynomial ? Cependant, aucune démonstration n'a été faite pour prouver que $NP \subseteq P$, ni que $NP \not\subseteq P$. La relation évidente c'est que $P \subseteq NP$. En effet, un problème qui peut être résolu en un temps polynomial par un algorithme déterministe, pourra aussi être résolu par un algorithme non déterministe. Ce qui est admis et connu jusqu'à maintenant, c'est que $P \neq NP$ [Garey et Johnson, 1979]. Néanmoins, aucune preuve n'a encore été prouvée jusqu'à ce jour. Si un jour on arrivera à trouver un algorithme polynomial permettant la résolution d'un problème NP-Complet, on pourra résoudre tous les problèmes NP-Complets en temps polynomial (voir section 3.2.1). Cook a prouvé dans [Cook, 1971] que tous les problèmes de la classe NP sont réductibles au problème de la satisfiabilité d'une formule logique, cela veut dire que si quelqu'un trouvera un algorithme polynomial pour le problème de SAT, alors la question « $P = NP$? » sera résolu!

Selon la figure 1.2, les problèmes NP faciles à résoudre peuvent construire des problèmes de la classe P. Les autres problèmes, i.e. les plus difficiles à résoudre, peuvent être partagés en deux autres groupes: le groupe des problèmes NP-Complets et le groupe des problèmes dont le statut est indéterminé car ces problèmes n'ont pas été prouvés comme appartenant à la classe P ni à la classe NP-Complet [Lacomme et al, 2003], comme le cas du problème d'isomorphisme³ de deux graphes [Lacomme et al, 2003]. En effet, il semblait que ce problème appartient à la classe P [Sakarovitch, 1984]. Néanmoins, jusqu'à ce jour aucune preuve n'a été montrée [Benatchba, 2005].

³ Deux graphes sont isomorphes si on peut transformer l'un en l'autre en renumérotant les sommets [Lacomme et al, 2003].

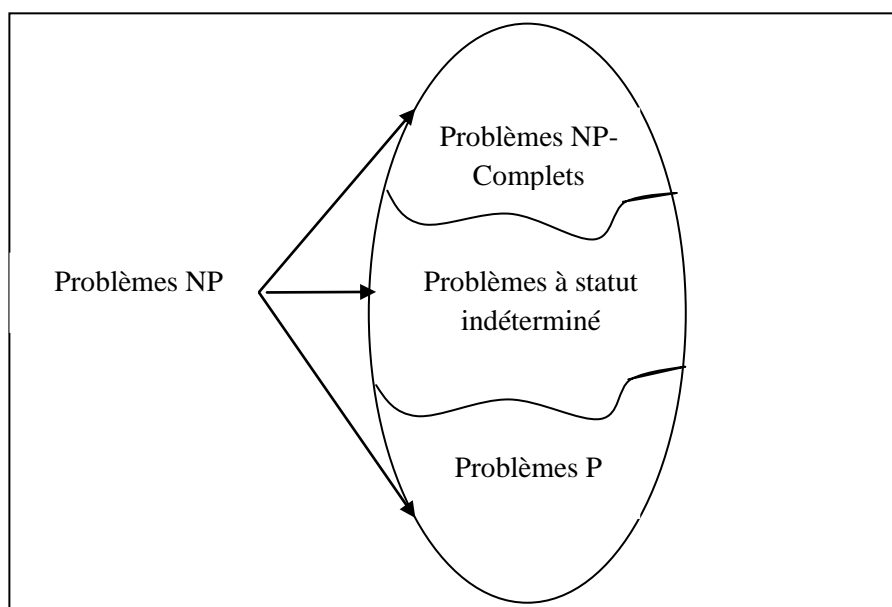


Figure 1.2. La relation entre les problèmes P, NP et NP-Complets.
[Lacomme et al, 2003]

1.4. Exemples de problèmes d'optimisation

Les problèmes d'optimisation peuvent être des problèmes académiques ou industriels. En fait, les problèmes industriels sont des projections des problèmes académiques sur le plan de la pratique (i.e. la vie réelle). Parmi les problèmes d'optimisation académiques, nous citons le problème de satisfiabilité booléenne MAX-SAT, le problème du sac à dos et ses différentes variantes, le problème du voyageur de commerce et ses différentes variantes, le problème de coloration de graphe, les problèmes d'ordonnancement...etc. Parmi les problèmes industriels nous citons les problèmes de design dans les domaines de l'ingénierie (mécanique, électronique, chimie...): ailes des avions, moteurs d'automobiles..., les problèmes d'investissements financiers, les problèmes d'ordonnancement en production, l'allocation des usines, la planification de trajectoires de robots mobiles, les problèmes de production agricoles, la gestion de containers, le design de réseaux du transport, les tracés autoroutiers, la distribution de l'eau, le design d'antennes, l'affectation de fréquences...etc.

L'objectif de cette section n'est plus de relater les principes des différents problèmes d'optimisation industriels et académiques, mais plutôt d'élucider le principe de quelques problèmes d'optimisation académiques en tant qu'exemples. Une importance particulière est affectée aux problèmes de la famille du problème du sac à dos et aussi au problème du voyageur de commerce car ils sont utilisés dans le test des algorithmes proposés dans le cadre de cette thèse.

1.4.1. Les problèmes de satisfiabilité booléenne SAT et MAX-SAT

Les problèmes de satisfiabilité sont des problèmes clés de plusieurs domaines de l'intelligence artificielle. Le problème SAT (Satisfiabilité booléenne) est un des problèmes de satisfiabilité les plus connus et les plus importants. Son importance est liée au fait qu'il se base sur la logique propositionnelle. Cette dernière facilite la représentation d'une grande gamme de problèmes dans différents domaines y compris: la vérification des logiciels dans le domaine du Software Engineering, la vérification des circuits dans le domaine du Hardware Engineering, les problèmes réseaux comme le routage, l'ordonnancement, la spécification de protocoles, les problèmes de l'Intelligence Artificielle comme la planification, le raisonnement l'apprentissage...etc [Layeb, 2010]. SAT est un problème de décision, il appartient à la classe de problèmes NP-Complets. C'est le premier problème qui a été prouvé comme étant un problème NP-Complet. Cela est établi en 1971 par Stephen Cook [Cook, 1971]. Stephen Cook a prouvé que tous les problèmes de la classe NP sont réductibles au problème SAT. Le problème SAT consiste à trouver un modèle d'une formule propositionnelle [Lardeux, 2005]. Autrement dit, il consiste à trouver une interprétation des variables satisfaisant l'ensemble de clauses⁴ d'une formule booléenne [Layeb, 2010].

Le problème MAX-SAT (MAXimum SATisfiabilité) est une des variantes fameuses du problème SAT. MAX-SAT est un problème d'optimisation combinatoire [Layeb, 2010]. Il construit une généralisation du problème SAT, lorsque la formule n'admet pas de solution, en permettant de déterminer une solution approximative [Layeb, 2010]. Il consiste à déterminer une affectation qui maximise le nombre des clauses vraies et minimise le nombre des clauses fausses, lorsqu'une formule propositionnelle sous FNC⁵ (Forme Normale Canonique) est non satisfiable.

1.4.2. Les problèmes d'ordonnancement

Un problème d'ordonnancement consiste à planifier la réalisation des tâches (jobs) en utilisant des ressources et en tenant compte de deux contraintes indispensables et souvent inséparables: le temps de réalisation et la disponibilité des ressources utilisées.

Une tâche est un ensemble d'opérations dont l'exécution peut être réalisée par morceaux ou sans interruption. L'ensemble des opérations constituant une tâche donnée est localisé dans le temps par des dates de début et des dates de fin (i.e. la réalisation de chaque tâche est

⁴ Une clause est une disjonction (proposition de la forme « $A \vee B$ ») de littéraux (variables de proposition).

⁵ Une proposition est en FNC si elle est composée de conjonctions (propositions de forme $A \wedge B$) de disjonctions.

limitée par une durée). En outre, la réalisation de l'ensemble des opérations des différentes tâches s'effectue par l'intermédiaire des ressources avec certaine disponibilité.

Les ressources sont des moyens permettant la réalisation des opérations. Une ressource peut être un homme, une machine, une matière première, l'argent, un équipement nécessaire... etc. Quelque soit la ressource utilisée, sa disponibilité n'est plus éternelle. Selon sa disponibilité, la ressource peut être *renouvelable* ou *consommable (non renouvelable)*. Une ressource est renouvelable si elle est à nouveau disponible en même qualité après avoir été allouée à une ou plusieurs opérations, comme le cas où la ressource est un homme ou une machine. Dans le cas contraire, la ressource est dite consommable comme le cas où la ressource est une matière première ou un budget [Bourazza, 2006].

Un problème d'ordonnancement peut être exprimé comme suit: étant donné un ensemble de tâches à réaliser et un ensemble de machines disponibles pour la réalisation des différentes tâches, le problème consiste à trouver l'ordre de passage des tâches par les différentes machines et la durée d'occupation (la date de début et la date de fin) de chaque machine par chaque tâche. La plupart des problèmes d'ordonnancement sont des problèmes d'optimisation qui appartiennent à la classe des problèmes NP-Difficiles [Pongchairerks, 2009]. Dans la résolution d'un problème d'ordonnancement, on cherche souvent à optimiser plusieurs critères [Bourazza, 2006] tels que: le temps total d'exécution, le temps moyen d'achèvement d'un ensemble de tâches, les différents retards (maximum, moyen, somme ...etc.) ou avances par rapport aux dates limites fixées.

Il existe plusieurs types de problèmes d'ordonnancement, les plus connus sont:

1.4.2.1. Le problème Job-Shop

Le problème Job-Shop est un des problèmes fameux d'optimisation combinatoires NP-Difficiles [Applegate et Cook, 1991]. Il peut être expliqué comme suit: Étant donné un ensemble de n jobs et un ensemble de m machines. Chaque job j est composé d'un ensemble ordonné de m opérations $O_1, O_2, O_3, \dots, O_n$. L'ordre des opérations ne peut pas être modifié, elles sont organisées par un ordre donné propre à chaque job. En outre, l'opération O_i doit être réalisée par une machine donnée durant un temps T_i sans interruption (i.e. lorsque l'opération commence, elle ne peut pas être interrompue ou stoppée temporairement jusqu'à ce qu'elle s'achève complètement). D'autre part, chaque machine peut exécuter un seul job à la fois (i.e. dans une période donnée) et chaque job peut être exécuté par une seule machine pendant une période de temps. Le problème consiste à trouver une allocation faisable (ou l'allocation

optimale) de toutes les opérations à des intervalles de temps et des machines disponibles en respectant les contraintes du problème. L'objectif final est de minimiser le temps d'exécution du dernier job. Il existe des variantes du problème Job-Shop comme celle nommée: Multi Purpose Machine Job-Shop [Pongchairerks, 2009].

1.4.2.2. Le problème Flow-Shop

Le problème Flow-Shop est un des problèmes d'optimisation combinatoires difficiles à résoudre [kouki et al, 2011], il ressemble au problème Job-Shop, sauf que dans le problème Flow-Shop on considère un ensemble de n jobs qui doivent être réalisés dans un ordre identique par m machines (i.e. tous les jobs passent par les m machine dans le même ordre). Idem que le problème Job-Shop, Il existe des variantes du problème Flow-Shop comme la variante nommée: no-Wait Flow-Shop [Liu et al, 2006] et la variante nommée Assembly Flow-Shop [Allahverdi et Al-Anzi, 2006].

1.4.2.3. Le problème Open-Shop

Le problème Open-Shop est un problème d'optimisation combinatoire difficile [Monette et al, 2007]. Il représente un cas spécial du problème Job-Shop. Dans le problème Open-Shop, les opérations d'un job peuvent être exécutées dans n'importe quel ordre. Autrement dit, les machines réalisant les jobs sont parcourues dans un ordre quelconque.

1.4.3. La famille du problème du sac à dos

1.4.3.1. Le problème du sac à dos

Le problème du sac à dos est un des problèmes d'optimisation académiques les plus importants. Il est noté par KP (De l'anglais, Knapsack Problem), son importance est liée en particulier au fait qu'il peut être utilisé en tant que sous problème dans de nombreux problèmes d'optimisation. Le principe du KP peut être expliqué par l'exemple suivant: considérant une famille qui veut se déplacer vers une ville côtière pour passer des vacances estivales. Le papa a demandé aux membres de sa famille de préparer une liste des objets importants qui leur seraient utiles et il a préparé sa voiture pour voyager. La capacité de la voiture est limitée, elle a une capacité maximale qu'elle ne peut pas dépasser. Chacun des membres de la famille a préparé sa propre liste et ils ont construit une grande liste qu'ils ont présentée au papa. Outre leurs poids (i.e. les poids des membres de la famille) non discutables, chaque objet de la liste préparée a un poids et une certaine importance (profit). De sa part, le papa a essayé de calculer le poids total des objets. Il a trouvé que la somme des poids des objets et des poids des membres de la famille dépasse la capacité maximale de la

voiture. Le papa s'est trouvé devant le dilemme de satisfaire les besoins des membres de sa famille et de ne pas dépasser la capacité maximale de sa voiture. Par conséquent, il a pensé à une solution qui consiste à sélectionner un sous ensemble de l'ensemble d'objets proposés de façon à réduire le poids total des objets et maximiser le profit global sans dépasser la capacité maximale de la voiture.

L'importance du problème du sac à dos a sollicité l'attention de plusieurs communautés de chercheurs, depuis longtemps et jusqu'aujourd'hui. En fait, il a fait l'objet de plusieurs travaux proposés dans la littérature comme: l'état de l'art proposé par Kellerer et ses collègues [Kellerer et al, 2004] et les différents algorithmes de résolution du problème comme les algorithmes proposés par: Martello et son collègue Toth [Martello et Toth, 1977] [Martello et Toth, 1990], Fayard et Plateau [Fayard et Plateau, 1982], Pisinger [Pisinger, 1995], Horowitz et Sahni [Horowitz et Sahni, 1974], Ahrens et Finke [Ahrens et Finke, 1975], Toth [Toth, 1980]. L'intérêt du problème du sac à dos n'est pas uniquement théorique. En fait, de nombreuses applications pratiques du problème du sac à dos peuvent être trouvées. Nous citons à titre d'exemple les problèmes des systèmes de cryptages [Merkle et Hellman, 1978], les problèmes d'investissements et sélection des projets [Lorie et Savage, 1955] [Nemhauser et Ulmann, 1969], la gestion de portefeuilles [Syam, 1998], le transport [Zhong et Young, 2010] et d'autres applications dans d'autres domaines [Bretthauer et Shetty, 2002].

Plusieurs variantes du problème du sac à dos ont été proposées dans la littérature. Elles peuvent être classées en sous classes selon plusieurs critères y compris le type de variables (variables binaires, entières et réelles), le nombre de contraintes (unidimensionnel, bidimensionnel ou multidimensionnel), le nombre d'objectifs (mono-objectif, multi-objectifs) etc. Dans la suite de cette section, nous citons quelques variantes.

Dans sa version la plus simple, dénommée « sac à dos unidimensionnel binaire », le KP peut être défini comme suit: Supposant que nous avons un sac à dos qui a une capacité maximale C . d'autre part, nous avons un ensemble de N objets. Chaque objet i a un poids w_i et un profit p_i . Le problème consiste à sélectionner un sous ensemble d'objets de manière à maximiser le profit total du sac à dos sans dépasser sa capacité maximale. Le problème peut être formulé comme suit:

$$\text{Maximiser} \quad \sum_{i=1}^N p_i x_i \quad (1.8)$$

$$\text{Avec } \sum_{i=1}^N w_i x_i \leq C \quad (1.9)$$

$$\text{Et } x_i = \begin{cases} 1 & \text{si l'objet } i \text{ est sélectionné} \\ 0 & \text{Sinon} \end{cases} \quad i=1, \dots, N$$

Il est à noter que le problème du sac à dos unidimensionnel (appelé tout court: problème du sac à dos) est un problème d'optimisation combinatoire [Jorge, 2010 ; Suryadi et Kartika, 2011] et qu'il appartient à la classe des problèmes NP-Difficiles [Pisinger, 2005].

1.4.3.2. Le problème du sac à dos multidimensionnel

La variante multidimensionnelle du problème du sac à dos est une issue de la classe des problèmes du sac à dos. Elle est notée par MKP (de l'anglais, Multidimensional Knapsack Problem). Le MKP est un problème d'optimisation combinatoire qui appartient à la classe des problèmes NP-Difficiles [Kong et Tian, 2006]. Dans ce problème, on suppose qu'on a un sac à dos avec M dimensions. Chaque dimension a une capacité maximale C_j ($j=1, \dots, M$). D'autre part, on a un ensemble d'objets. Chaque objet a une valeur (profit) p_i ($i=1, \dots, N$) et un poids w_{ji} dans la dimension j du sac à dos. Le problème qu'on cherche à résoudre consiste à trouver un sous ensemble d'objets de manière à maximiser la valeur totale du sac à dos sans dépasser les capacités de toutes ses dimensions. Le MKP peut être formulé comme suit:

$$\text{Maximiser } \sum_{i=1}^N p_i x_i \quad (1.10)$$

$$\text{Avec } \sum_{i=1}^N w_{ji} x_i \leq C_j, \quad j=1, \dots, M \quad (1.11)$$

$$x_i = \begin{cases} 1 & \text{Si l'objet } i \text{ est sélectionné} \\ 0 & \text{Sinon} \end{cases} \quad i=1, \dots, N$$

Le problème du Sac à Dos Multidimensionnel peut être utilisé pour formuler une panoplie de problèmes industriels tels que les problèmes de budgets, d'allocation des ressources physiques et logiques dans un système informatique distribué, de la découpe du stocke, de la gestion des projets et du stockage des conteneurs [Chu et Beasley, 1998]. Vu son importance et son appartenance à la classe des problèmes NP-Difficiles, le problème du sac à dos multidimensionnel a sollicité l'attention de plusieurs communautés de chercheurs. En effet, sa résolution a fait l'objectif principal de plusieurs travaux proposés par plusieurs chercheurs. Nous citons à titre d'exemple: Chu et Beasley [Chu et Beasley, 1998] qui ont proposé un algorithme génétique pour résoudre le MKP, Alonso et ses collègues [Alonso et

al, 2005] qui ont suggéré une stratégie évolutionnaire basée sur le calcul génétique de multiplicateurs de substitution, Li et ses collègues [Li et al, 2006] qui ont proposé un algorithme génétique basé sur le plan orthogonal pour le MKP, Zhou et ses collègues [Zhou et al, 2008a] qui ont proposé une combinaison entre les réseaux de neurones chaotiques et une stratégie heuristiques pour le MKP, Angelelli et ses collègues [Angelelli et al, 2010] qui ont proposé une heuristique générale pour le MKP, Kong et Tian [Kong et Tian, 2006] qui ont proposé une optimisation par essaim de particules pour résoudre le MKP.

1.4.3.3. Le problème du sac à dos à choix multiple

Le problème du sac à dos à choix multiple (MCKP: Multiple Choice Knapsack Problem) est une variante du problème du sac à dos dans laquelle les objets sont regroupés en K classes. Lors de la sélection des objets, on doit prendre en considération qu'on n'a pas le droit de sélectionner plus qu'un seul objet de chaque classe (i.e. un et un seul objet doit être sélectionné de chaque classe). Le problème peut être formulé comme suit:

$$\text{Maximiser } \sum_{j=1}^K \sum_{i \in N_j} x_{ji} p_{ji} \quad (1.12)$$

$$\text{Avec } \sum_{j=1}^K \sum_{i \in N_j} x_{ji} w_{ji} \leq C \quad (1.13)$$

$$\text{Et } \sum_{i \in N_j} x_{ji} \leq 1, \forall j \in \{1, \dots, K\} \quad (1.14)$$

$$x_{ji} = \begin{cases} 1 & \text{Si l'objet } i \text{ de la classe } j \text{ est sélectionné} \\ 0 & \text{Sinon} \end{cases}$$

Où K est le nombre total de classes. N_j avec $j \in \{1, \dots, K\}$ c'est la classe j .

Le problème du sac à dos à choix multiple peut être utilisé pour la formulation de nombreux problèmes dans différents domaines y compris les problèmes de gestion de budgets [Nauss, 1978], l'allocation de ressources [Sinha et Zoltners, 1979] et la fiabilité des systèmes [Tillman et al, 1980]. Vu son importance, le MCKP a attiré l'attention de plusieurs chercheurs qui ont mené leurs études dans l'objectif de résoudre ce problème, nous citons à titre d'exemple: Sinha et Zoltners [Sinha et Zoltners, 1979], Dyer et ses collègues Kayal et Walker [Dyer et al, 1984], Dyer et ses collègues Riha et Walker [Dyer et al, 1995], Pisinger [Pisinger, 1995b].

1.4.3.4. Le problème du sac à dos quadratique

Le problème du sac à dos quadratique (QKP: de l'anglais Quadratic Knapsack Problem) est une extension du problème du sac à dos dans laquelle les valeurs des profits ne sont pas affectées aux objets individuels uniquement mais aussi aux paires d'objets. Autrement dit, dans la version quadratique du problème du sac à dos, un profit supplémentaire est obtenu lorsque deux objets sont sélectionnés simultanément. Le problème peut être formulé comme suit:

$$\text{Maximiser} \quad \sum_{i=1}^N x_i p_i + \sum_{i=1}^N \sum_{j=1}^N x_i x_j p_{ij} \quad (1.15)$$

$$\text{Avec} \quad \sum_{i=1}^N w_i x_i \leq C \quad (1.16)$$

$$x_i = \begin{cases} 1 & \text{Si l'objet } i \text{ est sélectionné} \\ 0 & \text{Sinon} \end{cases} \quad i=1, \dots, N$$

Où p_{ij} est le profit associé au couple d'objets i et j .

Le problème du sac à dos quadratique est un problème difficile à résoudre, il appartient à la classe des problèmes NP-Difficiles [Julstrom, 2005]. Le QKP peut être utilisé pour la formulation de nombreux problèmes y compris: les problèmes financiers [Laughunn, 1970] et les problèmes de graphes [Johnson et al, 1993] [Park et al, 1996]. Gallo et ses collègues Hammen et Simeone sont les premiers qui ont décrit le problème du sac à dos quadratique [Gallo et al, 1980]. Ils ont essayé de le résoudre par un algorithme exact nommé « Branch and Bound ». Depuis sa description, plusieurs autres chercheurs ont tenté de résoudre le problème QKP comme Caprara et ses collègues Pisinger et Toth [Caprara et al, 1980], Hammer et Rader [Hammer et Rader, 1997], Helmberg et ses collègues Rendl, et Weismantel [Helmberg et al, 2000], Billionnet et Eric [Billionnet et Soutif, 2004], Billionnet et ses collègues Faye et Soutif [Billionnet et al, 1997], Kellerer et ses collègues Pferschy et Pisinger [Kellerer et al, 2004], Helmberg et ses collègues Rendl et Weismantel [Helmberg et al, 2000].

1.4.3.5. Le problème du sac à dos quadratique multidimensionnel

Le problème du sac à dos quadratique multidimensionnel est une extension de deux autres variantes du problème du sac à dos: le problème du sac à dos multidimensionnel (MKP) et le problème du sac à dos quadratique (QKP). Le problème du sac à dos quadratique multidimensionnel est noté par QMKP (de l'anglais: Quadratic Multidimensional Knapsack

Problem). Son principe combine les principes des variantes MKP et QKP. En effet, dans le QMKP, on considère un ensemble N d'objets et un sac à dos avec M dimensions. Chaque dimension a une capacité maximale C_j et chaque objet a une valeur (profit) p_i et un poids w_{ji} dans la dimension j du sac à dos. En plus du profit de chaque objet seul, un profit quadratique est affecté à chaque paire d'objets, i.e. un profit supplémentaire associé au paire d'objets i et j sera ajouté au profit total si les deux objets i et j sont dans le même sac (la même dimension). Lors de la sélection des objets, on doit veiller à ce qu'un objet ne peut être affecté qu'à un seul sac (dimension) au maximum. L'objectif du problème est de trouver la bonne distribution des objets sur les différentes dimensions du sac à dos de façon à maximiser le profit total des objets du sac à dos (avec toutes ses dimensions) sans dépasser les capacités de ses différentes dimensions. Le QMKP peut être formulé comme suit:

$$\text{Maximiser } \sum_{i=1}^N \sum_{j=1}^M x_{ij} p_i + \sum_{i=1}^{N-1} \sum_{k=i+1}^N \sum_{j=1}^M x_{ij} x_{kj} p_{ik} \quad (1.17)$$

$$\text{Avec } \sum_{i=1}^N x_{ij} w_{ij} \leq C_j \quad j=1, \dots, M \quad (1.18)$$

$$x_{ij} = \begin{cases} 1 & \text{Si l'objet } i \text{ est affecté au sac } j \\ 0 & \text{Sinon} \end{cases} \quad i=1, \dots, N ; j=1, \dots, M$$

Le problème du sac à dos quadratique multidimensionnel a été défini pour la première fois en 2006 par Hiley et Jultstrom [Hiley et Julstrom, 2006]. Le QMKP est une extension de deux problèmes d'optimisation combinatoires NP- Difficiles, il est aussi un des problèmes d'optimisation combinatoires appartenant à la classe des problèmes NP-Difficiles [Hiley et Julstrom, 2006]. Après la définition du QMKP, Hiley et Jultstrom ont proposé trois approches pour le résoudre. D'autres chercheurs comme Saraç et Sipahioglu [Saraç et Sipahioglu, 2007], Singh et Baghel [Singh et Baghel, 2007], Sundar et Singh [Sundar et Singh, 2010] ont aussi essayé de résoudre la variante Quadratique Multidimensionnelle du problème du sac à dos en s'inspirant des algorithmes proposés dans la littérature.

1.4.4. Le problème de tournées de véhicules

Le problème de tournées de véhicules (VRP: Vehicle Routing Problem) a été introduit pour la première fois en 1959 par Dantzig et Ramser [Dantzig et Ramser, 1959]. Le VRP est un des problèmes qui ont sollicité l'attention de plusieurs équipes de recherches vu son importance et son utilité dans la résolution de nombreux problèmes rencontrés dans la vie quotidienne. Le VRP est un problème d'optimisation combinatoire, il appartient à la classe

des problèmes NP-Difficiles [Savelsbergh, 1995] où il n'existe pas d'algorithme permettant la résolution d'un problème donné en un temps polynomial.

Le principe du problème de tournées de véhicules consiste à trouver les meilleurs trajets à parcourir par une flotte de M véhicules afin de visiter N clients (sites) pour des raisons de collecte ou de livraison. Tous les véhicules commencent et terminent leurs tournées à un site central appelé *dépôt*. Chaque client est affecté à une tournée et il doit être visité une seule fois par un seul véhicule. La capacité de transport de ce dernier pour une tournée est limitée et elle ne doit pas être dépassée. Tous les trajets proposés doivent respecter les différentes capacités des véhicules et satisfaire les quantités demandées par les clients ou à livrer à ces derniers [Kammarti, 2006]. La figure 1.3 représente un exemple de tournée avec quatre véhicules qui assurent la visite de 21 clients tout en partant et revenant au dépôt.

En général, l'objectif principal des problèmes de tournées de véhicules est de minimiser le coût (distance) total de parcours des trajets par un nombre minimum de véhicules. Outre son objectif principal, le VRP impose d'autres contraintes à respecter parmi lesquelles:

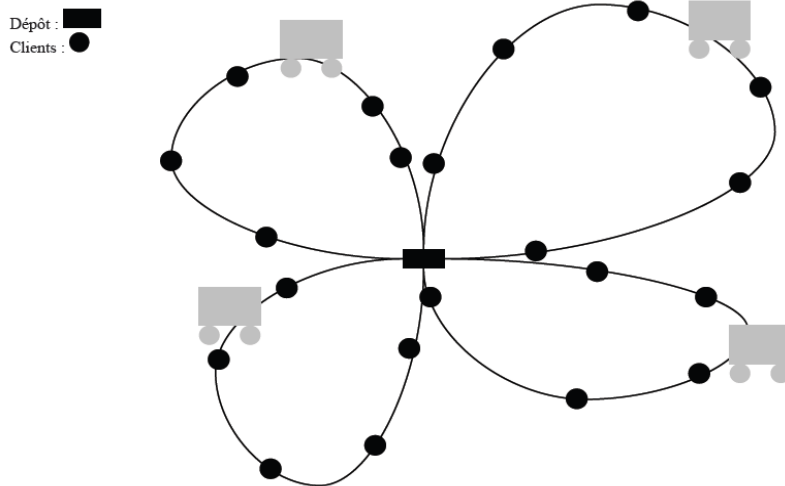


Figure 1.3. Exemple de tournées avec 4 véhicules.
[Kammarti, 2006]

- Chaque client doit être servi une seule fois par un et un seul véhicule.
- Le client visité doit impérativement être quitté.
- Le nombre de véhicules est limité.
- Les véhicules doivent tous commencer leurs tournées du dépôt.

- Les sous tours sont interdits, i.e. tous les véhicules reviennent au dépôt et non pas aux autres sites.

L'ajout, la modification ou l'élimination d'une ou des contraintes du VRP peut nous mener vers d'autres variantes qui sont essentiellement apparues suite aux activités des chercheurs qui travaillent de plus en plus sur les problèmes de transport et de distribution que rencontrent les sociétés [Kammarti, 2006]. Parmi les variantes du VRP, nous pouvons citer: le **CVRP** (Capacitated Vehicle Routing Problem) [Ralphs et al, 2001], le **DVRP** (Dynamic Vehicle Routing Problem) [Gendreau et Potvin, 1998], le **VRPTW** (Vehicle Routing Problem with Time Windows) [Kilby et al, 1999] [Cordeau et al, 2000], **DVRPTW** (Dynamic Vehicle Routing Problem with Time Windows) [Gendreau et al, 2000], le **VRPB** (Vehicle Routing Problem with backhauls) [Jacobs-Blecha et al, 1998], le **PVRP** (Periodic Vehicle Routing Problem) [Witucki et al, 1997], **SVRP** (Stochastic Vehicle Routing Problem) [Groth, 2002], **VRPPD** (Vehicle Routing Problem with Pickup and Delivery) [Mechti, 1995], **MDVRP** (Multi-Depot Vehicle Routing Problem) [Fischetti et al, 1999], **TRP** (Travelling Repairman Problem) [Krumke et al, 2001], **VRPHF** (Vehicle Routing Problem with Heterogeneous Fleet) [Taillard, 1999] [Prins, 2002]. **OVRP** (Open Vehicle Routing Problem) [Fu et al, 2003], **SDVRP** (Split Delivery Vehicle Routing Problem) [Archetti et al, 2002], **PDPTW** (Pickup and Delivery Problem with Time Windows) [Psaraftis, 1983].

1.4.5. Le problème du voyageur de commerce

Le problème du voyageur de commerce (en anglais Travelling Salesman Problem: TSP) est un problème classique d'optimisation combinatoire. Le TSP est un cas particulier du problème de Tournées de Véhicules sans contrainte de capacité et avec un seul véhicule [Rego et al 1994]. Dans le problème du TSP, on suppose un voyageur de commerce qui doit visiter n villes données, en passant par chaque ville exactement une et une seule fois. Il commence par une ville quelconque et doit terminer en retournant à la ville du départ. Les distances entre les villes sont connues. Le problème consiste à déterminer le chemin permettant de minimiser la distance parcourue. La notion de distance peut être remplacée par d'autres notions comme le temps qu'on met ou l'argent qu'on dépense: dans tous les cas, on parle du coût. De même, la notion de ville peut être remplacée par d'autres notions comme la location, l'entrepôt...etc. La figure 1.4 représente un exemple d'un problème de voyageur de commerce avec 9 villes. La figure représente 5 tournées possibles où chacune est représentée par une couleur.

Le problème du voyageur de commerce peut être formulé comme suit:

$$\text{Minimiser} \quad \sum_{i=1}^{n-1} (D(v_i, v_{i+1})) + D(v_n, v_1) \quad (1.19)$$

Où $D(v_i, v_{i+1})$ est la distance (ou le coût) entre les deux villes (points) v_i et v_{i+1} .

Il existe deux variantes principales du problème du voyageur de commerce: le problème du voyageur de commerce symétrique et le problème du voyageur de commerce asymétrique (le TSP symétrique et le TSP asymétrique). Dans la variante dite symétrique, la distance (le coût) entre deux villes (points) quelconques v_i et v_{i+1} est la même dans les deux sens (i.e. $D(v_i, v_{i+1}) = D(v_{i+1}, v_i)$). Néanmoins, dans la variante dite asymétrique, la distance (le coût) entre deux villes (points) quelconques v_i et v_{i+1} n'est plus la même dans les deux sens (i.e. $D(v_i, v_{i+1}) \neq D(v_{i+1}, v_i)$). Toutefois, un problème TSP asymétrique de taille n peut être transformé en un problème symétrique de taille $2*n$.

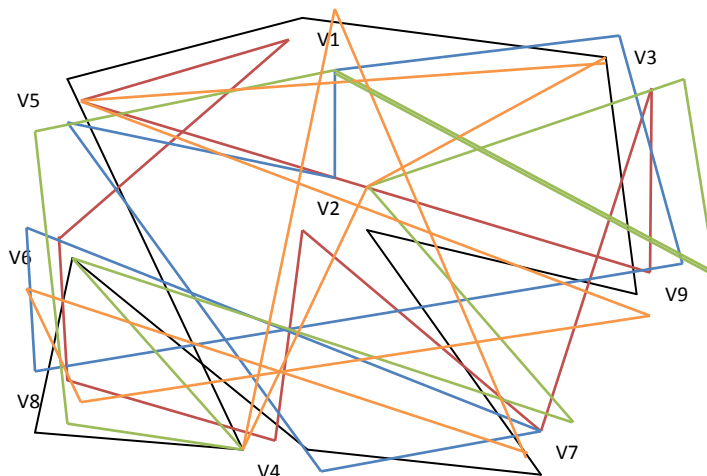


Figure 1.4. Exemple de solutions d'un problème de voyageur de commerce

Le TSP a des applications directes, notamment dans le transport et la logistique. Par exemple, trouver le chemin le plus court pour les bus de ramassage scolaire, ou dans l'industrie, pour trouver la plus courte distance que devra parcourir le bras mécanique d'une machine pour percer les trous d'un circuit imprimé [Bourazza, 2006]. On peut l'appliquer également dans la construction des cartes mères des ordinateurs afin de minimiser la longueur des fils de cuivre entre les puces. Le TSP peut aussi être utile en ordonnancement, si n commandes doivent être effectuées sur une seule machine et que le temps de mise en route dépend de l'ordre de passage, le TSP permettra de minimiser le temps de mise en route des commandes en déterminant l'ordre dans lequel celles-ci doivent être effectuées. Une autre application du TSP est possible en cristallographie afin de minimiser le temps de prise des mesures aux rayons X.

Les premières approches mathématiques proposées pour le TSP ont été traitées par les mathématiciens Sir William Rowan Hamilton et Thomas Penyngton Kirkman [Biggs et al, 1976]. Ensuite, le TSP a été développé et traité en profondeur par Karl Menger et Hassler Whitney et son collègue Merrill Flood [Schrijver, 1960]. En se basant sur ces efforts, Dantzig et ses collègues Fulkerson et Johnson ont résolu une instance du problème du voyageur de commerce avec 49 villes [Dantzig et al, 1954], Camerini et ses collègues Fratta et Maffioli ont résolu une instance avec 100 villes [Camerini et al, 1974], Padberg et Rinaldi ont résolu une instance avec 532 villes [Padberg et Rinaldi, 1987] et une autre avec 2392 villes [Padberg et Rinaldi, 1991], Applegate et ses collègues Bixby, Chvatal et Cook ont résolu une instance de taille 15112 villes [Applegate et al, 2001].

Le TSP appartient à la classe des problèmes NP-difficiles [Garey and Johnson, 1979], où l'existence d'un algorithme de complexité polynomiale reste inconnue. Un calcul rapide de la complexité montre qu'elle est en $O(n!)$. Avec n est le nombre de villes à visiter. En supposant que le temps pour évaluer un trajet est de $1\mu s$, le tableau 1.1 montre l'explosion combinatoire du TSP [Bourazza, 2006].

Nombre de villes	Nombre de possibilités	Temps de calcul
5	12	12 μ
10	181440	0,18 s
15	43 milliards	12 heures
20	60×10^{15}	1928 ans
25	310×10^{21}	9,8 milliards d'années (!)

Tableau 1.1. Nombre de trajets possibles et temps de calcul estimé

1.5. Conclusion

Dans ce chapitre, nous avons essayé d'aborder quelques notions de base en optimisation. Nous avons présenté des définitions de notions confrontées souvent dans le domaine de l'optimisation en allant de la définition d'un problème d'optimisation, passant par les types des problèmes d'optimisation (minimisation mono-objectif, minimisation multi-objectifs, maximisation mono-objectif, maximisation multi-objectifs), jusqu'à la définition d'une solution optimale d'un problème donné. Ensuite, nous avons dévoilé le rôle de la théorie de la complexité dans la classification des différents problèmes selon leurs complexités en

problèmes P et problèmes NP. Nous avons donné une vue générale sur les problèmes des deux classes et nous avons essayé d'éclaircir la relation entre les deux classes. Enfin, nous avons tenté d'élucider le principe de quelques problèmes d'optimisation académiques difficiles en donnant une importance particulière aux problèmes du sac à dos et au problème du voyageur de commerce car ils ont été utilisés dans le test des méthodes que nous avons proposées dans le cadre de notre thèse.

Dans le chapitre qui suit, nous présentons un état de l'art sur quelques méthodes de résolution de problèmes d'optimisation qui ont été proposées dans la littérature.

CHAPITRE 2

Les méthodes de résolution de problèmes d'optimisation

Sommaire

2.1	Introduction.....	41
2.2	Les méthodes exactes.....	45
2.3	Les métaheuristiques.....	47
2.4	Conclusion.....	76

2.1. Introduction

La résolution de différentes sortes de problèmes rencontrés dans notre vie quotidienne a poussé les chercheurs à proposer des méthodes de résolution et à réaliser de grands efforts pour améliorer leurs performances en termes de temps de calcul nécessaire et/ou de la qualité de la solution proposée. Au fil des années, de nombreuses méthodes de résolution de problèmes de différentes complexités ont été proposées. Ainsi, une grande variété et des différences remarquables au niveau du principe, de la stratégie et des performances ont été discernées. Cette variété et ces différences ont permis de regrouper les différentes méthodes de résolution de différents problèmes en deux classes principales: la classe de méthodes exactes et la classe de méthodes approchées. L'hybridation des méthodes de ces deux classes a donné naissance à une pseudo classe qui englobe des méthodes dites hybrides (voir figure 2.1).

Les méthodes exactes sont connues par le fait qu'elles garantissent l'optimalité de la solution mais elles sont très gourmandes en termes de temps de calcul et de l'espace mémoire nécessaire. C'est la raison pour laquelle, elles sont beaucoup plus utilisées pour la résolution de problèmes faciles. La nécessité de disposer d'une solution de bonne qualité (semi optimale) avec un coût de recherche (temps de calcul et espace mémoire) raisonnable a excité les chercheurs à proposer un autre type de méthodes de résolution de problèmes, communément connues par les méthodes approchées. Ces dernières constituent une alternative aux méthodes exactes. En fait, elles permettent de fournir des solutions de très bonne qualité en un temps de calcul raisonnable. De nombreuses méthodes approchées ont été proposées. Elles sont plus pratiques pour la résolution de problèmes difficiles ou de problèmes dont on cherche des solutions en un bref délai. Ces méthodes sont souvent classées en deux catégories: des méthodes heuristiques et des méthodes métaheuristiques (voir figure 2.1).

Les méthodes dites heuristiques sont des méthodes spécifiques à un problème particulier. Elles nécessitent des connaissances du domaine du problème traité. En fait, se sont des règles empiriques qui se basent sur l'expérience et les résultats acquis afin d'améliorer les recherches futures. Plusieurs définitions des heuristiques ont été proposées par plusieurs chercheurs dans la littérature, parmi lesquelles:

Définition 2.1. « Une heuristique (règle heuristique, méthode heuristique) est une règle d'estimation, une stratégie, une astuce, une simplification, ou tout autre type de dispositif qui

limite considérablement la recherche de solutions dans des espaces problématiques importants. Les heuristiques ne garantissent pas des solutions optimales. En fait, elles ne garantissent pas une solution du tout. Tout ce qui peut être dit d'une heuristique utile, c'est qu'elle propose des solutions qui sont assez bonnes la plupart du temps. » [Feigenbaum et Feldman 1963a].

Définition 2.2. « Une méthode heuristique (ou simplement une heuristique) est une méthode qui aide à découvrir la solution d'un problème en faisant des conjectures plausibles mais faillible de ce qui est la meilleure chose à faire. » [Feigenbaum et Feldman, 1963b].

Définition 2.3. « Une heuristique est une règle d'estimation, une stratégie, une méthode ou astuce utilisée pour améliorer l'efficacité d'un système qui tente de découvrir les solutions des problèmes complexes. » [Slagle, 1971].

Définition 2.4. « Les heuristiques sont des règles empiriques et des morceaux de connaissances, utiles (mais non garanties) pour effectuer des sélections différentes et des évaluations. » [Newell, 1980].

Définition 2.5. « Les heuristiques sont des ensembles de règles empiriques ou des stratégies qui fonctionnent, en effet, comme des règles d'estimation. » [Solso, 1979].

Définition 2.6. « Les heuristiques sont des critères, des méthodes ou des principes pour décider qui, parmi plusieurs d'autres plans d'action promet d'être le plus efficace pour atteindre un certain but. » [Pearl, 1984].

Les méthodes dites métaheuristiques sont des méthodes générales, des heuristiques polyvalentes applicables sur une grande gamme de problèmes. Elles peuvent construire une alternative aux méthodes heuristiques lorsqu'on ne connaît pas l'heuristique spécifique à un problème donné. Selon la définition proposée en 1996 par Osman et Laportes:

Définition 2.7. « Une métaheuristique est un processus itératif qui subordonne et guide une heuristique, en combinant intelligemment plusieurs concepts pour explorer et exploiter tout l'espace de recherche. Des stratégies d'apprentissage sont utilisées pour structurer l'information afin de trouver efficacement des solutions optimales, ou presque-optimales. » [Osman et Laporte, 1996].

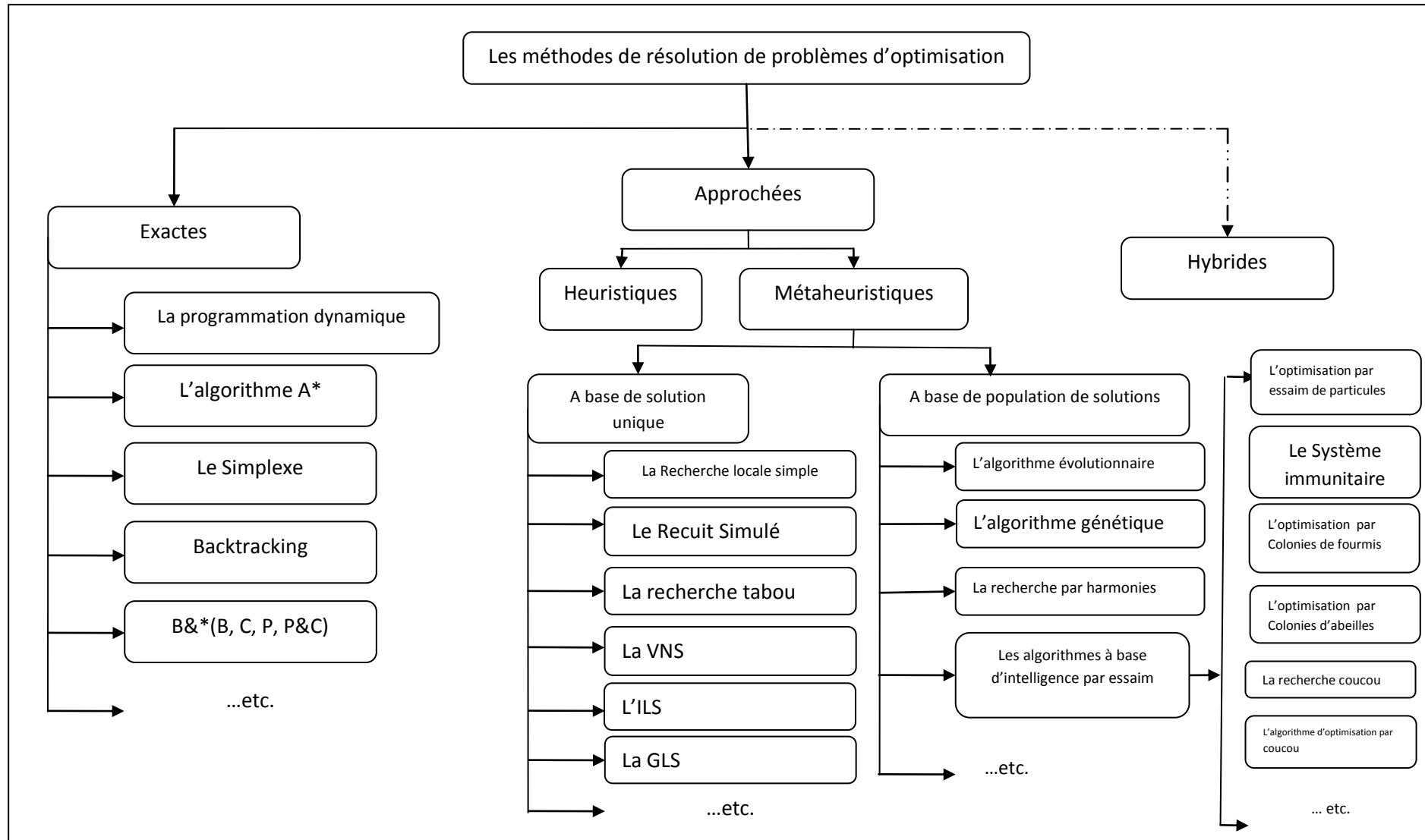


Figure 2.1. Classification de méthodes de résolution de problèmes d'optimisation.

La particularité qui différencie les méthodes métaheuristiques des méthodes heuristiques c'est que les métaheuristiques sont applicables sur de nombreux problèmes. Tandis que, les heuristiques sont spécifiques à un problème donné. De nombreuses métaheuristiques ont été proposées dans la littérature afin de faire face aux différents problèmes en minimisant le coût de la recherche. Les métaheuristiques peuvent être classées en deux catégories: les méthodes à base de solution unique et les méthodes à base de population de solutions. Les méthodes à base de solution unique se basent généralement sur une recherche locale (recherche par voisinage), malheureusement la plupart de ces méthodes souffrent du problème de la convergence de la recherche vers l'optimum local, sauf celles possédant des mécanismes d'échappement. Cependant, les méthodes à base de population de solutions se basent sur une recherche globale sur tout l'espace de recherche. En fait, elles permettent d'échapper au problème de l'optimum local et de déterminer l'optimum global.

L'efficacité d'une métaheuristique dans la résolution d'un problème d'optimisation est liée à sa capacité d'établir un certain équilibre entre l'exploitation des expériences acquises (i.e. les solutions trouvées) au cours de la recherche et l'exploration de l'espace de recherche pour identifier d'autres solutions de très bonne qualité [Stutzle, 1999]. Il est à noter que les termes « exploitation » et « exploration » peuvent être remplacés par les termes « intensification » et « diversification » respectivement. Selon Glover et Laguna, la différence principale entre l'intensification et la diversification réside dans le fait qu'une phase d'intensification se concentre sur le test du voisinage d'une solution élue. Tandis qu'une phase de diversification encourage le processus de recherche à examiner des régions non visitées et à découvrir des solutions différentes des solutions rencontrées dans des points divers [Glover et Laguna, 1997].

L'idée de combiner des méthodes exactes et/ou des méthodes approchées pour créer de nouvelles méthodes a donné naissance à une pseudo classe de méthodes. C'est la classe des méthodes hybrides. Ces dernières ont construit une tendance qui a suscité l'intérêt de plusieurs communautés de chercheurs. Le principe du théorème « No free lunch » stipule qu'aucune méthode n'est efficace pour tous les types de problèmes. En fait, chaque méthode propose des avantages dont on cherche à maximiser et des lacunes dont on cherche à combler. Partant de ce principe, beaucoup de chercheurs ont envisagé la combinaison des méthodes de résolution des problèmes afin de tirer profit des points forts de chacune et de proposer des alternatives plus efficaces et plus performantes. On distingue trois types d'hybridations: des hybridations entre des méthodes exactes, elles consistent à hybrider des méthodes exactes ou

des principes des méthodes exactes avec d'autres méthodes exactes ou des principes d'autres méthodes exactes. Des hybridations entre des méthodes exactes et des méthodes approchées, elles consistent à hybrider des méthodes exactes ou des principes des méthodes exactes avec des méthodes approchées ou des principes des méthodes approchées. Des hybridations entre des méthodes approchées et des méthodes approchées, elles consistent à hybrider des méthodes approchées ou des principes des méthodes approchées avec d'autres méthodes approchées ou des principes d'autres méthodes approchées.

2.2. Les méthodes exactes

L'intérêt des méthodes exactes réside dans le fait qu'elles assurent l'obtention de la solution optimale du problème traité. En fait, elles permettent de parcourir la totalité de l'ensemble de l'espace de recherche de manière à assurer l'obtention de toutes les solutions ayant le potentiel d'être meilleures que la solution optimale trouvée au cours de la recherche. Cependant, les méthodes exactes sont très connues par le fait qu'elles nécessitent un coût de recherche souvent prohibitif en termes de ressources requises. En effet, le temps de recherche et/ou l'espace mémoire nécessaire pour l'obtention de la solution optimale par une méthode exacte sont souvent trop grands, notamment avec des problèmes de grandes tailles. De ce fait, la complexité de ce type d'algorithme croît exponentiellement avec la taille de l'instance à traiter, elle devient très importante face à des problèmes comprenant plusieurs variables, fonctions «objectif»s et/ou critères.

Il existe de nombreuses algorithmes exacts y compris l'algorithme du simplexe, la programmation dynamique, l'algorithme A^* , les algorithmes de séparation et évaluation (Branch and Bound, Branch and Cut, Branch and Price et Branch and Cut and Price), les algorithmes de retour arrière (Backtracking), sans oublier les algorithmes spécifiques au problème traité comme l'algorithme de Johnson [Johnson, 1954] pour la résolution de problèmes d'ordonnancement. Notre vocation n'est plus de relater le principe de différentes méthodes exactes mais plutôt d'en citer quelques unes dans ce qui suit.

2.2.1. L'algorithme de retour arrière (Backtracking)

C'est l'algorithme d'énumération le plus classique. Son principe est de parcourir les valeurs des variables par instanciation en remettant à cause la dernière affectation si une contrainte du problème est violée. Afin d'aboutir à une configuration consistante, une nouvelle valeur est affectée à la variable x_i en respectant son domaine de définition. Le processus est répété à chaque itération jusqu'à l'émergence d'une solution complète. Si toutes

les possibilités ont été essayées sans donner de bonnes solutions, dans ce cas là on dit que le problème est irréalizable. Grâce à son principe qui permet une exploitation complète de l'espace de recherche, cet algorithme garantit l'aboutissement à la solution optimale. Néanmoins, il est très lent: si on se trouve devant un problème de n variables ayant chacune un domaine de définition de k valeurs, il y a k^n combinaisons possibles et donc une complexité et un temps de recherche exponentiels et peut être irraisonnable (dans le cas de problèmes de grande taille).

2.2.2. La méthode Branch and Bound (B&B)

La méthode par séparation et évaluation (nommée Branch and Bound en anglais) est notée B&B. C'est une des méthodes qui permettent la résolution exacte de problèmes d'optimisation, notamment les problèmes d'optimisation combinatoires dont on cherche à minimiser le coût de la recherche. B&B propose un mécanisme de recherche très intelligent, grâce à lequel elle permet une bonne exploitation de l'espace de recherche et l'aboutissement à la solution optimale plus rapidement que d'autres méthodes exactes en combinant deux principes primordiaux: la séparation et l'évaluation.

Le principe de base de la méthode B&B se base sur la technique «Diviser pour régner ». Elle consiste à dissocier le problème en sous problèmes de manière à représenter le problème sous forme d'une arborescence, où chaque nœud correspond à une solution partielle. Les solutions partielles se forment de manière incrémentale en s'enfonçant dans l'arbre. Chacune des solutions partielles potentielles possède une borne supérieure et une autre inférieure. Ces dernières sont utilisées pour couper quelques branches de l'arbre et ainsi éviter d'explorer tout l'arbre. En fait, si l'évaluation partielle d'un nœud x_i a montré que sa qualité est supérieure à la borne supérieure, le sous arbre en question sera élagué; sinon, le nœud sera divisé en sous nœuds. Ce processus se répète tant qu'il reste des branches non parcourues et la recherche continue jusqu'à trouver la solution optimale si elle existe.

L'utilisation de la méthode B&B nécessite:

- ✓ Une solution initiale permettant d'entamer la recherche.
- ✓ Une stratégie permettant la division du problème P en sous problèmes P_i .
- ✓ Une fonction permettant le calcul des différentes bornes.
- ✓ Une stratégie de parcours de l'arbre: parcourir en profondeur, en largeur...etc.

Le point fort de cette méthode réside dans le fait qu'elle ne parcourt pas les sous branches dont on peut savoir à priori qu'elles ne permettent pas d'améliorer la solution rencontrée ce qui est établi grâce aux bornes des nœuds, cela permet de trouver de bonnes solution en un temps de recherche raisonnable.

L'efficacité de la méthode B&B a attiré l'attention de nombreux chercheurs. Par conséquent, plusieurs améliorations de l'algorithme B&B ont été proposées, y compris les algorithmes: Branch and Cut (noté B&C) [Padberg et Rinaldi, 1991], Branch and Price (noté B&P) [Barnhart et al, 2005], Branch and Cut and Price (B&C&P).

2.3. Les métaheuristiques

Dans la vie pratique, on se retrouve souvent confronté à des problèmes de différentes complexités, pour lesquelles on cherche des solutions qui satisfont deux notions antagonistes: la rapidité et la qualité. Devant le coût de recherche prohibitif des méthodes exactes (particulièrement avec des problèmes de grande taille) et la spécificité des heuristiques au problème donné, les métaheuristiques construisent une solution moins exigeante. En fait, elles sont applicables sur une grande variété de problèmes d'optimisation de différentes complexités. En outre, elles permettent de fournir des solutions de très bonne qualité (pas nécessairement optimales) en temps de calcul raisonnable.

La majorité des métaheuristiques sont inspirées des systèmes naturels, nous pouvons citer à titre d'exemple: le recuit simulé qui est inspiré d'un processus métallurgique, les algorithmes évolutionnaires et les algorithmes génétiques qui sont inspirés des principes de l'évolution Darwinienne et de la biologie, la recherche tabou qui s'inspire de la mémoire des êtres humains, les algorithmes basés sur l'intelligence d'essaim comme l'algorithme d'optimisation par essaim de particules, l'algorithme de colonies de fourmis, l'algorithme de colonies d'abeilles, la recherche coucou et l'algorithme d'optimisation par coucou qui s'inspirent du comportement social de certaines espèces évoluant en groupe.

L'ensemble des métaheuristiques proposées dans la littérature sont partagées en deux classes: des métaheuristiques à base de solution unique et des métaheuristiques à base de population de solutions. Nous présentons quelques métaheuristiques des deux classes dans ce qui suit de cette section.

2.3.1. Les métaheuristiques à base de solution unique

Les métaheuristiques à base de solution unique débutent la recherche avec une seule solution initiale. Elles se basent sur la notion du voisinage pour améliorer la qualité de la solution courante. En fait, la solution initiale subit une série de modifications en fonction de son voisinage. Le but de ces modifications locales est d'explorer le voisinage de la solution actuelle afin d'améliorer progressivement sa qualité au cours des différentes itérations. Le voisinage de la solution s englobe l'ensemble des modifications qui peuvent être effectuées sur la solution elle-même. La qualité de la solution finale dépend particulièrement des modifications effectuées par les opérateurs de voisinages. En effet, les mauvaises transformations de la solution initiale mènent la recherche vers la vallée de l'optimum local d'un voisinage donné (peut être un mauvais voisinage) ce qui bloque la recherche en fournissant une solution de qualité insuffisante.

De nombreuses méthodes à base de solution unique ont été proposées dans la littérature. Parmi lesquelles: la descente, le recuit simulé, la recherche tabou, la recherche à voisinage variable (VNS: Variable Neighbourhood Search), la recherche locale réitérée (ILS: Iterated Local Search), la recherche locale guidée (GLS: Guided Local Search)...etc.

2.3.1.1. La recherche locale simple (la descente)

La recherche locale simple ou la descente est un algorithme d'amélioration très ancien. Son principe consiste à explorer le voisinage de la solution courante afin d'améliorer sa qualité progressivement, comme le montre la figure 2.2 qui représente un schéma d'évolution d'une recherche locale simple. A chaque itération du processus d'amélioration, l'algorithme modifie un ensemble de composantes de la solution courante pour permettre le déplacement vers une solution voisine de meilleure qualité. Le processus est répété itérativement jusqu'à la satisfaction du critère d'arrêt. Il est à noter qu'il existe trois types de la descente: la descente déterministe, la descente stochastique et la descente vers le premier meilleur voisin. L'algorithme 2.1 résume les étapes de l'algorithme général de la descente. L'algorithme entame la recherche par la construction d'une solution initiale s et l'évaluation de sa qualité $f(s)$. Ensuite il commence l'ensemble des étapes du processus d'amélioration suivantes:

- Modifier s pour obtenir une nouvelle solution s' de meilleure qualité que s et qui appartient à son voisinage. Le choix de la fonction de voisinage est important. Il dépend de l'objectif à atteindre. En fait, si l'objectif est de minimiser le coût on doit suivre la direction de la vallée. Sinon (dans le cas de maximisation), on doit suivre la direction du sommet.

- Evaluer la qualité $f(s')$ de la nouvelle solution s' .
- Remplacer s par s' , si s' est de meilleure qualité.

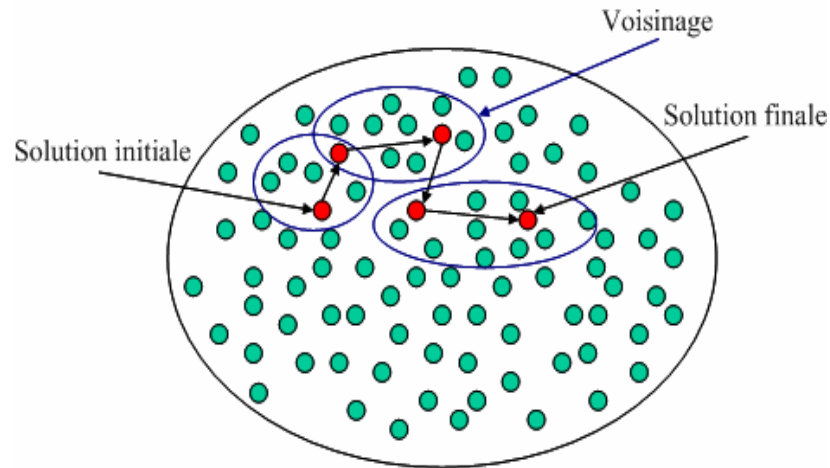


Figure 2.2. Un schéma d'évolution d'une recherche locale simple [Naimi ,2008].

Le processus d'amélioration sera répété jusqu'à arriver au cas où toutes les voisines candidates sont de piètre qualité que la solution courante. Autrement dit, la recherche s'arrête lorsqu'un optimum local est atteint.

L'avantage de la recherche locale simple revient à sa simplicité et à sa rapidité. Cependant, son problème réside dans le fait d'être bloquée par le premier optimum local rencontré. Ce dernier n'est pas forcément l'optimum global. Il peut être très loin de l'optimum global.

Algorithme 2.1. La recherche locale simple (la descente)

Début

Construire une solution initiale s ;

Calculer la fitness $f(s)$ de s ;

Tant que la condition d'arrêt n'est pas vérifiée **faire**

 Modifier s pour obtenir une nouvelle solution voisine s' ;

 Calculer $f(s')$;

Si $f(s')$ est meilleure que $f(s)$ **alors**

 Remplacer s par s' ;

Fin Si

Fin Tant que

Retourner s ;

Fin

2.3.1.2. Le recuit simulé

L'algorithme du recuit simulé a été proposé par Kirkpatrick, Gelatt et Vecchi [Kirkpatrick et al, 1983]. Son principe se base sur la procédure du recuit des métaux utilisée par les métallurgistes. Ces derniers chauffent à blanc le métal, puis ils laissent l'alliage se refroidir très lentement afin d'aboutir à un alliage sans défauts. En fait, les thermodynamiciens ont remarqué qu'une baisse brutale de la température d'un liquide entraîne une reproduction d'un optimum local, i.e. une structure amorphe. Alors qu'une baisse progressive de la température du liquide permet d'aboutir à un optimum global, i.e. une structure bien construite. C'est l'idée prise en considération par les métallurgistes qui savent que si le métal refroidit trop vite, il contiendra beaucoup de défauts microscopiques et s'il refroidit lentement ils obtiendront une structure bien ordonnée. La métaheuristique du recuit simulé s'inspire de l'algorithme de Métropolis [Métropolis et al, 1953], dont le principe (pour un problème de maximisation) peut être résumé comme suit:

- Entamer la recherche avec une solution initiale s ;
- Affecter une valeur initiale à la température T ;
- Calculer la fitness $f(s)$ de la solution initiale s .
- Générer une solution s' voisine de s ;
- Calculer la fitness $f(s')$ de s' ;
- Calculer l'écart de qualité (fitness) entre la solution s et la solution s' comme suit:

$$\Delta(f) = f(s') - f(s) \quad (2.1)$$

- **Si** $\Delta(f) \geq 0$ **alors** $s \leftarrow s'$
- **Sinon** générer un nombre aléatoire $r \in [0,1]$;
- **Si** $r < \exp(\Delta(f)/T)$ **alors** $s \leftarrow s'$.

Un schéma général de l'algorithme du recuit simulé (RS) est présenté dans l'algorithme 2.2. Le recuit simulé permet d'accepter une solution de piètre qualité que la solution courante afin de diversifier la recherche et échapper au piège de l'optimum local. Le fait d'accepter des solutions de mauvaises qualités peut mener la recherche vers la meilleure solution (l'optimum global) car cette dernière peut faire partie du voisinage d'une mauvaise solution et non pas d'une bonne solution (i.e. la solution courante qui est de meilleure qualité que sa voisine) qui peut représenter un optimum local. Cependant, l'acceptation de solutions de mauvaises qualités peut causer une perte de la meilleure solution rencontrée au cours de la recherche et entraîne une convergence vers une solution de mauvaise qualité qu'une autre déjà trouvée. Ce

problème peut être facilement résolu en ajoutant une variable permettant la mémorisation de la meilleure solution trouvée.

Algorithme 2.2. Le recuit simulé

Début

Construire une solution initiale s ;

Calculer la fitness $f(s)$ de s ;

Initialiser une valeur de la température T ;

$s_{\text{best}} = s$;

Tant que la condition d'arrêt n'est pas satisfaite **faire**

 Générer une solution s' voisine de s ;

 Calculer $f(s')$;

 Calculer $\Delta(f) = f(s') - f(s)$;

Si $\Delta(f) \geq 0$ **alors** // cas de maximisation

$s_{\text{best}} = s'$;

$s = s'$;

Sinon Si $r < \exp(\frac{\Delta(f)}{T})$ **alors**

$s = s'$;

Fin Si

 Décroître la température T ;

Fin Tant que

Retourner s_{best} ;

Fin

L'acceptation d'une solution de mauvaise qualité est établi en fonction de deux facteurs: l'écart de qualité entre la solution courante et sa voisine d'un coté, et la température de l'autre coté. Plus la température est élevée, plus la probabilité d'accepter des solutions de mauvaises qualités est forte. La valeur initiale de la température T décroît au cours de la recherche pour tendre vers le 0. Ce paramètre (i.e. la température) a un effet non négligeable sur la performance de l'algorithme. Il doit être soigneusement ajusté tout au long de la recherche. En fait, un refroidissement rapide de la valeur de T peut entrainer une convergence prématurée de l'algorithme vers un optimum local de mauvaise qualité. Tandis qu'un refroidissement lent peut mener la recherche vers une solution de bonne qualité. Cependant, le refroidissement trop lent nécessite un temps de calcul élevé. En outre, le choix de la valeur initiale de T joue un rôle primordial dans le processus de recherche. Il dépend de la qualité de la solution initiale. Si cette dernière est choisie aléatoirement, il sera préférable d'affecter une

valeur élevée à T pour donner plus de chance d'aboutissement à de bonnes solutions. Si le choix de la valeur initiale de la solution est expérimental ou empirique, la température initiale peut être basse.

Le recuit simulé est un algorithme basé sur la recherche à voisinage (recherche locale). C'est un algorithme simple, facile à implémenter et à adapter à un grand nombre de problèmes : traitement d'image, sac à dos, voyageur de commerce, ordonnancement, etc. Comparé avec la recherche locale simple, le RS permet de se sauver du piège de l'optimum local et d'offrir des solutions de bonne qualité comme il présente une souplesse d'intégration des contraintes liées au problème traité. En revanche, cet algorithme dispose d'un nombre important de paramètres (température initiale, paramètres liés à la fonction d'ajustement de la température... etc.) à ajuster. En outre, le RS est un algorithme lent surtout avec les problèmes de grande taille.

2.3.1.3. La recherche tabou

La recherche tabou (RT) est une métaheuristique à base d'une solution unique. Elle a été proposée en 1986 par Glover [Glover, 1986]. RT est une méthode de recherche locale avancée, elle fait appel à un ensemble de règles et de mécanismes généraux pour guider la recherche de manière intelligente [Glover et Laguna, 1997]. L'optimisation de la solution avec la recherche tabou se base sur deux astuces: l'utilisation de la notion du voisinage et l'utilisation d'une mémoire permettant le guidage intelligent du processus de la recherche. En parcourant le voisinage de la solution courante s , la recherche tabou ne s'arrête pas au premier optimum local rencontré. Elle examine un échantillonnage de solution du voisinage de s et retient toujours la meilleure solution voisine s' , même si celle-ci est de piètre qualité que la solution courante s , afin d'échapper de la vallée de l'optimum local et donner au processus de la recherche d'autres possibilités d'exploration de l'espace de recherche afin de rencontrer l'optimum global. En fait, les solutions de mauvaise qualité peuvent avoir de bons voisinages et donc guider la recherche vers de meilleures solutions. Cependant, cette stratégie peut créer un phénomène de cyclage (i.e. on peut revisiter des solutions déjà parcourues plusieurs fois). Afin de pallier à ce problème, la recherche tabou propose l'utilisation d'une mémoire permettant le stockage des dernières solutions rencontrées pour ne pas les visiter dans les prochaines itérations et tomber dans le problème du cyclage répétitif. Cette mémoire est appelée « la liste tabou », d'où le nom de la métaheuristique tabou. La taille de la liste tabou est limitée, ce qui empêche l'enregistrement de toutes les solutions rencontrées. C'est la raison pour laquelle la liste tabou procède comme une pile FIFO, où la plus ancienne solution

sera écartée pour laisser place à la dernière solution rencontrée. Le but de faciliter la gestion de la liste tabou en termes de temps de calcul ou d'espace mémoire nécessaires à pousser les chercheurs à proposer de ne conserver dans la liste que des attributs (i.e. caractéristiques) des solutions au lieu des solutions complètes. Particulièrement, dans le cas où le nombre de variables est très élevé. Toutefois, l'utilisation de la liste tabou peut mener à un blocage dans certains cas. En fait, les nouvelles solutions qui possèdent des attributs des solutions tabou, seront considérées tabou aussi même si elles ne sont pas encore été visitées. Pour remédier à ce problème, on a défini une technique appelée « critère d'aspiration » permettant de sortir du blocage causé par la ressemblance des attributs des solutions tabou. Le critère d'aspiration permet d'omettre le statut tabou sur une solution lorsque certaines circonstances sont respectées. Un critère d'aspiration très classique consiste à autoriser une solution tabou si celle-ci est l'une des meilleures solutions rencontrées depuis le démarrage de la recherche.

Algorithme 2.3. La recherche tabou

Début

Construire une solution initiale s ;

Calculer la fitness $f(s)$ de s ;

Initialiser une liste tabou vide ;

$s_{best} = s$;

Tant que le critère d'arrêt n'est pas vérifié **faire**

 Trouver la meilleure solution s' dans le voisinage de s qui ne soit pas tabou ou qui vérifie le critère d'aspiration ;

 Calculer $f(s')$;

Si fitness de (s') est meilleure que fitness de (s_{best}) **alors**

$s_{best} = s'$;

Fin Si

 Mettre à jour la liste tabou ;

$s = s'$;

Fin Tant que

Retourner s_{best} ;

Fin

La recherche tabou a été largement utilisée pour la résolution de problèmes d'optimisation difficiles comme: le problème du voyageur de commerce, les problèmes du sac à dos, les problèmes de routage, d'ordonnancement, de coloration de graphes, d'exploitation géologique, etc. C'est une méthode facile à mettre en œuvre, rapide, donne souvent de bons

résultats et permet de se sauver du premier optimum local rencontré contrairement à la méthode de la recherche locale simple. En revanche, la liste tabou demande de ressources importantes si elle est de grande taille. En fait, elle demande une gestion soigneuse de sa taille et de ce qu'elle doit contenir comme informations sur les solutions trouvées, de manière à ne pas être très exigeante en temps de parcours et d'espace mémoire requis et aussi pour éviter les confusions causées par la ressemblance des attributs des solutions qui bloquent la recherche.

Il est à noter qu'il existe plusieurs variantes de la recherche tabou. Ces variantes dépendent essentiellement du choix du voisinage et de la manière de gérer la liste tabou. L'algorithme 2.3 représente un schéma général de l'algorithme de la recherche tabou.

2.3.1.4. La recherche à voisinage variable

La recherche à voisinage variable est une méthode basée sur la recherche locale. Elle a été introduite en 1997 par Mladenovic et Hansen [Mladenovic et Hansen, 1997]. La recherche à voisinage variable est notée par RVV ou encore VNS (de l'anglais: **V**ariable **N**eighbourhood **S**earch). Elle se base sur un simple principe de recherche fondé sur le changement systématique du voisinage. A l'opposé d'autres méthodes basées sur la recherche locale, la VNS n'utilise pas un seul voisinage pour exploiter la solution courante. Par contre, elle utilise plusieurs voisinages dans un ordre prédéfini. Son principe de base consiste à entamer la recherche avec un voisinage V_1 d'une solution initiale s . Lorsque la recherche se bloque dans un optimum local, on continue la recherche avec un autre voisinage puis on revient au premier voisinage après chaque amélioration. La recherche continue en passant d'un voisinage à un autre à chaque blocage dans la vallée de l'optimum local du voisinage utilisé.

L'idée de parcourir un ensemble de voisinage permet de sauver la recherche du blocage causé par le piège de l'optimum local. Cette stratégie permet de diversifier l'exploration de l'espace de recherche et donne au processus de recherche plus de chances de rencontrer l'optimum global. Cependant, le choix et l'ordre de parcours des différents voisinages jouent un rôle non négligeable dans la performance de l'algorithme. En fait, s'il est bien étudié, la recherche converge vers la solution optimale (l'optimum global), sinon la convergence de la solution optimale ne sera pas garantie.

L'algorithme 2.4 représente un pseudo code résumant les étapes du schéma général de la recherche à voisinage variable. L'algorithme commence la recherche par une solution initiale s . Puis, il génère une solution s' du premier voisinage de la solution s . Après évaluation de la

qualité de s et celle de s' , si s' est meilleure que s , l'algorithme remplace s par s' et il continue l'amélioration de la solution courante en générant d'autres solutions du même voisinage. Sinon, l'algorithme continue la recherche avec le deuxième voisinage jusqu'à arriver à la satisfaction du critère d'arrêt.

Algorithme 2.4. La recherche à voisinage variable

Début

Définir la structure du voisinage N_k , où $k \in \{1, 2, \dots, k_{\max}\}$;

Construire une solution initiale s ;

Calculer la fitness $f(s)$ de s ;

Tant que la condition d'arrêt n'est pas vérifiée **faire**

$k=1$;

Tant que $k \leq k_{\max}$ **faire**

 Générer une solution s' voisine de s , où $s' \in N_k$;

 Calculer $f(s')$;

Si $f(s)$ est meilleure que $f(s')$ **alors**

$s = s'$;

$k=1$;

Sinon

$k=k+1$;

Fin Si

Fin Tant que

Fin Tant que

 Retourner s ;

Fin

2.3.2. Les métaheuristiques à base de population de solutions

Les métaheuristiques à base de population de solutions débutent la recherche avec une panoplie de solutions. Elles s'appliquent sur un ensemble de solutions afin d'en extraire la meilleure (l'optimum global) qui représentera la solution du problème traité. L'idée d'utiliser un ensemble de solutions au lieu d'une seule solution renforce la diversité de la recherche et augmente la possibilité d'émergence de solutions de bonne qualité. Une grande variété de méthodes basées sur une population de solutions a été proposée dans la littérature, commençant par les algorithmes évolutionnaires, passant par les algorithmes génétiques et arrivant aux algorithmes à base d'intelligence par essaims (l'algorithme d'optimisation par essaim de particules, l'algorithme de colonies de fourmis, l'algorithme de colonies d'abeilles,

la recherche coucou, l'algorithme d'optimisation par coucou...) qui ont connus une investigation remarquable ces deux dernières décennies.

2.3.2.1. Les algorithmes évolutionnaires

L'apparition des algorithmes évolutionnaires revient aux années soixante. Elles sont inspirées du principe de l'évolution naturelle des espèces biologiques. Précisément, de la théorie de l'évolution exposée par Charles Darwin [Darwin, 1859], qui se base sur le principe de la sélection naturelle. Le principe de cette dernière stipule que les individus bien adaptés à l'environnement ont plus de chances de se reproduire et de survivre que les autres individus. En fait, la combinaison des caractéristiques des individus peut former au fil des générations de nouveaux individus de plus en plus mieux adaptés à leur environnement et qui peuvent avoir plus de chances de survivre que leurs parents.

La découverte de l'ADN et de son rôle en 1944 par Tomas Avery a renforcé le principe de la théorie d'évolution proposée par Charles Darwin. En effet, chaque cellule de l'organisme de l'individu possède des chromosomes, ces derniers sont composés des molécules d'ADN qui contiennent tout le patrimoine génétique de l'individu. La reproduction qui se base sur la combinaison du patrimoine génétique des individus, consiste à échanger des gènes entre individus. L'échange du patrimoine génétique peut contribuer dans l'amélioration des caractéristiques des individus de la nouvelle génération et donc donne plus de chances de survie aux membres de la nouvelle génération.

Les algorithmes évolutionnaires s'inspire de l'évolution naturelle des êtres vivants. Ils adoptent une sorte d'évolution artificielle pour améliorer la qualité des individus de la population. En fait, ils font évoluer itérativement une population d'individus. Ces derniers représentent des solutions du problème traité. Les qualités des individus sont mesurées à chaque itération du processus de d'évolution. En fonction de leurs qualités, les meilleurs individus seront sélectionnés pour subir des combinaisons qui permettent la production d'une nouvelle population (dite: population d'enfants). Les individus (ou une partie des individus) de la nouvelle population vont remplacer les individus de la population courante (dite: population de parents) pour construire la nouvelle génération d'individus. Le processus d'évolution d'un algorithme évolutionnaire est basé sur trois opérations principales: la sélection, la reproduction et l'évaluation:

- **La Sélection:** Cette opération s'intervient dans deux phases. Elle est appliquée au premier lieu pour choisir les meilleurs individus parents qui vont se reproduire pour construire de

nouveaux individus enfants. Ensuite, elle est appliquée à la fin de chaque itération pour opter pour les individus qui vont survivre et construire la nouvelle population.

- **La reproduction:** Cette opération est en général composée de deux autres opérations: le croisement et la mutation. Elle permet la génération de nouveaux individus en combinant (phase de croisement) les caractéristiques des individus sélectionnés puis en appliquant quelques modifications de certains individus (phase de mutation) pour améliorer leurs qualités.
- **L'évaluation:** Cette opération consiste à mesurer la qualité de chaque individu (calculer la fitness des individus).

La figure 2.3 représente un diagramme résumant les étapes générales d'un algorithme évolutionnaire.

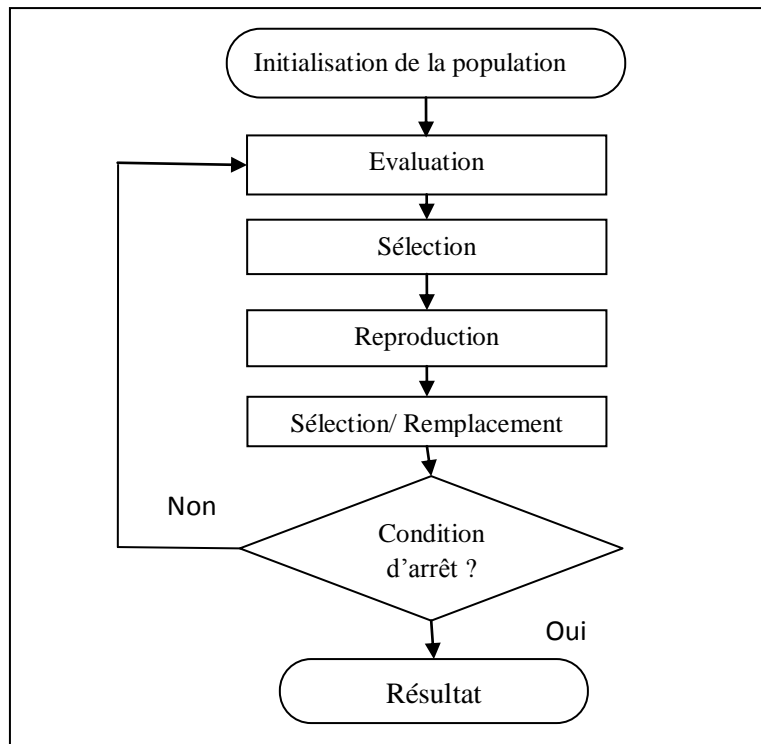


Figure 2.3. Organigramme d'un algorithme évolutionnaire

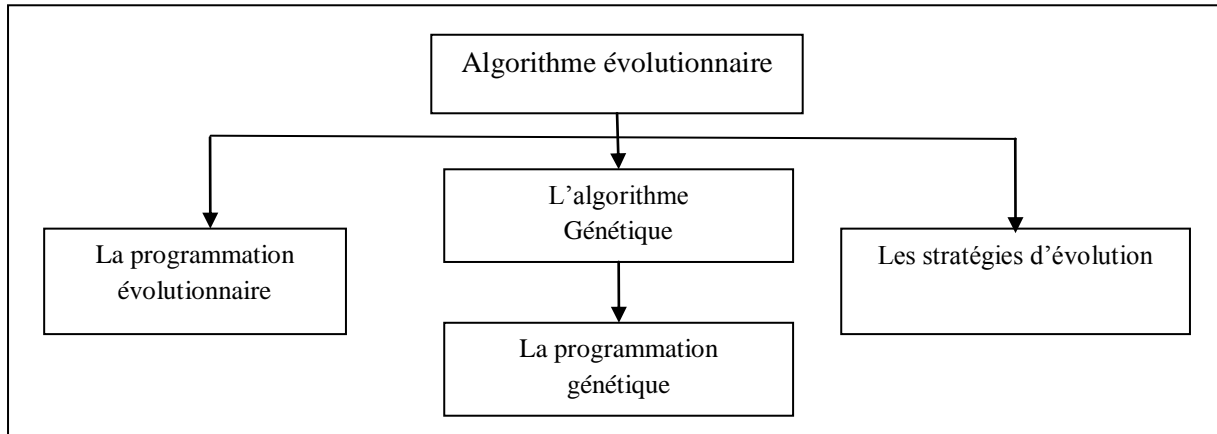


Figure 2.4. Les types des algorithmes évolutionnaires.

Les algorithmes évolutionnaires forment une classe principale de trois sous classes d'algorithmes (voir figure 2.4): les stratégies d'évolution, la programmation évolutionnaire et les algorithmes génétiques.

- **Les stratégies d'évolution** [Rechenberg, 1965; Beyer, 2001]: Ont été conçues pour la résolution des problèmes d'optimisation continus.
- **La programmation évolutionnaire** [Fogel et al, 1966]: Les algorithmes de la classe de la programmation évolutionnaire sont conçus pour faire évoluer des structures d'automates à état fini.
- **L'algorithme génétique** [Holland, 1973; Goldberg, 1989]: Il sera bien détaillé dans la section suivante. Il est à noter qu'il existe une sous classe de la classe des algorithmes génétiques appelée « La programmation génétique » [Koza, 1996]. Cette dernière utilise des structures arborescentes pour représenter les individus de la population.

2.3.2.2. L'algorithme génétique

L'algorithme génétique représente une célèbre métaheuristique évolutionnaire. Il a été proposé par Jhon Holland en 1975 [Holland, 1975]. L'algorithme génétique s'inspire des mécanismes biologiques tels que les lois de Mendel et la théorie de l'évolution proposée par Charles Darwin [Darwin, 1859]. Son processus de recherche de solutions à un problème donné imite celui des êtres vivants dans leur évolution. Il utilise le même vocabulaire que celui de la biologie et la génétique classique, on parle donc de: gène, chromosome, individu, population et génération.

- ✓ **Un gène:** est un ensemble de symboles représentant la valeur d'une variable. Dans la plupart des cas, un gène est représenté par un seul symbole (un bit, un entier, un réel ou un caractère).
- ✓ **Un chromosome:** est un ensemble de gènes, présentés dans un ordre donné de manière qui prend en considération les contraintes du problème à traiter. Par exemple, dans le problème du voyageur de commerce, la taille du chromosome est égale au nombre de villes à parcourir. Son contenu représente l'ordre de parcours de différentes villes. En outre, on doit veiller à ce qu'une ville (représentée par un nombre ou un caractère par exemple) ne doit pas figurer dans le chromosome plus qu'une seule fois.
- ✓ **Un individu:** est composé d'un ou de plusieurs chromosomes. Il représente une solution possible au problème traité.
- ✓ **Une population:** est représentée par un ensemble d'individus (i.e. l'ensemble des solutions du problème).
- ✓ **Une génération:** est une succession d'itérations composées d'un ensemble d'opérations permettant le passage d'une population à une autre.

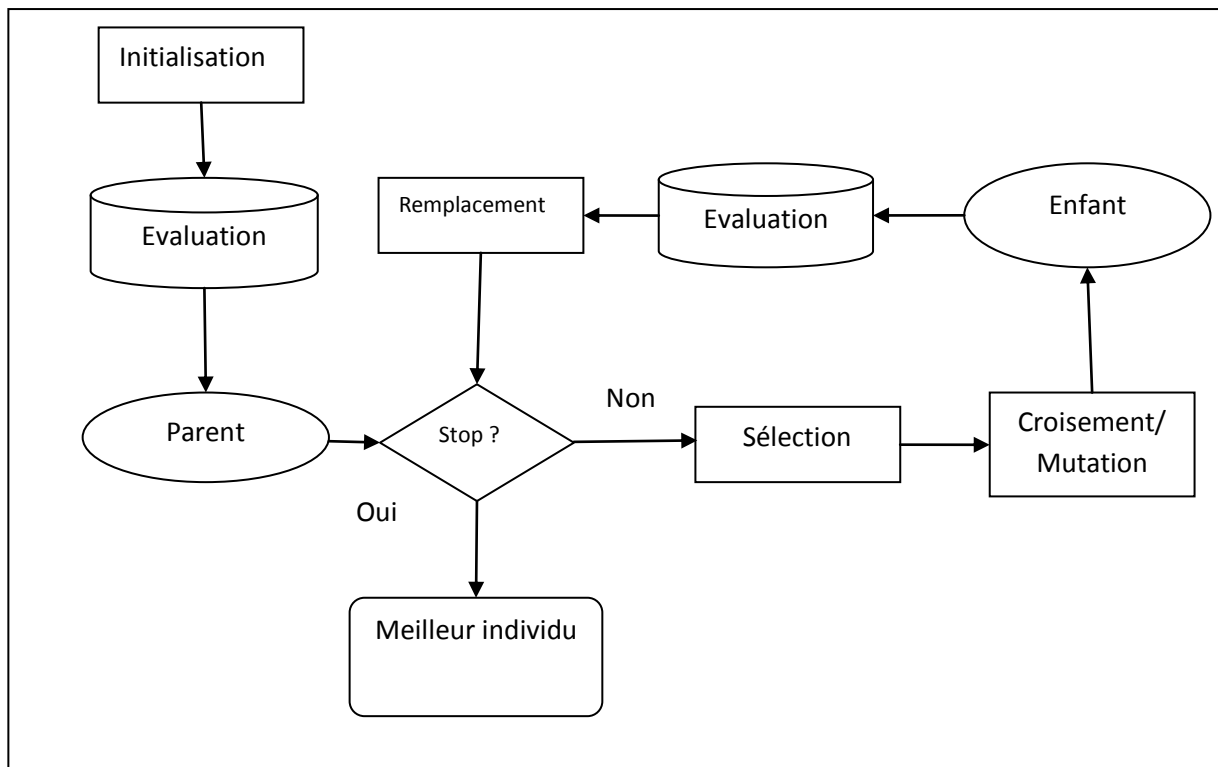


Figure 2.5. Démarche d'un algorithme génétique.

[Ghali, 2005]

L'algorithme génétique fait évoluer une population composée d'un ensemble d'individus pendant un ensemble de génération jusqu'à ce qu'un critère d'arrêt soit vérifié. Le passage d'une population à une autre est réalisé grâce à des opérations d'évaluation, de sélection, de reproduction (croisement et mutation) et de remplacement.

L'algorithme commence la recherche avec un ensemble d'individus. A chaque itération de la procédure de recherche, les meilleurs individus sont sélectionnés pour survivre et se reproduire. La sélection des individus est fondée sur leurs qualités qui sont mesurées à partir d'une fonction appelée « fonction objectif ou fonction fitness ». Ensuite, les individus (appelés parents) sont sélectionnés pour subir des opérateurs de croisement et de mutation permettant la génération d'une autre population d'individus (appelés enfants). Les individus de la nouvelle population seront évalués pour remplacer une partie des individus de la population courante. La figure 2.5 illustre un schéma général des étapes du processus de recherche de l'algorithme génétique.

Le processus de recherche de l'algorithme génétique est fondé sur les opérateurs suivants:

- **Un opérateur de codage des individus:** Il permet la représentation des chromosomes représentant les individus.
- **Un opérateur d'initialisation de la population:** Il permet la production des individus de la population initiale. Malgré que cet opérateur ne s'intervient qu'une seule fois et au début de la recherche, mais il joue un rôle non négligeable dans la convergence vers l'optimum global. En fait, le choix de la population initiale peut rendre la recherche de la solution optimale du problème traité plus facile et plus rapide.
- **Un opérateur de sélection:** Il permet de favoriser la reproduction des individus qui ont les meilleures fitness (i.e. les meilleures qualités).
- **Un opérateur de croisement:** Il permet l'échange des gènes entre parents (deux parent en général), pour créer 1 ou deux enfants en essayant de combiner les bonnes caractéristiques des parents. Le but de cet opérateur est la création de nouveaux individus en exploitant l'espace de recherche.
- **Un opérateur de mutation:** Il consiste à modifier quelques gènes des chromosomes des individus, dans le but d'intégrer plus de diversité au sein du processus de la recherche.

- **Un opérateur d'évaluation:** Il permet de valoriser la qualité des individus, en se basant sur la fonction «objectif» (la fonction fitness) qui permet de calculer la qualité de chaque individu.

En outre des différents opérateurs permettant de guider la recherche par l'algorithme génétique, ce dernier nécessite un certain nombre de paramètres de base, sur lesquels dépendent les différents opérateurs cités en dessus. Ces paramètres doivent être fixés à l'avance, ils jouent un rôle très important dans la performance de l'algorithme. On parle de: la taille de la population, la probabilité de croisement, la probabilité de mutation et le nombre maximum de génération.

- **La taille de la population:** Elle représente le nombre d'individus de la population. S'il est trop grand, le processus de recherche demande un coût de recherche élevé, que se soit en termes d'espace mémoire ou du temps de calcul nécessaires. Cependant, s'il est trop petit, l'algorithme risque d'être tomber dans le cas de la convergence prématurée à cause du manque de la diversité au sein de la population. Il est préférable donc de choisir une taille moyenne en prenant en considération l'instance du problème à traiter.
- **La probabilité de croisement:** Elle représente la probabilité d'échange de patrimoine (i.e. les gènes) entre deux individus (ou plus). Plus elle est grande, plus elle permet la génération de nouveaux enfants qui peuvent être meilleurs que leurs parents.
- **La probabilité de mutation:** Elle est en général faible, dans le but d'échapper aux possibilités de modifications radicales des solutions, particulièrement, des solutions de bonnes qualités qui ne nécessitent que peu d'amélioration pour passer aux solutions optimales.
- **Le nombre maximum de génération:** Ce paramètre peut jouer le rôle d'un critère d'arrêt. Il peut construire un obstacle pour l'algorithme. En fait, il peut empêcher les différents opérateurs d'aboutir à la meilleure solution s'il est trop petit. Comme il peut engendrer un temps de calcul prohibitif dans le cas ou il est trop grand. Ainsi, le choix de sa valeur peut se baser sur des tests préliminaires.

L'algorithme 2.5 représente l'ensemble des étapes de l'algorithme génétique.

Algorithme 2.5. L'algorithme génétique

Début

Initialiser les paramètres nécessaires ;
Initialiser une population de N individus ;
Evaluer les N individus ;

Tant que la condition d'arrêt n'est pas satisfaite faire

Utiliser l'opérateur de sélection pour sélectionner K individus ;
Appliquer l'opérateur de croisement sur les K individus avec la probabilité P_c ;
Appliquer l'opérateur de mutation sur les individus avec la probabilité P_m ;
Utiliser l'opérateur d'évaluation pour évaluer les enfants obtenus ;
Utiliser l'opérateur de sélection pour remplacer quelques individus parents par des individus enfants ;

Fin Tant que

Retourner la ou les meilleures solutions ;

Fin

L'algorithme génétique a été utilisé pour la résolution de nombreux problèmes académiques et/ou industriels. Son avantage principal est qu'il permet une bonne combinaison entre l'exploitation de solutions et l'exploration de l'espace de recherche. Cela est établi en fonction des opérateurs de croisement et de mutation respectivement. Cependant, son inconvénient réside dans deux points: un temps de calcul assez important pour pouvoir converger vers la solution optimale (quoique l'apparition des ordinateurs de plus en plus puissants a permis de remédier à ce problème). Et le nombre de paramètres importants (taille de la population, paramètres de sélection, paramètres de croisement, paramètres de mutation, critère d'arrêt...).

2.3.2.2.1. Le codage des individus

Avant de générer l'ensemble des individus représentant la population initiale, il faut d'abord penser à définir le codage convenable des individus de la population. Sachant que ces derniers sont représentés par un ou plusieurs chromosomes. Chaque chromosome est composé d'un ensemble de gènes et chaque gène exprime un paramètre ou une information. Tout ça nécessite la définition d'un codage permettant la modélisation et la manipulation des solutions (les individus) de la population. Le choix du codage joue un rôle très important. En fait, il doit permettre la représentation des différentes solutions possibles, comme il doit prendre en

considération le type du problème à traiter (continu, discret, binaire...). On distingue les types de codage suivants:

1. Le codage binaire

Le codage binaire consiste à utiliser des bits (0 ou 1) pour représenter les différentes solutions. Dans le sens où, les gènes sont représentés par des bits et les chromosomes sont représentés par des chaînes de bits. Le type de ce codage s'adapte bien aux problèmes de type binaire, comme le problème MAX SAT ou les problèmes du sac à dos. Prenons l'exemple du problème du sac à dos dans sa forme la plus simple (unidimensionnelle). Supposant, que nous devons sélectionner un sous ensemble d'objets parmi un ensemble de 10 objets. Les solutions peuvent être facilement représentées par des chaînes binaires de taille 10, où chaque bit représente l'état d'un objet: le bit 1 pour dire que l'objet est sélectionné, le bit 0 pour dire que l'objet est écarté. La figure 2.6 représente une des solutions du problème. Selon la figure 2.6, les objets 2, 3, 5, 8, 9, et 10 sont sélectionnés. Tandis que les autres (i.e. 1, 4, 6 et 7) sont écartés.

0	1	1	0	1	0	0	1	1	1
---	---	---	---	---	---	---	---	---	---

Figure 2.6. Codage binaire d'un chromosome.

2. Le codage réel

Ce type de codage est beaucoup plus efficace pour représenter des problèmes de type continu. Il représente les solutions par des suites de type réel, comme le montre la figure 2.7.

0.23	1.25	10.5	0.48	3.00	-20.87
------	------	------	------	------	--------

Figure 2.7. Codage réel d'un chromosome.

On peut distinguer un autre type spécial du codage réel, c'est le codage entier ou discret. Il utilise des entiers au lieu de réels. Ce type est beaucoup plus efficace pour la représentation de certain type de problèmes discrets, comme les problèmes d'ordonnancement et le problème du voyageur de commerce. Prenons l'exemple du problème du voyageur de commerce où un voyageur de commerce doit parcourir un ensemble de 10 villes en choisissant le plus court chemin permettant de passer par chacune une seule fois. La figure 2.8 représente une solution du problème où chaque gène représente une ville et l'ensemble des gènes représente l'ordre de parcours de l'ensemble des 10 villes.

2	6	3	7	5	4	10	8	1	9
---	---	---	---	---	---	----	---	---	---

Figure 2.8. Codage entier d'un chromosome.

3. Le codage à caractère

Il s'agit d'utiliser une suite de caractères différents pour représenter le chromosome, comme le montre la figure 2.9.

A	V	-	G	L	M
---	---	---	---	---	---

Figure 2.9. Codage à caractère d'un chromosome.

4. Le codage arborescent

Ce type de codage utilise une représentation arborescente des individus avec un ensemble de nœuds, où l'ensemble des nœuds représente un individu, comme le montre la figure 2.10.

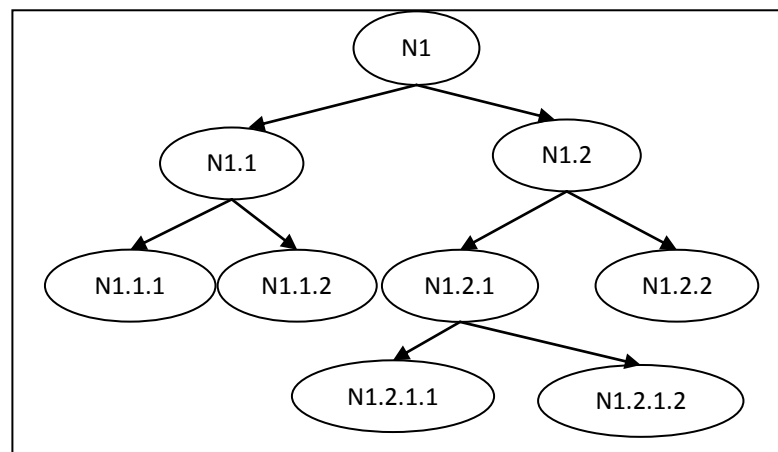


Figure 2.10. Codage arborescent d'un chromosome.

2.3.2.2.2. La sélection

La sélection joue un rôle très important dans le processus de recherche de l'algorithme génétique. En effet, elle s'intervient dans deux étapes de chaque itération:

- Au début de chaque itération pour sélectionner les individus parents qui vont se reproduire entre eux.
- A la fin de chaque itération pour sélectionner les individus enfants qui vont remplacer les individus parents dans le but de créer une nouvelle population en respectant la taille de la population pour ne pas entraîner une explosion démographique de la population.

La sélection des individus dépend essentiellement de leurs qualités (fitness). En effet, les meilleurs individus sont sélectionnés pour se reproduire, survivre et donc remplacer les moins

performants. On peut distinguer plusieurs types de sélections comme: la sélection à la roulette, la sélection par tournoi et la sélection élitiste [Layeb, 2010].

2.3.2.2.3. Le croisement

Le croisement consiste à générer un ou deux nouveaux individus à partir d'un couple de parents choisi par l'opérateur de sélection. C'est un opérateur qui permet la création de nouveaux individus en combinant les gènes des individus parents. Ainsi, après répartition de l'ensemble des individus (par sélection en utilisant l'opérateur de sélection) en couples, ces derniers subissent un opérateur d'échange de gènes permettant la génération de nouveaux individus. Cet échange est souvent fait au hasard, mais il permet des fois de remplacer les mauvais gènes d'un parent par les bons gènes de l'autre parent. Cela permet de créer des enfants de bonne qualité que leurs parents. On distingue deux types essentiels d'opérateurs de croisement: le croisement n-point et le croisement uniforme.

• Le croisement n-point

Ce type de croisement consiste à choisir n-points de coupures ($n=1, 2, \dots$), puis échanger les fragments de gènes délimités par les points de coupure choisis. On distingue dans ce type deux autres types de croisement qui sont très utilisés: le croisement 1-point et le croisement 2-points.

- **Le croisement 1-point:** consiste à choisir un seul point de coupure, puis échanger les fragments situés après ce point de coupure. Comme le montre la figure 2.11.

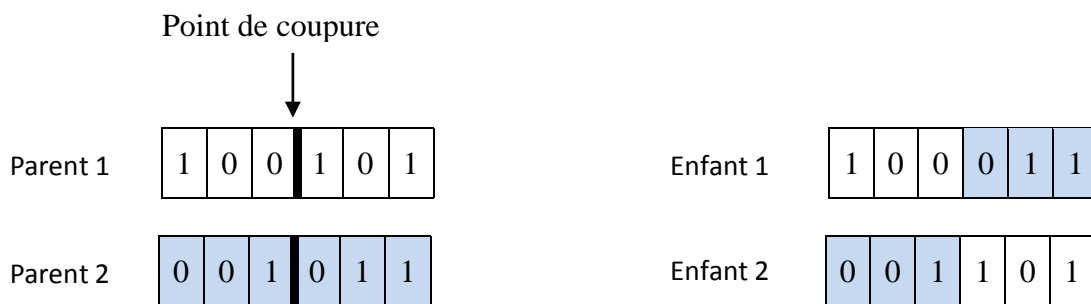


Figure 2.11. Croisement 1-point.

- **Le croisement 2-point:** consiste à choisir deux points de coupure, puis échanger les fragments situés entre ces deux points comme le montre la figure 2.12.

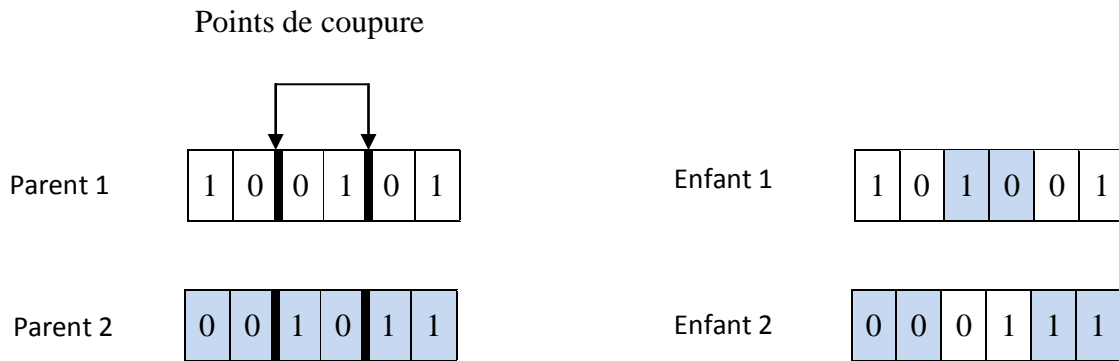


Figure 2.12. Croisement 2-points.

• Le croisement uniforme

Ce type de croisement est fondé sur la probabilité. En fait, il permet la génération d'un enfant en échangeant chaque gène des deux parents avec une probabilité égale à 0.5, comme le montre la figure 2.13.

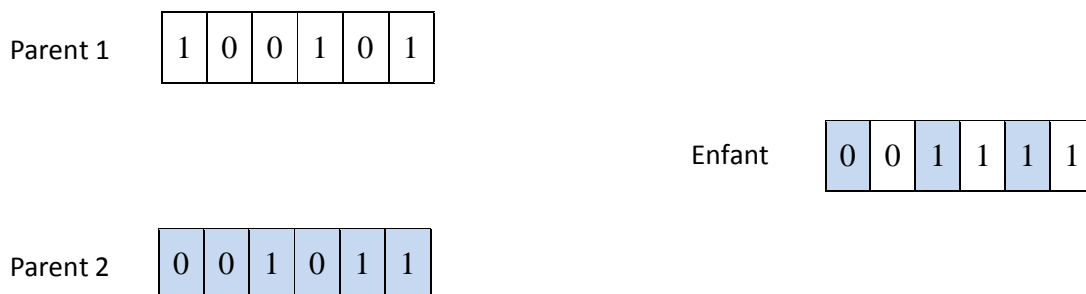


Figure 2.13. Croisement uniforme.

2.3.2.2.4. La mutation

L'opérateur de mutation permet d'apporter des modifications partielles (légères) aux chromosomes avec une certaine probabilité très faible. L'objectif de l'opérateur de mutation est d'aider l'algorithme à échapper au problème de stagnation de la recherche causé par les optima locaux. Il permet d'explorer l'espace de recherche en apportant à l'algorithme la possibilité de couvrir la totalité de l'espace de recherche [De Jong, 1975]. L'opérateur de mutation dépend du type de codage utilisé. Les opérateurs de mutation les plus utilisés sont: la mutation Bitflip et la mutation 1-bit [Naimi, 2008].

2.3.2.2.5. Le remplacement

Après croisement et mutation des individus, le nombre d'individus de la population augmente et la taille de cette dernière dépasse sa limite. En fait, la phase de reproduction (croisement et mutation) permet de créer une nouvelle population composée de deux groupes d'individus: parents et enfants. La phase de remplacement permet de décider quels sont les

individus qui vont représenter la nouvelle population. Parmi les opérateurs de remplacement existant, nous distinguons le remplacement stationnaire et le remplacement élitiste [Layeb, 2010].

2.3.2.3. La recherche par harmonies

La recherche par harmonies (HS: Harmony Search) est une très récente métaheuristique. Elle a été proposée par Geem et ses collègues [Geem et al, 2001; Geem et Choi, 2007]. A l'opposé des autres métaheurstiques qui s'inspirent des phénomènes naturels, la recherche par harmonies s'inspire du processus de recherche de la meilleure harmonie musicale. Les étapes du processus de recherche de l'algorithme de recherche par harmonies sont résumées dans l'algorithme 2.6 et expliquées dans ce qui suit.

L'algorithme HS commence par une étape d'initialisation des paramètres nécessaires et de la mémoire d'harmonies (population de solution) composée d'un ensemble de 1 à HMS harmonies (i.e. solutions) aléatoires (voir figure 2.14) et des paramètres nécessaires pour le fonctionnement du HS qui sont:

- la taille de la mémoire d'harmonies (i.e. la population), notée par HMS (de l'anglais: Harmony Memory Size).
- Le taux de considération de la mémoire harmonique, noté par HMCR (de l'anglais: Harmony Memory Considering Rate), dont le rôle est de décider si la mémoire HM sera utilisée ou non.
- Le paramètre PAR (de l'anglais: Pitch Adjusting Rate), représentant la probabilité d'apporter quelques modifications à un élément de la HM.
- Le critère d'arrêt (généralement un nombre maximum d'itérations).

$$HM = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_N^1 \\ x_1^2 & x_2^2 & \dots & x_N^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{HMS} & x_2^{HMS} & \dots & x_N^{HMS} \end{bmatrix}$$

Figure 2.14. La structure de la mémoire harmonique

Ensuite, l'algorithme passe à l'étape d'amélioration des harmonies. Cette étape consiste à améliorer une solution $x'_i = x'_1, x'_2, \dots, x'_N$ en se basant sur trois règles: la considération de la mémoire HM, l'ajustement des valeurs des variables de la solution et la sélection aléatoire. Après génération du nouveau vecteur (nouvelle solution) $x'_i = x'_1, x'_2, \dots, x'_N$, les composantes

obtenues par considération de la mémoire HM sont examinées pour décider s'ils devront être ajustées ou non. Suit à l'étape d'amélioration de solutions, le HS passe à l'étape de mise à jour de la mémoire HM. Cette étape consiste à remplacer la mauvaise solution de la matrice HM par la nouvelle solution trouvée si cette dernière est de meilleure qualité (comparée avec la mauvaise solution). Les étapes d'amélioration et de mise à jour seront répétées jusqu'à la satisfaction du critère d'arrêt.

Algorithme 2.6. L'algorithme général de la recherche par harmonies

Début

Initialiser les paramètres nécessaires;

Initialiser la mémoire HM (une population d'harmonies);

Tant que la condition d'arrêt n'est pas satisfaite **faire**

Produire une nouvelle solution en améliorant la solution x'_i ;

Mettre à jour la mémoire HM ;

Fin Tant que

Retourner la ou les meilleures solutions;

Fin

2.3.2.4. L'intelligence par essaim

Les algorithmes basés sur l'intelligence par essaim forment une branche d'algorithmes inspirés des phénomènes naturels. Ces algorithmes s'inspirent généralement des comportements collectifs de certaines espèces dans la résolution de leurs problèmes, pour développer des métaheuristiques permettant la résolution de différents problèmes d'optimisation. Le mot « essaim » est généralement utilisé pour désigner un ensemble fini de particules ou d'agents interactifs. Les oiseaux évoluant en groupes, les bancs de poissons, les colonies de fourmis, les colonies d'abeilles et même les systèmes immunitaires sont des exemples d'essaim. Les oiseaux évoluant en groupe forment des essaims dont les particules sont des oiseaux, les bancs de poissons forment des essaims dont les particules sont des poissons, les colonies de fourmis forment des essaims dont les particules sont des fourmis, les colonies d'abeilles forment des essaims dont les particules sont des abeilles, le système immunitaire forme un essaim de particules représenté par des cellules de reconnaissance et de protection. Ainsi, en imitant le comportement social des particules formant des essaims capable de s'auto-organiser, plusieurs algorithmes ont été proposés ces dernières décennies comme: L'optimisation par essaim de particules, le système immunitaire artificiel, les

colonies de fourmis artificielles, les colonies d'abeilles artificielles, la recherche coucou, l'algorithme d'optimisation par coucou...etc.

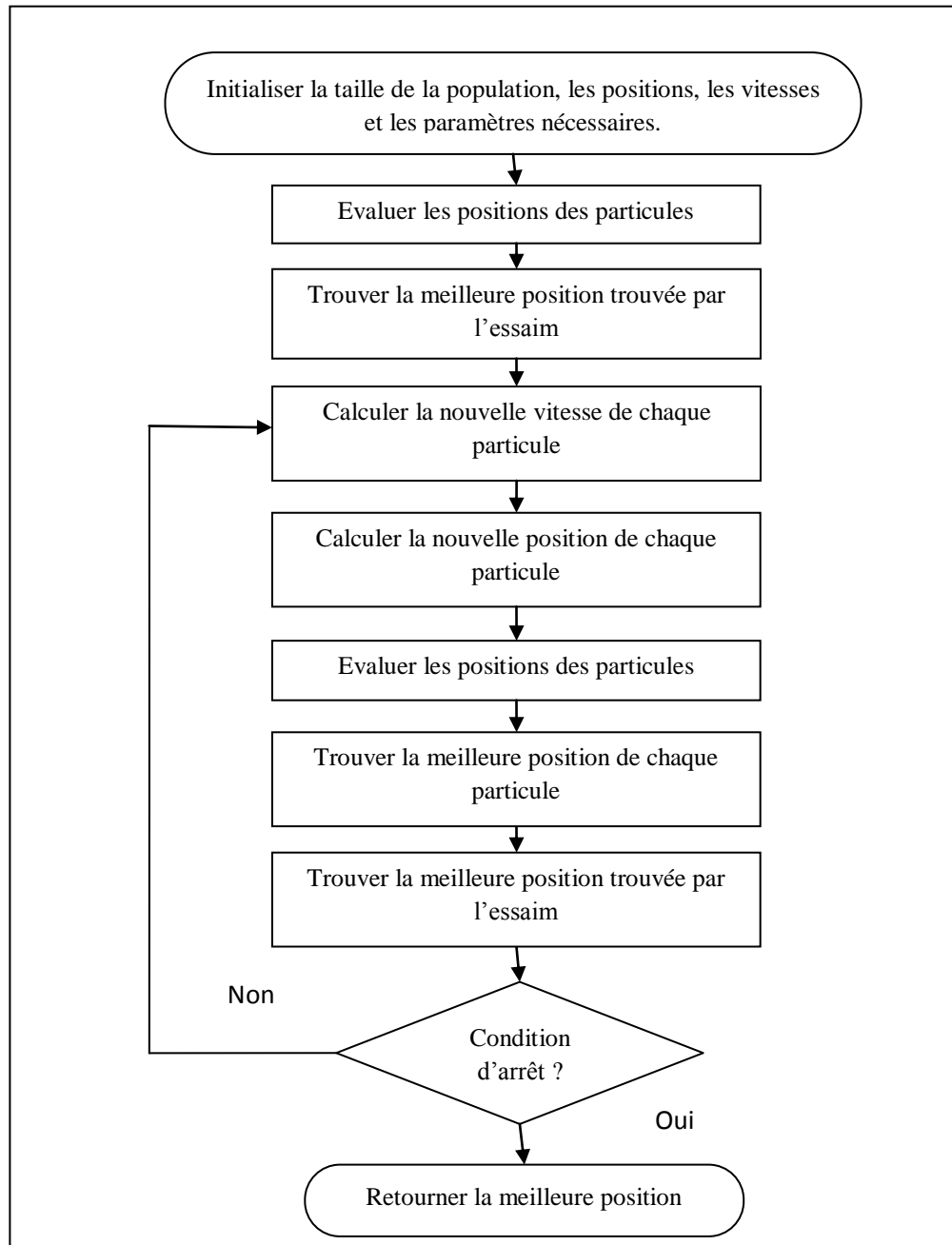


Figure 2.15. Organigramme de l'algorithme d'optimisation par essaim de particules.

2.3.2.4.1. L'optimisation par essaim de particules

L'optimisation par essaim de particules est une métaheuristique populaire basée sur l'intelligence par essaim. Elle a été proposée en 1995 par Kennedy et Eberhart. L'optimisation par essaim de particules s'inspire du comportement social des oiseaux évoluant en groupe et des bancs de poissons. L'algorithme d'optimisation par essaim de particule lance la recherche

avec une population de solutions, où chacune est appelée « particule ». Cette dernière est caractérisée par une vitesse de déplacement et une position dans l'espace de recherche. Au cours du processus de la recherche, chaque particule se déplace pour modifier sa position dans l'espace de recherche en fonction de sa vitesse actuelle, sa position actuelle, sa meilleure position trouvée au cours des itérations passées et la meilleure position trouvée par l'essaim. Son déplacement lui permet de mettre à jour sa position et sa vitesse de déplacement à chaque itération. Les étapes principales de l'algorithme d'optimisation par essaim de particules sont présentées dans l'organigramme présenté dans la figure 2.15. Plus de détail sur l'algorithme d'optimisation par essaim de particules est présenté dans le chapitre 3.

2.3.2.4.2. Le système immunitaire artificiel

Les systèmes immunitaires artificiels [Hunt et Cooke, 1996] sont des modèles des métaheuristiques inspirées des systèmes immunitaires naturels. En se basant sur ces derniers, plusieurs modèles de systèmes immunitaire artificiels ont été proposés pour faire face à de nombreux problèmes : sécurité des réseaux [Dasgupta et Gonzalez, 2002], Reconnaissance de caractères, alignement d'image, alignement multiple de séquences [Meshoul et al, 2005 ; Bendiab et al, 2003 ; Layeb et al, 2007]. Ainsi, on distingue: l'algorithme de sélection clonale [Nunes de Castro et Von Zuben, 2002], l'algorithme de sélection négative et l'algorithme du réseau immunitaire [Nunes de Castro et Timmis, 2002]. Le principe de l'algorithme de sélection clonale est présenté dans ce qui suit.

Algorithme 2.7. L'algorithme de la sélection clonale

Début

Initialiser une population de N anticorps;
Calculer leurs affinités;

Tant que la condition d'arrêt n'est pas satisfaite **faire**

Produire des clones des cellules de bonnes affinités ;

Muter les clones produits ;

Sélectionner les cellules de meilleures affinités ;

Remplacer des cellules de faibles affinités par les cellules sélectionnées;

Fin Tant que

Retourner la ou les meilleures solutions;

Fin

2.3.2.4.2.1. L'algorithme de la sélection clonale

Dans un algorithme de sélection clonale, le problème à traiter joue le rôle d'un antigène. Les solutions possibles jouent le rôle des anticorps d'une cellule B. Lorsque les anticorps d'une cellule B rencontrent un antigène, la cellule B commence à créer des clones (des copies exactes de la cellule B), ces derniers subissent des mutations afin d'améliorer leurs affinités en adaptant leurs anticorps aux antigènes envahissants. Les étapes générales de l'algorithme de la sélection clonale sont présentées dans l'algorithme 2.7.

2.3.2.4.3. L'algorithme de colonies de fourmis

L'algorithme de colonies de fourmis est un des algorithmes basés sur l'intelligence par essaim. Il a été introduit au début des années 90 par le trinôme Colorni, Dorigo et Maniezzo [Colorni et al, 1992 ; Dorigo et al, 1996]. L'idée de base du trinôme imite le comportement collectif des fourmis lors de leur déplacement entre la fourmilière et la source de nourriture. L'objectif du comportement collectif des fourmis est de collecter la nourriture sans perdre le chemin menant à leur nid. Les fourmis sont des insectes qui œuvrent pour le bien du groupe. Leurs capacités physiques et cognitives limitées n'ont jamais construit un obstacle pour elles. En effet, elles peuvent défier leurs capacités individuelles limitées et réaliser des tâches très complexes (construire des nids, rechercher la nourriture, élever les larves...) par coopération en regroupant leurs capacités disponibles et leurs expériences collectives.

Dans l'objectif de rechercher la nourriture en parcourant le plus court chemin, les fourmis se communiquent indirectement entre elles en provoquant des changements dans leur environnement. Au début de la recherche, les fourmis se propagent aléatoirement en prenant des chemins de différentes tailles (court, long,...) dont elles déposent sur le sol une matière odorante appelée « phéromone » d'intensités égales. Afin d'attirer l'attention de leurs congénères en retournant au nid, les fourmis déposent des phéromones un peu différents contenant un message concernant la qualité du site visité. Les fourmis ont tendance de suivre le chemin de plus forte intensité de phéromones. Plus le chemin est court, plus la quantité de phéromones y est déposée est élevée. Et plus l'intensité de phéromones est grande, plus le nombre de fourmis utilisant ce passage augmente. Par conséquent, le chemin le plus long sera abandonné car l'intensité de phéromones y compris est petite et s'évapore rapidement.

L'auto-organisation des fourmis basée sur l'utilisation des marqueurs chimiques (i.e. les phéromones) a construit le secret du succès de ces insectes dans la résolution de leurs problèmes. La fourmi informatique représente une solution au problème traité. Les

phéromones informatiques sont des valeurs associées à des solutions trouvées. Ces valeurs dépendent des qualités des solutions. En fait, chaque fourmi (i.e. solution) dépose une certaine quantité de phéromones qui dépend de sa qualité.

L'algorithme de colonies de fourmis a été proposé pour la première fois pour résoudre le problème du voyageur de commerce [Colormi et al, 1992], il se base sur trois phases essentielles:

Algorithme 2.8. L'algorithme de colonies de fourmis pour le TSP

Début

Initialiser une population de m fourmis ;

Evaluer les m fourmis ;

Tant que la condition d'arrêt n'est pas satisfaite **faire**

Pour i=1 à m **faire**

Construire le trajet de la fourmi i;

Déposer des phéromones sur le trajet de la fourmi i;

Fin pour

Evaluer les m fourmis;

Evaporer les pistes de phéromones;

Fin Tant que

Retourner la ou les meilleures solutions ;

Fin

- La construction du trajet de chaque fourmi.
- La distribution de phéromones sur le trajet de chaque fourmi.
- Evaporation des pistes de phéromones.

L'algorithme 2.8 représente le schéma général de l'algorithme de colonies de fourmis pour le problème du voyageur de commerce (TSP: Traveling Salesman Problem). Il est à noter qu'à part le problème du voyageur de commerce, l'algorithme de colonies de fourmis a été appliqué avec succès sur d'autres problèmes d'optimisation comme: les problèmes des tournées de véhicules, le problème d'affectation quadratique...etc.

2.3.2.4.4. L'optimisation par colonies d'abeilles

L'optimisation par colonies d'abeilles artificielles (ABC: Artificial Bee Colony) est une nouvelle métaheuristique qui a enrichi le nombre des méthodes d'optimisation basées sur l'intelligence par essaim. Elle a été proposée en 2005 par Karaboga [Karaboga, 2005].

Algorithme 2.9. L'algorithme de colonies d'abeilles

Début

Initialiser une population de N solution ;

Evaluer les N solutions ;

Cycle=1 ;

Tant que cycle<=MCN **faire**

Construire une nouvelle solution v_i pour chaque ouvrière i en utilisant l'équation 2.7, puis les évaluer ;

Sélectionner les ouvrières ;

Calculer les valeurs de probabilités P_i pour les solutions x_i en utilisant l'équation 2.8;

Construire la nouvelle solution de chaque abeille spectatrice à partir de la solution x_i sélectionnée en fonction de la probabilité p_i ;

Evaluer les nouvelles solutions ;

Sélectionner les spectatrices ;

Déterminer les solutions à abandonner par les scouts si elles existent et les remplacer par des solutions aléatoires;

Enregistrer la meilleure solution trouvée.

Cycle= Cycle+1 ;

Fin Tant que

Retourner la meilleure solution ;

Fin

L'algorithme ABC s'inspire du modèle naturel du comportement des abeilles mellifères lors de la recherche de leur nourriture. Le processus de recherche de nourriture chez les abeilles est fondé sur un mécanisme de déplacement très efficace. Il leur permet d'attirer l'attention d'autres abeilles de la colonie aux sources alimentaires trouvées dans le but de collecter des ressources diverses. En fait, les abeilles utilisent un ensemble de danses frétillantes comme moyen de communication entre elles. Ces danses permettent aux abeilles de partager des informations sur la direction, la distance et la quantité du nectar avec ses congénères. La collaboration et la connaissance collective des abeilles de la même colonie sont basées sur l'échange d'information sur la quantité du nectar dans la source de nourriture trouvée par les différents membres. Des études sur le comportement de danses frétillantes des abeilles ont montré [Chan et Tiwari, 2007]:

- La direction des abeilles indique la direction de la source de nourriture par rapport au soleil.

- L'intensité de la danse indique la distance de la source de nourriture.
- La durée de la danse indique la quantité du nectar dans la source de nourriture trouvée.

Dans un algorithme d'optimisation par colonies d'abeilles, une source de nectar correspond à une solution possible au problème à traiter. La colonie d'abeilles artificielle est composée de trois types d'abeilles: les ouvrières, les spectatrices et les scoutes.

- ***L'ouvrière*** exploite la source de nourriture trouvée. Elle se base sur sa mémoire et essaye d'apporter des modifications à sa position (solution) actuelle pour découvrir une nouvelle position (i.e. source de nourriture).
- ***L'abeille spectatrice*** attend le retour des ouvrières au champ de danse pour observer leurs danses et recueillir des informations sur les sources de nectar qu'elles ont trouvées.
- ***L'abeille scoute*** exploite l'espace de recherche en lançant une recherche aléatoire d'une nouvelle source de nourriture.

Une abeille ouvrière est assignée à chaque source de nourriture. La taille de la population de la colonie est égale au nombre des abeilles ouvrières et la quantité du nectar dans une source de nourriture correspond à la qualité (fitness) de la solution proposée. L'algorithme 2.9 représente un pseudo code de l'algorithme de colonies d'abeilles.

2.3.2.4.5. La recherche coucou

La recherche coucou (CS: Cuckoo Search) est une très récente métaheuristique. Elle a été proposée en 2009 par Yang et Deb [Yang et Deb, 2009; Yang et Deb, 2010]. La recherche coucou s'inspire du comportement de reproduction d'une espèce spéciale d'oiseaux parasites de nids appelés « Coucous ». Dans l'algorithme de la recherche coucou, une solution possible est appelée « nid » ou « coucou ». En fait, la recherche coucou part du principe que chaque nid comporte un seul coucou. Au cours du processus de la recherche de l'algorithme CS, chaque coucou crée son propre poussin en fonction de sa représentation actuelle (le coucou lui-même) et du vol de Lévy. L'évaluation de la qualité du poussin et de son père permet de sélectionner lequel d'entre eux survivra et subira quelques modifications dans le but d'améliorer sa qualité. La figure 2.20 représente un organigramme résumant les étapes de l'algorithme de la recherche coucou et le chapitre 5 représente plus de détail sur cette métaheuristique.

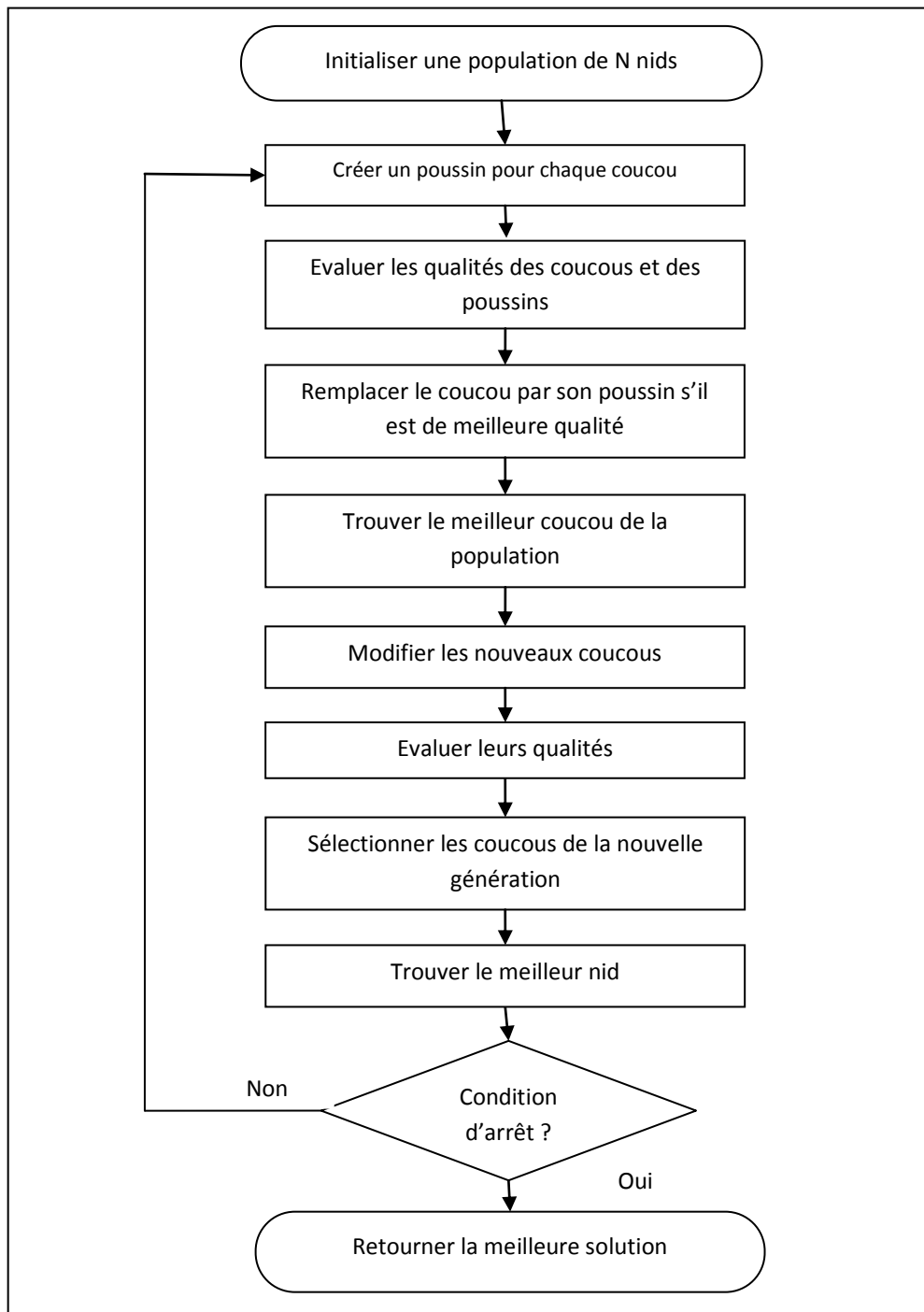


Figure 2.16. Organigramme de l'algorithme de la recherche coucou

2.3.2.4.6. L'algorithme de l'optimisation par coucou

L'algorithme d'optimisation par coucou (COA: Cuckoo Optimisation Algorithm) est aussi une très jeune métaheuristique. Elle a été proposée en 2011 par Rajabioun [Rajabioun, 2011]. L'algorithme d'optimisation par coucou s'inspire du comportement des coucous dans leur vie, reproduction et développement. Dans l'algorithme COA, une solution donnée est appelée « habitat ». Un habitat représente une position actuelle du coucou dans l'espace de recherche.

L'habitat de chaque coucou produit un certain nombre d'œufs, certains entre eux se développent et deviennent des coucous matures. La création des œufs est basée sur la position actuelle de l'habitat du coucou et sur la distance de ponte proportionnelle au nombre total des œufs du coucou lui-même et aussi à deux autres variables limitant l'intervalle d'où les valeurs des variables sont tirées (notées Var_{hi} et Var_{max}).

A chaque itération, chaque coucou pond un ensemble d'œufs, certains d'entre eux survivent et migrent dans le but de rechercher le meilleur habitat. Les autres seront tués par leurs hôtes (l'oiseau propriétaire du nid), ou détruits à cause de leurs mauvaises qualités. La figure 2.17 représente un organigramme résumant les étapes de l'algorithme COA et le chapitre 5 présente plus de détail sur cette métaheuristique.

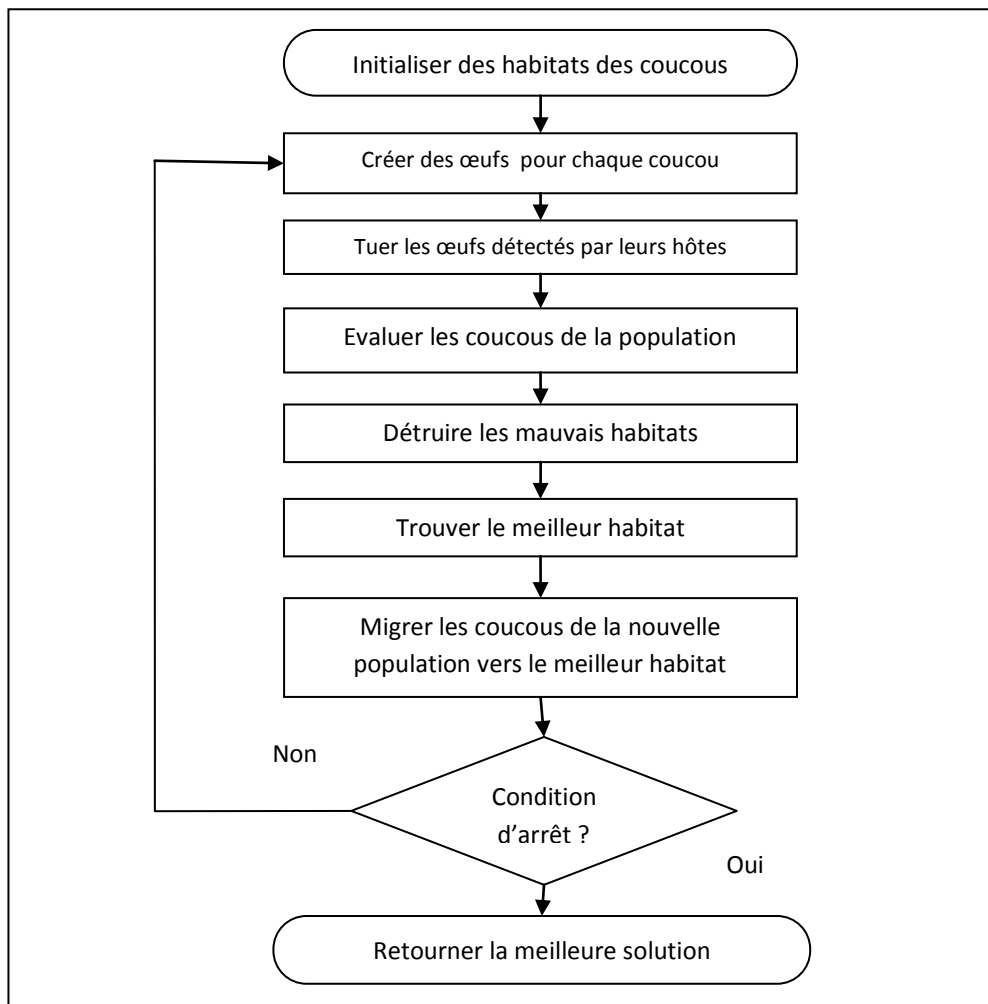


Figure 2.17. Organigramme de l'algorithme de l'optimisation par la recherche coucou.

2.4. Conclusion

Dans ce chapitre nous avons essayé de présenter un état de l'art sur les méthodes de résolution de différents problèmes d'optimisation présentées dans la littérature commençant

par les méthodes exactes aux méthodes approchées. Nous avons constaté que les méthodes exactes permettent d'aboutir à la solution optimale, mais elles sont trop gourmandes en termes de temps de calcul et d'espace mémoire requis. Cependant, les méthodes approchées demandent des coûts de recherche raisonnables. Mais, elles ne garantissent pas l'optimalité de la solution. Nous avons pu constater que les méthodes approchées peuvent être partagées en deux classes: des méthodes heuristiques et des méthodes métaheuristiques. Une méthode heuristique est applicable sur un problème donné. Tandis qu'une méthode métaheuristique est plus générique et elle peut être appliquée sur une panoplie de problèmes d'optimisation. En outre, nous avons constaté que les méthodes métaheuristiques peuvent être partagées en deux sous classes: des méthodes à base d'une solution unique et des méthodes à base de population de solutions. Les méthodes de la première sous classe (i.e. les méthodes à base d'une solution unique) se basent sur la recherche locale pour trouver la solution du problème à traiter. Elles sont souvent piégées par l'optimum local d'un voisinage donné. Par contre, les méthodes de la deuxième classe (i.e. les méthodes à base de population de solutions) se basent sur une recherche globale ce qui leur permet d'échapper au problème de la convergence vers l'optimum global et augmente leur possibilité de fournir des solutions de bonnes qualités. Nous avons essayé de présenter le principe de plusieurs méthodes métaheuristiques y compris les méthodes basées sur l'intelligence par essaim qui ont construit une tendance très active ces dernières décennies.

Dans le chapitre suivant, nous présentons notre première contribution dans laquelle nous avons proposé quatre classes d'algorithmes d'optimisation par essaim de particules, où chacune est composée de quatre algorithmes inspirés des variantes de l'algorithme d'optimisation par essaim de particules proposées dans la littérature.

CHAPITRE 3

Ensemble d'algorithmes d'optimisation par essaim de particules

Sommaire

3.1	Introduction.....	79
3.2	L'origine de l'idée de l'optimisation par essaim de particules	80
3.3	L'optimisation par essaim de particules (PSO).....	81
3.4	Les variantes de l'algorithme PSO	85
3.5	L'algorithme discret/binaire de l'optimisation par essaim de particules (BPSO)...	87
3.6	L'algorithme quantique de l'optimisation par essaim de particules (QPSO).....	89
3.7	Le PSO avec paramètres adaptatifs	90
3.8	Les applications de l'algorithme PSO	94
3.9	Les classes des algorithmes proposés	98
3.10	Comparaison et résultats expérimentaux	104
3.11	Conclusion.....	113

3.1. Introduction

La résolution de problèmes d'optimisation constitue un axe de recherche qui a sollicité l'attention de plusieurs équipes de recherche, vu l'importance capitale qu'elle revêt dans notre vie quotidienne. Cette branche de recherche fait appel à des astuces mathématiques et informatiques. Elle est intrinsèquement liée à la recherche opérationnelle, l'algorithmique et la théorie de la complexité. La résolution d'un problème d'optimisation consiste à rechercher solution d'une qualité suffisante parmi un ensemble de solutions au regard d'un (des) critère(s) donné (s) et des objectifs à satisfaire. Elle consiste à maximiser ou à minimiser une ou un ensemble de fonctions fitness en respectant des contraintes liées au problème traité. La résolution de n'importe quel problème d'optimisation s'effectue par un procédé algorithmique de complexité spatiale et temporelle qui selon leur degré, elles permettent de classer les problèmes d'optimisation en différentes classes (P, NP, NP-Complet et NP-Difficile).

Les méthodes de résolution de problèmes d'optimisation sont nombreuses. Elles sont souvent classées en deux grandes classes: La classe des méthodes exactes et la classe des méthodes approchées. Les méthodes exactes garantissent l'optimalité de la solution, mais elles sont souvent exigeantes en termes de temps de calcul nécessaire. Notamment, pour la résolution des problèmes de grande taille. Les méthodes approchées constituent une alternative des méthodes exactes. Elles ne garantissent pas l'optimalité de la solution mais elles ne sont pas trop exigeantes en termes de temps de calcul nécessaire. En fait, elles permettent la résolution des problèmes d'optimisation de différentes tailles en un temps de calcul raisonnable. Le coût prohibitif engendré par l'utilisation des méthodes exactes, a excité les chercheurs à faire recours à des méthodes approchées. L'investigation dans le domaine des méthodes approchées a donné naissance à une autre catégorie de méthodes appelées « Métaheuristiques ». Les métaheuristiques sont des méthodes générales et applicables sur une grande gamme de problèmes d'optimisation. Elles sont souvent inspirées des systèmes naturels dans différents domaines: physique, biologie, éthologie...etc.

L'idée de s'inspirer des systèmes naturels pour proposer des méthodes de résolution des problèmes d'optimisation a donné naissance à une sous classe des métaheuristiques, ce sont des métaheuristiques basées sur l'intelligence par essaim (en anglais: Swarm Intelligence). La métaheuristique d'optimisation par essaim de particules (PSO: Particle Swarm Optimization) est la métaheuristique principale dans la classe des métaheuristiques basées sur l'intelligence par essaim. C'est une des métaheuristiques à base de population de solutions inspirées par une analogie avec l'éthologie. L'optimisation par essaim de particules est une jeune

métaheuristique, elle imite le comportement social des animaux évoluant en groupe comme les oiseaux, les abeilles et les poissons. Le PSO a été proposé en 1995 par Kennedy et Eberhart pour la résolution des problèmes d'optimisation continus. La simplicité et la performance de cette méthode ont suscité l'intérêt de plusieurs communautés de chercheurs qui ont mené des études d'optimisation et d'application de cette métaheuristique pour la résolution de plusieurs problèmes d'optimisation continus et/ou discrets. Par conséquent, plusieurs alternatives de l'algorithme originel du PSO ont été proposées dans la littérature afin d'améliorer sa performance pour la résolution de différents problèmes. De notre part, Nous avons proposé dans cette contribution, quatre classes d'algorithmes binaires d'optimisation par essaim de particules. Dans chaque classe, nous avons proposé quatre algorithmes différents. Contrairement à la version originelle de l'algorithme PSO, l'avantage majeur des algorithmes proposés c'est qu'ils peuvent être utilisés pour la résolution des problèmes continus, discrets et discrets/binaires par une simple interprétation des opérateurs utilisés.

3. 2. L'origine de l'idée de l'optimisation par essaim de particules

L'idée de l'optimisation par essaim de particules trouve ses racines dans les années 80. Précisément en 1983, lorsque Reeves [Reeves, 1983] a essayé de résoudre le problème de rendu des images afin de simuler les phénomènes naturels en utilisant l'outil Informatique pour créer des scènes animées. Dans le cadre de son travail, Reeves a implémenté un système de particules qui œuvrent ensemble pour simuler un objet flou (nuage, explosion...). Le modèle proposé par Reeves considère que chaque particule est caractérisée par une position dans l'espace de recherche et une vitesse de déplacement. En effet, les particules de l'essaim se déplacent en fonction de leurs positions courantes et leurs vitesses qui seront adaptées au cours de la recherche.

D'autre part, Craig Reynolds [Reynolds, 1987] a été intrigué par l'organisation et l'esthétique du comportement social des oiseaux. Il a tenté d'améliorer l'idée de Reeves, en rendant le comportement du groupe des particules plus dynamique et plus organisé. Reynolds a ajouté la notion d'orientation et la notion de communication inter-particules: chaque particule doit rester proche des autres particules de l'essaim comme elle (i.e. la particule) doit éviter d'entrer en collision avec ses congénères. C'est la raison pour laquelle, chaque particule doit avoir conscience de la position des autres particules du groupe. Suite à sa recherche, Reynolds a découvert que l'implémentation d'un modèle simulant le comportement de particules tel qu'il est en réalité n'est pas faisable. En fait, il a découvert que son modèle

engendre une exécution très complexe surtout avec une population de grande taille. Afin de pallier à ce problème, Reynolds a proposé l'utilisation de la notion du voisinage.

De leurs parts, Heppner et Grenander [Heppner et Grenander, 1990] ont apprécié la manière de volées d'oiseaux. Ils ont établi une recherche sur les différentes règles permettant à un ensemble d'agents de se communiquer d'une manière très simple et de produire un comportement social imprévisible, bien organisé et intelligent.

En 1995, Kennedy et Eberhart se sont basés sur les idées et les études de Reeves, Reynolds et surtout des résultats de Heppner et Grenander afin de comprendre la stratégie de recherche de nourriture et d'affrontement des prédateurs que l'on retrouve chez les groupes d'animaux tels que les bancs de poissons, les volées d'oiseaux ou les essaims d'insectes. Le fruit des recherches de Kennedy et Eberhart était la proposition de la métaheuristique d'optimisation par essaim de particules dont le principe sera décrit dans ce qui suit.

3. 3. L'optimisation par essaim de particules (PSO)

L'optimisation par essaim de particules (le PSO en anglais: Particle Swarm Optimazation) est une métaheuristique à base de population de solution. Elle a été proposée en 1995 par Kennedy et Eberhart [Kennedy et Eberhart, 1995]. L'algorithme PSO est inspiré du comportement social d'animaux évoluant en essaim, tels que les poissons qui se déplacent en bancs ou les oiseaux migrateurs. En effet, on peut observer chez ces animaux des dynamiques de déplacement relativement complexes, alors qu'individuellement chaque individu a une intelligence limitée et une connaissance seulement locale de sa situation dans l'essaim. L'intelligence globale de l'essaim est donc la conséquence directe des interactions locales entre les différentes particules de l'essaim. La performance du système entier est supérieure de la somme des performances de ses parties. Kennedy et Eberhart se sont inspirés de ces comportements sociaux pour créer l'algorithme PSO. Contrairement aux autres algorithmes évolutionnaires tel que l'algorithme génétique où la recherche de la solution optimale évolue par compétition entre les individus en utilisant des opérateurs de croisements et de mutations, le PSO utilise plutôt la coopération entre les individus.

La méthode d'optimisation par essaim particulaire met en jeu un ensemble d'agents pour la résolution d'un problème donné. Cet ensemble est appelé essaim. L'essaim est composé d'un ensemble de membres, ces derniers sont appelés particules. Les particules de l'essaim représentent des solutions potentielles au problème traité. L'essaim de particules survole

l'espace de recherche, en quête de l'optimum global. Le déplacement de chaque particule est influencé par les trois composantes suivantes [Cooren, 2008] (Figure 3.1):

- Une composante physique: la particule tend à suivre sa direction de déplacement courante;
- Une composante cognitive: la particule tend à se diriger vers le meilleur site par lequel elle est déjà passée;
- Une composante sociale: la particule tend à se diriger vers le meilleur site déjà atteint par ses voisins.

Chaque particule i de l'essaim est définie par sa position $x_{id} = (x_{i1}, x_{i2}, \dots, x_{id}, \dots, x_{iD})$ et sa vitesse de déplacement $v_{id} = (v_{i1}, v_{i2}, \dots, v_{id}, \dots, v_{iD})$ dans un espace de recherche de dimension D . Cette particule garde en mémoire la meilleure position par laquelle elle est déjà passée et la meilleure position atteinte par toutes les particules de l'essaim, notées respectivement: $p_{bestid} = (p_{besti1}, p_{besti2}, \dots, p_{bestid}, \dots, p_{bestiD})$ et $g_{best} = (g_{best1}, g_{best2}, \dots, g_{bestd}, \dots, g_{bestD})$.

Le processus de recherche est basé sur deux règles :

- Chaque particule est dotée d'une mémoire qui lui permet de mémoriser la meilleure position par laquelle elle est déjà passée et elle a tendance à retourner vers cette position.
- Chaque particule est informée de la meilleure position connue au sein de son voisinage et elle a toujours tendance de se déplacer vers cette position.

La particule i va se déplacer entre les itérations t et $t+1$, en fonction de sa vitesse et des deux meilleures positions qu'elle connaît (la sienne et celle de l'essaim) suivant les deux équations suivantes [Kennedy et Eberhart, 1995]:

$$v_{id}(t) = v_{id}(t-1) + c_1 r_1 (p_{bestid}(t-1) - x_{id}(t-1)) + c_2 r_2 (g_{bestd}(t-1) - x_{id}(t-1)) \quad (3.1)$$

$$x_{id}(t) = x_{id}(t-1) + v_{id}(t) \quad (3.2)$$

Avec :

- $x_{id}(t), x_{id}(t-1)$: la position de la particule i dans la dimension d aux temps t et $t-1$, respectivement.
- $v_{id}(t), v_{id}(t-1)$: la vitesse de la particule i dans la dimension d aux temps t et $t-1$, respectivement.
- $p_{bestid}(t-1)$: la meilleure position obtenue par la particule i dans la dimension d au temps $t-1$.
- $g_{bestd}(t-1)$: la meilleure position obtenue par l'essaim dans la dimension d au temps $t-1$.

- c_1, c_2 : deux constantes qui représentent les coefficients d'accélération, elles peuvent être non constantes dans certains cas [Ratnaweera et al, 2004] et [Kuo et al, 2007].
- r_1, r_2 : nombres aléatoires tirés de l'intervalle $[0,1]$.
- $v_{id}(t-1), c_1 r_1 (p_{bestid}(t-1) - x_{id}(t-1)), c_2 r_2 (g_{bestid}(t-1) - x_{id}(t-1))$: représentent respectivement, les trois composantes citées au-dessus.

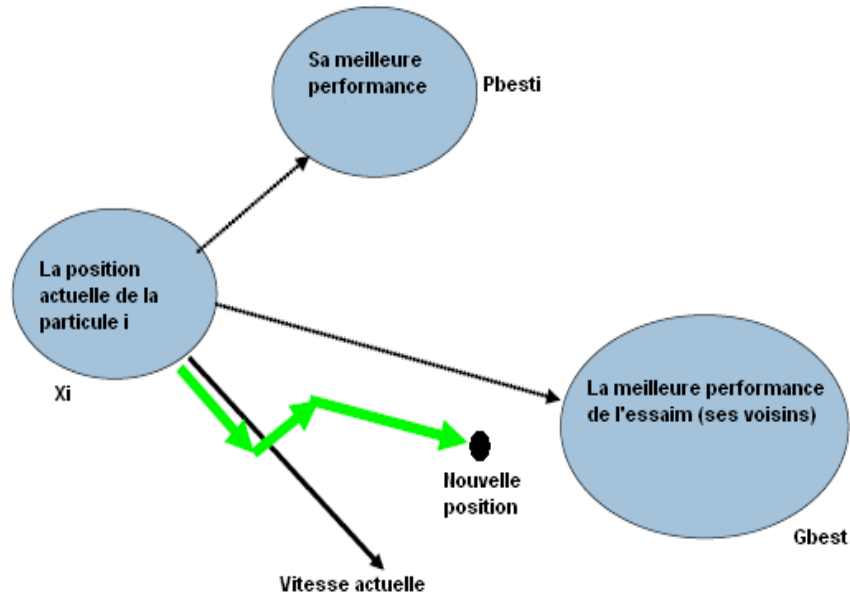


Figure 3.1. Déplacement d'une particule

Afin d'estimer la qualité de la particule i , il est indispensable de calculer sa fonction «objectif» (aussi appelée fitness). La valeur de la fonction «objectif» de la particule x_{id} est notée $f(x_{id})$. Cette dernière est calculée en utilisant une fonction spéciale au problème traité. Afin de mettre à jour les valeurs de x_{id} , p_{bestid} et g_{bestid} , leurs fitness sont calculées à chaque itération de l'algorithme. x_{id} est mise à jour selon l'équation (3.2). p_{bestid} et g_{bestid} sont mises à jour si les conditions (C1) et (C2) présentées ci-dessous sont vérifiées respectivement :

$$f(x_{id}) \text{ est meilleur que } f(p_{bestid}) \quad (C1)$$

$$f(p_{bestid}) \text{ est meilleur que } f(g_{bestid}) \quad (C2)$$

L'algorithme PSO commence par initialiser la taille de l'essaim ainsi que les différents paramètres, affecter à chaque particule une position et une vitesse initiales et initialiser les p_{bestid} . Ensuite, calculer les fitness des particules afin de pouvoir calculer la meilleure position trouvée par l'essaim g_{bestid} . À chaque itération du processus de la recherche, les particules se déplacent en fonction des équations (3.1) et (3.2). Leurs fitness sont calculées, les p_{bestid} et la

g_{bestd} sont mises à jour. Le processus est répété jusqu'à la satisfaction du critère d'arrêt. L'algorithme 3.1 représente un pseudo code de l'algorithme PSO.

Algorithme 3.1. L'optimisation par essaim de particules

Début

Initialiser les paramètres et la taille S de l'essaim;
Initialiser les vitesses et les positions aléatoires des particules dans chaque dimension de l'espace de recherche;
Pour chaque particule, $p_{bestid} = x_{id}$;
Calculer $f(x_{id})$ de chaque particule;
Calculer g_{bestd} ; // la meilleure p_{bestid}

Tant que (la condition d'arrêt n'est pas vérifiée) **faire**

Pour (i allant de 1 à S) **faire**

Calculer la nouvelle vitesse à l'aide de l'équation (3.1) ;
Trouver la nouvelle position à l'aide de l'équation (3.2) ;
Calculer $f(x_{id})$ de chaque particule;

Si ($f(x_{id})$ est meilleur que $f(p_{bestid})$) **alors**

$p_{bestid} = x_{id}$;

Si ($f(p_{bestid})$ est meilleur que $f(g_{bestd})$) **alors**

$g_{bestd} = p_{bestid}$;

Fin pour

Fin tant que

Afficher la meilleure solution trouvée g_{bestd} ;

Fin

Dans le PSO, chaque particule est influencée à la fois par ses propres informations acquises par les expériences passées et par celles de l'essaim. Malgré les différences remarquables entre le principe directeur de l'algorithme PSO et celui des algorithmes évolutionnaires en général, l'optimisation par essaim de particules fait tout de même partie de cette classe. Wilson [Wilson, 1975], démontra que le partage d'informations entre les individus d'un même groupe était, dans certains cas beaucoup plus avantageux pour la survie de l'espèce que la compétition entre les individus. La coopération est ainsi un mécanisme évolutif au même titre que la sélection, le croisement et la mutation. De plus, les mécanismes de sélection, de croisement et de mutation sont bien présents mais sous une forme moins implicite [Mathieu, 2008].

3. 4. Les variantes de l'algorithme PSO

L'idée des pionniers de l'optimisation par essaim de particules: Kennedy et Eberhart a sollicité l'attention de plusieurs chercheurs qui ont mené des études dans l'objectif d'améliorer la performance de la méthode proposée. En fait, malgré l'efficacité et la simplicité de la mise en œuvre de l'algorithme d'optimisation par essaim de particules, ce dernier souffre du problème de la convergence prématurée [Van den Bergh, 2001].

En 1996 Eberhart et ses collègues ont proposé de limiter la vitesse de la particule par l'intervalle $[-v_{max}, v_{max}]$ [Eberhart et al, 1996]. Leur objectif était d'échapper au problème de déviation des particules de l'espace de recherche lors de leur déplacement. Le nouveau paramètre v_{max} permet de mieux contrôler le mouvement de particules. D'autres études sur l'introduction de v_{max} sont disponibles dans: [Eberhart et Shi, 2000], [Fan et al, 2001], [Cai et Tan, 2009], [Deep et Bansal, 2009], [Barrera et Coello, 2009].

En 1998, Shi et Eberhart ont proposé dans [Shi et Eberhart, 1998] une variante de l'équation (3.1). La modification apportée consiste à appliquer un facteur d'inertie pour contrôler la vitesse des particules de la manière suivante:

$$v_{id}(t) = \omega v_{id}(t-1) + c_1 r_1 (p_{bestid}(t-1) - x_{id}(t-1)) + c_2 r_2 (g_{besti}(t-1) - x_{id}(t-1)) \quad (3.3)$$

Où ω est un coefficient d'inertie, généralement une constante qui sert à contrôler l'influence de la vitesse de la particule sur son prochain déplacement afin de garder un équilibre entre l'exploitation et l'exploration de l'espace de recherche. Elle peut être variable dans certains cas: Eberhart et Shi [Eberhart et Shi, 1998; Eberhart et Shi, 1999] voient que la valeur raisonnable de ω doit diminuer linéairement au cours du processus de l'optimisation. De même Kusum Deep et Jagdish Chand Bansal [Deep et Bansal, 2009] proposent de réduire la valeur de ω linéairement de 0.8 à 0.4. D'autres études sont disponibles dans: [Chatterjee et Siarry, 2006], [Fan et Chang, 2007], [Zhang et Qiu, 2010].

De sa part, Clerc a proposé [Clerc, 1999], [Clerc et Kennedy, 2002] une autre variante de l'équation (3.1). Sa variante consiste à ajouter un facteur de constriction K dont l'objectif est de contrôler la vitesse des particules afin d'échapper au problème de la divergence de l'essaim qui cause la convergence prématurée de l'algorithme. L'équation permettant la mise à jour de la vitesse est la suivante:

$$v_{id}(t) = K [v_{id}(t-1) + c_1 r_1 (p_{bestid}(t-1) - x_{id}(t-1)) + c_2 r_2 (g_{besti}(t-1) - x_{id}(t-1))] \quad (3.4)$$

$$\text{Où } K = \frac{2}{|2 - \varphi - \sqrt{\varphi(\varphi - 4)}|}$$

Avec $\varphi = c_1 + c_2$ et $\varphi > 4$; $c_1 = c_2 = 2.05$; ce qui donne : $K=0.729844$.

Partant de l'équation (3.1), une autre variante a été proposée en 2004 par Parsopoulos et Vrahatis [Parsopoulos et Vrahatis, 2004]. Ils ont proposé de calculer la vitesse de la particule à partir d'une combinaison de la vitesse locale et la vitesse globale qui sont définies de la manière suivante:

$$G(t) = K [v_{id}(t-1) + c_1 r_1 (p_{bestid}(t-1) - x_{id}(t-1)) + c_2 r_2 (g_{bestid}(t-1) - x_{id}(t-1))] \quad (3.5)$$

$$L(t) = K [v_{id}(t-1) + c_1 r'_1 (p_{bestid}(t-1) - x_{id}(t-1)) + c_2 r'_2 (g_{bestid}(t-1) - x_{id}(t-1))] \quad (3.6)$$

Où $G(t)$ c'est la vitesse globale et $L(t)$ c'est la vitesse locale.

g_{bestid} c'est la meilleure position trouvée dans un voisinage.

La mise à jour de l'essaim est établie selon les équations suivantes:

$$U(t) = (1-u) L(t) + u G(t) \quad (3.7)$$

$$x(t) = x(t-1) + U(t) \quad (3.8)$$

u est appelé facteur d'unification, il est tiré de l'intervalle $[0,1]$.

L'équation (3.7) peut être écrite comme suit [Parsopoulos et Vrahatis, 2004]:

$$U(t) = (1-u) L(t) + r_3 u G(t) \quad (3.9)$$

$$\text{Ou } U(t) = r_3 (1-u) L(t) + u G(t) \quad (3.10)$$

Où : r_3 est un paramètre aléatoire.

Dans un but d'assurer la diversité de l'essaim, Hi et al [Hi et al, 2004] ont proposé de leur part de mettre à jour la vitesse des particules selon l'équation (3.11):

$$v_{id}(t) = \omega v_{id}(t-1) + c_1 r_1 (p_{bestid}(t-1) - x_{id}(t-1)) + c_2 r_2 (g_{bestid}(t-1) - x_{id}(t-1)) + c_3 r_3 (P_{id}^r(t-1) - x_{id}(t-1)) \quad (3.11)$$

Où P_{id}^r est la position de la particule i dans la dimension d de l'espace de recherche. Cette particule est sélectionnée aléatoirement à chaque itération. Le rôle de la composante $(P_{id}^r(t-1) - x_{id}(t-1))$ est d'assurer la diversité de l'essaim selon la valeur du coefficient d'accélération c_3 .

Une autre variante a été proposée par Pongchairerks et Kachitvichyanukul nommée GLNPSO [Pongchairerks et Kachitvichyanukul, 2005]. Afin de mettre à jour la vitesse de

chaque particule, Pongchairerks et Kachitvichyanukul se sont basés sur l'équation (3.3) ainsi que sur la meilleure position locale, globale et celle trouvée dans le plus proche voisinage. L'avantage de l'algorithme proposé (GLNPSO) c'est qu'il permet d'explorer différentes zones de l'espace de recherche simultanément. La position de la particule est mise à jour selon l'équation (3.2) alors que la vitesse de la particule est mise à jour selon l'équation suivantes:

$$v_{id}(t) = \omega(t) v_{id}(t-1) + c_p r_1 (p_{bestid} - x_{id}(t-1)) + c_g r_1 (g_{bestd} - x_{id}(t-1)) + c_l r_1 (L_{bestd} - x_{id}(t-1)) + c_n r_1 (n_{bestd} - x_{id}(t-1)) \quad (3.12)$$

Avec : c_p , c_g , c_l et c_n sont des coefficients d'accélération.

L_{bestd} , n_{bestd} sont respectivement, la meilleure position dans un voisinage donné et la meilleure position dans le plus proche voisinage.

Avec $v_{id}(t) \in [-v_{max}, v_{max}]$ et $x_{id}(t) \in [-x_{max}, x_{max}]$.

3. 5. L'algorithme discret/binaire de l'optimisation par essaim de particules (BPSO)

L'optimisation par essaim particulaire est une jeune métaheuristique. Grâce à son efficacité, ses concepts simples et sa convergence rapide, elle a été utilisée pour la résolution de plusieurs problèmes dans différents domaines. Au début, la métaheuristique d'optimisation par essaim de particules a été proposée pour résoudre des problèmes d'optimisation continus. Toutefois, maints problèmes d'optimisation sont discrets ou discrets/binaires. Par conséquent, une tendance active a été remarquée ces dernières décennies, c'est l'adaptation de l'algorithme PSO pour résoudre des problèmes d'optimisation discrets et discrets/binaires.

La première version de l'algorithme BPSO (l'algorithme standard discret/binaire de l'optimisation par essaim de particules) a été proposée en 1997 par Kennedy et Eberhart [Kennedy et Eberhart, 1997]. Dans l'algorithme BPSO (de l'anglais: Binary Particle Swarm Optimization), la position de la particule i est représentée par un ensemble de bit. La vitesse v_{id} de la particule i est calculée à partir de l'équation (3.1). v_{id} est un ensemble de nombres réels qui doivent être transformés en un ensemble de probabilités en utilisant la fonction sigmoïde comme suit:

$$Sig(v_{id}) = \frac{1}{1 + e^{-v_{id}}} \quad (3.13)$$

Où $Sig(v_{id})$ représente la probabilité du bit x_{id} de prendre la valeur 1.

Afin d'échapper au problème de la divergence de l'essaim, La vitesse v_{id} est généralement limitée par une valeur maximale v_{max} et une valeur minimale $-v_{max}$, i.e. $v_{id} \in [-v_{max}, v_{max}]$.

En se basant sur la vitesse v_{id} de la particule i et sur la valeur de la probabilité $Sig(v_{id})$, la position de la particule x_{id} est calculée comme suit:

$$x_{id} = \begin{cases} 1 & \text{Si } r < Sig(v_{id}) \\ 0 & \text{Sinon} \end{cases} \quad (3.14)$$

Quelques années après l'apparition de la première version de l'algorithme BPSO, beaucoup de chercheurs qui s'intéressent à la résolution des problèmes d'optimisation ont proposé d'autres versions comme:

Wen-liang Zhong et al [Zhong et al, 2007] ont proposé une autre version de l'algorithme BPSO avec intégration d'un nouvel facteur de mutation c_3 dans le but de maintenir un bon équilibre entre l'intensification et la diversification de la recherche. Ils ont utilisé leur algorithme dans la résolution du problème du voyageur de commerce. Zhong et ses collègues ont prouvé que leur algorithme fonctionne bien même avec 200 villes.

Yalan Zhou et ses collègues [Zhou et al, 2007] ont proposé une autre version de l'algorithme BPSO fondée sur l'estimation de distribution afin de générer de nouvelles particules. L'algorithme proposé est nommé DEDPSO (Discret Estimation of Distribution PSO). Il a été appliqué sur des problèmes d'optimisation combinatoires. Les résultats expérimentaux montrent que l'algorithme proposé est plus performant que d'autres versions de l'algorithme BPSO.

Le but de participer dans l'optimisation de la conception des cellules de fabrication, Duran et ses collègues [Duran et al, 2008] ont proposé un algorithme d'optimisation par essaim particulaire binaire pour améliorer le processus du regroupement des machines dans les cellules de fabrication afin de minimiser les mouvements inter-cellules en tenant compte de la taille limitée des cellules. Les résultats obtenus prouvent la performance de l'algorithme proposé.

Fengqin Xu et ses collègues ont proposé en 2008 dans leur papier [Xu et al, 2008] un algorithme BPSO basé sur la coopération entre essaims. L'algorithme proposé est nommé CDPSO, il a été utilisé pour résoudre le problème du voyageur de commerce. Les résultats expérimentaux montrent que l'utilisation de cet algorithme a apporté des optimisations remarquables aux résultats par rapport à d'autres algorithmes BPSO classiques.

D'autre part et dans la même année, Hualong Yu et ses collègues ont publié un papier [Yu et al, 2008] dans lequel ils ont exposé un nouvel algorithme BPSO pour la sélection du plus petit ensemble de gènes responsables du cancer afin de pallier au problème de l'inexactitude des diagnostics médicaux. Les résultats expérimentaux montrent l'efficacité et la performance de la version proposée par rapport à d'autres méthodes en termes de précision et d'exploitation des données de grandes dimensions, notamment dans la reconnaissance du plus petit ensemble des gènes responsables des tumeurs du colon.

De leur part, Zhi-hui Zhan et Jun Zhang ont proposé en 2009 un algorithme BPSO [Zhan et Zhang, 2009] pour résoudre le problème de routage multi destination. L'algorithme proposé a donné de bons résultats par rapport à d'autres algorithmes.

Un autre algorithme BPSO multi-objectif est présenté dans [Sharaf et El-Gammal, 2009].

3.6. L'algorithme quantique de l'optimisation par essaim de particules (QPSO)

Les algorithmes d'optimisation par essaim de particules quantiques (QPSO: Quantum Particle Swarm Optimization) ont construit une nouvelle tendance qui a été remarquée cette dernière décennie. Cette nouvelle tendance vise à combiner les principes de la mécanique quantique et l'algorithme d'optimisation par essaim de particules afin de donner naissance à une nouvelle classe d'algorithmes d'optimisation par essaim de particules pour résoudre différents problèmes d'optimisation. Dans le modèle classique du PSO, la particule se déplace en tenant compte de sa position actuelle dans l'espace de recherche et sa vitesse de déplacement, ce qui n'est pas le cas dans le modèle du QPSO où la vitesse n'a pas d'influence sur le déplacement de la particule. En fait, dans la version quantique de l'algorithme PSO la particule se déplace selon l'équation (3.15) introduite dans [Sun et al, 2004]:

$$x_{id}(t) = \begin{cases} p_{id} - \beta \cdot |m_{bestid} - x_{id}(t)| \cdot \ln(1/u) & \text{Si } r \geq 0.5 \\ p_{id} + \beta \cdot |m_{bestid} - x_{id}(t)| \cdot \ln(1/u) & \text{Sinon} \end{cases} \quad (3.15)$$

$$\text{Avec } p_{id} = \varphi \cdot p_{bestid} + (1 - \varphi) \cdot g_{bestid} \quad (3.16)$$

$$\text{Et } m_{bestid} = \frac{1}{N} \sum_N p_{bestid} \quad (3.17)$$

m_{bestid} c'est la valeur moyenne des p_{bestid} . β est nommé coefficient de contraction-dilatation. u , r et φ sont tirés aléatoirement de l'intervalle $[0,1]$.

En se basant sur le principe de l'algorithme QPSO, plusieurs d'autres versions et applications de l'algorithme QPSO ont été proposées dans la littérature, nous en citons quelques unes dans ce qui suit de cette section.

Jiahai Wang et Yanlan Zhou ont proposé une autre version de l'algorithme QPSO nommée LQPSO [Wang et Zhou, 2007] dans laquelle ils ont intégré un opérateur de recherche locale pour permettre une bonne exploitation du voisinage de la meilleure solution trouvée par l'essaim.

De leur part, Coelho et ses collègues [Coelho et al, 2008] ont développé un algorithme d'optimisation par essaim de particules quantique en utilisant une distribution gaussienne pour régler les paramètres d'une conception basée sur la logique floue et les contrôleurs PID (Proportional-Integral-Derivative).

L'article publié par Millie Pant et ses collègues [Pant et al, 2008] présente une variante de l'algorithme QPSO nommée Q-QPSO. L'algorithme a été proposé pour résoudre les problèmes d'optimisation. Il a été testé et comparé avec un algorithme PSO classique, un autre algorithme QPSO et deux autres algorithmes tirés de la littérature sur six problèmes benchmarks. Les résultats obtenus montrent que la performance de Q-QPSO surpasse clairement celle des autres algorithmes qui ont été utilisés dans la comparaison.

En se basant sur deux types de la sélection : la sélection par tournoi et la sélection par roulette, Haixia Long et ses collègues ont proposé dans [Long et al, 2009] deux algorithmes inspirés QPSO: le QPSO-TS et le QPSO-RS. Les algorithmes proposés ont été testés sur des fonctions benchmarks et ils ont prouvé qu'ils sont plus performants que l'algorithme PSO standard et le QPSO simple.

Une revue sur l'optimisation par essaim de particules quantique est disponible dans [Fang et al, 2010].

3.7. Le PSO avec paramètres adaptatifs

Avant de se pencher dans la résolution de n'importe quel problème d'optimisation et penser aux différentes étapes de l'algorithme permettant d'aboutir à la solution convenable du problème à traiter, il faut d'abord songer à décrire le problème et spécifier les différents paramètres permettant de guider le processus de recherche. Afin de décrire le problème d'optimisation, un certain nombre de paramètres comme le domaine de définition des différentes variables, des paramètres de la (les) fonction (s) objectif (s) et le critère d'arrêt

nécessitent l'assistance de l'utilisateur afin de les définir car l'algorithme ne peut pas les définir tout seul [Cooren et al, 2009]. Cependant, certains paramètres peuvent être spécifiés et adaptés automatiquement au cours de la recherche en fonction des résultats trouvés. Une nouvelle tendance s'est développée afin d'affranchir l'utilisateur de la tâche du choix et d'adaptation des paramètres de contrôle comme la taille de l'essaim, la vitesse maximale de la particule, le coefficient d'inertie, les coefficients d'accélération...etc. L'idée consiste à confier cette responsabilité à l'algorithme lui-même. En fait, il peut décider et adapter les valeurs convenables des paramètres de contrôles sans assistance de l'utilisateur. Partant de cette idée, plusieurs chercheurs ont envisagé la possibilité d'adaptation dynamique et automatique des paramètres de l'algorithme PSO. Par conséquent, plusieurs études ont été proposées dans la littérature, nous en citons quelques unes dans ce qui suit de cette section.

L'étude analytique sur l'impact du coefficient d'inertie ω ainsi que la valeur maximale v_{\max} de la vitesse des particules a fait l'objectif de l'article présenté par Shi et Eberhart [Shi et Eberhart, 1998]. Selon les résultats expérimentaux, les auteurs ont proposé d'utiliser un système flou permettant le réglage dynamique de la valeur du coefficient d'inertie ω .

En 2003 Maurice Clerc a proposé un algorithme d'optimisation par essaim de particules nommé TRIBES [Clerc, 2003]. TRIBES est une nouvelle version de l'algorithme d'optimisation par essaim de particules totalement adaptative: sans paramètres de contrôle. En fait, TRIBES est caractérisé par une autonomie d'adaptation des paramètres. L'objectif de cet algorithme est d'exempter son utilisateur de la phase du réglage et du contrôle des paramètres de l'algorithme et de limiter son rôle à la spécification du problème en définissant uniquement la fonction objectif et le critère d'arrêt. Afin de garantir une bonne et rapide exploration de l'espace de recherche, Clerc propose de diviser l'essaim en sous-essaim de tailles différentes où chaque sous-essaim est appelé 'Tribu'. L'article présenté par Yann Cooren et ses collègues en 2009 [Cooren et al, 2009] expose une étude de performance de TRIBES. L'étude réalisée montre que TRIBES présente des performances compétitives à celles de la plupart des autres algorithmes PSO qui nécessitent un réglage manuel des paramètres.

Une version multi-objectif de TRIBES nommée MO-TRIBES est présentée dans [Cooren et al, 2011]. Elle garde le principe d'autonomie d'adaptation des paramètres présentés par TRIBES au fil de la procédure de recherche des solutions du problème posé. MO-TRIBES utilise une approche basée sur la dominance au sens de Pareto. Les résultats expérimentaux montrent que MO-TRIBES est un algorithme compétitif à d'autres algorithmes d'optimisation multi-objectif comme NSGA-II et MOPSO.

Yasuda et Iwasaki ont proposé dans [Yasuda et Iwasaki, 2004] un algorithme d'optimisation par essaim de particule qui s'occupe de la définition des valeurs des paramètres selon les résultats trouvés par les particules.

D'autre part, Ratnaweera et ses collègues [Ratnaweera et al, 2004] ont proposé un algorithme d'optimisation par essaim de particules nommé HPSO-TVAC qui s'occupe de la définition automatique des paramètres d'accélération, afin de remédier au problème de la convergence prématurée du PSO.

En outre, Chatterjee et Siarry ont proposé une version de l'algorithme d'optimisation par essaim de particules qui s'occupe de l'adaptation dynamique du coefficient d'inertie. Afin de permettre une adaptation convenable du coefficient d'inertie. L'algorithme proposé nécessite une spécification préalable du nombre maximum des itérations de l'algorithme. L'article proposé [Chatterjee et Siarry, 2006] présente également une nouvelle méthode pour la détermination d'un ensemble complet de paramètres pour n'importe quel problème donné. Les résultats expérimentaux montrent la performance de l'algorithme proposé qui a été testé sur plusieurs problèmes benchmarks et comparé avec d'autres algorithmes.

De même, l'article présenté par Fan et Chang [Fan et Chang, 2007] propose une optimisation par essaim particulaire avec intégration d'une fonction non linéaire afin de permettre l'adaptation dynamique du coefficient d'inertie ω au cours de l'optimisation. L'algorithme proposé a été testé et comparé avec d'autres algorithmes en utilisant des fonctions benchmarks. Les résultats obtenus montrent quelques promesses en termes de performance.

Très récemment, Liu et Abraham ont proposé [Liu et Abraham, 2009] un algorithme d'optimisation par essaim de particules avec perturbation de la vitesse des particules afin d'échapper à la stagnation de la recherche causée par la diminution de la vitesse. L'algorithme est nommé TPSO, il utilise une vitesse minimale v_{\min} pour contrôler la vitesse des particules. v_{\min} est adaptée par un contrôleur à base de la logique floue afin de régler la vitesse des particules durant la recherche. TPSO a montré de bons résultats par rapport à l'algorithme standard du PSO, l'algorithme génétique et l'algorithme du recuit simulé.

Le problème	Le nom de l'algorithme	L'année	Les auteurs	Les équations de déplacement
Le voyageur de commerce	○ C3DPSO	○ 2007	○ Zhong et al	○ Shi et Eberhart, 1998, avec adaptation Avec : $\omega=0.6$, $c_1=1.5$, $c_2=2$.
	○ CDPSO	○ 2008	○ Xu et al	○ Shi et Eberhart, 1998. Avec : $\omega \in [0.1, 0.9]$ $c_1=0.7$, $c_2=0.7$.
Traitement du cancer	○ A novel DPSO	○ 2008	○ Yu et al	○ Kennedy et Eberhart, 1995, avec adaptation.
	○ MPSO	○ 2008	○ Soundararajan et al	○ Shi et Eberhart, 1998. Avec : ω adaptatif, $c_1=4$, $c_2=4$.
	○ IPSO	○ 2009	○ Mohamad et al	○ Shi et Eberhart, 1998. Avec adaptation.
Financier	○ CPSO	○ 2007	○ Lee et al	Shi et Eberhart, 1998.
	○ BPSO01	○ 2007	○ Lee et al	Avec : $\omega \in [0.1, 0.4]$ $c_1=2$, $c_2=2$.
	○ BPSO02	○ 2007	○ Lee et al	Et v_{\max} dans CPSO=0.3 v_{\max} dans BPSO01=0.6 v_{\max} dans BPSO01=0.6
Conception des cellules de fabrication	○ A novel Discret PSO Algorithm	○ 2008	○ Duran et al	○ Shi et Eberhart, 1998.
Routage	○ DPSO	○ 2009	○ Zhan et Zhang	○ Shi et Eberhart, 1998. Avec : $\omega \in [0.4, 0.9]$ $c_1=2$, $c_2=2$.
	○ PSO Method	○ 2010	○ Moghadam et Seyedhosseini	○ Shi et Eberhart, 1998 Avec adaptation.
Approximation polygonale	○ DPSO-EDA	○ 2008	○ Wang. J et al	○ Kennedy et Eberhart, 1995, avec adaptation.
Segmentation d'image	○ PSO	○ 2005	○ Feng et al	○ Shi et Eberhart, 1998. Avec : $c_1=2$, $c_2=2$.
	○ PSO-based algorithm	○ 2006	○ Yin	○ Clerc et Kennedy, 2002.
	○ TRIBES	○ 2008	○ Nakib et al	○ Kennedy et Eberhart, 1995, avec adaptation.
La reconnaissance vocale	○ PSOBW	○ 2008	○ Yang et al	○ Shi et Eberhart, 1998. Avec : $\omega=0.7298$, $c_1=1.49618$, $c_2=1.49618$.

Tableau 3.1. Les applications des algorithmes PSO (Partie 1)

Le problème	Le nom de l'algorithme	L'année	Les auteurs	Les équations de déplacement
Le contrôle des systèmes	<ul style="list-style-type: none"> ○ G-QPSO ○ MTribes 	<ul style="list-style-type: none"> ○ 2008 ○ 2009 	<ul style="list-style-type: none"> ○ Coelho et al ○ Coelho et al 	<ul style="list-style-type: none"> ○ Equation spéciale avec $c_1=2.05$, $c_2=2.05$. ○ Kennedy et Eberhart, 1995, avec adaptation.
L'ordonnancement	<ul style="list-style-type: none"> ○ PSO+DR ○ HPSO ○ PSO algorithm ○ HPSO ○ GLN-PSOc 	<ul style="list-style-type: none"> ○ 2006 ○ 2007 ○ 2009 ○ 2005 ○ 2009 	<ul style="list-style-type: none"> ○ Allahverdi et Al-Anzi ○ Kuo et al ○ Sha et Lin ○ Liu et al ○ Pongchairerks 	<ul style="list-style-type: none"> ○ Kennedy et Eberhart, 1995, avec $v_{max}=0.95$. ○ Shi et Eberhart, 1998. <p>Avec : ω, c_1 et c_2 non constantes (diminuent non linéairement).</p> <ul style="list-style-type: none"> ○ Shi et Eberhart, 1998. <p>Avec : $\omega \in [0.7, 0.3]$, $c_1=0.7$ et $c_2=0.1$</p> <ul style="list-style-type: none"> ○ Shi et Eberhart, 1998. <p>Avec $v_{max}=4$ et : $\omega=1$, $c_1=2$, $c_2=2$.</p> <ul style="list-style-type: none"> ○ Pongchairerks et Kachitvichyanukul, 2005.
Dimensionnement de circuits électroniques analogiques	<ul style="list-style-type: none"> ○ TRIBES ○ MO-TRIBES 	<ul style="list-style-type: none"> ○ 2008 ○ 2008 	<ul style="list-style-type: none"> ○ Cooren ○ Cooren 	<ul style="list-style-type: none"> ○ Kennedy et Eberhart, 1995, avec adaptation. ○ Kennedy et Eberhart, 1995, avec adaptation.
Sac à dos multidimensionnel	<ul style="list-style-type: none"> ○ PSO-R 	<ul style="list-style-type: none"> ○ 2006 	<ul style="list-style-type: none"> ○ Kong et Tian 	<ul style="list-style-type: none"> ○ Kennedy et Eberhart, 1995 avec $v_{max}=2$ et $c_1=2$ et $c_2=2$.
Tournée de véhicules avec collecte et livraison	<ul style="list-style-type: none"> ○ PSO Method 	<ul style="list-style-type: none"> ○ 2008 	<ul style="list-style-type: none"> ○ Ai et Kachitvichyanukul 	<ul style="list-style-type: none"> ○ Pongchairerks et Kachitvichyanukul, 2005.

Tableau 3.1. Les applications des algorithmes PSO (Partie 2)

3.8. Les applications de l'algorithme PSO

Malgré son jeune âge par rapport à d'autres méthodes de résolution des problèmes d'optimisation, l'optimisation par essaim particulaire a été utilisée pour résoudre plusieurs problèmes dans différents domaines (La médecine, les finances, l'électronique, le traitement d'image, les problèmes d'ordonnancement ...etc). L'objectif de cette section est de présenter quelques applications récentes de l'algorithme PSO pour la résolution de plusieurs problèmes d'optimisation.

En 2005, Du Feng et ses collègues ont mis en œuvre l'algorithme PSO avec facteur d'inertie proposé en 1998 par Shi et Eberhart. L'algorithme proposé a été utilisé pour la

segmentation des images infrarouges [Feng et al, 2005]. L'étude expérimentale a montré de très bons résultats en termes de qualité de la segmentation (optimale) et du coût de calcul.

Allahverdi et Al-Anzi ont utilisé l'algorithme PSO pour résoudre le problème d'ordonnancement flow shop assemblé (The assembly flowshop scheduling problem) qui peut être utilisé dans la modélisation de plusieurs problèmes y compris la planification des requêtes dans les systèmes de bases de données distribuées et la fabrication des ordinateurs [Allahverdi et Al-Anzi, 2006]. Les résultats de comparaison de l'algorithme proposé avec d'autres algorithmes montrent qu'il est plus performant pour la résolution des problèmes où les priodes d'échéances sont très limitées.

Dans l'objectif de participer dans la résolution des problèmes d'optimisation combinatoires, Kong et Tian ont proposé en 2006 [Kong et Tian, 2006] un algorithme PSO binaire pour résoudre le problème du sac à dos multidimensionnel. Les auteurs ont intégré un opérateur de réparation spécifique au problème traité (Problem-Specific Repair Operator), afin de prendre en considération les contraintes du problème et garantir l'aboutissement à des solutions réalisables au cours de la recherche.

En 2006, Yin a appliqué la version de l'algorithme PSO proposée par Clerc et Kennedy en 2002 dans le domaine de l'imagerie [Yin, 2006]. L'objectif de Yin était d'améliorer la performance de la méthode à base du seuillage minimum de l'entropie croisée nommée: MCET (the minimum cross entropy thresholding) dans la segmentation des images dans le cas du seuillage multi-niveau. Les résultats expérimentaux montrent que la méthode proposée est efficacement adaptée aux applications temps réel.

En 2007, Kuo et ses collègues ont proposé une implémentation de la version continue de l'algorithme PSO basée sur le principe du codage à base de clé aléatoire (Random Key Encoding Scheme) [Kuo et al, 2007]. L'algorithme proposé est nommé HPSO. Il a été utilisé pour résoudre le problème d'ordonnancement Flow-shop: un des problèmes d'optimisation combinatoires. L'objectif était de trouver une séquence de tâches permettant de minimiser le temps d'exécution maximale. En termes de qualité des solutions, l'étude expérimentale montre que l'algorithme HPSO est plus performant que d'autres algorithmes. Cependant, il souffre du problème d'être tomber dans la vallée de l'optimum local.

Lee et ses collègues [Lee et al, 2007] ont proposé trois implémentations de l'algorithme PSO: PSO continu nommé CPSO, PSO discret binaire nommé BPSO01 et PSO discret binaire

avec mutation nommé BPSO02. Ces trois algorithmes ont été utilisés dans un contexte financier.

Sha et Lin ont proposé une version multi-objectif de l'algorithme PSO [Sha et Lin, 2009] pour résoudre le problème d'ordonnancement Flow-shop multi objectif respectant l'approche Pareto. L'étude expérimentale réalisée sur des problèmes benchmarks, montrent que la performance de l'algorithme proposé surpasse celles des autres algorithmes traditionnels en termes d'efficacité et de qualité de la recherche.

Soundararajan et ses collègues ont proposé en 2008 une application de l'algorithme PSO avec adaptation du coefficient d'inertie ω pour la résolution du problème de la chimiothérapie du cancer, où un équilibre entre la gestion des médicaments pour réduire la taille de la tumeur et les effets secondaires causés par ces médicaments est indispensable [Soundararajan et al, 2008]. Les auteurs ont établi une étude comparative entre la performance de l'algorithme proposé et celle des autres versions de l'algorithme PSO qui ont été proposées dans [Shi et Eberhart, 1998], [Chatterjee et Siarry, 2006] et [Fan et Chang, 2007].

Wang et ses collègues [Wang. J et al, 2008] ont proposé une application de l'algorithme PSO pour l'approximation polygonale utilisée dans le domaine de la reconnaissance des formes, l'infographie et la vision par ordinateur. L'algorithme est nommé DPO-EDA, il a été testé sur des problèmes benchmarks. L'étude expérimentale montre que l'algorithme DPO-EDA est plus performant que d'autres algorithmes proposés dans la littérature comme: l'algorithme génétique, la recherche tabou, le PSO standard et l'optimisation par colonies de fourmis.

Nakib et ses collègues et Cooren et ses collègues [Nakib et al, 2008], [Cooren et al, 2008a] ont utilisé l'algorithme PSO avec paramètres adaptatifs nommé TRIBES [Clerc, 2003], pour la résolution du problème de segmentation d'image IRM du cerveau en utilisant l'entropie exponentielle 2D et la segmentation d'image avec approximation de l'histogramme par une somme gaussienne, respectivement.

Deux autres applications de l'algorithme TRIBES et sa version multi-objectif nommée MO-TRIBES [Cooren et al, 2011] sont présentées dans [Cooren.Y, 2008] pour le traitement du problème de dimensionnement de circuits électroniques analogiques. Dans la première application, TRIBES a été utilisé dans le but de maximiser le gain d'un amplificateur de tension faible bruit par dimensionnement des transistors de celui-ci. Dans la deuxième application MO-TRIBES a été utilisé pour la satisfaction de deux objectifs simultanément :

minimiser la résistance d'entrée et maximiser la fréquence de coupure d'un convoyeur de courant de seconde génération.

D'autre part, Coelho et Bernert ont proposé une application modifiée de l'algorithme TRIBES pour la mise au point d'un contrôleur PID (proportional-integral-derivative), dans le cadre du contrôle des systèmes chaotiques [Coelho et Bernert, 2009]. L'algorithme proposé est nommé MTribes. Sa performance a été comparée avec celles des autres algorithmes d'optimisation continus y compris l'algorithme TRIBES et l'algorithme PSO classiques.

Mohamad et ses collègues ont proposé un algorithme binaire de l'optimisation par essaim de particules dans le cadre de la classification du cancer [Mohamad et al, 2009]. L'algorithme proposé a été utilisé pour la sélection du plus petit sous ensemble des gènes responsable du cancer. Les résultats obtenus montrent que cet algorithme est plus performant que d'autres algorithmes en termes de précision de la classification et du nombre des gènes sélectionnés.

Ai et Kachitvichyanukul ont proposé une application d'un algorithme PSO pour la résolution du problème de tournée de véhicules avec collecte et livraison simultanées. Selon les auteurs du papier [Ai et Kachitvichyanukul, 2008], l'algorithme proposé a donné de bons résultats.

Wang et ses collègues [Wang et al, 2009] ont proposé une adaptation de l'algorithme PSO pour résoudre les problèmes d'optimisation discrets binaires. L'algorithme proposé a été utilisé pour résoudre le problème du sac à dos multidimensionnel. L'étude expérimentale est réalisée sur des benchmarks du problème du sac à dos multidimensionnel. Une facilité de mise en œuvre et de bons résultats ont été remarqués par l'application de l'algorithme proposé.

Pongchairerks a proposé trois applications d'un algorithme PSO nommé GLNPSO, pour résoudre trois variantes du problème d'ordonnancement: problème d'ordonnancement Job-shop, problème d'ordonnancement Job-shop avec machines à usage multiple et le problème d'ordonnancement open-shop. L'algorithme proposé est nommé GLN-PSOc [Pongchairerks, 2009]. Les résultats expérimentaux prouvent la performance de l'algorithme GLN-PSOc.

Moghadam et Seyedhosseini ont proposé [Moghadam et Seyedhosseini, 2010] un algorithme d'optimisation par essaim de particules pour résoudre une variante spéciale du problème du routage de véhicule où la demande est supposée incertaine avec une distribution inconnue. L'algorithme a été appliqué pour résoudre un cas réel: la distribution de la drogue.

Pei a proposé dans [Pei, 2010], une application de l'algorithme PSO multi-objectif pour la résolution du problème d'ordonnancement work-flow. L'algorithme proposé est nommé MOPSO. Selon Yunxia Pei, l'algorithme MOPSO peut trouver facilement et avec précision les relations entre les tâches parallèles comme il peut réduire le temps d'exécution des tâches. Les résultats de comparaison de cet algorithme avec le fameux algorithme NSGA-II, montrent qu'il est plus performant en termes de temps de calcul.

Dans le cadre de la reconnaissance vocale continue, Yang et ses collègues ont réalisé une étude comparative d'application et de performance de l'algorithme PSO et de l'algorithme génétique [Yang et al, 2008] pour l'optimisation du HMM (Hidden Markov Model): la technologie de reconnaissance vocale la plus connue. Les résultats expérimentaux montrent que l'algorithme PSO est plus performant. Une autre étude comparative entre la performance de l'algorithme PSO binaire (BPSO) et la performance de l'algorithme génétique est disponible dans [Băutu. A et Băutu. E, 2007].

Dans le tableau 3.1 nous présentons les applications les plus récentes de l'algorithme d'optimisation par essaim de particules en spécifiant les différentes modifications apportées aux équations de déplacement des particules.

3. 9. Les classes des algorithmes proposés

Avant de présenter les algorithmes que nous avons proposés, nous expliquons d'abord la représentation des solutions ainsi que les différents opérateurs que nous avons utilisés.

3. 9.1. La représentation de solutions

Dans la version binaire de l'algorithme d'optimisation par essaim de particules, chaque particule est représentée par un ensemble de bits. Ainsi, pour représenter les positions et les vitesses des particules nous avons utilisé des vecteurs binaires de taille D. Où, D est le nombre d'objets à sélectionner (D=N). Les objets sélectionnés sont représentés par le bit 1, par contre les autres (i.e. les objets non sélectionnés) sont représentés par le bit 0. La représentation de la position (solution) d'une particule i est comme suit:

$$x_{id} = [x_{i1}, x_{i2}, \dots, x_{id}, \dots, x_{iD}]$$

$$\text{Où } x_{id} = \begin{cases} 1 & \text{Si l'objet est sélectionné} \\ 0 & \text{Sinon} \end{cases}$$

Supposant que nous avons 6 objets à sélectionner, donc $D = 6$. Une des solutions est la suivante:

$$x_{id} = [0, 0, 1, 0, 1, 1]$$

Ce qui veut dire: les objets 3, 5 et 6 sont sélectionnés. Par contre, les objets 1, 2 et 4 ne sont pas sélectionnés.

3. 9.2. La mise à jour des vitesses et des positions des particules

Dans l'optimisation par essaim de particules, la particule tend souvent à suivre sa meilleure position rencontrée au cours de la procédure de recherche ainsi que la meilleure position rencontrée par ses congénères. Afin de représenter ce principe, nous avons besoin à un certain nombre d'opérations et d'opérateurs qui sont définis ci-dessous:

1) $x_2 - x_1$

Supposant deux positions x_1 et x_2 :

$$x_2 - x_1 = \begin{cases} 1 & \text{Si } x_2 \neq x_1 \\ 0 & \text{Sinon (i.e. } x_2 = x_1) \end{cases}$$

2) $v_2 + v_1$

L'addition de deux vitesses v_1 et v_2 est une nouvelle vitesse v qui prend la valeur 1 si une ou les deux vitesses sont égales à 1, sinon v est égale à 0 (i.e. si $v_1 = v_2 = 0$).

$$v_2 + v_1 = \begin{cases} 1 & \text{Si } v_2 = 1 \text{ et /ou } v_1 = 1 \\ 0 & \text{Sinon (i.e. } v_1 = v_2 = 0) \end{cases}$$

3) Coefficient $\times L(t)$

$L(t)$ peut être une vitesse ou une position obtenue au temps t . Le coefficient peut être le coefficient de constriction, le coefficient d'inertie, un coefficient d'accélération...etc. Dans tous les cas, Coefficient $\times L(t)$ est défini comme suit :

$$\text{Coefficient} \times L(t) = \begin{cases} L(t) & \text{Si Coefficient} > r \\ 0 \text{ ou } L(t-1) & \text{Sinon} \end{cases}$$

Où r est un nombre aléatoire tiré de l'intervalle $[0,1]$.

4) $v+x$

L'addition de la vitesse et la position résulte à une nouvelle position x qui prend la valeur de la plus grande valeur entre v et x .

$$v+x = \begin{cases} 1 & \text{Si } x=1 \text{ et /ou } v=1 \\ 0 & \text{Sinon (i.e. } x = v = 0) \end{cases}$$

3.9.3. Les algorithmes proposés

Dans le but de résoudre le problème du sac à dos, nous avons proposé quatre classes d'algorithmes à base de la métaheuristique PSO. Dans chaque classe nous avons proposé quatre algorithmes avec équations et /ou paramètres différents.

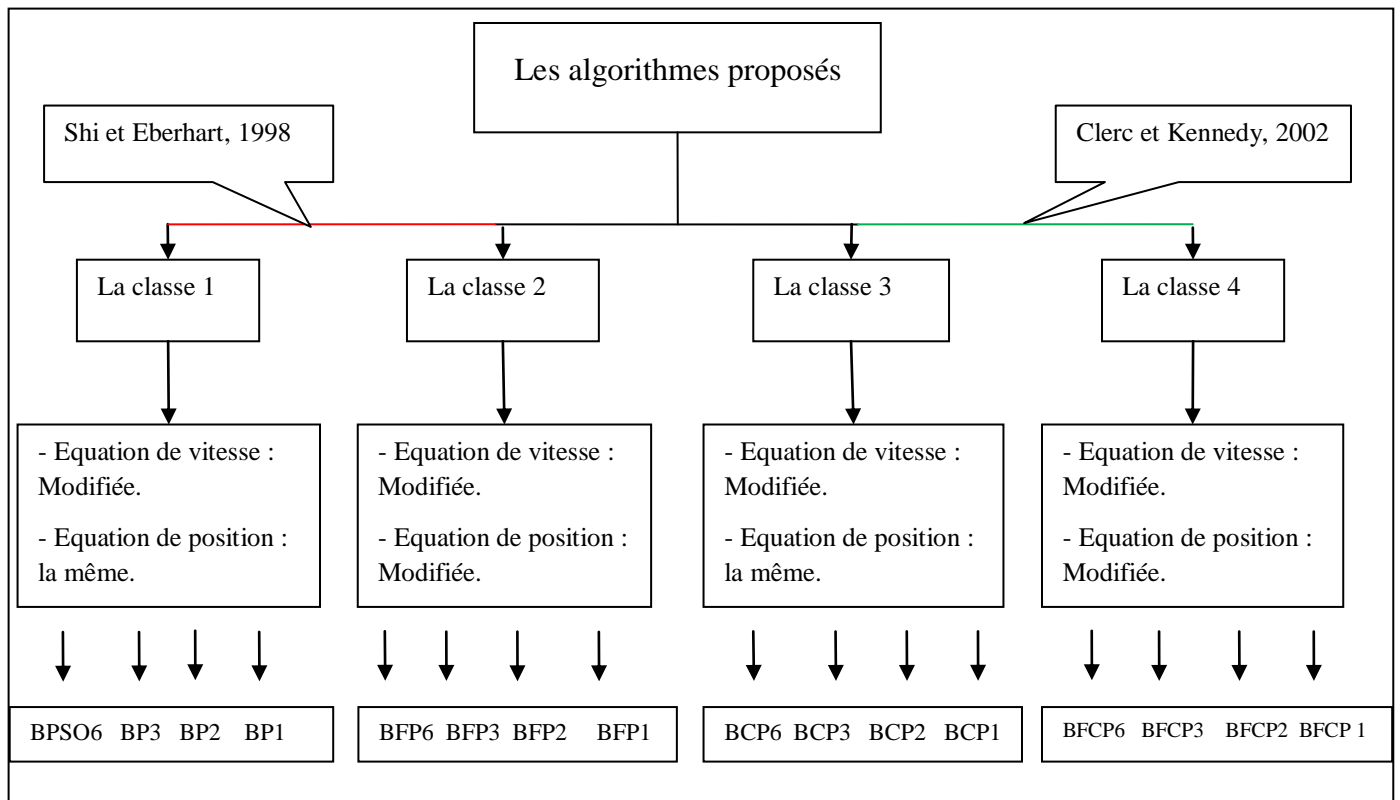


Figure 3.2. Les algorithmes proposés.

Dans la première classe nous avons utilisé avec adaptation la version de l'algorithme PSO avec facteur d'inertie ω proposé en 1998 par Shi et Eberhart [Shi et Eberhart, 1998]. Les algorithmes de cette classe sont : BPSO6, BP1, BP2 et BP3. Dans la deuxième classe nous nous sommes inspirés de l'idée d'utilisation du facteur de mutation proposé par Zhong et Zhang dans l'algorithme nommé C3DPSO [Zhong et Zhang, 2007] et nous avons proposé un nouveau facteur d'accélération F que nous avons appliqué pour la mise à jour des positions des particules. L'application du facteur F sur les algorithmes proposés dans la première classe

a donné naissance aux quatre algorithmes suivants: BFPSO6, BFP1, BFP2 et BFP3. Dans la troisième classe nous avons adapté et utilisé la version de l'algorithme PSO proposée par Clerc et Kennedy [Clerc et Kennedy, 2002]. Comme pour les autres classes, la troisième classe englobe les quatre algorithmes proposés dans la première classe mais avec facteur de constriction K proposé par Clerc et Kennedy et non pas avec facteur d'inertie ω . Les algorithmes proposés dans cette classe sont: BCP6, BCP1, BCP2 et BCP3. Quant à la quatrième classe, elle englobe les algorithmes définis dans la troisième classe avec application du nouveau facteur d'accélération F. La figure 3.2 présente les différents algorithmes proposés.

3. 9.3.1. Les algorithmes de la première classe

1. BPSO6

BPSO6 est une adaptation du standard PSO2006. Dans le BPSO6, nous avons proposé de mettre à jour les positions des particules selon l'équation (3.2) et les vitesses selon l'équation (3.18) comme suit:

$$v_{id}(t) = \omega v_{id}(t-1) + r_1 c_1 \times (p_{bestid}(t-1) - x_{id}(t-1)) + r_2 c_2 \times (l_{bestd}(t-1) - x_{id}(t-1)) \quad (3.18)$$

Où l_{bestd} est la meilleure position trouvée par les particules dans la dimension d d'un voisinage donné. c_1 et c_2 sont choisis aléatoirement à chaque itération. A l'opposé du standard PSO2006, la taille de l'essaim dans l'algorithme BPSO6 est égale à la taille du problème.

2. BP3

Dans le BP3, les positions des particules sont mises à jour selon l'équation (3.2) alors que les vitesses sont mises à jour en utilisant l'équation (3.18). c_1 et c_2 sont des constantes et la taille de l'essaim est égale à la taille du problème.

3. BP2

Dans le BP2, nous avons proposé de mettre à jour les positions des particules selon l'équation (3.2) et les vitesses selon l'équation (3.19) définie comme suit:

$$v_{id}(t) = \omega v_{id}(t-1) + r_1 c_1 \times (p_{bestid}(t-1) - x_{id}(t-1)) + r_2 c_2 \times (l_{bestid}(t-1) - x_{id}(t-1)) + r_3 c_3 \times (g_{bestd}(t-1) - x_{id}(t-1)) \quad (3.19)$$

Où c_1 , c_2 et c_3 sont des constantes et la taille de l'essaim est égale à la taille du problème.

4. BP1

Dans l'objectif d'apporter plus de diversification au sein de l'essaim, nous nous sommes inspirés de l'algorithme PSOPC [He et al, 2004] et nous avons proposé l'algorithme BP1 dans lequel les positions des particules sont mises à jour selon l'équation (3.2) et les vitesses des particules sont mises à jour selon l'équation (3.20) définie comme suit:

$$v_{id}(t) = \omega v_{id}(t-1) + r_1 c_1 \times (p_{bestid}(t-1) - x_{id}(t-1)) + r_2 c_2 \times (l_{bestid}(t-1) - x_{id}(t-1)) + r_3 c_3 \times (g_{bestid}(t-1) - x_{id}(t-1)) + r_4 c_4 \times (P_{id}^r(t-1) - x_{id}(t-1)) \quad (3.20)$$

Où P_{id}^r est la position d'une particule i dans la dimension d . c_1 , c_2 , c_3 et c_4 sont des constantes.

3.9.3.2. Les algorithmes de la deuxième classe

Les algorithmes de cette classe se basent sur les algorithmes proposés dans la première classe avec modification de l'équation de mise à jour des positions des particules comme expliqué ci-dessous.

1. BFP6

Dans l'algorithme BFP6, les positions des particules sont mises à jour selon l'équation (3.21):

$$x_{id}(t) = rF \times x_{id}(t-1) + v_{id}(t) \quad (3.21)$$

Les vitesses des particules sont mises à jour selon l'équation (3.18).

2. BFP3

Dans le BFP3, les positions des particules sont mises à jour selon l'équation (3.21) et les vitesses des particules sont mises à jour selon l'équation (3.18), mais c_1 et c_2 sont des constantes.

3. BFP2

Dans le BFP2, Les positions des particules sont mises à jour selon l'équation (3.21) et les vitesses sont mises à jour selon l'équation (3.19).

4. BFP1

Dans le BFP1, Les positions des particules sont mises à jour selon l'équation (3.21). Les vitesses sont mises à jour selon l'équation (3.20).

3.9.3.3. Les algorithmes de la troisième classe

Dans la troisième classe, la taille de l'essaim est égale à la dimension du problème, les positions des particules sont mises à jour selon l'équation (3.2).

1. BCP6

Dans l'algorithme BCP6, les vitesses des particules sont mises à jour selon l'équation suivante:

$$v_{id}(t) = K [v_{id}(t-1) + r_1 c_1 \times (p_{bestid}(t-1) - x_{id}(t-1)) + r_2 c_2 \times (l_{bestid}(t-1) - x_{id}(t-1))] \quad (3.22)$$

Où c_1 et c_2 sont choisis aléatoirement à chaque itération.

2. BCP3

Dans le BCP3, les vitesses des particules sont mises à jour à partir de l'équation (3.22), mais c_1 et c_2 sont des constantes.

3. BCP2

Dans le BCP2, les vitesses des particules sont mises à jour selon l'équation suivante:

$$v_{id}(t) = K [v_{id}(t-1) + r_1 c_1 \times (p_{bestid}(t-1) - x_{id}(t-1)) + r_2 c_2 \times (l_{bestid}(t-1) - x_{id}(t-1)) + r_3 c_3 \times (g_{bestid}(t-1) - x_{id}(t-1))] \quad (3.23)$$

Où c_1 , c_2 et c_3 sont des constantes.

4. BCP1

Dans le BCP1 nous avons appliqué le facteur de constriction proposée par Clerc et Kennedy [Clerc et Kennedy, 2002] sur l'équation (3.20) que nous avons proposée dans l'algorithme BP1 de la première classe. Nous avons donc proposé de mettre à jour les vitesses des particules selon l'équation (3.24) définie ci-dessous:

$$v_{id}(t) = K [v_{id}(t-1) + r_1 c_1 \times (p_{bestid}(t-1) - x_{id}(t-1)) + r_2 c_2 \times (l_{bestid}(t-1) - x_{id}(t-1)) + r_3 c_3 \times (g_{bestid}(t-1) - x_{id}(t-1)) + r_4 c_4 \times (P_{id}^r(t-1) - x_{id}(t-1))] \quad (3.24)$$

Où c_1 , c_2 , c_3 et c_4 sont des constantes ; $K=0.7$.

3.9.3.4. Les algorithmes de la quatrième classe

Dans les algorithmes de la quatrième classe, les positions des particules sont mises à jour selon l'équation (3.21).

1. BFCP6

Dans l'algorithme BFCP6, les vitesses sont mises à jour selon l'équation (3.22).

2. BFCP3

Dans le BFCP3, les vitesses sont mises à jour selon l'équation (3.22), mais c_1 et c_2 sont des constantes.

3. BFCP2

Dans le BFCP2, les vitesses sont mises à jour selon l'équation (3.23).

4. BFCP1

Dans le BFCP2, les vitesses sont mises à jour selon l'équation (3.24).

3. 10. Comparaison et résultats expérimentaux

Pour vérifier, prouver et comparer la performance des algorithmes proposés, nous avons généré six instances avec nombre d'objets différents. Dans la première instance le nombre N d'objets est égal à 120. Dans la deuxième instance, N=200. Dans la troisième, N=500. Ensuite, N=700, 900, 1000 dans la quatrième, cinquième et sixième instance respectivement. La capacité du sac est calculée à partir de la formule suivante:

$$C = \frac{3}{4} \sum_{i=1}^N w_i \quad (3.25)$$

Au début, nous avons réalisé une étude comparative entre les algorithmes des quatre classes proposées. Ensuite, nous avons comparé les 16 algorithmes avec l'algorithme binaire standard de l'optimisation par essaim de particules (Standard BPSO) [Kennedy et Eberhart, 1997] et avec le standard PSO nommé PSO2006 que nous avons adapté selon la représentation binaire que nous avons utilisée et appliquée sur le problème du sac à dos. En outre, nous avons essayé de valoriser nos résultats statistiquement en utilisant le test statistique de Friedman afin de tester la signifiante de la différence des résultats obtenus par nos algorithmes, le standard BPSO et le standard PSO2006. Chaque algorithme a été exécuté 25 fois à chaque expérimentation.

Les poids ω_i et les valeurs p_i des objets, ont été choisis aléatoirement. Pour les 16 algorithmes, la taille de l'essaim est égale au nombre d'objets (i.e. $S=N$). Dans la 1^{ère} et la 2^{ème} classe, $\omega=0.7$. Dans la 3^{ème} et la 4^{ème} classe, K n'a pas été calculé à partir de la formule définie

par Clerc et Kennedy (i.e. $K = \frac{2}{|2 - \varphi - \sqrt{\varphi(\varphi - 4)}|}$), par contre il a été fixé à 0.7.

Les valeurs des paramètres c_1 , c_2 , c_3 , c_4 et F sont respectivement: 0.3, 0.4, 0.7, 0.1 et 0.9. Exceptionnellement dans BPSO6, BFP6, BCP6 et BFCP6, les paramètres c_1 , c_2 sont tirés aléatoirement de l'intervalle [0,1].

Les positions initiales des particules sont initialisées aléatoirement à chaque exécution. Les vitesses sont initialisées avec la valeur 0. Le nombre d'itérations à chaque exécution est fixé à 15 et il est utilisé comme critère d'arrêt pour chaque exécution.

Quant aux paramètres du PSO2006 Standard, nous avons gardé les mêmes paramètres définis dans «<http://www.particleswarm.info/Programs.html>», mais avec représentation binaire des positions et des vitesses des particules.

En ce qui concerne le Standard BPSO, nous avons respecté les équations, la représentation et les paramètres définis dans «<http://www.particleswarm.info/Programs.html>», sauf que les valeurs de c_1 et c_2 sont égales à celles utilisées pour le test des algorithmes que nous avons proposés, i.e. $c_1, c_2 = 0.3, 0.4$, respectivement.

Les tableaux de 3.2 à 3.5 représentent les résultats expérimentaux des algorithmes de chaque classe. Pour chaque tableau, la première colonne (i.e. Instance) représente le nombre d'objets et le numéro d'expérimentation. La deuxième colonne (Best known) représente la meilleure solution trouvée par les quatre algorithmes de la classe concernée. Les colonnes 3, 4, 5 et 6 représentent les meilleures solutions et les moyennes (averages) des solutions des algorithmes de la classe concernée. Les résultats sont testés en 5 expérimentations pour chaque instance. Chacune est testée 25 fois indépendamment. « Average » représente la moyenne des résultats (Avg) de l'ensemble des exécutions (i.e. pour 125 exécutions).

Les tableaux 3.6 et 3.7 présentent un résumé comparatif des résultats obtenus par les algorithmes des quatre classes: classe 1 et 2 dans le tableau 3.6 et classe 3 et 4 dans le tableau 3.7. La première colonne de chaque tableau représente l'instance (i.e. le nombre d'objets). La deuxième colonne et la troisième colonne (Classe 1 et Classe 2 dans le tableau 3.6 et Classe 3 et Classe 4 dans le tableau 3.7) présentent la meilleure (best) solution et la moyenne (average) obtenues avec chaque instance par les algorithmes de chaque classe.

Le tableau 3.8 complète les tableaux 3.6 et 3.7. Il présente les résultats expérimentaux des algorithmes proposés, le Standard PSO2006 et le Standard BPSO pour chaque instance durant les 125 exécutions. La première colonne présente l'instance. La deuxième colonne présente les meilleures valeurs des meilleures solutions et des meilleures moyennes obtenues par les algorithmes des quatre classes. La troisième colonne et la quatrième colonne représentent les meilleures solutions et les meilleures moyennes obtenues par le Standard BSPO et le Standard PSO2006 respectivement.

Pour chaque instance dans les tableaux 3.6, 3.7 et 3.8, la première ligne représente la meilleure solution et la deuxième ligne représente la moyenne.

Les résultats de comparaison des algorithmes des quatre classes nous montrent que:

- ✓ L'utilisation de la version de l'algorithme PSO avec facteur de constriction K donne de bons averages par rapport à la version de l'algorithme PSO avec facteur d'inertie ω .
- ✓ L'utilisation de la version de l'algorithme PSO avec facteur de constriction donne de bonnes solutions (en termes de meilleures solution trouvées: the best) dans la plupart des cas.
- ✓ Dans la plupart des cas, l'ajout du coefficient d'accélération F que nous avons proposé aux algorithmes de la première classe a amélioré leurs résultats en termes de average.
- ✓ L'ajout du coefficient d'accélération F aux algorithmes de la troisième classe a amélioré leurs résultats.
- ✓ Dans la plupart des cas, l'application du facteur d'accélération F sur la version de l'algorithme PSO avec facteur de constriction donne de bons averages comparé avec son application sur la version du PSO avec facteur d'inertie.
- ✓ Les meilleurs résultats sont obtenus par les B*P3, i.e. BP3, BFP3, BCP3 et BFCP3.

Le test statistique de Friedman présenté dans la figure 3.3 présente une comparaison des résultats obtenus par les algorithmes que nous avons proposés, le standard BPSO et le standard PSO2006. Selon la figure 3.3, la performance de tous les algorithmes proposés dépasse celle du standard BSPO et PSO2006. En termes de signifiante statistique des performances, le test de Friedman nous montre que la différence entre les résultats obtenus par les couples d'algorithmes (BCP3, BPSO), (BFCP1, BPSO) et (BFCP3, BPSO) est signifiante (grande et remarquable).

La figure 3.4 représente une comparaison de la moyenne du temps de calcul nécessaire calculé en secondes pour les 16 algorithmes proposés, le Standard PSO2006 (SPSO2006 dans la figure) et le Standard BPSO (SBPSO dans la figure). La figure représente les résultats de 125 exécutions de l'instance de 1000 objets.

Instance		Best Known	BPSO6		BP1		BP2		BP3	
N	N°-ex		Best	Avg	Best	Avg	Best	Avg	Best	Avg
120	1	4552	4439	4206	4368	4128	4552	4141	4390	4257
	2	4472	4390	4178	4296	4107	4472	4153	4395	4251
	3	4457	4342	4167	4402	4131	4456	4111	4457	4220
	4	4452	4439	4196	4374	4131	4452	4150	4425	4256
	5	4469	4357	4157	4469	4155	4383	4125	4391	4198
Average				4180,8		4130,4		4136		4236,4
200	1	7522	7340	7135	7522	6992	7256	6925	7422	7195
	2	7642	7363	7025	7457	7000	7642	6985	7461	7144
	3	7559	7559	7131	7277	7008	7398	6966	7468	7181
	4	7501	7420	7077	7501	6990	7423	6861	7356	7133
	5	7590	7426	7154	7277	6956	7590	6953	7490	7196
Average				7104,4		6989,2		6938		7169,8
500	1	17794	17733	17007	17520	16917	17206	16640	17794	17231
	2	17776	17776	16837	17647	16883	16869	16546	17674	17210
	3	18058	17463	17001	17495	16854	16975	16520	18058	17257
	4	17698	17698	16922	17365	16800	17662	16600	17532	17177
	5	17682	17530	16978	17183	16787	17682	16686	17620	17245
Average				16949		16848,2		16598,4		17224
700	1	24431	24296	23318	24263	23354	24101	23073	24431	23885
	2	24295	24220	23227	23904	23272	23902	23058	24295	23916
	3	24678	24678	23290	23955	23352	23372	22984	24388	23848
	4	24429	24376	23202	23890	23333	23915	23011	24429	23906
	5	24640	24640	23226	24063	23367	23712	22971	24363	23783
Average				23252,6		23335,6		23019,4		23867,6
900	1	31686	31220	29758	30415	29797	29611	29245	31686	30644
	2	31242	31242	29861	30750	29947	30367	29407	30925	30439
	3	30956	30803	29466	30868	29841	29734	29322	30956	30415
	4	31651	30517	29539	30898	29823	31215	29386	31651	30594
	5	31428	31428	29673	30500	29761	30783	29485	31022	30457
Average				29659,4		29833,8		29369		30509,8
1000	1	34476	34467	32808	33599	33035	33964	32496	34476	33846
	2	34401	34301	32976	34319	33139	33032	32520	34401	33775
	3	34654	34654	32682	33603	32950	33582	32569	34364	33752
	4	35019	34334	33001	33925	33081	34213	32676	35019	33906
	5	34528	34368	33094	34008	33120	33214	32532	34528	33774
Average				32912,2		33065		32558,6		33810,6

Tableau 3.2. L'étude expérimentale des résultats des algorithmes de la 1^{ère} classe.

Instance		Best Known	BFP6		BFP1		BFP2		BFP3	
N	N°-ex		Best	Avg	Best	Avg	Best	Avg	Best	Avg
120	1	4412	4412	4196	4358	4131	4349	4145	4397	4210
	2	4564	4383	4194	4355	4150	4354	4101	4564	4245
	3	4472	4310	4213	4389	4177	4348	4126	4472	4257
	4	4497	4463	4195	4481	4082	4497	4137	4438	4224
	5	4444	4434	4218	4417	4162	4444	4178	4395	4246
Average				4203,2		4140,4		4137,4		4236,4
200	1	7648	7467	7136	7266	6942	7648	6954	7498	7129
	2	7445	7445	7119	7313	6965	7367	6933	7395	7158
	3	7624	7412	7089	7371	6977	7360	6959	7624	7178
	4	7428	7380	7092	7306	6960	7428	7068	7422	7168
	5	7491	7491	7107	7339	7017	7387	6985	7483	7209
Average				7108,6		6972,2		6979,8		7168,4
500	1	17747	17145	17531	16861	16782	16589	17620	17194	17747
	2	17598	17007	17642	16858	16962	16539	17597	17100	17598
	3	17810	17138	17550	16857	17647	16615	17974	17141	17810
	4	17527	17105	17330	16914	16934	16505	17735	17246	17527
	5	17652	17076	17244	16834	17177	16573	17554	17198	17652
Average				17094,2		16864,8		16564,2		17175,8
700	1	24809	24809	23778	23905	23404	24177	22973	24216	23789
	2	24469	24326	23810	24012	23338	24469	23213	24236	23843
	3	24563	24318	23775	24407	23393	23629	22977	24563	23921
	4	24398	24297	23714	23869	23331	24003	22988	24398	23876
	5	24509	24165	23761	23896	23376	24061	22988	24509	23901
Average				23767,6		23368,4		23027,8		23866
900	1	31525	30991	30257	30185	29698	31525	29326	30943	30476
	2	31192	31192	30520	30829	29860	30726	29359	30980	30491
	3	31108	31108	30322	30778	29860	30408	29312	30956	30521
	4	31096	30953	30381	31018	29847	30777	29386	31096	30496
	5	31172	30989	30476	30839	29943	29772	29267	31172	30553
Average				30391,2		29841,6		29330		30507,4
1000	1	34558	34514	33377	33721	33024	33928	32582	34558	33933
	2	34596	34365	33738	33707	32958	33057	32467	34596	33881
	3	34568	34568	33717	33442	32933	32999	32582	34421	33838
	4	34536	34536	33717	34273	33138	33032	32560	34381	33919
	5	34847	34847	33636	34009	33062	33074	32525	34426	33841
Average				33637		33023		32543,2		33882,4

Tableau 3.3. L'étude expérimentale des résultats des algorithmes de la 2^{ème} classe.

Instance		Best Known	BCP6		BCP1		BCP2		BCP3	
N	N°-ex		Best	Avg	Best	Avg	Best	Avg	Best	Avg
120	1	4427	4423	4226	4367	4204	4427	4261	4360	4240
	2	4545	4467	4226	4408	4219	4545	4214	4391	4230
	3	4533	4519	4190	4497	4216	4434	4208	4533	4262
	4	4543	4543	4248	4370	4203	4392	4149	4444	4247
	5	4434	4413	4241	4434	4239	4412	4188	4387	4233
Average				4226,2		4216,2		4204		4242,4
200	1	7484	7417	7151	7484	7090	7356	7129	7285	7139
	2	7556	7473	7098	7444	7117	7397	7118	7556	7189
	3	7503	7447	7163	7503	7086	7474	7166	7418	7235
	4	7476	7428	7137	7476	7169	7341	7086	7420	7203
	5	7614	7506	7148	7614	7169	7394	7053	7411	7149
Average				7139,4		7126,2		7110,4		7183
500	1	17983	17922	17135	17983	17154	17384	17137	17739	17261
	2	17744	17744	17132	17712	17057	17514	17133	17693	17288
	3	17810	17438	16998	17511	17066	17810	17192	17810	17268
	4	17906	17744	17087	17740	17090	17672	17200	17906	17304
	5	17802	17802	16994	17791	17102	17726	17136	17651	17206
Average				17069,2		17093,8		17159,6		17265,4
700	1	24729	24208	23493	24384	23747	24558	23808	24729	23866
	2	24603	24399	23566	24603	23612	24441	23790	24433	23900
	3	24639	24639	23617	24212	23568	24312	23680	24534	23915
	4	24552	24552	23623	24135	23595	24424	23743	24466	23984
	5	24651	24651	23649	24408	23665	24396	23915	24471	23969
Average				23589,6		23637,4		23787,2		23926,8
900	1	31137	31045	30095	31018	30073	31017	30352	31137	30543
	2	31488	31488	30230	30895	30034	31105	30583	30744	30459
	3	31081	30927	30047	31013	30321	30875	30407	31081	30550
	4	31295	30905	30141	31295	29947	30993	30496	30941	30496
	5	31240	30891	30042	30910	29940	31069	30358	31240	30640
Average				30111		30063		30439,2		30537,6
1000	1	34811	34294	33332	34657	33428	34671	33793	34811	33927
	2	34682	34680	33281	34464	33281	34466	33816	34682	33906
	3	34609	33835	33156	34487	33322	34609	33774	34341	33891
	4	34606	34099	33275	34606	33524	34188	33806	34456	33918
	5	35097	34491	33414	34630	33369	34863	33909	35097	33912
Average				33291,6		33384,8		33819,6		33910,8

Tableau 3.4. L'étude expérimentale des résultats des algorithmes de la 3^{ème} classe.

Chapitre 3: Ensemble d'algorithmes d'optimisation par essaim de particules

Instance		Best Known	BFCEP6		BFCEP1		BFCEP2		BFCEP3	
N	N°-ex		Best	Avg	Best	Avg	Best	Avg	Best	Avg
120	1	4512	4369	4205	4409	4229	4512	4222	4407	4269
	2	4538	4536	4247	4538	4277	4471	4207	4478	4252
	3	4552	4552	4240	4414	4194	4361	4212	4376	4238
	4	4468	4379	4205	4468	4214	4435	4220	4487	4289
	5	4468	4444	4252	4468	4238	4427	4174	4412	4239
Average				4229,8		4230,4		4207		4257,4
200	1	7646	7436	7178	7646	7097	7376	7081	7555	7213
	2	7681	7531	7111	7429	7178	7681	7145	7443	7166
	3	7484	7484	7115	7461	7105	7348	7088	7486	7188
	4	7490	7413	7168	7467	7148	7373	7075	7415	7220
	5	7420	7420	7183	7361	7140	7376	7096	7490	7231
Average				7151		7133,6		7097		7203,6
500	1	18332	17481	17151	17629	17216	17482	17178	18332	17299
	2	17828	17514	17149	17666	17197	17705	17125	17828	17271
	3	17742	17642	17194	17576	17106	17439	17081	17742	17280
	4	17838	17410	17160	17838	17205	17624	17151	17639	17197
	5	17731	17620	17225	17673	17238	17731	17112	17665	17293
Average				17175,8		17192,4		17129,4		17268
700	1	24577	24480	23799	24577	23892	24268	23803	24496	23902
	2	24951	24540	23935	24634	23802	24326	23755	24951	23952
	3	24460	24352	23876	24460	23887	24195	23782	24403	23904
	4	24825	24250	23865	24579	23721	24415	23785	24825	23988
	5	24633	24509	23905	24633	23873	24163	23738	24479	23927
Average				23876		23835		23772,6		23934,6
900	1	31478	31233	30529	31392	30551	31021	30510	31478	30507
	2	31425	31001	30487	31425	30446	30877	30391	30906	30422
	3	31359	31209	30337	31027	30531	30981	30384	31359	30490
	4	31200	31120	30412	31142	30586	31067	30564	31200	30630
	5	31448	31324	30475	31448	30409	31082	30427	31091	30427
Average				30448		30504,6		30455,2		30495,2
1000	1	34737	34416	33851	34737	33850	34398	33798	34280	33787
	2	34934	34934	33930	34745	33893	34334	33765	34667	33849
	3	34822	34822	33954	34426	33833	34357	33765	34581	33883
	4	34583	34453	33881	34583	33964	34404	33762	34279	33823
	5	34717	34251	33781	34697	33872	34510	33790	34717	33863
Average				33879,4		33882,4		33776		33841

Tableau 3.5. L'étude expérimentale des résultats des algorithmes de la 4^{ème} classe.

Instance	Classe 1				Classe 2			
	BPSO6	BP1	BP2	BP3	BFP6	BFP1	BFP2	BFP3
120	4439	4469	4552	4457	4463	4489	4497	4564
	4180,8	4130,4	4136	4236,4	4203,2	4140,4	4137,4	4236,4
200	7559	7522	7642	7490	7491	7339	7648	7624
	7104,4	6989,9	6938	7169,8	7108,6	6972,2	6979,8	7168,4
500	17776	17647	17682	18058	17810	17642	17647	17974
	16949	16848,2	16598,4	17224	17094,2	16864,8	16564,2	17175,8
700	24678	24407	24101	24431	24809	24407	24469	24563
	23252,6	23335,6	23019,4	23867,6	23767,6	23368,4	23027,8	30507,4
900	31428	30898	31215	31686	31192	31018	31525	31276
	29659,4	29833,8	29369	30509,8	30391,2	29841,6	29330	30507,4
1000	34654	34319	34213	35019	34847	34273	33999	34596
	32912,2	33065	32558,6	33810,6	33637	33023	32543,2	33882,4
2000	68857	66605	66546	68548	67914	67110	67507	67829
	63232,75	65030,6	63754,6	66478	66185,2	65037	63725	66716,6

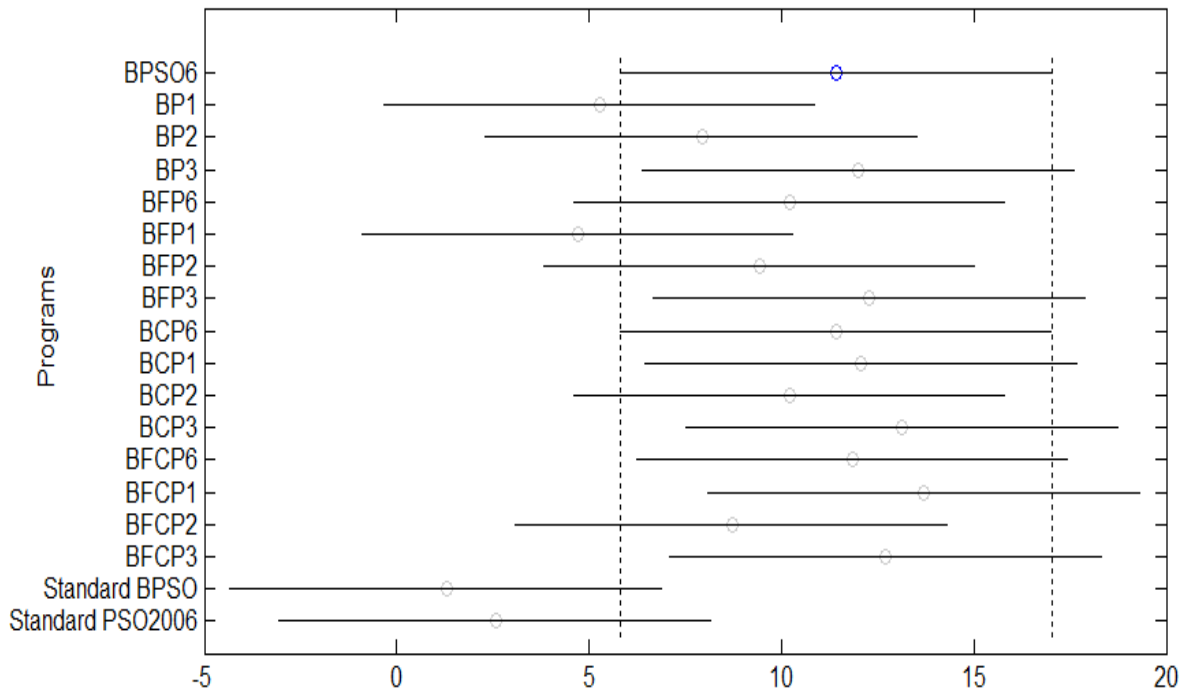
Tableau 3.6. Comparaison des résultats obtenus par les algorithmes des classes 3 et 4.

Instance	Classe 3				Classe 4			
	BCP6	BCP1	BCP2	BCP3	BFCP6	BFCP1	BFCP2	BFCP3
120	4543	4497	4545	4533	4552	4538	4512	4487
	4226,2	4216,2	4204	4242,4	4229,8	4230,4	4207	4257,4
200	7506	7614	7474	7556	7531	7646	7681	7555
	7139,4	7126,2	7110,4	7183	7151	7133,6	7097	7203,6
500	17922	17983	17810	17906	17642	17838	17731	18332
	17069,2	17093,8	17159,6	17265,4	17175,8	17192,4	17129,4	17268
700	24651	24603	24558	24729	24540	24634	24326	24951
	23589,6	23637,4	23787,2	23926,8	23876	23835	23772,6	23934,6
900	31488	31295	31105	31240	31324	31448	31082	31478
	30111	30063	30439,2	30537,6	30448	30504,6	30455,2	30495,2
1000	34680	34606	34863	35897	34934	34745	34510	34717
	33291,6	33384,8	33819,6	33910,8	33879,4	33882,4	33776	33863
2000	66703	68266	67856	67995	68784	68378	67645	67689
	65275,2	66656,8	66615	66779,75	66322,8	66740	66682,4	66630,8

Tableau 3.7. Comparaison des résultats obtenus par les algorithmes des classes 3 et 4.

Instance	Best Known	Standard BPSO	Standard PSO2006
120	4564	4296	4331
	4257,4	3840,8	4027
200	7681	7456	7391
	7203,6	5703	6819,4
500	18332	13116	17618
	17268	12471,2	16244,4
700	24951	18276	23893
	30537,6	17097,4	22400,2
900	31686	22857	30770
	30537,6	21736,6	28574,2
1000	35897	24933	34025
	33910,8	24050	31682,2
2000	68857	47674	67006
	66779,75	46538,8	63265,8

Tableau 3.8. Comparaison de meilleures solutions obtenues par les algorithmes proposés, le Standard PSO2006 et le Standard BPSO.



The mean column ranks of groups BCP3 and Standard BPSO are significantly different

The mean column ranks of groups BFCP1 and Standard BPSO are significantly different

Figure 3.3. Le résultat du test de Friedman entre les algorithmes proposés, le standard BPSO et le standard PSO2006 sur des instances KP

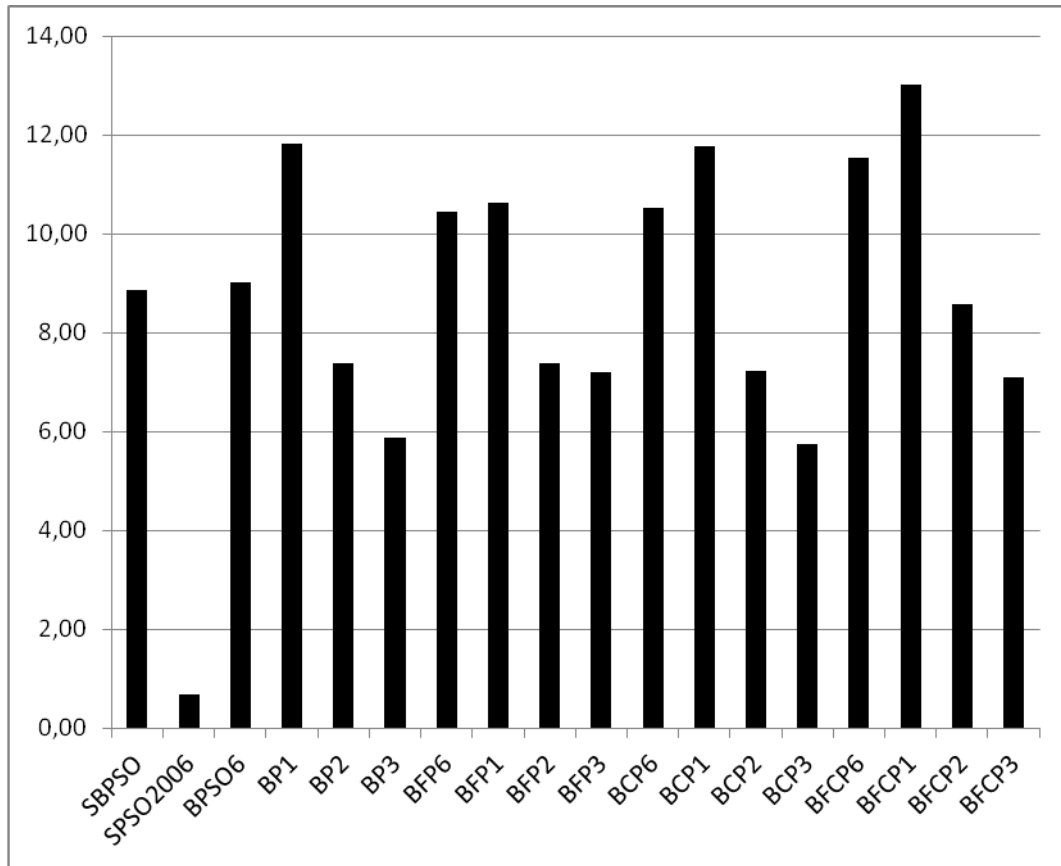


Figure 3.4. Moyenne de temps d'exécution des algorithmes proposés, le Standard PSO2006 et le Standard BPSO.

En terme de temps de calcul, la figure 3.4 nous montre que:

- ✓ l'algorithme BCP3 est le moins couteux. Le BFCP1 est le plus couteux entre les 16 algorithmes proposés.
- ✓ Les algorithmes B*P6 et B*P1 (i.e. BP2, BFP2, BCP2, BFCP2, BP3, BFP3, BCP3 et BFCP3) convergent plus rapidement que le Standard BPSO (SBPSO dans la figure).
- ✓ Le Standard PSO2006 converge plus rapidement que les algorithmes proposés.

Toutefois, les résultats présentés dans le tableau 3.8, montrent que les performances des 16 algorithmes proposés dépassent celles du Standard PSO2006 et du Standard BPSO en termes de meilleures solutions trouvées (Best) et de moyennes de solutions trouvées (Avg).

3.11. Conclusion

L'optimisation par essaim de particules (PSO) est une jeune métaheuristique qui a été créée en 1995 par Kennedy et Eberhart. C'est une métaheuristique à base de population de solution inspirée par une analogie avec l'éthologie. La simplicité de mise en œuvre et

l'efficacité de la méthode d'optimisation par essaim de particules ont suscité l'intérêt de plusieurs communautés de chercheurs qui ont mené des études d'optimisation et d'application de cette métaheuristique pour la résolution de plusieurs problèmes d'optimisation de différents types et complexités.

Plusieurs versions de l'algorithme original du PSO ont été proposées dans la littérature afin d'améliorer sa performance. Dans cette contribution, nous nous sommes inspirés de quelques travaux et applications de l'algorithme PSO présentés dans la littérature, et nous avons proposé 16 versions de l'algorithme PSO avec différentes équations de mise à jour des vitesses et des positions des particules. Nous avons regroupé les algorithmes proposés en quatre classes: dans la première classe, nous avons adapté et utilisé la version du PSO avec facteur d'inertie. Ensuite, nous avons proposé et utilisé un nouveau facteur d'accélération F pour contrôler la mise à jour des positions des particules. Nous avons appliqué le nouveau facteur d'accélération F sur les algorithmes que nous avons proposés dans la première classe pour donner naissance aux algorithmes de la deuxième classe. Dans la troisième classe, nous avons adapté et utilisé la version de l'algorithme PSO avec facteur de constriction. En fait, nous avons remarqué que peu de travaux présentés dans la littérature utilisent cette version. Dans la quatrième classe, nous avons utilisé le facteur d'accélération F pour la mise à jour des positions des particules ainsi que le facteur de constriction pour la mise à jour des vitesses des particules.

Afin de tester la performance des algorithmes proposés, nous les avons utilisés pour la résolution du problème du sac à dos. Notre objectif derrière le choix du problème du sac à dos est double. Premièrement: sa difficulté et son importance. En effet, c'est un des problèmes d'optimisation académiques combinatoires NP-Difficiles, qui a de nombreuses applications pratiques. Deuxièmement, montrer que les algorithmes proposés peuvent être utilisés pour la résolution de problèmes d'optimisation binaires. En outre, la recherche que nous avons réalisée sur les différentes applications du PSO pour la résolution de différents problèmes d'optimisation, nous a mené à constater que le problème du sac à dos (avec sa version unidimensionnelle) n'a pas été beaucoup traité en utilisant l'algorithme du PSO. Notamment, avec la version du PSO avec facteur de constriction K . Selon notre connaissance, aucune application de la version de l'algorithme PSO avec facteur de constriction n'a été proposée dans la littérature jusqu'à Décembre 2010. La caractéristique avantageuse des algorithmes proposés par rapport au standard PSO (qui est dédié aux problèmes continus) et même au Standard BPSO (qui est dédié aux problèmes discrets/binaires) c'est que nos algorithmes

peuvent facilement être utilisés dans la résolution des problèmes de différents types (continu, discret ou binaire) par le choix approprié du type des solutions et de l'interprétation des opérateurs des différentes équations.

Les résultats expérimentaux et l'étude comparative réalisée entre les 16 algorithmes proposés nous montrent des améliorations des résultats avec l'application du nouveau facteur d'accélération F pour la mise à jour des positions des particules et l'application du facteur de constriction K pour la mise à jour des vitesses des particules. En outre, la comparaison de la totalité des 16 algorithmes proposés avec le Standard PSO2006 et le Standard BPSO nous montre que la performance des algorithmes proposés dépasse celles du Standard PSO2006 et du Standard BPSO en termes de meilleures solutions et de moyennes des valeurs des solutions trouvées. Les algorithmes proposés ont fait l'objet des articles suivants:

1. Amira Gherboudj et Salim Chikhi. Algorithme d'OEPB pour Résoudre le Problème du Sac à Dos. Proceedings of the first International Conference on Information Systems and Technologies. pp. 460-466. ISBN: 978-9931-9004-0-5. ICIST 2011.
2. Amira Gherboudj and Salim Chikhi. BPSO Algorithms for Knapsack Problem. A. Özcan, J. Zizka, and D. Nagamalai (Eds.): WiMo/CoNeCo 2011, CCIS 162, pp. 217-227. ISSN 1865-0929. Springer, 2011.

Dans le chapitre suivant, nous présentons notre deuxième contribution dans laquelle nous avons proposé un nouvel algorithme hybride que nous avons nommé MHPSO et que nous avons utilisé pour la résolution de deux problèmes d'optimisation académiques NP-Difficiles: le problème du sac à dos multidimensionnel et le problème du voyageur de commerce.

CHAPITRE 4

Un algorithme hybride basé sur l'optimisation par essaim de particules et l'opération du croisement

Sommaire

4.1	Introduction.....	117
4.2	Le PSO et l'hybridation	118
4.3	L'algorithme proposé	128
4.4	Application du MHPSO sur le problème du sac à dos multidimensionnel.....	130
4.5	Application du MHPSO sur le problème du voyageur de commerce.....	143
4.6	Conclusion.....	151

4.1. Introduction

Les problèmes d'optimisation peuvent être classés en deux classes principales: des problèmes continus et des problèmes discrets. Dans la classe des problèmes d'optimisation continus, les solutions sont représentées par un ensemble de nombres réels. Tandis que, dans la classe des problèmes d'optimisation discrets, les solutions sont représentées par un ensemble de nombres entiers. En outre, il existe une sous classe de problèmes d'optimisation discrets qui englobe les problèmes d'optimisation binaires. Dans cette sous classe, les solutions sont représentées par un ensemble de bits.

De nombreux algorithmes ont été proposés dans la littérature et utilisés pour résoudre différents problèmes d'optimisation. Nous citons à titre d'exemple: l'algorithme génétique, le recuit simulé, la recherche tabou...etc. Le choix de l'algorithme et de son paramétrage joue un rôle primordial dans la détermination de la qualité des résultats. Malheureusement, il existe très peu de règles universelles permettant de connaître le paramétrage idéal et généralement, seules l'expertise et l'expérience de l'utilisateur permettront de faire un choix [Mathieu, 2008].

L'optimisation par essaim de particules (PSO: Particle Swarm Optimisation) est une jeune métaheuristique. Elle a prouvé sa simplicité, son efficacité et sa convergence très rapide [Gherboudj et Chikhi, 2011]. Cependant, l'algorithme original du PSO n'opère que dans des espaces continus. En outre, la sélection et l'adaptation du nombre important des paramètres de l'algorithme PSO (la taille de l'essaim, le coefficient d'inertie ω , les coefficients d'accélération c_1 et c_2 ...) jouent un rôle primordial dans l'efficacité et la performance de l'algorithme. D'autre part, le PSO peut être facilement tombé dans la vallée de l'optimum local si la meilleure position globale et la meilleure position locale sont égales à la position de la particule durant plusieurs itérations [Olamaei et Niknam, 2008].

Dans le but de bénéficier des avantages de l'algorithme d'optimisation par essaim de particules (simplicité, efficacité et rapidité), nous nous sommes inspirés de quelques principes du PSO et du croisement de l'algorithme génétique qui permet une bonne exploitation de l'espace de recherche et nous avons proposé un nouvel algorithme hybride que nous avons nommé MHPSO (Modified Hybrid Particle Swarm Optimisation). A l'opposé de l'algorithme standard du PSO, l'algorithme proposé peut être utilisé pour résoudre tous types de problèmes d'optimisation (continus, discrets, binaires) par le choix approprié du type des solutions manipulées. En outre, l'algorithme proposé se caractérise par un bon équilibre entre l'exploitation et l'exploration de l'espace de recherche et aussi il nécessite peu de paramètres à

initialiser contrairement à l'algorithme PSO. En fait, la taille de la population est le seul paramètre mis en jeu, ce qui affranchit l'utilisateur de la phase du choix du grand nombre des paramètres de l'algorithme.

4.2. Le PSO et l'hybridation

Le théorème « No free lunch » qui a été introduit en 1997 par David H. Wolpert et William G. Macready [Wolpert et Macready, 1997] stipule qu'aucune méthode n'est systématique (efficace 100%) pour la résolution de tous les problèmes d'optimisation. En effet, chaque méthode a ses avantages et ses inconvénients (ou lacunes). Plusieurs communautés de chercheurs ont envisagé la combinaison des méthodes de résolution des problèmes d'optimisation, afin de tirer profit des avantages de chaque méthode et de combler certaines de ses lacunes. Par conséquent, ils ont donné naissance à une nouvelle classe de méthodes de résolution de problèmes d'optimisation : c'est la classe des méthodes hybrides. L'objectif de cette section est de présenter quelques travaux qui ont été menés dans le but d'améliorer la performance de la méthode d'optimisation par essaim de particulaire, par l'hybridation de cette dernière avec d'autres méthodes.

4.2.1. Le PSO et l'algorithme génétique

Dans le but d'aboutir à un bon équilibre entre l'intensification et la diversification, Lope et Coelho ont proposé dans leur article [Lope et Coelho, 2005] d'hybrider l'algorithme PSO avec une méthode de recherche locale afin d'améliorer la recherche localement, comme ils ont utilisé les concepts de l'algorithme génétique pour mieux explorer l'espace de recherche. L'algorithme proposé a été utilisé pour la résolution du problème du voyageur de commerce (TSP). Il a été testé sur des instances de 76 à 2103 villes. Selon les auteurs du papier, les résultats obtenus encouragent la recherche et l'amélioration des modèles hybrides pour la résolution des problèmes d'optimisation combinatoires difficiles.

Lian et ses collègues ont songé à utiliser l'algorithme PSO pour résoudre des problèmes d'optimisation discrets. Ils ont proposé [Lian et al, 2006] une nouvelle version de l'algorithme PSO en intégrant les opérateurs de croisement et de mutation dans les étapes de la version standard de l'algorithme PSO. L'algorithme proposé est nommé NPSO, il a été utilisé dans la résolution du problème discret d'ordonnancement Flow Shop. La performance de l'algorithme NPSO a été comparée avec celle de l'algorithme génétique. Les résultats expérimentaux prouvent que le NPSO est plus performant que se soit en termes du temps de calcul nécessaire pour la recherche ou en termes de la qualité de la solution trouvée.

De même et dans un but de soutenir les diagnostics médicaux dans l'amélioration de la classification du cancer, Li et ses collègues ont proposé en 2008 [Li et al, 2008] un algorithme PSO hybridé avec l'algorithme génétique pour la sélection des gènes. Les résultats expérimentaux obtenus avec trois Benchmarks, montrent l'efficacité et la précision de la classification établie par l'application de l'algorithme proposé.

Une autre hybridation de l'algorithme PSO avec l'algorithme génétique est proposée par Liu Zhiming et ses collègues dans leur papier publié en 2008 [Zhiming et al, 2008]. L'algorithme proposé est nommé MGPSO. Il a été utilisé pour résoudre les problèmes d'optimisation avec contraintes (constrained optimisation problems). L'algorithme MGPSO a été testé sur des fonctions benchmarks comme il a été comparé avec l'algorithme original de l'optimisation par essaim de particules et un autre algorithme évolutionnaire. Les résultats obtenus montrent la robustesse et l'efficacité de l'algorithme proposé.

En 2008, Jeff Achtnig [Achnig, 2008] a proposé une nouvelle implémentation de l'algorithme proposé par Clerc et Kennedy en 2002 [Clerc et Kennedy, 2002]. Achtnig a combiné la version de l'algorithme PSO avec coefficient de constriction K avec un opérateur de mutation dans le but de maintenir la diversité de la population. A l'opposé de la version proposée par Clerc, dans l'algorithme proposé par Achtnig la valeur du coefficient K est dynamique. En fait, Achtnig a proposé de générer la valeur du facteur de constriction K à partir de l'intervalle [0.2, 0.9] à chaque itération de l'algorithme.

Dans la même année (i.e. en 2008), Zhang et ses collègues ont proposé [Zhang et al, 2008] une hybridation de l'algorithme PSO avec l'opérateur de mutation inspiré de l'algorithme génétique. L'algorithme proposé a été utilisé pour résoudre le problème du regroupement spatial avec contraintes (Clustering with Obstacles Constraints).

4.2.2. Le PSO et l'optimisation par colonies de fourmis

En 2007, Shelokar et ses collègues ont proposé une hybridation entre l'algorithme d'optimisation par essaim de particules et l'algorithme de colonies de fourmis. L'algorithme proposé est nommé PSACO, il a été utilisé pour l'optimisation des fonctions multimodales continues. Dans le PSACO, l'algorithme de colonies de fourmis a été utilisé en tant que méthode de recherche locale permettant la mise à jour des positions des particules. Les résultats des tests et de comparaison de PSACO avec d'autres algorithmes, prouvent sa performance et son efficacité. Il est à noter qu'une version antérieure de cette hybridation a été proposée en 2004 dans [Shelokar et al, 2004].

Dans la même année (i.e. 2007), Zhang et ses collègues ont proposé une hybridation de la version de l'algorithme PSO proposée par Shi et Eberhart en 1998 avec l'optimisation par colonies de fourmis [Zhang et al, 2007]. L'hybridation proposée est nommée PKSCOC. Elle a été utilisée pour résoudre le problème du regroupement spatial (Spatial clustering). Les résultats expérimentaux ainsi que la comparaison de l'algorithme PKSCOC avec d'autres algorithmes ont prouvé la performance de l'algorithme proposé.

En 2009, Zang et ses collègues ont proposé dans [Y.Zang et al, 2009] une hybridation fondée sur l'optimisation par essaim de particules et l'optimisation par colonies de fourmis. L'algorithme PSO a été utilisé pour optimiser les paramètres de l'algorithme de colonies de fourmis. Les résultats expérimentaux montrent que cette hybridation améliore la performance de l'algorithme de colonies de fourmis. En outre, sa performance dépasse celles des algorithmes existants pour la résolution du problème du voyageur de commerce.

Restant en 2009, Serapiao et Mendes ont proposé d'intégrer le principe de l'optimisation par colonies de fourmis dans les étapes de l'algorithme PSO afin d'augmenter son efficacité. L'algorithme proposé [Serapiao et Mendes, 2009] a été utilisé pour classer automatiquement les étapes du processus du forage de puits. La méthode proposée a deux applications: la première consiste à rechercher et analyser les événements non prévus dans le sens d'analyser chaque puits et de déterminer le temps nécessaire pour chaque phase du forage. La deuxième application consiste à mettre en œuvre un système informatique pour produire en ligne un compte rendu des étapes exécutées dans la plateforme. L'algorithme proposé est nommé PSO/ACO2, il implémente les équations proposées dans [Clerc et Kennedy, 2002]. Des études comparatives de PSO/ACO2 avec d'autres méthodes indiquent qu'il est moins performant.

D'autres hybridations de l'algorithme PSO et l'algorithme de colonies de fourmis ont été proposées dans le cadre de la bioinformatique dans [Holden et Freitas, 2005] et [Holden et Freitas, 2006], pour la classification hiérarchique fonctionnelle des données biologiques (enzymes et les récepteurs couplés des G-protéines, respectivement).

4.2.3. Le PSO et le recuit simulé

Une intégration de la méthode du recuit simulé dans les étapes de l'algorithme PSO a été proposée par Jiang et ses collègues dans leur papier publié en 2004 [Jiang et al, 2004]. L'algorithme développé permet de réduire le problème des particules piégées par des optimums locaux. Les résultats de test montrent que l'algorithme proposé est plus performant que l'algorithme standard de l'optimisation par essaim de particules.

En 2006, Bo Liu et ses collègues ont proposé dans leur papier [Liu et al, 2005] une hybridation entre l'algorithme PSO et l'algorithme du recuit simulé. L'algorithme proposé est nommé HPSO. Le HPSO a été utilisé pour résoudre un problème d'optimisation combinatoire NP-difficile qui exige un processus de production sans interruption entre des machines consécutives, il s'agit du problème d'ordonnancement flow-shop sans attente (No-wait flow shop problem). Les résultats obtenus avec des benchmarks ainsi que des comparaisons avec d'autres algorithmes, ont prouvé l'efficacité de l'algorithme proposé.

Chen et ses collègues ont proposé dans [Chen et al, 2006] une hybridation de l'algorithme PSO quantique discret proposé en 2004 par Yang et ses collègues dans [Yang et al, 2004] avec l'algorithme du recuit simulé afin de résoudre le problème NP-Difficile de tournée de véhicules avec capacité (CVRP: Capacitated Vehicle Routing Problem). L'algorithme proposé a montré son efficacité dans la résolution du problème CVRP et dans la résolution les problèmes de grandes tailles.

De leur part, Liu et ses collègues [Liu et al, 2006] ont proposé une hybridation entre le PSO quantique (QPSO) et le mécanisme du recuit simulé (SA: Simulated Annealing). L'intégration du mécanisme de SA a permis d'augmenter la capacité du QPSO d'échapper au piège de l'optimum local.

Le regroupement de données consiste à classer les données en groupes selon leurs degrés de similitudes, deux types de regroupement de données sont distingués: le regroupement hiérarchique et le regroupement partitionné. En 2008, Nikman et ses collègues ont proposé [Niknam et al, 2008] un algorithme d'optimisation par essaim de particules hybridé avec l'algorithme du recuit simulé pour la résolution du problème du regroupement de données partitionné où l'ensemble de données est regroupé en sous groupes selon certaines mesures de similarité. L'algorithme proposé est nommé PSO-SA. Il a été comparé avec l'algorithme original du PSO, l'algorithme du recuit simulé et la méthode la plus connue et utilisée dans le regroupement partitionné: «K-means». Les résultats de test de l'algorithme PSO-SA montrent son efficacité dans le regroupement optimal des données.

En 2009, Chen et ses collègues [Chen et al, 2009] ont présenté une hybridation entre le BPSO et la méthode du recuit simulé pour résoudre le problème d'ordonnancement de grille où un échange d'informations entre différentes procédures dans un système de grille distribué doit être établi. Les résultats des tests montrent l'efficacité de l'hybridation proposée.

4.2.4. Le PSO et la recherche tabou

En 2005, Wanhui Wen et Guangyuan Liu ont proposé une hybridation entre l'algorithme PSO et la stratégie de double-tabou [Wen et Liu, 2005]. L'hybridation proposée était une tentative pour la résolution efficace des problèmes d'optimisation combinatoires NP-Difficiles. En effet, les auteurs ont utilisé l'algorithme proposé pour la résolution du problème du voyageur de commerce. Les tests réalisés ont donné de bons résultats.

En 2007, Nakano et ses collègues [Nakano et al, 2007] ont tenté de tirer profit de l'avantage de la recherche tabou qui permet d'échapper de l'optimum local et ils ont proposé un algorithme d'optimisation par essaim particulaire basé sur le principe de la recherche tabou. Une autre version de cette hybridation est exposée dans [Nakano et al, 2010].

Dans la même année (i.e. 2007), Shen et ses collègues ont proposé une hybridation entre l'algorithme PSO et la recherche tabou [Shen et al, 2007]. L'algorithme proposé est nommé HPSOTS, il a été utilisé dans la sélection des gènes pour la classification des tumeurs. L'utilisation de la recherche tabou a permis à l'algorithme d'échapper au piège de l'optimum local. Par conséquent, l'algorithme proposé a montré de bonnes performances par rapport à d'autres algorithmes avec lesquels il a été comparé, notamment pour la sélection des gènes et l'exploration des données de grandes dimensions.

4.2.5. Le PSO et la recherche à voisinage variable

H. Liu et A. Abraham ont proposé en 2007 [Liu et Abraham, 2006] une hybridation entre l'algorithme PSO et la recherche à voisinage variable. L'algorithme proposé est nommé VNPSO, il a été appliqué dans la résolution du problème d'affectation quadratique (QAP). Les résultats des tests montrent que le VNPSO peut résoudre le QAP efficacement.

De leur part, Sevkli et ses collègues ont proposé en 2008 [Sevkli et Sevilgen, 2008] une hybridation entre le PSO et la méthode de la recherche à voisinage variable réduite notée: RVNS. Les auteurs ont utilisé la méthode RVNS en tant que méthode de recherche locale permettant une bonne exploitation des solutions afin d'empêcher la convergence prématurée de l'algorithme PSO.

4.2.6. Le PSO et le système immunitaire

Xiaorong Pu et ses collègues se sont inspirés de la capacité de reconnaissance des visages dans le système visuel humain, et ils ont proposé un nouvel algorithme d'optimisation par essaim de particules binaire à base du système immunitaire [Pu et al, 2007]. L'algorithme

proposé a été utilisé pour résoudre le problème de la reconnaissance du visage en tenant compte des caractéristiques globales et partielles du visage ainsi que l'éclairage et l'expression de ce dernier. Les résultats expérimentaux montrent que l'algorithme proposé est plus performant que d'autres algorithmes existants.

De leur part, Chang et ses collègues ont proposé en 2009 [Chang et al, 2009] un algorithme d'optimisation par essaim particulaire fondé sur l'algorithme inspiré du système immunitaire. Les résultats des tests ont montré que l'algorithme proposé est faisable pour la résolution des problèmes des systèmes complexes non linéaires.

4.2.7. Le PSO et la méthode simplexe

Une hybridation entre le PSO et la méthode simplexe de Nelder-Mead a été développée en 2004 par Zahara et ses collègues [Zahara et al, 2004]. Selon les auteurs du papier, l'objectif principal de l'hybridation proposée est de combler les lacunes de la méthode Otsu [Otsu, 1979] dans le domaine de la segmentation des images. Précisément, dans le cas du seuillage multi-niveau. Les auteurs ont proposé deux versions de la méthode qu'ils ont nommé NM-PSO. La première version est nommée NM-PSO-Otsu. NM-PSO-Otsu est une hybridation entre NM-PSO et la méthode Otsu. La deuxième version est nommée NM-PSO-curve. NM-PSO-curve est une combinaison de NM-PSO et la courbe Gaussienne. Les algorithmes ont été testés sur des images de taille 256*256 pixel avec 256 niveaux de gris prises sous un éclairage de chambre naturel. Les résultats des tests montrent que NM-PSO-Otsu améliore la performance de la méthode Otsu dans le cas du seuillage multi niveau sans dégrader la qualité de la segmentation d'image. Alors que NM-PSO-curve offre de bonnes qualités dans le sens de visualisation, de la taille de l'objet et du contraste. Particulièrement, lorsque l'image a une structure complexe ou dans le cas où le contraste entre l'objet et le fond est vaste.

D'autre part, Chen-Chien Hsu et ses collègues ont proposé en 2008 [Hsu et al, 2008] une hybridation entre l'algorithme PSO et une méthode améliorée de la recherche à base du simplexe de Nelder et Mead (SNM) [Nelder et Mead, 1965]. Le but de cette hybridation est de proposer un contrôleur numérique optimal pour les systèmes avec intervalles incertaines comme les véhicules de vols, les moteurs électriques et les robots où l'incertitude est causée par la variation des paramètres, du bruit des capteurs...etc. Dans le même contexte, Chen-Chien Hsu et ses collègues ont exposé en 2010 [Hsu et al, 2010] une autre application de l'hybridation entre l'algorithme PSO et la SNM.

4.2.8. Le PSO et la méthode GRASP

Marinakis et ses collègues ont proposé en 2008 [Marinakis et al, 2008] une hybridation de l'algorithme PSO avec facteur de constriction proposé par Clerc [Clerc, 1999], [Clerc et Kennedy, 2002] et la méthode GRASP (Greedy Randomized Adaptive Search Procedure). Ils ont utilisé une structure multi essaim composée de 5 essaims chacun est composé de 20 particules afin de permettre un bon équilibre entre l'exploitation des solutions et l'exploration de l'espace de recherche au cours de la recherche. L'algorithme proposé est nommé MSCPSO-GRASP, il a été utilisé pour le regroupement (clustering) optimal de N objets en K classes. MSCPSO-GRASP utilise l'algorithme MSCPSO (Multi-Swarm Constriction Particale Swarm Optimization) dans la phase de la sélection et la méthode GRASP dans la phase du regroupement (classement). L'hybridation proposée a été testée sur plusieurs benchmarks. Comme elle a été comparée avec d'autres métaheuristiques à base de population de solutions. Les résultats obtenus prouvent que l'algorithme proposé est plus performant que d'autres algorithmes, notamment pour le problème du regroupement.

4.2.9. Autres hybridations

Thiago R. Machado et Heitor S. Lopes ont proposé en 2005 [Machado et Lopes, 2005] une combinaison entre l'algorithme PSO, l'algorithme génétique et la descente de gradient rapide (Fast local search *ou* fast hill climbing) [Voudouris et Tsang ,1999]. L'hybridation proposée a été utilisée pour résoudre une variante spéciale du problème de voyageur de commerce: le voyageur de commerce symétrique aveugle (Symetric blind traveling Salesman Problem).

En 2006, Bo liu et ses collègues ont proposé une hybridation [Liu. B et al, 2006] entre l'algorithme PSO, l'algorithme mimétique (AM) et une méthode de recherche locale fondée sur le recuit simulé. L'objectif de cette hybridation est de maintenir un bon équilibre entre l'exploitation des solutions et l'exploration de l'espace de recherche. L'algorithme proposé a été utilisé dans la résolution du problème de voyageur de commerce: un des problèmes appartenant à la classe NP-difficile. Les résultats expérimentaux montrent la performance de l'algorithme proposé en termes de robustesse et de qualité de la recherche.

De leur part, Habibi et ses collègues ont proposé une hybridation [Habibi et al, 2006] entre l'algorithme PSO, la recherche par colonies de fourmis et le recuit simulé. L'algorithme proposé a été utilisé pour résoudre le problème du voyageur de commerce. Dans cette hybridation l'optimisation par colonies de fourmis est utilisée pour remplacer la meilleure

position de chaque particule trouvée par le PSO. Tandis que la méthode à base du recuit simulé est utilisée pour contrôler l'exploitation de la meilleure particule du groupe.

En 2007, Xianjun Shen et ses collègues ont proposé [Shen. X et al, 2007] un algorithme à base de l'optimisation par essaim de particules nommé SA-GPSO. Au cours de la recherche, l'algorithme SA-GPSO fait appel à la méthode du recuit simulé et aussi aux opérateurs de l'algorithme génétique: le croisement et la mutation. Le SA-GPSO a été appliqué dans la résolution du problème du découpage unidimensionnel multi-spécifications du stock (Multi-specification one-dimensional cutting stock problem).

Chen et Wang ont proposé une hybridation entre la version binaire de l'algorithme d'optimisation par essaim de particule et une méthode de recherche locale [Chen et Wang, 2007]. L'hybridation proposée a été utilisée pour la résolution du problème d'assignation de fréquence du spectre déterminé (Fixed-Spectrum Frequency Assignment), où l'objectif est de minimiser le coût de l'interférence provenant d'une solution. L'algorithme proposé a été testé sur huit problèmes benchmarks connus. Les résultats obtenus prouvent son efficacité.

Jens Czogalla et Andreas Fink ont proposé [Czogalla et Fink, 2008] une hybridation entre l'algorithme PSO, la recherche locale à voisinage variable et l'opérateur du croisement de l'algorithme génétique. L'hybridation proposée a été utilisée pour la résolution du problème d'ordonnancement flow shop. De bons résultats ont été trouvés.

Lianguo Wang et ses collègues ont proposé une nouvelle variante de l'algorithme PSO, dans laquelle ils ont intégré des concepts de l'algorithme génétique et le principe du système multi-agents. L'algorithme proposé est nommé MAGPSO [Wang et al, 2008]. Selon ses créateurs, l'algorithme MAGPSO maintient efficacement la diversité de l'essaim, augmente la précision de l'algorithme PSO et limite d'une manière efficace l'inconvénient de la convergence prématurée. Les résultats de test de l'algorithme proposé ont été comparés avec les résultats obtenus par d'autres méthodes d'optimisation. L'étude expérimentale montre que l'algorithme MAGPSO est très performant pour l'optimisation des fonctions de grandes dimensions.

Zhou et ses collègues ont proposé une hybridation entre l'algorithme PSO, l'algorithme génétique et le principe des systèmes multi agents pour résoudre les problèmes d'optimisation combinatoires [Zhou et al, 2008]. Dans l'algorithme proposé, la particule est remplacée par un agent qui représente une solution au problème traité. Les agents œuvrent en coopération afin d'aboutir à une solution optimale dans un bref délai. D'après Zhou et ses collègues, les tests

expérimentaux de l'algorithme proposé montrent qu'il est plus performant que d'autres algorithmes.

Min-Rong Chen et ses collègues [Chen. M et al, 2009] ont proposé une hybridation entre l'algorithme PSO et l'heuristique de recherche locale EO (External Optimization). L'introduction de l'EO dans le corps de l'algorithme PSO lui a permis de remédier au problème de la convergence prématurée. L'algorithme proposé est nommé PSO-EO. Il a été testé sur des fonctions benchmarks uni-modales/multimodales. Les résultats des tests montrent que la performance de l'algorithme PSO-EO dépasse celle des autres métaheuristiques utilisées dans la comparaison.

Xingjuan Cai et YingTan ont proposé dans [Cai et Tan, 2009] une hybridation entre l'algorithme PSO et la programmation génétique pour la récupération automatique de la structure des réseaux des gènes des plantes.

Une hybridation entre le système multi agents et l'algorithme PSO a été présentée par Rajesh Kumar et ses collègues [Kumar et al, 2009]. L'algorithme proposé a été testé sur plusieurs problèmes d'optimisation. Les résultats prouvent sa performance en termes de qualité des solutions et de précision de la solution.

Li et al ont publié en 2010 un article [Li et al, 2010] dont ils ont proposé une hybridation entre le PSO, la méthode simplexe non linéaire (MSN) et la méthode de recherche tabou, afin d'augmenter la vitesse de convergence de l'algorithme PSO. L'objectif d'intégration de la MSN est de renforcer la capacité de recherche locale du PSO. Alors que, l'objectif de l'utilisation du RT est de permettre aux particules de s'échapper du piège de la vallée de l'optimum local. Le processus de fonctionnement ainsi que son efficacité ont été testé en utilisant quelques fonctions benchmarks.

Valdez. F et ses collègues ont proposé une hybridation entre l'algorithme PSO, l'algorithme génétique et le principe de la logique floue (LF) pour l'adaptation des paramètres et l'intégration des résultats [Valdez et al, 2010]. L'hybridation proposée a été utilisée pour l'optimisation de l'architecture du réseau neuronal. Des versions antérieures publiées en 2008 et 2009 [Valdez et al, 2008] [Valdez et al, 2009] ont été proposées pour optimiser les fonctions mathématiques et elles ont donné de bons résultats.

L'hybridation	L'année	Les Auteurs	Le problème traité
<ul style="list-style-type: none"> ○ PSO / GA /VNS ○ PSO / GA /VNS ○ PSO / GA ○ PSO/ GA/ FL ○ PSO/ GA ○ PSO/ Mutation 	<ul style="list-style-type: none"> ○ 2005 ○ 2005 ○ 2008 ○ 2010 ○ 2007 ○ 2008 	<ul style="list-style-type: none"> ○ Lope et Coelho ○ Machado et Lopes ○ Li et al ○ Valdez et al ○ Chen et Wang ○ Zhang et al 	<ul style="list-style-type: none"> ○ Le voyageur de commerce ○ Le voyageur de commerce ○ Traitement du cancer ○ Optimisation de l'architecture du réseau neuronal ○ Engagement d'unité ○ le regroupement de données spatiales avec contraintes
<ul style="list-style-type: none"> ○ ACO/ PSO 	<ul style="list-style-type: none"> ○ 2009 ○ 2005 ○ 2006 ○ 2007 ○ 2007 ○ 2009 	<ul style="list-style-type: none"> ○ Zhang et al ○ Holden et Freitas ○ Holden et Freitas ○ Zhang et al ○ Shelokar et al ○ Serapiao et Mendes 	<ul style="list-style-type: none"> ○ Le voyageur de commerce ○ la classification hiérarchique fonctionnelle d'enzymes ○ la classification hiérarchique fonctionnelle des récepteurs couplés des G-protéines ○ le regroupement de données spatiales ○ l'optimisation des fonctions multimodales continues ○ Classement des étapes de l'exploitation du forage de puits
<ul style="list-style-type: none"> ○ BPSO/ SA 	<ul style="list-style-type: none"> ○ 2009 ○ 2008 ○ 2006 	<ul style="list-style-type: none"> ○ Chen et al ○ Niknam et al ○ Chen et al 	<ul style="list-style-type: none"> ○ L'ordonnancement de grille ○ le regroupement partitionné de données ○ Tournée de véhicules avec capacité
<ul style="list-style-type: none"> ○ PSO/ TS 	<ul style="list-style-type: none"> ○ 2005 ○ 2007 	<ul style="list-style-type: none"> ○ Wen et Liu ○ Shen et al 	<ul style="list-style-type: none"> ○ Le voyageur de commerce ○ la sélection des gènes pour la classification des tumeurs
<ul style="list-style-type: none"> ○ PSO/ VNS 	<ul style="list-style-type: none"> ○ 2006 	<ul style="list-style-type: none"> ○ Liu et Abraham 	<ul style="list-style-type: none"> ○ Affectation quadratique
<ul style="list-style-type: none"> ○ PSO/IS 	<ul style="list-style-type: none"> ○ 2007 	<ul style="list-style-type: none"> ○ Pu et al 	<ul style="list-style-type: none"> ○ La reconnaissance du visage
<ul style="list-style-type: none"> ○ PSO/NMS 	<ul style="list-style-type: none"> ○ 2004 ○ 2008 	<ul style="list-style-type: none"> ○ Zahara et al ○ Hsu et al 	<ul style="list-style-type: none"> ○ Segmentation d'image ○ Le contrôle des systèmes avec intervalles incertains
<ul style="list-style-type: none"> ○ PSO/ GRASP 	<ul style="list-style-type: none"> ○ 2008 	<ul style="list-style-type: none"> ○ Marinakis et al 	<ul style="list-style-type: none"> ○ le regroupement d'objets en classes
<ul style="list-style-type: none"> ○ BPSO/ MAM 	<ul style="list-style-type: none"> ○ 2008 	<ul style="list-style-type: none"> ○ Zhou et al 	<ul style="list-style-type: none"> ○ bipartite subgraphe problem
<ul style="list-style-type: none"> ○ PSO/ MA/ SA ○ PSO/ ACO/ SA ○ PSO/ techniques statistiques ○ PSO/ SA/ GA ○ PSO/ GP ○ PSO/ VNS/ Croisement ○ BPSO/ LS 	<ul style="list-style-type: none"> ○ 2006 ○ 2006 ○ 2004 ○ 2007 ○ 2009 ○ 2008 ○ 2007 	<ul style="list-style-type: none"> ○ Liu.B et al ○ Habibi et al ○ Ko et Lin ○ Shen. X et al ○ Cai et Tan ○ Czogalla et Fink ○ Chen et Wang 	<ul style="list-style-type: none"> ○ Le voyageur de commerce ○ Le voyageur de commerce ○ Analyse de prévisions de gain dans un contexte financier ○ Le découpage unidimensionnel multi-spécifications du stock ○ la récupération automatique de la structure des réseaux des gènes des plantes ○ L'ordonnancement (Flow-shop) ○ Assignment de fréquence du spectre déterminée

Tableau 4.1. Les travaux d'hybridations à base de l'algorithme PSO.

Xumei Zhang et Hanguang Qiu ont proposé dans [Zhang et Qiu, 2010] un algorithme PSO dans lequel ils ont intégré une méthode basée sur le réglage dynamique de la vitesse des particules ainsi qu'une méthode à base d'analyse du regroupement d'essaim avec la méthode K-centres afin de pallier au problème de tomber dans le piège de l'optimum local. L'algorithme proposé a été appliqué pour résoudre le problème du voyageur de commerce.

Dans le tableau 4.1, nous résumons les différents travaux récents d'hybridation de l'algorithme PSO avec d'autres algorithmes.

4.3. L'algorithme proposé

L'optimisation par essaim de particules est une des jeunes métaheuristiques à base de population de solutions. Malgré son jeune âge, elle a prouvé sa simplicité, son efficacité et sa convergence très rapide [Gherboudj et Chikhi, 2011]. En revanche, le choix et l'adaptation du nombre important des paramètres de l'algorithme d'optimisation par essaim de particules (la taille de l'essaim, le coefficient d'inertie ω , les coefficients d'accélération c_1 et c_2 ...) jouent un rôle primordial dans l'efficacité et la performance de l'algorithme. En fait, l'algorithme PSO peut être facilement tombé dans la vallée de l'optimum local si la meilleure position globale et la meilleure position locale sont égales à la position de la particule durant plusieurs itérations [Olamaei et Niknam, 2008].

Dans le but de bénéficier de tous ces avantages et de combler toutes ces lacunes, nous nous sommes inspirés du principe de l'optimisation par essaim de particules, du croisement de l'algorithme génétique qui permet une bonne exploitation de l'espace de recherche. Notre objectif est de proposer un nouvel algorithme hybride basé sur l'optimisation par essaim de particules avec minimum de paramètres et applicable sur tous les types des problèmes d'optimisation (continus, discrets, binaires). Nous avons dénoté notre algorithme par MHPSO : A Modified Hybrid Particle Swarm Optimization Algorithm. Son principe est décrit dans ce qui suit.

L'algorithme MHPSO, lance la recherche par une population de solutions appelée essaim. Cet essaim est composé d'un ensemble de particules, où chacune possède une position dans l'espace de recherche notée par $x_{id} = x_{i1}, x_{i2}, \dots, x_{id}, \dots, x_{iD}$ et se souvient de la meilleure position qu'elle a rencontrée au cours du processus d'amélioration de la qualité de sa position. Chaque position d'une particule représente une solution possible au problème à traiter. Comme n'importe quel algorithme, la première étape dans l'algorithme MHPSO est

d'initialiser quelques paramètres nécessaires pour un bon et efficace déroulement du processus de la recherche. Comparé avec d'autres métaheuristiques à base de population de solutions comme l'algorithme d'optimisation par essaim de particules et l'algorithme génétique, le MHPSO ne nécessite pas trop de paramètres à initialiser. En fait, La taille de la population est le seul paramètre essentiel à initialiser.

Les étapes de l'algorithme proposé (MHPSO) sont résumées comme suit: l'algorithme commence par l'initialisation de la taille de l'essaim et des positions x_{id} aléatoires des particules de l'essaim. Ensuite, il initialise la meilleure position rencontrée par chaque particule (notée par $p_{bestid} = p_{besti1}, p_{besti2}, \dots, p_{bestid}, \dots, p_{bestiD}$) par sa position initiale x_{id} . Puis, en se basant sur la fonction objectif, l'algorithme calcule la qualité (i.e. fitness) de chaque particule pour pouvoir tirer la meilleure position rencontrée par l'essaim (notée par $g_{bestd} = g_{best1}, g_{best2}, \dots, g_{bestd}, \dots, g_{bestD}$). Ensuite, l'algorithme entame l'ensemble des itérations permettant la génération de nouvelles solutions dans le but d'améliorer la qualité de la meilleure solution rencontrée par l'essaim. À chaque itération, une nouvelle position est calculée pour chaque particule de l'essaim en fonction de sa position actuelle, sa meilleure position et la meilleure position rencontrée par l'essaim. En fait, la nouvelle position est la meilleure position trouvée après croisement de la solution actuelle de la particule avec sa meilleure position d'une part et croisement de la solution actuelle de la particule avec la meilleure position trouvée par l'essaim d'autre part, comme le montre l'équation 4.1.

$$x_{id} = \text{Max}[(p_{bestid} \otimes x_{id}), (g_{bestd} \otimes x_{id})] \quad (4.1)$$

Où « \otimes » est un opérateur de croisement, choisi par le programmeur en tenant compte les contraintes du problème à traiter.

Après création des nouvelles solutions, l'algorithme passe à les évaluer (i.e. calculer la fitness $f(x_{id})$ de chaque nouvelle particule) pour pouvoir mettre à jour les valeurs des p_{bestid} et aussi de g_{bestd} , comme le montre les formules 4.2 et 4.3, respectivement.

$$\text{Si } (f(x_{id}) \text{ est meilleur que } f(p_{bestid})) \text{ alors } p_{bestid} = x_{id} \quad (4.2)$$

$$\text{Si } (f(p_{bestid}) \text{ est meilleur que } f(g_{bestd})) \text{ } g_{bestd} = p_{bestid} \quad (4.3)$$

Où: $i=1, \dots, N$ (N représente la taille de l'essaim).

L'ensemble des opérations de chaque itération se répète jusqu'à la satisfaction de la condition d'arrêt. L'algorithme 4.1, présente un schéma général de l'algorithme MHPSO.

Algorithme 4.1. Un schéma général de l'algorithme MHPSO

Début

Initialiser la taille de l'essaim S;

Initialiser aléatoirement les positions des particules x_{id} ;

Pour chaque particule, $p_{bestid} = x_{id}$;

Calculer la fitness de chaque particule;

Calculer g_{bestd} ;

Tant que le critère d'arrêt n'est pas vérifié **faire**

Pour $i=1$ à N **faire**

 Calculer la nouvelle position x_{id} en utilisant l'équation (4.1);

 Calculer la fitness de x_{id} ;

Si ($f(x_{id})$ est meilleur que $f(p_{bestid})$) **alors**

$p_{bestid} = x_{id}$;

Fin pour

 Mettre à jour g_{bestd} ;

Fin Tant que

Retourner g_{bestd} ;

Fin

4.4. Application du MHPSO sur le problème du sac à dos multidimensionnel

4.4.1. Représentation

L'avantage principal de l'algorithme proposé c'est qu'il peut être utilisé pour résoudre tous les types de problèmes d'optimisation (problème d'optimisation continu, discret et discret/binaire) par le choix approprié du type de la population. En effet, puisque le problème du sac à dos multidimensionnel est un problème d'optimisation combinatoire binaire, il sera un bon choix d'initialiser et de représenter la population par des particules binaires. Dans ce but, nous avons utilisé des vecteurs binaires de taille D pour représenter les différentes particules. La représentation de la particule i est comme suit:

$$x_{id} = [x_{i1}, x_{i2}, \dots, x_{id}, \dots, x_{iD}]$$

$$\text{Où } x_{id} = \begin{cases} 1 & \text{Si l'objet est sélectionné} \\ 0 & \text{Sinon} \end{cases}$$

Algorithme 4.2. Algorithme de Croisement

Entrée : Deux particules x_1 et x_2

Sortie : Une particule x_i

Début

Choisir le pas p ;

Choisir deux positions aléatoires c_1 et c_2 à partir de x_1 et x_2 :
 $c_1, c_2 \in \{1, \dots, D-p\}$;

Permuter les éléments de x_1 de c_1 à c_1+p avec ceux de x_2 de c_2 à c_2+p ;

Permuter les éléments de x_1 de c_2 à c_2+p avec ceux de x_2 de c_1 à c_1+p

Calculer la fitness des nouvelles particules et choisir la meilleure entre elles.

Fin

4.4.2. L'opérateur de croisement

L'opérateur du croisement est un des opérateurs de l'algorithme génétique, son rôle est de créer une nouvelle population d'individus. Il consiste à combiner les caractéristiques des deux individus (parents) pour produire un ou deux individus (enfants). Supposons que nous avons deux particules x_1 , x_2 et nous voulons les croiser ensemble afin d'obtenir une nouvelle particule. Nous commençons par l'initialisation du pas p , deux nombres aléatoires c_1 et c_2 . Ensuite, nous suivons les étapes présentées dans l'algorithme 4.2 et expliquées par un exemple dans la figure 4.1. Où l'algorithme 4.2 présente un pseudo code de l'algorithme du croisement utilisé et la figure 4.1 représente un exemple du croisement proposé entre deux particules. Le croisement proposé donne naissance à deux nouveaux enfants. Pour choisir quel enfant représentera la nouvelle particule, nous calculons la fitness $f(x_i)$ de chaque enfant et nous sélectionnons le meilleur enfant.

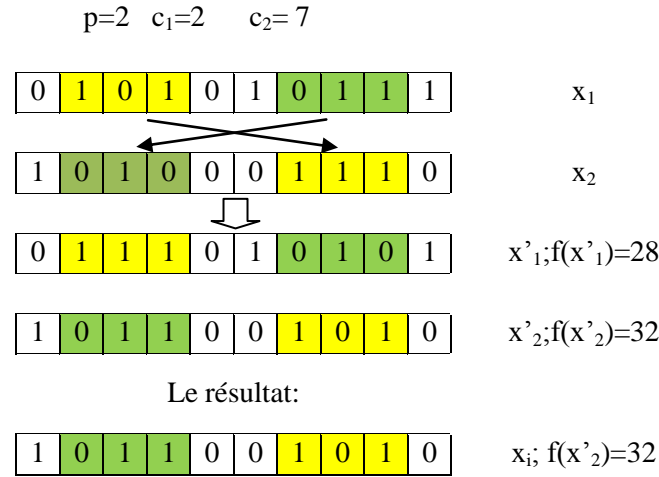


Figure 4.1. Un exemple de croisement de deux particules.

Algorithme 4.3. Algorithme de Réparation d'une Particule (PRA)

Entrée: Vecteur de la solution x

Sortie: Vecteur de la solution réparée x

Début

Calculer $R_j = \sum_{i=1}^n w_{ji}x_i, j=1, \dots, m$; // m est le nombre des dimensions du sac à dos.

Pour ($j=1, \dots, m$) **faire**

Tant que $R_j > C_j$ **faire**

Sélectionner aléatoirement $i \in \{1, \dots, n\}$;

Si $x_i = 1$ **alors**

$x_i = 0$;

Calculer $R_j = \sum_{i=1}^n w_{ji}x_i, j=1, \dots, m$;

Fin si

Fin Tant que

Fin Pour

Fin

4.4.3. L'algorithme de Réparation des Particules (PRA)

Dans le problème du sac à dos multidimensionnel, la solution doit vérifier les m contraintes du sac à dos pour qu'elle soit acceptée comme solution faisable. Si le poids total des objets sélectionnés dans la solution i dépasse la capacité de n'importe qu'elle dimension du sac à dos, la solution sera considérée infaisable et elle ne sera pas acceptée. Pour réparer

une solution i , nous avons proposé un algorithme de réparation de solution que nous avons dénoté par PRA (Particle Repair Algorithm). L'algorithme PRA permet la transformation d'une solution infaisable en une solution faisable. Un pseudo code le l'algorithme PRA est présenté dans l'algorithme 4.3.

4.4.4. L'algorithme "Check and Repair Operator (CRO)"

L'algorithme PRA permet d'améliorer la qualité d'une solution infaisable. Cependant, il n'est pas très efficace avec des instances de grandes tailles. Pour pallier à ce problème, nous avons utilisé l'algorithme « Check and Repair Operator: CRO » [Cotta et Troya, 1998] comme alternative de l'algorithme PRA pour faire face à des instances de grandes tailles. Comme il a été décrit dans [Cotta et Troya, 1998 ; Li et al, 2006], l'algorithme CRO est basé sur deux phases dont les étapes sont présentées dans les algorithmes 4.4 et 4.5 respectivement. Afin de réparer une solution donnée, ces deux phases se basent sur « the profit density » de chaque objet dans chaque dimension du sac à dos qui est calculée comme suit:

$$\delta_{ij} = C_j \cdot p_i / w_{ij} \quad (4.4)$$

Dans la première phase, l'algorithme 4.4 est utilisé pour transformer une solution infaisable en une solution faisable. Dans la deuxième phase, l'algorithme 4.5 est utilisé pour améliorer la qualité de la solution obtenue (i.e. réparée par l'algorithme 4.4) [Li et al, 2006].

Algorithme 4.4. Algorithme de réparation de solution

Début

Calculer $\delta_{ij} = C_j \cdot p_i / w_{ij}$ pour chaque objet, dans chaque dimension;

Calculer la plus petite valeur de $\delta = \min\{C_j \cdot p_j / w_{ij}\}$ pour chaque objet;

Trier les objets selon l'ordre croissant des δ_i ;

Tant que la solution faisable n'est pas rencontrée **faire**

Supprimer l'objet qui a la plus petite δ_i (i.e. changer la valeur du gène de 1 à 0);

Fin Tant que

Fin

Algorithme 4.5. Algorithme d'amélioration de la solution

Début

Calculer $\delta_j = C_j \cdot p_j / w_{ij}$ de chaque objet qui ne fait pas partie des objets du sac à dos;

Calculer la plus petite valeur de $\delta = \min\{C_j \cdot p_j / w_{ij}\}$ pour chaque objet;

Trier les objets selon l'ordre décroissant des δ_i ;

Répéter

Ajouter l'objet de la plus grande valeur de δ_i (i.e. changer la valeur du gène de 0 à 1);

Jusqu'à arriver à une solution infaisable

Fin

4.4.5. Les algorithmes proposés

4.4.5.1. Le premier algorithme (MHPSO1)

Dans le but de créer une nouvelle population, nous avons appliqué l'opérateur de croisement entre la meilleure position p_{bestid} trouvée par la particule i et sa position actuelle x_{id} pour produire une nouvelle particule (nouvel enfant). Ensuite, nous avons appliqué l'opérateur de croisement entre la nouvelle particule et la meilleure position (particule) g_{bestd} trouvée par l'essaim afin de créer une nouvelle particule.

Les étapes de cet algorithme sont résumées dans l'organigramme présenté dans la figure 4.2 et expliquées dans ce qui suit:

Étape 1: Choisir une taille S de l'essaim, initialiser les positions aléatoires des particules et la meilleure position de chaque particule ($p_{bestid} = x_{id}$);

Étape 2: Réparer les solutions infaisables en utilisant l'algorithme PRA, puis calculer leurs fitness en utilisant la fonction objectif.

Étape 3: Trouver la meilleure solution g_{bestd} ;

Étape 4: Calculer la nouvelle position x_{id} de chaque particule en utilisant les équations suivantes:

$$cx_{id} = p_{bestid} \otimes x_{id} \quad (4.5)$$

$$x_{id} = g_{bestd} \otimes cx_{id} \quad (4.6)$$

Où le \otimes est un opérateur de croisement, dont les étapes sont présentées dans l'algorithme 4.2.

Étape 5: Appliquer l'algorithme PRA sur les solutions infaisables et évaluer les particules ;

Étape 6: Mettre à jour les meilleures positions P_{bestid} des particules en fonction de la formule 4.2, et aussi la meilleure position g_{bestd} trouvée par l'essaim en fonction de la formule 4.3.

Étape 7: Arrêter la recherche si la condition d'arrêt est vérifiée et afficher la meilleure solution g_{bestd} . Sinon, retourner à l'étape 4.

4.4.5.2. Le deuxième algorithme (MHPSO2)

Dans cet algorithme, la nouvelle solution est la meilleure position trouvée après croisement de la solution actuelle de la particule avec sa meilleure position d'une part et croisement de la solution actuelle de la particule avec la meilleure position trouvée par l'essaim d'autre part. Après initialisation des différentes positions de particules, l'algorithme passe à leur réparation en utilisant l'algorithme PRA. Une fois la boucle de recherche est lancée, les solutions sont réparées à chaque itération en utilisant soit l'algorithme PRA ou CRO.

Les étapes de cet algorithme sont résumées dans l'organigramme présenté dans la figure 4.3 et expliquées dans ce qui suit:

Étape 1: Choisir une taille S de l'essaim, initialiser les positions aléatoires des particules et la meilleure position de chaque particule ($p_{bestid} = x_{id}$);

Étape 2: Réparer les solutions infaisables en utilisant l'algorithme PRA, puis calculer leurs fitness en utilisant la fonction objectif.

Étape 3: Trouver la meilleure solution g_{bestd} ;

Étape 4: Calculer la nouvelle position x_{id} de chaque particule en utilisant l'équation 4.1

Étape 5: Appliquer l'algorithme PRA ou l'algorithme CRO sur les solutions infaisables et évaluer les particules ;

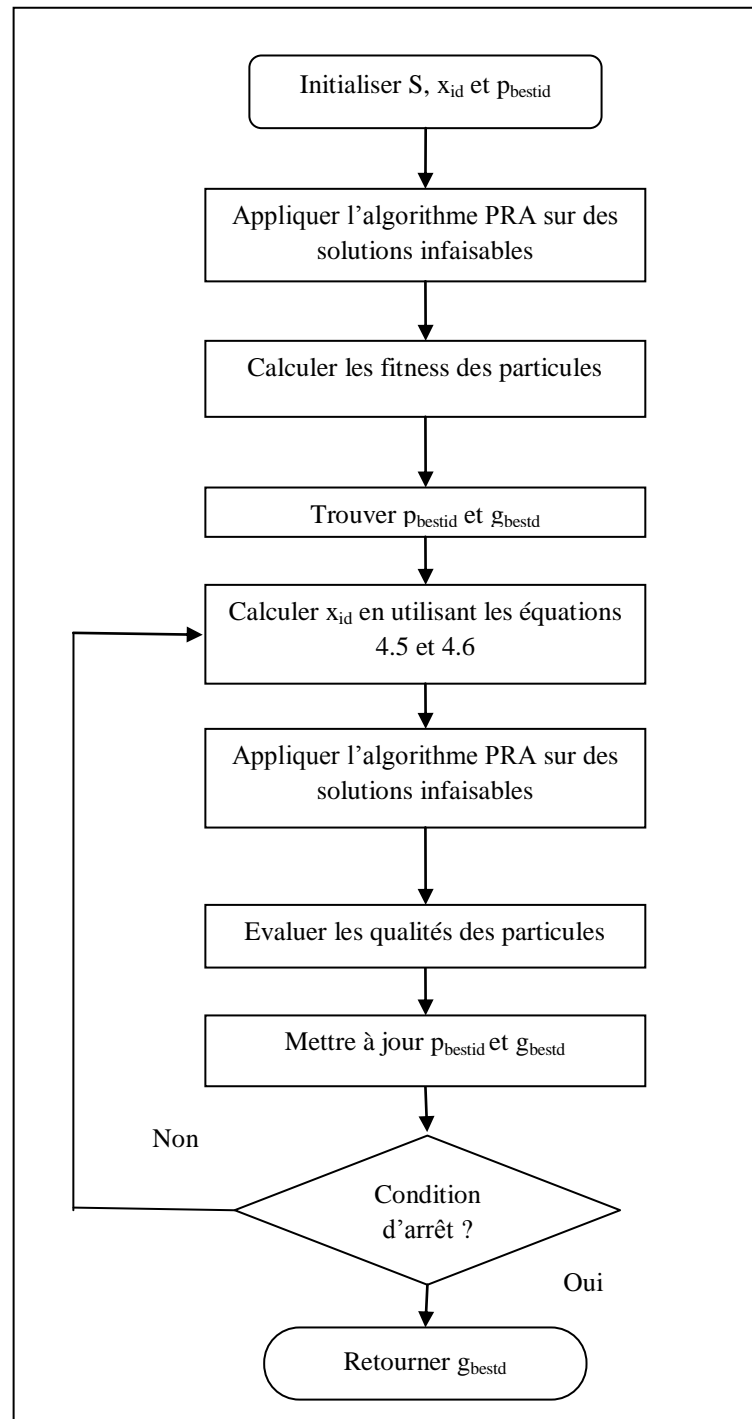


Figure 4.2. Organigramme de l'algorithme MHPSO1.

Étape 6: Mettre à jour les meilleures positions des particules P_{bestid} en fonction de la formule 4.2, et aussi la meilleure position trouvée par l'essaim g_{bestd} en fonction de la formule 4.3.

Étape 7 : Arrêter la recherche si la condition d'arrêt est vérifiée et afficher la meilleure solution g_{bestd} . Sinon, retourner à l'étape 4.

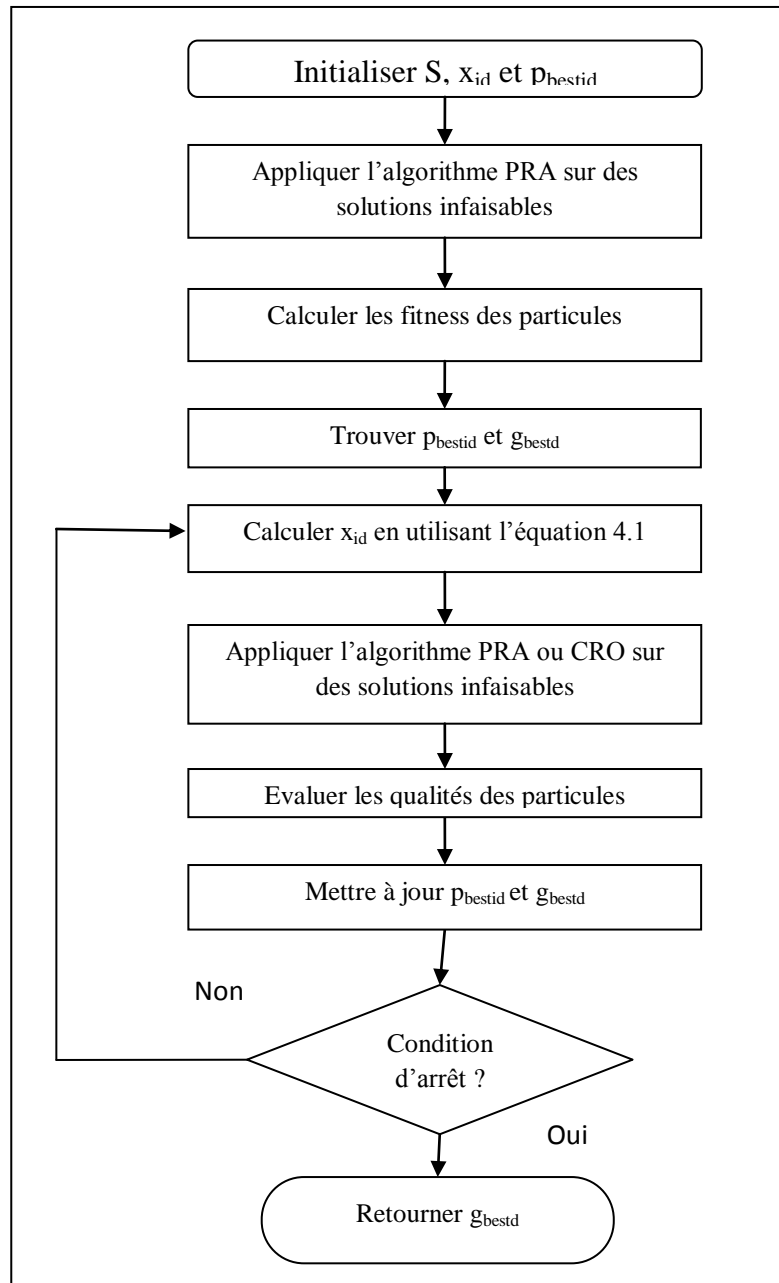


Figure 4.3. Organigramme de l'algorithme MHPSO2.

4.4.6. Les résultats expérimentaux

Pour tester l'efficacité et la performance des deux versions de l'algorithme MHPSO (i.e. MHPSO1 et MHPSO2), nous les avons utilisées pour résoudre quelques instances tirées de OR-Library [OR-Library, 1975]. Les tests ont été partagés en deux parties. Premièrement, nous avons testé le MHPSO1 et le MHPSO2 avec quelques instances du MKP de petites tailles. Les benchmarks utilisés dans cette partie d'expérimentations sont nommés : mknaps1, HP, SENTO, WEING et WEISH. Dans la deuxième partie d'expérimentations, nous avons testé le MHPSO1 et le MHPSO2 avec quelques instances du MKP de grandes tailles tirées des

benchmarks nommés mknapcb1 et mknapcb 4. Nous avons utilisé 5 instances du benchmark mknapcb1 (5.100) qui propose 5 contraintes (dimensions) et 100 objets. D'autre part, nous avons utilisé 5 instances du benchmark mknapcb 4 (10.100) qui propose 10 contraintes et 100 objets.

Les résultats obtenus ont été comparés en termes de meilleure solution obtenue (The best solution) et de moyenne des solutions obtenues (Average). Nous avons comparé nos résultats avec la solution optimale (best known), les résultats obtenus avec l'algorithme PSO-P [Kong et Tian, 2006] et les résultats obtenus avec l'algorithme QICSA [Layeb, 2011]. Sachant que, le PSO-P est une version modifiée de l'algorithme binaire standard de l'optimisation par essaim de particules. Il utilise la technique de la fonction de pénalité pour faire face aux problèmes d'optimisation avec contraintes. Le QICSA c'est la version quantique de la nouvelle métaheuristique « Cuckoo Search » proposée par Yang et Deb [Yang et Deb, 2010].

Instance	n	m	best known	MHPSO1	MHPSO2
HP1	28	4	3418	3418	3418
HP2	35	4	3186	3186	3186
PB5	20	10	2139	2139	2139
PB6	40	30	776	776	776
PB7	37	30	1035	1035	1035
SENT01	60	30	7772	7772	7772
SENT02	60	30	8722	8722	8722
WEING1	28	2	141278	141278	141278
WEING2	28	2	130883	130883	130883
WEING3	28	2	95677	95677	95677
WEING4	28	2	119337	119337	119337
WEING7	105	2	1095445	1095445	1095445
WEISH01	30	5	4554	4554	4554
WEISH06	40	5	5557	5557	5557
WEISH10	50	5	6339	6339	6339
WEISH15	60	5	7486	7486	7486
WEISH18	70	5	9580	9580	9580
WEISH22	80	5	8947	8947	8947

Tableau 4.2. Les résultats expérimentaux avec quelques instances tirées de la littérature.

Le tableau 4.2 représente les résultats expérimentaux obtenus par le MHPSO1 et le MHPSO2 avec quelques instances tirées de la littérature. La première colonne indique le nom de l'instance. La deuxième et la troisième colonne représentent la taille de l'instance i.e. le nombre d'objets et le nombre des dimensions du sac à dos respectivement. La quatrième colonne représente la solution optimale (best known) déclarée dans OR-Library. La colonne 5

et 6, représentent la meilleure solution obtenue par le MHPSO1 et le MHPSO2. Le tableau 4.2 montre que les algorithmes proposés sont capables de trouver la solution optimale (best known) de toutes les instances.

Le tableau 4.3 représente les résultats expérimentaux des exécutions comparatives en termes de meilleures et de moyennes des solutions (Best et Avg dans le tableau) entre les algorithmes MHPSO1, MHPSO2 et le PSO-P avec les instances nommées « mknapi ». La 1^{ère} colonne représente le nom du benchmark. La 2^{ème} colonne indique le nombre des objets. La 3^{ème} colonne indique le nombre des dimensions du sac à dos. Les colonnes 5, 6 et 7 représentent les meilleures (Best) et les moyennes (Avg) des solutions obtenues par le MHPSO1, le MHPSO2 et le PSO-P durant 30 exécutions de chaque instance.

Nom	N	M	best known	MHPSO1		MHPSO2		PSO-P [Kong et Tian, 2006]	
				Best	Avg	Best	Avg	Best	Avg
mknapi1	6	10	3800	3800	3800	3800	3800	3800	3800
mknapi2	10	10	8706,1	8706,1	8706,1	8706,1	8706,1	8706,1	8570.7
mknapi3	15	10	4015	4015	4015	4015	4015	4015	4014.7
mknapi4	20	10	6120	6120	6120	6120	6120	6120	6118
mknapi5	28	10	12400	12400	12385	12400	12386	12400	12394
mknapi6	39	5	10618	10618	10570	10618	10566	10618	10572
mknapi7	50	5	16537	16537	16459,2	16537	16460	16537	16389

Tableau 4.3. Comparaison des résultats obtenus par le MHPSO1, le MHPSO2 et le PSO-P avec quelques instances de petites tailles.

Les résultats présentés dans le tableau 4.3 montrent que nos algorithmes donnent de bons résultats comparés avec les résultats obtenus par le PSO-P qui est basé sur la technique de fonction de pénalité.

Le tableau 4.4 représente les résultats expérimentaux des exécutions comparatives en termes de meilleures et de moyennes des solutions obtenues par l'algorithme MHPSO et le PSO-P avec des instances de grandes tailles nommées mknapi1 et mknapi4. La 1^{ère} colonne représente le nom du benchmark. La 2^{ème} colonne indique la taille du problème. La 3^{ème} colonne représente la solution exacte déclarée dans OR-Library. Les colonnes 4 et 5

représentent les meilleures solutions (Best) et les moyennes (Avg) des solutions obtenues par le MHPSO1 et le PSO-P.

Nom des Benchmarks	Taille du Problème	Best known	MHPSO1		PSO-P [Kong et Tian, 2006]	
			Best	Avg	Best	Avg
mknapcb1	5.100.00	24381	23936	23549	22525	22013
	5.100.01	24274	23827	23475	22244	21719
	5.100.02	23551	23234	22921	21822	21050
	5.100.03	23534	23032	22722	22057	21413
	5.100.04	23991	23652	23169	22167	21677
mknapcb 4	10.100.00	23064	22687	22260	20895	20458
	10.100.01	22801	22256	21804	20663	20089
	10.100.02	22131	21744	21233	20058	19582
	10.100.03	22772	22341	21920	20908	20446
	10.100.04	22751	22204	21844	20488	20025

Tableau 4.4. Les résultats expérimentaux avec mknapcb1 et mknapcb 4 obtenus par MHPSO1 et PSO-P

Nom des Benchmarks	Taille du Problème	Best known	MHPSO1	MHPSO2	QICSA [Layeb, 2011]
mknapcb1	5.100.00	24381	23936	24329	23416
	5.100.01	24274	23827	24149	22880
	5.100.02	23551	23234	23494	22525
	5.100.03	23534	23032	23370	22727
	5.100.04	23991	23652	23889	22854
mknapcb 4	10.100.00	23064	22687	22983	21796
	10.100.01	22801	22256	22657	21348
	10.100.02	22131	21744	21853	20961
	10.100.03	22772	22341	22511	21377
	10.100.04	22751	22204	22614	21251

Tableau 4.5. Les résultats expérimentaux avec mknapcb1 et mknapcb 4 obtenus par MHPSO1, MHPSO2 et QICSA.

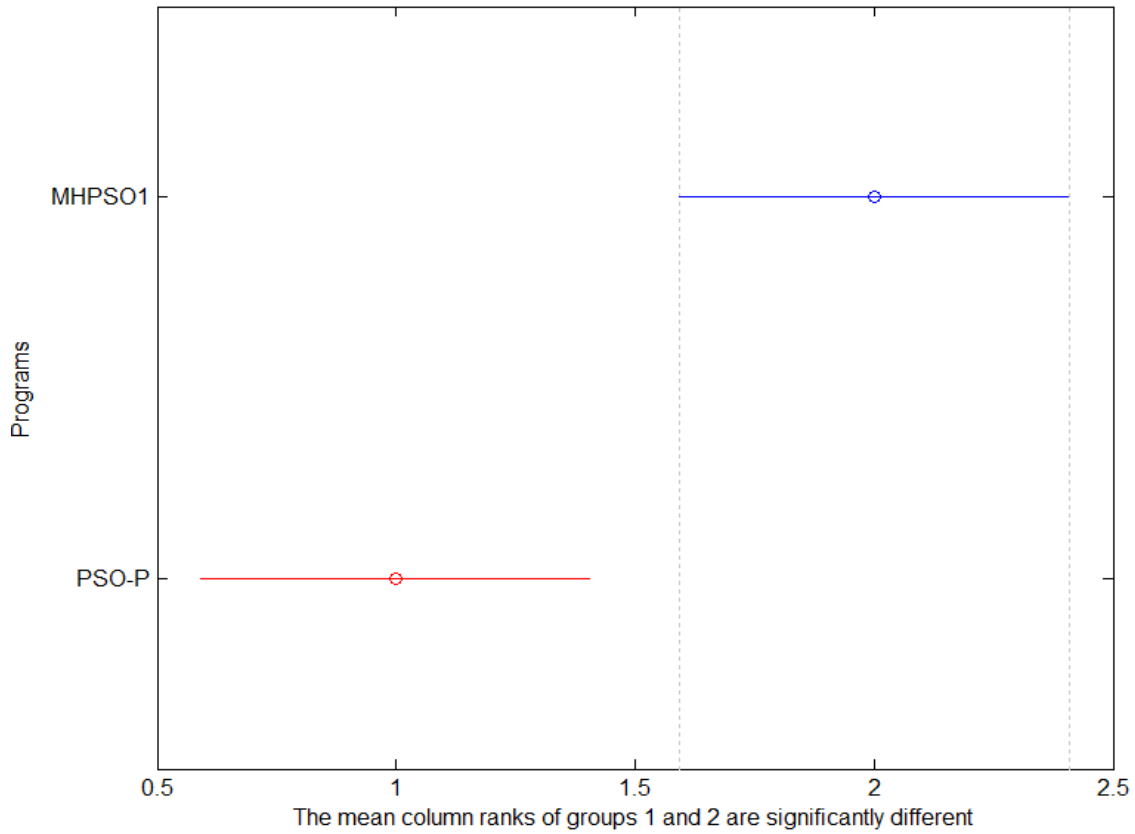


Figure 4.4. Le résultat du test de Friedman entre l'algorithme MHPSO1 et le PSO-P.

Les résultats présentés dans la table 4.4, montrent que l'algorithme MHPSO1 est plus performant que l'algorithme PSO-P. En fait, les résultats obtenus par notre algorithme sont tous meilleurs que ceux obtenus par le PSO-P.

Le tableau 4.5 représente les résultats expérimentaux des exécutions comparatives en termes de Best entre nos algorithmes MHPSO1, MHPSO2 et le QICSA avec quelques instances de mknpcb1 et mknpcb 4. La 1^{ère} colonne représente le nom du benchmark. La 2^{ème} colonne indique la taille du problème. La 3^{ème} colonne représente la solution optimale déclarée dans OR-Library. Les colonnes 4, 5 et 6 représentent les meilleures solutions (Best) obtenus par le MHPSO1, le MHPSO2 et le QICSA.

Les résultats présentés dans le tableau 4.5, montrent que nos algorithmes (MHPSO1 et MHPSO2) sont plus performants que l'algorithme QICSA. En fait, les résultats obtenus par les algorithmes MHPSO1 et MHPSO2 sont tous meilleurs que ceux obtenus par le QICSA. En outre, le tableau 4.5 nous montre que les résultats obtenus par l'algorithme MHPSO2 sont meilleurs que ceux obtenus par l'algorithme MHPSO1.

La figure 4.4 représente le résultat d'une comparaison statistique par le test de Friedman des moyennes des résultats obtenus par l'algorithme MHPSO1 et l'algorithme PSO-P. Ce test montre que la différence entre les résultats obtenus par l'algorithme MHPSO1 et l'algorithme PSO-P est significative.

Le test statistique de Friedman présenté dans la figure 4.5 représente une comparaison des résultats obtenus par les algorithmes MHPSO1, MHPSO2, QICSA et la solution optimale (Best Known) déclarée dans OR-Library. Selon la figure 4.5, la performance de l'algorithme MHPSO2 dépasse celles des algorithmes MHPSO1 et QICSA. En outre, la différence entre la performance de l'algorithme MHPSO2 et QICSA est significative. Le test de Friedman nous montre que la différence entre les résultats obtenus par les couples d'algorithmes (MHPSO2, QICSA), (MHPSO1, Best Known) et (QICSA, Best Known) est significative.

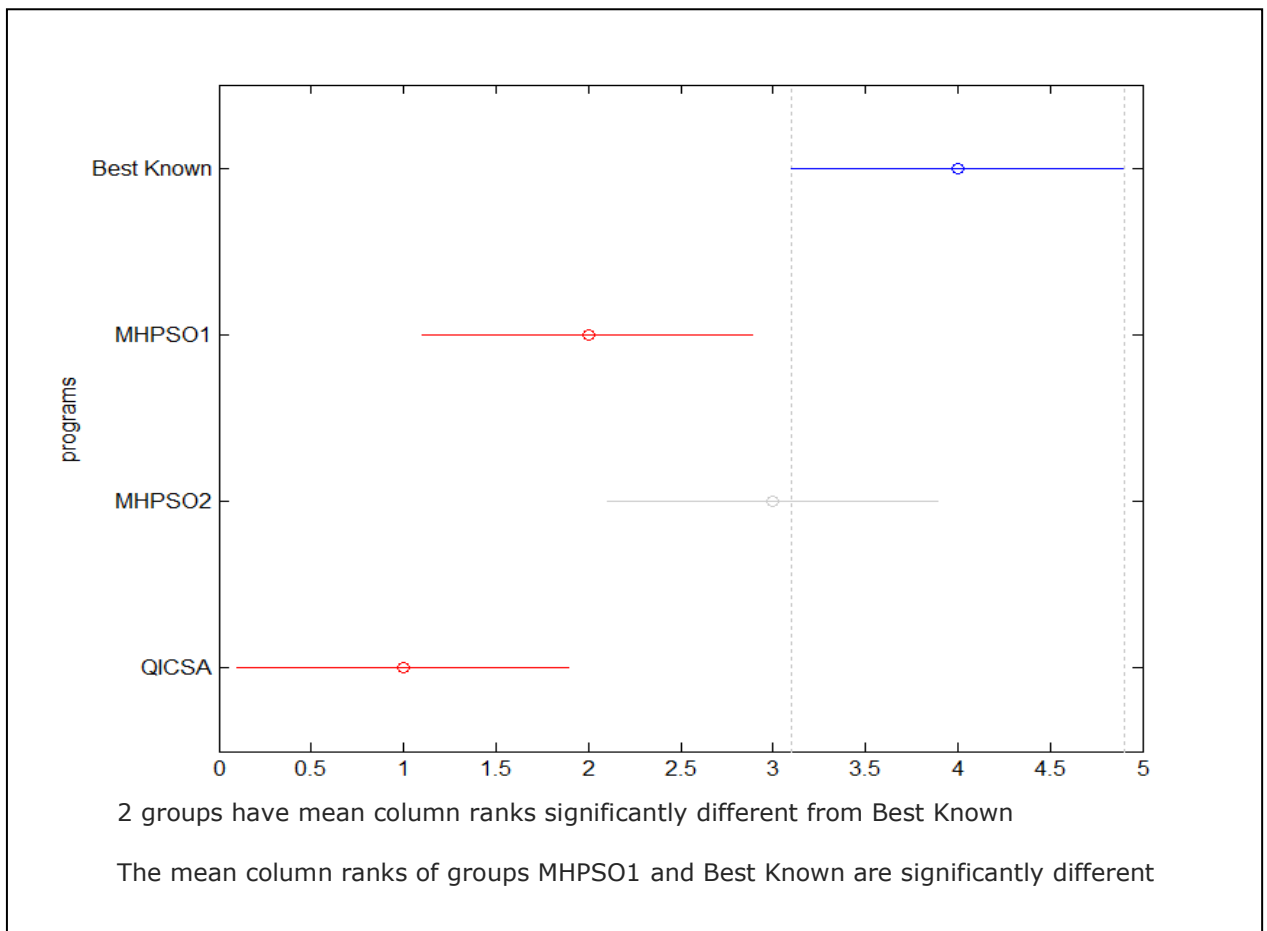


Figure 4.5. Le résultat du test de Friedman entre le MHPSO1, le MHPSO2, le QICSA et la solution optimale.

4.5. Application du MHPSO sur le problème du voyageur de commerce

4.5.1. L'algorithme 2-opt

2-opt est une heuristique de recherche locale, permettant d'obtenir des résultats faisables dans un délai raisonnable. Elle est utilisée pour améliorer la qualité d'une solution donnée. L'algorithme 2-opt commence avec une tournée admissible donnée, ensuite elle cherche dans le voisinage de la solution courante toute tournée améliorant la configuration actuelle.

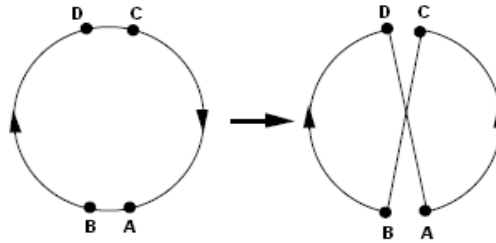


Figure 4.6. Exemple d'application de 2-opt

A chaque étape de la procédure d'amélioration, l'algorithme examine si l'échange de 2 arêtes produit une tournée plus courte pour améliorer la tournée. L'algorithme continue ainsi jusqu'à ce qu'aucune amélioration ne soit plus possible (Figure 4.6). Les étapes de l'algorithme sont :

Étape 1 : Prendre deux paires de nœuds consécutifs de la tournée : (A, B) et (C, D);

Étape 2: Vérifier si la distance $(AB+CD)$ est supérieure à $(AC+DB)$;

Étape 3: Si vérifiée, alors échanger A avec C en inversant le tour entre ces deux nœuds;

Étape 4: Si amélioration de la tournée, alors aller à 1, sinon s'arrêter.

4.5.2. Représentation

Dans le but de représenter les différentes particules (solutions), nous avons utilisé des vecteurs de taille D . Où D est le nombre de villes à parcourir. Chaque solution x_{id} est représentée par un ensemble d'entier, où chacun représente une ville donnée. L'ordre de présentation des villes représente l'ordre de parcourir de ces dernières.

Supposant que nous avons un problème de taille 6 (i.e. nombre de villes à parcourir est égale à 6). Une des solutions possibles à cette instance du problème du voyageur de commerce est la suivante:

$$x_{id} = [5, 2, 4, 1, 3, 6]$$

Cette solution représente l'ordre de parcourir des villes de la tournée.

4.5.3. L'opérateur de croisement

Dans cette contribution, nous avons appliqué l'opérateur de croisement entre la meilleure position p_{bestid} trouvée par la particule i et sa position actuelle x_{id} pour produire une nouvelle position. Ensuite, nous avons appliqué l'opérateur de croisement entre la meilleure position g_{bestid} trouvée par l'essaim et la position actuelle, pour avoir une autre position. Enfin, nous avons choisi, la meilleure entre les deux positions trouvées pour qu'elle représente la nouvelle position de la particule.

Une panoplie d'opérateurs de croisement a été proposée dans la littérature pour résoudre le problème du voyageur de commerce. Nous citons à titre d'exemple: le ERX (Edge Recombination Crossover), le PMX (Partially Mapped Crossover), le OBX (Order-Based Crossover), le DPX (Distance Preserving Crossover), le MPX (Maximal Preservative Crossover), et le ECH (Exchange Crossover Heuristic) [Li et Zhang, 2007]. En s'inspirant de l'algorithme ECH, nous avons proposé une version modifiée et plus adaptée au problème du voyageur de commerce. Elle permet la génération des solutions plus faisables qui respectent rigoureusement les contraintes du problème du voyageur de commerce.

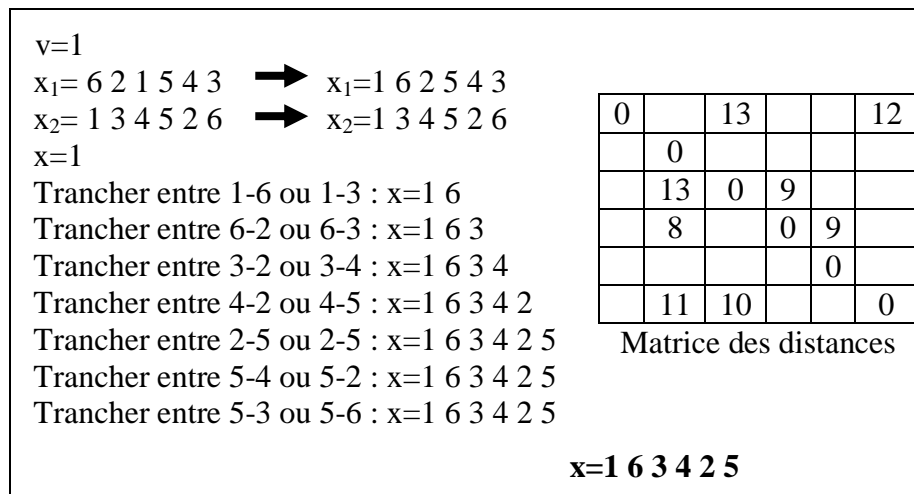


Figure 4.7. Un exemple du croisement MECH

Algorithme 4.6. L'algorithme MECHEntrée : Deux solutions x_1 et x_2 Sortie : Une solution x **Début**Choisir une ville aléatoire v ;Déplacer la ville v en amont de x_1 et x_2 ;Initialiser x par v : $x=v$; $i=2; j=2;$ **Tant que** $[(i \& j) \leq N]$ **faire****Si** $[(x_1(i) \& x_2(j)) \in x]$ **alors** $i++; j++;$ **Sinon-si** $[x_1(i) \in x]$ Concaténer $x_2(j)$ à x ; $j++$;**Sinon-si** $[x_2(j) \in x]$ Concaténer $x_1(i)$ à x ; $i++$;**Sinon-si** $[(x_1(i) \& x_2(j)) \notin x]$ $F =$ dernière ville dans x ;**Si** distance $[F, x_1(i)] <$ distance $[F, x_2(j)]$ **alors**Concaténer $x_1(i)$ à x ; $i++$;**Sinon**Concaténer $x_2(j)$ à x ; $j++$;**Fin si****Fin si****Fin tant que****Fin**

Supposons que nous avons deux particules (solutions) x_1 , x_2 et nous voulons les croiser ensemble afin d'obtenir une nouvelle particule. Nous commençons par choisir une ville aléatoire v à partir de l'ensemble des villes à visiter. Ensuite, nous générons 2 nouvelles particules en déplaçant la ville v en amont des villes de x_1 , x_2 . Après, nous initialisons la nouvelle particule x (l'enfant ou la nouvelle solution) par la ville v . Et nous poursuivons la construction de x , en concaténant les villes une après l'autre.

A chaque itération de la procédure de construction de x , nous nous trouvons devant 2 villes à choisir, la ville v_1 du premier parent (x_1) et la ville v_2 du deuxième parent (x_2). Pour trancher sur une des deux villes v_1 et v_2 , nous choisissons la ville la plus proche de la dernière ville de x . Lors de la procédure de construction de x , nous devons veiller à ce que les deux villes

candidates (v_1 et v_2) ne figurent pas dans la nouvelle solution x plus qu'une seule fois car nous devons respecter la contrainte principale du problème du voyageur de commerce (une ville doit être parcourue une seule fois). L'algorithme du croisement proposé est nommé MECH (de l'anglais, MECH : Modified ECH). L'algorithme 4.6 représente un pseudo code de l'algorithme MECH. Son principe est expliqué par un exemple dans la Figure 4.7

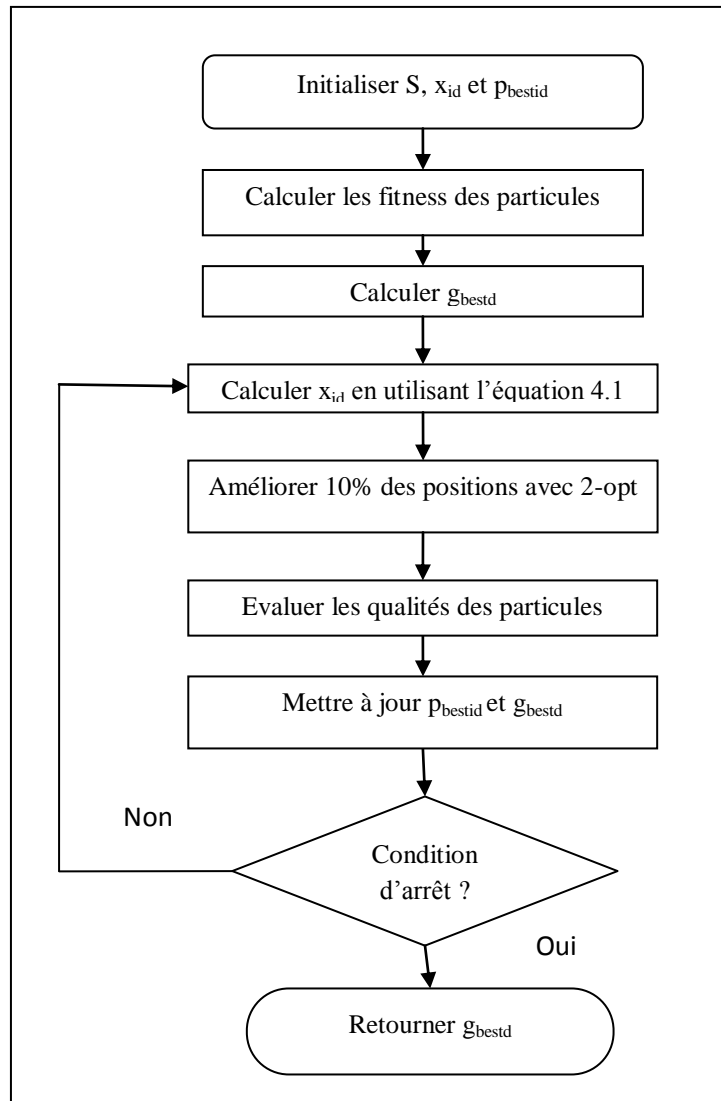


Figure 4.8. L'organigramme de l'algorithme MHP SO appliqué sur le problème du voyageur de commerce.

4.5.4. L'algorithme proposé

Les étapes de l'algorithme proposé (MHP SO) sont présentées ci-dessous.

Étape 1: - Initialiser la taille de l'essaim S ;

- Initialiser aléatoirement les positions des particules x_{id} ;
- Pour chaque particule, mettre $p_{bestid} = x_{id}$;

Étape 2: Calculer la fitness de chaque particule;

Étape 3: Calculer g_{bestd} ;

Étape 4: Calculer pour chaque particule sa nouvelle position en utilisant l'équation 4.1.
Où, « \otimes » est l'opérateur de croisement MECH.

Étape 5: Amélioration de 10% des positions de particules (solutions) avec l'heuristique 2-opt.

Étape 6: Mettre à jour p_{bestid} et g_{bestd} selon les formules 4.2 et 4.3 respectivement.

Étape 7: Stopper l'exécution si la condition d'arrêt est vérifiée. Retourner à l'étape 4, sinon.

La figure 4.8 représente un organigramme de l'algorithme proposé (MHPSO).

4.5.5. Les résultats expérimentaux

Pour valider la faisabilité et l'efficacité de l'algorithme proposé, nous l'avons appliqué sur quelques instances de la bibliothèque TSPLIB [TSPLIB] pour le problème du voyageur du commerce. Les tests ont été partagés en deux parties. Premièrement, nous avons appliqué le MHPSO sur quelques instances du problème du voyageur de commerce. Dans la deuxième partie d'expérimentations, Les résultats obtenus ont été comparés en termes de meilleure solution obtenue avec la solution optimale (best known), et les résultats obtenus avec l'algorithme DPSO [Shi et Liang, 2007] et l'algorithme CDPSO [Xu et al, 2008].

Le tableau 4.6 représente les résultats expérimentaux obtenus par notre algorithme MHPSO avec quelques instances tirées de la bibliothèque TSPLIB. La première colonne indique le nom de l'instance. La deuxième colonne représente la taille de l'instance (i.e. le nombre de villes). La troisième colonne représente la solution optimale déclarée dans TSPLIB. La colonne 4 représente la meilleure solution obtenue par le MHPSO. Les résultats présentés dans le tableau 4.6, montrent que notre algorithme est capable de trouver la meilleure solution de la plupart des instances.

Le test statistique de Friedman représenté dans la figure 4.9 représente une comparaison statistique des résultats obtenus par l'algorithme MHPSO et la solution optimale déclarée dans OR-Library. Selon la figure 4.9, la différence entre les résultats trouvés par le MHPSO et la solution optimale n'est pas significative (n'est pas très grande). Cela prouve la performance de l'algorithme proposé.

Instance	Nombre De villes	Solution optimale	Résultat MHPSO
bays29	29	2020	2020
berlin52	52	7542	7542
dantzig42	42	699	699
eil51	51	426	426
eil76	76	538	538
eil101	101	629	629
fri26	26	937	937
kroA100	100	21282	21282
kroB100	100	22141	22141
kroC100	100	20749	20749
kroD100	100	21294	21309
kroE100	100	22068	22068
kroB150	150	26130	26130
pr107	107	44303	44391
Pr124	124	59030	59030
Pr76	76	108159	108159
Rat99	99	1211	1212
Pr144	144	58537	58537
Suiss42	42	1273	1273
St70	70	675	675

Tableau 4.6. Résultats du MHPSO

Les tableaux 4.7, 4.8 et 4.9 représentent les résultats expérimentaux obtenus par l'exécution des trois algorithmes: DPSO, CDPSO et MHPSO sur les mêmes instances. Dans la 1^{ère} colonne, le nom de l'instance est affiché. La 2^{ème} colonne indique la solution optimale déclarée dans TSPLIB. Les colonnes 3 et 4 désignent la meilleure et la mauvaise solution obtenues en exécutant 100 fois l'algorithme sur la même instance. Enfin la cinquième colonne indique le taux d'erreur calculé selon la formule suivante:

$$\text{Erreur} = (\text{avg} - \text{opt}) / \text{opt} * 100\% \quad (4.7)$$

Tel que : opt est la solution optimale du problème, et avg est la valeur moyenne des solutions trouvées.

D'après ses résultats, on constate que le taux d'erreur de l'algorithme MHPSO ne dépasse pas le 1%, cela confirme sa robustesse et son efficacité.

Le tableau 4.10 représente les résultats expérimentaux des exécutions comparatives en termes de meilleure solution trouvée, entre notre algorithme MHPSO, l'algorithme DPSO et l'algorithme CDPSO. La 1^{ère} colonne représente le nom de l'instance, la 2^{ème} indique la

solution optimale déclarée dans TSPLIB. Les colonnes 3,4 et 5 représentent les meilleures solutions obtenues par le MHPSO, le DPSO et le CDPSO. N/D dans le tableau 4.10 veut dire résultat Non Disponible dans le papier référence. Les résultats obtenus sont très encourageants, ils prouvent l'efficacité de l'algorithme proposé.

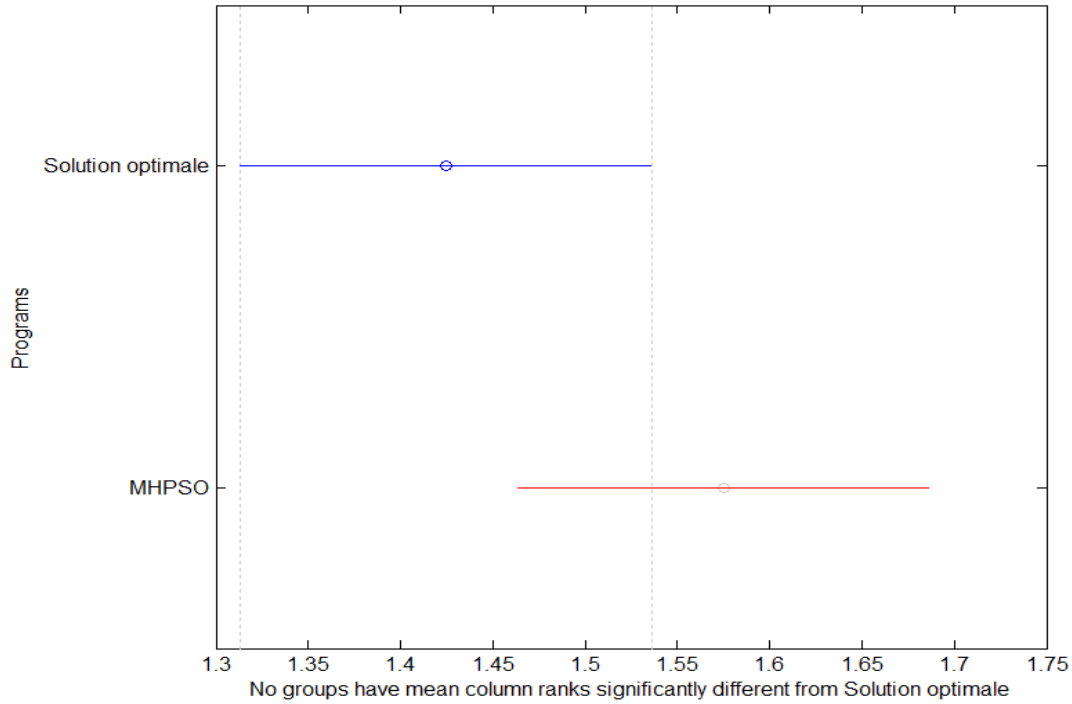


Figure 4.9. Le résultat du test de Friedman entre le MHPSO et la solution optimale.

Instance	Solution optimale	Meilleure solution	Mauvaise solution	Erreur (%)
eil5	426	427	437	2.38
berlin52	7542	7542	8024	3.85
st70	675	675	697	3.34
eil76	538	546	560	4.17
Pr76	108159	108280	111690	3.82
kroA100	21282	N/D	N/D	-
kroE100	22068	N/D	N/D	-
Pr124	59030	N/D	N/D	-
Pr144	58537	N/D	N/D	-
kroB150	26130	N/D	N/D	-

Tableau 4.7. Résultats obtenus par DPSO [Shi et Liang, 2007]

Instance	Solution optimale	Meilleure solution	Mauvaise solution	Erreur (%)
eil5	426	426	437	1.23
berlin52	7542	7542	8024	1.53
st70	675	675	697	1.06
eil76	538	538	560	2.39
Pr76	108159	108159	111690	1.18
kroA100	21282	21282	22258	1.54
kroE100	22068	22068	23049	1.66
Pr124	59030	59030	60329	0.87
Pr144	58537	58537	59636	0.56
kroB150	26130	26141	27736	4.23

Tableau 4.8. Résultats obtenus par CDPSO [Xu et al, 2008]

Instance	Solution optimale	Meilleure solution	Mauvaise solution	Erreur (%)
eil5	426	426	427	0.11
berlin52	7542	7542	7542	0.00
st70	675	675	681	0.23
eil76	538	538	542	0.29
Pr76	108159	108159	108469	0.09
kroA100	21282	21282	21282	0.00
kroE100	22068	22068	22278	0.56
Pr124	59030	59030	59164	0.04
Pr144	58537	58537	58570	0.01
kroB150	26130	26130	26572	0.49

Tableau 4.9. Résultats obtenus par MHPSO

Instance	Solution optimale	Meilleure solution MHP SO	Meilleure solution DPSO [Shi et Liang, 2007]	Meilleure Solution CDPSO [Xu et al, 2008]
eil51	426	426	427	426
berlin52	7542	7542	7542	7542
st70	675	675	675	675
eil76	538	538	546	538
Pr76	108159	108159	108280	108159
kroA100	21282	21282	N/D	21282
kroE100	22068	22068	N/D	22068
Pr124	59030	59030	N/D	59030
Pr144	58537	58537	N/D	58537
kroB150	26130	26130	N/D	26141

Tableau 4.10. Résultats comparatifs entre MHP SO, DPSO et CDPSO

4.6. Conclusion

Dans cette contribution, nous avons proposé un nouvel algorithme hybride que nous avons nommé MHP SO. Le MHP SO est une combinaison de quelques principes de l'algorithme d'optimisation par essaim de particules et du croisement de l'algorithme génétique. A l'opposé de l'algorithme originel d'optimisation par essaim de particules conçu pour la résolution des problèmes continus, notre algorithme peut résoudre tous les types de problèmes d'optimisation. Il nécessite peu de paramètres (la taille de la population) et il représente un bon équilibre entre l'exploitation et l'exploration de l'espace de recherche. En fait, la création d'une nouvelle solution est basée sur une recherche locale autour de la solution courante de la particule et sa meilleure solution d'une part, et une recherche globale autour de la solution courante et la meilleure solution trouvée par l'essaim d'autre part.

Dans le but de tester la performance de l'algorithme proposé, nous l'avons utilisé dans la résolution de deux problèmes d'optimisation académiques NP-Difficiles: le problème du sac à dos multidimensionnel et le problème du voyageur de commerce. Dans la première application (i.e. sur le problème du sac à dos multidimensionnel), nous avons proposé deux variantes de l'algorithme MHP SO: MHP SO1 et MHP SO2. Dans l'algorithme MHP SO1 nous avons modifié l'équation de mise à jour de solution et nous avons utilisé l'algorithme de

croisement et l'algorithme de réparation de solution (PRA) que nous avons proposés. Dans l'algorithme MHPSO2, nous avons intégré l'algorithme de croisement et l'algorithme de réparation de solution (PRA) que nous avons proposés et un autre algorithme de réparation de solution nommé CRO [Cotta et Troya, 1998] dans l'algorithme MHPSO. Dans la deuxième contribution (i.e. sur le problème du voyageur de commerce), nous avons intégré l'algorithme de croisement MECH (que nous avons proposé) et l'heuristique 2-opt dans l'algorithme MHPSO (Voir figure 4.10). Les résultats expérimentaux ont montré de bons résultats qui nous encouragent à l'utiliser pour la résolution d'autres problèmes d'optimisation difficiles.

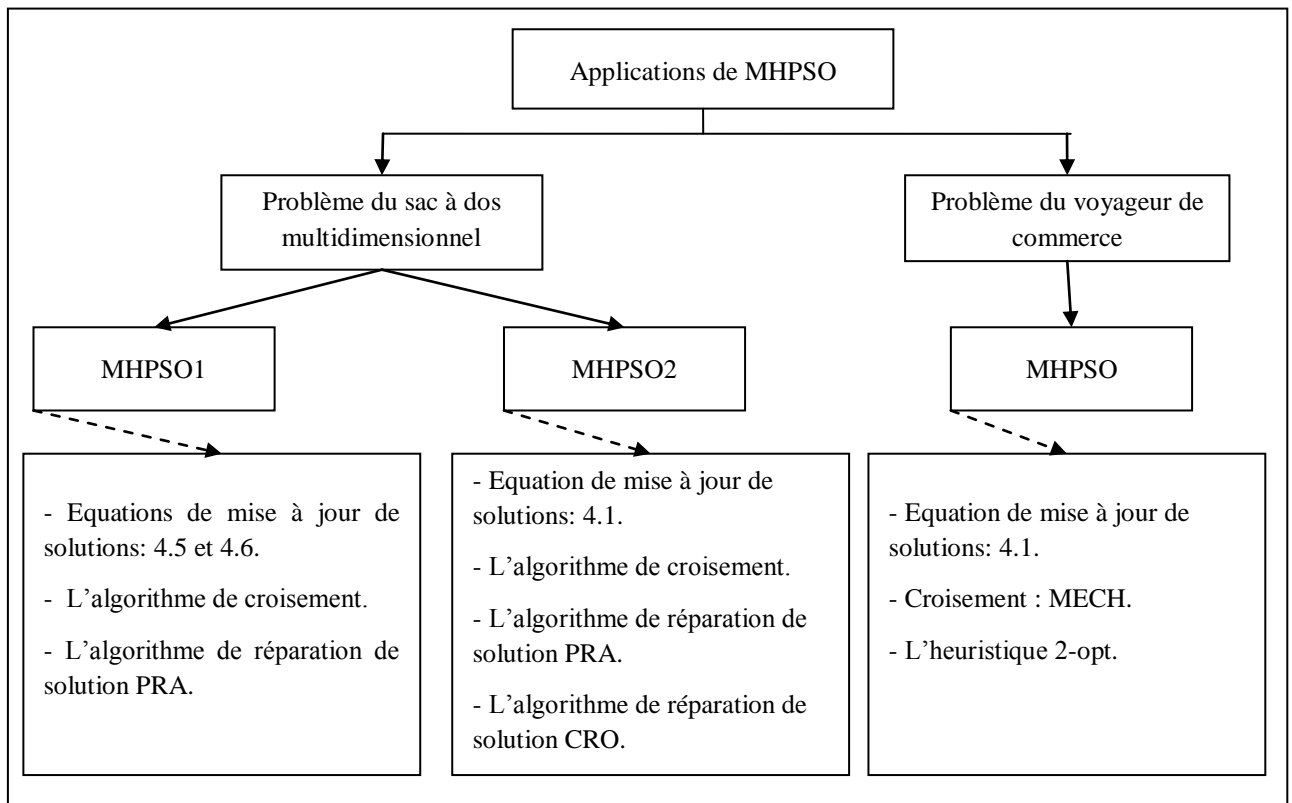


Figure 4.10. Applications de l'algorithme MHPSO

Les algorithmes proposés ont fait l'objet des articles suivants:

1. Amira Gherboudj, Said Labed and Salim Chikhi. A New Hybrid Binary Particle Swarm Optimization Algorithm for Multidimensional Knapsack Problem. Advances in Computer Science, Engineering and Applications. Proceedings of the Second International Conference on Computer Science, Engineering and Applications (ICCSEA 2012), Vol 166. pp. 489-498. David C. Wyld, Jan Zizka, and Dhinakaran Nagamalai (Eds). ISSN 1867-5662. Springer-Verlag Berlin Heidelberg, 2012.
2. Said Labed, Amira Gherboudj, Salim Chikhi. A Modified Hybrid Particle Swarm Optimization Algorithm for Multidimensional Knapsack Problem. International Journal of

Computer Applications (0975 - 8887). Vol 34, N°.2, pp. 11-16. ISBN: 978-93-80865-38-9. November 2011.

3. Said Labed, Amira Gherboudj, Salim Chikhi. A Modified Hybrid Particle Swarm Optimization Algorithm For Solving The Traveling Salesmen Problem. Journal of Theoretical and Applied Information Technology. Vol. 39 N°.2. pp. 132- 138. ISSN: 1992-8645. May 2012.

Dans le chapitre suivant, nous présenterons notre troisième contribution dans laquelle nous avons proposé une version binaire du nouvel algorithme nommé « la recherche coucou » et que nous avons utilisé pour résoudre le problème d'optimisation combinatoire NP-Difficile du sac à dos unidimensionnel et sa version multidimensionnelle.

CHAPITRE 5

L'algorithme binaire de la recherche coucou

Sommaire

5.1	Introduction.....	155
5.2	Le comportement et la reproduction des Coucous	157
5.3	Les métaheuristiques inspirées Coucou	160
5.4	Discussion et comparaison.....	168
5.5	La méthode proposée: Une version binaire de la recherche coucou	171
5.6	Les résultats expérimentaux	178
5.7	Discussion	184
5.8	Conclusion.....	185

5.1. Introduction

L'optimisation joue un rôle très essentiel dans la recherche opérationnelle, informatique et mathématique. Elle consiste à ajuster des entrées selon des caractéristiques souhaitées. En fait, l'optimisation est un processus mathématique ou expérimental permettant d'aboutir à un résultat minimal ou maximal. Dans un problème d'optimisation, l'entrée est un ensemble de variables et de contraintes, le traitement est un ensemble d'opérations permettant la satisfaction d'une fonction objectif (connue aussi par: fonction fitness) et la sortie est une solution qui a une certaine qualité calculée selon la fonction objectif du problème traité. De nombreux algorithmes de résolution de problèmes d'optimisation ont été proposés dans la littérature. On peut distinguer des algorithmes basés sur la recherche locale et des algorithmes basés sur la recherche globale. Les algorithmes basés sur la recherche locale sont souvent inefficaces face à des problèmes difficiles de grandes tailles. Par contre, la recherche globale peut être plus efficace dans l'obtention des solutions optimales [Deb 1995, Arora 1989, Yang, X.S 2005, Yang 2008].

Les algorithmes évolutionnaires ont prouvé leur efficacité dans la résolution des problèmes d'optimisation complexes. En fait, ils permettent une efficace recherche globale de la solution optimale en présentant plusieurs caractéristiques intéressantes y compris l'adaptation, l'émergence et l'apprentissage [Fogel, 2001]. Les réseaux de neurones artificiels, l'algorithme génétique et l'intelligence par essaim sont des exemples des systèmes bio-inspirés utilisés dans le domaine de l'optimisation. Ces dernières décennies, l'optimisation par intelligence d'essaim est devenue une orientation intéressante pour la communauté des chercheurs des méthodes d'optimisation, notamment pour la communauté des chercheurs des méthodes d'optimisation évolutionnaires. Plusieurs algorithmes basés sur l'intelligence d'essaim ont apparus comme l'algorithme de colonies de fourmis [Dorigo et Gambardella 1997, Hu et al 2008], les algorithmes inspirés des systèmes écologiques [Nabti et Meshoul 2009, Zheng 2010], ainsi que l'algorithme de colonies d'abeilles artificielles [Yang 2005, Zhang.J et al 2007, Karaboga et Basturk 2007]. L'algorithme d'optimisation par essaim particulaire (PSO) [Kennedy et Eberhart, 1995] est l'algorithme principal dans la classe des algorithmes basés sur l'intelligence d'essaim. Le PSO est inspiré du comportement collectif que l'on trouve au sein de groupes d'animaux tels les bancs de poissons, les volées d'oiseaux ou les essaims d'insectes. IL a été utilisé avec succès dans la résolution d'une panoplie de problèmes d'optimisation difficiles à résoudre. L'efficacité, la simplicité d'implémentation et d'utilisation du PSO sont des caractéristiques avantageuses de l'algorithme comparé avec

d'autres algorithmes existants. En fait, le mécanisme de recherche et de mise à jour des solutions dans le PSO est fondé sur deux facteurs (la vitesse de déplacement et la position des particules) calculés par deux équations très simples.

Deux variantes très récentes du PSO ont été proposées dans la littérature: l'algorithme de la recherche coucou (CS) et l'algorithme d'optimisation par coucou (COA). Ils sont inspirés du comportement de reproduction et de vie d'une espèce spéciale d'oiseaux nommés Coucou. Ces derniers ont un caractère parasitaire dans la ponte, l'incubation et la nourriture de leurs poussins. En fait, certains coucous ne construisent jamais leurs propres nids pour pondre leurs propres œufs. Ils parasitent des nids d'autres oiseaux d'autres espèces. Après la ponte, les coucous se sauvent en s'affranchissant de leurs responsabilités parentales envers leurs œufs. Ils confient leurs œufs aux oiseaux hôtes qui seront leurs parents adoptifs. Les œufs coucous risquent d'être tués par leurs parents adoptifs si leur degré de similitude avec le modèle des œufs de l'oiseau hôte est faible. La première métaheuristique inspirée coucou (i.e. CS) a été proposée en 2009 par Xin-She Yang et Suash Deb [Yang et Deb, 2009 ; Yang et Deb, 2010]. Tandis que, la deuxième (i.e. COA) a été proposée en 2011 par Ramin Rajabioun [Rajabioun, 2011]. Afin de proposer leur nouvelle métaheuristique, Yang et Deb se sont basés sur le mécanisme du Vol de Lévy et sur un côté particulier du comportement des coucous, c'est bien leur comportement de reproduction parasitaire. Alors que, Rajabioun s'est basé sur le mode de vie général des coucous (leur habitat, leur reproduction, la migration...etc). L'apparition de l'algorithme CS a précédé celle de l'algorithme COA par deux ans. Dès son apparition, l'algorithme CS a ouvert une nouvelle porte d'investigation dans le domaine de méthodes de résolution de problèmes d'optimisation. En équilibre avec son jeune âge, quelques applications de l'algorithme CS ont été proposées dans la littérature comme la version discrète appliquée sur le problème d'ordonnancement des infirmiers (Nurse-scheduling) [Tein et Ramli, 2010], la classification par des réseaux de neurones (feedforward neural network) [Valian et al, 2011], L'agrégation de données dans un réseau de capteurs [Dhivyaet et Sundarambal, 2011] et la version quantique appliquée sur des problèmes du sac à dos [Layeb, 2011].

La version originelle de l'algorithme CS opère dans des espaces de recherche continus. Ce qui permet de générer des solutions avec des valeurs de type réel. Aucune version binaire n'a été proposée dans la littérature afin de faire face aux problèmes d'optimisation binaires. Raison pour laquelle nous avons songé à proposer une version binaire de l'algorithme CS. Nous nous sommes donc inspirés de la version binaire de l'algorithme d'optimisation par

essaim particulaire: l'algorithme BPSO [Kennedy et Eberhart, 1997]. Dans le BPSO, Kennedy et Eberhart ont intégré la fonction Sigmoidale dans le corps de l'algorithme d'optimisation par essaim particulaire afin de permettre la génération des solutions binaires au problème à traiter. De même, dans cette contribution nous avons combiné les étapes de l'algorithme de la recherche coucou et l'algorithme de transformation binaire des solutions que nous avons nommé BSR. Le BSR se base sur la fonction Sigmoidale pour la transformation des solutions réelles vers des solutions binaires. Notre premier objectif était de proposer une version binaire pour la nouvelle métaheuristique CS afin d'étendre son utilisation à un type spécial de problèmes d'optimisation: le type de problèmes d'optimisation binaires. Notre deuxième objectif était de tester et prouver l'efficacité de la version proposée dans la résolution des problèmes d'optimisation difficiles. Nous avons donc utilisé notre algorithme pour la résolution de deux problèmes d'optimisation combinatoires NP-Difficiles: Le problème du sac à dos unidimensionnel binaire et le problème du sac à dos multidimensionnel binaire.

5.2. Le comportement et la reproduction des Coucous

Toutes les espèces d'oiseaux ont la même approche de prolifération. Aucun oiseau ne donne naissance à des petits vivants. Après l'accouplement, l'oiseau pond un œuf recouvert d'une coquille protectrice qui est ensuite incubée en dehors du corps. Vu la richesse en protéine des œufs d'oiseau, ils construisent un volumineux repas pour toutes sortes de prédateurs. Afin de protéger leurs œufs des menaces des différents prédateurs, les oiseaux doivent trouver un endroit sûr pour donner plus de chance d'éclosion à leurs poussins. Trouver un endroit pour placer en toute sécurité, couvrir leurs œufs et élever leurs poussins au point de l'indépendance, est un défi pour les oiseaux. Ces derniers l'ont résolu de nombreuses manières astucieuses en utilisant des conceptions complexes et artistiques. De nombreux oiseaux construisent des nids isolés, cachés dans la végétation pour échapper au risque d'être détecté par les prédateurs. Certains d'entre eux sont des spécialistes dans le choix du bon emplacement de leurs nids. Ils cachent bien leurs nids de façon à ce que même les yeux de l'homme qui voit tout n'aient pratiquement jamais pu les détecter [Attenborough, 2012]. Il ya un autre type d'oiseaux qui se dispensent des responsabilités imposées par leur lien de parentalité. Ce sont des oiseaux parasites. Ils ne construisent pas leurs propres nids. En effet, ils pondent leurs propres œufs dans les nids des autres espèces en confiant la responsabilité d'incubation, de nourriture et d'élevage de leurs œufs aux parents adoptifs.

Les coucous sont un bon exemple d'oiseaux parasites de couvée. Ce sont des oiseaux fascinants, ils ont un bon chant. Ils appartiennent à la classe des cuculidés. Cette dernière compte environ 140 espèces réparties partout dans le monde. Plusieurs espèces de coucous sont des parasites. Ils ne construisent pas leur propre nid pour pondre leurs œufs. Quelques espèces entre eux s'engagent à pondre leurs œufs dans des nids communautaires où plus d'une femelle pond dans le même nid, comme l'Ani et le Guira [Bouglouan, 2012]. En fait, ils pondent leurs œufs dans des grands nids collectifs des membres de leur propre espèce puis ils se chargent de l'élevage collectif de leurs petits poussins. D'autres espèces pondent leurs œufs dans des nids d'autres espèces [Robert, 2005] puis ils se sauvent, comme le coucou gris, le coucou koël, Le Coucou plaintif et le coucou geai. Les coucous de ce dernier type ne couvent pas et ne nourrissent jamais leurs poussins. Quelques explications tentent de traduire ce comportement par le fait que le coucou parent se nourrit d'insectes toxiques. Cette nourriture menace la vie de ses poussins ce qui pousse le coucou parent à confier la responsabilité de nourriture et d'élevage de ses petits à d'autres espèces capables de leur fournir la bonne et la saine nourriture.

De nombreuses espèces d'oiseaux apprennent à reconnaître les œufs coucous déversés dans leurs propres nids. Dans le cas où l'oiseau hôte arrive à découvrir l'œuf coucou dans son nid, il va soit jeter l'œuf étranger hors son nid ou abandonner son propre nid. Tandis que les hôtes tentent de trouver des moyens de détection de l'œuf parasite, le coucou cherche constamment à améliorer le mimétisme du modèle des œufs de ses hôtes [Attenborough, 2012]. Le coucou est un expert dans l'art de tromperie. Sa stratégie se base sur la furtivité, la surprise et la vitesse [Attenborough, 2012]. Après l'accouplement, le coucou pond ses œufs dans les nids des autres espèces. L'œuf coucou risque de ne pas survivre au cas où l'oiseau hôte découvre qu'il n'est pas le sien. Dans le but d'augmenter la ressemblance entre l'œuf coucou et les œufs d'accueil, certaines femelles coucou sont très spécialisées dans le mimétisme de couleurs et de modèles des œufs de l'hôte. Chaque femelle coucou se spécialise sur une espèce hôte particulière. En outre, Les mamans coucous peuvent détruire les œufs de l'oiseau hôte afin d'hausser la possibilité d'éclosion et de nourriture de leurs œufs.

En général, les œufs du coucou se développent plus rapidement et éclosent plus tôt que ceux de l'oiseau d'accueil. Dès son éclosion, le poussin coucou dupe ses parents adoptifs. Il éjecte les œufs d'accueil par aveuglement ce qui lui offre la monopolisation des soins de ses parents adoptifs et de la nourriture destinée à toute une couvée. Il incite ses parents adoptifs à suivre le rythme de son taux de croissance élevé par son appel de mendicité rapide [Adams,

2012] qui imite l'appel des poussins d'accueil et aussi sa bouche ouverte qui construit un fort stimulus [Campbell, 1996]. En effet, il grandit plus vite au détriment des poussins d'accueil.

5.2.1. L'habitat des coucous

L'habitat est le milieu de vie, de reproduction et de développement d'une population d'une espèce donnée. De même pour les coucous, il constitue une source de nourriture et un lieu de reproduction. Les Coucous se produisent dans une grande variété d'habitats. La majorité des espèces survivent dans les forêts et les bois, principalement dans les forêts tropicales à feuilles persistantes. En plus des forêts, certaines espèces de coucous occupent des environnements plus ouverts, ce qui peut inclure même les zones arides comme les déserts [Rajabioun, 2011]. A titre d'exemple, le Coucou plaintif fréquente une assez grande variété d'habitats tels que les bois ouverts, les forêts secondaires, arbustes et broussailles, les champs cultivés et aussi les jardins, aussi bien en milieu rural que citadin. On peut aussi le voir dans les herbages et les marais. Le Coucou gris fréquente les forêts de conifères ou de feuillus, et les zones boisées en général, les espaces boisés ouverts, les lisières de forêts et les clairières, les steppes arborées, les prairies, les marais et les roselières, ainsi que les zones cultivées avec des arbres et des buissons. Le Coucou geai fréquente des habitats semi-arides tels que les zones boisées ouvertes (surtout avec des acacias et des arbustes épineux), les contreforts rocheux des collines dans les savanes sèches, et les zones agricoles sèches avec des arbres et des buissons. On le voit souvent voler au-dessus des espaces découverts. Selon la distribution, et particulièrement en Europe, on peut le trouver dans les landes de bruyères avec du chêne-liège et des conifères du genre *Pinus pinea*. Il fréquente également les bosquets et les plantations d'oliviers [Bouglouan, 2012].

5.2.2. La migration des coucous

La migration est un phénomène très répandu chez certains animaux y compris les oiseaux. Ce phénomène n'est pas obligatoire. En fait, il est lié à des paramètres alimentaires, physiologiques et hormonaux. En fonction de ces paramètres, l'oiseau est incité à se déplacer ou non. La date de migration n'est pas commune. Certaines espèces partent avant de manquer de nourriture, d'autres attendent quelques changements climatiques. En effet, l'oiseau risque de perdre sa vie en cas de baisse de disponibilité de nourriture ou des conditions climatiques rigoureuses. La plupart des espèces de coucous sont sédentaires, mais plusieurs espèces de coucou entreprennent des migrations saisonnières, et plusieurs autres entreprennent des migrations partielles sur une partie de leur distribution. La migration peut être diurne, comme celle du coucou 'Channel-billed' comme elle peut être nocturne, comme

celle du coucou à bec noir et du coucou à bec jaune qui se reproduisent en Amérique du nord et migrent de nuit à travers le Mexique et l'Amérique centrale pour hiverner en Amérique du Sud. En général, les coucous parasites migrent sur de longues distances et ont un vol rapide et direct. Leurs déplacements coïncident non seulement avec la saison de reproduction des espèces hôtes, mais aussi avec l'émergence des chenilles qui constituent leur nourriture favorite [Bouglouan, 2012]. Une preuve tangible a été fournie par un coucou bagué en 1939, à sa naissance, et repris en 1952, en Allemagne. Cet oiseau avait effectué 26 trajets entre l'Europe et l'Afrique, soit au minimum 150 000 km. L'équivalent de presque quatre fois le tour du globe.

5.3. Les métaheuristiques inspirées Coucou

En se basant sur le comportement des coucous ainsi que sur leur mode de vie et de reproduction, deux versions d'une nouvelle métaheuristique ont été proposées très récemment. La première version est nommée: La recherche coucou, en anglais: Cuckoo Search (CS). La deuxième version est nommée: Algorithme d'optimisation par coucou, en anglais: Cuckoo Optimization Algorithm (COA). Le principe ainsi que les étapes de chaque version seront bien étalés dans ce qui suit.

5.3.1. La recherche Coucou (CS)

En 2009, Xin-She Yang et Suash Deb ont proposé une nouvelle métaheuristique nommée la recherche coucou (CS). La recherche coucou est une métaheuristique très récente. Elle a enrichi le nombre des métaheuristiques à base de population de solutions. C'est une des variantes de l'algorithme d'optimisation par essaim particulaire (PSO). Les pionniers de l'algorithme CS se sont inspirés du comportement de reproduction parasitaire de quelques espèces de coucous qui pondent leurs œufs dans les nids des autres espèces en confiant la responsabilité d'incubation, de nourriture et d'élevage de leurs poussins aux oiseaux hôtes. Ces derniers peuvent détecter les œufs coucous dans leurs nids, dans ce cas là l'oiseau hôte va ou bien éjecter l'œuf coucou hors son nid ou abandonner son propre nid et construire un autre dans un autre emplacement. Yang et Deb se sont basés sur ce comportement parasitaire des coucous et sur le mécanisme du vol de Lévy qui permet la modélisation mathématique des déplacements aléatoires pour proposer une nouvelle méthode d'optimisation: La recherche coucou (CS). Malgré le nombre limité des travaux d'applications du CS [Tein et Ramli 2010, Layeb 2011, Gherboudj et al 2012] pour la résolution des problèmes d'optimisation, les

résultats obtenus sont prometteurs en termes de performance et d'efficacité de la nouvelle métaheuristique dont le principe est élucidé dans ce qui suit.

5.3.1.1. Le vol de Lévy

De nombreux phénomènes naturels ou sociaux, peuvent être décrits en termes de marche aléatoire: diffusion d'un soluté dans un solvant, de la chaleur dans un gaz, de la lumière dans le brouillard, la récolte de nourriture par certains animaux... [Mercadier, 2008]. Ces derniers cherchent leur nourriture de manière aléatoire. En général, le processus de recherche de nourriture chez les animaux est effectivement aléatoire. En fait, leur déplacement est basé sur leur position actuelle ainsi qu'une probabilité du déplacement vers une autre position. Des études expérimentales sur le comportement de certains animaux et insectes ont montré que leur comportement peut être modélisé par un schéma mathématique nommé vol de Lévy (en anglais Lévy flight) [Brown et al 2007, Reynolds and Frye 2007, Pavlyukevich 2007].

Le vol de Lévy ou Lévy flight a été proposé par le mathématicien français Paul Pierre Lévy, un des fondateurs de la théorie moderne de probabilités. Depuis sa création, le vol de Lévy a donné des interprétations théoriques à plusieurs phénomènes physiques, chimiques, biologiques et naturels. En fait, le vol de Lévy permet de modéliser des marches aléatoires composées d'un grand nombre de pas où les transitions sont basées sur des probabilités. En terminologie mathématique, le vol de Lévy est une *marche aléatoire* (en anglais, Random walk: une formalisation mathématique d'une trajectoire composée d'un ensemble de pas aléatoires) dans laquelle la distance entre les pas a une *distribution probabilitaire* (en anglais, probability distribution: une fonction qui représente la probabilité d'un nombre aléatoire de prendre une valeur donnée) à *queue-lourde* (en anglais, heavy-tail: dont les queues ne sont pas bornées de façon exponentielle) [Shlesinger et al 1995, Ben-Avraham et Havlin 2002].

Plusieurs études récentes montrent qu'une panoplie de phénomènes dans différents domaines peuvent être modélisés par le vol de Lévy: Le déplacement des mouches [Reynolds et Frye, 2007], la diffusion de la lumière [Barthelemy et al ,2008], Les mouvements des organismes biologiques [Viswanathana et al, 2002], la recherche aléatoire des objets [Viswanathana et al, 2000] et le domaine de l'optimisation et la recherche optimale où les études expérimentales ont montré des résultats encourageants [Shlesinger, 2006] [Pavlyukevich, 2007]. Restant dans le domaine de l'optimisation et des métaheuristiques, Yang et Deb [Yang et Deb, 2009] ont intégré de leur part le vol de Lévy dans leur récente métaheuristique: La recherche coucou (CS) pour générer de nouvelles solutions.

5.3.1.2. Le principe et les étapes de la recherche coucou

En s'inspirant du comportement des coucous dans leur reproduction, Yang et Deb se sont basés sur trois principes pour proposer leur nouvelle métaheuristique [Yang et Deb, 2010].

- ✓ Chaque coucou pond un seul œuf à la fois. Il le dépose dans un nid qu'il choisit aléatoirement.
- ✓ Les meilleurs nids qui incluent des œufs (solutions) de bonnes qualités vont être les élus qui construisent les membres de la nouvelle génération.
- ✓ Le nombre des nids hôtes valides est fixé. L'oiseau hôte peut détecter le coucou étranger avec une probabilité $P_a \in [0,1]$. Dans ce cas là, l'oiseau hôte tranche entre écarter le coucou de son nid en lui éjectant hors nid ou abandonner son nid pour aller construire un autre dans une nouvelle position.

La probabilité P_a représente la fraction de N nids qui vont être remplacés par de nouveaux nids (avec de nouvelles solutions aléatoires dans de nouvelles positions dans l'espace de recherche). La qualité d'un nid ou d'une solution est mesurée en fonction de la fonction fitness qui se varie d'un problème à un autre.

Afin de générer une nouvelle solution $X_{(t+1)}$ pour un coucou i , Yang et Deb ont intégré le vol de Lévy de la manière suivante:

$$x_i(t+1) = x_i(t) + \alpha \oplus \text{Lévy}(\lambda) \quad (5.1)$$

Où $\alpha > 0$ est la taille du pas, elle est liée au problème traité.

La nouvelle solution sera donc générée en fonction de deux facteurs indispensables:

- La position actuelle du coucou.
- La nouvelle direction mesurée par le vol de Lévy.

Le vol de Lévy représente une marche aléatoire dont les pas aléatoires sont définis à partir de la distribution de Lévy (voir équation 2). Il est à noter que la distribution de Lévy a une panoplie de variantes avec une infinité de sens.

$$\text{Lévy} \sim u = t^{-\lambda}, (1 < \lambda \leq 3) \quad (5.2)$$

L'algorithme 5.1 résume les étapes générales de l'algorithme CS, comme elles sont implémentées dans le code standard du CS [Mathworks, 2012].

Algorithme 5.1. L'algorithme de la recherche coucou (CS)

Début

Initialiser une population de N coucous (solutions);

Tant que (le critère d'arrêt n'est pas satisfait) **faire**

Pour chaque coucou s **faire**

 Créer son poussin g en utilisant le vol de Lévy;

 Calculer la fitness de s et de g;

 Remplacer s par g si f (g) est meilleur que f (s);

Fin pour

 Trouver le meilleur coucou;

Pour chaque coucou s **faire**

 Modifier une fraction P_a de son contenu pour obtenir une nouvelle solution s';

 Evaluer la fitness de s';

 Remplacer s par s' si f (s') est meilleur que f (s);

Fin pour

 Trouver le meilleur coucou;

Fin tant que

Retourner la meilleure solution ;

Fin

L'observation attentive des étapes de l'algorithme CS montre qu'il tourne autour de trois phases: la sélection de la meilleure solution, l'exploitation de la solution par la recherche locale aléatoire et l'exploration de l'espace de recherche par la création aléatoire de nouvelles solutions en utilisant le vol de Lévy [Yang et Deb, 2010]. En s'inspirant d'un des types de la sélection utilisée dans l'algorithme génétique, Yang et Deb ont utilisé la sélection par élitisme afin d'échapper au problème de perdre la meilleure solution et garantir son passage aux itérations suivantes. Dans le CS, l'exploitation autour de la meilleure solution est établie par l'utilisation de la recherche locale comme le montre l'équation 5.3.

$$x(t+1) = x(t) + \alpha \varepsilon(t) \quad (5.3)$$

$\varepsilon(t)$ peut être inspiré de deux types de distributions, la distribution gaussienne et la distribution Lévy. Dans le premier cas (distribution gaussienne), la génération de nouvelles solutions revient à une recherche aléatoire standard. En revanche, dans le cas de la distribution Lévy, le pas du déplacement est grand et il peut être potentiellement efficace [Yang et Deb, 2010]. Cependant, si ce pas est très grand, on risque de s'éloigner trop de la solution actuelle et d'aboutir à des solutions de piètre qualité. C'est dans ce cas là où on peut

très bien comprendre le rôle de la sélection par élitisme qui en retenant les meilleures solutions, permet une exploitation autour du voisinage des meilleures solutions.

La plupart des métaheuristiques utilisent soit des distributions uniformes soit des distributions gaussiennes afin d'explorer l'espace de recherche et générer de nouvelles solutions [Geem et al 2001, Blum et Rilo 2003]. En cas d'espace de recherche très vaste, la distribution Lévy est généralement plus efficace [Yang et Deb, 2010]. C'est la raison pour laquelle, Yang et Deb ont intégré le vol de Lévy dans l'algorithme CS afin de bénéficier de sa capacité de génération des solutions suffisamment diversifiée. La bonne combinaison des trois composantes citées ci-dessus a mené à un efficace algorithme nommé « la recherche coucou: CS ».

Les différentes simulations, les résultats expérimentaux et la comparaison réalisés par les créateurs de l'algorithme CS montrent qu'il est plus générique et plus efficace pour une grande gamme de problèmes d'optimisation. En outre, il (i.e. CS) se caractérise par un bon équilibre entre l'exploitation et l'exploration de l'espace de recherche.

5.3.2. L'algorithme d'optimisation par coucou (COA)

L'optimisation par coucou est une très récente métaheuristique à base de population de solutions. Elle a été proposée en 2011 par Ramin Rajabioun. Le pionnier de l'algorithme d'optimisation par coucou s'est inspiré du comportement des coucous dans leur vie, reproduction et développement pour proposer une nouvelle métaheuristique évolutionnaire. Il résume le principe de sa métaheuristique comme suit.

Le COA commence par une population initiale de coucous. Ces derniers ont quelques œufs à pondre dans des nids d'autres oiseaux hôtes. Certains de ces œufs dont la ressemblance est très grande à celle des œufs de l'oiseau hôte ont la chance de se développer et devenir des coucous matures. Les autres seront tués après être détectés par l'oiseau hôte. Le bon développement et la bonne qualité de certains coucous révèlent l'opportunité de leurs positions dans l'espace de recherche. Ce qui pousse les autres coucous à les rejoindre afin de bénéficier des privilèges offerts par leurs positions. Donc la position dans laquelle plusieurs coucous survivent sera l'objectif d'optimisation du COA. Afin d'augmenter la probabilité de survie de leurs œufs, les coucous cherchent les environnements les plus opportuns pour les pondre. Les œufs sauvés survivent et deviennent plutôt des coucous matures. Ils se regroupent en sociétés, chacun vit dans sa propre région d'habitats. Le meilleur habitat de toutes les sociétés sera la destination des coucous des autres sociétés qui émigrent vers le meilleur

habitat. Ils habiteront quelque part près du meilleur habitat. Considérant le nombre d'œufs de chaque coucou ainsi que la distance entre le coucou et le meilleur habitat, chacun des coucous est dédié d'un rayon de ponte de ses œufs. En effet, chaque coucou pond ses œufs dans des nids aléatoires situés dans son rayon de ponte des œufs. Ce processus continue jusqu'à l'obtention de la meilleure position avec meilleur profit (qualité) et que la plupart des coucous de la population seront regroupés autour de la même position [Rajabioun, 2011].

5.3.2.1. Le principe et les étapes du COA

Généralement la solution d'un problème d'optimisation est représentée par un tableau d'éléments. Différentes expressions ont été utilisées pour désigner cette structure, dans la terminologie de l'algorithme génétique ce tableau est nommé « chromosome », alors que dans celle de l'algorithme d'optimisation par essaim particulaire le vecteur représentant une solution donnée est nommé « position de la particule ». Tandis que, dans le COA, Ramin Rajabioun a proposé le nom « habitat ». Il définit l'habitat par un vecteur de taille $1 \times N_{var}$ représentant la position actuelle du coucou dans un problème d'optimisation de N_{var} dimensions. Ce vecteur est représenté comme suit [Rajabioun, 2011]:

$$\text{habitat} = [x_1, x_2, \dots, x_{N_{var}}]$$

Où, chacune des valeurs des variables ($x_1, x_2, \dots, x_{N_{var}}$) est un nombre flottant.

Dans la vie réelle des coucous, chaque femelle pond un nombre limité d'œufs. En outre, les mamans coucous se déplacent d'une distance maximale de leurs habitats afin de pondre leurs œufs. En s'inspirant du comportement naturel des coucous, Ramin Rajabioun a proposé de limiter le nombre des œufs de chaque coucou par un nombre minimal et un autre maximal. D'autre part, il a proposé de calculer la distance de ponte des œufs pour chaque coucou. Cette distance est nommée « Rayon de ponte des œufs, en anglais: Egg Laying Radius (ELR) ». L'ELR de chaque coucou est calculé en fonction du nombre total des œufs de coucous, le nombre d'œufs du coucou lui-même et des valeurs Var_{hi} et Var_{low} . Où Var_{hi} et Var_{low} représentent des limites de l'intervalle $[Var_{low}, Var_{hi}]$ d'où les valeurs des variables représentant une solution donnée seront tirées. L'équation de calcul de l'ELR est la suivante [Rajabioun, 2011]:

$$ELR = \alpha \times \frac{\text{Nombre des œufs du coucou}}{\text{Nombre total des œufs des coucous}} \times (var_{hi} - var_{low}) \quad (5.4)$$

Où α est un nombre entier dont le rôle est d'équilibrer la valeur maximale d'ELR.

Après avoir calculé l'ELR de chaque coucou, ce dernier commence à pondre ses œufs aléatoirement dans des nids d'autres oiseaux hôtes en fonction de son ELR calculé. La figure 5.1 donne une bonne simulation de ce comportement.

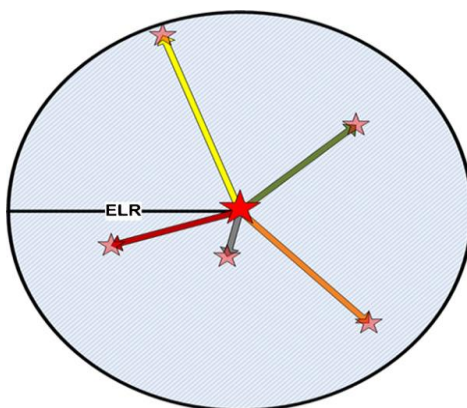


Figure 5.1. Ponte aléatoire d'un œuf dans l'ELR du coucou, l'étoile centrale rouge représente l'habitat initial d'un coucou avec 5 œufs, les étoiles roses représentent les nouveaux nids des œufs [Rajabioun, 2011].

Après la ponte, les œufs coucous dont la ressemblance aux œufs hôtes est faible seront détectés par l'oiseau hôte et rejetés hors nid. Donc après la ponte de nouveaux œufs, une portion p des œufs de mauvaise qualité seront éliminés car ils n'ont aucune chance de se développer ni de grandir. En outre, il est à noter qu'un seul œuf a la chance de survivre dans chaque nid.

Après la ponte, l'éclosion et la sélection des nouveaux coucous, ces derniers bénéficient d'une bonne nourriture qui leur permet de grandir rapidement et devenir des coucous matures. Ils vivent dans leurs sociétés une période définie. Lorsque la période de ponte des œufs approche, les coucous matures migrent vers de nouveaux et de meilleurs habitats où la similarité entre les œufs des oiseaux hôtes et les siens est grande et où les conditions de nourriture de leurs poussins sont bonnes. Les coucous se réunissent en groupe dans différents endroits. La meilleure société dont la valeur du profit (la qualité des solutions) est la meilleure sera sélectionnée pour représenter le point but et la tendance de migration des autres coucous. Dans son papier [Rajabioun, 2011], Ramin Rajabioun ajoute: si les coucous matures vivent partout dans l'espace, il sera difficile de retrouver que tel coucou appartient à quel groupe. Pour résoudre ce problème, le pionnier de l'algorithme COA a proposé de regrouper les coucous en utilisant la méthode de classification K-means. Une fois les coucous seront regroupés, leurs valeurs de profit (fitness) seront calculées afin de trouver le meilleur groupe qui construira le nouvel habitat des autres coucous. Lors du déplacement vers le meilleur groupe, les coucous ne parcourent pas tout le chemin menant à leur destination (le meilleur

habitat), ils parcourent une partie du chemin uniquement avec une certaine déviation. Ce mouvement est représenté clairement par la figure 5.2. Comme le montre la figure 5.2, chaque coucou parcourt uniquement une partie $\lambda\%$ du chemin menant au meilleur habitat. Il se dévie d'un angle de φ radians du chemin menant au meilleur habitat (le point but dans la figure). Ces paramètres (i. e: λ et φ) aident les coucous à détecter d'autres positions dans l'espace de recherche. Pour chaque coucou, λ et φ sont définies comme suit:

$$\lambda \in [0,1]$$

$$\varphi \in [-\omega, \omega]$$

λ est un nombre aléatoire tiré de l'intervalle $[0,1]$. ω est un paramètre de contrôle de la déviation de la meilleure position (habitat). Un $\omega = \pi/6$ semble être suffisant pour une bonne convergence de la population des coucous vers le meilleur habitat [Rajabioun, 2011].

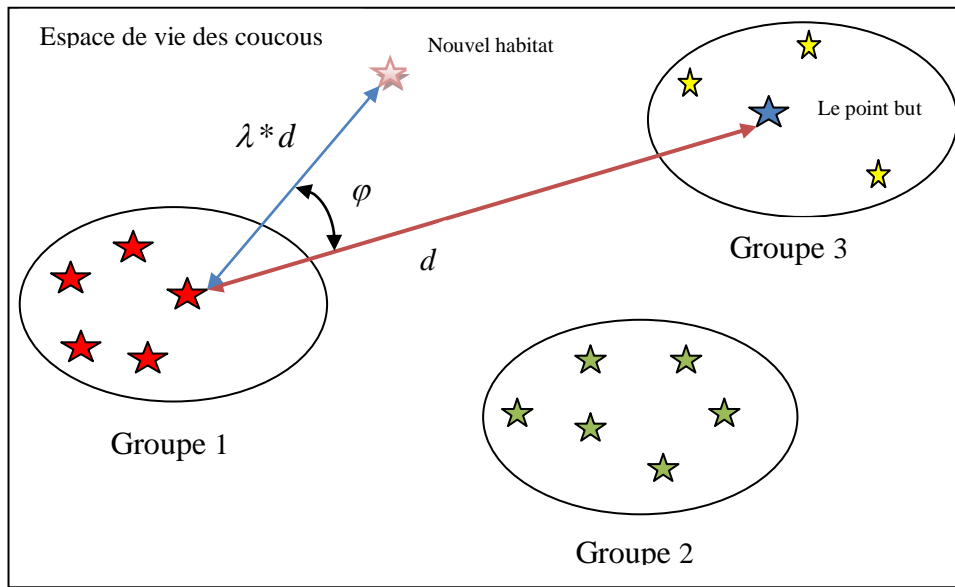


Figure 5. 2. Migration d'un coucou vers le meilleur habitat [Rajabioun, 2011]

La migration des coucous vers le meilleur habitat fait apparaître de nouveaux habitats. Un certain nombre d'œufs va être dédié à chaque nouveau coucou mature. En tenant compte de ce nombre, un ELR va être calculé pour chaque coucou et le processus de ponte des œufs recommence.

Algorithme 5.2. L'algorithme d'optimisation par coucou

Début

Initialiser les habitats des coucous;

Tant que la condition d'arrêt n'est pas satisfaite **faire**

 Dédier quelques œufs à chaque coucou;

 Calculer l'ELR de chaque coucou;

 Pour chaque coucou, créer des œufs en tenant compte de leurs ELR;

 Éliminer les œufs détectés par les hôtes;

 Laisser les œufs éclore et les poussins grandir;

 Évaluer l'habitat de chaque nouveau coucou;

 Limiter le nombre des coucous et éliminer ceux qui vivent dans de mauvais habitats;

 regrouper les coucous, retrouver le meilleur groupe et sélectionner le meilleur habitat;

 Migrer les nouveaux coucous vers le meilleur habitat;

Fin Tant que

Fin

Au cours du cycle de reproduction des coucous, leur nombre augmente mais plusieurs d'entre eux n'auront pas la chance de survivre. En fait, plusieurs facteurs peuvent menacer leur vie. A titre d'exemple: être détecté par leurs hôtes à cause de la faible ressemblance aux œufs hôtes, être tué par des prédateurs ou même à cause du manque de nourriture. Pour limiter le nombre de coucous, Ramin Rajabioun a proposé un paramètre N_{\max} dont le rôle est de contrôler le nombre de coucous au cours du processus d'optimisation pour qu'il ne dépasse pas un nombre maximal. Dans son papier [Rajabioun, 2011], Ramin Rajabioun a proposé de sélectionner les N_{\max} coucous en fonction de leurs fitness. En effet, seuls les N_{\max} coucous avec les meilleures fitness survivent, les autres seront éliminés. Les étapes principales du COA sont présentées dans l'algorithme 5.2.

Dans la section suivante, nous présentons et discutons une étude comparative que nous avons établie entre les deux métaheuristiques CS et COA.

5.4. Discussion et comparaison

En 2009, Xin-She Yang et son collègue Suash Deb ont proposé la nouvelle métaheuristique CS. Après deux ans: en 2011, Ramin Rajabioun a enrichi le nombre des métaheuristiques présentées dans la littérature par la proposition de sa nouvelle métaheuristique qu'il a nommé COA.

CS et COA sont deux métaheuristiques basées population de solutions. Elles sont inspirées du comportement d'un type spécial d'oiseaux parasites de couvée nommés Coucou. Chacune d'entre elles (CS et COA) commence la recherche par une population de solutions. Au cours de la recherche et de l'optimisation, de nouvelles solutions apparaissent et d'autres disparaissent. Dans l'algorithme CS, la nouvelle solution $x(t+1)$ est créée en fonction de la solution actuelle $x(t)$ et du vol de Lévy. A l'opposé, dans l'algorithme COA la nouvelle solution $x(t+1)$ est créée en fonction de la solution actuelle $x(t)$, l'ERL du coucou (solution) plus les paramètres λ , φ et ω .

	CS	COA
Auteurs	Xin-She Yang et Suash Deb	Ramin Rajabioun
Année d'apparition	2009	2011
Type de métaheuristique	à base de population de solutions	
Source d'inspiration	La vie et le comportement des coucous	
Similitude avec le comportement naturel des coucous	Quelques notions	Grande similitude
Espace de recherche	Continu	
Paramètres	<ul style="list-style-type: none"> ▪ Taille de la population initiale ▪ P_a 	<ul style="list-style-type: none"> ▪ Taille de la population initiale ▪ Nombre des œufs (Min et Max) ▪ Nombre de coucous par groupe ▪ Nombre maximum de coucous de la population ▪ λ, φ et ω
Taille de la population	Inchangeable	Changeable (augmente)
Création des nouvelles solutions	En fonction de: <ul style="list-style-type: none"> ▪ la solution actuelle ▪ Le vol de Lévy 	En fonction de: <ul style="list-style-type: none"> ▪ la solution actuelle ▪ ELR ▪ λ, φ et ω
Sélection des coucous de la nouvelle génération	Selon leurs qualités (fitness): Sélection par élitisme	
Destruction des coucous	Selon leurs qualités	<ul style="list-style-type: none"> ▪ Selon leurs qualités ▪ Selon leurs représentations (la solution elle même)

Tableau 5. 1. Comparaison entre le CS et le COA

Pour sélectionner les solutions de la nouvelle génération, les auteurs des deux méthodes ont utilisé la sélection par élitisme qui consiste à opter pour les meilleures solutions pour qu'elles représentent les membres de la nouvelle génération (population) et éliminer les solutions de mauvaise qualité. En outre, dans le CS la destruction des solutions est effectuée

dans un seul cas. C'est le cas où la qualité de la nouvelle solution (le coucou poussin) est meilleure que celle de sa dérivée (le coucou parent). A l'opposé, dans le COA l'élimination des solutions est effectuée dans deux cas. Le premier, c'est lorsque une solution x figure dans la population plus qu'une fois. Dans ce cas là, les exemplaires d'une solution seront éliminés pour échapper à la redondance des solutions de la population. Le deuxième cas c'est après classification (tri) des solutions selon leur qualité. Les mauvaises solutions seront éliminées pour ne pas dépasser la taille maximale de la population et tomber dans le problème de l'explosion démographique de la population.

Dans l'algorithme CS, le nombre des solutions constituant la population est stable et fixe. En fait, l'algorithme entame la recherche par N solutions et converge par N solutions. En revanche, dans le COA la taille de la population n'est pas stable. Le COA entame la recherche par une population de taille N et converge par une autre de taille M , où $M > N$. Car dans le COA chaque coucou pond plusieurs œufs à la fois (par itération) et ils peuvent tous survivre. Alors que dans le CS le coucou pond un seul œuf à la fois qui selon sa qualité peut survivre en remplaçant son parent ou il sera détruit à cause de sa piètre qualité comparée à celle de son parent. En outre, l'observation des étapes des deux algorithmes nous permet de conclure que le COA présente beaucoup de simulations du comportement naturel des coucous (le nombre des œufs de chaque coucou, l'habitat, la migration, le regroupement des coucous en société...).

Essentiellement, à part la taille de la population, nous distinguons dans l'algorithme CS un seul paramètre en jeu, c'est le paramètre P_a . Comparé avec d'autres algorithmes comme l'algorithme génétique ou l'optimisation par essaim particulaire, le CS nécessite peu de paramètres à manipuler ce qui offre plus de souplesse d'utilisation pour les utilisateurs. En fait, ceci permet à l'utilisateur de gagner du temps en s'affranchissant de la phase d'étude et du choix convenable du grand nombre des paramètres de l'algorithme. En outre, les pionniers de l'algorithme CS ont montré que le taux de convergence de leur algorithme est insensible au choix du paramètre P_a . Cela veut dire qu'on n'a pas besoin d'affiner ce paramètre pour un problème donné [Yang et Deb, 2010]. En observant les paramètres de l'algorithme COA, nous distinguons essentiellement six paramètres en jeu, à part la taille de la population:

- Le **nombre des œufs** (minimal et maximal) de chaque coucou : chaque coucou pond un certain nombre de coucous, ce dernier doit être limité par une valeur minimale et une autre maximale.

- *Les paramètres λ , φ et ω* nécessaires lors du déplacement du coucou vers le meilleur habitat.
- *Le nombre de coucous par groupe*: après leur déplacement les coucous se regroupent pour construire des sociétés où chacune est composée d'un certain nombre de coucous.
- *Le nombre maximal des coucous de la population*. En fait, au cours du processus de reproduction des coucous, leur nombre augmente. Ramin Rajabioun a proposé de limiter le nombre maximal de la population par un nombre N_{\max} afin d'échapper au problème d'explosion de la population.

Malgré la différence remarquable entre les étapes du CS et du COA, les résultats expérimentaux présentés par leurs auteurs ont prouvé l'efficacité et la performance de chacune d'entre elles. Cependant, les deux algorithmes ont été proposés pour opérer dans des domaines de recherche continus où la solution est représentée par des réels [Rajabioun, 2011] [Gherboudj et al, 2012].

Vu leur très jeune âge, le CS et le COA ouvrent plusieurs portes de recherche et de contribution devant la communauté des chercheurs, notamment ceux qui travaillent sur les méthodes d'optimisation et leurs applications pour la résolution des problèmes d'optimisation académiques et/ou industriels. De notre part, nous avons proposé une version binaire de la méthode CS afin de faire face aux problèmes d'optimisation binaires où les variables sont représentées par des bits. Le tableau 5.1 présente un résumé de l'étude comparative que nous avons élaborée entre les deux métaheuristiques CS et COA.

5.5. La méthode proposée: une version binaire de la recherche coucou

Les problèmes d'optimisation peuvent être classés en deux classes principales: la classe des problèmes d'optimisation continus et la classe des problèmes d'optimisation discrets. Dans la première classe de problèmes (i.e. les problèmes continus), la solution est représentée par des nombres réels. Cependant, dans la deuxième classe la solution est représentée par des nombres entiers. Une autre classe un peu spéciale peut être citée, elle regroupe un type spécial des problèmes d'optimisation, c'est la classe des problèmes d'optimisation binaires. Cette dernière est une sous classe de la deuxième classe principale des problèmes d'optimisation (i.e. les problèmes discrets). Dans les problèmes d'optimisation binaires, la solution est représentée par des bits (0 et 1). Maints problèmes d'optimisation peuvent être modélisés en tant que problèmes d'optimisation binaires comme le problème d'ordonnancement Flow-shop

[Liao et al, 2007], le problème d'ordonnancement Job-shop [Pongchairerks, 2009], les problèmes de routage [Zhan et Zhang, 2009], le problème du sac à dos [Gherboudj et Chikhi, 2011] et ses variantes tels que le problème du sac à dos multidimensionnel [Kong et Tian, 2006], le problème du sac à dos quadratique [Julstrom, 2005], le problème du sac à dos multidimensionnel quadratique [Singh et Baghel, 2007] ...etc.

La version originale de l'algorithme de la recherche coucou est basée sur le mécanisme du vol de Lévy. En effet, elle opère dans des espaces de recherche continus et permet d'aboutir à des solutions de type réel. Cependant, les problèmes d'optimisation binaires nécessitent des solutions de type binaire. Pour ce type de problème, les solutions de type réel ne sont pas acceptées, elles sont illégales et infaisables. Une des solutions du problème posé par le type de la solution consiste à convertir les composants de la solution du réel vers le binaire. Dans le but d'étendre l'algorithme CS aux espaces binaires, nous en avons proposé une version binaire que nous avons nommée BCS (voir figure 5.3).

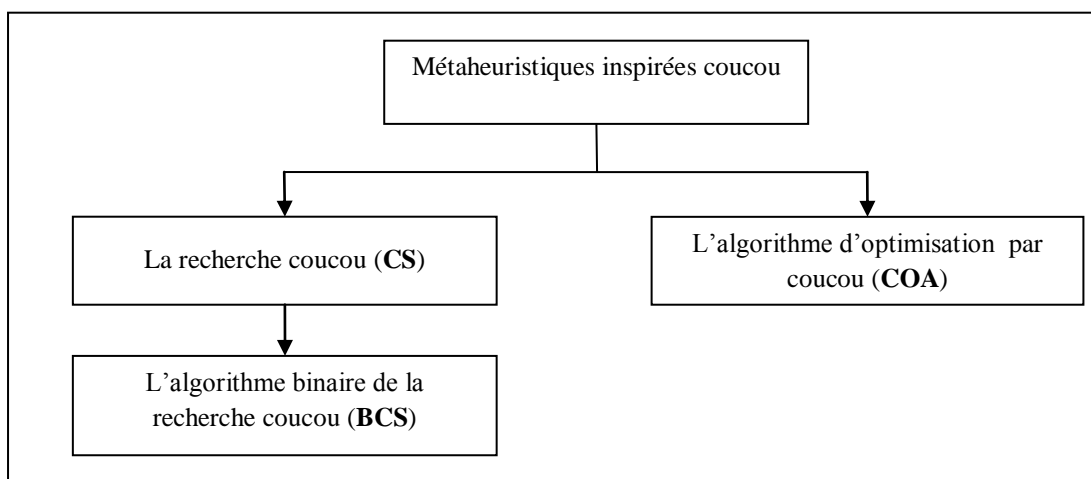


Figure 5.3. Les métaheuristiques inspirées coucou

La différence principale entre le CS originel et notre BCS (Binary Cuckoo Search) est que dans le CS la solution finale du problème est représentée par des réels, alors que dans le BCS elle est représentée par des bits. Le BCS se caractérise par l'utilisation de la fonction Sigmoidale afin de générer des valeurs binaires. Le but de cette contribution est double, à part la proposition d'une version binaire de l'algorithme CS, notre deuxième but est de prouver l'efficacité de notre version dans le traitement et la résolution des problèmes d'optimisation académiques difficiles. Raison pour laquelle, nous avons utilisé le BCS pour la résolution de quelques instances du problème du sac à dos et aussi des instances du problème de sac à dos

multidimensionnel qui sont des problèmes d'optimisation académiques combinatoires classés parmi les problèmes NP-Difficiles.

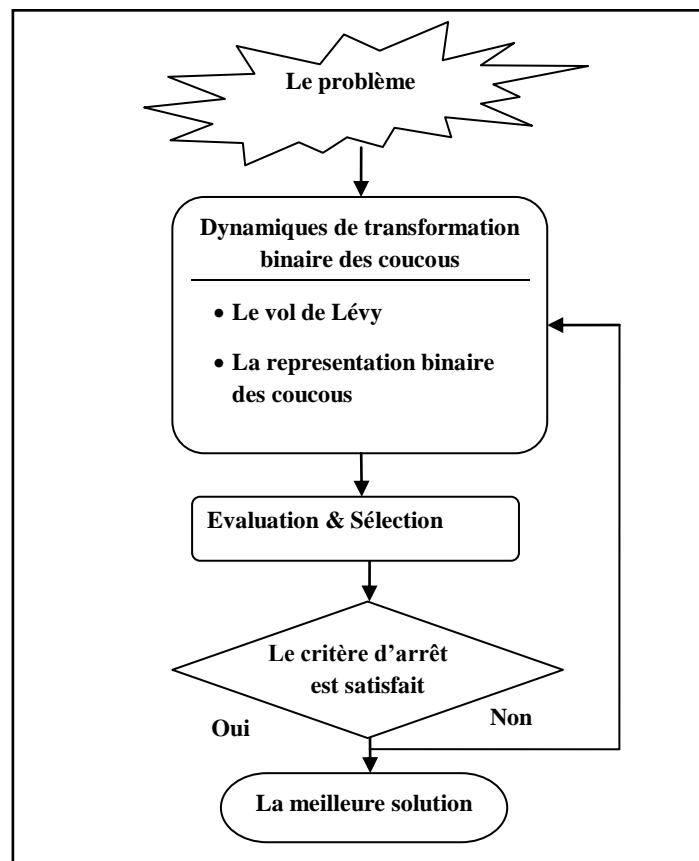


Figure 5.4. L'architecture du BCS

5.5.1. L'architecture générale du BCS

L'architecture générale du BCS est composée de deux modules essentiels. Le premier module concerne les dynamiques principales de génération des coucous (solutions) binaires. En fait, ce module est composé de deux opérations essentielles: la création des coucous par le vol de Lévy et la transformation binaire des coucous en utilisant la fonction Sigmoid. Ces deux opérations combinent l'algorithme CS et la fonction Sigmoid afin d'obtenir une version binaire de la recherche par coucou. Après création des nouveaux coucous (nouvelles solutions) en utilisant le vol de Lévy, la fonction Sigmoid sera appelée en deuxième étape afin de calculer les probabilités de transformation en bit de chaque coucou. Par conséquent, en se basant sur les probabilités de transformation de chaque composante de la solution en bit, les valeurs binaires qui composeront la solution seront calculées. Le deuxième module contient la fonction fitness et l'opérateur de sélection. La sélection utilisée est similaire à la sélection par élitisme utilisée dans l'algorithme génétique. En fait, les coucous représentants

de la nouvelle génération seront sélectionnés par élitisme en fonction de leurs fitness. La figure 5.4 représente l'organigramme de l'architecture du BCS. Cette dernière sera bien éclaircie dans ce qui suit.

5.5.2. La représentation binaire de la solution

L'objectif principal de l'algorithme BCS est de faire face aux problèmes qui peuvent être représentés par des structures binaires. Il permet la transformation de la solution x_i d'un espace de recherche réel vers un espace de recherche binaire pour obtenir sa représentation binaire. Pour réaliser cet objectif, nous avons proposé de limiter la solution x_i dans l'intervalle $[0, 1]$ en utilisant la fonction Sigmoïde (voir équation 5.5).

$$Sig(x_i) = \frac{1}{1 + e^{-x_i}} \quad (5.5)$$

$Sig(x_i)$ représente la probabilité de transformation de la valeur réelle vers une valeur binaire x'_i . Elle représente la probabilité pour que le bit x'_i prenne la valeur 1.

Algorithme 5.3. L'algorithme BSR

Entrée: Une représentation réelle de la solution x_i

Sortie: Une représentation binaire de la solution x'_i

Début

Pour (i allant de 1 à (Taille du problème)) **faire**

$$Sig(X_i) = \frac{1}{1 + e^{-x_i}};$$

Si (le nombre aléatoire $r < Sig(X_i)$) **alors**

$$x'_i = 1;$$

Sinon

$$x'_i = 0;$$

Fin Si

Fin Pour

Fin

Afin d'obtenir la représentation binaire de la solution x_i , nous devons générer un nombre aléatoire r de l'intervalle $[0, 1]$ pour chaque dimension i de la solution x . Ensuite, nous comparons la valeur r avec la valeur $Sig(x_i)$ de la solution x_i comme le montre l'équation 5.6.

Si le nombre r est inférieur à $\text{Sig}(x_i)$ de la solution x_i , x_i' prendra la valeur 1. Sinon x_i' prendra la valeur 0.

$$x_i' = \begin{cases} 1 & \text{si } r < \text{Sig}(X_i), r \in [0,1] \\ 0 & \text{sinon} \end{cases} \quad (5.6)$$




Population initiale	$\left\{ \begin{array}{cccccc} -3.0480 & 0.0265 & 3.6780 & -1.2769 & 0.9055 & -0.4691 \\ 0.6244 & 0.0180 & 0.7031 & 0.3270 & 0.9737 & 0.4604 \\ 0.5901 & 0.0401 & 0.5346 & 0.1247 & 0.9758 & 0.3754 \end{array} \right\}$
	
Chances de transformation	$\left\{ \begin{array}{cccccc} 0.0453 & 0.5066 & 0.9753 & 0.2181 & 0.7121 & 0.3848 \\ 0.6512 & 0.5045 & 0.6689 & 0.5810 & 0.7259 & 0.6131 \\ 0.6434 & 0.5100 & 0.6306 & 0.5311 & 0.7263 & 0.5928 \end{array} \right\}$
	
Les nombres aléatoires r	$\left\{ \begin{array}{cccccc} 0.1978 & 0.5000 & 0.6099 & 0.8055 & 0.2399 & 0.4899 \\ 0.0305 & 0.4799 & 0.6177 & 0.5767 & 0.8865 & 0.1679 \\ 0.7441 & 0.9047 & 0.8594 & 0.1829 & 0.0287 & 0.9787 \end{array} \right\}$
	
La population binaire	$\left\{ \begin{array}{cccccc} 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{array} \right\}$

Figure 5.5. Exemple d'exécution de l'algorithme BSR

Supposant la solution x_i encodée par des nombres réels, la fonction Sigmoidale est utilisée pour calculer un ensemble de probabilités dont chacune représente la chance de transformation du bit i vers la valeur 1. En fait, la chance de transformation de chaque valeur permet la génération de la solution binaire x_i' . Supposons par exemple: que nous sommes devant un problème du sac à dos avec un nombre d'objets égal à 6 (i.e. la taille N du problème est égale à 6) ; $x_i = [2.314, -3.4510, 1.9412, 0.3498, -1.5634, 3.8461]$ est la solution obtenue par l'algorithme CS ; $\text{Sig}(x_i) = [0.9100, 0.0307, 0.8745, 0.5866, 0.1732, 0.9791]$ est l'ensemble des chances de transformation (probabilités) de chaque bit x_i' obtenus par la fonction Sigmoidale: $\text{chance}_i = [91.00\%, 3.07\%, 87.45\%, 58.66\%, 17.32\%, 97.91\%]$; afin de

générer la solution binaire, nous devons dans cette étape générer 6 nombres aléatoires à partir de l'intervalle $[0, 1]$; soit $r = [0.8147, 0.1270, 0.9134, 0.6324, 0.0975, 0.5469]$. Suivant les instructions de l'équation 5.6, le premier, le cinquième et le sixième bit de la solution binaire vont prendre la valeur 1 car leurs chances de transformation (0.9100, 0.1732, 0.9791, respectivement) sont supérieures à leurs nombres aléatoires (0.8147, 0.0975, 0.5469, respectivement).

Algorithme 5.4. L'algorithme proposé (BCS)

Début

Initialiser une population de N coucous (solutions);

Tant que (le critère d'arrêt n'est pas satisfait) **faire**

Pour chaque coucou s **faire**

 Créer son poussin g en utilisant le vol de Lévy;

Obtenir la représentation binaire du coucou s en utilisant l'algorithme BSR;

Obtenir la représentation binaire du poussin g en utilisant l'algorithme BSR;

 Evaluer la qualité de s et de g ;

 Remplacer s par g si $f(g)$ est meilleur que $f(s)$;

Fin pour

 Trouver le meilleur coucou;

Pour chaque coucou s **faire**

 Modifier une fraction P_a de son contenu pour Obtenir une nouvelle solution s' ;

Obtenir la représentation binaire de la nouvelle solution s' en utilisant l'algorithme BSR;

 Evaluer la fitness de s'

 Remplacer s par s' si $f(s')$ est meilleur que $f(s)$;

Fin pour

 Trouver le meilleur coucou;

Fin tant que

Fin

Cependant, le deuxième, le troisième et le quatrième bit vont prendre la valeur 0, car leurs chances de transformation (0.0307, 0.8745, 0.5866, respectivement) sont inférieures à leurs nombres aléatoires (0.1270, 0.9134, 0.6324, respectivement). En résultat, $x_i' = [1, 0, 0, 0, 1, 1]$ sera la représentation binaire de la solution x_i . Ce qui veut dire que les objets sélectionnés sont 1, 5 et 6. L'algorithme permettant la transformation binaire des solutions (le BSR : Binary

Solution Representation) est présenté dans l'algorithme 5.3. Ainsi, un exemple d'exécution de l'algorithme BSR sur une population de trois coucous et un problème de taille six (i.e. six objets) est présenté dans la figure 5.5.

5.5.3. L'algorithme proposé

Maintenant, nous décrivons comment la combinaison entre l'algorithme CS et l'algorithme BSR a donné naissance à une nouvelle version de la métaheuristique CS et comment cette combinaison peut trouver des solutions binaires à des problèmes d'optimisation binaires. L'ensemble des étapes du BCS tournent autour de trois opérations: la sélection de la meilleure solution, l'exploitation locale par la recherche locale et l'exploitation globale par le vol de Lévy. Les étapes de l'algorithme BCS sont représentées dans l'algorithme 5.4.

Comme n'importe quelle métaheuristique basée population, la première étape dans l'algorithme BCS consiste à initialiser les paramètres de l'algorithme. Un avantage principal de l'algorithme BCS est qu'il nécessite peu de paramètres comparé avec d'autres métaheuristicues à base de population de solutions comme le PSO. En fait, dans le BCS nous ne distinguons que deux paramètres essentiels à initialiser: la taille de la population N et la fraction P_a des composants des solutions qui doivent être rejetés et remplacés par d'autres composants dans le but d'améliorer les qualités des solutions courantes. Dans la deuxième étape, un ensemble de N nids (coucous) hôtes sera créé dans des positions aléatoires de l'espace de recherche afin de représenter des solutions initiales. Il est à noter que dans le but de réduire le temps de convergence de l'algorithme, il est recommandé d'entamer la recherche par une population variée composée d'un mélange de solutions de bonne qualité et d'autres de piètre qualité. En effet, le programmeur peut utiliser des heuristiques permettant la construction d'une bonne population initiale. L'algorithme progresse selon un certain ordre d'un nombre d'instructions selon la dynamique BCS. Durant chaque itération, les tâches suivantes seront réalisées: de nouvelles solutions (coucous) seront créées suivant le mécanisme du vol de Lévy qui permet la modélisation mathématique des déplacements aléatoires basés sur des probabilités. L'étape suivante consiste à évaluer les nouvelles solutions. Afin de le faire, un appel de l'algorithme BSR (Algorithme 5.3) sera indispensable pour obtenir des solutions binaires qui représentent des solutions potentielles du problème traité. Après cette étape, nous remplaçons le coucou parent (la solution actuelle) par son poussin si ce dernier est de meilleure qualité. Ensuite nous passons à améliorer la qualité

d'une portion P_a des composants des solutions par une recherche locale aléatoire permettant l'exploitation des solutions.

Après l'étape de l'exploitation des solutions, nous refaisons appel à l'algorithme BSR afin d'obtenir des représentations binaires des solutions obtenues qui seront évaluées afin de sélectionner les meilleures pour qu'elles puissent représenter les membres de la nouvelle génération et éliminer certaines à cause de leurs mauvaises qualités. Dans le BCS, la sélection des meilleures solutions est similaire à la sélection par élitisme utilisée dans l'algorithme génétique, qui assure le passage de la meilleure solution entre les différentes itérations de l'algorithme. Enfin, la meilleure solution sera mise à jour si une autre solution de meilleure qualité est trouvée. La procédure de recherche et d'optimisation de la meilleure solution continue jusqu'à la satisfaction du critère d'arrêt.

5.6. Les résultats expérimentaux

Afin de tester et de prouver la performance de notre algorithme (BCS), nous l'avons utilisé pour résoudre deux problèmes d'optimisation appartenant à la classe des problèmes NP-Difficiles: Le problème du sac à dos (KP) et le problème du sac à dos multidimensionnel (MKP). Trois expériences ont été réalisées. Dans la première expérience, nous avons utilisé notre BSC pour la résolution des instances du problème du sac à dos. D'une part, nous l'avons appliqué sur des instances de petites tailles qui ont été utilisées par Dexuan Zou et ses collègues dans [Zou et al, 2011] pour pouvoir comparer nos résultats avec ceux obtenus par Dexuan Zou et ses collègues qui ont proposé un algorithme basé sur l'algorithme de la recherche par harmonies (Harmony Search) pour résoudre le problème du sac à dos. D'autre part, nous avons testé et comparé nos résultats avec ceux obtenus par l'algorithme binaire standard d'optimisation par essaim de particules (BPSO) [Kennedy et Eberhart, 1997] qui a un point commun avec notre BCS. En fait, les deux algorithmes (le BCS et le BPSO) utilisent la fonction Sigmoidale pour générer des solutions binaires. Dans cette expérience, nous avons utilisé d'autres instances de grandes tailles du problème de sac à dos, les mêmes instances proposées et utilisées par nous même dans le papier [Gherboudj et Chikhi, 2011]. Nous avons opté pour sept instances de tailles différentes où les poids et les valeurs des objets sont sélectionnés aléatoirement. Les tailles des instances utilisées sont: 120, 200, 500, 700, 900, 1000 et 2000. Pour toutes ces instances, la capacité du sac à dos est calculée en utilisant la formule 5.7. Où le facteur $\frac{3}{4}$ indique que 75% des objets candidats peuvent être sélectionnés.

$$C = \frac{3}{4} \sum_{i=1}^N w_i \quad (5.7)$$

L'instance	La taille	La solution du BCS	La solution du NGHS [Zou et al, 2011]
f1	10	295	295
f2	20	1024	1024
f3	4	35	35
f4	4	23	23
f5	15	481.0694	481.0694
f6	10	52	50
f7	7	107	107
f8	23	9767	9761
f9	5	130	130
f10	20	1025	1025

Tableau 5. 2. Les résultats expérimentaux du BCS et NGHS [Zou et al, 2011] avec quelques instances KP.

Dans la deuxième expérience, nous avons évalué la performance de notre BCS avec des instances du sac à dos multidimensionnel que nous avons tirées de la bibliothèque OR-Library. Dans un premier pas, nous avons testé le BCS sur quelques instances de petites tailles tirées du benchmark nommé mknapi. Ensuite, nous avons testé le BCS sur des instances de grandes tailles tirées des benchmarks nommés mknapcb1 et mknapcb4. Nous avons utilisé cinq instances de mknapcb1 (5, 100) avec 5 contraintes (dimensions) et 100 objets et aussi cinq instances de mknapcb4 (10, 100) avec 10 contraintes (dimensions) et 100 objets. Les résultats obtenus ont été comparés avec la solution optimale (best known) déclarée dans OR-Library, les résultats obtenus par le standard PSO basé sur la fonction de pénalité (dénnoté: PSO-P) qui sert à faire face aux problèmes avec contraintes [Kong et Tian, 2006] et aussi avec la version quantique de l'algorithme CS dénotée QICSA [Layeb, 2011]. Finalement, nous avons essayé d'évaluer nos résultats statistiquement en utilisant le test statistique du Friedman afin de tester la signifiante de la différence des résultats obtenus par notre version et les autres méthodes utilisées dans la comparaison.

Le tableau 5.2 représente les résultats obtenus par notre algorithme et par l'algorithme de la recherche par harmonies (NGHS) sur dix instances de tailles différentes. La première colonne indique le nom de l'instance, la deuxième colonne indique la taille de l'instance (i.e. le nombre des objets). La troisième colonne représente les résultats obtenus par le BCS et la dernière colonne représente les résultats obtenus par l'algorithme NGHS. L'observation des résultats présentés dans le tableau 5.2 montre que la performance de notre algorithme dépasse celle du NGHS dans deux cas: le cas de l'instance f6 et le cas de l'instance f8. Dans les autres cas (les autres instances), le BCS et le NGHS ont les mêmes résultats.

Le tableau 5.3 représente les résultats obtenus par le BCS et le BPSO sur des instances du problème du sac à dos. La colonne 1 représente la taille du problème (i.e. L'instance), la deuxième et la troisième colonne représentent les résultats obtenus par l'algorithme BCS et par le BPSO respectivement. Le but de cette expérience est de comparer la performance de l'algorithme BCS avec celle du BPSO. Les résultats expérimentaux montrent que la performance de l'algorithme BCS dépasse celle du BPSO, spécialement avec les instances de grandes tailles. En outre, le test statistique du Friedman (présenté dans la figure 5.6) montre que malgré la différence entre les résultats obtenus par le BCS et le BPSO n'est pas significative, mais la performance du BCS est meilleure que celle du BPSO.

L'instance	La solution du BCS	La solution du BPSO
120	4210	4296
200	6989	7456
500	16364	13116
700	22734	18276
900	29322	22857
1000	31679	24933
2000	61497	47674

Tableau 5. 3. Les résultats expérimentaux du BCS et BPSO avec quelques instances KP.

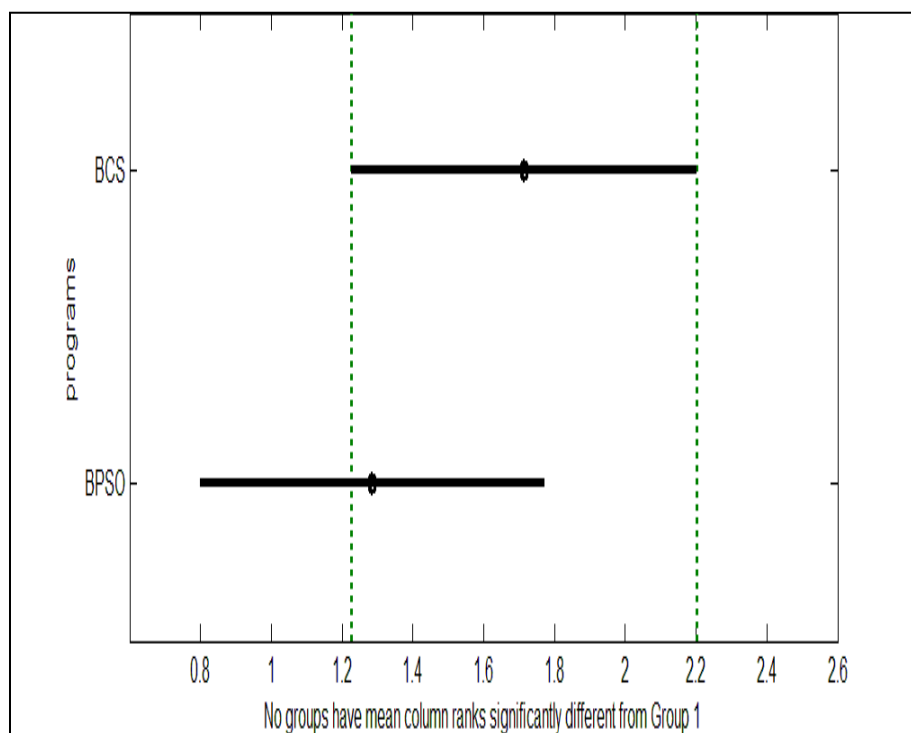


Figure 5.6. Le résultat du test de Friedman entre le BCS et le BPSO sur des instances KP

L'instance	n	m	La solution optimale	La solution du BCS
mknapi1	6	10	3800	3800
mknapi2	10	10	8706,1	8706,1
mknapi3	15	10	4015	4015
mknapi4	20	10	6120	6120
mknapi5	28	10	12400	12400
mknapi6	39	5	10618	10618
mknapi7	50	5	16537	16537

Tableau 5.4. Les résultats expérimentaux sur des instances MPK de l'instance mknapi

Les tableaux 5.4 et 5.5 représentent les résultats obtenus par le BCS avec quelques instances du problème d'optimisation du sac à dos multidimensionnel. Le tableau 5.4 représente les résultats obtenus par le BCS sur les sept instances du benchmark mknapi qui sont classées parmi les instances faciles à résoudre. La première colonne, représente le nom de l'instance. La deuxième colonne et la troisième colonne indiquent le nombre des objets et les dimensions du sac à dos respectivement. La quatrième colonne et la cinquième colonne

indiquent la solution optimale et la solution obtenue par le BCS respectivement. La lecture dans le tableau 5.4 montre que notre algorithme est capable de trouver la solution optimale de toutes les instances du mknapi.

Finalement, le tableau 5.5 représente les résultats obtenus par les algorithmes BCS, PSO-P et QICSA sur des instances des benchmarks mknapi1 et mknapi4 qui sont classées parmi les instances difficiles à résoudre. La colonne 1 représente le nom du benchmark. La colonne 2 représente la taille du problème. La troisième colonne indique la solution optimale déclarée dans OR-Library. Les colonnes 4, 5 et 6 indiquent les meilleures solutions obtenues par le BCS, le PSO-P et le QICSA respectivement. Les résultats obtenus montrent que pour tous les cas, l'algorithme BCS donne de bons résultats comparés à ceux obtenus par le PSO-P. En outre, la performance de l'algorithme BCS dépasse celle de l'algorithme QICSA dans la plupart des cas (i.e. 5.100.00, 5.100.01, 5.100.04, 10.100.00, 10.100.01, 10.100.03 et 10.100.04).

Le test statistique de Friedman présenté dans la figure 5.7 représente une comparaison des résultats obtenus par notre algorithme (le BCS), le PSO-P et le QICSA avec la solution optimale (best known). Selon la figure 5.7, notre algorithme est classé dans la deuxième position (après la solution optimale). Ce test statistique montre que la différence entre la solution optimale et les résultats obtenus par l'algorithme PSO-P et l'algorithme QICSA est significative. Cependant, la différence entre la solution exacte et la solution du BCS n'est pas très significative statistiquement (il n'y a pas une grande différence entre les deux résultats). Par conséquent, les résultats obtenus confirment que la performance de notre algorithme dépasse celles des algorithmes PSO-P et QICSA. D'autre part, ces résultats prouvent que notre algorithme est capable d'aboutir à des solutions encourageantes et de bonne qualité comparés avec la solution optimale déclarée dans OR-Library. En effet, la performance de l'algorithme BCS peut être améliorée considérablement par l'introduction des heuristiques ou des opérateurs spécifiques au problème traité qui permettent une recherche plus efficace en prenant en compte les contraintes du problème traité, comme l'opérateur de réparation de la solution du problème du sac à dos multidimensionnel qui a été utilisé par Gupta et Garg dans [Gupta et Garg, 2009].

Benchmark	La taille du problème	La solution optimale	La solution du BCS	La solution PSO-P [Kong et Tian, 2006]	La solution QICSA [Layeb, 2011]
mknapcb1	5.100.00	24381	23510	22525	23416
	5.100.01	24274	22938	22244	22880
	5.100.02	23551	22518	21822	22525
	5.100.03	23534	22677	22057	22727
	5.100.04	23991	23232	22167	22854
mknapcb4	10.100.00	23064	21841	20895	21796
	10.100.01	22801	21708	20663	21348
	10.100.02	22131	20945	20058	20961
	10.100.03	22772	21395	20908	21377
	10.100.04	22751	21453	20488	21251

Tableau 5. 5. Les résultants expérimentaux sur des instances MPK des instances mknapcb1 et mknapcb4

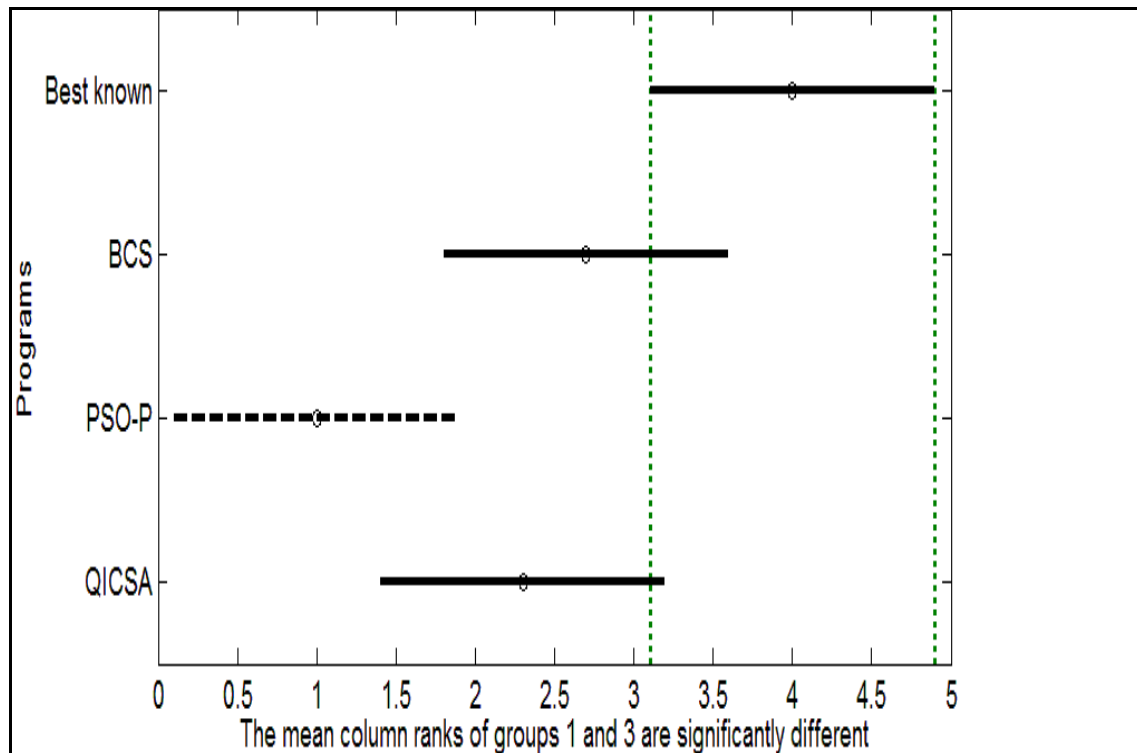


Figure 5.7. Le résultat du test de Friedman entre la solution exacte, le BCS, le PSO-P et le QICSA sur des instances MKP

5.7. Discussion

L'algorithme CS est une nouvelle métaheuristique à base de population de solution. Vu son jeune âge, elle n'a pas été bien exploitée comme les autres métaheuristiques. En fait, peu de travaux basés CS sont présentés dans la littérature. Jusqu'à Avril de l'année 2012, aucune version binaire de l'algorithme CS n'a été proposée pour faire face aux problèmes d'optimisation binaires. Notre objectif principal dans cette contribution était de prouver la capacité et l'efficacité de l'algorithme CS dans la résolution des problèmes d'optimisation combinatoires binaires. C'est la raison pour laquelle nous avons songé à introduire la technique de la fonction Sigmoïde dans le corps de l'algorithme CS. Ceci nous a mené à créer la première version binaire de l'algorithme CS que nous avons noté: BCS. Notre version permet de faire face aux problèmes binaires d'optimisation. Afin de tester et prouver la performance de notre algorithme, nous l'avons utilisé pour la résolution de deux problèmes d'optimisation combinatoire binaires NP-Difficiles: le problème du sac à dos et le problème du sac à dos multidimensionnel. Ces deux problèmes académiques sont très importants dans la modélisation d'une grande gamme de problèmes industriels. En outre, la résolution du problème du sac à dos et la résolution du problème du sac à dos multidimensionnel ont largement fait l'objectif de maintes recherches vu leur importance et leur complexité. Après tests et comparaisons, nous pouvons dire que les résultats obtenus sont très encourageants. En fait, ils prouvent l'efficacité du BCS dans le traitement des problèmes du sac à dos en tant que modèle de problèmes d'optimisation binaires. Il est clair que l'algorithme BCS est plus efficace dans la recherche de l'optimum global comparé avec l'algorithme de l'optimisation par essaim particulaire PSO-P, l'algorithme de la recherche d'harmonie NGHS et la version quantique de la recherche coucou QICSA. L'algorithme proposé (le BCS) et les deux modèles de l'algorithme d'optimisation par essaim particulaire (i.e. le BPSO et le PSO-P) utilisent la même technique pour la génération des solutions binaires, mais les résultats expérimentaux montrent que l'algorithme BCS est plus performant que le BPSO et le PSO-P. L'efficacité de notre approche est expliquée par le bon équilibre entre l'exploitation de la solution par la recherche locale aléatoire autour de la solution et aussi par l'exploration de l'espace de recherche en utilisant le mécanisme du vol de Lévy. La bonne combinaison entre le vol de Lévy et la recherche locale aléatoire permet à l'algorithme de bien explorer l'espace de recherche et bien localiser les solutions potentielles. La diversité de l'algorithme BCS est assurée par l'utilisation de la sélection par élitisme qui permet de garantir le passage des bonnes solutions aux nouvelles générations. En outre, la détection des œufs étrangers avec

une certaine probabilité permet de remplacer quelques composants des solutions par d'autres composants, dans le but de créer des solutions de meilleures qualités. Un autre avantage qui caractérise notre algorithme est lié au nombre de ses paramètres. En fait, à part la taille de la population le P_a est le seul paramètre qui doit être sélectionné par l'utilisateur, ce qui donne plus de souplesse et de simplicité d'utilisation de l'algorithme comparé à d'autres algorithmes notamment le BPSO. Par conséquent, l'algorithme proposé est plus générique et il peut être facilement implémenté pour résoudre d'autres problèmes d'optimisation binaires.

5.8. Conclusion

Dans ce chapitre, nous avons présenté notre contribution avec la nouvelle métaheuristique CS. Avant de relater son principe, nous avons présenté un aperçu sur le style de vie des coucous dans la nature. Ensuite, nous avons élucidé les étapes et le principe des deux métaheuristicues inspirées du comportement des coucous dans leur monde naturel. En outre, nous avons établi une petite étude comparative entre les deux métaheuristicues inspirées coucou. Après, nous avons expliqué le principe de notre contribution qui consiste à proposer une version binaire de l'algorithme de la recherche coucou que nous avons nommée BCS (Binary Cuckoo Search).

L'objectif principal de la version proposée est de faire face aux problèmes d'optimisation binaires. En fait, la version originale de l'algorithme CS a été proposée pour résoudre des problèmes d'optimisation continus. Cependant, les problèmes d'optimisation ne sont pas tous de type continu, ils peuvent aussi être de type discret ou binaire. La caractéristique principale de l'algorithme proposé est qu'il utilise la fonction Sigmoidale afin de transformer les valeurs réelles de la solution vers des valeurs binaires. Afin de prouver l'efficacité de notre algorithme, nous l'avons utilisé pour la résolution de deux problèmes d'optimisation académiques combinatoires appartenant à la classe NP-Difficile: le problème du sac à dos et le problème du sac à dos multidimensionnel. Les résultats obtenus ont été comparés avec ceux obtenus par deux versions binaires (le BPSO et le PSO-P) de l'algorithme principal de la classe des algorithmes basés sur l'intelligence par essaim: l'algorithme d'optimisation par essaim de particules. Le point commun entre notre version, le BPSO et le PSO-P est que les trois algorithmes utilisent la même technique de transformation des solutions réelles vers des solutions binaires (i.e. la fonction Sigmoidale). D'autre part, nous avons comparé nos résultats avec les résultats obtenus avec l'algorithme de la recherche par harmonies et aussi avec les résultats obtenus par la version quantique de l'algorithme CS. Les résultats obtenus, montrent

l'efficacité de notre algorithme. Cette dernière revient au bon équilibre entre l'exploitation des solutions et l'exploration de l'espace de recherche, et aussi au petit nombre des paramètres de l'algorithme. En fait, à part la taille de la population, l'utilisateur de l'algorithme BCS n'a essentiellement qu'un seul paramètre à fixer.

L'algorithme proposé a fait l'objet de l'article suivant:

Amira Gherboudj, Abdesslem Layeb, Salim Chikhi. Solving 0-1 knapsack problems by a discrete binary version of cuckoo search algorithm. *International Journal of Bio-Inspired Computation*. Vol. 4, N°.4, pp. 229-236. Inderscience Publishers ISSN (Print): 1758-0366. ISSN (online): 1758-0374. DOI:10.1504/IJBIC.2012.048063. Juillet 2012.

Conclusion générale

Nous nous sommes intéressés dans ce mémoire aux méthodes de résolution de problèmes d'optimisation académiques difficiles. Notre objectif est de proposer des méthodes pour la résolution de ce type de problèmes. Afin de réaliser notre but, nous avons organisé nos recherches en deux phases. Dans la première phase, nous nous sommes concentrés sur l'établissement d'un état de l'art sur les problèmes d'optimisation et les différentes méthodes de résolution proposées dans la littérature. Nous avons essayé de récolter des notions de bases en optimisation, comprendre le rôle de la théorie de la complexité qui permet d'évaluer la complexité de problèmes et les classer en problèmes P ou NP. Nous avons ensuite essayé d'aborder quelques problèmes d'optimisation académiques difficiles à résoudre: leurs principes, leurs formulations, leurs variantes et leurs applications. Ensuite, nous avons établi une recherche sur les méthodes de résolution existantes : leurs origines, leurs principes, leurs étapes, leurs avantages, leurs lacunes et leurs applications. Cette première phase, nous a permis d'acquérir des informations et des connaissances dans notre domaine de recherche. Cela nous a orienté vers plusieurs axes de recherche et nous a permis d'avoir plusieurs idées que nous avons essayées de réaliser au cours de ces trois années de thèse.

Dans la première contribution, nous nous sommes inspirés de la jeune métaheuristique d'optimisation par essaim de particules, la principale métaheuristique basée sur l'intelligence par essaim. Malgré ses lacunes (trop de paramètres et convergence prématurée), l'optimisation par essaim de particules a prouvé sa simplicité de mise en œuvre et son efficacité qui ont suscité l'intérêt de plusieurs communautés de chercheurs. En effet, plusieurs variantes de l'algorithme d'optimisation par essaim de particules ont été proposées afin d'améliorer sa performance. De notre part, nous nous sommes inspirés de quelques travaux et applications de l'algorithme d'optimisation par essaim de particules présentés dans la littérature, et nous en avons proposé 16 versions avec différentes équations de mise à jour de vitesses et de positions des particules. Afin de tester la performance des algorithmes proposés, nous les avons appliqués pour la résolution du problème du sac à dos unidimensionnel. Notre objectif derrière le choix du problème du sac à dos est double: premièrement, montrer que nos algorithmes peuvent être utilisés pour la résolution de problèmes d'optimisation académiques difficiles. Deuxièmement, montrer que nos algorithmes ne sont pas réservés qu'aux problèmes d'optimisation continus, par contre ils peuvent être facilement adaptés pour la résolution de problèmes pouvant être modélisés par des représentations binaires.

Le problème du choix et du réglage des paramètres des algorithmes a construit depuis longtemps un sujet de recherche de plusieurs communautés de chercheurs. En fait, le choix du paramétrage influence directement sur la qualité de la solution proposée par un algorithme donné. Il varie d'un problème à un autre et d'une instance à une autre. Il est souvent basé sur l'expertise et l'expérience de l'utilisateur. Ce qui rend cette tâche très fastidieuse surtout pour des utilisateurs novices. L'idée d'affranchir l'utilisateur de la responsabilité du choix de quelques paramètres de l'algorithme a donné naissance à une nouvelle tendance très active qui consiste à confier cette tâche à l'algorithme lui-même. De notre part, ce problème nous a poussés à proposer une méthode de résolution de problème d'optimisation avec minimum de paramètres. Ainsi, dans notre deuxième contribution, nous nous sommes inspirés du principe de l'algorithme d'optimisation par essaim de particules et de l'opération de croisement de l'algorithme génétique et nous avons proposé un nouvel algorithme hybride avec un seul paramètre à choisir par l'utilisateur. Afin de tester la performance de notre algorithme, nous l'avons utilisé pour résoudre deux problèmes d'optimisation académiques NP-Difficiles: le problème du sac à dos multidimensionnel et le problème du voyageur de commerce. Notre objectif derrière le choix de ces deux problèmes est double aussi: premièrement, prouver l'efficacité de l'algorithme proposé dans la résolution de problèmes d'optimisation académique difficiles à résoudre. Deuxièmement, montrer que notre algorithme n'est pas réservé qu'à la résolution de problèmes binaires ou discrets. En fait, il peut être appliqué sur des problèmes de plusieurs types par le choix convenable du type de solutions initiales.

L'apparition de la métaheuristique appelée « la recherche coucou » a sollicité notre attention. C'est une nouvelle métaheuristique qui a enrichi le nombre de métaheuristicues basées sur l'intelligence par essaim. Elle est inspirée du comportement d'une catégorie spéciale d'oiseaux appelés « coucous ». En essayant de comprendre le principe de la métaheuristique de la recherche coucou, nous avons détecté que malgré ses avantages (particulièrement : peu de paramètre et un bon équilibre entre l'exploitation et l'exploration de l'espace de recherche), cette métaheuristique est conçue pour résoudre un seul type de problèmes d'optimisation qui est le type de problèmes continus. Nous avons donc pensé à pallier à cette lacune par la proposition d'une version binaire pour la méthode de la recherche coucou pour faire face à une grande gamme de problèmes pouvant être modélisés par des structures binaires. Ainsi, la version binaire de la méthode de la recherche coucou a fait le sujet de notre troisième contribution. Afin de tester la performance de notre algorithme dans la résolution de problèmes d'optimisation difficiles, nous l'avons appliqué sur deux

problèmes d'optimisation académiques NP-Difficiles: le problème du sac à dos unidimensionnel et le problème du sac à dos multidimensionnel.

Les trois années de cette thèse sont achevées. Cette période nous a permis d'acquérir beaucoup d'expériences. Nous avons pu réaliser quelques idées, mais la recherche n'a plus de limites tant qu'on a la volonté d'optimiser et de découvrir.

Ainsi, nos perspectives sont résumées dans ce qui suit:

- Appliquer les algorithmes proposés sur d'autres problèmes d'optimisation de différents types.
- Appliquer les algorithmes proposés sur des problèmes multi objectifs.
- Appliquer les algorithmes proposés sur des problèmes industriels.
- Améliorer les performances des algorithmes proposés par l'intégration de méthodes de création de solutions initiales de bonnes qualités.
- Améliorer les performances des algorithmes proposés par l'intégration de mécanismes de parallélisme dans le processus de recherche afin d'accélérer la convergence.
- Concevoir et développer une plateforme de résolution de problèmes d'optimisation de différentes complexités dans laquelle nous intégrons nos méthodes et d'autres méthodes existantes pour fournir un moyen pédagogique permettant de faciliter la mise en œuvre de différentes métaheuristiques et méthodes exactes pour la résolution de problèmes d'optimisation.

Bibliographie

- [Achnig, 2008] J. Achnig. Particle Swarm Optimization with Mutation for High Dimensional Problems. *Engineering Evolutionary Intelligent Systems*. pp 423-439. Springer, 2008.
- [Adams, 2012] S. Adams. Cuckoo chicks dupe foster parents from the moment they hatch. <http://www.telegraph.co.uk/earth/wildlife/4109282/Cuckoo-chicks-dupe-foster-parents-from-the-moment-they-hatch.html>. (Récupéré le 10.05.2012).
- [Ahrens et Finke, 1975] J. H. Ahrens et G. Finke. Merging and sorting applied to the zero-one knapsack problem. *Operations Research*. Vol. 23, N° 6, pp 1099-1109, 1975.
- [Ai et Kachitvichyanukul, 2008] T J. Ai, V. Kachitvichyanukul. A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery. *Computers and Operations Research*. Vol. 36, N°. 5, pp. 1693-1702, 2009.
- [Allahverdi et Al-Anzi, 2006] A. Allahverdi, F. Al-Anzi. A pso and a tabu search heuristics for the assembly scheduling problem of the two-stage distributed database application. *Computers and Operations Research*. Vol. 33, N°. 4, pp.1056-1080, 2006.
- [Alliot et Schiex, 1994] J-M. Alliot et T. Schiex. *Intelligence Artificielle & Informatique Théorique*. Cépadués Editions, 1994.
- [Alonso et al, 2005] C-L. Alonso, F. Caro, J-L. Montana. An Evolutionary Strategy for the Multidimensional 0-1 Knapsack Problem Based on Genetic Computation of Surrogate Multipliers. In: Proc. J. Mira and J.R. Alvarez (Eds.): *IWINAC 2005*, LNCS 3562, pp. 63-73. Springer, 2005.
- [Angelelli et al, 2010] E. Angelelli, R. Mansini, M.G. Speranza. Kernel search: A general heuristic for the multi-dimensional knapsack problem. *Computers & Operations Research*. Vol. 37, N° 37, pp 2017-2026. Elsevier, 2010.
- [Applegate et al, 2001] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. TSP cuts which do not conform to the template paradigm. In Michael Junger and Denis Naddef, editors, *Computational Combinatorial Optimization*. Vol. 2241 of *Lecture Notes in Computer Science*. Springer, 2001.
- [Applegate et Cook, 1991] D. Applegate, W. Cook. A Computational Study of The Job Shop Scheduling Problem. *ORSA Journal on Computing*. Vol. 3, N° 2, pp. 149-156. Spring, 1991.
- [Archetti et al, 2002] C. Archetti, A. Hertz and U. Derigs. A Tabu Search Algorithm for the Split Delivery Vehicle Routing Problem. *Mathematics of Information Technology and Complex Systems*, 2002.
- [Arora, 1989] J S. Arora, J. Introduction to Optimum Design, ISBN: 007002460X, 9780070024601, McGraw-Hill, 1989.
- [Asmussen, Søren, 2003] S. Asmussen, Søren. *Applied probability and queues*. Berlin: Springer. ISBN 978-0-387-00211-8, 2003.
- [Attenborough, 2012] D. Attenborough. *The Life of Birds, Parenthood*.

Bibliographie

<http://www.pbs.org/lifeofbirds/home/index.html>. (Récupéré le 10.05.2012).

[Banks et al, 2007a] A. Banks, J. Vincent, C. Anyakoha. A review of Particle Swarm Optimization. Part I: background and development. *Natural Computing*. Vol. 6, N°4, pp. 467-484, 2007.

[Banks et al, 2007b] A. Banks, J. Vincent, C. Anyakoha. A review of Particle Swarm Optimization. Part II: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. *Natural Computing*. Vol. 7, N°1, pp. 109-124, 2007.

[Barnhart et al, 2005] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: column generation for solving huge integer programs. *Operations Research*. Vol. 46, N° 3, pp 316-329, 1998.

[Barrera et Coello, 2009] J. Barrera, C. A.C. Coello. Limiting the velocity in particle swarm optimization using a geometric series. *Genetic And Evolutionary Computation Conference, Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 1739-1740, 2009.

[Basseur, 2005] M. Basseur. Conception d'Algorithmes Coopératifs Pour L'optimisation Multi-Objectif: Application Aux Problèmes D'ordonnancement De Type Flow-Shop. Thèse de doctorat. Université de Lille, France. 2005

[Băutu. A et Băutu. E, 2007] A. Băutu, E. Băutu. Searching Ground States of Ising Spin Glasses with Genetic Algorithms and Binary Particle Swarm Optimization. *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*, pp.85-94, Springer, 2007.

[Benameur et al, 2010] L. Benameur, J. Alami, A. El Imrani. A hybrid discrete particle swarm algorithm for solving the fixed-spectrum frequency assignment problem. *International Journal of Computational Science and Engineering*. Vol. 5, N° .1, pp. 68 - 73, 2010.

[Benatchba, 2005] K. Benatchba. Modèle d'exécution pour l'aide à la résolution du problème MAX-SAT. Thèse de doctorat. L'institut national d'Informatique, Alger, Algérie. 2005.

[Ben-Avraham et Havlin, 2002] D. Ben-Avraham et S. Havlin. *Diffusion and Reactions in Fractals and Disordered Systems* (Cambridge Univ. Press, Cambridge), 2000.

[Bendiab et al, 2003] E. Bendiab, S. Meshoul and M. Batouche. An AIS for Multi-Modality Image Alignment. *Second International Conference on Artificial Immune systems, ICARIS 2003*, Edinburgh, UK, September 1-3, pp. 13-21. *Proceedings, Lecture Notes in Computer Science*, LNCS 2787, ISSN: 0302-9743. Springer-Verlag Berlin Heidelberg 2003.

[Beyer, 2001] H.G. Beyer. *The theory of evolution strategies*. *Natural Computing Series*, ISBN: 9783540672975. Springer, 2001.

[Biggs et al, 1976] N.L.Biggs, E.K.Lloyd et R.J.Wilson. *Graph Theory 1736-1936*. Clarendon Press. Oxford. 1976.

[Billionnet et al, 1997] A. Billionnet, A. Faye, et E. Soutif. An exact algorithm for the 0-1 quadratic knapsack problem. In *ISMP'97*, Lausanne, Switzerland, 1997.

Bibliographie

- [Billionnet et Soutif, 2004] A. Billionnet et E. Soutif. Using a mixed integer programming tool for solving the 0-1 quadratic knapsack problem. *INFORMS Journal on Computing*. Vol. 16, N° 2, pp 188-197, 2004.
- [Blum et Rilo, 2003] C. Blum et A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.* Vol. 35, N° 3, pp 268-308, 2003.
- [Bouglouan, 2012] N. Bouglouan. <http://www.oiseaux-birds.com>. (récupéré le 10.05.2012).
- [Bourazza, 2006] S. Bourazza. Variantes d'algorithmes génétiques appliquées aux problèmes d'ordonnancement. Thèse de doctorat. Université du Havre, France. 2006.
- [Bretthauer et Shetty, 2002] K. M. Bretthauer et B. Shetty. The nonlinear knapsack problem - algorithms and applications. *European Journal of Operational Research*. Vol. 138, N° 3, pp 459-472, May 2002.
- [Brown et al, 2007] C. Brown, L. S. Liebovitch, R. Glendon. Lévy flights in Dobe Ju/'hoansi foraging patterns. *Human Ecol.* Vol. 35, pp 129-138, 2007.
- [Cai et al, 2009] X. Cai, P. Koduru, S. Das, S. M. Welch. Simultaneous structure discovery and parameter estimation in gene networks using a multi-objective GP-PSO hybrid approach. *International Journal of Bioinformatics Research and Applications*. Vol. 5, N° 3, pp. 254 - 268, 2009.
- [Cai et Tan, 2009] X. Cai, Y. Tan. A study on the effect of v_{max} in particle swarm optimization with high dimension. *International Journal of Bio-Inspired Computation (IJBIC)*. Vol. 1, N° 3, pp. 210 - 216, 2009.
- [Camerini et al, 1974] P. M. Camerini, L. Fratta et F. Maffioli. On improving relaxation methods by modified gradient techniques. *Mathematical Programming Study*. Vol. 3, pp 26-34, 1974.
- [Campbell, 1996] N.A. Campbell. Fixed action patterns. In: *Biology*, fourth ed., Benjamin Cummings, ISBN 0-8053-1957-3. New York, 1996.
- [Caprara et al, 1980] A. Caprara, D. Pisinger et P. Toth. Exact solution of the quadratic knapsack problem. *INFORMS Journal on Computing*. Vol. 11, N° 2, pp 125-137, 1999.
- [Chan et Tiwari, 2007] F. T. S. Chan et M. K. Tiwari. *Swarm Intelligence: Focus on Ant and Particle Swarm Optimization*. ISBN 978-3-902613-09-7. pp. 532, Itech Education and Publishing, Vienna, Austria. December, 2007.
- [Chang et al, 2004] B.C. H. Chang, A. Ratnaweera, S. K. Halgamuge, H. C. Watson. Particle swarm optimization for protein motif discovery. *Genet Program Evolvable Mach.* Vol. 5, N° 2, pp. 203-214, 2004.
- [Chang et al, 2009] J-X. Chang, F. Wan, Q. Huang, W-l. Yuan, Y-M Wang. Application of particle swarm optimisation based on immune evolutionary algorithm for optimal operation of cascade reservoirs. *International Journal of Modelling, Identification and Control (IJMIC)*. Vol. 8, N° 3, pp. 233 - 239, 2009.

Bibliographie

- [Charon et al, 1996] I. Charon, A. Germa et O. Hudry. Méthodes d'optimisation combinatoire. Masson, Paris. 1996.
- [Chatterjee et Siarry , 2006] A. Chatterjee, P. Siarry. Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization. Computers & Operations Research. Vol. 33, N°. 3, pp. 859-871, 2006.
- [Chen et al, 2006] A. Chen, G. Yang, Z. Wu. Hybrid discrete particle swarm optimization algorithm for capacitated vehicle routing problem. Journal of Zhejiang University Science. Vol.7, N°.4, pp. 607-614, 2006.
- [Chen et al, 2009] R-M. Chen, D-F. Shiao ,S-T. Lo. Combined Discrete Particle Swarm Optimization and Simulated Annealing for Grid Computing Scheduling Problem. Emerging Intelligent Computing Technology and Applications. With Aspects of Artificial Intelligence, pp. 242-251. Springer, 2009.
- [Chen et Wang, 2007] Y.M. Chen, W-S. Wang. Fast solution technique for unit commitment by particle swarm optimisation and genetic algorithm. International Journal of Energy Technology and Policy. Vol. 5, N°.4, pp. 440 -456, 2007.
- [Chen. M et al, 2009] M-R. Chen, X. Li, X. Zhang, Y-Z. Lu. A novel particle swarm optimizer hybridized with extremal optimization. Applied Soft Computing. Vol. 10, N°. 2, pp. 367-373, 2010.
- [Chu et Beasley, 1998] P.C. Chu et J.E. Beasley. A Genetic Algorithm for the Multidimensional Knapsack Problem. Journal of Heuristics. Vol. 4, N° 1, pp. 63-86, 1998.
- [Civicioglu et Besdok, 2011] P. Civicioglu et E. Besdok. A conceptual comparison of the Cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms. Artif Intell Rev. DOI 10.1007/s10462-011-9276-0. Springer, 2011.
- [Civicioglu et Besdok, 2011] P. Civicioglu et E. Besdok. A conception comparison of the cuckoo search, particle swarm optimization, differential evolution and artificial bee colony algorithms. Artificial Intelligence Review, DOI 10.1007/s10462-011-92760, 2011.
- [Clerc et Kennedy, 2002] M. Clerc, J. Kennedy. The particle swarm: explosion, stability, and convergence in multi-dimensional complex space. IEEE Transactions on Evolutionary Computation. Vol. 6, N° 6, pp. 58-73, 2002.
- [Clerc, 1999] M. Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. Proceedings, 1999 ICEC, Washington, DC, pp.1951-1957, 1999.
- [Clerc, 2003] M. Clerc. TRIBES- Un exemple d'optimisation par essaim particulaire sans paramètre de contrôle. Conférence OEP'03, Paris, France, 2 Octobre 2003.
- [Coelho et al, 2008] L. D. S. Coelho, N. Nedjah, L. D. M. Mourelle. Gaussian Quantum-Behaved Particle Swarm Optimization Applied to Fuzzy PID Controller Design. Quantum Inspired Intelligent Systems, pp 1-15. Springer, 2008.
- [Coelho et Bernert, 2009] L. D. S. Coelho, D. L. D. A. Bernert. PID control design for chaotic synchronization using a tribes optimization approach. Chaos, Solitons & Fractals.

Vol. 42, N° 1, pp. 634-640, 2009.

[Colorni et al, 1992] A. Colorni, M. Dorigo, V. Maniezzo. Distributed Optimization by Ant Colonies. Proceedings of the 1st European Conference on Artificial Life. pp 134-142, Elsevier Publishing.

[Cook, 1971] S. Cook. The Complexity of Theorem-Proving Procedure. Proc. 3rd ACP Symp. On Theory of Complexity, Association of Computing Machinery. New York. pp. 151-158, 1971.

[Cooren et al, 2008a] Y. Cooren, A. Nakib, P. Siarry. Image Thresholding using TRIBES, a parameter-free Particle Swarm Optimization Algorithm. Proceedings of the International Conference on Learning and Intelligent Optimization, pp 81-94. Springer, 2008.

[Cooren et al, 2009] Y. Cooren, M. Clerc, P. Siarry. Performance evaluation of TRIBES, an adaptive particle swarm optimization algorithm. Swarm Intelligence. Vol. 3, N° 2, pp 149-178, 2009.

[Cooren et al, 2011] Y. Cooren, M. Clerc, P. Siarry. MO-TRIBES, an adaptive multiobjective particle swarm optimization algorithm. Computational Optimization and Applications. Vol. 49, N° 2, pp 379-400. Springer, 2011.

[Cooren, 2008] Y. Cooren. Perfectionnement d'un algorithme adaptatif d'Optimisation par Essaim Particulaire, Applications en génie médical et en électronique. Thèse de Doctorat, Université Paris 12 Val de Marne, France, 2008.

[Cordeau et al, 2000] J-F. Cordeau, G. Desaulniers, J. Desrosiers, M. Solomon et F. Soumis. The VRP with Time Windows. Technical Report, Département de Mathématiques et de Génie Industriel, Ecole polytechnique de Montréal, Canada. June, 2000.

[Cormen et al, 1994] T. Cormen, C. Leiserson et R. Rivest. Introduction à l'algorithmique. Edition Dunod. 1994.

[Cotta et Troya, 1998] C. Cotta, J. M. Troya. A Hybrid Genetic Algorithm for the 0-1 Multiple Knapsack Problem. Artificial Neural Nets and Genetic Algorithms. Vol. 3, pp 250-254. New York, 1998.

[Czogalla et Fink, 2008] J. Czogalla, A. Fink. On the Effectiveness of Particle Swarm Optimization and Variable Neighborhood Descent for the Continuous Flow-Shop Scheduling Problem. Metaheuristics for Scheduling in Industrial and Manufacturing Applications. pp. 61-89. Springer, 2008.

[Dantzig et al, 1954] G. B. Dantzig, R. Fulkerson, S. Johnson. Solution of a large-scale traveling salesman problem. Operations Research. Vol. 2, N° 4, pp 393- 410, 1954.

[Dantzig et Ramser, 1959] G. B. Dantzig, J. H Ramser. The Truck dispatching problem. Lehrstuhl für Wirtschaftsinformatik und Operations Research. Universität zu Köln, Mgmt Sci. Vol. 6, N° 1, pp 80-91, 1959.

[Dantzig, 1957] G. B. Dantzig. Discrete-Variable Extremum Problems. Operations Research. Vol. 5, N° 2, pp 266-277. Avril, 1957.

Bibliographie

- [Darwin, 1859] C. Darwin. The origin of species by means of natural selection. Mentor Reprint, New York, 1859.
- [Dasgupta et Gonzalez, 2002] D. Dasgupta, F. Gonzalez. An Immunity-Based Technique to Characterize Intrusions in Computer Networks. IEEE Trans. Evol. Comput. Vol. 6, N° 3, pp 281-291, 2002.
- [De Jong, 1975] K. A. De Jong. An analysis of the behavior of a class of genetic adaptive systems. Ph. D. Thesis, University of Michigan, 1975.
- [Deb, 1995] K. Deb. Optimisation for Engineering Design, Prentice-Hall. New Delhi, 1995.
- [Deep et Bansal, 2009] K. Deep, J. C. Bansal. Hybridization of particle swarm optimization with quadratic approximation. OPSEARCH. Vol. 46, N° 1, pp. 3-24, 2009.
- [Devarenne, 2007] I. Devarenne. Etude en recherche locale adaptative pour l'optimisation combinatoire. Thèse de doctorat. Université de Franche-Comté, France. 2007.
- [Dhivyaet et Sundarambal, 2011] M. Dhivyaet M. Sundarambal. Cuckoo search for data gathering in wireless sensor networks. International Journal of Mobile Communications. Vol. 9, N° 6, pp. 642-656, 2011.
- [Dorigo et al, 1996] M. Dorigo, V. Maniezzo, A. Coloni. Ant System: optimization by a colony of cooperating agents. IEEE Transactions on Man. Cyber, Part B. Vol. 26, N°1, pp. 29-41, 1996.
- [Dorigo et Gambardella, 1997] M. Dorigo et L.M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation. Vol. 1, N°. 1, pp.53-66. 1997.
- [Duran et al, 2008] O. Duran, N. Rodriguez, L. A. Consalter. Hybridization of PSO and a discrete Position Update Scheme Techniques for Manufacturing Cell Design. MICAI 2008: Advances in Artificial Intelligence, pp. 503-512. Springer, 2008.
- [Dyer et al, 1984] M. E. Dyer, N. Kayal et J. Walker. A branch and bound algorithm for solving the multiple choice knapsack problem. Journal of Computational and Applied Mathematics. Vol. 11, N° 2, pp 231-249, 1984.
- [Dyer et al, 1995] M. E. Dyer, W. O. Riha et J. Walker. A hybrid dynamic programming/branch and bound algorithm for the multiple-choice knapsack problem. European Journal of Operational Research. Vol. 58, N° 1, pp. 43-54, 1995.
- [Eberhart et al, 1996] R.C. Eberhart, P. Simpson, R. Dobbins. Computational PC Tools. Chapter 6, AP Professional. pp. 212-226, 1996.
- [Eberhart et Shi, 2000] R.C. Eberhart, Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. Proceedings of the 6th IEEE Congress on Evolutionary Computation, IEEE Press. pp. 84-88, 2000.
- [Fan et al, 2001] H.Y. Fan, Y. Shi. Study on Vmax of particle swarm optimization. Proceedings of the 2001 Workshop on Particle Swarm Optimization, Indiana University-Purdue University Indianapolis Press. 2001.

Bibliographie

- [Fan et Chang, 2007] S-K S. Fan, J-M Chang. A Modified Particle Swarm Optimizer Using an Adaptive Dynamic Weight Scheme. *Digital Human Modeling*. pp. 56-65. Springer, 2007.
- [Fang et al, 2010] W. Fang, J. Sun, Y. Ding, X. Wu, W. Xu. A Review of Quantum-behaved Particle Swarm Optimization. *IETE Technical Review*. Vol. 27, N° 4, pp. 336-348, 2010.
- [Fayard et Plateau, 1982] D. Fayard et G. Plateau. An algorithm for the solution of the 0-1 knapsack problem. *Computing*. Vol. 28, pp. 269-287, 1982.
- [Feigenbaum et Feldman, 1963a] E. A. Feigenbaum, j. Feldman, (Edirors). *Computers and thought*. McGraw-Hill Inc. pp.6. New York, 1963.
- [Feigenbaum et Feldman, 1963b] E. A. Feigenbaum and J. Feldman. (Edirors). *Computers and thought*. McGraw-Hill Inc. pp.192. New York, 1963.
- [Feng et al, 2005] D. Feng, S. Wenkang, C. Liangzhou, D. Yong, Z. Zhenfu. Infrared image segmentation with 2-D maximum entropy method based on particle swarm optimization (PSO). *Pattern Recognition Letters*. Vol. 26, N° 5, pp. 597-603, 2005.
- [Ferro et al, 2005] L. Ferro, S. Khin, N. Salman. Résolution pratique de problèmes NP-complets. 26 Juin, 2005.
- [Fischetti et al, 1999] M. Fischetti, A. Lodi and P. Toth. A Branch-and-Cut Algorithm for the Multiple Depot Vehicle Scheduling Problem. *Dipartimento di Elettronica e Informatica, Universita di Padova. Italy*, 1999.
- [Fogel et al, 1966] L.J. Fogel, A.J. Owens, M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, Inc., New York, 1966.
- [Fogel, 2001] D.B. Fogel. An introduction to evolutionary computation, tutorial. *Congress on Evolutionary Computation (CEC'2001)*, Seoul, Korea, 2001.
- [Fonseca et Fleming, 1993] Fonseca C. M et Fleming P. J. Genetic Algorithm for Multiobjective Optimization: Formulation, Discussion and Generalization. In *Proceedings of the Fifth International Conference on Genetic Algorithms*. pp. 416-423. San Mateo, California. 1993.
- [Foo et al, 2005] YC. Foo, SF. Chien, L. Aly, CF .Teo. New strategy for optimizing wavelength converter placement. *Opt Express*. Vol. 13, N°2, pp. 545-551, 2005.
- [Fu et al, 2003] Z. Fu, R. W. Eglese and L. Li. A tabu search heuristic for the open vehicle routing problem. *LUMS working paper*, 2003.
- [Gallo et al, 1980] G. Gallo, P. L. Hammer, B. Simeone. Quadratic knapsack problems. *Mathematical Programming*. Vol. 12, pp. 132-149, 1980.
- [Gandomi et al, 2011] H. Gandomi, X. Yang et A. Alavi. Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. *Int. Engineering with Computers*, DOI: 10.1007/s00366-011-0241-y, 2011.
- [Garey et Johnson, 1979] R. M. Garey et D. S. Johnson. *Computers and Intractability. A guide to the Theory of NP-Completeness*. W.H.Freeman and CO, San Francisco. 1979.
- [Geem et al 2001] Z. W. Geem, J. H. Kim, G. V. Loganathan. A new heuristic optimization

Bibliographie

- algorithm: Harmony search. Simulation. Vol. 76, N° 2, pp. 60-68, 2001.
- [Geem et al, 2001] Z. W. Geem, J. H. Kim and G. V. Loganathan. A New Heuristic Optimization Algorithm: Harmony Search. Simulation. Vol. 76, N° 2, pp. 60-68, 2001.
- [Geem et Choi, 2007] Z. W. Geem and J.Y Choi. Music composition Using Harmony Search Algorithm. M. Giacobini et al. (Eds.): EvoWorkshops 2007, LNCS 4448, pp. 593-600. Springer, 2007.
- [Gendreau et al, 2001] M. Gendreau, G. Laporte, F. Semet. A dynamic model and parallel tabu search heuristic for real-time ambulance relocation. Parallel Computing. Vol. 27, N° 12, pp 1641-1653, 2001.
- [Gendreau et Potvin, 1998] M. Gendreau, J.Y. Potvin. Dynamic vehicle routing and dispatching. Working paper, Université de Montréal, 1998.
- [Gens et Levner, 1979] G. V. Gens, E. V. Levner. Computational complexity of approximation algorithms for combinatorial problems. In Mathematical Foundations of Computer Science, pp 292-300, 1979.
- [Ghali, 2005] K. Ghali. Méthodologie de conception système a base de plateformes reconfigurables et programmables. Thèse de doctorat. Université Paris XI, France. 2005.
- [Gherboudj et al, 2012] A. Gherboudj, A. Layeb, S. Chikhi. Solving 0-1 knapsack problems by a discrete binary version of cuckoo search algorithm. International Journal of Bio-Inspired Computation. Vol. 4, N° 4, pp 229-236. Inderscience Publishers ISSN (Print): 1758-0366, ISSN (online): 1758-0374. DOI: 10.1504/IJBIC.2012.048063. 2012.
- [Gherboudj et Chikhi, 2011] A. Gherboudj, S. Chikhi. BPSO Algorithms for Knapsack Problem. A. Özcan, J. Zizka, and D. Nagamalai (Eds.): WiMo/CoNeCo 2011, CCIS 162, pp. 217-227. Springer, 2011.
- [Glover et Laguna, 1997] F. Glover and M. Laguna. Tabu Search. Kluwer Academic Publishers, Boston. 1997.
- [Glover, 1986] F. Glover. Future paths for integer programming and links to artificial intelligence. Computers and Operations Research. Vol. 13, N° 5, pp 533-549, 1986.
- [Goldberg, 1989] D.E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning, ISBN: 0201157675, Addison-Wesley, 1989.
- [Groth, 2002] M. J. Groth. Stochastic considerations in vehicle routing Problems. Transport Optimization, June 17, 2002.
- [Gupta et Garg, 2009] S. Gupta et M.L. Garg. An Improved Genetic Algorithm Based on Adaptive Repair Operator for Solving the Knapsack Problem. Journal of Computer Science. Vol. 5, N° 8, pp. 544-547, 2009.
- [Habibi et al, 2006] J. Habibi, S.A. Zonouz, M. Saneei. A hybrid PS-based optimization algorithm for solving traveling salesman problem. In: IEEE symposium on frontiers in networking with applications (FINA 2006). Vienna, Austria, pp. 18-20. April, 2006.
- [Hammer et Rader, 1997] P. L. Hammer et D. J. Jr. Rader. Efficient methods for solving

quadratic 0-1 knapsack problems. *INFOR*. Vol. 35, N° 3, pp. 170-182, 1997.

[He et al, 2004] S. He, QH. Wu, JY. Wen, JR. Saunders, RC. Paton. A particle swarm optimizer with passive congregation. *Biosystems*. Vol. 78, N° 1, pp. 135- 147. 2004.

[Helmberg et al, 2000] C. Helmberg, F. Rendl, et R.Weismantel. A semidefinite programming approach to the quadratic knapsack problem. *Journal of Combinatorial Optimization*. Vol. 4, N° 2, pp. 197-215, 2000.

[Heppner et Grenander, 1990] F. Heppner, U. Grenander. A stochastic nonlinear model for coordinated bird flocks. In: S Krasner (eds) *The ubiquity of chaos*. AAAS Publications, Washington, DC.

[Hiley et Julstrom, 2006] A. Hiley, B. A. Julstrom. The Quadratic Multiple Knapsack Problem and Three Heuristic Approaches to it. In: *Proceedings of the GECCO*. pp. 547- 552. ACM Press. New York, 2006.

[Holden et Freitas, 2005] N. Holden, A.A. Freitas. A hybrid particle swarm/ant colony algorithm for the classification of hierarchical biological data. In: *Proc. of 2005 IEEE Swarm Intelligence Symposium*, Pasadena, California, pp. 100-107. 2005.

[Holden et Freitas, 2006] N. Holden, A.A. Freitas. Hierarchical classification of G-protein-coupled receptors with a PSO/ACO algorithm. In: *Proc. of 2006 IEEE Swarm Intelligence Symposium*, Indianapolis. pp. 77-84. 2006.

[Holland, 1973] J.H. Holland. *Genetic Algorithms and the optimal allocation of trials*, SIAM Journal of Computing. Vol. 2, N° 2, pp. 88-105, 1973.

[Horowitz et Sahni, 1974] E. Horowitz, S. Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM*. Vol. 21, N° 2, pp. 277-292, 1974.

[Hsu et al, 2008] C-C Hsu, C-H Gao. Digital redesign of uncertain interval systems via a hybrid particle swarm optimizer. *International Journal of Innovative Computing and Applications*. Vol. 1, N° 4, pp. 232 - 243. 2008.

[Hsu et al, 2010] C-C Hsu, W-Y Shieh, C-H Gao. Digital redesign of uncertain interval systems based on extremal gain/phase margins via a hybrid particle swarm optimizer. *Applied Soft Computing*. Vol. 10, N° 2, pp. 602-612. 2010.

[Hu et al, 2008] X. Hu, J. Zhang, Y. Li. Orthogonal methods based ant colony search for solving continuous optimization problems. *Journal of Computer Science and Technology*. Vol. 23, N°. 1, pp. 2-18. 2008.

[Hunt et Cooke, 1996] J. E. Hunt and D. E. Cooke. Learning Using an Artificial Immune System. *Journal of Network and Computer Applications*. Vol.19, N° 2 pp. 189-212, 1996.

[Jacobs-Blecha et al, 1998] C. Jacobs-Blecha, M. Goetschalckx. *The vehicle Routing Problem with Backhauls: Properties and Solution Algorithms*. School of Industrial and systems engineering. 1998.

[Jiang et al, 2004] J.Q. Jiang, Y.C. Liang, X.H. Shi, H.P. Lee. A Hybrid Algorithm Based on PSO and SA and Its Application for Two-Dimensional Non-guillotine Cutting Stock

Bibliographie

- Problem. Computational Science - ICCS 2004. pp. 666-669. Springer, 2004.
- [Johnson et al, 1993] E. L. Johnson, A. Mehrotra, G. L. Nemhauser. Min-cut clustering. Mathematical Programming. Vol. 62, pp. 133-152, 1993.
- [Johnson, 1954] S. M. Johnson. Optimal two-and three-stage production schedules with setup time included. Naval Research Logistic Quarterly. Vol. 1, pp. 61-81, 1954.
- [Jorge, 2010] J. Jorge. Nouvelles propositions pour la résolution exacte du sac à dos multi-objectif unidimensionnel en variables binaires. Thèse de doctorat. Université de Nante, France. 2010.
- [Julstrom, 2005] B-A. Julstrom. Greedy, genetic, and greedy genetic algorithms for the quadratic knapsack problem. In Proc. GECCO '05 Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, Publisher, ACM. ISBN: 1595930108, pp.607-614, 2005.
- [Kammarti, 2006] R. Kammarti. Approches Evolutionnistes Pour La Resolution Du 1-PDPTW Statique Et Dynamique. Thèse de doctorat. Université de Lille, France. 2006.
- [Karaboga et Basturk, 2007] D. Karaboga, B. Basturk. Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems. Lecture Notes Computer Science. Vol. 4529, pp. 789-798, 2007.
- [Karaboga, 2005] D. Karaboga. An idea based on honeybee swarm for numerical optimization. Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.
- [Kellerer et al, 2004] H. Kellerer, U. Pferschy, D. Pisinger. Knapsack Problems. Springer, 2004.
- [Kennedy et Eberhart, 1995] J. Kennedy, R. C. Eberhart. Particle swarm optimization. Proceedings of the IEEE International Conference Neural Networks. Vol. 4, pp. 1942-1948, 1995.
- [Kennedy et Eberhart, 1997] J. Kennedy, R.C. Eberhart. A discrete binary version of the particle swarm algorithm. Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics, NJ: Piscataway. pp. 4104-4109, 1997.
- [Kilby et al, 1999] P. Kilby, P. Prosser, and P. Shaw. Guided Local Search for the Vehicle Routing Problem with Time Windows. Metaheuristics Advanced and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publisher, Bosten, 1999.
- [Kirkpatrick et al, 1983] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi. Optimization by simulated annealing. Journal of Science. Vol. 220, N° 4598, pp. 671-680, 1983.
- [Ko et Lin, 2004] PC. Ko, PC. Lin. A hybrid swarm intelligence based mechanism for earning forecast. Asian Journal of Information Technology. Vol. 3, N° 3, pp.180-187, 2004.
- [Kong et Tian, 2006] M. Kong, P. Tian. Apply the Particle Swarm Optimization to the Multidimensional Knapsack Problem. In proc. L. Rutkowski et al. (Eds.): ICAISC 2006, LNAI 4029, pp. 1140 -1149. Springer, 2006.

Bibliographie

- [kouki et al, 2011] S. kouki, M. Jemni, T. Ladhari. Solving the Permutation Flow Shop Problem with Makespan Criterion using Grids. *International Journal of Grid and Distributed Computing*. Vol. 4, N° 2, pp. 53-64. June, 2011.
- [Koza, 1996] J. Koza. *Genetic Programming. On the Programming of Computers by Means of Natural Selection*. A Bradford Book. MIT Press, 1996.
- [Krumke et al, 2001] S.O. Krumke, W.E de Paepe, D. Poensgen and L. Stougie. *News from the Online Travelling Repairman*. *Lecture Notes in Computer Science*, 2001.
- [Kumar et al, 2009] R. Kumar , D. Sharma , A. Kumar. A new hybrid multi-agent-based particle swarm optimisation technique. *International Journal of Bio-Inspired Computation*. Vol. 1, N° 4, pp. 259 - 269, 2009.
- [Kuo et al , 2007] I-H. Kuo, S-J. Horng, T-W. Kao, T-L. Lin, P. Fan. An Efficient Flow-Shop Scheduling Algorithm Based on a Hybrid Particle Swarm Optimization Model. *New Trends in Applied Artificial Intelligence*. pp.303-312, Springer.
- [Lacomme et al, 2003] P. Lacomme, C. Prins, M. Sevaux. *Algorithmes de Graphes*. Eyrolles, Paris, 2003.
- [Lardeux, 2005] F. Lardeux. *Approches hybrides pour les problèmes de satisfiabilité (SAT et MAX-SAT)*. Thèse de doctorat. Université d'Angers, France. 2005.
- [Laughunn, 1970] D. L. Laughunn. Quadratic binary programming with applications to capital budgeting problems. *Operations Research*. Vol. 18, N° 3, pp. 454-461, 1970.
- [Layeb et al, 2007] A. Layeb, A. Deneche. Multiple Sequence Alignment by Immune Artificial System. In: *IEEE/ACS International Conference on Computer Systems and Applications (AICCSA 2007)*, Jordan, pp. 336-342. ISBN: 1-4244-1031-2, 2007.
- [Layeb, 2010] A. Layeb. *Utilisation des Approches d'Optimisation Combinatoire pour la Vérification des Applications Temps Réel*. Thèse de doctorat. Université de Constantine 2, Algérie. 2010.
- [Layeb, 2011] A. Layeb. A novel quantum inspired cuckoo search for knapsack problems. *Int. J. Bio-Inspired Computation*. Vol. 3, N° 5, pp 297-305, 2011.
- [Lee et al, 2007] S. Lee, J. Lee, D. Shim, M. Jeon. Binary Particle Swarm Optimization for Black-Scholes Option Pricing. *Knowledge-Based Intelligent Information and Engineering Systems*. pp. 85-92. Springer, 2007.
- [Li et al, 2006] H. Li, Y-C Jiao, L. Zhang, Z-W. Gu. Genetic Algorithm Based on the Orthogonal Design for Multidimensional Knapsack Problems. In: *Proc. L. Jiao et al. (Eds.): ICNC 2006, Part I, LNCS*. Vol. 4221, pp 696-705. Springer, 2006.
- [Li et al, 2008] S. Li, X. Wu, M. Tan. Gene selection using hybrid particle swarm optimization and genetic algorithm. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*. Vol. 12, N° 11, pp. 1039-1048, 2008.
- [Li et Zhang, 2007] L. Li, Y. Zhang. An Improved Genetic Algorithm for the Traveling Salesman Problem. *Communications in Computer and Information Science*. Vol. 2, Part 5,

pp. 208-216, 2007.

[Li et al, 2010] Z. Li, D. Zheng, H. Hou. A Hybrid Particle Swarm Optimization Algorithm Based on Nonlinear Simplex Method and Tabu Search. *Advances in Neural Networks. ISNN 2010. Lecture Notes in Computer Science*. Vol. 6063, pp. 126-135. Springer, 2010.

[Lian et al, 2006] Z. Lian, X. Gu, B. Jiao. A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan. *Chaos, Solitons and Fractals*. Vol. 35, N° 5, pp. 851-861, 2008.

[Liao et al, 2007] C-J. Liao, C-T. Tseng, P. Luarn. A discrete version of particle swarm optimization for flowshop scheduling problems. *Computers and Operations Research*. Vol. 34, pp. 3099 -3111. Elsevier, 2007.

[Liu et Abraham, 2006] H. Liu, A .Abraham. An Hybrid Fuzzy Variable Neighborhood Particle Swarm Optimization Algorithm for Solving Quadratic Assignment Problems. *Journal of Universal Computer Science*. Vol. 13, N° 9, pp. 1309-1331. 2007.

[Liu et Abraham, 2009] H. Liu, A. Abraham. Turbulent Particle Swarm Optimization Using Fuzzy Parameter Tuning. *Foundations of Computational Intelligence*. pp. 291-312. Springer, 2009.

[Liu et al, 2006] J. Liu, J. Sun, W. Xu. Improving Quantum-Behaved Particle Swarm Optimization by Simulated Annealing. *Computational Intelligence and Bioinformatics*. pp. 130-136. Springer, 2006.

[Liu et al, 2005] B. Liu, L. Wang, Y-h. Jin. An effective hybrid particle swarm optimization for no-wait flow shop scheduling. *The International Journal of Advanced Manufacturing Technology*. Vol. 31, N° 9-10, pp. 1001-1011. Springer.

[Liu.B et al, 2006] B. Liu, L. Wang, Y-h Jin, D-X Huang. An Effective PSO-Based Memetic Algorithm for TSP. *Intelligent Computing in Signal Processing and Pattern Recognition*. pp. 1151-1156. Springer, 2006.

[Long et al, 2009] H. Long, J. Sun, X. Wang, C-H. Lai, W. Xu. Using selection to improve quantum-behaved particle swarm optimization. *Int. J. of Innovative Computing and Applications*. Vol. 2, N° 2, pp. 100 - 114, 2009.

[Lope et Coelho, 2005] H. S. Lope, L. S. Coelho. Particle Swarm Optimization with Fast Local Search for the Blind Traveling Salesman Problem. *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*. pp. 245-250, 2005.

[Lorie et Savage, 1955] J. H. Lorie, L. J. Savage. Three problems in capital rationing. *The Journal of Business*. Vol. 28, pp. 229-239, 1955.

[Machado et Lopes, 2005] T. R. Machado, H. S. Lopes. A hybrid particle swarm optimization model for the traveling salesman problem. *Adaptive and Natural Computing Algorithms*. pp. 255-258. Springer, 2005.

[Marinakis et al, 2008] Y. Marinakis, M. Marinaki, N. Matsatsinis. A Hybrid Clustering Algorithm Based on Multi-swarm Constriction PSO and GRASP. *Data Warehousing and*

Bibliographie

Knowledge Discovery. pp. 186-195. Springer, 2008.

[Martello et Toth, 1977] S. Martello, P. Toth. An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *European Journal of Operational Research*. Vol. 1, pp. 169-175, 1977.

[Martello et Toth, 1990] S. Martello, P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. J. Wiley, 1990.

[Mathieu, 2008] B. Mathieu. Méthode de comparaison statistique des performances d'algorithmes évolutionnaires. Thèse de Doctorat. Université du Québec, Canada. 2008.

[Mathworks, 2012] http://www.mathworks.com/matlabcentral/fileexchange/29809-cuckoo-search-cs-algorithm/content/cuckoo_search.m. (Récupéré le 28/05/2012)

[Mechti, 1995] R. Mechti. Tournée de véhicules à la demande: problèmes et méthodes. Technical report, Laboratoire PRiSM, Université de Versailles-St. Quentin, France, 1995.

[Mercadier, 2008] N. Mercadier. Diffusion à Résonance Atomique, Application au mélange à 4 ondes et au vol de Lévy du photon. *Ecole Centrale Paris*. pp. 35, 2008.

[Merkle et Hellman, 1978] R. Merkle et M. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*. Vol. 24, N° 5, pp. 525-530, 1978.

[Meshoul et al, 2005] S. Meshoul, A. Deneche and M. Batouche. Combining an artificial immune system with a clustering method for effective pattern recognition. In *proceedings of the international conference on machine intelligence*. pp. 782-787, Tozeur-Tunisia, November 5-7, 2005.

[Metropolis et al, 1953] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller. Equation of state calculations by fast computing machines. *Journal of Chem. Physics*. Vol. 21, N° 66, pp. 1087-1092, 1953.

[Mladenovic et Hansen, 1997] N. Mladenovic, P. Hansen. Variable neighbourhood arc routing problem. *Computers and Operations Research*. Vol. 34, pp. 1097-1100, 1997.

[Moghadam et Seyedhosseini, 2010] B. F. Moghadam, S. M. Seyedhosseini. A particle swarm approach to solve vehicle routing problem with uncertain demand: A drug distribution case study. *International Journal of Industrial Engineering Computations*. Vol. 1, pp. 55-66, 2010.

[Mohamad et al, 2009] M. S. Mohamad, S. Omatu, S. Deris, M. Yoshioka1, A. Zainal. An Improved Binary Particle Swarm Optimisation for Gene Selection in Classifying Cancer Classes. *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*. pp. 495-502, Springer, 2009.

[Monette et al, 2007] J-N. Monette, Y. Deville, P. Dupont. A Position-Based Propagator for the Open-Shop Problem. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. *Lecture Notes in Computer Science*. Vol. 4510, pp. 186-199. 2007.

Bibliographie

- [Nabti et Meshoul, 2009] S. Nabti, S. Meshoul. Predator prey optimisation for snake based contour detection. *International Journal of Intelligent Computing and Cybernetics – IJICC*. Emerald Publishers. Vol. 2, N° 2, pp. 228-242, 2009.
- [Naimi ,2008] M. Naimi. Amélioration de la Performance des Algorithmes Evolutionnaires Multiobjectifs: Application au Problème de Sac à Dos Multidimensionnel Multiobjectif. Thèse de doctorat. Université Hassan 2 Casablanca, Maroc. 2008.
- [Nakano et al, 2010] S. Nakano, A. Ishigame, K. Yasuda. Consideration of Particle Swarm Optimization combined with tabu search. *Electrical Engineering in Japan*. Vol. 172, N° 4, pp. 31-37, 2010.
- [Nakano et al, 2007] S. Nakano, A. Ishigame, K. Yasuda. Particle swarm optimization based on the concept of tabu search. CEC 2007. IEEE Congress on Evolutionary Computation. 2007.
- [Nakib et al, 2008] A. Nakib, Y. Cooren, H. Oulhadj, P. Siarry. Magnetic Resonance Image Segmentation Based on Two-Dimensional Exponential Entropy and a Parameter Free PSO. *Artificial Evolution*. pp. 50-61. Springer, 2008.
- [Nauss, 1978] R. Nauss. The zero-one knapsack problem with multiple-choice constraints. *European Journal of Operational Research*. Vol. 2, N° 2, pp. 125-131, 1978.
- [Nelder et Mead, 1965] N.A. Nelder, R. Mead. A simplex method for function minimization. *Computer Journal*. Vol. 7, N° 4, pp. 308-313, 1965.
- [Nemhauser et Ulmann, 1969] G. L. Nemhauser, Z. Ulmann. Discrete dynamic programming and capital allocation. *Management Science*. Vol. 15, N° 9, pp. 494-505, 1969.
- [Newella, 1980] Newella. The heuristic of George Polya and its relation to artificial intelligence. A paper given at The International Symposium on the Methods of Heuristic. University of Bern, Switzerland, Sept. 15-18, pp.16. (Published in Groner et al. (1983), pp. 195-244. 1980.
- [Niknam et al, 2008] T. Niknam, B. Amiri, J. Olamaei, A. Arefi. An efficient hybrid evolutionary optimization algorithm based on PSO and SA for clustering. *Journal of Zhejiang University - Science A*. Vol. 10, N° 4, pp. 512-519, 2008.
- [Nunes de Castro et Timmis, 2002] L. Nunes de Castro and J. Timmis. An artificial immune network for multimodal function optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'2002)*. Vol. 1, pp. 669-674, Honolulu, Hawaii, May 2002.
- [Nunes de Castro et Von Zuben, 2002] L. Nunes de Castro and F.J. Von Zuben. Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*. Vol. 6, N° 3, pp. 239-251, 2002.
- [Olamaei et Niknam, 2008] J. Olamaei, T. Niknam, G. Gharehpetian. Application of particle swarm optimization for distribution feeder reconfiguration considering distributed generators. *Appl. Math. Comput*. Vol. 201, N° 1-2, pp. 575-586, 2008.
- [Osman et Laporte, 1996] I.H. Osman, G. Laporte. Metaheuristics: A bibliography. *Ann.*

Oper. Res. Vol. 63, N° 5, pp. 513-623, 1996.

[Otsu, 1979] N. Otsu. A threshold selection method for gray-level histogram. IEEE Trans. On System Man Cybernetics. Vol. SMC-9, N°. 1, pp. 62-66, 1979.

[Padberg et Rinaldi, 1987] M. Padberg, G. Rinaldi. Optimization of a 532 city symmetric traveling salesman problem by branch and cut. Operations Research Letters. Vol. 6, N° 1, pp. 1-7. 1987.

[Padberg et Rinaldi, 1991] M. Padberg et G. Rinaldi. A branch and cut algorithm for the resolution of large- scale symmetric traveling salesman problem. SIAM Review. Vol. 33, N° 1, pp. 60- 100, 1991.

[Pant et al, 2008] M. Pant, R. Thangaraj, A. Abraham. A new quantum behaved particle swarm optimization. Proceedings of the 10th annual conference on Genetic and evolutionary computation, pp. 87-94, 2008.

[Papadimitriou et Steiglitz, 1982] C.H. Papadimitriou, K. Steiglitz, Combinatorial optimization - algorithms and complexity. Prentice Hall, 1982.

[Park et al, 1996] K. Park, K. Lee, S. Park. An extended formulation approach to the edge-weighted maximal clique problem. European Journal of Operational Research. Vol. 95, N° 3, pp. 671-682, 1996.

[Parsopoulos et Vrahatis, 2004] KE. Parsopoulos, MN. Vrahatis. UPSO: A unified particle swarm optimization scheme. In: Lecture series on computer and computational sciences. Proceedings of international conference on computational methods in sciences and engineering (ICCMSE 2004), VSP International Science Publishers, Zeist, The Netherlands. Vol.1, pp. 868-873, 2004.

[Pavlyukevich, 2007] I. Pavlyukevich. Lévy flights, non-local search and simulated annealing. J. Computational Physics. Vol. 226, N° 2, pp. 1830-1844, 2007.

[Pearl, 1984] J. Pearl. Heuristics: intelligent search strategies for computer problem solving. pp. 3. Addison-Wesley Publ. Co, London, 1984.

[Pei, 2010] Y. Pei. A MOPSO Approach to Grid Workflow Scheduling. Asia-Pacific Conference on Wearable Computing Systems (APWCS). pp. 403- 406, 2010.

[Pisinger, 1995a] D. Pisinger. An expanding-core algorithm for the exact 0-1 knapsack problem. European Journal of Operational Research. Vol. 87. pp. 175-187, 1995.

[Pisinger, 1995b] D. Pisinger. A minimal algorithm for the multiple-choice knapsack problem. European Journal of Operational Research. Vol. 83, pp. 394-410, 1995.

[Pisinger, 2005] D. Pisinger. Where are the hard knapsack problems? Computers and Operations Research. Vol. 32, N° 9, pp. 2271-2284, 2005.

[Pongchairerks et Kachitvichyanukul, 2005] P. Pongchairerks, V. Kachitvichyanukul. A non-homogenous particle swarm optimization with multiple social structures. In: Proceedings of international conference on simulation and modeling. pp. A5-02, 2005.

[Pongchairerks, 2009] P. Pongchairerks. Particle swarm optimization algorithm applied to

scheduling problems. *ScienceAsia*. Vol. 35, N° 1, pp. 89-94, 2009.

[Prins, 2002] C. Prins. Efficient heuristics for the heterogeneous fleet multitrip vehicle routing problem. *Journal of Mathematical Modelling and Algorithms*. Vol. 1, N° 2. pp. 135-150, 2002.

[Psaraftis, 1983] H.N. Psaraftis. An exact algorithm for the single vehicle manyto- many immediate request dial-a-ride problem with time windows. *Transportation Science*. Vol. 17, pp. 351-357, 1983.

[Pu et al, 2007] X. Pu, Z. Yi, Z. Fang. Holistic and partial facial features fusion by binary particle swarm optimization. *Neural Computing and Applications*. Vol. 17, N° 5-6, pp. 481-488, 2008.

[Rajabioun, 2011] R. Rajabioun. Cuckoo Optimization Algorithm. *Applied Soft Computing*. Vol. 11, N° 8, pp. 5508-5518, 2011.

[Ralphs et al, 2001] T.K. Ralphs, L. Kopman, W.R. Pulleyblank, L.E. Trotter. On the capacitated vehicle routing problem. *Mathematical Programming*. Vol. 94, N° 2, pp. 343-359, 2001.

[Ratnaweera et al, 2004] A. Ratnaweera, SK. Halgamuge, HC. Watson. Self-organising hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Trans Evol Comput*. Vol. 8, N° 3, pp. 240-255, 2004.

[Rechenberg, 1965] I. Rechenberg. Cybernetic Solution Path of an Experimental Problem, Royal Aircraft Establishment Library Translation, 1965.

[Reeves, 1983] WT. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Trans Graphics*. Vol. 2, N° 2, pp. 91-108, 1983.

[Rego et al 1994] C. Rego et C. Roucairol. Le problème de tournées de véhicules : Etude et Résolution Approchée. Technical Report. INRIA, Février 1994.

[Reynolds et Frye, 2007] A. M. Reynolds, M. A. Frye. Free-flight odor tracking in *Drosophila* is consistent with an optimal intermittent scale-free search. *PLoS One*, 2, e354. 2007.

[Reynolds, 1987] C.W. Reynolds. Flocks, herds, and schools: a distributed behavioral model. *Computer Graphics. (Proc SIGGRAPH '87)*. Vol. 21, N° 4, pp. 25-34, 1987.

[Ribeiro et Maculan, 1994] C.C. Ribeiro, N. Maculan (Eds.). Applications of combinatorial optimization. *Annals of Operations Research*. Vol. 50, 1994.

[Robert, 2005] P. B. Robert. The Cuckoos. Oxford University Press. 2005.

[Sakarovitch, 1984] M. Sakarovitch. Optimisation combinatoire, Méthodes mathématiques et algorithmiques - Programmation discrète. Hermann. ISBN: 2-7056-5976-5, 1984.

[Saraç et Sipahioglu, 2007] T. Saraç, A. Sipahioglu. A Genetic Algorithm for the Quadratic Multiple Knapsack Problem. In: Mele, F., Ramella, G., Santillo, S., Ventriglia, F. (eds.) *BVAI 2007. LNCS*. Vol. 4729, pp. 490-498. Springer, Heidelberg, 2007.

[Savelsbergh, 1995] M.W.P. Savelsbergh. Local Search for Routing Problems with Time

Bibliographie

- Windows. *Annals of Operations Research*. Vol. 4, N° 1, pp. 285-305. 1995.
- [Schrijver, 1960] A. Schrijver. On the history of combinatorial optimization, 1960.
- [Serapiao et Mendes, 2009] A. B. S. Serapião, J. R. P. Mendes. Classification of Petroleum Well Drilling Operations with a Hybrid Particle Swarm/Ant Colony Algorithm. *Next-Generation Applied Intelligence*, pp. 301-310. Springer, 2009.
- [Sevkli et Sevilgen, 2008] Z. Sevkli, F. E. Sevilgen. A Hybrid Particle Swarm Optimization Algorithm for Function Optimization. *Applications of Evolutionary Computing*. pp. 585-595. Springer, 2008.
- [Sha et Lin, 2009] D. Y. Sha, Hsing Hung Lin. A particle swarm optimization for multi-objective flowshop scheduling. *The International Journal of Advanced Manufacturing Technology*. Vol. 45, N° 7-8, pp. 749-758, 2009.
- [Sharaf et El-Gammal, 2009] A. M. Sharaf, A.A.A. El-Gammal. A novel discrete multi-objective particle swarm optimisation (MOPSO) technique for optimal hybrid power filter compensator schemes. *International Journal of Power and Energy Conversion (IJPEC)*. Vol. 1, N°. 2/3, 2009.
- [Shelokar et al, 2004] P.S. Shelokar, V.K. Jayaraman, B.D. Kulkarni. An ant colony classifier system: application to some process engineering problems. *Computers & Chemical Engineering*. Vol. 28, N° 9, pp. 1577-1584, 2004.
- [Shelokar et al, 2007] P.S. Shelokar, P. Siarry, V.K. Jayaraman, B.D. Kulkarni. Particle swarm and ant colony algorithms hybridized for improved continuous optimization. *Applied Mathematics and Computation*. Vol. 188, N° 1, pp. 129-142, 2007.
- [Shen et al, 2007] Q. Shen, W-M Shi, W. Kong. Hybrid particle swarm optimization and tabu search approach for selecting genes for tumor classification using gene expression data. *Computational Biology and Chemistry*. Vol. 32, N° 1, pp. 53-60, 2007.
- [Shen. X et al, 2007] X. Shen, Y. Li, B. Zheng, Z. Dai. General Particle Swarm Optimization Based on Simulated Annealing for Multi-specification One-Dimensional Cutting Stock Problem. *Computational Intelligence and Security*. pp. 67-76. Springer, 2007.
- [Shi et Eberhart, 1998] Y. Shi, R.C. Eberhart. Parameter selection in particle swarm optimization. In: *Evolutionary Programming VII. Proc. EP98*. pp. 591-600. Springer, 1998.
- [Shi et Eberhart, 1999] Y. Shi, R.C. Eberhart. Empirical study of particle swarm optimization. In: *Proceedings of the Congress on Evolutionary Computation*. pp. 1945-1950, 1999.
- [Shi et Liang, 2007] X.H. Shi, Y.C. Liang, H.P. Lee, C. Lu & Q.X. Wang. Particle swarm optimization-based algorithms for TSP and generalized TSP. *Information Processing Letters*. Vol. 103, N° 5, pp. 169 -176. Elsevier, 2007.
- [Shlesinger et al, 1995] M. F. Shlesinger, G. M. Zaslavsky, U. Frisch. (eds) *Lévy Flights and Related Topics in Physics*. Springer, Berlin, 1995.
- [Shlesinger, 2006] M. F. Shlesinge. Search research. *Nature*. Vol. 443, pp. 281-282, 2006.

Bibliographie

- [Singh et Baghel, 2007] A. Singh, A.S. Baghel. A New Grouping Genetic Algorithm for the Quadratic Multiple Knapsack Problem. In: Cotta, C., van Hemert, J. (eds.) EvoCOP 2007. LNCS. Vol. 4446, pp. 210-218. Springer, Heidelberg, 2007.
- [Sinha et Zoltners, 1979] A. Sinha et A. A. Zoltners. The multiple-choice knapsack problem. Operations Research. Vol. 27, pp. 503-515, 1979.
- [Slagle, 1971] J. R. Slagle. Artificial intelligence: The heuristic programming approach. McGraw-Hill. pp. 3. New York, 1971.
- [Solso, 1979] R. L. Solso. Cognitive psychology. Harcourt Brace Jovanovich, Inc., New York. pp. 436, 1979.
- [Soundararajan et al, 2008] H. C. Soundararajan, J. Raman, R. Muthucumaraswamy. Modified Particle Swarm Optimizer with Adaptive Dynamic Weights for Cancer Combinational Chemotherapy. Advanced Data Mining and Applications. pp. 563-571. Springer, 2008.
- [Standard PSO2006] Standard PSO2006: <http://www.particleswarm.info/Programs.html>.
- [Sun et al, 2004] J. Sun, B. Feng, W. Xu. Particle Swarm Optimization with Particles Having Quantum Behavior. Proc. Congress on Evolutionary Computation, pp. 325-331, 2004.
- [Sundar et Singh, 2010] S. Sundar et A. Singh. A Swarm Intelligence Approach to the Quadratic Multiple Knapsack Problem. In Proc: K.W. Wong et al. (Eds.): ICONIP 2010, Part I, LNCS 6443. pp. 626-633. Springer-Verlag Berlin Heidelberg, 2010.
- [Suryadi et Kartika, 2011] D. Suryadi, E. K. Kartika. Viral Systems Application for Knapsack Problem. Third International Conference on Computational Intelligence, Communication Systems and Networks. DOI 10.1109/CICSyN.2011.16. pp. 11-16, 2011.
- [Syam, 1998] S. S. Syam. A dual ascent method for the portfolio selection problem with multiple constraints and linked proposals. European Journal of Operational Research. Vol. 108, N° 1, pp. 196-207. July, 1998.
- [Taillard, 1999] E.D. Taillard. A heuristic column generation method for the heterogeneous fleet VRP. RAIRO-Operations Research. Vol. 33, pp. 1-14, 1999.
- [Tein et Ramli, 2010] L.H. Tein, R. Ramli. Recent advancements of nurse scheduling models and a potential path. In Proc. 6th IMT-GT Conference on Mathematics, Statistics and its Applications (ICMSA 2010). pp. 395-409, 2010.
- [Tillman et al, 1980] F. Tillman, C. Hwang et W. Kuo. Optimisation of System Reliability. Marcel Dekker (editeurs), New York, 1980.
- [Toth, 1980] P. Toth. Dynamic programming algorithms for the zero-one knapsack problem. Computing. Vol. 25. pp. 29-45, 1980.
- [TSPLIB] TSPLIB, <http://comopt.ifl.uni-idelberg.de/software/TSPLIB95/>.
- [Valdez et al, 2008] F. Valdez, P. Melin, O. Castillo. A New Evolutionary Method Combining Particle Swarm Optimization and Genetic Algorithms Using Fuzzy Logic. Soft Computing for Hybrid Intelligent Systems. pp. 347-361, Springer.

Bibliographie

- [Valdez et al, 2009] F. Valdez, P. Melin, O.Castillo. Fuzzy Logic for Combining Particle Swarm Optimization and Genetic Algorithms: Preliminary Results. MICAI 2009: Advances in Artificial Intelligence. pp. 444-453, Springer.
- [Valdez et al, 2010] F. Valdez, P. Melin, O. Castillo. Evolutionary method combining Particle Swarm Optimization and Genetic Algorithms using fuzzy logic for parameter adaptation and aggregation: the case neural network optimization for face recognition. International Journal of Artificial Intelligence and Soft Computing. Vol. 2, N° 1/2, pp. 77-102, 2010.
- [Valian et al, 2011] E. Valian, S. Mohannaet S. Tavakoli. Improved cuckoo search algorithm for feedforward neural network training, Int. Artificial Intelligence and Applications. Vol. 2, N° 3, pp. 36-43, 2011.
- [Van den Bergh, 2001] F. Van den Bergh. An Analysis of Particle Swarm Optimizers. PhD Thesis. University of Pretoria. Nov, 2001.
- [Viswanathana et al, 2000] G.M. Viswanathana, V. Afanasyev, S. V. Buldyrev, S. Havlin, M.G.E. da Luz, E.P. Raposo, H. Eugene Stanley. Lévy flights in random searches. Physica A. Vol. 282, N° 1-2, pp. 1-12, 2000.
- [Viswanathana et al, 2002] G. M. Viswanathan, F. Bartumeus, S. V. Buldyrev, J. Catalan, U. L. Fulco, S. Havlin, M.G.E. da Luz, M. L. Lyra, E.P. Raposo, H. E. Stanley. Lévy flight random searches in biological phenomena. Physica A. Vol. 314, pp. 208-213, 2002.
- [Voudouris et Tsang ,1999] C. Voudouris, E. Tsang. Guided local search and its application to the traveling salesman problem. Eur J Oper Res. Vol. 113, pp. 469-499, 1999.
- [Wang et al, 2008] L. Wang, Y. Hong, F. Zhao, D. Yu. A Multiagent Genetic Particle Swarm Optimization. Advances in Computation and Intelligence. pp. 659-668. Springer, 2008.
- [Wang et al, 2009] L. Wang, X-T. Wang, M-R. Fei. An adaptive mutation-dissipation binary particle swarm optimisation for multidimensional knapsack problem. Int. J. of Modelling, Identification and Control. Vol. 8, N° 4, pp. 259 - 269, 2009.
- [Wang et Zhou, 2007] J. Wang, Y. Zhou. Quantum-behaved particle swarm optimization with generalized local search operator for global optimization. Advanced Intelligent Computing Theories and Applications With Aspects of Artificial Intelligence. pp. 851-60, Springer.
- [Wang. J et al, 2008] J. Wang, Z. Kuang , X. Xu, Y. Zhou. Discrete particle swarm optimization based on estimation of distribution for polygonal approximation problems. Expert Systems with Applications: An International Journal. Vol. 36, N° 5, pp. 9398-9408. July, 2009.
- [Wen et Liu, 2005] W. Wen, G. Liu. Swarm Double-Tabu Search. Advances in Natural Computation. pp. 1231-1234. Springer, 2005.
- [Wilson, 1975] E. O. Wilson. Sociobiology: The new synthesis, Ist ed. Cambridge, Mass: Belknap Press of Harvard University Press. pp. 697, 1975.
- [Witucki et al, 1997] M. Witucki, P. Dejax, M. Haouari. Un modèle et un algorithme de

résolution exacte pour le problème de tournées de véhicules multipériodique. Ecole Centrale Paris, Laboratoire Productique-Logistique, Grande Voie des Vignes, 92295 Châtenay-Malabry. Ecole Polytechnique de Tunisie, 2070 La Marsa. Tunisia, 1997.

[Wolpert et Macready, 1997] D. H. Wolpert , W. G. Macready. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation. Vol. 1, N° 1, pp. 67-82, 1997.

[Xu et al, 2008] Y. Xu, Q. Wang, J. Hu. An improved Discrete Particle Swarm Optimization Based on Cooperative Swarms. International Conference on Web Intelligence and Intelligent Agent Technology. IEEE/WIC/ACM. pp. 79-82, 2008.

[Xuong, 1992] N. H. Xuong. Mathématiques Discrètes et Informatique. Masson, Paris, 1992.

[Yang et al, 2004] S.Y. Yang, M. Wang, L.C. Jiao. A Quantum Particle Swarm Optimization. Proceeding of the 2004 IEEE Congress on Evolutionary Computation. Vol. 1, pp. 320-324, 2004.

[Yang et al, 2008] F. Yang, C. Zhang, T. Sun. Comparison of Particle Swarm Optimization and Genetic Algorithm for HMM training. 19th International Conference on Pattern Recognition, 2008.

[Yang et Deb, 2009] X-S Yang, S. Deb. Cuckoo search via Lévy flights. World Congress on Nature & Biologically Inspired Computing (NaBIC 2009). pp. 210-214, 2009.

[Yang et Deb, 2010] X-S Yang, S. Deb. Engineering optimisation by cuckoo search. Int. J. Mathematical Modelling and Numerical Optimisation. Vol. 1, N° 4, pp. 330-343, 2010.

[Yang, 2005] X. Yang. Engineering optimizations via nature-inspired virtual bee algorithms. Lecture Notes Comput Sci. Vol. 3562, pp. 317-323, 2005.

[Yang, 2008] X. S. Yang. Nature-Inspired Metaheuristic Algorithms, Luniver Press, 2008.

[Yang. X.S, 2005] X. S. Yang. Biology-derived algorithms in engineering optimization. Chapter 32, in Handbook of Bioinspired Algorithms and Applications (eds Olariu & Zomaya), Chapman & Hall / CRC. 2005.

[Yasuda et Iwasaki, 2004] K. Yasuda, N. Iwasaki. Adaptive particle swarm optimization using velocity information of swarm. Proceedings of the IEEE Conference on System, Man and Cybernetics. IEEE Press. pp. 3475-3481, 2004.

[Yin, 2007] P-Y Yin. Multilevel minimum cross entropy threshold selection based on particle swarm optimization. Applied Mathematics and Computation. Vol. 184, N° 2, pp. 503-513, 2007.

[Yu et al, 2008] H. Yu, G. Gu, H. Liu, J. Shen, C. Zhu. A Novel Discrete Particle Swarm Optimization Algorithm for Microarray Data-Based Tumor Marker Gene Selection. International Conference on Computer Science and Software Engineering, pp. 1057-1060, 2008.

[Zahara et al, 2004] E. Zahara, S. Fan, D. Tsai. Optimal multi-thresholding using a hybrid optimization approach. Pattern Recognition Letters. Vol. 26, N° 8, pp. 1082- 1095, 2005.

Bibliographie

- [Zhan et Zhang, 2009] Z-H. Zhan, J. Zhang. Discrete Particle Swarm Optimization for Multiple Destination Routing Problems. In proc. M. Giacobini et al. (Eds.): EvoWorkshops, LNCS 5484. pp. 117-122. Springer, 2009.
- [Zhang et al, 2007] X. Zhang, J. Wang, Z. Fan, B. Li. Spatial Clustering with Obstacles Constraints Using Ant Colony and Particle Swarm Optimization. Emerging Technologies in Knowledge Discovery and Data Mining. Vol. 4819, pp. 344-356. Springer, 2007.
- [Zhang et al, 2008] X. Zhang, H. Yin, H. Zhang, Z. Fan. Spatial Clustering with Obstacles Constraints by Hybrid Particle Swarm Optimization with GA Mutation. Advances in Neural Networks - ISNN 2008. pp. 569-578. Springer, 2008.
- [Zhang et al, 2009] Y. Zhang, M. Zhang, Y.C Liang. A hybrid ACO/PSO algorithm and its applications. International Journal of Modelling, Identification and Control (IJMIC). Vol. 8, N° 4, 2009.
- [Zhang et Qiu, 2010] X. Zhang, H. Qiu. Hybrid particle swarm optimisation with k-centres method and dynamic velocity range setting for travelling salesman problems. International Journal of Bio-Inspired Computation. Vol. 2, N° 1, pp. 34-41, 2010.
- [Zhang.J et al, 2007] J. Zhang, H. Chung, W. Lo. Engineering optimizations via nature-inspired virtual bee algorithms. IEEE Trans Evol Comput. Vol. 11, N° 3, pp. 326-335, 2007.
- [Zhao et al, 2007] Y Zhao, Z Fang, K. Wang, H. Pang. Multilevel Minimum Cross Entropy Threshold Selection Based on Quantum Particle Swarm Optimization. ACIS International Conference on Software Engineering. Artificial Intelligence, Networking and Parallel/Distributed Computing. Vol. 2, pp. 65-69, 2007.
- [Zheng, 2010] S. Zheng. An intensive restraint topology adaptive snake model and its application in tracking dynamic image sequence. Inf. Sci. Vol. 180, N° 16, pp. 2940-2959, 2010.
- [Zhiming et al, 2008] L. Zhiming, W. Cheng, L. Jian. Solving Constrained Optimization via a Modified Genetic Particle Swarm Optimization. First International Workshop on Knowledge Discovery and Data Mining (WKDD 2008). pp. 217-220, 2008.
- [Zhong et al, 2007] W-l. Zhong, J. Zhang, W-N. Chen. A novel discrete particle swarm optimization to solve traveling salesman problem. IEEE Congress on Evolutionary Computation. CEC 2007. pp. 3283-3287, 2007.
- [Zhong et Young, 2010] T. Zhong et R. Young. Multiple Choice Knapsack Problem: Example of planning choice in transportation. Evaluation and Program Planning. Vol. 33. N° 2. pp. 128-137. Mai, 2010.
- [Zhong et Zhang, 2007] W-l. Zhong, J. Zhang, W-N. Chen. A novel discrete particle swarm optimization to solve traveling salesman problem. IEEE Congress on Evolutionary Computation. CEC 2007. pp. 3283-3287, 2007.
- [Zhou et al, 2007] Y. Zhou, J. Wang, J. Yin. A Discrete Estimation of Distribution Particle Swarm Optimization for Combinatorial Optimization Problems. Proceedings of the Third International Conference on Natural Computation (ICNC 2007). Vol. 4, pp. 80-84, 2007.

Bibliographie

[Zhou et al, 2008] Y. Zhou, J. Wang, J. Yin. Genetic Particle Swarm Optimization Based on Multiagent Model for Combinatorial Optimization Problem. IEEE International Conference on Networking, Sensing and Control. ICNSC 2008, pp.293-297, 2008.

[Zhou et al, 2008a] Y. Zhou, Z. Kuang, J. Wang. A Chaotic Neural Network Combined Heuristic Strategy for Multidimensional Knapsack Problem. In: Proc. L. Kang et al. (Eds.): ISICA 2008, LNCS 5370. pp. 715-722. Springer, 2008.

[Zou et al, 2011] D. Zou, L. Gao, S. Li, J. Wu. Solving 0-1 knapsack problem by a novel global harmony search algorithm. Applied Soft Computing. The Impact of Soft Computing for the Progress of Artificial Intelligence. Vol. 11, N° 2, pp. 1556-1564. March, 2011.