الجمهوريـة الجزائـريـة الديمقراطيـة الشعبيـة
**République Algérienne Démocratique et Populaire**
وزارة التعليـم العالـي والبحـث العلمـي
**Ministère de l'Enseignement Supérieur et de la Recherche Scientifique**

**Nº Réf :……**

**Centre Universitaire**
**Abdelhafid Boussouf Mila**

**Institut des Sciences et Technologie** **Département de Mathématiques et Informatique**

# Mémoire préparé en vue de l'obtention du diplôme de Master

**En : Informatique**

**Spécialité : Sciences et Technologies de l'Information et de la Communication (STIC)**

# Thème :
# *Nouvelles variantes des méta-heuristiques compactes*

**Préparé par :** Chetioui Besma
Hallel Nawel

**Soutenue devant le jury**

| | | |
|---|---|---|
| **Président** | Bouchemal Nardjes | **Grade** MCB |
| **Examinateur** | Abdderrezak Samira | **Grade** MAA |
| **Encadreur** | Khalfi Souheila | **Grade** MAB |

**Année Universitaire : 2018/2019**


بسم الله الرحمن الرحيم

# Table des matières

# Table des figures

# Liste des tableaux

# Liste des Algorithmes

# Glossaire

**AG** algorithms génétiques.

**BBOB** black-box-optimization benchmarking.

**cGA** compact genetic algorithm.
**cPSO** compact particle swarm optimization.

**cDE** compact differential evolution.
**c_WWO** compact water waves optimization.
**cTLBO** compact Teaching- Learning-Based Optimization.
**cEAs** compact evolutionary algorithms.
**COCO** comparing continuous optimizer.

**DEcDE** disturbed exploitation compacte differential evolution.

**EA** evolutionary algorithms.

**ED** Evolution différentielle.

**ERT** expected runnig time.

**IA** intelligence artificielle.

**ne_cGA** elitiste non persistante compact genetic algorithm.
**ne_rcGA** élitiste non persistante réel compact genetic algorithm.
**ne_cDE** élitiste non persistante compact differential evolution.

**POC** problème d'optimisation combinatoire.
**PSO** particle swarm optimization.
**PV** probability vector.
**PDF** probability of density function.
**pe_cGA** élitiste persistant compact genetic algorithm.
**pe_rcGA** élitiste persistant réel compact genetic algorithm.

**rcGA** réel compact génétic Algorithme.
**RO** recherche opérationnelle.
**SI** swarm intelligence.
**VNS** variable neighbourhood search.
**WWO** water waves Optimization.

*À mes chers parents, À mes frères et mes sœurs,*
*je dédié ce modeste travail,*
*Besma.*

*À mes chers parents, À mes frères et mes sœurs,*
*À ma chère grand-mère,*
*je dédié ce modeste travail,*
*Nawel.*

# *Remerciement*

*Tout travail réussi dans la vie nécessite en premier lieu la Bénédiction de* ***ALLAH****, et ensuite l'aide et le support de plusieurs personnes.*

*Un remerciement particulier à notre encadreur Khalfi Souheila pour sa présence, son aide et surtout pour ses précieux conseils durant toute la période du travail.*

*Nous remercions également les membres de jury $M^{me}$ Bouchemal Nardjes pour avoir accepté de présider et $M^{me}$ Abdderrezak Samira pour avoir accepté d'examiner notre travail.*

*Nous remercions aussi le corps professoral et administratif de l'institut des sciences et de la technologie pour la richesse et la qualité de leur enseignement, qui déploient de grandes efforts pour assurer à leurs étudiants une formation actualisée.*

*Nous tenons à remercier et à adresser notre reconnaissance à toutes les personnes qui ont participé de prés ou de loin à la réalisation de ce travail.*

# Résumé

Les Méta-heuristiques constituent une nouvelle génération de méthodes approchées qui résolvent souvent la plupart des problèmes d'optimisation et peuvent s'adapter à n'importe quel problème quel que soit sa complexité ou sa nature. Ces méthodes étaient caractérisées par l'existence de deux catégories : les méthodes à base de solution unique et les méta-heuristiques à base de population. Cette dernière catégorie est caractérisée par une population dont sa taille évolue en fonction de la dimension du problème à résoudre. Cependant, ce facteur est devenu sa plus grande faiblesse dans certains domaines de la vie nécessitant l'utilisation d'appareils à mémoire limitée. Par conséquent, les chercheurs ont eu recours aux méta-heuristique compacte car elles constituent une bonne alternative. Le but de ce mémoire est de contribuer au domaine et de laisser une empreinte permettant de résoudre des problèmes en temps réel. Donc, nous avons proposée six variantes qui se basent sur des algorithmes compacts existants à savoir : cPSO_cDE$v1$, cPSO_cDE$v2$, pe_cDE$v2$, ne_cDE$v2$, DEcDE_$CR_{binomiale}$ et une version compacte de la nouvelle méta-heuristique WWO (*water wave optimisation*). Ces approches sont évaluées et testée sur l'ensemble de fonctions BBoB 2015. Les résultats trouvés après une analyse des algorithmes compacts prouvent la supériorité des approches proposées. Surtout le DEcDE_$CR_{binomiale}$ et le cPSO_cDE$v2$ qui ont pu fournir des résultats qui surpassent les algorithmes compacts de base.

**Mots clés** : Optimisation, Méta-heuristiques, Méta-heuristiques compactes, Matériel limité en mémoire.

# Abstract

Meta-heuristics are a new generation of approximate methods that often solve most optimization problems and can adapt to any problem regardless of its complexity or nature. These methods were characterized by the existence of two categories : single-solution methods and population-based meta-heuristics. This last category is characterized by a population whose size evolves according to the dimension of the problem to be solved. However, this factor has become its greatest weakness in some areas of life requiring the use of limited memory devices. As a result, researchers have used compact meta-heuristics as a good alternative. The purpose of this thesis is to contribute to the field and leave a fingerprint to solve problems in real time. So, we have proposed six variants that are based on existing compact algorithms namely : cPSO_cDE $v1$, cPSO_cDE $v2$, pe_cDE $v2$, ne_cDE $v2$, DEcDE_$CR_{binomial}$ and a compact version of the new meta-heuristic WWO( textit water wave optimization). These approaches are evaluated and tested on the BBoB 2015 function set. The results found after an analysis of the compact algorithms prove the superiority of the proposed approaches. Especially the DEcDE_$CR_{binomial}$ and the cPSO_cDE$v2$ that could provide results that surpass the basic computational algorithms.

**Key words** :Optimization, meta-heuristics, compact meta-heuristics, limited-memory hardware .

# ملخص

الخوارزميات التقريبية هي جيل جديد من الطرق التقريبية التي غالباً ما تحل معظم مشكلات التحسين ويمكنها التكيف مع أي مشكلة تحسين، بغض النظر عن تعقيدها أو طبيعتها. وقد تميزت هذه الطرق بوجود فئتين، طرق تعتمد على حل واحد، وطرق تعتمد على مجموعة من الحلول. تتميز هذه الفئة الأخيرة بتطور حجم مجموعة الحلول وفقًا لبعد المشكلة المراد حلها. ومع ذلك، فقد أصبح هذا العامل أكبر نقاط الضعف في بعض مجالات الحياة التي تتطلب استخدام أجهزة ذاكرة محدودة. نتيجة لذلك، استخدم الباحثون الخوارزميات التقريبية المدمجة كبديل جيد. والغرض من هذه الأطروحة هو المساهمة في هذا المجال وترك بصمة من اجل المساهمة في حل مشاكل الوقت الحقيقي. لذلك، اقترحنا ستة متغيرات تستند إلى خوارزميات مضغوطة موجودة، كذلك اخترنا خوارزمية تقريبية جديدة لم يتم ضغطها من قبل مستوحاة من أمواج المياه وقمنا بضغطها لنتحصل في الاخير على النسخة المضغوطة من خوارزمية أمواج المياه.

**الكلمات المفتاحية**: التحسين، الخوارزميات التقريبية، الخوارزميات المدمجة، أجهزة ذات ذاكرة محدودة.

# Introduction Générale

Les méta-heuristiques forment une famille d'algorithmes stochastiques destinée à la résolution de problèmes "d'optimisation difficile". Ces méthodes ont été conçues afin de résoudre un panel très large de problèmes, sans pour autant nécessiter de changements profonds dans les structures des algorithmes. Ces méthodes sont inspirées par des analogies avec la physique (recuit simulé), la génétique (algorithmes évolutionnaires), l'éthologie (colonies de fourmis) et (l'Optimisation par Essaim Particulaire). Ces algorithmes d'intelligence informatique ont également été utilisés avec succès pour résoudre des problèmes d'optimisation dans les domaines de l'ingénierie, de la finance et de la gestion depuis quelques années.

Mais, dans de nombreuses applications du monde réel, un problème d'optimisation doit être résolu malgré le fait qu'un dispositif informatique à pleine puissance peut ne pas être disponible en raison de contraintes de coût et / ou d'espace. Cette situation est typique de la robotique et des problèmes de contrôle. Par exemple, un robot aspirateur commercial. Le processus d'apprentissage peut être de plusieurs types, par ex : un apprentissage de réseau de neurones. Bien que le robot doit contenir un noyau de calcul, par ex. dans une carte de contrôle pas chère, il ne peut évidemment pas contenir tous les composants d'alimentation d'un ordinateur moderne, car ils augmenteraient le volume, la complexité et le coût de l'ensemble du périphérique.

Ainsi, une méta-heuristique d'optimisation traditionnelle peut être inadéquate dans ces circonstances. Afin de surmonter cette catégorie de problèmes, les "algorithmes compacts" sont une réponse prometteuse aux limitations matérielles car ils utilisent un compromis efficace pour présenter certains avantages des algorithmes basés sur une population, sans stocker une population réelle de solutions.
Les algorithmes compacts simulent le "comportement" des algorithmes basés sur la population en utilisant leur " représentation probabiliste" au lieu d'une population de solutions. Par conséquent, les algorithmes compacts nécessitent moins de mémoire pour stocker le nombre de paramètres par rapport à leurs structures correspondantes basées sur la population.

Donc, ce travail se concentre sur la découverte de nouvelles approches de méta-

heuristiques compactes. Afin de présenter notre travail, nous l'avons organisé de la manière suivante :

Le **premier chapitre** (1), nous donnons une définition de l'optimisation combinatoire, introduisons la théorie de la complexité et pose les bases théoriques qui seront utilisées par la suite. Nous ferons en particulier une description des problèmes d'optimisation combinatoire, et quelques problèmes classiques, ainsi que les différentes classe de problème. Ensuite nous présentons les familles de méthodes de résolution. A la fin de ce chapitre nous détaillons les méta-heuristiques basées sur la population les plus populaires utilisées dans l'optimisation continue.

Dans le **second chapitre** (2), nous introduisons la classe des algorithmes compacts, qui simulent le comportement des algorithmes basés sur la population en utilisant, au lieu d'une population de solutions, sa représentation probabiliste.

Dans le **troisième chapitre** (3), nous présentons les principales contributions de notre travail. Nous commençons par les motivations qui nous ont poussée à choisir de découverte de nouvelles approches de méta-heuristiques compactes. Ensuite, nous introduisons quelques nouvelles algorithmes que nous avons atteints.

le **dernier chapitre** (4) présente les résultats expérimentaux des approches proposées que nous avons testées et validées a l'aide de l'environnement COCO, et le langage de programmation MATLAB. Ensuite nous discutons les résultats de la comparaison des variantes que nous avons proposées avec les méta-heuristiques compactes décrites au chapitre (2).

En dernier lieu, nous terminerons notre mémoire par une conclusion générale qui récapitule notre travail et les résultats obtenus tout en suggérant d'éventuelles perspectives.

# Chapitre 1

# Optimisation combinatoire et méta-heuristique

## Sommaire

# Introduction

L'optimisation combinatoire est une voie d'études importante, qui regroupe une large classe de problèmes ayant des applications dans de nombreux domaines applicatifs, elle occupe une place très considérable en recherche opérationnelle , en mathématiques discrètes et en informatique. Cela justifie son importance sans oublier la grande difficulté des problèmes d'optimisation.

Dans ce chapitre, nous allons décrire brièvement quelques notions de l'optimisation combinatoire, ses problèmes classiques, ainsi que quelques méthodes de résolutions en particulier les méta-heuristiques.

## 1.1 Optimisation combinatoire

L'optimisation combinatoire est une technique mathématique, qui consiste à minimiser ou maximiser une fonction objectif (Coût, temps, distance, etc.) dont le but est de rechercher et définir une solution optimale la plus appropriée pour l'optimisation à partir d'un ensemble de solutions possible [1].

## 1.2 Définition d'un problème d'optimisation combinatoire

Un problème d'optimisation est défini par un ensemble de variables, une fonction objective sous certaines contraintes. L'espace de recherche est l'ensemble des solutions possibles du problème. Résoudre un problème d'optimisation consiste à trouver la meilleure solution soit en minimisant ou maximisant la fonction objective de problème posé [1]. Lorsque le domaine de valeurs des variables est discret, on parle alors de problème d'optimisation combinatoire (POC). Formellement un problème d'optimisation est défini comme suit :

$$(P) : \begin{cases} \min\limits_{s \in S} f(s) \end{cases} \tag{1.1}$$

Où ;
$S$ : l'ensemble des solutions réalisables ou admissibles.
$f : S \to \mathbb{R}$ la fonction objectif à minimiser [2].
Le problème de minimisation $(P)$ se ramène facilement à un problème de maximisation telle que :

$$\begin{cases} \max\limits_{s \in S} f(s) = -\min(-f(s)) \end{cases} \tag{1.2}$$

Pour un problème de minimisation, il s'agit de trouver une solution $s \in S$ réalisable, qui minimise la fonction $f$ [2].

Une solution $s \in S$ pour le problème 1.1 peut être :

- **Minimum local** :Une solution $s \in S$ est un minimum local pour la fonction objectif $f$ s'il existe un voisinage $N(s)$, tel que $\forall s' \in N(s), f(s) \leqslant f(s')$ [3].
- **Minimum global** :Une solution $s \in S$ est un minimum global pour la fonction objectif $f$ si $\forall s' \in N(s), f(s) \leqslant f(s')$ [3].
- **Voisinage** :le voisinage est une fonction notée $N$ qui associe un sous ensemble de $S$ à toute solution $s$, les voisins de $s$ sont $s' \in N(s)$ [3].

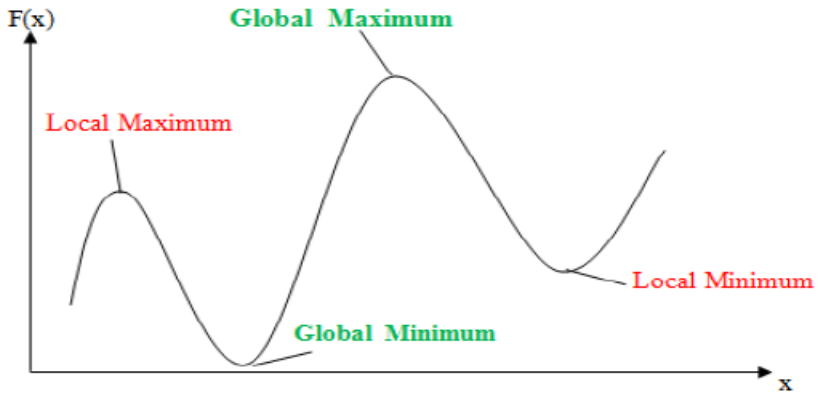



FIGURE 1.1 – Différence entre un optimum global et des optima locaux

## 1.3 Quelques problèmes classiques en optimisation combinatoire

### 1.3.1 Le problème du voyageur de commerce

Le problème du voyageur de commerce, étudie depuis le 19 me siècle, est l'un des plus anciens problèmes d'optimisation combinatoire. Il s'agit d'un représentant de commerce ayant $n$ villes à visiter, qui souhaite établir une tournée en passant exactement une fois par chaque ville puis revenir à son point de départ pour un moindre coût, c'est-à-dire en parcourant la plus petite distance possible.
La notion de distance peut être remplacée par d'autres notions comme le temps qu'il met ou l'argent qu'il dépense pour effectuer la tache [2].

### 1.3.2 Le problème du sac à dos

Le problème du sac à dos fait partie des problèmes d'optimisation combinatoire les plus étudiés en raison de ces nombreuses applications dans le monde réel. Il s'agit d'un

problème de sélection qui consiste à maximiser un critère de qualité sous une contrainte linéaire de capacité de ressource. Étant donné un sac à dos de capacité maximale $b$, et $n$ objets possédant chacun un poids et une valeur, le problème qui se pose est le suivant : comment remplir le sac à dos de sorte que le poids total des objets choisis n'excède pas sa capacité $b$, tout en maximisant la valeur totale des objets [2].

### 1.3.3 Le problème d'affectation

Le problème d'affectation consiste à établir des liens entre les éléments de deux ensembles distincts, de façon à minimiser un coût tout en respectant les contraintes d'unicité de lien pour chaque élément.

Étant donnés $m$ tâches et $n$ ouvriers, l'affectation de la tâche $i$ à l'ouvrier $j$ entraine un coût de réalisation noté $c_{ij}$.
Cette dernière se fait de sorte que :

- Chaque ouvrier $j$ ait une seule tâche.
- Chaque tâche $i$ est attribuée à un seul ouvrier.

Donc, le problème consiste à trouver une affectation de coût minimum [2].

### 1.3.4 Le problème d'ordonnancement

Le problème d'ordonnancement consiste à séquencer et à placer dans le temps, un ensemble d'activités (entités élémentaires de travail), compte tenu de contraintes temporelles (délais, contraintes d'enchaînements, ...) et de contraintes portant sur l'utilisation et la disponibilité des ressources requises par les activités, pour optimiser un ou plusieurs objectifs (délai total de réalisation du projet, coût total du projet, ...) [2].

Il s'agit d'un problème de satisfaction de contrainte qui trouve ses applications dans divers domaines tels que la gestion de projets, ateliers de production, ... Contrairement aux autres problèmes que nous avons définis précédemment, ce problème ne fait pas référence à un problème totalement défini pour lequel il existe une formulation mathématique directe, mais plutôt à une famille de problèmes. En effet, un problème d'ordonnancement est défini par la donnée des activités et des ressources qui le constituent, ces éléments peuvent être de nature très variée [2].

## 1.4 Complexité algorithmique

La complexité d'un problème est équivalente à la complexité du meilleur algorithme qui résout ce problème. C'est la raison pour laquelle elle est considérée comme un critère crucial pour le choix de la méthode de résolution adéquate pour le problème en main [4].

Pour cela, avant d'aborder les différentes méthodes de résolution des problèmes d'optimisation combinatoire, nous introduisons quelques définitions et notions sur la complexité des (POC).

### 1.4.1 Notions sur la complexité

La complexité d'un algorithme est un concept fondamental qui permet de mesurer sa performance asymptotique dans le pire et le meilleur des cas [2]. Généralement, le temps d'exécution est le facteur majeur qui détermine l'efficacité d'un algorithme, alors la complexité en temps d'un algorithme est le nombre d'instructions nécessaires (affectation, comparaison, opérations algébriques, lecture et écriture, etc.) que comprend cet algorithme pour une résolution d'un problème quelconque [5].

### 1.4.2 Classes de problèmes

On distingue quatre classes de problèmes :

- **Classe** $P$ : La classe $P$ contient tous les problèmes relativement faciles c'est à dire ceux pour lesquels on connaît des algorithmes efficaces. Plus formellement, ce sont les problèmes pour lesquels on peut construire une machine déterministe (e.g. une machine de Turing [1]) dont le temps d'exécution est de complexité polynomiale (le sigle $P$ signifie $\prec$ *Polynomial time* $\succ$ ) [6].

- **Classe** $NP$ : La classe $NP \prec Nondeterministic\ Polynomial\ time \succ$ [6] contient les problèmes de décision qui peuvent être décidés sur une machine non déterministe en temps polynomial. C'est la classe des problèmes qui admettent un algorithme polynomial capable de tester la validité d'une solution du problème. Intuitivement, les problèmes de cette classe sont les problèmes qui peuvent être résolus en énumérant l'ensemble de solutions possibles et en les testant à l'aide d'un algorithme polynomial [5].

- **Classe** $NP - complet$ : La classe $NP - complet$ parmi l'ensemble des problèmes appartenant à $NP$, il en existe un sous ensemble qui contient les problèmes les plus difficiles : on les appelle les problèmes $NP - complets$. Un problème $NP - complet$ possède la propriété que tout problème dans $NP$ peut être transformé (réduit) en

1. Une machine de Turing est constituée d'un ensemble fini de bandes composées d'un nombre infini de cellules. Chaque cellule, si elle n'est pas vide, contient un symbole représentant une information nécessaire au calcul. Chaque bande ne comporte qu'un nombre fini de cellules non vides. Sur chacune des bandes une tête de lecture-écriture se déplace de cellule en cellule faisant ainsi évoluer les informations contenues dans les bandes et le comportement de la est entièrement décrit par une table finie représentant les actions possibles des têtes de lecture-écriture.

celui-ci en temps polynomial. C'est à dire qu'un problème est $NP-complet$ quand tous les problèmes appartenant à $NP$ lui sont réductibles. Si on trouve un algorithme polynomial pour un problème $NP-complet$, on trouve alors automatiquement une résolution polynomiale de tous les problèmes de la classe $NP$ [5].

– **Classe** $NP-difficile$ : Un problème est $NP-difficile$ s'il est plus difficile qu'un problème $NP-complet$, c'est à dire s'il existe un problème $NP-complet$ se réduisant à ce problème par une réduction de Turing [5].

## 1.5 Approches de résolution

L'optimisation combinatoire occupe une place très importante en recherche opérationnelle, en mathématiques discrètes et en informatique. Bien que les problèmes d'optimisation combinatoire soient souvent faciles à définir, ils sont généralement difficiles à résoudre. Étant donné l'importance de ces problèmes, de nombreuses méthodes de résolution ont été développées en recherche opérationnelle (RO) et en intelligence artificielle(IA) [7] .
Ces méthodes peuvent être classées sommairement en deux grandes catégories illustrées dans la (figure 1.2) :

– Les méthodes exactes (complètes) qui garantissent la complétude de la résolution.
– Les méthodes approchées (incomplètes) qui perdent la complétude pour gagner en efficacité

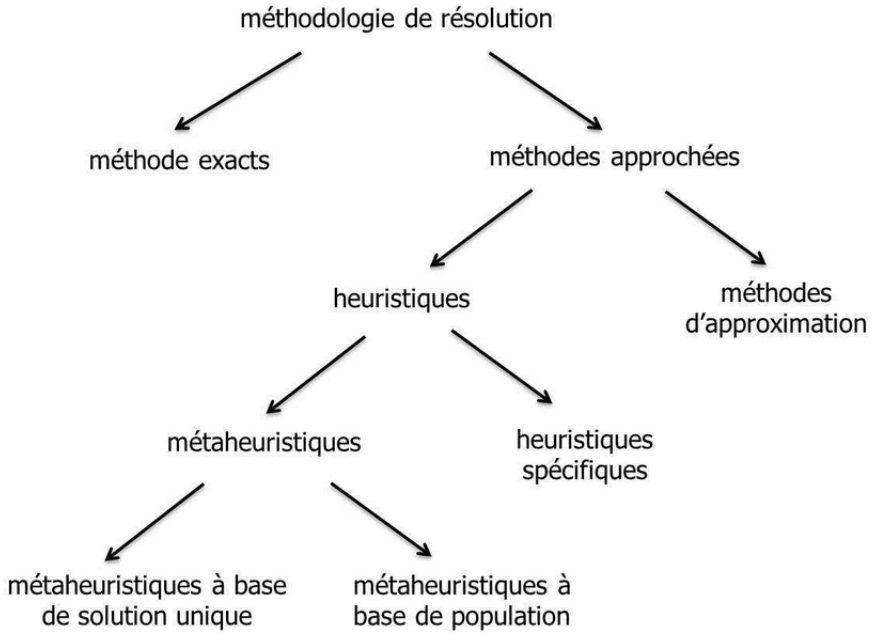



FIGURE 1.2 – Classification des méthodes d'optimisation

### 1.5.1 Les méthodes exactes

Les méthodes exactes cherchent à trouver de manière certaine la solution optimale en examinant de manière explicite ou implicite la totalité de l'espace de recherche. Elles ont l'avantage de garantir la solution optimale néanmoins le temps de calcul nécessaire pour atteindre cette solution peut devenir très excessif en fonction de la taille du problème (explosion combinatoire) et le nombre d'objectifs à optimiser. Ce qui limite l'utilisation de ce type de méthode aux problèmes bi-objectifs de petites tailles. Ces méthodes génériques sont : *Branch and " Bound / Cut / Price ", Cut and Price*. D'autres méthodes sont moins générales, comme :*La programmation dynamique, simplexe, La programmation linéaire en nombre entiers, L'algorithme* $A^*$ [8]

### 1.5.2 Les méthodes approchées

Une méthode approchée est une méthode d'optimisation qui a pour but de trouver une solution réalisable de la fonction objectif en un temps raisonnable, mais sans garantie d'optimalité. L'avantage principal de ces méthodes est qu'elles peuvent s'appliquer à n'importe quelle classe de problèmes, faciles ou très difficiles [1].
Ils existe deux méthodes : les méthodes heuristiques et les méta-heuristiques. Les *heuristiques* sont des méthodes rapides adaptées à résoudre un problème donné. Mais ces méthodes sont incapable d'optimiser un autre problème et sont conçue pour un problème particulier. Par contre la nouvelle génération des méthodes approchées ; les " *méta-heuristiques* " ; sont des méthodes puissantes et générales qui peuvent être appliquées pour résoudre presque n'importe quel problème d'optimisation. Dans la section 1.6, nous présentons cette classe des méthodes d'optimisation (Méta-heuristiques).

### 1.5.3 Les méthodes hybrides

Les approches hybrides forment une autre classe des méthodes de résolution des problèmes d'optimisation combinatoire. On parle d'hybridation des qu'il y a combinaison d'au moins deux approches dans le but de tirer profit de leurs points forts et d'améliorer la qualité des solutions obtenus. Plusieurs types d'hybridations sont possibles, nous citons : l'hybridation de méthodes exactes-exactes, l'hybridation de méthodes heuristiques-exactes et l'hybridation de méthodes heuristiques-heuristiques. Cette dernière représente la partie la plus large de toutes les hybridations. Ces approches hybrides fournissent de bons résultats au prix d'une complexité sans cesse accrue [2].

## 1.6 Les méta-heuristiques

Avant d'expliquer et définir le mot « méta-heuristique » comme méthode de la résolution des problèmes d'optimisation on observe que ce dernier mot se décompose en deux mots grecs : heuristique qui vient de verbe *heurisken* et qui signifie « trouver » et *Meta* qui est un préfixe signifiant « au-delà » ou « plus haut ». Les méta-heuristiques sont apparues dans les années 1980 et forment une famille d'algorithmes d'optimisation visant à résoudre des problèmes d'optimisation difficile. Elles sont généralement utilisées comme des méthodes génériques pouvant optimiser une large gamme de problèmes différents, sans nécessiter de changements profonds dans l'algorithme employé [1].

D'après plusieurs définitions que nous avons vu, on peut définir d'une manière simple une méta-heuristique comme étant une méthode approchée polyvalente de type stochastique applicable à un large genre de problèmes donc n'est pas propre ou liée à un problème précis, elle cherche un objet mathématiques (vecteur,..) tout en maximisant ou minimisant une fonction objectif (fitness) qui décrit la qualité d'une solution au problème [9].

### 1.6.1 Diversification vs intensification

Toutes les méta-heuristiques s'appuient sur un équilibre entre *'l'intensification'* de la recherche et la *'diversification'* de celle-ci. D'un côté, *l'intensification* permet de rechercher des solutions de plus grande qualité en s'appuyant sur les solutions déjà trouvées et de l'autre, la *diversification* met en place des stratégies qui permettent d'explorer un plus grand espace de solutions et d'échapper à des minima locaux [10]. Ces deux points représentent les caractéristiques majeures de n'importe quelle méta-heuristique.

### 1.6.2 Classification des méta-heuristiques

Plusieurs classifications des méta-heuristiques ont été proposées, la plupart distinguent globalement deux catégories :

- Les méthodes à base de solution courante unique : travaillent sur un seul point de l'espace de recherche à un instant donné [6]. Cette sorte de méta-heuristiques lance la recherche avec une solution initiale, et essaye au fur et à mesure d'améliorer sa qualité au cours de la procédure de recherche tout en choisissant une nouvelle solution dans son voisinage, ils sont appelés aussi les méthodes de recherche locale ou méthodes de trajectoire car ils construisent une trajectoire dans l'espace des solutions tout en se redirigeons vers des solutions optimales. Le recuit simulé, la recherche tabou, la recherche à voisinage variable (VNS : *Variable Neighbourhood Search*) sont des instances typiques des méthodes à base de solution unique [9].

– Les méta-heuristiques à base de population : travaillent sur un ensemble de points de l'espace de recherche [6]. Nous présentons cette dernière catégorie dans la section suivant.

## 1.7 Méta-heuristiques à base de population

Contrairement aux algorithmes partant d'une solution singulière, les méthodes de recherche à population, travaillent sur une population de solutions et non pas sur une solution unique. Le principe général de toutes ces méthodes consiste à combiner des solutions entre elles pour en former de nouvelles en essayant d'hériter des "bonnes" caractéristiques des solutions parents. Un tel processus est répété jusqu'à ce qu'un critère d'arrêt soit satisfait (nombre de générations maximum, nombre de générations sans améliorations, temps maximum, etc.) [10].

On distingue dans cette catégorie, les algorithmes évolutionnaires (1.7.1), et les algorithmes d'intelligence en essaim (1.7.2) qui, de la même manière que les algorithmes évolutionnaires, proviennent d'analogies avec des phénomènes biologiques naturels.

### 1.7.1 Les algorithmes évolutionnaires

Les algorithmes *évolutionnaires* (EA : Evolutionary Algorithms), sont une famille d'algorithmes s'inspirant de la théorie de l'évolution « darwinienne » pour résoudre des problèmes divers. Selon la théorie du naturaliste Charles Darwin, énoncée en 1859, l'évolution des espèces est la conséquence de la conjonction de deux phénomènes : d'une part la sélection naturelle qui favorise les individus les plus adaptés à leur milieu à survivre et à se reproduire, laissant une descendance qui transmettra leurs gènes et d'autre part, la présence de variations non dirigées parmi les traits génétiques des espèces (mutations) [11]. La figure 1.3 décrit le squelette d'un algorithme *évolutionnaire* type, commun à la plupart des instances classiques d'EAs.

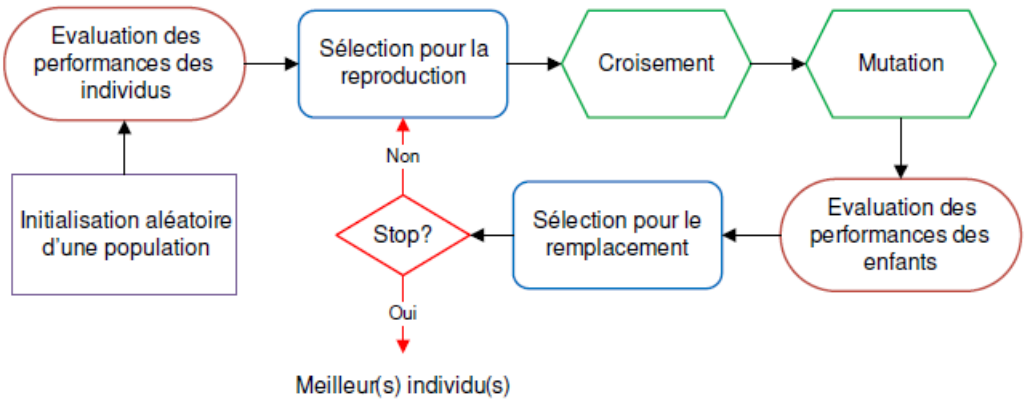



FIGURE 1.3 – Principe d'un algorithme évolutionnaire

#### 1.7.1.1 Les algorithmes génétiques

Les premiers travaux sur les algorithmes génétiques (AG) ont commencé dans les années cinquante lorsque plusieurs biologistes américains ont simulé des structures biologiques sur ordinateur. Puis dans les années 1960, *John Holland* étudie les systèmes évolutifs et en 1975, il a introduit le premier modèle formel des algorithmes génétiques Il expliqua comment ajouter de l'intelligence dans un programme informatique avec les croisements (échangeant le matériel génétique) et la mutation (source de la diversité génétique) [7].

– **Principe de base de l'AG**

Les AGs, sont des algorithmes d'optimisation s'appuyant sur des techniques dérivées de la génétique et des mécanismes d'évolution de la nature : croisement, mutation, sélection. L'analogie entre la théorie de l'évolution consiste à considérer les solutions appartenant à l'espace de recherche du problème à optimiser comme des chromosomes et des individus soumis à l'évolution. Le vocabulaire utilisé est le même que celui de la théorie de l'évolution et de la génétique, on emploie le terme individu (solution potentielle). Chaque individu est un codage ou une *représentation* (binaire, entiers, ...) pour une *solution* candidate d'un problème donné (de même qu'un chromosome est formé d'une chaine de *gènes*). Dans chaque *chromosome*, les gènes sont les variables du problème et leurs valeurs possibles sont appelées *allèles* représenté dans la figure 1.4

La fonction de codage associe à chaque *phénotype* (la solution du problème réel) une représentation de génotype [7]. Le *génotype* est utilisé au cours de l'étape de *reproduction* de l'algorithme alors que le *phénotype* est nécessaire pour *l'évaluation* du coût d'un individu [7].

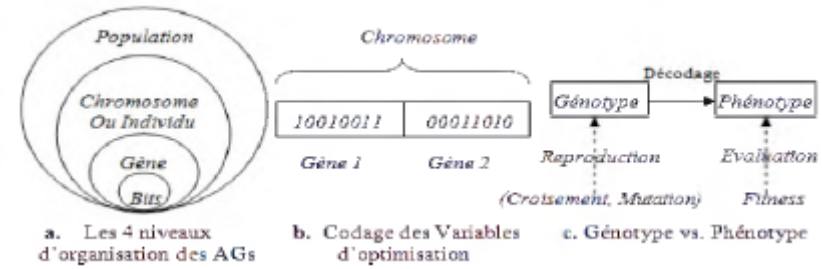



FIGURE 1.4 – Les concepts principaux utilisés dans les algorithmes génétiques

– Le fonctionnement des AGs est montré dans l'algorithme 1.

**Algorithme 1** Algorithme génétique

Initialiser la population initiale ;
Évaluer les individus ;
**Répéter**
  Sélectionner les parents ;
  Croiser les parents et muter les nouveaux individus ;
  Évaluer les nouveaux individus ;
  Conserver les meilleurs individus ;
**Jusqu'à** critère d'arrêt vérifié
**Renvoyer** le meilleur individu de la population

#### 1.7.1.2 Algorithme à évolution différentielle

L'algorithme à ED est parmi les AE les plus simples et les plus performants proposés par *Storn* et *Price* en 1997. Initialement conçue pour les problèmes sans contraintes à variables continues, l'ED a été étendu aux problèmes mixtes et aux problèmes contraints. Sa robustesse et son nombre faible d'hyper-paramètres en font un outil de choix pour la résolution de problèmes difficiles. L'algorithme s'est notamment illustré dans des problèmes d'entraînement de réseaux de neurones, de conception aérodynamique, de décision multicritère, d'approximation polynomiale et d'ordonnancement de tâches.

Contrairement aux AG dont les nouveaux individus proviennent des opérateurs de croisement et de mutation, l'ED n'est pas inspiré de l'évolution naturelle mais génère de nouveaux individus via des opérations géométriques. Basée sur le concept de différence de vecteurs, l'ED combine avec une certaine probabilité les composantes d'individus existants pour former de nouveaux individus [12].

– Le principe des algorithmes à ED est montré dans l'algorithme 2.

**Algorithme 2** Evolution différentielle

**Fonction** EvolutionDifférentielle ($f$ : fonction objectif, $N_P$ : taille de la population, $F$ : facteur d'amplitude, $CR$ : taux de croisement)
  $P \leftarrow$ population initiale aléatoire ;
  **Répéter**
    **/Population temporaire ;
    $P' = \phi$ ;
    **Pour** $x \in P$ **Faire**
      $(u, v, w) \leftarrow$ ChoixParents$(x, P)$ ;
      **/génère un nouvel individu ;
      $y \leftarrow$ Croisement$(x, u, v, w, F, CR)$ ;
      **Si** $f(y) < f(x)$ **Alors**
        **/$y$ remplace $x$ ;
        $P' \leftarrow P' \cup \{y\}$ ;
      **Sinon**
        **/$x$ est conservé ;
        $P' \leftarrow P' \cup \{x\}$ ;
      **FinSi**
    **FinPour**
    **/la population temporaire remplace la population ;
    $P \leftarrow P'$ ;
  **Jusqu'à** critère d'arrêt vérifié
  **Renvoyer** meilleur individu de $P$
**FinFonction**

### 1.7.2 L'intelligence en essaim

L'intelligence en essaim (SI : Swarm Intelligence) est née de la modélisation mathématique et informatique des phénomènes biologiques rencontrés en éthologie [14]. Elle recouvre un ensemble d'algorithmes, à base de population d'agents simples (entités capables d'exécuter certaines opérations), qui interagissent localement les uns avec les autres et avec leur environnement. Ces entités, dont la capacité individuelle est très limitée, peuvent conjointement effectuer de nombreuses tâches complexes nécessaires à leur survie. Bien qu'il n'y ait pas de structure de contrôle centralisée qui dicte la façon dont les agents individuels devraient se comporter, les interactions locales entre les agents conduisent souvent à l'*émergence* d'un comportement collectif global et *auto-organisée*. Parmi les phares algorithmes de l'intelligence en essaim sont les algorithmes d'optimisation par essaim particulaire.

#### 1.7.2.1 Optimisation par essaims particulaires

L'optimisation par essaim particulaire (PSO : Particle Swarm Optimization) a été inventée par *Russel Eberhart* et *James Kennedy* en 1995. Elle s'inspire des déplacements collectifs observés chez certains animaux sociaux tels que les poissons et les oiseaux mi-

grateurs qui ont tendance à imiter les comportements réussis qu'ils observent dans leur entourage, tout en y apportant leurs variations personnelles. Elle trouve ses origines dans les travaux de *Reynolds* et de *Heppner* et *Grenander* qui ont créé des modèles mathématiques permettant de simuler des vols groupés d'oiseaux et de bancs de poissons [11]. L'essaim particulaire correspond à une population d'agents appelés « particules ». Chaque particule, modélisée comme une solution potentielle au problème d'optimisation, parcourt l'espace de recherche, en quête de l'optimum global. Le déplacement d'une particule [11] est influencé par trois composantes (figure 1.5) :

1. *une composante physique* : la particule tend à suivre sa direction courante de déplacement ;
2. *une composante cognitive* : la particule tend à se diriger vers le meilleur site par lequel elle est déjà passée ;
3. *une composante sociale* : la particule tend à se fier à l'expérience de ses congénères et, ainsi, à se diriger vers le meilleur site déjà atteint par ses voisins.

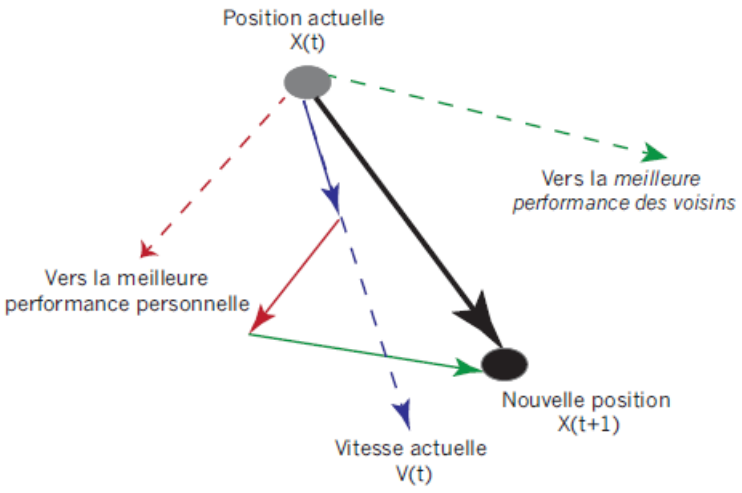



FIGURE 1.5 – Déplacement d'une particule (PSO)

– **Formules de mise à jour**

La vitesse d'une particule $V^{(k+1)}$ à la génération $k+1$ est fonction de sa position courante $X^{(k)}$, sa vitesse courante $V^{(k)}$, sa meilleure position connue $X_l$ et la meilleure position connue $X_g$ de l'essaim :

$$V^{(k+1)} = \omega V^{(k)} + \alpha(X_l - X^{(k)}) + \beta(X_g - X^{(k)}) \tag{1.3}$$

Où $\omega$ est un hyperparamètre (facteur d'inertie) décidé par l'utilisateur, et $\alpha$ et $\beta$ sont des vecteurs tirés dans $[0, 1]^{dim}$ avec une probabilité uniforme. La position de

la particule est ensuite simplement mise à jour :

$$X^{(k+1)} = X^{(k)} + V^{(k+1)} \tag{1.4}$$

– Le principe de base est présenté dans l'algorithme 3.

**Algorithme 3** Optimisation par essaims particulaires

Initialiser l'essaim (positions et vitesses des particules) ;
Évaluer les particules ;
**Répéter**
  **Pour** chaque particule de l'essaim **Faire**
    Mettre à jour la vitesse de la particule ;
    Mettre à jour la position de la particule ;
    Mettre à jour la meilleure position connue de la particule ;
    Mettre à jour la meilleure position connue de l'essaim ;
  **FinPour**
**Jusqu'à** critère d'arrêt vérifié
**Renvoyer** meilleure position connue de l'essaim

# Conclusion

Dans ce chapitre, nous avons présenté d'une façon introductive, quelques notions sur l'optimisation combinatoire et ses problèmes classiques, ainsi que les méta-heuristiques les plus répondues. Ces méthodes peuvent être partagées en deux sous classes : des méthodes à base d'une solution unique et des méthodes à base de population de solutions. Tant que cette classe des méthodes est basées sur des populations, alors sa taille grandit en fonction de la dimension du problème à résoudre. Cela constitue un point faible dans leur implémentation hardware. Les chercheurs proposent alors les méta-heuristiques compactes comme une solution pour résoudre des problèmes qui nécessitent des ressources matériels limitées.

Dans le chapitre suivant, nous allons présenter les algorithmes compacts en détaille.

# Chapitre 2

# Les méta-heuristiques compactes

## Sommaire

# Introduction

Comme les méthodes exactes consomment beaucoup de mémoire et de temps pour obtenir un optimum, et les méthodes heuristiques sont conçue pour un problème particulier, les méta-heuristiques ont été utilisées avec succès pour résoudre des problèmes d'optimisation dans différents domaines. Cependant certaines applications modernes nécessitent encore des processus d'optimisation complexes dans des conditions matérielles très limitées. Dans ces cas, les algorithmes compacts constituent une alternative efficace pour résoudre ces problèmes d'optimisation, car ils se comportent comme des méta-heuristiques sans traiter en réalité une population réelle de solutions candidates.

Dans ce chapitre, nous avons introduire quelques algorithmes des méta-heuristiques compactes. Nous traiterons le cas des solutions qui ont une représentation binaire ainsi que le cas des solutions qui ont une représentation réelles. Nous présenterons le cGA, rcGA, cDE, le cPSO et le DEcDE avec leur pseudo codes.

## 2.1 Les méta-heuristiques compactes

Les algorithmes compacts sont des versions 'light' des méta-heuristiques. Étant donné que ces techniques utilisent un nombre réduit d'individus sur la population, le besoins en mémoire des dispositifs de calcul est réduis. L'une des stratégies utilisées pour réduire les besoins en mémoire consiste à représenter la population sous la forme d'une distribution de probabilité sur l'ensemble des solutions possibles. Contrairement aux méta-heuristiques basées sur la population, les méta-heuristiques compactes reposent sur l'augmentation de la distribution de probabilité pour chaque position de bit des solutions possibles.[13]

### 2.1.1 Différents travaux des méta-heuristiques compactes

L'optimisation compacte a commencé en 1999 quand *Harik et al* ont proposé le premier algorithme génétique compact (cGA) qui utilise une taille de population réduite par rapport à l'algorithme génétique de base, cette population est représenté par un vecteur de probabilité de nombre binaire 0 ou 1 [14].

En 2008, *Ernesto Mininno et al* ont proposé le premier algorithme compact qui fonctionne avec des valeurs réelles. Leur variante est appelée algorithme génétique compact à valeurs réelle (rcGA). Il donne de meilleurs résultats que le cGA binaire [15].

Depuis l'apparition du rcGA, plusieurs auteurs ont proposé la version compacte d'autres algorithmes d'optimisation avec représentation réelle. Parmi ces algorithmes nous mentionnons à titre d'exemple :

- En 2011, l'évolution différentielle compacte (cDE) a été proposée dans [16].
- En 2012, la version compacte du PSO (cPSO) a également été proposée dans [17].
- En 2013, *Yang et al* ont proposé un algorithme compact d'une méta-heuristique basé sur l'enseignement et l'apprentissage des élèves appelé cTLBO (*Compact Teaching-Learning-Based Optimization*) [18].
- En 2014, *A.S. Soares et al* ont proposé un algorithme génétique compact basé sur l'opérateur de mutation [19].
- En 2016, *B. V. Ha et al* [20] ont proposé une version modifié de l'algorithme génétique compact binaire qui fonctionne avec plus d'un vecteur de probabilité.

### 2.1.2 Population virtuelle dans les algorithmes compacts

Les algorithmes compacts sont caractérisé par l'absence de la population. En d'autre terme, ils traitent une population réel de solutions à travers une population virtuelle, cette dernière est représenté dans un vecteur appelé vecteur de probabilité $PV$ (probabilistic vector). On a deux sortes de $PV$ :

- Un $PV$ de valeurs binaires comme dans les algorithmes compacts binaires tel que cGA.
- Un $PV$ qui est une matrice de dimensions $dim \times 2$ :

$$PV^t = [\mu^t, \sigma^t] \tag{2.1}$$

Où les valeurs $\mu$ sont égales à 0 alors que les valeurs $\sigma$ sont égales à un grand nombre (une constante $\lambda = 10$). Ce $PV$ est utilisé dans plusieurs algorithmes compacts tel que rcGA, cDE, cPSO et bien d'autres.

## 2.2 L'algorithme génétique compact binaire

La population est représentée par un vecteur de probabilité $PV$ binaire. Ce dernier contient la probabilité de chaque bit d'être égal à 0 ou à 1. Par conséquent, on dit que la population est compactée et représentée par le $PV$. À chaque génération, deux individus sont échantillonnés à l'aide de $PV$. La compétition entre ces deux individus déterminera la manière dont le $PV$ sera corrigé.

- Si le bit $(i)$ du gagnant (*winner*) égale à 1 et que celui du perdant (*loser*) est 0, la composante $(i)$ de $PV$ sera augmentée de $1/Np$, où $Np$ est la taille hypothétique (*virtuelle*) de la population simulée. Inversement, il sera diminué avec $1/Np$.
- Lorsque le $(i)$ ème bit du gagnant est égal au $(i)$ ème bit du perdant, le $(i)$ ème composante du $PV$ ne sera pas corrigée.

Le fonctionnement de cGA est montré dans l'algorithme 4

**Algorithme 4** L'algorithme génétique compact binaire (cGA)

Compteur $t = 0$ ;
**/Initialisation du $PV$ ;
**Pour** $i = 1 : dim$ **Faire**
  $PV[i] = 0.5$ ;
**FinPour**
**Tantque** le critère d'arrêt n'est pas atteint **Faire**
  Générer 2 individus $(a, b)$ à partir de $PV$ ;
  $[winner, loser] = ComparerFitness(a, b)$ ;
  **/Mise à jour du $PV$ ;
  **Pour** $i = 1 : dim$ **Faire**
    **Si** $winner[i] != loser[i]$ **Alors**
      **Si** $winner[i] == 1$ **Alors**
        $PV[i] = PV[i] + \frac{1}{N_p}$ ;
      **Sinon**
        $PV[i] = PV[i] - \frac{1}{N_p}$ ;
      **FinSi**
    **FinSi**
  **FinPour**
  $t = t + 1$ ;
**FinTantque**

## Exemple illustratif

– Au début, l'algorithme initialise le vecteur de probabilité $PV$, on suppose que la dimension de ce vecteur est égale à 3 donc :

$$PV = [0.5\ , 0.5\ , 0.5]$$

– Ensuite l'algorithme génère aléatoirement deux individus $G1$ et $G2$, avec la même dimension du $PV$ ($dim = 3$).

$$G1 = [1\ , 0\ , 1]$$

$$G2 = [0\ , 1\ , 1]$$

Nous supposons ici que $G1$ et le gagnant et $G2$ et le perdant. Donc, l'adaptation du vecteur de probabilité $PV$ est effectué comme suit :
On a : $G1[1] = 1$ et $G2[1] = 0$. L'algorithme calcule la nouvelle valeur du $PV[1]$ et rajoute à l'ancienne valeur du $PV$, une quantité $(1/Np)$, où $Np$ est la taille virtuelle de la population.

$$PV[1] = [0.5] + (1/Np)$$

– Passent maintenant au deuxième chiffre des deux vecteurs $G1[2] = 0$ et $G2[2] = 1$.

La nouvelle valeur du $PV$ est modifiée de façon qu'on diminue de sa valeur actuelle une quantité = $(1/Np)$.

$$PV[1] = [0.5] - (1/Np)$$

– Concernant le 3 ème bit, aucune modification du $PV$ ne sera effectuée car ils ont le même bit.

## 2.3 Élitisme dans les algorithmes compacts

Les nouvelles variantes basées sur l'algorithme génétique compacte de base (cGA) sont appelées "*cGA élitiste persistant* " (pe_cGA) et " *cGA élitiste non persistante* " (ne_cGA).
La principale particularité de l'algorithme pe_cGA est que le "gagnant" du tournoi est toujours gardé en mémoire, et remplacé uniquement si le cGA génère un nouvel individu avec une meilleure fonction objectif. Ceci peut être obtenu pratiquement en modifiant le pseudo-code de cGA standard comme décrit dans l'algorithme 5.

**Algorithme 5** pseudo-code pe_cGA

Compteur $t = 0$ ;
**/Initialisation du $PV$ ;
**Pour** $i = 1 : dim$ **Faire**
  $PV[i] = 0.5$ ;
**FinPour**
Générer **elite** à partir de $PV$ ;
**Tantque** le critère d'arrêt n'est pas atteint **Faire**
  Générer 1 individus $(a)$ à partir de $PV$ ;
  **/Sélection du l'elite ;
  $[winner, loser] = ComparerFitness(a, elite)$ ;
  **Si** $a == winner$ **Alors**
    $elite = a$ ;
  **FinSi**
  **/Mise à jour du $PV$ ;
  **Pour** $i = 1 : dim$ **Faire**
    **Si** $winner[i] != loser[i]$ **Alors**
      **Si** $winner[i] == 1$ **Alors**
        $PV[i] = PV[i] + \frac{1}{N_p}$ ;
      **Sinon**
        $PV[i] = PV[i] - \frac{1}{N_p}$ ;
      **FinSi**
    **FinSi**
  **FinPour**
  $t = t + 1$ ;
**FinTantque**

On sait qu'un fort élitisme peut conduire à une convergence prématurée et à des solutions de mauvaises qualités. Pour cette raison, le ne_cGA introduit un paramètre de contrôle $\eta$ limitant la « persistance » de la solution élitiste.

Comme dans le pe_cGA, le perdant est toujours remplacé par un nouvel individu généré à partir du $PV$. Mais le gagnant de la compétition entre la solution élite et la solution nouvellement générée n'est passé à la génération suivante que si la « longueur de l'héritage » $\theta$ ne dépasse pas le seuil prédéfini $\eta$. Le principe est décrit dans l'algorithme 6.

**Algorithme 6** pseudo-code ne_cGA

Compteur $t = 0$ et $\theta = 0$ ;
**/Initialisation du $PV$ ;
**Pour** $i = 1 : dim$ **Faire**
  $PV[i] = 0.5$ ;
**FinPour**
Générer **elite** à partir de $PV$ ;
**Tantque** le critère d'arrêt n'est pas atteint **Faire**
  Générer 1 individus $(a)$ à partir de $PV$ ;
  **/Sélection de l'elite ;
  $[winner, loser] = ComparerFitness(a, elite)$ ;
  $\theta = \theta + 1$ ;
  **Si** $a == winner$ **or** $\theta \geq \eta$ **Alors**
    $elite = a$ ;
    $\theta = 0$ ;
  **FinSi**
  **/Mise à jour du $PV$ ;
  **Pour** $i = 1 : dim$ **Faire**
    **Si** $winner[i] != loser[i]$ **Alors**
      **Si** $winner[i] == 1$ **Alors**
        $PV[i] = PV[i] + \frac{1}{N_p}$ ;
      **Sinon**
        $PV[i] = PV[i] - \frac{1}{N_p}$ ;
      **FinSi**
    **FinSi**
  **FinPour**
  $t = t + 1$ ;
**FinTantque**

## 2.4 L'algorithme génétique compacte avec représentation réelle

Est un algorithme génétique compact qui fonctionne avec des valeurs réelles. Il donne de meilleurs résultats que cGA binaire tout en réduisant significativement le coût de calcul

en évitant même les opérations supplémentaires liées aux conversions du binaire vers le réel. Le rcGA utilise une fonction de densité de probabilité gaussienne (PDF : *Probability of Density Function*) pour représenter les solutions. Le $PV$ de rcGA est une matrice qui contient la moyenne et l'écart-type voir la (formule 2.1).

Tel que $\mu$ est un vecteur des valeurs moyennes et $\sigma$ est la déviation standard d'une fonction gaussien de distribution de probabilité dans l'intervalle $[-1, 1]$. $\mu[i]$ est mis a 0 et $\sigma[i]$ est égale à $\lambda$ qui est une grande constante positive $= 10$ pour l'initialisation du $PV$.

Il reste maintenant à trouver un moyen d'ajuster le $PV$ en fonction des itérations. Pour ce faire, deux individus sont générés. Puis, un tournoi est effectué entre eux. Par conséquent, par rapport au critère d'optimisation, un gagnant et un perdant sont obtenus. Selon leurs allèles respectifs, une règle de mise à jour, qui est obtenue en supposant que le perdant a été remplacé par le gagnant dans la population virtuelle représentée par $PV$, est ensuite effectuée en utilisant les formules suivantes :

$$\mu^{t+1}[i] = \mu^{t}[i] + \frac{1}{N}(winner[i] - loser[i]) \tag{2.2}$$

$$\sigma^{t+1}[i] = \sqrt{(\sigma^{t}[i])^2 + (\mu^{t}[i])^2 - (\mu^{t+1}[i])^2 + \frac{1}{N_p}(winner^2[i] - loser^2[i])} \tag{2.3}$$

l'algorithme 7 illustre le processus génétique d'un algorithme compacte avec représentation réelle

## 2.5 L'algorithme de l'évolution différentielle compacte (cDE)

L'algorithme de l' Évolution Différentiel compact(cDE), défini dans [16], modifie le rcGA en échantillonnant à partir du PDF, pas seulement une solution $x$, mais plusieurs solutions en fonction des règles de mutation de l'évolution différentielle (DE). Il utilise une matrice $PV$ comme indiqué pour le rcGA : les valeurs $\mu$ sont égales à 0, tandis que les valeurs $\sigma$ sont égales à un grand nombre $\lambda = 10$.

L'obtention des individus dans le cDE peut aller jusqu'à 3 individus et un Elite. Pour la phase de mutation, on utilise les même schémas existants dans le standard $DE/rand/bin$. Pour chaque itération, 3 nouvelles solutions à savoir $x_r, x_s$ et $x_t$, sont échantillonnées à partir de $PV$. Par la suite, la mutation différentielle pour obtenir un individu mutant $x_{off}$ est effectuée comme indiquée dans la formule (2.4).

$$x_{off} = x_t + F(x_r - x_s) \tag{2.4}$$

**Algorithme 7** Algorithme génétique compacte réel (pe_rcGA)

Compteur $t = 0$ ;
**/Initialisation du $PV$ ;
**Pour** $i = 1 : dim$ **Faire**
  Initialization $\mu[i] = 0$ ;
  Initialization $\sigma[i] = 10$ ;
**FinPour**
Générer **elite** à partir du $PV$ ;
**Tantque** le critère d'arrêt n'est pas atteint **Faire**
  Générer 1 individus $x$ a partir de $PV$ ;
  **/Elite selection ;
  $[winner, loser] = ComparerFitness(x, elite)$ ;
  **Si** $x == winner$ **Alors**
    $elite = x$ ;
  **FinSi**
  **/Mise à jour du $PV$ ;
  **Pour** $i = 1 : dim$ **Faire**
    $\mu^{t+1}[i] = \mu^t[i] + \frac{1}{N_p}(winner[i] - loser[i])$ ;
    $\sigma^{t+1}[i] = \sqrt{(\sigma^t[i])^2 + (\mu^t[i])^2 - (\mu^{t+1}[i])^2 + \frac{1}{N_p}(winner^2[i] - loser^2[i])}$ ;
  **FinPour**
  $t = t + 1$ ;
**FinTantque**

où $F \in [0, 2]$ est un facteur d'échelle qui contrôle la longueur du vecteur d'exploration $(x_r - x_s)$.

### 2.5.1 Le croisement binomial

Est un croisement différentiel s'effectue entre l'élite et le $x_{off}$ avec une probabilité de croisement $C_r$ pour engendrer un $x_{off}$ final, qui sera comparer avec l'élite par rapport à leur fitness ; le gagnant sera l'élite.

- Le principe de cDE est montré dans l'algorithme 8.

**Algorithme 8** pseudo-code pe-cDE/rand/1/bin

Compteur $t = 0$ ;
**/Initialisation du $PV$ ;
**Pour** $i = 1 : dim$ **Faire**
  Initialization $\mu[i] = 0$ ;
  Initialization $\sigma[i] = 10$ ;
**FinPour**
Générer **elite** à partir du $PV$ ;
**Tantque** le critère d'arrêt n'est pas atteint **Faire**
  Générer 1 individus $x$ a partir de $PV$ ;
  **/Mutation ;
  Générer 3 individus $x_r, x_s$, et $x_t$ a partir de $PV$ ;
  $x'_{off} = x_t + F(x_r - x_s)$ ;
  **/Croisement binomial ;
  **Pour** $i = 1 : dim$ **Faire**
    Générer $rand(0, 1)$ ;
    **Si** $rand(0, 1) > C_r$ **Alors**
      $x_{off}[i] = elite[i]$ ;
    **FinSi**
  **FinPour**
  **/Sélection Elite ;
  $[winner, loser] = ComparerFitness(x_{off}, elite)$ ;
  **Si** $x_{off} == winner$ **Alors**
    $elite = x_{off}$ ;
  **FinSi**
  **/Mise à jour du $PV$ ;
  **Pour** $i = 1 : dim$ **Faire**
    $\mu^{t+1}[i] = \mu^t[i] + \frac{1}{N_p}(winner[i] - loser[i])$ ;
    $\sigma^{t+1}[i] = \sqrt{(\sigma^t[i])^2 + (\mu^t[i])^2 - (\mu^{t+1}[i])^2 + \frac{1}{N_p}(winner^2[i] - loser^2[i])}$ ;
  **FinPour**
  $t = t + 1$ ;
**FinTantque**

### 2.5.2 Le croisement exponentiel

Le croisement exponentiel est une autre option populaire pour les schémas DE. Dans ce cas, une variable de conception du fils provisoire $x_{off}[i]$ est sélectionné de manière aléatoire puis copié dans la variable de conception de la solution élite (sa copie). Cela garantit que l'élite et le fils ont des génotypes différents. Par la suite, un ensemble de nombres aléatoires compris entre 0 et 1 est généré.

Tant que $rand(0, 1) \leq C_r$, où le taux de croisement $C_r$ est un paramètre prédéterminé, les variables de conception du fils provisoire(mutant) sont copiés dans les positions correspondantes de l'élite. Dés que $rand(0, 1) > C_r$, le processus de copie est interrompu. Ainsi, toutes les variables de conception restantes de la progéniture sont copiées à partir

du parent. Pour plus de clarté, l'algorithme 9 illustre le principe de fonctionnement.

**Algorithme 9** pseudo-code croisement exponentiel

$x_{off} =$ **elite** ;
Générer $i = round(dim.rand(0,1))$ ;
$x_{off}[i] = x'_{off}[i]$ ;
**Tantque** $rand(0,1) \leq C_r$ **Faire**
  $x_{off}[i] = x'_{off}[i]$ ;
  $i = i + 1$ ;
  **Si** $i == dim$ ; **Alors**
    $i = 1$ ;
  **FinSi**
  Générer $rand(0,1)$ ;
**FinTantque**

## 2.6 L'algorithme de l'évolution différentielle compacte avec croisement exponentiel et perturbation du vecteur probabiliste

L'algorithme DEcDE est composé d'un cadre $pe\_cDE/rand/1/exp$ qui utilise la mutation trigonométrique proposée dans [21]. Avec une probabilité prédéfinie $M_t$, au lieu d'appliquer la mutation $DE/rand/1$ et le croisement exponentiel, l'individu résultant est généré par des moyens de la procédure suivante :

$$x_{off} = \frac{(x_r + x_s + x_t)}{3} + (p_s - p_r)(x_r - x_s) + (p_t - p_s)(x_s - x_t) + (p_r - p_t)(x_t - x_r) \quad (2.5)$$

Où pour $k = r, s, t$,

$$p_k = \frac{|f(x_k)|}{|f(x_r)| + |f(x_s)| + |f(x_t)|} \quad (2.6)$$

– **La mutation trigonométrique**
  Est un opérateur glouton qui pour trois points donnés, génère un fils en exploitant les directions de recherche les plus prometteuses. L'emploi de cet opérateur au sein de DEcDE est censé d'offrir une alternative exploitante à la règle d'exploration standard de DE. La mutation trigonométrique a donc pour rôle de favoriser la génération des individus fils (offspring) selon des directions (localement) optimales. En ce sens, cette mutation spéciale peut être vue comme une recherche locale. Cette dernière est contrebalancée par une recherche globale effectuée indirectement en perturbant le $PV$.

Le DEcDE proposé introduit des règles supplémentaires pour la modification des valeurs du $PV$. Plus précisément, avec une probabilité $M_p$, le $PV$ est perturbé. Chaque

composante $\mu[i]$ du vecteur de la moyenne $\mu$ est perturbée selon la formule 2.7.

$$\mu^{t+1}[i] = \mu^{t+1}[i] + 2\tau.rand(0,1) - \tau \tag{2.7}$$

Où $\tau$ est un poids représentant l'amplitude maximale de la perturbation. La règle de perturbation pour le $\sigma$ est donnée par :

$$(\sigma^{t+1}[i])^2 = (\sigma^{t+1}[i])^2 + \tau.rand(0,1) \tag{2.8}$$

Le principe de fonctionnement de DEcDE est donné dans l'algorithme 10.

## 2.7 L'optimisation par Essaim de particules compacte (cPSO)

Le PSO compacte [17] a été proposé par *Neri et al* en 2013. Il a des caractéristiques d'algorithmes PSO (Particle Swarm Optimization) standard, mais utilise un modèle probabiliste pour représenter l'ensemble de solutions 'Swarms'.

Une fois la nouvelle solution (particule) est générée, le mouvement de la particule dans cPSO est identique à celui de l'algorithme PSO standard, et la position $x$ et la vitesse $v$ sont mises à jour par la formule (2.9) et la formule (2.10) :

$$v^{t+1} = \phi_1 v^t + \phi_2(x_{lb}^t - x^t) + \phi_3(x_{gb}^t - x^t) \tag{2.9}$$

$$x^{t+1} = x^t + v^{t+1} \tag{2.10}$$

Le principe général de cPSO existe dans l'algorithme 11.

**Algorithme 10** pseudo-code DEcDE

Compteur $t = 0$ ;
**/Initialisation du $PV$ ;
**Pour** $i = 1 : dim$ **Faire**
  Initialization $\mu[i] = 0$ ;
  Initialization $\sigma[i] = 10$ ;
**FinPour**
Générer **elite** à partir du $PV$ ;
**Tantque** le critère d'arrêt n'est pas atteint **Faire**
  **Si** $rand(0,1) < M_t$ ; **Alors**
    **/Mutation trigonométrique ;
    Générer 3 individus $x_r, x_s$, et $x_t$ a partir de $PV$ ;
    $x_{off} = \frac{(x_r+x_s+x_t)}{3} + (p_s - p_r)(x_r - x_s) + (p_t - p_s)(x_s - x_t) + (p_r - p_t)(x_t - x_r)$ ;
  **Sinon**
    **/Mutation ;
    Générer 3 individus $x_r, x_s$, et $x_t$ a partir de $PV$ ;
    $x'_{off} = x_t + F(x_r - x_s)$ ;
    **/Croisement ;
    Appliquer le croisement exponentiel montrer dans l'algorithme 9 et générer $x_{off}$ ;
  **FinSi**
  **/Sélection Elite ;
  $[winner, loser] = ComparerFitness(x_{off}, elite)$ ;
  **Si** $x_{off} == winner$ **Alors**
    $elite = x_{off}$ ;
  **FinSi**
  **/Mise à jour du $PV$ ;
  **Pour** $i = 1 : dim$ **Faire**
    $\mu^{t+1}[i] = \mu^t[i] + \frac{1}{N_p}(winner[i] - loser[i])$ ;
    $\sigma^{t+1}[i] = \sqrt{(\sigma^t[i])^2 + (\mu^t[i])^2 - (\mu^{t+1}[i])^2 + \frac{1}{N_p}(winner^2[i] - loser^2[i])}$ ;
  **FinPour**
  **/Perturbation du $PV$ ;
  **Si** $rand(0,1) < M_p$ **Alors**
    **Pour** $i = 1 : dim$ **Faire**
      $\mu^{t+1}[i] = \mu^{t+1}[i] + 2\tau.rand(0,1) - \tau$ ;
      $\sigma^{t+1}[i] = \sqrt{(\sigma^{t+1}[i])^2 + \tau.rand(0,1)}$ ;
    **FinPour**
  **FinSi**
  $t = t + 1$ ;
**FinTantque**

## 2.8 Domaines d'applications

Comme nous l'avons mentionné précédemment dans notre travail, les méta-heuristiques ont été utilisés avec succès pour résoudre des problèmes d'optimisations dans différents domaines. Les méta-heuristiques compactes ont joué également le même rôle dans les applications qui ont des dispositifs de calculs limités. Parmi ces domaines, nous mentionnons

**Algorithme 11** pseudo-code cPSO

compteur $t = 0$ ;
**/initialisation du $PV$ ;
**Pour** $i = 1 : dim$ **Faire**
Initialization $\mu[i] = 0$ ;
Initialization $\sigma[i] = 10$ ;
**FinPour**
**/Initialisation du Global best ;
Générer $x_{gb}$ a partir de $PV$ ;
générer de manière aléatoire $x^t$ et $v^t$ ;
**Tantque** le critère d'arrêt n'est pas atteint **Faire**
**/Nouvelle initialisation du local best ;
$x_{lb}^t =$ générer de$(PV_{lb})$ ;
**/Mettre à jour la position et la vélocité ;
$v^{t+1} = \phi_1 v^t + \phi_2(x_{lb}^t - x^t) + \phi_3(x_{gb}^t - x^t)$ ;
$x^{t+1} = \gamma_1 x^t + \gamma_2 v^{t+1}$ ;
**/Sélection du local best ;
$[winner, loser] = ComparerFitness(x^{t+1}, x_{lb})$ ;
**/Mise à jour du $PV$ ;
**Pour** $i = 1 : dim$ **Faire**
$\mu^{t+1}[i] = \mu^t[i] + \frac{1}{N_p}(winner[i] - loser[i])$ ;
$\sigma^{t+1}[i] = \sqrt{(\sigma^t[i])^2 + (\mu^t[i])^2 - (\mu^{t+1}[i])^2 + \frac{1}{N_p}(winner^2[i] - loser^2[i])}$ ;
**FinPour**
Mise à jour du Global best ;
$[winner, loser] = ComparerFitness(x^{t+1}, x_{gb})$ ;
$x_{gb} = winner$ ;
$t = t + 1$ ;
**FinTantque**

les suivants :

- Amélioration du vol stationnaire dans un micro-véhicule aérien à ailes battantes à l'échelle de l'insecte en utilisant un lgorithme génétique compacte. *"A Hardware Compact Genetic Algorithm for Hover Improvement in an Insect-Scale Flapping-Wing Micro Air Vehicle"* [22].
- Le problème des services de covoiturage en utilisant un algorithme génétique compacte. *"Services-Oriented Computing Using the Compact Genetic Algorithm for Solving the Carpool Services Problem"* [23].
- Le contrôle de robot cartésien en utilisant l'algorithme d'évolution différentielle compacte mémétique . *"Memetic Compact Differential Evolution for Cartesian Robot Control"* [24].
- Distribution intégrée e chargement planifié via un algorithme compact *"Integrated distribution and loading planning via a compact metaheuristic algorithm"* [25].
- Le contrôle de schéma de topologie dans les réseaux de capteurs sans fil en utili-

sant un algorithme d'optimisation compacte par colonies d'abeilles. *"A Compact Artificial Bee Colony Optimization for Topology Control Scheme in Wireless Sensor Networks"* [26].

# Conclusion

Dans ce chapitre, nous avons identifié certains algorithmes de méta-heuristique compacte. L'une des raisons qui a attiré notre attention sur l'optimisation compacte est que non seulement ses algorithmes donnent fréquemment des résultats satisfaisants. Mais aussi il ne consomment pas les coûts en terme de mémoire et de calcul. Aussi, ils sont très facile à mettre en œuvre. Cette curiosité nous a fait entrer via un portail " méta-heuristique compact " car ils se comportent comme des " méta-heuristiques standards ".

Dans le chapitre suivant, nous essayerons de garder notre empreinte dans ce domaine en proposant de nouveaux algorithmes compacts efficaces.

# Chapitre 3

# Contribution

## Sommaire

# Introduction

Nous avons décrit dans le chapitre précèdent différentes méta-heuristiques compactes. Elles ont été utilisées dans plusieurs domaines d'applications (avec mémoire limité) vu les résultats satisfaisants qu'elles présentent.

Dans ce chapitre, nous essayerons de proposer des nouvelles variantes des méta-heuristiques compactes en utilisant non seulement une hybridation entre celle existantes (cPSO, cDE, DEcDE, ...etc).
mais aussi une version compacte d'une nouvelle méta-heuristique qui n'a jamais été compactée. Nous commençons d'abord par présenter la motivation qui nous a poussé à choisir l'étude des méta-heuristiques compactes. Ensuite, nous entamons en détail notre contribution proposée.

## 3.1 Problématique et motivation

Les méta-heuristiques basées populations sont des approches efficaces pour élaborer des algorithmes de recherche. Elles nécessitent un espace mémoire et une capacité de calcul élevée pour résoudre un problème donnée. Pour atteindre une solution optimale, un grand nombre d'individus traités est requis dans chaque génération. Cependant, dans certaines situations, les dispositifs de calcul disponibles présentent certaines contraintes de coût et de l'espace mémoire. Selon [17], cette situation est typique en ingénierie de contrôle et en robotique (principalement pour la robotique commerciale). Par exemple, des robots à usage domestique (Ex : un robot aspirateur) doivent suivre un processus d'apprentissage. Ce problème d'optimisation complexe doit être résolu rapidement et sans compter sur un ordinateur à pleine capacité, en raison de contraintes de taille et de coût.

Les algorithmes compacts ont été conçus afin de traiter cette catégorie de problèmes. Ils offrent une alternative et assurent une réduction des capacités de calcul requises. Ils se caractérisent par leur simplicité et leur efficacité et donnent des résultats satisfaisantes dans pas mal de problèmes. Ces avantages nous ont motivé à adapter des propositions nous les mentionnerons en détails dans la section (3.2).

## 3.2 Algorithme d'optimisation par vagues d'eau

L'optimisation par vagues d'eau " *water waves Optimization*" (WWO) qui a été développée par *Zheng* [27], est une nouvelle méthode d'optimisation inspirée de la nature, où l'espace de résolution du problème est analogue à celui des fonds marins. Chaque solution de la population étant analogue à " une onde ou vague d'eau" de hauteur $h$ et une longueur

$\lambda$. Supposons qu'il existe un problème de maximisation $F$ avec sa fonction objectif $f$ , le problème $F$ peut être comparé au modèle de vagues d'eaux peu profondes. La relation correspondante est représentée dans la figure (3.1).

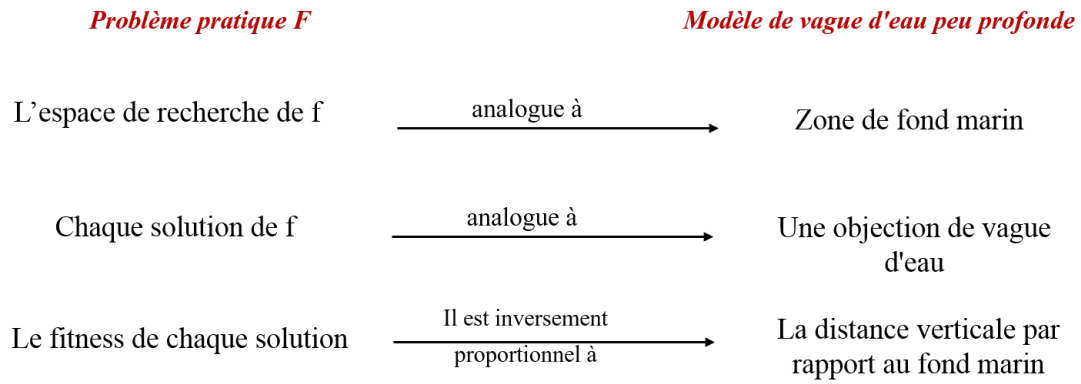



FIGURE 3.1 – Relation entre un problème $F$ et le modèle de vagues d'eaux peu profondes

Lors de l'initialisation de la population, pour chaque vague, sa hauteur $h$ est initialisée à une constante $h_{max}$ et sa longueur d'onde $\lambda$ à $0,5$. A chaque génération, WWO utilise trois types d'opérations : propagation, rupture et réfraction.

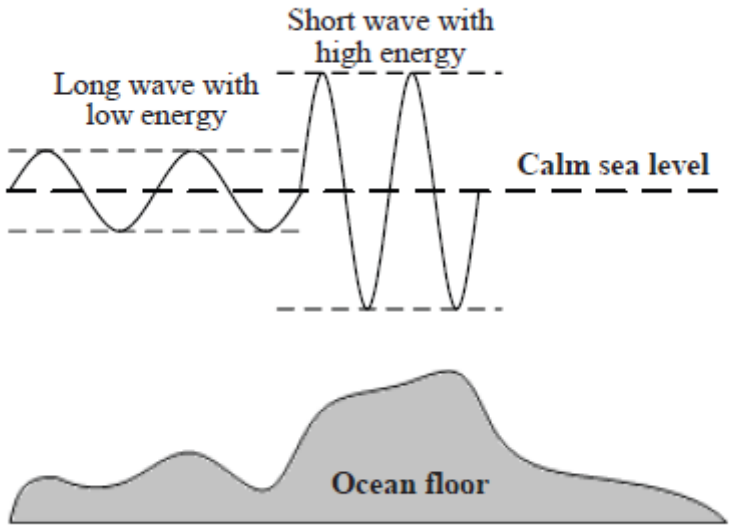



FIGURE 3.2 – Différentes formes de vagues dans les eaux profondes et peu profondes

### 3.2.1 La propagation

Pour chaque onde $x$, l'opérateur de propagation crée une nouvelle onde $x'$ en calculant un déplacement différent à chaque dimension $d$ et l'ajoute à l'onde d'origine :

$$x'_d = x_d + rand(-1, 1).\lambda L_d \tag{3.1}$$

Où *rand* est une fonction générant un nombre aléatoire uniformément réparti dans la plage spécifiée. $L_d$ est la longueur de la $d$ième dimension de l'espace de recherche.
Ensuite, nous évaluons la fonction objectif de $x'$ :

Si $f(x')$ est supérieur à $f(x)$, $x'$ remplace $x$ dans la population et sa hauteur est réinitialisée à $h_{max}$ ;

Sinon, $x$ est restera et sa hauteur est diminuée d'un. Après chaque génération, WWO calcule la longueur de chaque onde $x$ comme suit :

$$\lambda = \lambda.\alpha^{-(f(x)-f_{min}+\epsilon)/(f_{max}-f_{min}+\epsilon)} \tag{3.2}$$

Où $\alpha$ est un paramètre de contrôle appelé coefficient de réduction de la longueur d'onde, $f_{min}$ et $f_{max}$ représentent les valeurs maximales et minimales de la fonction objective de la population actuelle. Et $\epsilon$ est une petite constante positive permettant d'éviter la division par zéro.

### 3.2.2 La rupture

L'opérateur de rupture brise une vague dans un train d'ondes solitaires lorsque sa vitesse de crête dépasse la célérité de la vague. WWO effectue uniquement la coupure sur une vague $x$ qui atteint un emplacement meilleur que la meilleure solution actuelle $x_{best}$. Chaque onde solitaire $x'$ de $x$ est obtenue en sélectionnant de manière aléatoire $k$ dimensions (où $k$ est un nombre aléatoire entre 1 et un nombre prédéfini $k_{max}$) et en ajoutant un décalage différent à la position d'origine à chaque dimension $d$ comme suit :

$$x'_d = x_d + Gaussien(0,1).\beta L_d \tag{3.3}$$

Où $\beta$ est le coefficient de rupture, et $gaussien(0,1)$ est une fonction générant un nombre aléatoire gaussien avec une moyenne de 0 et un écart type 1. Si la fonction objectif de $x'$ est mieux que $x, x'$ prend la place de $x$ dans la population.

### 3.2.3 La réfraction

L'opérateur de réfraction crée une onde $x$ dont la hauteur diminue à zéro et apprend de $x_{best}$. Par réfraction, chaque dimension de la nouvelle position de $x'$ est calculée comme un nombre aléatoire centré à mi-chemin entre la position d'origine et $x_{best,d}$ :

$$x'_d = Gaussien(\frac{x_{best,d} + x_d}{2}, \frac{x_{best,d} - x_d}{2}) \tag{3.4}$$

Ensuite, la hauteur d'onde de $X'$ est réinitialisée à $h_{max}$ et sa longueur d'onde est mise

à jour comme suit :

$$\lambda' = \lambda \frac{f(x)}{f(x')} \tag{3.5}$$

Le principe de WWO est présenté dans l'algorithme 12

**Algorithme 12** Optimisation par vagues d'eau (WWO)

```
Initialiser aléatoirement une population P de n solutions (ondes) ;
Tantque le critère d'arrêt n'est pas satisfait Faire
  Pour chaque vague x ∈ P Faire
    Propager x à un nouveau x' selon Eq. (3.1) ;
    Si f(x') > f(x) Alors
      Si f(x') > f(x_best) ; Alors
        Rupture x' en nouvelles vagues selon Eq. (3.3) ;
        Mettre à jour x_best avec x' ;
      FinSi
      Remplacer x par x' ;
    Sinon
      x.h = x.h − 1 ;
      Si x.h = 0 Alors
        Réfracter x à un nouveau x' selon Eq. (3.4) et (3.5) ;
      FinSi
    FinSi
    Metter à jour les longueurs d'onde selon Eq. (3.2) ;
  FinPour
FinTantque
Renvoyer la meilleure solution trouvée jusqu'à présent
```

## 3.3 Proposition d'un algorithme d'optimisation par vagues d'eau compacte (cWWO)

Dans cette section on va présenter la version compacte de l'algorithme par vagues d'eau expliqué en haut, appelé cWWO pour 'compact water wave optimization '. Elle suit le même principe des algorithmes compactes expliqué dans le chapitre (2). Aussi, nous avons effectué des changements sur les formules utilisées avec un réglage de paramètres de la méta-heuristique originale. Il est important de noter que nous avons implémenté une version simple de WWO dont l'étape de réfraction n'est pas prise en considération et nous avons introduit un nouveau paramètre $C$ divisé par $N_p$ dans la mise à jour du $PV$. Pour mieux comprendre le comportement de cWWO, il est présenté dans l'algorithme 13

**Algorithme 13** Optimisation par vagues d'eau compact (cWWO)

compteur $t = 0$ ;
**/initialisation du $PV$ ;
**Pour** $i = 1 : dim$ **Faire**
  Initialization $\mu[i] = 0$ ;
  Initialization $\sigma[i] = 10$ ;
**FinPour**
**/Initialisation du Global best ;
Générer $x_{best}$ vague à partir de $PV$ ;
**/Initialisation du local best ;
Générer $x_{lb}$ à partir de $PV$ ;
$x_{lb}$ = générer de$(PV)$ ;
**Tantque** le critère d'arrêt n'est pas atteint **Faire**
  $C = MaxIt/(C_{min} \times C_{max} \times dim)^{iter} \times rand(0, 1)$ ;
  **/Mettre à jour de $x_{lb}$ ;
  $x'_{lb} = x_{lb} + rand(-1, 1).\lambda$ ;
  Sélection du local best ;
  $[winner, loser] = ComparerFitness(x'_{lb}, x_{lb})$ ;
  **Si** $winner == x'_{lb}$ **Alors**
    $x_{lb} = x'_{lb}$ ;
    $[winner, loser] = ComparerFitness(x'_{lb}, x_{best})$ ;
    **Si** $winner == x'_{lb}$ **Alors**
      $x_{best} = x'_{lb}$ ;
      $\beta = 1/((t.t_{max}.dim)^t)$ ;
      $x'_{lb} = x_{lb} + randn(-1, 1).\beta.rand(0, 1)$ ;
    **FinSi**
  **Sinon**
    $[winner, loser] = ComparerFitness(x_{lb}, x_{best})$ ;
    **Si** $winner == x_{lb}$ ; **Alors**
      $x_{best} = x_{lb}$ ;
    **FinSi**
  **FinSi**
  **/Mise à jour du $PV$ ;
  **Pour** $i = 1 : dim$ **Faire**
    $\mu^{t+1}[i] = \mu^t[i] + \frac{C}{N_p}(winner[i] - loser[i])$ ;
    $\sigma^{t+1}[i] = \sqrt{(\sigma^t[i])^2 + (\mu^t[i])^2 - (\mu^{t+1}[i])^2 + \frac{C}{N_p}(winner^2[i] - loser^2[i])}$ ;
  **FinPour**
  **/Mettre à jour de $\lambda$
  $\lambda = \lambda.\alpha^{-(f(x)-f_{min}+\epsilon)/(f_{max}-f_{min}+\epsilon)}$ ;
  $t = t + 1$ ;
**FinTantque**

## 3.4 Les algorithmes compactes proposés

### 3.4.1 Hybridation basée cPSO_cDE (variante une)

L'optimisation par essaim de particules compact est caractérisée par sa robustesse. Aussi, elle est facile à combiner avec d'autres méthodes d'optimisation. L'algorithme hybride proposé que nous avons nommé (cPSO_cDE $v1$) est basé sur le cDE et le cPSO avec perturbation du $PV$. Cette nouvelle approche utilise les mêmes règles qui caractérisent le PSO compacte et l'opérateur de croisement binomial du cDE ($C_r = 0.5$).
Cet opérateur est appliqué entre la solution actuelle ($x^t$) et l'élite ($x_{gb}$). Il est à noter que nous avons utilisé le mécanisme de perturbation après la mise à jour du $PV$. Le principe de cette variante est présenté dans l'algorithme 14.

### 3.4.2 Hybridation basée cPSO_cDE (variante deux)

Cette approche proposée nommée (cPSO-cDE $v2$ ) est basée aussi sur le cPSO et le cDE. Nous avons utilisé l'opérateur de croisement du cDE $C_r(C_r = 0.1)$ pour la mise à jour de la vélocité des particules du cPSO. Le principe est présenté dans l'algorithme 15

### 3.4.3 Hybridation basée DEcDE

Cette nouvelle approche est basée sur le DEcDE expliquée précédemment avec tous ses paramètres. Nous avons introduit aussi l'opérateur de croisement binomiale($C_r = 0.3$). Cette proposition a pour but de bénéficier des avantages des deux opérateurs de croisement. Le détail de cette approche est montré dans l'algorithme 16

**Algorithme 14** (cPSO_cDE $v1$ ) pseudo-code

compteur $t = 0$ ;
**/initialisation du $PV$ ;
**Pour** $i = 1 : dim$ **Faire**
  Initialization $\mu[i] = 0$ ;
  Initialization $\sigma[i] = 10$ ;
**FinPour**
**/Initialisation du Global best ;
Générer $x_{gb}$ à partir de $PV_{lb}$ ;
générer de manière aléatoire $x^t$ et $v^t$ ;
**Tantque** le critère d'arrêt n'est pas atteint **Faire**
  **/Nouvelle initialisation du local best ;
  $x^t_{lb}$ = générer de$(PV_{lb})$ ;
  **/Mettre à jour la position et la vélocité ;
  $v^{t+1} = \phi_1 v^t + \phi_2(x^t_{lb} - x^t) + \phi_3(x^t_{gb} - x^t)$ ;
  $x^{t+1} = \gamma_1 x^t + \gamma_2 v^{t+1}$ ;
  **/Croisement ;
  **Pour** $i = 1 : dim$ **Faire**
    Générer $rand(0, 1)$ ;
    **Si** $rand(0, 1) > C_r$ **Alors**
      $x^{t+1}[i] = x_{gb}[i]$ ;
    **FinSi**
  **FinPour**
  Sélection du local best ;
  $[winner, loser] = ComparerFitness(x^{t+1}, x_{lb})$ ;
  Mise à jour du $PV_{lb}$ selon l'algorithme (11) ;
  **/Perturbation du $PV$ ;
  **Si** $rand(0, 1) < M_p$ **Alors**
    **Pour** $i = 1 : dim$ **Faire**
      $\mu^{t+1}[i] = \mu^{t+1}[i] + 2\tau.rand(0, 1) - \tau$ ;
      $\sigma^{t+1}[i] = \sqrt{(\sigma^{t+1}[i])^2 + \tau.rand(0, 1)}$ ;
    **FinPour**
  **FinSi**
  Mise à jour du Global best selon l'algorithme (11) ;
  $t = t + 1$ ;
**FinTantque**

### 3.4.4 Ajustement des paramètres du cDE

Le choix des paramètres joue un rôle crucial pour l'efficacité des algorithmes compacts. Le réglage des paramètres des algorithmes influence directement sur leurs performances puisqu'il varie d'un problème à un autre et souvent basé sur l'expertise de l'utilisateur.

La probabilité d'application de l'opérateur croisement $CR$ de l'algorithme cDE est l'un des paramètres que nous avons essayé d'ajuster. L'opérateur $C_r$ doit être choisi dans l'intervalle $[0, 1]$. Nous avons essayé plusieurs valeurs pour ce paramètre et nous avons remarqué que la valeur $(Cr = 0.1)$ donne des résultats mieux que ceux du cDE et le

**Algorithme 15** (cPSO_cDE $v2$ ) pseudo-code

compteur $t = 0$ ;
**/initialisation du $PV$ ;
**Pour** $i = 1 : dim$ **Faire**
Initialization $\mu[i] = 0$ ;
Initialization $\sigma[i] = 10$ ;
**FinPour**
**/Initialisation du Global best ;
Générer $x_{gb}$ à partir de $PV_{lb}$ ;
générer de manière aléatoire $x^t$ et $v^t$ ;
**Tantque** le critère d'arrêt n'est pas atteint **Faire**
**/Nouvelle initialisation du local best ;
$x_{lb}^t =$ générer de$(PV_{lb})$ ;
**/Mettre à jour la vélocité ;
**Si** $rand(0,1) > C_r$ **Alors**
$v^{t+1} = \phi_1 v^t + \phi_2(x_{lb}^t - x^t) + \phi_3(x_{gb}^t - x^t)$ ;
**FinSi**
**/Mettre à jour la position ;
$x^{t+1} = \gamma_1 x^t + \gamma_2 v^{t+1}$ ;
Sélection du local best ;
$[winner, loser] = ComparerFitness(x^{t+1}, x_{lb})$ ;
Mise à jour du $PV_{lb}$ selon l'algorithme (11) ;
**/Perturbation du $PV$ ;
**Si** $rand(0,1) < M_p$ **Alors**
**Pour** $i = 1 : dim$ **Faire**
$\mu^{t+1}[i] = \mu^{t+1}[i] + 2\tau.rand(0,1) - \tau$ ;
$\sigma^{t+1}[i] = \sqrt{(\sigma^{t+1}[i])^2 + \tau.rand(0,1)}$ ;
**FinPour**
**FinSi**
Mise à jour du Global best selon l'algorithme (11) ;
$t = t + 1$ ;
**FinTantque**

ne_cDE de base.

**Algorithme 16** pseudo-code DEcDE_$CR_{binomiale}$

Compteur $t = 0$ ;
**/Initialisation du $PV$ ;
**Pour** $i = 1 : dim$ **Faire**
  Initialization $\mu[i] = 0$ ;
  Initialization $\sigma[i] = 10$ ;
**FinPour**
Générer **elite** à partir du $PV$ ;
**Tantque** le critère d'arrêt n'est pas atteint **Faire**
  **Si** $rand(0, 1) < M_t$ ; **Alors**
    **/Mutation trigonométrique ;
    Générer 3 individus $x_r, x_s$, et $x_t$ a partir de $PV$ ;
    $x_{off} = \frac{(x_r+x_s+x_t)}{3} + (p_s - p_r)(x_r - x_s) + (p_t - p_s)(x_s - x_t) + (p_r - p_t)(x_t - x_r)$ ;
  **Sinon**
    **/Mutation ;
    Générer 3 individus $x_r, x_s$, et $x_t$ a partir de $PV$ ;
    $x'_{off} = x_t + F(x_r - x_s)$ ;
    **/Croisement ;
    Appliquer le croisement exponentiel montrer dans l'algorithme 9 et générer $x_{off}$ ;
  **FinSi**
  **/Sélection Elite ;
  $[winner, loser] = ComparerFitness(x_{off}, elite)$ ;
  **Si** $x_{off} == winner$ **Alors**
    $elite = x_{off}$ ;
  **FinSi**
  **/Mise à jour du $PV$ ;
  Mise à jour du $PV$ Appliquer selon l'algorithme (10) ;
  **Si** $rand(0, 1) > M_t$ ; **Alors**
    Générer 1 individus $ind$ a partir de $PV$ ;
    **/Croisement ;
    Appliquer le croisement binomiale ;
    **Pour** $i = 1 : dim$ **Faire**
      Générer $rand(0, 1)$ ;
      **Si** $rand(0, 1) > C_r$ **Alors**
        $ind[i] = elite[i]$ ;
      **FinSi**
    **FinPour**
  **FinSi**
  **/Sélection Elite ;
  $[winner, loser] = ComparerFitness(ind, elite)$ ;
  **Si** $ind == winner$ **Alors**
    $elite = ind$ ;
  **FinSi**
  **/Mise à jour du $PV$ ;
  **/Perturbation du $PV$ ;
  Mise à jour et la Perturbation du $PV$ Appliquer selon l'algorithme (10) ;
  $t = t + 1$ ;
**FinTantque**

# Conclusion

Dans ce chapitre, nous avons abordé en détails quelques variantes de méta-heuristiques compactes. Nous avons insisté sur l'utilité de l'hybridation afin d'améliorer les algorithmes compacts existants en plus nous avons introduit une nouvelle approche de méta-heuristique compacte (cWWO). Toutes ces nouvelles variantes présentées dans ce chapitre n'exigent que des capacités de calculs minimales.

Dans le chapitre qui suit, les performances de ces algorithmes seront étudiées, testées et validées sur un ensemble de fonctions Benchmark tout en les comparant aux méta-heuristiques compactes de l'état de l'art.

# Chapitre 4

# Étude expérimentale et validation

## Sommaire

# Introduction

L'analyse comparative des algorithmes d'optimisation est une étape cruciale pour évaluer les performances des algorithmes d'optimisation et comprendre leur points faibles ainsi que leur points forts faiblesses. Cependant, cette tâche s'avère fastidieuse et difficile à réaliser, même dans le cas d'un objectif unique.

Dans ce chapitre, nous allons tester, discuter et comparer l'efficacité des algorithmes proposés dans le chapitre précédent. Nous allons utiliser la plateforme COCO (Comparing Continous Optimiser) qui fournit à l'utilisateur un ensemble de fonctions Benchmark BBOB (Black Box Optimization Benchmarking) pour faciliter une grande partie de ce travail. Ce dernier contient des fonctions de test qui permettront de comparer et valider les performances de nos contribution et aussi de les classer par rapport aux algorithmes présentés dans le Chapitre 2.

## 4.1 Choix des paramètres

Les algorithmes proposés cPSO_cDE$v1$, cPSO_cDE$v2$, pe_cDE$v2$, ne_cDE$v2$, DEcDE_$CR_{binomiale}$ et l'algorithme compacté cWWO, sont comparés ici à six algorithmes évolutifs compacts (cEAs) :

- Deux algorithmes génétiques avec compacts représentation réel : pe_rcGA et ne_rcGA,
- Deux algorithmes d'évolution différentielle compacts avec la stratégie "$/rand/1/$" : persistant (pe_cDE), et non persistant (ne_cDE).
- Optimisation par essaim de particules compact (cPSO),
- L'algorithme de l'évolution différentielle compact avec croisement exponentiel et perturbation du vecteur probabiliste (DEcDE).

Les paramètres de chaque algorithme de comparaison utilisé dans cette expérimentation sont tirés de la articles originaux.
Les valeurs communes des paramètres pour tous les algorithmes de comparaison et les algorithmes proposés sont résumées dans la table 4.1. Aussi, la comparaison des six algorithmes compacts sur le testbeds BBOB- 2015 est présenté dans la figure 4.1.

| **Nom du paramètres** | **Valeur** |
|---|---|
| La taille de population : $NP$ | 300 |
| Les dimensions : $dim$ | $2, 3, 5, 10, 20, 40$ |
| Nombre maximum des évaluations | $10000 * dim$ |
| Nombre des exécutions | 15 |

TABLE 4.1 – Paramètres communs de tous les algorithmes compacts

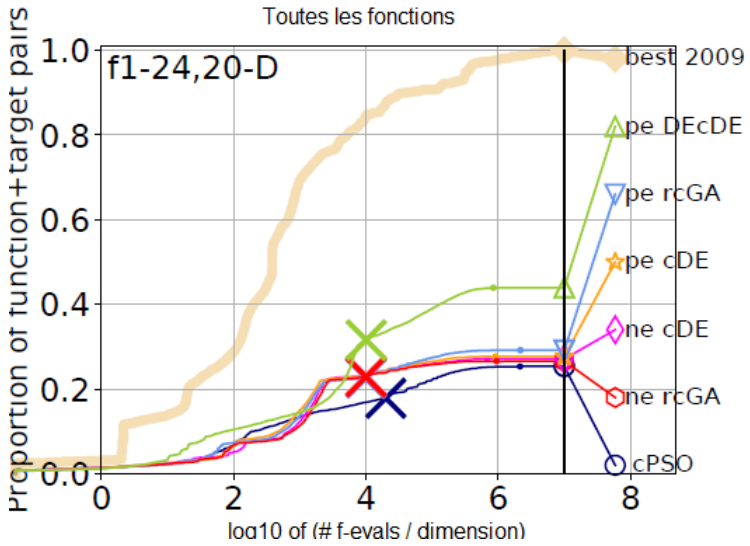



FIGURE 4.1 – Comparaison des six algorithmes compacts

Le choix des paramètres (Tuning of parameters) est directement proportionnelle à la performance des algorithmes. Autrement dit, le réglage des paramètres des algorithmes influence directement sur leurs performances. Il varie d'un problème à un autre est souvent basé sur l'expertise de l'utilisateur.

Pour l'algorithme d'optimisation par vagues d'eau compact cWWO, les valeurs des paramètres ont été réglés comme montré dans la table 4.2

Nous avons ajusté le paramètre $CR$ "*la probabilité de croisement*" dans les algorithmes proposés de la façon suivante :

- Les deux variantes des algorithmes hybrides cPSO_cDE$v1$, et cPSO_cDE$v2$ gardent les mêmes paramètres du cPSO avec leurs valeurs et nous avons choisi la valeur 0.5 pour $C_r$. Ce choix est effectué pour assurer un croisement de la nouvelle position avec le meilleur global dans la première variante. Quant à la deuxième variante, ce choix est définie comme condition pour mettre à jour la vélocité.
- Pour l'algorithme d'évolution différentielle compact élitiste persistant (pe_cDE$v2$) et non persistant (ne_cDE$v2$), nous avons choisi pour $C_r$ la valeur 0.1 et les autres paramètres restent les mêmes pour les deux algorithmes.
- L'algorithme hybride DEcDE_$CR_{binomiale}$ garde tous les paramètres de l'algorithme

| **Nom du paramètres** | **Valeur** |
|---|---|
| La longueur d'onde : $\lambda$ | 1 |
| Le coefficient de réduction de la longueur d'onde : $\alpha$ | 1.002 |
| Petite constante : $\epsilon$ | 0.0001 |
| Max des itérations :$MaxIt$ | $maxfunevals$ |
| Paramètre dynamique : $C$ | $MaxIt/(C_{min} \times C_{max} \times dim)^{iter} \times rand(0,1)$ |
| Le coefficient de rupture : $\beta$ | $1/((iter \times MaxIt \times dim)^{iter})$ |

TABLE 4.2 – Paramètres de l'algorithme d'optimisation par vagues d'eau compact

(DEcDE) tout en introduisant un autre opérateur $C_r$ nommé $C_{r1}$ qui prend la valeur 0.3 utilisé dans son article d'origine .

## 4.2 Fonctions benchmarks

La plateforme BBOB (Black-Box-Optimization Benchmarking)[28] introduite par la série d'ateliers BBOB-GECCO qui déjà établie et couronnée de succès est devenue une norme bien conçue pour l'analyse comparative des algorithmes d'optimisation continue. Cette plateforme contient 24 fonction de test et permet aussi de :

- Développer une méthodologie standard similaire pour l'analyse comparative des algorithmes d'optimisation de type boîte noire pour des domaines discrets et combinatoires.
- Fournit des bancs d'essai de fonctions de référence.
- La génération des données de sortie.
- La présentation et le post-traitement des résultats sous forme des graphes et tableaux.

### 4.2.1 Symboles, Constantes et Paramètres

- $f_{opt}$ : Valeur optimal de la fonction objectif, définie individuellement pour chaque fonction de becnhmark.
- $\Delta f$ : Précision à atteindre, c'est-à-dire la plus petite différence possible de la valeur $f_{opt}$.
- $f_{target} = f_{opt} + \Delta f$ : une valeur cible de la fonction objectif à atteindre. La plus petite valeur de la fonction finale définie est $f_{target} = f_{opt} + 10^{-8}$, Aussi des valeurs plus importantes pour $f_{target}$ sont évaluées.

- $Ntrial = 15$ est le nombre d'essais qu'on a utilisé pour chaque configuration unique, c'est-à-dire chaque fonction et chaque dimension. Une instance de fonction différente est utilisée dans chaque essai. La performance est évaluée sur tous les essais $Ntrial$.
- $D = 2; 3; 5; 10; 20; 40$ les dimensions de l'espace de recherche utilisées pour toutes les fonctions. La dimension 40 est facultative et peut être omise.

### 4.2.2 Protocole expérimental

Les expérimentations numériques sont menées sur un banc d'essai comprenant 24 fonctions de test sans bruit (noisless functions). Ces fonctions ont été construites de telle sorte qu'elles reflètent les difficultés d'application réelles et elles sont divisées en différent groupes : fonctions séparables, fonctions à conditionnement faible ou modérées, fonctions avec un mauvais conditionnement et uni-modales, multimodales avec une structure globale adéquate, multimodales avec une faible structure globale. Nous pouvons utiliser l'espace de recherche $2, 3, 5, 10, 20$ et $40$ pour toutes les fonctions. En outre, toutes les fonctions ont leur optimum dans l'intervalle $[-5; 5]$, qui peut être un paramètre raisonnable pour le domaine de recherche. Chaque fonction est testée sur cinq instances différentes. Les tests sont répétées trois fois pour chaque instance [4]. Donc la performance de l'algorithme est évaluée sur 15 essais. Le critère de réussite d'une exécution est d'atteindre la valeur cible $f_{target} = f_{opt} + \Delta f$. Ces expérimentations ont été menées avec un processeur CPU Intel (R) Core (TM) $i5 - 3340$M CPU avec $2.70GHz$ et $4Go$ de RAM en utilisant le code Matlab fourni par BBOB.

### 4.2.3 Entrées d'algorithme et initialisation

Les algorithmes réels proposés dans nos contributions, sont testés et exécutés avec leur paramètres au testbeds-BBoB, qui mis à disposition un code MATLAB pour l'implémentation des fonctions. Nous l'avons modifié afin de l'adapter à notre travail. Il a comme entrée les paramètres suivantes : $(FUN, Dim, f_{target}, maxfunevals)$.

Le nombre maximal d'évaluations de la fonction fitness ($maxfunevals$) est fixé à $10000 * D$. Ce nombre limite l'utilisation de la fonction d'évaluation pour accorder une comparaison équitable entre différents algorithmes. L'évaluation des solution se fait par une fonction prédéfinie $FUN$ jusqu'à atteindre le nombre maximal d'évaluation de la fonction objectif :$maxfunevals$ ou son $f_{target}$.

### 4.2.4 Post-traitement

Des algorithmes avec n'importe quel budget d'évaluations de fonctions, petites ou grandes, sont pris en compte dans l'analyse des résultats. Donc, nous avons utilisé python

pour convertir les fichiers que nous avons obtenus après l'implémentation et l'exécution des codes Matlab en graphes et tableaux pour faciliter la lecture des résultats. Étant donné que les données de sortie se trouvent dans un (des) dossier (s), une commande de python est nécessaire pour interpréter les résultats obtenu par le BBoB. La mesure de performances adoptées dans notre étude est la proportion des problèmes résolus par un algorithme pour un grande budget.

## 4.3 Résultats expérimentaux

Dans cette section, nous allons présenter deux séries d'expérimentations. Dans un premier temps, nous comparons les variantes proposées avec les six algorithmes compacts décrits au chapitre (2). Par la suite, nous comparons l'ensemble des algorithmes proposées pour déterminer les meilleures d'entre eux voir les figures 4.6 - 4.7 .

### 4.3.1 Résultats et discussion

Dans ce qui suit, nous allons interpréter et discuter les résultats obtenus du post traitement de notre étude, résumés par les figures 4.3 - 4.4 et par les tableaux de 4.3 à 4.6 dans les dimensions 5 et 20 respectivement.

Nous prendrons dans chaque figure, les graphes qui représentent les fonctions de la distribution empirique cumulée des temps d'exécution espérés (ERT), exprimée en terme d'évaluations de la fonction objectif divisés par la dimension. La ligne "Best 2009" correspond au meilleur ERT (Expected Runnig Time) observé depuis BBOB-2009. Nous verrons le comportement de chaque variante et chaque algorithme compact.

La figure 4.2 représente la comparaison entre d'un coté le cGA avec élitisme persistant (pe_cGA) et non persistante (ne_cGA) sans utilisation du croisement binomial et de l'autre coté le pe_cGA , ne_cGA avec croisement dans la dimension 5 et 20. Nous avons observé une amélioration remarquable. Ce que nous considérons être comme des nouvelles variantes.

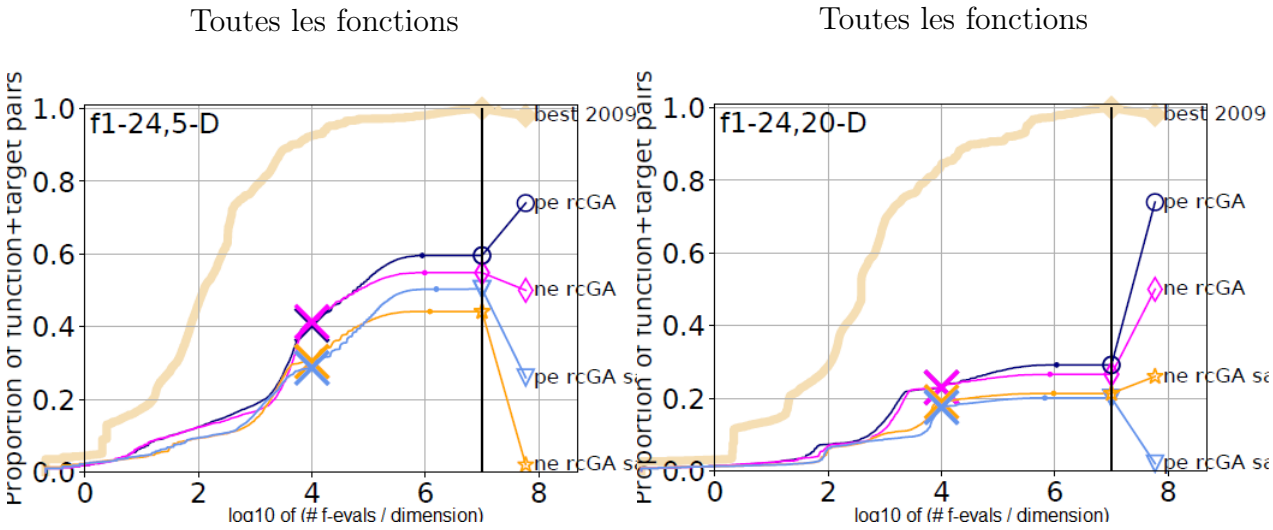


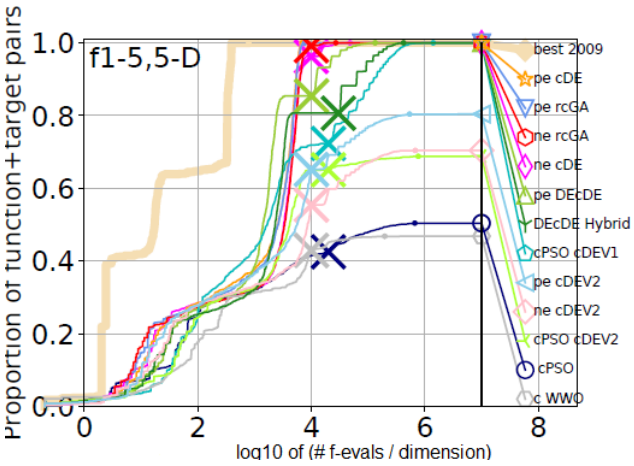

FIGURE 4.2 – Comparaison des quatre algorithmes compacts dans la dimension 5 et 20

Les performances résultant de la comparaison des toutes les algorithmes dans la dimension 5 sont illustrées par la figure (4.3). On observe que pour le même budget ($10000 * dimFEs$), le DEcDE_$CR_{binomiale}$(DEcDE Hybrid) conserve presque son statut et son classement par rapport aux autre algorithmes dans tout les groupes de fonctions et pour l'agrégation de toutes les fonctions, alors que les autres algorithmes affichent un comportement différent. Un algorithme ne se comporte pas de la même manière lorsqu'il passe d'un ensemble de fonctions à un autre.

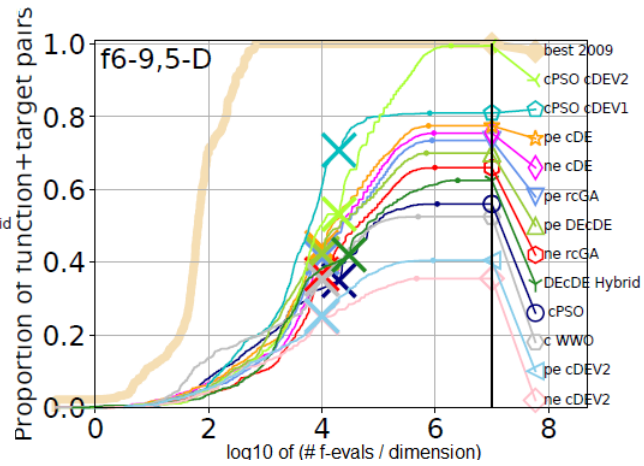
Dans le cas des fonctions séparables, les algorithmes proposés (DEcDE Hybrid, et cPSO_cDE$v1$), ont le même comportement des algorithmes d'origines(pe_rcGA, ne_rcGA, pe_cDE, ne_cDE, et pe_DEcDE). Ils ont donné de très bons résultats car ils offert un comportement assez similaire à celui du meilleur artificiel (best 2009) ce qui est généralement exceptionnel. Alors que la performance des algorithmes (cPSO_cDE$v2$, pe_cDE$v2$, ne_cDE$v2$ et cWWO) est un peu faible. Les trois variantes(pe_cDE$v2$, ne_cDE$v2$, et cPSO_cDE$v2$) ont le même comportement. Ils restent meilleurs que cWWO qui adopte le même comportement que cPSO.

Concernant les fonctions modérées et mal conditionnées, elles sont caractérisées par la présence des optima locaux isolés et donc l'algorithme aura peu de chance de trouver l'optimum global. Ce type de fonctions peut alors influencer sur les performances d'un algorithme mais dans notre cas la variante cPSO_cDE$v2$ domine toujours les autre algorithmes avec un grand succès dans les fonctions modérées. Elle aussi offert un comportement assez similaire à celui du meilleur artificiel (best 2009), car elle a été capable de résoudre la presque totalité des problèmes ce qui le rend très intéressante pour la résolution des problèmes réels qui ont les caractéristiques de ce type de fonctions, suivi de la deuxième variante cPSO_cDE$v1$ de(80%). Il reste toujours meilleur dans le comporte-

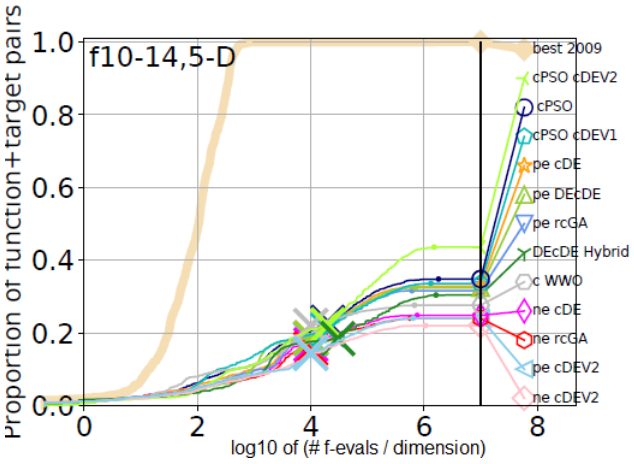
Fonctions séparables



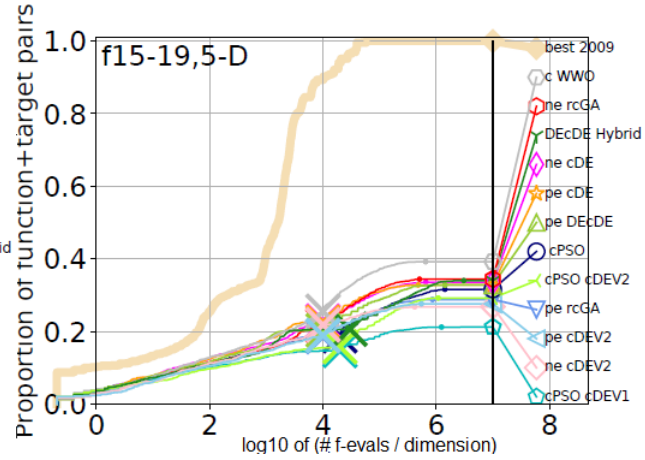

Fonctions modérées



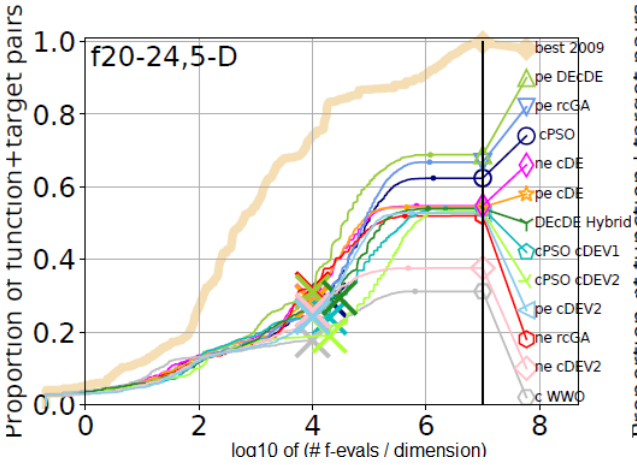

Fonctions mal conditionnées




Fonctions multimodale




Fonctions multimodale à structure globale faible




Toutes les fonctions

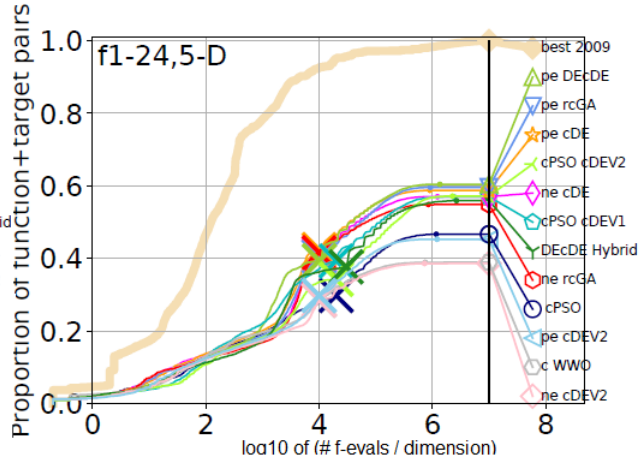



FIGURE 4.3 – Fonctions de distribution cumulative empirique (ECDFs) de l'ERT par groupe de fonctions pour la dimension 5

ment que tous les algorithmes en les fonctions mal conditionnées, mais il résout près de la moitié du problème.

Dans les fonctions multimodales, notre algorithme compact 'cWWO' présente un comportement supérieur à celui des autres algorithmes, suivi de le DEcDE Hybrid et le reste des algorithmes ont presque le même comportement. Pendant que dans les fonctions multimodales à structure globale faible le DEcDE Hybrid, cPSO_cDE$v$1, pe_cDE$v$2, et cPSO_cDE$v$2 exhibent un comportement relativement similaire en résolvant la même proportion des problèmes. Contrairement aux algorithmes ne_cDE$v$2 et cWWO qui n'ont pas pu résoudre qu'une petite partie des problème.

Finalement, dans le dernier graphe qui englobe les 24 fonctions, on peut dire que DEcDE Hybrid, cPSO_cDE$v$2, et cPSO_cDE$v$2 étaient nettement supérieur à toute autre variante et montrent plus ou moins le même comportement, le pe_cDE$v$2 est dans la moyenne et le ne_cDE$v$2, et cWWO restent toujours moins performant.

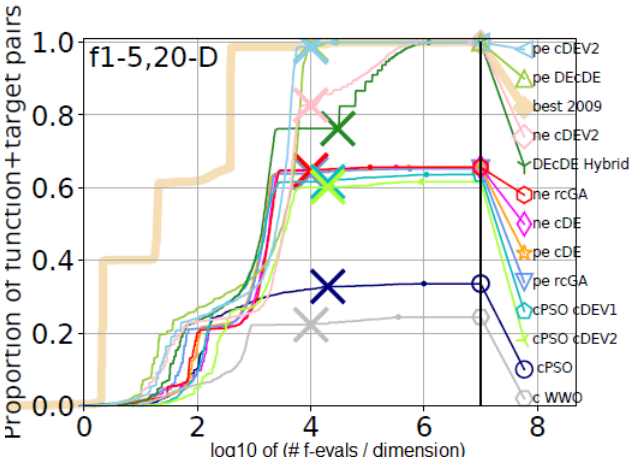
Comme l'indique la figure 4.4, les performances des 6 algorithmes se détériorent dans le cas de la dimension 20, car la difficulté globale du problème augmente fortement avec les dimensions élevées. La chance d'un algorithme de trouver l'optimum global sera alors faible. Cela n'empêche pas les variantes DEcDE Hybrid, cPSO_cDE$v$2, cPSO_cDE$v$1,pe_cDE$v$2, et ne_cDE$v$2 de surpasser, généralement, ses concurrentes. Il est aussi remarquable que la variante cWWO, tout comme dans la cas de la dimension 5, reste moins puissante.

Sur les fonctions séparables, les trois variantes DEcDE Hybrid, pe_cDE$v$2,
et ne_cDE$v$2 ont marqué de très bons résultats par rapport aux autres algorithmes de base compacts sauf le pe_DEcDE qui présente un similaire similaire. Ils sont tous similaires dans leur comportement au meilleur artificiel (best 2009).

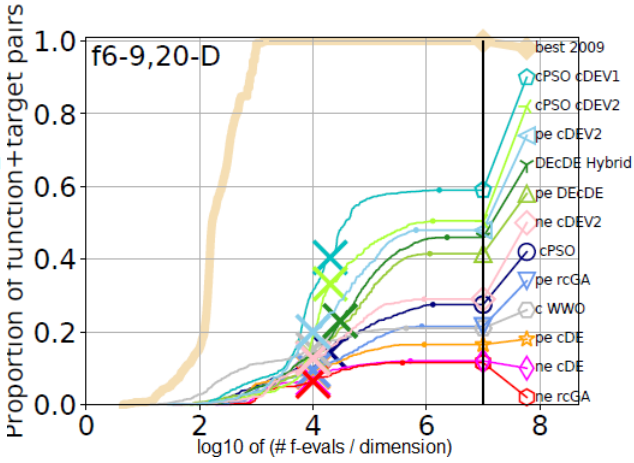
Le truc qu'on a remarqué ici, c'est que le DEcDE qui évolue en utilisant un croisement exponentiel a donné des résultats remarquables par rapport aux autres algorithmes, suivi par le ne_cGA en utilisant un croisement binomiale. C'est pour cela qu'on a constaté que pour les fonctions séparables, un croisement vu dans le DEcDE, et ne_cGA améliore les performances des algorithmes et son absence dans le cPSO a rendu cet algorithme le moins performant, par conséquent on a constaté une amélioration dans DEcDE Hybrid en ajoutant le croisement binomiale. Donc, on a gagné les avantages des deux croisement dans DEcDE Hybrid et la point fort de cPSO_cDE$v$1. Les variantes compactes cPSO_cDE$v$1, et cPSO_cDE$v$2 ont réussi à résoudre 63% des problèmes, avec le même comportement et ont affiché environ trois fois plus de performances que la variable cWWO qui n'a résolu qu'une petite portion des problèmes (23%).

Pour les fonctions modérées, bien qu'elle soit caractérisée par le problème de optima locaux, où on peut bien observer la supériorité de la variante compacte cPSO_cDE$v$1 qui

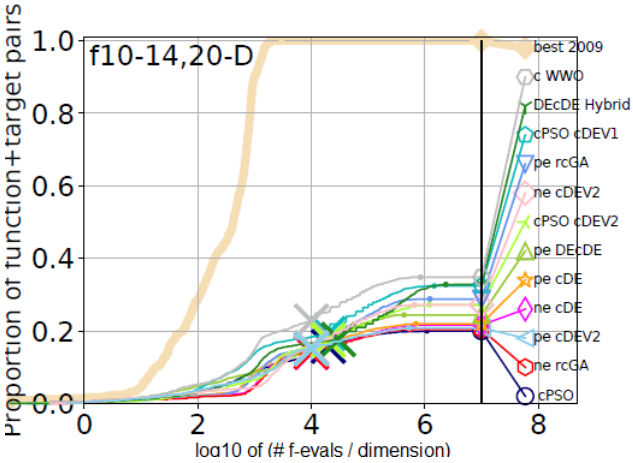
Fonctions séparables



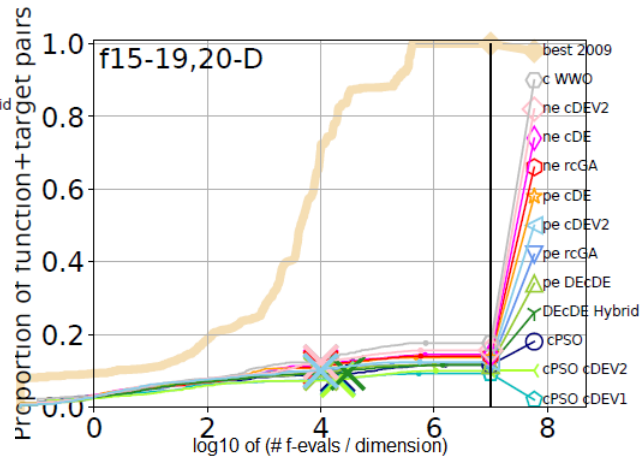

Fonctions modérées



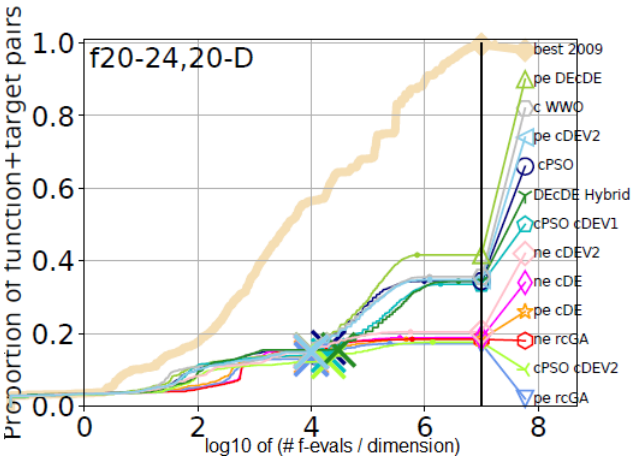

Fonctions mal conditionnées




Fonctions multimodale




Fonctions multimodale à structure globale faible




Toutes les fonctions

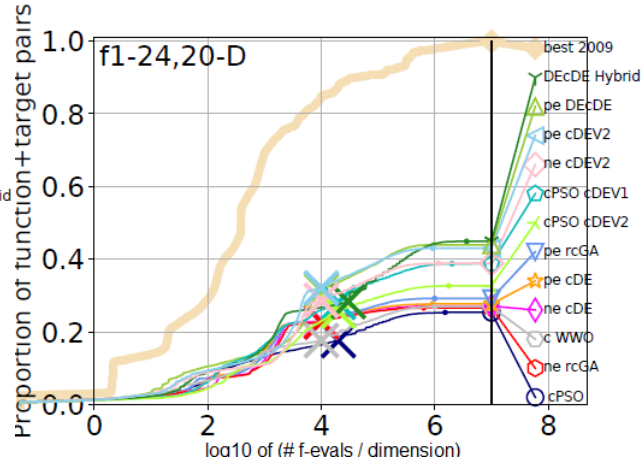



FIGURE 4.4 – Fonctions de distribution cumulative empirique (ECDFs) de l'ERT par groupe de fonctions pour la dimension 20

offert un bon comportement, suivi par le cPSO_cDE$v2$ , DEcDE Hybrid, et pe_cDE$v2$ presque par le même comportement. Comme toujours le cWWO, reste moins puissante et le ne_cDE$v2$ est en dessous de la moyenne. Donc, on pense que la perturbation du $PV$ a permis au cPSO_cDE$v1$ à résoudre 60% des problèmes en évitant un blocage sur les optima locaux.

Dans le cas des fonctions mal conditionnées on observe qu'elles ont une influence considérable sur les performance des algorithmes compacts. Mais le cWWO domine toujours les autres algorithmes suivi par le DEcDE Hybrid, cPSO_cDE$v2$. Il est clair que les performances se détériorent d'une manière remarquable puisque les algorithmes n'arrivent pas à faire une bonne balance entre l'exploration et l'exploitation, et donc de converger vers l'optimum global. On peut s'apercevoir, globalement, les même remarques dans le cas des fonctions multimodales et multimodales à structure globale faible.

On voit que tous les algorithmes sont regroupés avec des performances faibles, on constate que dans les grande dimension ces algorithmes sont moins performants pour ce genre de fonctions.

Finalement, et afin déterminer le meilleur algorithme d'optimisation compacts, on s'intéresse a l'analyse du graphe qui englobe toutes les fonctions BBOB, car il est difficile de déterminer exactement le meilleur algorithme a partir des groupes de fonctions qui ont des propriétés très générales. Dans notre cas on peut voir la supériorité de la variante DEcDE Hybrid (DEcDE_$CR_{binomiale}$) et pe_cDE$v2$, suivie par le ne_cDE$v2$ et le cPSO_cDE$v1$ puis cPSO_cDE$v2$ qui a été moins puissant que ces variantes, et a la fin vient le cWWO. Ce dernier n'est peut-être pas l'un des premiers ici mais il était dominant et classent le premier dans le groupe des fonctions qui souffrent de problèmes des optima locaux. Donc l'algorithme permet un bon équilibre entre l'exploration et l'exploitation mais il doit être amélioré.

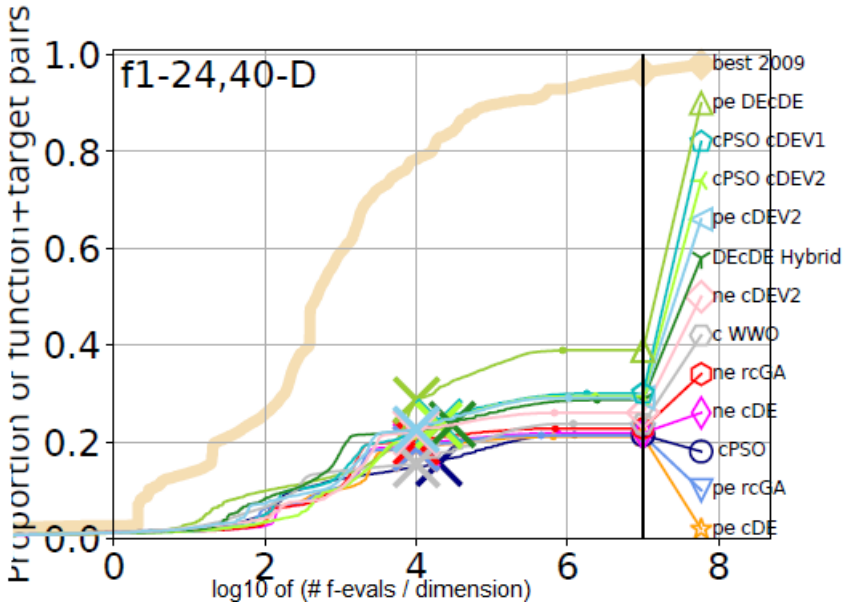
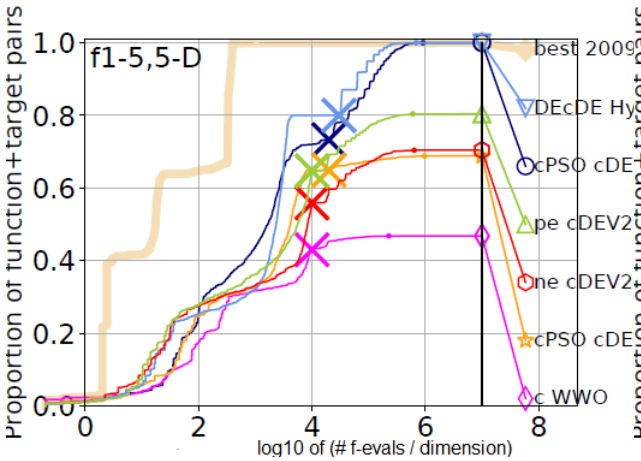


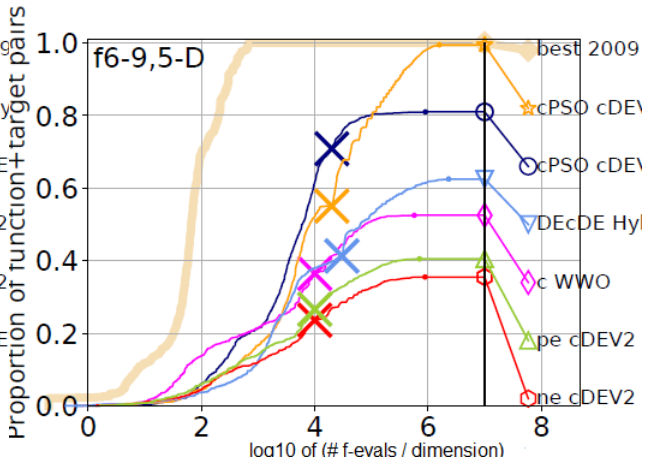

FIGURE 4.5 – Fonctions de distribution cumulative empirique (ECDFs) de l'ERT du Toutes les fonctions pour la dimension 40

Pour valider l'efficacité, la qualité et le succès de la meilleure variante de notre étude expérimentale, nous présentons dans les tableaux de 4.3 à 4.6 les taux de succès obtenus par chaque variante, exprimés en nombre des instances résolus pour les dimensions 5 et 20 respectivement. Ce qui confirme au mieux les résultats graphique déjà discutés.

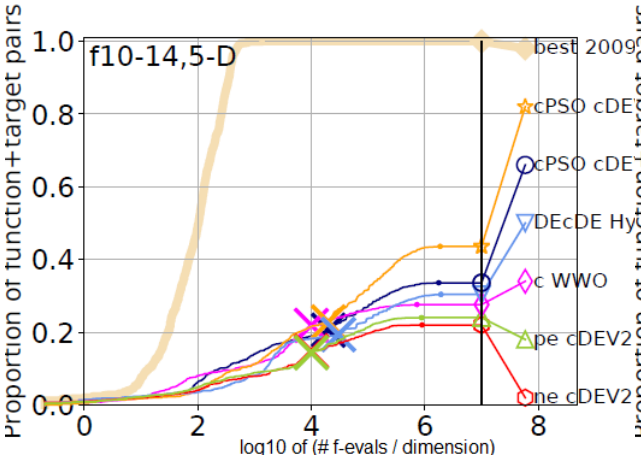
Fonctions séparables



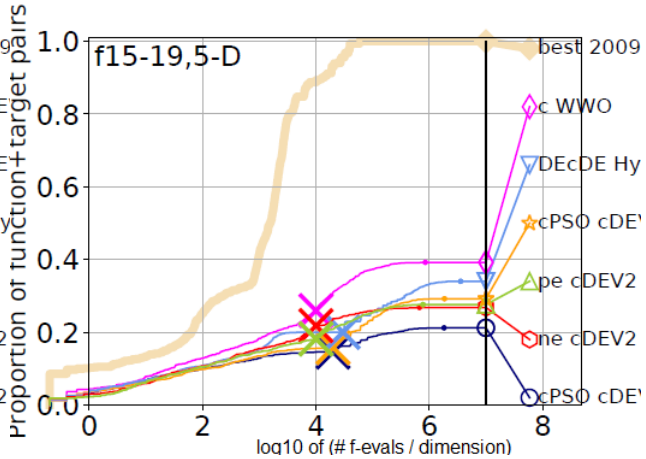

Fonctions modérées



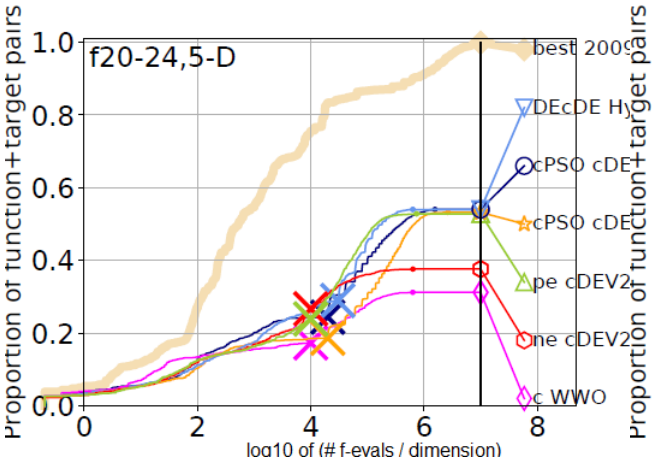

Fonctions mal conditionnées




Fonctions multimodale




Fonctions multimodale à structure globale faible




Toutes les fonctions

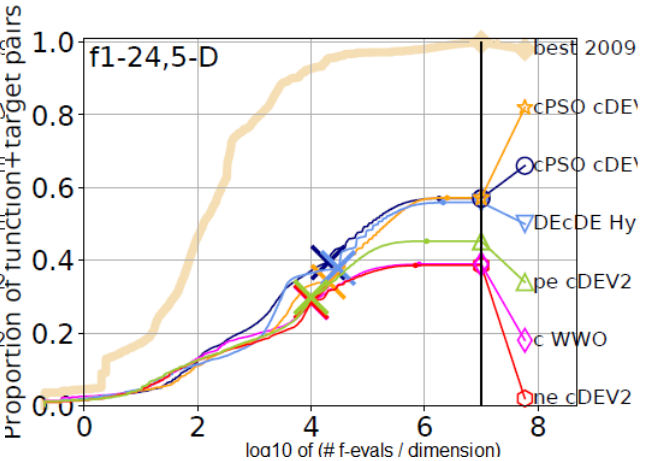
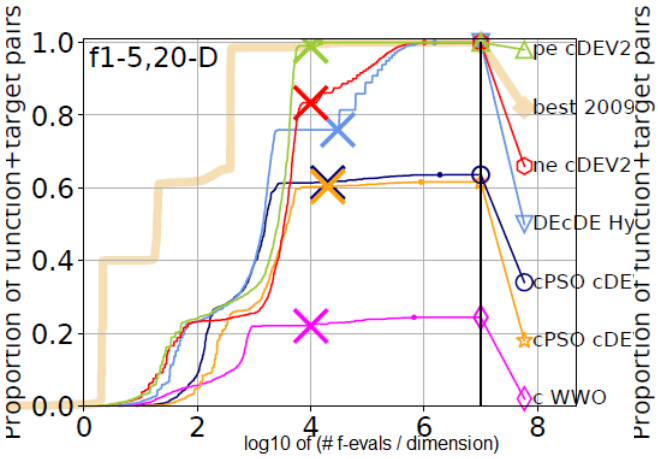
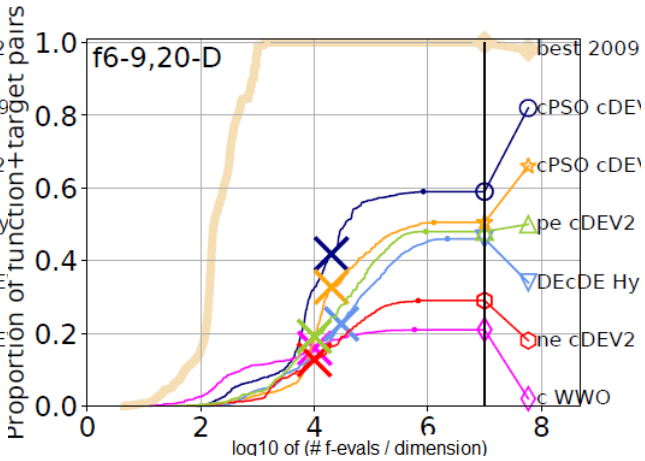



FIGURE 4.6 – Fonctions de distribution cumulative empirique (ECDFs) pour les variantes proposés par groupe de fonctions pour la dimension 5

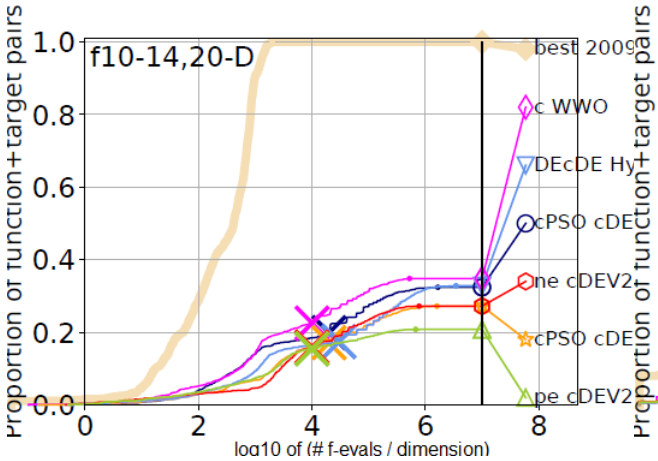
Fonctions séparables



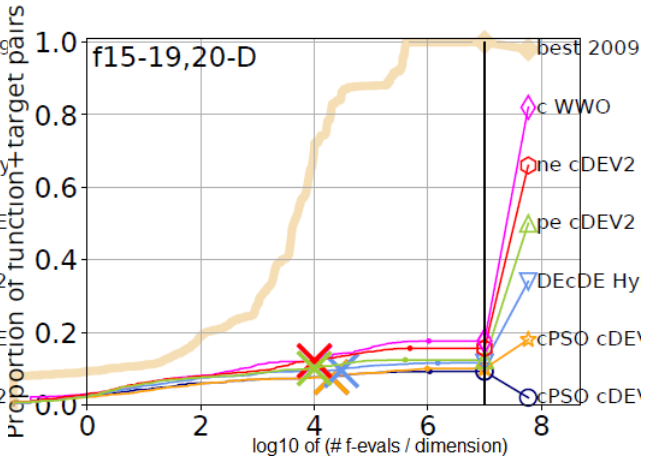

Fonctions modérées



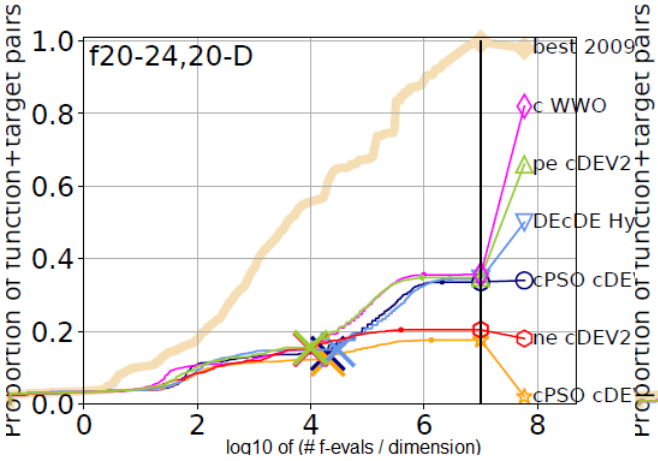

Fonctions mal conditionnées




Fonctions multimodale




Fonctions multimodale à structure globale faible




Toutes les fonctions

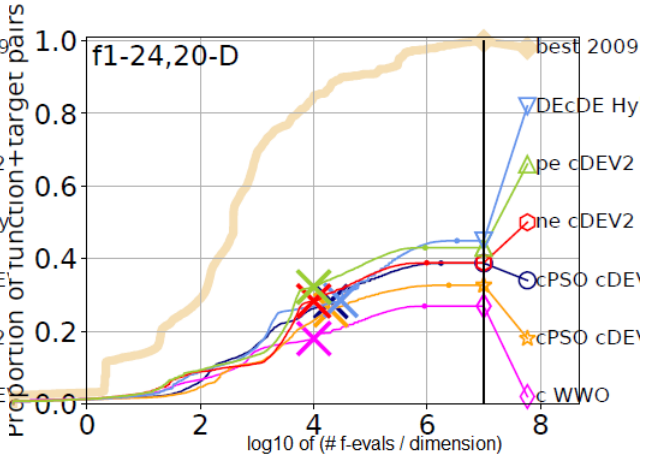



FIGURE 4.7 – Fonctions de distribution cumulative empirique (ECDFs) pour les variantes proposés par groupe de fonctions pour la dimension 20

# Conclusion

Dans ce chapitre, nous avons présenté les résultats expérimentaux des algorithmes proposés dans le chapitre précédent en les comparant avec les algorithmes compacts de base. Ces tests ont permis de mettre en évidence la relative efficacité des approches proposées par rapport aux algorithmes compacts de base. Nous avons pu constaté, à travers cette analyse, que les variantes proposées ont donné des résultats prometteurs dans ce nouveau domaine, en particulier la variante DEcDE Hybrid qui a pu surpassé les performances obtenues par tous les autre algorithmes. En outre, nous avons observé une baisse des performances de tous les algorithmes compacts étudiés dans la dimension 20. En conclue donc que les algorithmes compacts ne tolèrent pas la mise à échelle.

| **Fonctions Séparables** | | | | | |
|---|---|---|---|---|---|
| **Fonctions** / **Algorithme** | $f1$ | $f2$ | $f3$ | $f4$ | $f5$ |
| **cPSO** | 0/15 | 0/15 | 0/15 | 0/15 | 15/15 |
| **ne cDE** | 15/15 | 15/15 | 15/15 | 12/15 | 15/15 |
| **pe cDE** | 15/15 | 15/15 | 15/15 | 15/15 | 15/15 |
| **pe rcGA** | 15/15 | 15/15 | 15/15 | 15/15 | 15/15 |
| **ne rcGA** | 15/15 | 15/15 | 15/15 | 14/15 | 15/15 |
| **pe DEcDE** | 15/15 | 15/15 | 10/15 | 6/15 | 15/15 |
| **cPSO cDEV1** | 15/15 | 15/15 | 2/15 | 2/15 | 15/15 |
| **cWWO** | 12/15 | 0/15 | 0/15 | 0/15 | 15/15 |
| **cPSO cDEV2** | 15/15 | 15/15 | 0/15 | 0/15 | 15/15 |
| **DEcDE Hybrid** | 15/15 | 15/15 | 6/15 | 5/15 | 15/15 |
| **ne cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | 15/15 |
| **pe cDEV2** | 1/15 | 0/15 | 0/15 | 0/15 | 15/15 |
| **Fonctions modérées** | | | | | |
| **Fonctions** / **Algorithme** | $f6$ | $f7$ | $f8$ | $f9$ | |
| **cPSO** | 0/15 | 5/15 | 0/15 | 0/15 | |
| **ne cDE** | 10/15 | 1/15 | 0/15 | 0/15 | |
| **pe cDE** | 13/15 | 1/15 | 0/15 | 0/15 | |
| **pe rcGA** | 13/15 | 1/15 | 0/15 | 0/15 | |
| **ne rcGA** | 10/15 | 1/15 | 0/15 | 0/15 | |
| **pe DEcDE** | 13/15 | 1/15 | 0/15 | 0/15 | |
| **cPSO cDEV1** | 14/15 | 0/15 | 14/15 | 9/15 | |
| **cWWO** | 0/15 | 4/15 | 0/15 | 0/15 | |
| **cPSO cDEV2** | 1/15 | 1/15 | 13/15 | 9/15 | |
| **DEcDE Hybrid** | 12/15 | 0/15 | 0/15 | 0/15 | |
| **ne cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | |
| **pe cDEV2** | 1/15 | 0/15 | 0/15 | 0/15 | |
| **Fonctions mal conditionnées** | | | | | |
| **Fonctions** / **Algorithme** | $f10$ | $f11$ | $f12$ | $f13$ | $f14$ |
| **cPSO** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **ne cDE** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe cDE** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe rcGA** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **ne rcGA** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe DEcDE** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **cPSO cDEV1** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **cWWO** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **cPSO cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **DEcDE Hybrid** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **ne cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |

TABLE 4.3 – Taux de succès dans la dimension 5 pour les fonctions benchmarks de $f1$ à $f14$

| **Fonctions multimodales** | | | | | |
|---|---|---|---|---|---|
| **Fonctions** / **Algorithme** | $f15$ | $f16$ | $f17$ | $f18$ | $f19$ |
| **cPSO** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **ne cDE** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe cDE** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe rcGA** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **ne rcGA** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe DEcDE** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **cPSO cDEV1** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **cWWO** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **cPSO cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **DEcDE Hybrid** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **ne cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **Fonctions multimodales à structure globale faible** | | | | | |
| **Fonctions** / **Algorithme** | $f20$ | $f21$ | $f22$ | $f23$ | $f24$ |
| **cPSO** | 0/15 | 3/15 | 4/15 | 0/15 | 0/15 |
| **ne cDE** | 3/15 | 4/15 | 0/15 | 0/15 | 0/15 |
| **pe cDE** | 0/15 | 4/15 | 3/15 | 0/15 | 0/15 |
| **pe rcGA** | 1/15 | 2/15 | 0/15 | 0/15 | 0/15 |
| **ne rcGA** | 10/15 | 2/15 | 0/15 | 0/15 | 0/15 |
| **pe DEcDE** | 1/15 | 4/15 | 4/15 | 0/15 | 0/15 |
| **cPSO cDEV1** | 0/15 | 1/15 | 5/15 | 0/15 | 0/15 |
| **cWWO** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **cPSO cDEV2** | 0/15 | 1/15 | 1/15 | 0/15 | 0/15 |
| **DEcDE Hybrid** | 0/15 | 4/15 | 5/15 | 0/15 | 0/15 |
| **ne cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe cDEV2** | 0/15 | 3/15 | 0/15 | 0/15 | 0/15 |

TABLE 4.4 – Taux de succès dans la dimension 5 pour les fonctions benchmarks de $f15$ à $f24$

| **Fonctions Séparables** | | | | | |
|---|---|---|---|---|---|
| **Fonctions** / **Algorithme** | $f1$ | $f2$ | $f3$ | $f4$ | $f5$ |
| **cPSO** | 0/15 | 0/15 | 0/15 | 0/15 | 15/15 |
| **ne cDE** | 15/15 | 15/15 | 0/15 | 0/15 | 15/15 |
| **pe cDE** | 15/15 | 15/15 | 0/15 | 0/15 | 15/15 |
| **pe rcGA** | 15/15 | 15/15 | 0/15 | 0/15 | 15/15 |
| **ne rcGA** | 15/15 | 15/15 | 0/15 | 0/15 | 15/15 |
| **pe DEcDE** | 15/15 | 14/15 | 14/15 | 14/15 | 15/15 |
| **cPSO cDEV1** | 15/15 | 15/15 | 0/15 | 0/15 | 15/15 |
| **cWWO** | 15/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **cPSO cDEV2** | 15/15 | 15/15 | 0/15 | 0/15 | 15/15 |
| **DEcDE Hybrid** | 15/15 | 15/15 | 6/15 | 2/15 | 15/15 |
| **ne cDEV2** | 15/15 | 15/15 | 13/15 | 1/15 | 15/15 |
| **pe cDEV2** | 15/15 | 15/15 | 13/15 | 14/15 | 15/15 |
| **Fonctions modérées** | | | | | |
| **Fonctions** / **Algorithme** | $f6$ | $f7$ | $f8$ | $f9$ | |
| **cPSO** | 0/15 | 0/15 | 0/15 | 0/15 | |
| **ne cDE** | 0/15 | 0/15 | 0/15 | 0/15 | |
| **pe cDE** | 0/15 | 0/15 | 0/15 | 0/15 | |
| **pe rcGA** | 0/15 | 0/15 | 0/15 | 0/15 | |
| **ne rcGA** | 0/15 | 0/15 | 0/15 | 0/15 | |
| **pe DEcDE** | 0/15 | 0/15 | 0/15 | 0/15 | |
| **cPSO cDEV1** | 0/15 | 0/15 | 9/15 | 11/15 | |
| **cWWO** | 0/15 | 0/15 | 0/15 | 0/15 | |
| **cPSO cDEV2** | 0/15 | 0/15 | 14/15 | 0/15 | |
| **DEcDE Hybrid** | 0/15 | 0/15 | 0/15 | 0/15 | |
| **ne cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | |
| **pe cDEV2** | 1/15 | 0/15 | 0/15 | 0/15 | |
| **Fonctions mal conditionnées** | | | | | |
| **Fonctions** / **Algorithme** | $f10$ | $f11$ | $f12$ | $f13$ | $f14$ |
| **cPSO** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **ne cDE** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe cDE** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe rcGA** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **ne rcGA** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe DEcDE** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **cPSO cDEV1** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **cWWO** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **cPSO cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **DEcDE Hybrid** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **ne cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |

TABLE 4.5 – Taux de succès dans la dimension 20 pour les fonctions benchmarks de $f1$ à $f14$

| **Fonctions multimodales** | | | | | |
|---|---|---|---|---|---|
| **Fonctions** / **Algorithme** | $f15$ | $f16$ | $f17$ | $f18$ | $f19$ |
| **cPSO** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **ne cDE** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe cDE** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe rcGA** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **ne rcGA** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe DEcDE** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **cPSO cDEV1** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **cWWO** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **cPSO cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **DEcDE Hybrid** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **ne cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **Fonctions multimodales à structure globale faible** | | | | | |
| **Fonctions** / **Algorithme** | $f20$ | $f21$ | $f22$ | $f23$ | $f24$ |
| **cPSO** | 0/15 | 3/15 | 0/15 | 0/15 | 0/15 |
| **ne cDE** | 3/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe cDE** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe rcGA** | 1/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **ne rcGA** | 10/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe DEcDE** | 1/15 | 1/15 | 0/15 | 0/15 | 0/15 |
| **cPSO cDEV1** | 0/15 | 1/15 | 0/15 | 0/15 | 0/15 |
| **cWWO** | 0/15 | 1/15 | 0/15 | 0/15 | 0/15 |
| **cPSO cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **DEcDE Hybrid** | 0/15 | 1/15 | 0/15 | 0/15 | 0/15 |
| **ne cDEV2** | 0/15 | 0/15 | 0/15 | 0/15 | 0/15 |
| **pe cDEV2** | 0/15 | 1/15 | 0/15 | 0/15 | 0/15 |

TABLE 4.6 – Taux de succès dans la dimension 20 pour les fonctions benchmarks de $f15$ à $f24$

# Conclusion Générale

Les problèmes d'optimisation sont et resteront des problèmes difficiles quelles que soient les méthodes algorithmiques mises au point pour les résoudre. Dans tous les cas, il est clair qu'il est irréaliste d'imaginer un algorithme évolutif efficace pour tout type de problème.
Dans ce mémoire, nous avons présenté un travail basé sur des algorithmes compacts. Ils simulent le comportement des algorithmes basés sur la population et utilisent une représentation probabiliste, au lieu d'une population de solutions. Le point fort des algorithmes compacts vient du fait qu'ils ne nécessitent qu'un minimum de mémoire et de capacités de calcul. En principe, ils peuvent être implémentés sur les supports électroniques les plus simples. Pour cela, plusieurs variantes ont été proposées. Une hybridation entre le PSO compact et le DE compact était possible qui a mener à deux variantes différentes cPSO_cDE$v1$, et cPSO_cDE$v2$. Une variante basé sur le DEcDE et utilise le croisement binomial. Nous avons également proposé une nouvelle version compacte d'une méta-heuristique qui n'a jamais été compacté. Nous avons trouvé aussi la meilleur valeur du paramètre $CR$ pour les algorithmes compacts persistant (pe_cDE$v2$) et non persistant (ne_cDE$v2$). Donc en tout, Nous avons proposé six variantes à savoir : cPSO_cDE$v1$, cPSO_cDE$v2$, pe_cDE$v2$, ne_cDE$v2$, DEcDE_$CR_{binomiale}$ et l'algorithme compacté cWWO. Après étude des résultats fournis par l'application des variantes proposées sur BBoB 2015, nous avons constaté que les nouvelles variantes compactes ont donné des performances prometteuses, mais les deux variantes
DEcDE_$CR_{binomiale}$ et le cPSO_cDE$v2$ ont pu fournir des résultats qui surpassent les algorithmes compacts de base.

Il est vrai que les algorithmes proposés ont rencontré des difficultés lorsque la dimension du problème était 20, alors que les résultats dans la dimension 5 étaient forts et prometteurs. Mais le nouvel algorithme compacté cWWO était meilleur que cinq algorithmes de base, même à la dimension 40 voir la figure 4.5. Les résultats obtenus confirme encore la puissance et la robustesse des notre approches.

Enfin, Dans une perspective d'amélioration des performances de l'algorithme compacte résultante, cWWO, on prévoit d'ajouter la technique de la recherche locale. Ceci

peut améliorer considérablement les performance de cette variante. On propose l'introduction d'une autre variante auto-adaptative de DEcDE Hybrid en utilisant les formules dynamique pour les paramètre de contrôle ($F, CR$ et $NP$) et pourquoi pas dans les autres variantes.

# Bibliographie

[1] Mahboub Brahim. Trou noir, nouvelle heuristique d'optimisation, Thèse de master, Université Mohamed Khider de Biskra, 2018.

[2] Amara Kahina et Mehenni Massinissa. Résolution de jeux par les métaheuristiques, Thèse de master, Université A. Mira de Béjaia, 2017.

[3] Lemouari Ali. Cour, Introduction aux Métaheuristiques, Université de Jijel, 2014.

[4] Chettah Khadidja et Maafi Chaima. Techniques d'adaptation de la taille de la population dans l'évolution différentielle sinusoïdale, Thèse de master, Université Abdelhamid Mehri Constantine 2, 2017.

[5] Approche heuristique pour la minimisation des transferts intercellulaires (Handovers) dans les réseaux mobiles, Thèse de master, Université des Sciences et de la Technologie d'Oran Mohamed Boudiaf, 2016.

[6] Troudi Fatiha. Résolution du problème de l'emploi du temps : Proposition d'un algorithme évolutionnaire multi objectif, Thèse de master, Université Mentouri – Constantine, 2006.

[7] Souier Imane et Youbi Fatiha. Selection des variables a base des Métaheuristiques, Thèse de master, Université Abou Bakr Belkaïd de Tlemcen, 2016.

[8] Mahdi Samir. Optimisation multiobjectif par un nouveau schéma de coopération Méta/Exacte, Thèse de master, Université Mentouri de Constantine .

[9] Melle Ferhat Halima. Optimisation de la QoS dans un réseau de radio cognitive en utilisant l'algorithme de la recherche par harmonie, Thèse de master, Université Abou Bakr Belkaid– Tlemcen, 2015.

[10] Marc Sevaux. Métaheuristiques Stratégies pour l'optimisation de la production de biens et de services, Thèse de doctorat, Université de Valenciennes, 2004.

[11] Ilhem Boussaid. Perfectionnement de métaheuristiques pour l'optimisation continue, Thèse de doctorat, Université des sciences et de la téchnologie Houari boumediene, 2013.

[12] Charlie Vanaret. Hybridation d'algorithmes évolutionnaires et de Méthodes d'intervalles pour l'optimisation de problèmes Difficiles, Thèse de doctorat, Université de Toulouse, 2015.

[13] Automated Functional Test Generation for Digital Systems Through a Compact Binary Differential Evolution Algorithm, Alfonso Martinez Cruz, Ricardo Barron Fernandez, Heron Molina Lozano, Marco Antonio Ramırez Salinas, Luis Alfonso Villa Vargas, 361–380, 2015, Springer.

[14] The compact genetic algorithmmm, Harik, Georges R and Lobo, Fernando G and Goldberg, David E, IEEE transactions on evolutionary computation, 3(4) :287–297, 1999, IEEE.

[15] Real-valued compact genetic algorithms for embedded microcontroller optimization, Mininno, Ernesto and Cupertino, Francesco and Naso, David, IEEE Transactions on Evolutionary Computation, 12(2) :32–54, 2011, IEEE.

[16] Compact differential evolution, Mininno, Ernesto and Neri, Ferrante and Cupertino, Francesco and Naso, David, IEEE Transactions on Evolutionary Computation, 15(1) :32–54, 2011, IEEE.

[17] Compact particle swarm optimization, Neri, Ferrante and Mininno, Ernesto and Iacca, Giovanni, Information Sciences, 239, 96–121, 2013, Elsevier.

[18] A new compact teaching-learning-based optimization method, Yang, Zhile and Li, Kang and Guo, Yuanjun, International Conference on Intelligent Computing, 717–726, 2014, Springer.

[19] Mutation-based compact genetic algorithm for spectroscopy variable selection in determining protein concentration in wheat grain, Soares, AS and De Lima, TW and Soares, FAAMN and Coelho, CJ and Federson, FM and Delbem, ACB and Van Baalen, J, Electronics Letters, 50(13) :932–934, 2014, IET.

[20] Modified compact genetic algorithm for thinned array synthesis, Ha BV, Mussetta M, Pirinoli P, Zich RE, IEEE Antennas and Wireless Propagation Letters, 15, 1105–1108 ,2016, IEEE.

[21] Disturbed exploitation compact differential evolution for limited memory optimization problems, Neri, Ferrante and Iacca, Giovanni and Mininno, Ernesto, Information Sciences, 181, 12, 2469–2487, 2011, Elsevier.

[22] Kathleen M. Timmerman, A Hardware Compact Genetic Algorithm for Hover Improvement in an Insect-Scale Flapping-Wing Micro Air Vehicle, Master's thesis, Wright State University, 2012.

[23] Services-Oriented Computing Using the Compact Genetic Algorithm for Solving the Carpool Services Problem,Ming-Kai Jiau, Shih-Chia Huang, IEEE Transactions on intelligent transportation systems, 2015,IEEE.

[24] Memetic Compact Differential Evolution for Cartesian Robot Control, Ferrante Neri, Ernesto Mininno, University of Jyväskylä, FINLAND, IEEE computational intelligence magazine,2010.

[25] Integrated distribution and loading planning via a compact metaheuristic algorithm, Emmanouil E. Zachariadis, Christos D. Tarantilis , Chris T. Kiranoudis,European Journal of Operational Research ,228, 56–71, 2013, Elsevier.

[26] A Compact Articial Bee Colony Optimization for Topology Control Scheme in Wireless Sensor Networks, Thi-Kien Dao, Tien-Szu Pan, Trong-The Nguyen, Journal of Information Hiding and Multimedia Signal Processing, 6, 2, 2015.

[27] Water wave optimization : a new nature inspired metaheuristic,Y.J. Zheng, Computers and Operations Research, 55, 1—11, 2015.

[28] Real-parameter black-box optimization benchmarking 2009 : Presentation of the noiseless functions, Hansen, Nikolaus and Finck, Steffen and Ros, Raymond and Auger, Anne, 2010, Citeseer.