République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي والبحث العلمي
Université Kasdi Merbah - Ouargla
جامعة قاصدي مرباح - ورقلة-

Mémoire de fin d'études

Pour l'obtention du diplôme du Master en Informatique

Option : Informatique Fondamentale

The Archerfish Hunting Optimizer : a new metaheuristic algorithm for global optimization

Réalisé par :

☞ M. Abdelghani Belkeram

☞ M. Lokman Elhakim Baba Hammou

Encadré par :

☞ Dr. Farouq Zitouni

Soutenu le 19 Juin 2021, Devant le jury composé de :

Dr. Khadidja Ameur :	UKMO - Président
Dr. Chahrazad Toumi :	UKMO - Examinateur

Promotion : 2020/2021

Resumé :

L'optimisation globale résout les problèmes du monde réel numériquement ou analytiquement en minimisant leurs fonctions objectives. La plupart des algorithmes analytiques sont gourmands en termes de ressources calculatoires (i.e., temps et mémoire). Les métaheuristiques sont des algorithmes d'optimisation inspirés de la nature. Ils trouvent numériquement une solution quasi-optimale aux problèmes d'optimisation dans un temps raisonnable. Nous proposons un nouvel algorithme métaheuristique pour l'optimisation globale. Il est basé sur les comportements de tir et de saut des poissons archers pour chasser les insectes aériens. Nous l'appelons « Archerfish Hunting Optimizer (AHO) ». Les performances de l'algorithme AHO sont comparées à celles de 12 récents algorithmes métaheuristiques en utilisant dix fonctions de test du benchmark CEC 2020 pour l'optimisation sans contraintes. Les résultats numériques sont évalués en utilisant les tests de Wilcoxon et Friedman. Les indicateurs statistiques montrent que l'algorithme AHO a une excellente capacité à atteindre des performances supérieures en concurrence avec les optimiseurs utilisés.

Mots clés : Optimisation globale, algorithmes métaheuristiques, optimisation sans contrainte, comportement de chasse des poissons archers, Benchmark CEC 2020.

Abstract :

Global optimization solves real-world problems numerically or analytically by minimizing their objective functions. Most of the analytical algorithms are greedy and computationally intractable. Metaheuristics are nature-inspired optimization algorithms. They numerically find near-optimal solutions for optimization problems in a reasonable amount of time. We propose a novel metaheuristic algorithm for global optimization. It is based on the shooting and jumping behaviors of the archerfish for hunting aerial insects. We name it the Archerfish Hunting Optimizer (AHO). AHO's performance is compared to 12 recent metaheuristic algorithms (the accepted algorithms for the 2020's competition on single objective bound-constrained numerical optimization) on ten test functions of the benchmark CEC 2020 for unconstrained optimization. The experimental results are evaluated using the Wilcoxon signed-rank and the Friedman tests. The statistical indicators illustrate that the Archerfish Hunting Optimizer has an excellent ability to accomplish higher performance in competition with the well-established optimizers.

Keywords : Global optimization, Metaheuristic algorithms, Unconstrained optimization, Hunting behavior of archerfish, Benchmark CEC 2020.

ملخص

خوارزميات البحث عن الحلول المثالية الكلية تحل المشاكل التطبيقية إما عدديًا أو تحليليًا عن طريق تصغير قيم الدوال الذاتية. حيث أن معظم الخوارزميات التحليلية معقدة جدا وشرهة مثال (الوقت و الذاكرة).

الخوارزميات الهروستيكية هي طرق مستوحاة من الطبيعة للبحث عن الحلول المثالية الكلية . تتميز عائلة الخوارزميات الهروستيكية بحل مشاكل البحث عن الحلول المثالية الكلية بطريقة شبه مثالية وفي وقت معقول .

وفي هذه المذكرة نقترح خوارزمية هروستيكية جديدة للبحث عن الحلول المثالية الكلية ،هذه الخوارزمية مستوحاة من طريقة صيد الخاصة بسمكة القوس للحشرات الطائرة (القفز والرماية)، لقد قمنا بمقارنة أداء الخوارزمية المقترحة مع 12 خوارزمية هورستيكية جديدة (الخوارزميات الهورستيكية المقبول لمنافسة CEC 2020) ،وقد قمنا بتحليل النتائج العددية المتحصل عليها باستعمال فحصين إحصائيين (فرد مال و ولتكسون) .حيث أثبتت المؤشرات الإحصائية أن الخوارزمية الهروستيكية المقترحة هي جد تنافسية بالمقارنة مع الخوارزميات الأخرى.

كلمات مفتاحية

خوارزميات البحث عن الحلول المثالية الكلية ، الخوارزميات الهروستيكية ،البحث عن الحلول المثالية الغير مقيدة، سلوك صيد سمكة القوس، بنش مارك CEC 2020

Dédicace

Je remercie tout d'abord le bon Dieu le tout puissant qui m'a donné la force et le courage pour terminer ce travail.

Je dédie ce modeste travail à mon professeur et mon frère **Dr. Farouq Zitouni** et mon ex-directeur **M. Azzazen Mohamed**.

Je dédie ce modeste travail à ma précieuse mère et mon cher père.

Je dédie ce modeste travail à ma femme Hassini A.

Je dédie ce modeste travail à mes enfants ROUCHDI, ROUA, RAID et RAOUANE.

Je dédie ce modeste travail à toute la famille BELKERAM et HASSINI.

Je dédie ce modeste travail à tous mes collègues et professeurs, promotion 2021.

Merci

Abdelghani Belkeram

Dédicace

Je dédie cet ouvrage à mes chers parents, pour tous leurs sacrifices, leur amour, leur tendresse, leur soutien et leurs prières tout au long de mes études.

Je dédie cet ouvrage à mon professeur **Dr. Farouq Zitouni**.

Je dédie cet ouvrage à mes chers frères, pour leur appui et leur encouragement.

Je dédie cet ouvrage à toute ma famille pour leur soutien tout au long de mon parcours universitaire.

Je dédie cet ouvrage à tous mes amis qui m'ont toujours encouragé, et à qui je souhaite plus de succès.

Je dédie cet ouvrage à tous ceux que j'aime.

Merci

Lokman Elhakim Baba Hammou

Remerciements

Tout d'abord, nous remercions Allah le tout puissant de nous avoir donné le courage et la patience nécessaires pour mener ce travail à son terme.

Nous tenons à remercier tout particulièrement notre encadrant **Dr. Farouq Zitouni**, pour l'aide compétente qu'il nous a apportée, pour sa patience et son encouragement. Son œil critique nous a été très précieux pour structurer le travail et pour améliorer la qualité des différentes sections.

Nous remercions spécialement le professeur **Saad Harous** (Professeur au Faculté des sciences des technologies de l'information de l'Université des Émirats arabes unis).

Que les membres de jury trouvent, ici, l'expression de nos sincères remerciements pour l'honneur qu'ils nous font en prenant le temps de lire et d'évaluer ce travail.

Nous souhaitons aussi remercier les équipes pédagogiques et administratives du département d'informatique de l'université de Ouargla pour leurs efforts dans le but de nos offrir une excellente formation.

Pour finir, nous remercions toute personne ayant contribué de prés ou de loin à la réalisation de ce mémoire.

Liste des acronymes

AHO	:	Archerfish Hunting Optimizer
IA	:	Intelligence Artificielle
IC	:	Intelligence Computationnelle
OG	:	Optimisation Globale
NFL	:	No-Free-Lunch
AG	:	Algorithmes Génétiques
SE	:	Stratégie Evolution
RNA	:	Réseaux de Neurones Artificiels
SVM	:	Machines à Vecteurs de Support
UM	:	UniModale
BC	:	Basique
HD	:	Hybride
CM	:	Composite
STD	:	valeurs de l'écart-type
SPSS	:	Statistical Package for the Social Sciences

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Introduction générale

Problématique et motivations

L'intelligence artificielle (IA) est une discipline qui a pour objectif sublime la conception de machines intelligentes. La recherche est un concept clé en IA, car elle sert toutes les autres disciplines (e.g., résolution de problèmes). En général, l'espace de recherche couvert par les problèmes pratiques est considérable. Ainsi, la possibilité d'énumérer toutes les solutions réalisables et en déterminer l'optimalité n'est pas un choix judicieux. Donc, l'utilisation des méthodes traditionnelles basées sur le calcul et l'énumération est exclue. Par conséquent, des paradigmes d'intelligence computationnelle (IC) ont été initiés pour cette fin.

L'optimisation est le processus de recherche des solutions optimales à un problème donné. Les trois approches de recherche sont : (i) analytique ; (ii) énumérative ; et (iii) heuristique. La recherche analytique est basée sur le calcul des dérivées, jacobiennes et hessiennes ; et donne des solutions optimales. La recherche énumérative établit une liste exhaustive de toutes les solutions réalisables ; puis retourne les solutions optimales. La recherche heuristique suit des chemins aléatoires guidés ; et retourne des solutions approchées.

L'IC est une branche de l'IA, qui étudie les mécanismes d'adaptation et comportements intelligents dans les environnements complexes. Contrairement à l'IA, qui s'appuie sur des connaissances provenant de l'expertise humaine, l'IC s'appuie sur des données numériques. L'IC comprend un ensemble de paradigmes de calcul inspirés par la nature, à savoir : (i) réseaux de neurones artificiels pour la reconnaissance de formes ; (ii) systèmes flous pour le raisonnement incertain ; et (iii) calculs évolutionnaires pour l'optimisation stochastique.

Depuis la nuit des temps, la nature était toujours la principale source d'inspiration pour de nouveaux paradigmes de calcul. Par exemple, la cybernétique de Wiener [1] s'est inspirée du processus de contrôle par rétroaction des systèmes biologiques. Les changements dans la nature (de l'échelle microscopique à l'échelle écologique) peuvent être vus comme des calculs. Les processus naturels atteignent toujours un équilibre optimal. De telles analogies peuvent être utilisées pour trouver des solutions aux problèmes de recherche et optimisation : e.g., réseaux de neurones artificiels [2] ; recuit simulé [3] ; algorithmes génétiques [4] ; systèmes immunitaires artificiels [5] ; calcul moléculaire [6] ; informatique quantique [7] ; informatique membranaire [8] ; et automates cellulaires [9].

Les algorithmes d'approximation ont été introduits dans les années 60, pour générer des solutions approchées aux problèmes d'optimisation, qui ne pouvaient pas être résolus efficacement par les méthodes dites exactes [10-13]. Plus tard, avec l'apparition de la $\mathscr{NP}$-complétude comme théorie dans les années 70, les algorithmes d'approximation ont pu gagner une importance prestigieuse dans la communauté scientifique, car le besoin de générer des solutions approchées aux problèmes d'optimisation $\mathscr{NP}$-difficiles est devenu la seule voie pour traiter leurs complexités computationnelles. Les algorithmes d'approximation, basés sur la randomisation, sont devenus populaires dans les années 80, où l'introduction de nouvelles techniques pour résoudre les problèmes de programmation linéaire a déclenché une nouvelle vague d'algorithmes d'approxi-

mation qui ont mûri et connu une croissance énorme dans les années 90. Les algorithmes d'approximation basés sur la randomisation s'appellent communément métaheuristiques. Au début de l'histoire, certaines métaheuristiques ont été introduites dans les années 80 et 90, afin de résoudre spécifiquement quelques problèmes non-approximables. Parmi ces métaheuristiques, nous pouvons citer : le recuit simulé [3], l'optimisation par colonie de fourmis [14], les algorithmes évolutionnaires [15, 16], la recherche taboue [17], etc. Au cours des trois dernières décennies, plusieurs métaheuristiques ont été proposées dans la littérature ; où ces dernières ont été expérimentalement évaluées et ont montré leur pouvoir de résolution des problèmes pratiques en ingénierie et industrie.

De nos jours, les métaheuristiques possèdent une stature comparable à celle des algorithmes exacts en optimisation (les métaheuristiques se sont imposées comme un domaine de recherche à part entière). Pour un problème d'optimisation, le but des métaheuristiques est de fournir les meilleures solutions possibles qui répondent à certains critères.

Des bactéries aux humains, toutes les entités biologiques sont liées par des interactions sociales. Les métaheuristiques imitent le comportement collectif des populations biologiques. Ainsi, la résolution coopérative de problèmes est une approche qui utilise la coopération d'un groupe d'entités autonomes, pour atteindre un objectif donné. Les mécanismes de coopération sont communs en informatique basée-agents.

Contribution

La contribution de ce mémoire s'inclut dans le cadre de l'optimisation globale (OG). L'OG est un domaine de recherche multidisciplinaire : i.e., croisement entre mathématiques appliquées et analyse numérique. L'OG a pour objectif le calcul des optimums globaux d'une fonction non-convexe dans une certaine région réalisable. Les problèmes d'OG (issus du monde réel) sont généralement assez difficiles à résoudre par les méthodes exactes, car la majorité des problèmes sont $\mathscr{NP}$-difficiles. Par conséquent, trouver les solutions optimales à de tels problèmes est un vrai défi.

Organisation

Les chapitres de la présente dissertation peuvent être divisés en deux parties. Les chapitres1 et 2 sont consacrés à la description et présentation des concepts fondamentaux de l'OG, afin d'aider le lecteur à comprendre la problématique du manuscrit. Le chapitre3 est complètement dévoué à l'explication détaillée de l'algorithme métaheuristique que nous avons proposé. Bien évidemment, une conclusion générale et quelques perspectives sont données à la fin, pour : faire la synthèse des choses ; citer les différentes difficultés rencontrées ; et donner quelques idées pour des travaux futurs. Les chapitres se résument comme suit :

1. Le chapitre1 vous aiderait à comprendre le concept d'algorithmes d'optimisation. D'abord, nous montrons la structure générale d'un algorithme d'approximation ; et donnons quelques méthodes de résolution. Ensuite, nous expliquons les problèmes d'optimisation ; et mettons l'accent sur leurs caractéristiques et propriétés. Enfin, nous discutons brièvement les algorithmes inspirés par la nature (i.e., métaheuristiques) ; et décrivons leurs composantes principales.
2. Le chapitre2 vous aiderait à analyser proprement les métaheuristiques. D'abord, nous donnons un aperçu sur les différents paradigmes, qui peuvent être utilisés pour modéliser les métaheuristiques. Ensuite, nous élaborons une analyse profonde de quelques algorithmes populaires, dans le but de doter le lecteur des outils nécessaires à la critique objective de n'importe quelle métaheuristique. Enfin, nous parlons de réglage et contrôle de paramètres des métaheuristiques, et mettons la lumière sur l'importance crucial de cette étape.

3. Le chapitre3 présente l'algorithme pour l'optimisation globale que nous avons proposé. Cette métaheuristique a été baptisée « Archerfish Hunting Optimizer (AHO) ». En fait, l'algorithme AHO est inspiré par les comportements de tir et de saut des poissons archerfish lors de la capture des insectes aériens. Nous avons comparé les performances de l'algorithme AHO avec 12 métaheuristiques récentes sur dix fonctions de test du benchmark CEC 2020 pour l'optimisation sans contraintes. Les résultats expérimentaux sont examinés en utilisant les tests de Wilcoxon et de Friedman. Les indicateurs statistiques illustrent que l'algorithme AHO est très compétitif par rapport aux algorithmes utilisés pour l'étude comparative.

Chapitre 1

Algorithmes d'optimisation

Sommaire

1.1 Introduction

L'optimisation est un composant clé dans de nombreuses applications issues de l'ingénierie et industrie [18-23]. Les objectifs de l'optimisation sont divers : e.g., minimiser les consommations d'énergie [24-27] ou maximiser les performances [28, 29]. Il n'est pas exagéré de dire que l'optimisation est omniprésente : e.g., conception [30, 31], planification [32, 33], routage [34, 35], etc. Etant donné que les ressources, i.e., temps et argent, sont toujours limités dans les applications issues du monde réel, alors le besoin de développer des solutions pour les utiliser de manière optimale, sous diverses contraintes, s'avère inéluctable. L'optimisation est l'étude de ces problèmes et solutions à l'aide d'outils mathématiques [36]. Étant donné que la majorité des problèmes sont non-linéaires, alors il est indispensable d'utiliser des outils d'optimisation sophistiqués pour y faire face. De nos jours, la simulation informatique est devenue un outil crucial de résolution de tels problèmes d'optimisation [37].

Ce chapitre présente les algorithmes d'optimisation. Nous donnons la formulation générale et caractéristiques d'un problème d'optimisation, et décrivons quelques approches de résolution. Un bref historique des métaheuristiques est donné à la fin du chapitre.

1.2 Qu'est-ce qu'un algorithme ?

Un algorithme est un processus séquentiel et itératif pour fournir des calculs et résultats [38, 39]. Les étapes d'un algorithme dépendent du problème en question. Nous nous intéressons particulièrement aux algorithmes d'optimisation, et nous mettons l'accent sur les procédures itératives de construction d'algorithmes. Par exemple, un algorithme du calcul de la racine carrée d'un nombre positif k peut être donné par l'équation 1.1.

$$x_{t+1} = \frac{1}{2}(x_t + \frac{k}{x_t}) \tag{1.1}$$

nous commençons par une valeur initiale $x_0 \neq 0$. L'indice t est appelé : pseudo-temps ou compteur de générations. Cette équation itérative provient du réarrangement de $x^2 = k$ sous la forme suivante :

$$\frac{x}{2} = \frac{k}{2x} \Rightarrow x = \frac{1}{2}(x + \frac{k}{x}) \tag{1.2}$$

pour $k = 7$ et avec $x_0 = 1$, nous avons les itérations suivantes :

$$\begin{aligned} &x_1 = \tfrac{1}{2}(x_0 + \tfrac{7}{x_0}) = 4 \quad , \quad x_2 = \tfrac{1}{2}(x_1 + \tfrac{7}{x_1}) = 2.875 \\ &x_3 = \tfrac{1}{2}(x_2 + \tfrac{7}{x_2}) \approx 2.654891304 \quad , \quad x_4 = \tfrac{1}{2}(x_3 + \tfrac{7}{x_3}) \approx 2.645767044 \\ &x_5 = \tfrac{1}{2}(x_4 + \tfrac{7}{x_4}) \approx 2.6457513111 \quad , \quad \ldots \end{aligned} \tag{1.3}$$

Nous observons que la valeur de x_5 est très proche de la vraie valeur de $\sqrt{7}$, ce qui montre l'efficacité de cet algorithme d'optimisation. La raison de bon fonctionnement de cet algorithme est que la série $x_1, x_2, \ldots, x_t$ converge vers $\sqrt{k}$ quand $t \to \infty$. Un bon choix de la valeur initiale x_0 accélérera la convergence. Par contre, un mauvais choix de x_0 pourrait faire échouer l'algorithme d'optimisation. Par exemple, nous ne pouvons pas utiliser $x_0 \leq 0$ car $\sqrt{k} > 0$.

Au début, si $x_0^2 < k$, alors x_0 est la borne inférieure et k/x_0 est la borne supérieure. Si $x_0^2 > k$, alors x_0 est la borne supérieure et k/x_0 est la borne inférieure. Pour le reste des itérations, les nouvelles bornes seront x_t et k/x_t. En fait, la valeur de x_{t+1} est toujours entre les deux bornes x_t et k/x_t, et la nouvelle estimation de x_{t+1} est donc la moyenne des deux bornes. Ceci garantirait que la série converge vers la vraie valeur de $\sqrt{k}$. Cette méthode est similaire à la méthode de bissection [40, 41]. Il convient de noter que le résultat

final peut grandement dépendre de la valeur initiale. Il s'agit d'une caractéristique et d'un inconvénient des algorithmes déterministes.

D'un autre côté, nous pouvons nous demander pourquoi $x^2 = k$ est converti en équation 1.1 ? Pourquoi ne pas écrire la formule itérative simplement comme suit :

$$x_{t+1} = \frac{k}{x_t} \tag{1.4}$$

pour $k = 7$ et $x_0 = 1$, nous avons les itérations suivantes :

$$\begin{aligned} x_1 &= \tfrac{7}{x_0} = 7 \quad , \quad x_2 = \tfrac{7}{x_1} = 1 \\ x_3 &= \tfrac{7}{x_2} = 7 \quad , \quad x_4 = \tfrac{7}{x_3} = 1 \\ x_5 &= \tfrac{7}{x_4} = 7 \quad , \quad \dots \end{aligned} \tag{1.5}$$

ce qui donne une fonction oscillante à deux valeurs distinctes : 1 et 7. Nous pouvons nous demander que ce pourrait être le problème de la valeur initiale de x_0. En fait, pour toute valeur initiale de $x_0 \neq 0$, cette formule entraînera des oscillations entre deux valeurs : x_0 et k. Ce qui démontre que la façon de concevoir la formule itérative est très importante.

Pour un problème donné, si un algorithme A génère une nouvelle solution x_{t+1} à partir d'une solution actuelle x_t, alors nous exprimons ça mathmatiquement par l'équation 1.6.

$$x_{t+1} = \alpha f_A(x_t) + \beta \tag{1.6}$$

où f_A est une fonction mathématique de x_t ; α et β sont deux scalaires.

Les formes d'index supérieur $x^{(t+1)}$ ou x^{t+1} ne signifient pas que x est levé à la puissance de $(t+1)$. Dans le reste de la thèse, les écritures suivantes sont équivalentes : $x_{t+1} \equiv x^{(t+1)} \equiv x^{t+1}$ et $x_t \equiv x^{(t)} \equiv x^t$.

1.3 Méthode de Newton

La méthode de Newton [42, 43] est utilisée pour trouver les zéros d'une fonction non-linéaire $f(x) = 0$, sur un intervalle $[a,b]$. Elle a été formulée par Newton en 1669, et après Raphson a appliqué cette idée aux polynômes en 1690. Cette méthode est également appelée méthode de Newton-Raphson.

En tout point x_t, nous développons la fonction $f(x)$ par une série de Taylor [44] pour $\Delta x = x_{t+1} - x_t$, ce qui donne :

$$f(x_{t+1}) = f(x_t + \Delta x) \approx f(x_t) + f'(x_t)\Delta x \tag{1.7}$$

ce qui conduit à :

$$x_{t+1} - x_t = \Delta x \approx \frac{f(x_{t+1}) - f(x_t)}{f'(x_t)} \tag{1.8}$$

enfin, nous avons :

$$x_{t+1} \approx x_t + \frac{f(x_{t+1}) - f(x_t)}{f'(x_t)} \tag{1.9}$$

Puisque nous essayons de trouver une approximation de $f(x) = 0$ avec $f(x_{t+1})$, nous pouvons utiliser l'approximation $f(x_{t+1}) \approx 0$ dans l'équation 1.9. Donc, nous obtenons la formule itérative standard de Newton.

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)} \tag{1.10}$$

La procédure d'itération commence par une valeur initiale x_0, et se poursuit jusqu'à ce qu'une certaine condition soit satisfaite. Une bonne valeur initiale conduirait à un nombre petit d'itérations ; cependant, s'il n'y a pas une bonne valeur initiale, alors tout point de l'intervalle $[a,b]$ peut être utilisé comme point de départ. Lorsque la valeur initiale est trop éloignée du vrai zéro, alors le processus d'itération peut échouer (une bonne idée consiste à limiter le nombre d'itérations). La méthode de Newton-Raphson est très efficace et largement utilisée.

Pour les équations non-linéaires, il existe souvent plusieurs racines, et le choix de la valeur initiale peut affecter la racine vers laquelle la procédure itérative converge. Il arrive parfois que la procédure itérative diverge pour une certaine valeur initiale.

Par exemple, nous savons que l'équation non-linéaire donnée par l'équation 1.11 possède deux racines : $x_1^* = 0$ et $x_2^* = 2,718281828459$.

$$x^x = e^x \, , x \in [0, \infty[\tag{1.11}$$

maintenant, nous essayons de résoudre l'équation 1.11 en utilisant la méthode de Newton-Raphson. Tout d'abord, nous la réécrivons par l'équation 1.12 :

$$f(x) = x^x - e^x = 0 \tag{1.12}$$

nous avons $f'(x) = x^x(\ln x + 1) - e^x$. Si nous prenons $x_0 = 5$, alors nous avons les itérations suivantes :

$$\begin{array}{llll} x_1 = 4.6282092 & x_2 = 5.2543539 & x_3 \approx 3.8841063 & \dots \\ x_7 \approx 2.7819589 & \dots & x_{10} \approx 2.7182818 & \dots \end{array} \tag{1.13}$$

La solution x_{10} est très proche de la vraie solution x_2^*. Cependant, si nous partons de $x_0 = 10$ comme valeur initiale, il faudra environ 25 itérations pour obtenir $x_{25} \approx 2,7182819$ (la convergence est très lente). En revanche, si nous considérons $x_0 = 1$, alors nous obtiendrons $x_1 = 0$, qui est la solution exacte de l'autre racine $x^* = 0$. De plus, si nous partons d'une valeur initiale $x_0 \leq 0$, alors la méthode de Newton-Raphson n'est pas applicable, en raison de la singularité des logarithmes. D'un autre côté, si nous partons de n'importe quelle valeur de l'intervalle $[0.01, 0.99]$, alors la méthode ne fonctionnera pas aussi. Cela montre l'importance de choisir la bonne valeur initiale. Il est important de souligner que le taux de convergence devient très lent au voisinage du vrai zéro.

Par ailleurs, la méthode de Newton-Raphson peut être étendue pour trouver l'optimum d'une fonction $f(\mathbf{x})$, ce qui est équivalent à trouver les racines de $f'(\mathbf{x}) = 0$ dans un espace à d dimensions. Ici $\mathbf{x} = (x_1, x_2, \dots, x_d)^T$ est un vecteur de d variables (l'exposant T signifie la transposition). Cette notation facilite l'extension des fonctions à une seule variable aux fonctions à plusieurs variables, car la forme est identique et la seule différence est de convertir un scalaire x en vecteur $\mathbf{x}$ (en gras maintenant pour éviter toute confusion).

1.4 Problèmes d'optimisation

Généralement, il est possible d'écrire la plupart des problèmes d'optimisation sous la forme générique donnée par les équations 1.14, 1.15 et 1.16.

$$\min_{\mathbf{x} \in R^d} f_m(\mathbf{x}) \, , m = \{1, 2, \dots, M\} \tag{1.14}$$

sous les contraintes :

$$h_j(\mathbf{x}) = 0 \, , j = \{1, 2, \dots, J\} \tag{1.15}$$

$$g_k(\mathbf{x}) \leq 0 \, , k = \{1, 2, \dots, K\} \tag{1.16}$$

où $f_m(\mathbf{x})$, $h_j(\mathbf{x})$ et $g_k(\mathbf{x})$ sont des fonctions du vecteur

$$\mathbf{x} = (x_1, x_2, \ldots, x_d)^T \tag{1.17}$$

Les composantes x_i de $\mathbf{x}$ sont appelées variables de décision, et elles peuvent être : continues, discrètes ou mixtes. Les fonctions $f_m(\mathbf{x})$, où $m = \{1,2,\ldots,M\}$, sont appelées fonctions objectives. Dans le cas où $M = 1$, alors il n'y a qu'une seule fonction objective. L'espace couvert par les variables de décision est appelé espace de recherche R^d ; tandis que l'espace formé par les valeurs des fonctions objectives est appelé espace de solutions. Les égalités $h_j = 0$, où $j = \{1,2,\ldots,J\}$, et les inégalités $g_k \leq 0$, où $k = \{1,2,\ldots,K\}$, sont appelées contraintes. Il convient de noter que nous pouvons également écrire les inégalités dans l'autre sens (i.e., ≥ 0), et nous pouvons également formuler les fonctions objectives comme un problème de maximisation. Dans des cas extrêmement rares où il n'y a pas des fonctions objectives et il n'y a que des contraintes, un tel problème est appelé problème de faisabilité et toute solution réalisable est optimale.

Si nous classons les problèmes d'optimisation suivant le nombre de fonctions objectives, alors il existe principalement deux catégories : i) problèmes mono-objective (i.e., $M = 1$) et ii) problèmes multi-objectives (i.e., $M > 1$). L'optimisation multi-objectives est également appelée optimisation multi-critères ou multi-attributs. Dans la pratique, la majorité des problèmes d'optimisation sont multi-objectives. De même, nous pouvons classer les problèmes d'optimisation suivant le nombre de contraintes (i.e., $J+K$). S'il n'y a aucune contrainte du tout, (i.e., $J = K = 0$), alors nous avons un problème d'optimisation sans contraintes. Si $K = 0$ et $J \geq 1$, alors nous avons un problème d'optimisation avec contraintes d'égalité. Si $J = 0$ et $K \geq 1$, alors nous avons un problème d'optimisation avec contraintes d'inégalité.

Il convient de souligner que, dans certaines formulations de problèmes d'optimisation, les contraintes d'égalités ne sont pas explicitement incluses et seules les contraintes d'inégalités sont considérées. En fait, une contrainte d'égalité peut être écrite comme deux contraintes d'inégalités : i.e., $h(\mathbf{x}) = 0 \equiv (h(\mathbf{x}) \leq 0) \wedge (h(\mathbf{x}) \geq 0)$. Les contraintes d'égalité possèdent des propriétés spéciales et requièrent une attention particulière. Lors de la résolution d'un problème d'optimisation, il est très difficile d'avoir des observations, de l'espace de recherche, qui satisfont exactement les contraintes d'égalité ; donc, une certaine tolérance est considérée pour contourner ces limitations.

Nous pouvons également utiliser les fonctions pour classer les problèmes d'optimisation. Les fonctions objectives f_m peuvent être linéaires ou non-linéaires. Si les contraintes h_j et g_k sont toutes linéaires, alors nous avons un problème d'optimisation avec contraintes linéaires. Si les fonctions objectives et contraintes sont toutes linéaires, alors nous avons un problème de programmation linéaire. Si tous les f_m, h_j et g_k sont non-linéaires, alors nous avons un problème d'optimisation non-linéaire. Nous distinguons principalement deux familles de méthodes d'optimisation : optimisation descendante et optimisation ascendante.

1.4.1 Optimisation descendante

La méthode de Newton-Raphson présentée ci-dessus concerne les fonctions à une seule variable. Maintenant, nous l'étendons aux fonctions à plusieurs variables. Pour toute fonction continue et différenciable $f(\mathbf{x})$ à optimiser, nous avons la forme quadratique de développement de Taylor [45] au voisinage d'un point connu $\mathbf{x} = \mathbf{x}_t$ et $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}_t$:

$$f(\mathbf{x}) = f(\mathbf{x}_t) + (\nabla f(\mathbf{x}_t))^T \Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T \nabla^2 f(\mathbf{x}_t)\Delta\mathbf{x} + \ldots \tag{1.18}$$

La fonction $f(\mathbf{x})$ est minimisée au voisinage d'un point critique lorsque $\Delta\mathbf{x}$ est la solution de l'équation linéaire 1.19.

$$\nabla f(\mathbf{x}_t) + \nabla^2 f(\mathbf{x}_t)\Delta\mathbf{x} = 0 \tag{1.19}$$

ce qui donne

$$\mathbf{x} = \mathbf{x}_t - H^{-1}\nabla f(\mathbf{x}_t) \tag{1.20}$$

où $H = \nabla^2 f(\mathbf{x}_t)$ est la matrice de Hesse [46], et qui est définie par l'équation 1.21.

$$H(\mathbf{x}) \equiv \nabla^2 f(\mathbf{x}) \equiv \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_d} & \cdots & \frac{\partial^2 f}{\partial x_d \partial x_d} \end{pmatrix} \tag{1.21}$$

cette matrice est symétrique du fait que :

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i}$$

Si la procédure d'itération commence par un vecteur initial $\mathbf{x}^{(0)}$, alors la formule itérative de Newton-Raphson peut être écrite comme suit :

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - H^{-1}(\mathbf{x}^{(t)}) f(\mathbf{x}^{(t)}) \tag{1.22}$$

Où $H^{-1}(\mathbf{x}^{(t)})$ est l'inverse de la matrice de Hesse. Il convient de souligner que si $f(\mathbf{x})$ est quadratique, alors la solution peut être trouvée exactement en une seule étape. Cependant, cette méthode n'est pas efficace pour les fonctions non-quadratiques.

Pour accélérer la convergence, nous pouvons utiliser une variable $\alpha \in [0,1]$, et nous modifions l'équation 1.22 comme suit :

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha H^{-1}(\mathbf{x}^{(t)}) f(\mathbf{x}^{(t)}) \tag{1.23}$$

Parfois, le calcul de la matrice de Hesse peut être coûteux. De coup, nous avons une bonne alternative qui est l'utilisation de la matrice d'identité : i.e., nous posons $H = I$ pour que $H^{-1} = I$. Maintenant, nous modifions l'équation 1.23 comme suit :

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha I \nabla f(\mathbf{x}^{(t)}) \tag{1.24}$$

cette méthode est appelée : méthode de descente et elle est généralement beaucoup plus rapide [47, 48]. L'objectif de la méthode de descente est de trouver la valeur minimale de $f(\mathbf{x})$ pour un vecteur $\mathbf{x}^{(t)}$.

Du développement de Taylor de $f(\mathbf{x})$ au voisinage de $\mathbf{x}^{(t)}$ nous avons :

$$f(\mathbf{x}^{(t+1)}) = f(\mathbf{x}^{(t)} + \Delta\mathbf{s}) \approx f(\mathbf{x}^{(t)} + (\nabla f(\mathbf{x}^{(t)}))^T)\Delta\mathbf{s} \tag{1.25}$$

où $\Delta\mathbf{s} = \mathbf{x}^{(t+1)} - \mathbf{x}^{(t)}$ est le vecteur d'incrément. Puisque nous essayons de trouver une meilleure approximation de la fonction objective, alors il est requis que le deuxième terme du côté droit soit négatif.

$$f(\mathbf{x}^{(t)} + \Delta\mathbf{s}) - f(\mathbf{x}^{(t)}) = (\nabla f)^T \Delta\mathbf{s} < 0 \tag{1.26}$$

Nous savons que le produit scalaire $\mathbf{u}^T\mathbf{v}$ des vecteurs $\mathbf{u}$ et $\mathbf{v}$ est à sa valeur maximale lorsque $\mathbf{u}$ et $\mathbf{v}$ sont parallèles et dans des directions opposées. Par conséquent, nous avons :

$$\Delta\mathbf{s} = -\alpha \nabla f(\mathbf{x}^{(t)}) \tag{1.27}$$

où $\alpha > 0$ est la longueur du pas.

Le choix de α est très important. Une très petite valeur de pas signifie un mouvement lent vers le minimum local (i.e., convergence lente) ; tandis qu'une grande valeur de pas peut causer le contournement du minimum local, et par conséquent nous nous y éloignons. Une bonne alternative consiste à faire varier la

valeur de α à chaque itération, et cette valeur doit être choisie de manière à minimiser la fonction objective $f(\mathbf{x}^{(t+1)}) = f(\mathbf{x}^{(t)}, \alpha^{(t)})$. Donc, nous modifions l'équation 1.24 comme suit :

$$f(\mathbf{x}^{(t+1)}) = f(\mathbf{x}^{(t)}) - \alpha^{(t)}(\nabla f(\mathbf{x}^{(t)}))^T \nabla f(\mathbf{x}^{(t)}) \tag{1.28}$$

À chaque itération, le gradient et la longueur de pas sont calculés. Une bonne estimation de $\mathbf{x}^{(0)}$ et $\alpha^{(0)}$ est très importante. Nous donnons un exemple illustratif. Nous considérons à minimiser la fonction à deux variables donnée par l'équation 1.29.

$$f(x_1, x_2) = 10x_1^2 + 5x_1x_2 + 10(x_2 - 3)^2 \tag{1.29}$$

où $(x_1, x_2) \in [-10, 10] \times [-15, 15]$.

Nous utilisons la méthode de descente décrite ci-dessus. Nous prenons comme vecteur initial $\mathbf{x}^{(0)} = (10, 15)^T$. Nous savons que le gradient est :

$$\nabla f = (20x_1 + 5x_2, 5x_1 + 20x_2 - 60)^T$$

par conséquent, $\nabla f(\mathbf{x}^{(0)}) = (275, 290)^T$. Dans la première itération, nous avons :

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \alpha^{(0)}(275, 290)^T$$

La valeur de $\alpha^{(0)}$ doit être choisie de telle sorte que $f(\mathbf{x}^{(1)})$ soit minimisée, ce qui signifie que :

$$f(\alpha^{(0)}) = 10(10 - 275\alpha^{(0)})^2 + 5(10 - 275\alpha^{(0)})(15 - 290\alpha^{(0)}) + 10(12 - 290\alpha^{(0)})^2$$

devrait être minimisé. Nous obtenons un problème d'optimisation à une seule variable $\alpha^{(0)}$. Nous utilisons la méthode de Newton-Raphson pour trouver $\alpha^{(0)} \approx 0.04001$. Ensuite, les itérations se poursuivent jusqu'à ce qu'une certaine condition soit satisfaite. Après trois itérations, nous obtenons la solution approchée $\mathbf{x}^{(3)} \approx (-0.8000299, 3.20029)^T$, qui est très proche de la solution exacte $\mathbf{x}^* = (-0.8, 3.2)^T$.

Nous remarquons que la méthode de descente nous donne presque la solution exacte après seulement trois itérations. Pour trouver la valeur de $\alpha^{(t)}$, nous résolvons simplement l'équation $df(\alpha^{(t)})/d\alpha^{(t)} = 0$. Dans une telle situation, nous pouvons dire : si nous pouvons utiliser cette équation pour obtenir la valeur de $\alpha^{(t)}$, pourquoi ne pas l'utiliser pour obtenir le minimum de $f(\mathbf{x})$ en premier lieu ? Là nous avons deux raisons : (i) il s'agit d'un exemple simple pour montrer comment utiliser la méthode de descente ; (ii) et pour des fonctions à plusieurs variables $f(x_1, \ldots, x_d)$, $f(\alpha^{(t)})$ à n'importe quelle étape t est toujours une fonction à une seule variable, et son optimisation est beaucoup plus simple en comparaison avec $f(x_1, \ldots, x_d)$. De plus, la valeur de $\alpha^{(t)}$ peut être obtenue en utilisant un algorithme d'optimisation simple et efficace.

Il convient de souligner que plus nous approchons du minimum local, plus la convergence de la méthode de descente est lente. La raison de ce comportement est qu'au voisinage du minimum local la valeur du gradient est presque nulle. Aussi, nous rajoutons que le minimum local dans certains problèmes d'optimisation est inexistant.

1.4.2 Optimisation ascendante avec redémarrage aléatoire

Les problèmes abordés dans les sections précédentes sont relativement simples. Parfois, même des problèmes qui apparaissent simples peuvent être difficiles à résoudre. Par exemple, la fonction suivante :

$$f(x, y) = (x - y)^2 e^{(-x^2 - y^2)} \tag{1.30}$$

possèdent deux maximums globaux à $(1/\sqrt{2}, -1/\sqrt{2})$ et $(-1/\sqrt{2}, 1/\sqrt{2})$ avec $f_{\max} \approx 0.735758882$.

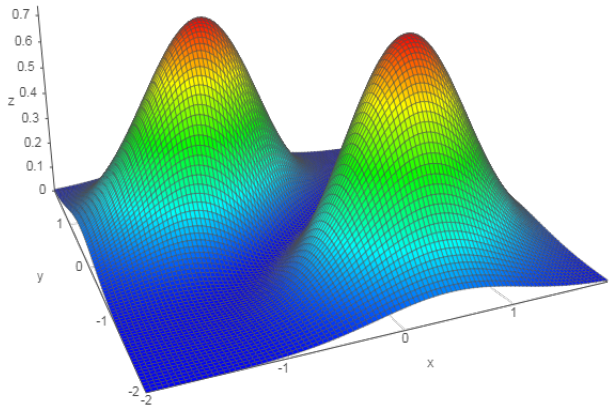


FIGURE 1.1 – Paysage (landscape) de la fonction donnée par l'équation 1.30.

Si nous utilisons une méthode basée sur le gradient, le résultat final peut grandement dépendent du vecteur initial $\mathbf{x}^0 = (x_0, y_0)$. En fait, nous pouvons essayer plusieurs algorithmes, et nous constaterons toujours que le résultat final dépendrait largement de la valeur du vecteur initial $\mathbf{x}^0$. Le problème de maximisation défini par l'équation 1.30 est équivaut à grimper sur deux pics distincts (voir la figure 1.1), où nous ne pouvons atteindre qu'un seul pic à la fois. En d'autres termes, le pic que nous cherchions dépendrait largement de notre point de départ. Pour que les deux pics soient atteints, les points de départ doivent être distribués aléatoirement dans l'espace de recherche. Si nous tirons aléatoirement un point de départ qui se trouve toujours dans une même région, alors l'autre pic peut ne jamais être atteint.

Une stratégie commune pour s'assurer que tous les sommets soient accessibles est de réaliser une optimisation ascendante avec plusieurs redémarrages aléatoires. C'est une stratégie simple mais très efficace. Une fonction avec plusieurs pics ou vallées est une fonction multimodale. Nous supposons une fonction $f(\mathbf{x})$ possédant k pics. Si nous effectuons une optimisation ascendante avec redémarrage aléatoire n fois $(n >> k)$; et si les points de départ $\mathbf{x}^0$ proviennent des différentes régions de l'espace de recherche, alors il est susceptible d'atteindre tous les pics de $f(\mathbf{x})$. En réalité, les choses ne sont pas si simples, à cause de plusieurs raisons. Primo, nous ne pouvons pas généralement savoir avec exactitude le nombre de pics et vallées d'une fonction, et souvent il n'y a aucune garantie que tous les pics sont atteints. Secundo, la majorité des problèmes réels ne possèdent pas une forme analytique ou explicite de la fonction objective. Tertio, beaucoup de problèmes peuvent prendre des valeurs continues et discrètes à la fois et sont non-dérivables. Par exemple, la fonction suivante :

$$g(x,y) = (|x| + |y|)e^{(-x^2-y^2)}$$

possède un minimum global $f_{\min} = 0$ à $(0,0)$. Cependant, la dérivée à $(0,0)$ n'existe pas. Dans ce cas, toutes les méthodes basées sur le gradient ne sont pas applicables.

Dans la pratique, les problèmes d'optimisation sont beaucoup plus compliqués et les calculs des dérivées peuvent être soit impossible, soit trop coûteux. Par conséquent, les méthodes sans gradient sont préférées. Il est important de souligner que la majorité des algorithmes inspirés par la nature n'utilisent pas le gradient.

1.5 Caractéristiques des problèmes d'optimisation

Après la formulation correcte d'un problème d'optimisation, la tâche suivante consiste à trouver ses solutions optimales en utilisant les bonnes techniques mathématiques. Au sens figuré, la recherche de ces solutions est comme la chasse au trésor dans un paysage vallonné en un temps limite. Dans ce cas, deux scénarios sont possibles. Primo, si nous supposons que nos yeux sont bandés sans les moindres directives, alors la recherche est purement aléatoire et généralement inefficace. Secundo, on nous dit que le trésor se trouve au plus haut pic d'une certaine région, alors nous y grimpons progressivement et tentons de l'atteindre, ce qui correspond à une optimisation ascendante. Dans la plupart des cas, la recherche se fait entre les pics : nous n'avons pas les yeux bandés et nous ne savons pas où chercher. Il est extrêmement coûteux de scruter chaque centimètre carré d'une très grande région vallonnée pour trouver le trésor, alias la solution optimale.

Le scénario de recherche le plus adéquat est de faire des pas aléatoires, tout en suivant des indices. Nous cherchons aléatoirement dans une région, puis nous irons à une autre région vraisemblable, ensuite vers une autre région plausible, et ainsi de suite. Une telle démarche aléatoire est une caractéristique principale des algorithmes de recherche modernes. Il y a deux manières différentes pour chercher la solution optimale. Primo, la recherche peut se faire par un seul agent (i.e., recherche basée-trajectoire, telle que : recuit-simulé [3]). Secundo, la recherche peut se faire par un groupe d'agents qui partagent des informations sur l'espace de recherche (i.e., recherche basée-population, telles que : optimisation par essaims particulaires [49] ou algorithme d'optimisation par lucioles [50]).

Dans la recherche utilisant un groupe d'agents, la stratégie de recherche peut être raffinée (i.e., certains agents sont meilleurs que d'autres), alors nous pouvons garder que les meilleurs agents et replacer les mauvais par de nouveaux agents, tel est le principe des algorithmes évolutionnaires [51, 52]. La majorité des métaheuristiques modernes utilisent les meilleures solutions (i.e., agents) et remplacent/randomisent les mauvais. Avec de telles stratégies, nous essayons de concevoir des algorithmes d'optimisation meilleurs et efficaces.

La classification des algorithmes d'optimisation peut se faire en considérant différents critères. Principalement, nous trouvons les algorithmes déterministes, indéterministe et hybrides. Les algorithmes d'optimisation déterministes suivent une procédure rigoureuse. Les chemins et valeurs des variables de décisions et fonction objective sont répétés. Autrement dit, pour un même point de départ un algorithme déterministe suivra toujours le même chemin (e.g., l'optimisation descendante). En revanche, les algorithmes d'optimisation indéterministes suivent toujours des chemins aléatoires. Les algorithmes génétiques [53] en sont un exemple typique. Les solutions dans la population initiale sont différentes lors de chaque exécution. Toutefois, les solutions finales ne sont pas grandement différentes. Les chemins et valeurs des solutions ne sont pas exactement répétés. Enfin, les algorithmes d'optimisation hybrides utilisent des algorithmes déterministes, mais les points de départs sont aléatoires. L'optimisation ascendante avec redémarrage aléatoire en est un exemple typique. Ces algorithmes présentent l'avantage de ne pas être coincés dans des minimums locaux [54].

L'ensemble des optimums globaux d'un problème d'optimisation peut être donné par l'équation 1.31 comme suit :

$$S = \{X_i^* | X_i^* \in D\} \tag{1.31}$$

cela signifie qu'un problème d'optimisation peut avoir : un seul optimum global, plusieurs optimums globaux ou une infinité d'optimums globaux. L'objectif des algorithmes d'optimisation est de trouver les solutions optimales ou quasi-optimales. Dans la suite, nous donnons quelques définitions spécifiques aux caractéristiques des problèmes d'optimisation.

Définition 1.1 (fonction continue)

Une fonction $f : \mathbb{R}^d \to \mathbb{R}$ est continue au point $\boldsymbol{x}_0 \in \mathbb{R}^d$, si pour tout $\varepsilon > 0$, il existe $\delta > 0$, tel que [55] :

$$\|\boldsymbol{x} - \boldsymbol{x}_0\| < \delta \Rightarrow \|f(\boldsymbol{x}) - f(\boldsymbol{x}_0)\| < \varepsilon \tag{1.32}$$

où $\|.\|$ est la norme euclidienne.

Définition 1.2 (fonction différentiable)
Une fonction $f : \mathbb{R}^d \to \mathbb{R}$ est différenciable au point $\boldsymbol{x}_0 \in \mathbb{R}^d$, s'il existe un maping linéaire continu $\bigtriangledown f(\boldsymbol{x}_0) : \mathbb{R}^d \to \mathbb{R}$, tel que [55] :

$$\lim_{h \to 0} \frac{f(\boldsymbol{x}_0 + h) - f(\boldsymbol{x}_0) - \bigtriangledown f(\boldsymbol{x}_0).h}{\|h\|} = 0 \tag{1.33}$$

où $\|.\|$ est la norme euclidienne.

Définition 1.3 (convexité)
Une fonction $f : \mathbb{R}^d \to \mathbb{R}$ est convexe, si le segment de ligne entre deux points quelconques de son paysage se trouve toujours au-dessus de ce dernier [55].

Définition 1.4 (dimensionnalité)
La dimensionnalité d'un problème se réfère au nombre de ses variables de décision. Habituellement, la difficulté d'un problème augmente avec sa dimensionnalité : i.e., l'espace de recherche augmente exponentiellement avec la dimension du problème considéré [56, 57]. Dans les problèmes non-linéaires, le nombre de variables de décision peut considérablement entraver la majorité des algorithmes d'optimisation.

Définition 1.5 (séparabilité)
Une fonction $f : \mathbb{R}^d \to \mathbb{R}$ est séparable, si elle peut être écrite comme une somme de d fonctions à une seule variable $h_i : \mathbb{R} \to \mathbb{R}$ [58]. Mathématiquement, cela s'exprime comme suit :

$$f \text{ est séparable } \Rightarrow f(x_1, x_2, \ldots, x_d) = \sum_{i=1}^{d} h_i(x_i) \tag{1.34}$$

Si une fonction est séparable, alors la solution optimale peut être facilement trouvée, car chaque fonction peut être optimisée indépendamment.

Définition 1.6 (modalité)
La modalité d'une fonction correspond au nombre de pics ambigus dans son paysage [59]. Les pics sont une source sérieuse de problèmes : i.e., les algorithmes d'optimisation peuvent être piégés dans l'un d'eux durant le processus de recherche. Ainsi, les pics peuvent traîner les algorithmes d'optimisation vers des optimums locaux ; et par conséquent, s'éloigner du vrai optimum global.

Définition 1.7 (bassin)
Étant donné le paysage d'une fonction objective. Un bassin est une petite baisse dans une grande surface [60]. Les algorithmes d'optimisation sont couramment attirés par les bassins. Si la recherche est piégée dans un bassin, alors ceci aurait un impact majeur sur la qualité des solutions trouvées. Il existe une autre description d'un bassin, c'est un plateau dans un problème de maximisation.

Définition 1.8 (vallée)
Étant donné le paysage d'une fonction objective. Une vallée est une grande baisse dans une certaine région [60]. Les algorithmes d'optimisation sont couramment attirés par les vallées. Si la recherche est piégée dans une certaine vallée, alors ceci ralentirait considérablement le processus de recherche. Il y a une autre description d'une vallée, c'est une montagne dans un problème de maximisation.

1.6 Théorème de « pas-de-repas-gratuit »

Une question courante posée par de nombreux chercheurs est la suivante : « il existe plusieurs algorithmes d'optimisation, alors qu'en est-il le meilleur ? »

C'est une question simple, mais malheureusement, il n'y a pas de réponse simple. Il y a beaucoup de raisons pour lesquelles nous ne pouvons pas répondre simplement à cette question. Une des raisons est que la complexité et diversité des problèmes réels sont grandement différentes : i.e., certains problèmes sont plus faciles à résoudre, tandis que d'autres peuvent être extrêmement difficiles à résoudre. Par conséquent, il est peu probable qu'une méthode unique et universelle soit capable de faire face à tous les types de problèmes. Une autre raison est qu'il existe un théorème qui s'appelle « pas de repas gratuit », de l'anglais No-Free-Lunch (NFL) [61], qui montre qu'il n'y a pas un algorithme d'optimisation universel qui peut résoudre tous les problèmes.

1.6.1 Théorème de NFL

La classe des théorèmes de NFL se prononce comme suit : « si les performances d'un algorithme A surpassent celles d'un autre algorithme B, pour la recherche d'un optimum global d'une certaine fonction objective $f(\mathbf{x})$, alors les performances de B surpasseront celles de A pour d'autres fonctions objectives $g(\mathbf{x}), h(\mathbf{x}), \ldots$ » [61].

De plus, tous les théorèmes de NFL suggèrent que la moyenne des performances sur toutes les fonctions objectives possibles sont les mêmes pour tous les algorithmes d'optimisation. Mathématiquement parlant, si $P(s_m|f,m,A)$ dénote les performances statistiques (e.g., moyenne, écart-type, etc.) de l'algorithme A exécuté m fois sur une fonction objective f en considérant l'échantillon s_m, alors nous avons la formule suivante pour les performances moyennes de deux algorithmes (la preuve par induction de cette déclaration peut être trouvée dans le travail [62]) :

$$\sum_f P(s_m|f,m,A) = \sum_f P(s_m|f,m,B) \tag{1.35}$$

La formule 1.35 annonce que les performances d'un algorithme A sont indépendantes de l'algorithme lui-même. Autrement dit, tous les algorithmes d'optimisation donneront les mêmes performances lorsqu'ils sont moyennés sur toutes les fonctions possibles, ce qui montre l'inexistence d'un algorithme universel pour tous les problèmes d'optimisation.

Nous pourrions nous dire qu'il n'est pas nécessaire de développer de nouveaux algorithmes, car leurs performances moyennes sont similaires. Ceci n'est pas vraiment l'intention de théorème de NFL. Le noyau du théorème est l'assertion « performances moyennes sur toutes les fonctions et tous les problèmes ». Cela ne signifie pas que tous les algorithmes possèdent exactement les mêmes performances sur toutes les fonctions et tous les problèmes : i.e., un algorithme d'optimisation n'a pas besoin de mesurer ses performances moyennes sur toutes les fonctions et tous les problèmes possibles.

Même si le théorème de NFL est mathématiquement valide, son impact sur l'optimisation est limité. Pour un ensemble de fonctions objectives, certains algorithmes fonctionnent beaucoup mieux que d'autres. Pour un problème spécifique avec des fonctions objectives, il existe généralement des algorithmes qui sont plus efficaces que d'autres, bien sûr si nous n'avons pas besoin de mesurer leurs performances moyennes. Le défi principal est plutôt de savoir comment trouver ces meilleurs algorithmes pour un problème donné.

Il convient de souligner que le théorème de NFL est prouvé pour les problèmes d'optimisation mono-objective, et pour les problèmes multi-objectives son extension est toujours sous étude [63]. Certaines études suggèrent que les hypothèses de base de ce théorème pourraient ne pas être valides pour les domaines

continus. Par exemple, le travail dans [64] déclare que les problèmes continus peuvent être libres. De plus, le travail dans [65] met l'accent sur l'hypothèse d'interdire de visiter un point plusieurs fois, et annonce que ceci n'est pas valable dans la pratique.

1.6.2 Choix des algorithmes

Maintenant, nous mettons de côté le théorème de NFL, et pensons comment choisir un algorithme et quels sont les problèmes à résoudre ? Ici, il y a deux questions pertinentes :

1. Pour une certaine famille de problèmes, quel est le meilleur algorithme à utiliser ?
2. Pour un algorithme donné, quelles familles de problèmes peut-il résoudre ?

La première question est plus difficile que la deuxième, bien qu'il ne soit pas facile de répondre aux deux questions. Pour une certaine famille de problèmes, il peut y avoir un ensemble d'algorithmes efficaces pour résoudre de tels problèmes. Toutefois, dans de nombreux cas, nous ne savons pas l'efficacité d'un algorithme avant de l'essayer. De plus, de tels algorithmes doivent encore être développé dans certains cas. Même pour les algorithmes existants, le choix dépend largement de l'expertise du décideur, les ressources disponibles et le type de problème. Idéalement, les meilleurs algorithmes et outils disponibles doivent être utilisés pour résoudre un problème donné ; cependant, la bonne utilisation de ces outils peut encore dépendre de l'expertise du décideur. En plus, les ressources (e.g., coûts computationnels, disponibilité des logiciels et temps alloué pour produire les solutions) sont également un facteur important pour décider quels sont les algorithmes à utiliser [62].

Pour un algorithme donné, les problèmes qu'il peut résoudre peuvent être explorés en résolvant diverses familles de problèmes, puis comparant et classant les performances afin de découvrir son efficacité. De cette façon, les avantages et les inconvénients de chaque algorithme peuvent être identifiés. Par la suite, ces connaissances peuvent être utilisées pour guider le choix des algorithmes à utiliser lors de la résolution d'un certain problème. Donc, toute connaissance spécifique à un problème particulier est utile pour choisir le meilleur algorithme d'optimisation [62].

1.7 Algorithmes inspirés par la nature

La majorité des algorithmes d'optimisation conventionnels sont déterministes. La méthode du simplex [66], en programmation linéaire, est déterministe. Quelques algorithmes déterministes sont basés sur le gradient (e.g., la méthode de Newton-Raphson [42, 43] qui fonctionne bien pour les problèmes unimodaux). Les méthodes basées sur le gradient ne fonctionnent pas bien lorsque la fonction objective est discontinue. Dans ce cas, les algorithmes qui n'utilisent pas le gradient sont préférés. Les algorithmes sans-gradient n'utilisent aucune dérivée, seulement les valeurs de fonctions sont prises en compte. Les algorithmes d'optimisation présentés dans [67, 68] sont sans-gradient.

Pour les algorithmes stochastiques ou indéterministes, nous avons deux types : heuristique et métaheuristique ; bien que la différence entre les deux termes soit minime [69]. Primo, le terme heuristique signifie « découvrir par essai et erreur ». Avec ces algorithmes, nous obtenons en général de très bonnes solutions aux différents problèmes d'optimisation en des délais raisonnables, mais rien ne garantit l'optimalité des solutions obtenues. Nous pouvons nous attendre à ce que ces algorithmes fonctionnent la plupart du temps ; et c'est largement suffisant lorsque nous ne voulons pas forcément des solutions optimales, mais plutôt des solutions facilement accessibles. Secundo, le terme méta signifie « niveau supérieur ». Toutes les métaheuristiques partagent un même principe : « trouver un compromis entre recherche globale et recherche locale ». Dans la littérature, il n'existe aucun consensus sur l'usage des termes « heuristique » et « métaheuristique » ; et certains auteurs tendent à les utiliser d'une manière interchangeable. La tendance récente tend

à nommer tous les algorithmes stochastiques avec randomisation et recherche locale comme métaheuristique. La randomisation offre un bon moyen pour s'éloigner de la recherche locale et faire la rechercher globale. Par conséquent, les métaheuristiques sont préférées pour l'optimisation globale. Les performances des métaheuristiques sont très souvent meilleures que celles des heuristiques [62].

En considérant un problème donné, les métaheuristiques produisent généralement des solutions acceptables en des temps pratiquement raisonnables. La complexité des problèmes en question rend très souvent impossible de vérifier l'optimalité de toutes les solutions (i.e., combinaisons possibles des valeurs de variables de décision). L'objectif étant de trouver de bonnes solutions réalisables en un délai acceptable. Une métaheuristique n'offre aucune garantie sur l'obtention de la solution optimale, et nous ne savons même pas pourquoi elle fonctionne ou non sur un certain problème. L'idée est d'avoir une métaheuristique efficace mais pratique qui fonctionnerait la plupart du temps, et qui est capable de produire des solutions de bonnes qualités. Parmi ces solutions, nous nous attendons à ce que certaines soient presque optimales, bien qu'il n'y ait aucune garantie pour une telle optimalité.

Toutes les métaheuristiques possèdent deux composantes principales : i.e., intensification et diversification ou exploitation et exploration. La diversification signifie : « générer aléatoirement des solutions diverses afin d'explorer globalement l'espace de recherche ». L'intensification signifie : « concentrer la recherche dans une région locale en exploitant les informations de celle-ci ». En plus de ces deux composantes, nous avons la sélection des meilleures solutions. La sélection garantit la convergence des solutions vers l'optimalité, alors que la diversification par randomisation évite que les solutions soient coincées dans des optimums locaux et, en parallèle, augmente la diversité des solutions. La bonne combinaison de ces concepts-clés garantirait généralement l'optimalité globales des solutions obtenues.

Les métaheuristiques peuvent être classées de plusieurs façons. Une façon consiste à les classer en fonction du nombre d'agents qui naviguent ou essaiment dans l'espace de recherche. Nous trouvons principalement deux catégories : métaheuristiques basées-population et métaheuristiques basées-trajectoire. Les algorithmes génétiques [53], l'optimisation par essaims particulaires [49] et l'algorithmes des lucioles [50] sont des métaheuristiques basées-population (utilisant plusieurs agents ou solutions candidates). Le recuit simulé [3] est une métaheuristique basée-trajectoire (utilisant un seul agent).

1.8 Historique des métaheuristiques

Tout au long de l'histoire de l'humanité, en particulier aux premiers temps, nous avons toujours utilisé des heuristiques ou métaheuristiques pour résoudre les différents problèmes. De nombreuses découvertes importantes ont été faites, accidentellement, en sortant des sentiers battus. L'apprentissage quotidien des humains est principalement heuristique.

Malgré leurs omniprésences, les métaheuristiques, en tant que méthodes scientifiques de résolution de problèmes, sont en fait récentes, bien qu'il soit difficile de déterminer leur premier usage. Le mathématicien Alan Turing a probablement été le premier à utiliser des algorithmes heuristiques, durant la Seconde Guerre mondiale, lorsqu'il brisait les chiffres Enigma à Bletchley Park. Turing a appelé sa méthode de recherche une recherche heuristique, car nous pouvons nous attendre à ce qu'elle fonctionne la plupart du temps, mais il n'y avait aucune garantie de trouver la solution optimale ; cependant, sa méthode était un énorme succès. En 1945, Turing a été recruté au laboratoire national de physique au Royaume-Uni. Dans un rapport sur les machines intelligentes en 1948 [70, 71], il a décrit ses idées innovantes sur l'intelligence et apprentissage des machines, les réseaux de neurones et les algorithmes évolutionnaires.

Les années 1960 et 1970 ont été les deux décennies importantes pour le développement des algorithmes évolutionnaires. John Holland et ses collaborateurs ont développé les algorithmes génétiques (AG) dans les années 60 et 70. Dès 1962, Holland a étudié les systèmes adaptatifs, et a été le premier à utiliser des

croisements et recombinaisons pour les modéliser. Son livre fondateur, résumant le développement des AG, a été publié en 1975 [4]. La même année, Kenneth De Jong a terminé sa dissertation en montrant la puissance des AG à résoudre un large éventail de problèmes d'optimisation [72].

Un AG est une méthode de recherche basée sur l'abstraction de l'évolution darwinienne et la sélection naturelle des systèmes biologiques, qui sont représentées sous la forme d'opérateurs mathématiques : i.e., croisement, mutation, fitness et sélection du plus apte. Les AG sont devenus très efficaces pour résoudre un large éventail de problèmes d'optimisation. Plusieurs milliers d'articles de recherche et des centaines de livres ont été publiés sur les applications des AG [73, 74]. Les statistiques montrent que la majorité des sociétés les utilisent régulièrement pour résoudre les différents problèmes d'optimisation combinatoires : e.g., planification [75, 76], fouille de données [77], ordonnancement [78-80], etc.

Durant la même période, Ingo Rechenberg et Hans-Paul Schwefel ont développé une technique de recherche pour résoudre les problèmes d'optimisation en génie aérospatial, appelée Stratégie d'Evolution (SE) [81, 82]. Plus tard, Peter Bienert a construit un expérimentateur automatique [83] en utilisant des règles simples de mutation et sélection. Il n'y avait aucun croisement dans sa technique ; seule la mutation a été utilisée pour produire des progénitures, et à chaque génération une solution améliorée est conservée. Dès 1960, Lawrence J. Fogel avait l'intention d'utiliser l'évolution simulée comme processus d'apprentissage et outil pour étudier l'intelligence artificielle. Puis, en 1966, Fogel et ses collaborateurs ont développé la programmation évolutionnaire, en représentant les solutions comme des machines à états finis et mutant aléatoirement l'une de ces machines [84]. Ces idées et méthodes innovantes ont évolué vers une discipline beaucoup plus large, appelée algorithmes évolutionnaires [85, 86].

D'autres algorithmes peuvent être aussi considérés comme des techniques d'optimisation métaheuristiques. Ces méthodes incluent les réseaux de neurones artificiels [87], machines à vecteurs de support [88] et de nombreuses techniques d'apprentissage automatique [89]. Ces techniques essayent de minimiser les erreurs d'apprentissage et prédiction, via des essais et erreurs itératifs.

Les réseaux de neurones artificiels (RNA) sont couramment utilisés dans de nombreuses applications. En 1943, Warren McCulloch et Walter Pitts ont proposé les neurones artificiels comme simples unités de traitement de l'information. Le concept de RNA a probablement été proposé pour la première fois par Alan Turing dans son rapport sur les machines intelligentes [70, 71]. Dès lors, des développements significatifs ont été réalisés dans le domaine des RNA [90].

Les machines à vecteurs de support (SVM), en tant que technique de classification, proviennent des travaux précoces de Vladimir Vapnik en 1963 sur les classifiers linéaires ; la classification non-linéaire avec des noyaux a été développée par Vapnik et ses collaborateurs dans les années 90. Un résumé systématique a été publié dans le livre de Vapnik, « The Nature of Statistical Learning Theory », en 1995 [91].

Les décennies des années 80 et 90 ont été la période la plus excitante pour les métaheuristiques. Le grand pas était le développement du recuit simulé (Simulated Annealing) en 1983, une technique d'optimisation inspirée par le processus de recuit des métaux [92]. Il s'agit d'un algorithme de recherche basé-trajectoire, qui commence par une solution initiale à haute température puis refroidit progressivement le système. Une nouvelle solution est acceptée si elle est meilleure ; sinon, elle est acceptée avec une certaine probabilité, ce qui permet au système d'échapper aux optimums locaux. Il est alors admis que si le système est refroidi assez lentement, alors l'optimum global peut être atteint.

L'utilisation réelle des mémoires dans les métaheuristiques modernes est probablement due à la recherche taboue (Tabu Search) de Fred Glover en 1986, bien que son livre fondateur sur la recherche taboue ait été publié plus tard, en 1997 [86].

En 1992, Marco Dorigo a soutenu sa thèse de doctorat sur l'optimisation et algorithmes naturels [93], dans laquelle il décrivait ses travaux innovants sur l'optimisation par colonies de fourmis (Ant Colony Op-

timization). Cet algorithme imite le comportement des fourmis sociales qui utilisent la phéromone comme moyen de communication.

En 1992, John R. Koza a publié un traité sur la programmation génétique, dans lequel il a mis les bases d'un tout nouveau domaine de l'apprentissage automatique, révolutionnant ainsi la programmation informatique [15]. L'idée de base est de faire croiser des programmes informatiques, afin de produire progressivement de meilleurs programmes pour un problème donné.

En 1995, des progrès importants ont été réalisés avec le développement de l'optimisation par essaims particulaires (Particle Swarm Optimisation) [49]. Cet algorithme imite l'intelligence en essaim des troupeaux. Au début, chaque particule est initialisée d'une manière aléatoire (i.e., position) ; puis, toutes les particules essaiment harmonieusement dans l'espace de recherche. Elles partagent toujours la position de la meilleure particule (optimum global). Depuis son apparition, il y a eu environ une vingtaine de variantes de PSO [94], qui ont été appliquées à plusieurs problèmes d'optimisation difficiles.

Vers 1996 et plus tard en 1997, Rainer Storn et Kenneth Price ont développé leur algorithme évolutionnaire à base de vecteurs appelé évolution différentielle (Differential Evolution) [95], qui s'avère plus efficace que les AG dans de nombreuses applications [96].

En 1997, la publication du théorème de NFL a semé la panique au sein de la communauté spécialisée en optimisation [61, 97]. Depuis l'apparition des métaheuristiques, les chercheurs ont toujours essayé de concevoir de meilleurs algorithmes, voire même un algorithme universel et robuste pour tous les problèmes d'optimisation, et en particulier les problèmes d'optimisation $\mathscr{NP}$-difficiles. Ce théorème s'annonce comme suit : si A et B sont deux algorithmes et $P = \{p_1, p_2, \ldots, p_n\}$ est l'ensemble de tous les problèmes d'optimisation ; et si les performances de A sont meilleures que celles de B pour un sous-ensemble $P_1 \subseteq P$, alors les performances de B sont meilleures que celles de A pour le sous-ensemble $P_2 = P \setminus P_1$. De plus, si les performances de A et B sont moyennées sur P, alors leurs performances sont similaires. D'une manière simple, l'existence d'un algorithme universel qui résout parfaitement tous les problèmes d'optimisation est impossible. Par la suite, les chercheurs se sont réalisés qu'il n'est pas nécessaire de moyenner les performances d'un algorithme sur tous les problèmes possibles, ce que nous voulons c'est de trouver les meilleurs algorithmes pour les problèmes considérés. Ainsi, les chercheurs se sont concentrés plutôt sur la recherche de meilleurs algorithmes pour un ensemble donné de problèmes.

Au tournant du 21^{e} siècle, les choses se sont devenues encore plus excitantes, et la littérature concernant les métaheuristiques s'est rapidement gorgée. Ainsi, le nombre d'algorithmes inspirées par la nature a vertigineusement augmenté. Les travaux publiés dans [98, 99] répertorient les différentes métaheuristiques proposées dans la littérature.

1.9 Conclusion

Les algorithmes d'optimisation occupent une place importante dans la communauté scientifique, et font toujours couler beaucoup d'encre depuis leur apparition. Nous avons introduit le concept d'algorithmes d'optimisation et fourni quelques méthodes d'approximation pour la résolution efficace de problèmes d'optimisation. De plus, nous avons présenté formellement les problèmes d'optimisation, leurs caractéristiques et les deux grandes familles d'algorithmes d'optimisation.

Nous avons également évoqué l'un des théorèmes fondamentaux en optimisation, qui démontre la non-existence d'un algorithme d'approximation universel pouvant résoudre efficacement tous les problèmes d'optimisation. Enfin, nous avons mis l'accent sur les caractéristiques des métaheuristiques et en donné un bref historique.

Dans le prochain chapitre, nous allons fournir une analyse formelle des métaheuristiques, et débattre quelques algorithmes populaires en termes d'opérateurs évolutionnaires, exploration, exploitation et paramétrage.

Chapitre 2

Analyse des métaheuristiques

Sommaire

2.1 Introduction

Durant les dernières décennies, les algorithmes inspirés par la nature ont suscité beaucoup d'intérêt dans la littérature, et devenus très populaires. Les algorithmes inspirés par la nature (alias métaheuristiques) sont maintenant parmi les algorithmes largement utilisés pour résoudre les différents problèmes d'optimisation, à cause de leurs nombreux avantages par rapport aux algorithmes conventionnels. Dans ce chapitre, nous analysons les composantes principales des métaheuristiques, en termes d'opérateurs évolutionnaires, exploration et exploitation. L'objectif majeur est de fournir une vue d'ensemble et les outils nécessaires, afin de pouvoir analyser de manière critique les différentes métaheuristiques.

Comme nous l'avons vu au chapitre 1, un algorithme d'optimisation est une procédure itérative avec un état initial ; et après un certain nombre d'itérations (supposé suffisant) l'algorithme convergerait vers un état stable, qui est dans le cas idéal la solution optimale au problème considéré [100, 101]. Il s'agit essentiellement d'un système auto-organisé, où les solutions sont les états et les solutions visitées sont les attracteurs. Un tel système itératif et auto-organisé évolue en suivant certaines règles et équations mathématiques. Par conséquent, un système aussi complexe peut interagir et s'auto-organiser en certains états convergents, en montrant ainsi quelques caractéristiques émergentes d'auto-organisation. En ce sens, la conception appropriée d'un algorithme d'optimisation efficace équivaut à trouver un moyen efficace d'imiter l'évolution d'un système auto-organisé, en particulier l'évolution des systèmes biologiques [102, 103].

Alternativement, nous pouvons voir un algorithme comme une chaîne de Markov [104], où son comportement est contrôlé par les solutions visitées (i.e., états) et transitions. En effet, différentes vues peuvent aider à analyser les algorithmes sous différentes perspectives. Nous pouvons également analyser un algorithme en termes de ses composantes clés : i.e., exploration et exploitation (générer des solutions à l'aide d'opérateurs évolutionnaires). Dans ce chapitre, nous examinons et discutons quelques métaheuristiques sous différents angles.

2.2 Analyse des algorithmes d'optimisation

Un algorithme d'optimisation peut être analysé sous différentes perspectives. Dans cette section, nous analysons un algorithme comme : (i) une procédure itérative, (ii) un système auto-organisé, (iii) deux composantes conflictuelles et (iv) des opérateurs évolutionnaires.

2.2.1 Procédure itérative

Mathématiquement parlant, un algorithme A est un processus itératif, qui vise à générer une nouvelle et meilleure solution $\mathbf{x}^{t+1}$, à un problème donné, à partir d'une solution courante $\mathbf{x}^t$ à l'itération t. Par exemple, la méthode de Newton-Raphson [42, 43] (pour trouver l'optimum de $f(\mathbf{x})$) équivaut à trouver les racines de $f'(\mathbf{x}) = 0$ dans un espace à d dimensions [105, 106].

$$\mathbf{x}^{t+1} = \mathbf{x}^t - \frac{f'(\mathbf{x}^t)}{f''(\mathbf{x}^t)} = A(\mathbf{x}^t) \tag{2.1}$$

De toute évidence, le taux de convergence devient très lent au voisinage de l'optimum ($f'(\mathbf{x}) \to 0$). Parfois, le véritable taux de convergence peut ne pas être aussi rapide qu'il devrait l'être. Un moyen simple pour améliorer la convergence consiste à modifier l'équation 2.1 en introduisant un paramètre ρ, comme suit :

$$\mathbf{x}^{t+1} = \mathbf{x}^t - \rho \frac{f'(\mathbf{x}^t)}{f''(\mathbf{x}^t)} \text{ , où } \rho = \frac{1}{1 - A'(\mathbf{x}_*)} \tag{2.2}$$

Ici $\mathbf{x}_*$ est la solution optimale. Il convient de souligner que la convergence de la méthode de Newton-Raphson nécessite un paramétrage optimal de ρ, qui dépend de la formule itérative, optimalité de $\mathbf{x}_*$ et fonction objective $f(\mathbf{x})$. En général, nous pouvons écrire l'équation itérative précédente comme suit :

$$\mathbf{x}^{t+1} = A(\mathbf{x}^t, \rho) \tag{2.3}$$

qui est valable pour les méthodes déterministes ; cependant, dans les métaheuristiques, la randomisation est souvent utilisée, et dans de nombreux cas elle apparaît sous la forme d'un ensemble de m variables aléatoires $\mathbf{v} = (v_1, \ldots, v_m)$. Par exemple, le recuit simulé [92] a une seule variable, et l'optimisation par essaims particulaires [49] a deux variables. De plus, il y a souvent un ensemble de k paramètres dans une métaheuristique. Par exemple, l'optimisation par essaims particulaires [49] a quatre paramètres : deux paramètres d'apprentissage, un poids d'inertie et la taille de la population. En général, nous avons un vecteur de paramètres $\mathbf{p} = (p_1, \ldots, p_k)$. Mathématiquement, nous pouvons écrire un algorithme avec k paramètres et m variables aléatoires comme suit :

$$\mathbf{x}^{t+1} = \mathbf{A}(\mathbf{x}^t, \mathbf{p}(t), \mathbf{v}(t)) \tag{2.4}$$

où $\mathbf{A}$ est un mapping non-linéaire d'un vecteur $\mathbf{x}^t$ à un nouveau vecteur $\mathbf{x}^{t+1}$. Il convient de souligner que la formule 2.4 concerne uniquement les métaheuristiques basées-trajectoire. Pour les métaheuristiques basées-population de taille n, nous pouvons étendre la formule itérative 2.4 comme suit :

$$\begin{pmatrix} \mathbf{x}_1^{t+1} \\ \vdots \\ \mathbf{x}_n^{t+1} \end{pmatrix} = \mathbf{A}((\mathbf{x}_1^t, \ldots, \mathbf{x}_n^t), \mathbf{p}(t), \mathbf{v}(t)) \begin{pmatrix} \mathbf{x}_1^t \\ \vdots \\ \mathbf{x}_n^t \end{pmatrix} \tag{2.5}$$

Il est admis que si le nombre d'itérations t (nécessaire pour trouver une solution avec une certaine précision) est assez grand, alors les performances de l'algorithme seront grandement détériorées. Donc, un meilleur algorithme devrait utiliser moins de calculs et moins d'itérations. Dans le cas idéal, un algorithme devrait faire une seule itération (i.e., $t = 1$) pour arriver à la solution optimale. Ainsi, nous pouvons nous demander si un tel algorithme existe dans la pratique ; la réponse est « oui, ça dépend ». Par exemple, pour la fonction quadratique $f(x) = ax^2$ où $a > 0$, la méthode de Newton-Raphson peut être appliquée en prenant $x^0 = b$, alors nous avons :

$$x^1 = x^0 - \frac{f'(x^0)}{f''(x^0)} = b - \frac{2ab}{2a} = 0$$

ce qui donne la solution optimale globale $f_*(0) = 0$ à $x_* = 0$.

Évidemment, nous pouvons étendre cette idée à toutes les fonctions quadratiques. Il est même possible de l'étendre aux fonctions convexes. Cependant, de nombreux problèmes ne sont pas convexes et certainement pas quadratiques. Par conséquent, l'algorithme idéal n'existe pas en général. Donc, il n'y a pas un bon algorithme pour résoudre les problèmes $\mathscr{NP}$-difficiles.

Il existe de nombreux algorithmes d'optimisation dans la littérature, et il n'y a aucun qui convient à tous les problèmes. La quête d'algorithmes efficaces fait toujours couler beaucoup d'encre dans la communauté scientifique, et certainement cette mission de recherche du Saint-Graal se poursuivrait jusqu'à preuve analytique de contraire.

2.2.2 Système auto-organisé

Un système complexe peut s'auto-organiser dans les bonnes conditions : viz. lorsque la taille du système est suffisamment grande, avec un nombre assez élevé d'états possibles. De plus, le système doit évoluer durant une longue période loin du bruit et états d'équilibre. Également, un mécanisme de sélection doit être mis en place pour garantir l'auto-organisation [107].

En d'autres termes, un système possédant S états possibles évoluant vers un état auto-organisé S_* (piloté par un mécanisme $\alpha(t)$ avec un ensemble de paramètres) peut être représenté comme suit :

$$S \xrightarrow{\alpha(t)} S_* \tag{2.6}$$

Ainsi, l'équation 2.4 peut être vue comme un système auto-organisé. Ce système est initialisé à l'état $\mathbf{x}^t$ et tente de converger vers l'état optimal $\mathbf{x}_*$ en suivant l'algorithme **A**. Autrement dit, nous avons l'équation suivante :

$$f(\mathbf{x}^t) \xrightarrow{A} f_{\min}(\mathbf{x}_*) \tag{2.7}$$

Il existe des différences importantes entre un système auto-organisé et un algorithme. Pour un système auto-organisé, la façon d'accéder aux états peut ne pas être claire et le temps n'est pas un facteur important. Pour un algorithme, la manière et vitesse de convergence sont cruciales, pour que les coûts computationnels soient minimisés.

2.2.3 Exploration and exploitation

Les algorithmes d'optimisation inspirés par la nature peuvent également être analysés selon la manière d'observer l'espace de recherche. Toutes les métaheuristiques possèdent deux composantes principales : exploitation (intensification) et exploration (diversification) [108].

L'exploitation utilise toutes les informations relatives au problème pour générer de nouvelles meilleures solutions. L'exploitation peut être vue comme une recherche locale (les informations utilisées sont locales). Par exemple, l'optimisation descendante utilise des dérivées pour guider le processus de recherche. L'avantage majeur de l'exploitation est qu'elle permet généralement d'atteindre des taux de convergence très élevés (i.e., précision des solutions finales). Cependant, le processus de recherche peut se coincer dans des optimums locaux et, par conséquent, la solution finale dépendrait largement des solutions de départ.

L'exploration permet de scruter plus efficacement l'espace de recherche, où de nouvelles diverses solutions sont générées (i.e., loin des solutions actuelles). L'exploration peut être vue comme une recherche globale (les informations utilisées sont globales). L'avantage majeur de l'exploration est qu'elle permet d'éviter les optimums locaux et, par conséquent, l'optimum global est plus accessible. Cependant, le processus de recherche peut avoir une convergence lente et beaucoup d'efforts computationnels sont gaspillés, car de nombreuses nouvelles solutions peuvent être loin de l'optimum global.

Pour qu'une métaheuristique puisse montrer de bonnes performances, alors un bon équilibrage entre ces deux types de recherche est crucial. Par exemple, (i) trop d'exploitation et peu d'exploration signifient que l'algorithme peut converger plus rapidement, et la probabilité de trouver le véritable optimum global est très faible ; (ii) peu d'exploitation et trop d'exploration signifient que l'algorithme peut converger très lentement avec beaucoup d'efforts computationnels. Un équilibrage optimal signifie les bonnes quantités d'exploration et exploitation, dans une métaheuristique, qui donnent les meilleures performances. Ainsi, l'équilibrage entre exploration et exploitation est d'une importance primordiale.

Il convient de souligner que la façon d'atteindre un tel équilibrage est une piste de recherche très active. Dans la littérature actuelle, aucun algorithme ne peut prétendre avoir atteint un équilibrage optimal entre exploration et exploitation. Il s'agit d'un problème d'hyper-optimisation (i.e., optimisation d'un algorithme d'optimisation). De plus, nous rajoutons que cet équilibrage dépend de nombreux facteurs : (i) mécanisme de fonctionnement d'une métaheuristique ; (ii) valeurs, réglage et contrôle des paramètres ; (iii) problème en question, etc. Par conséquent, cet équilibrage peut ne pas exister universellement, et peut varier d'un problème à un autre.

2.2.4 Opérateurs évolutionnaires

Il est utile de voir le fonctionnement d'une métaheuristique en considérant ses opérations. Nous prenons l'exemple des algorithmes génétiques (AG). Les AG sont une classe d'algorithmes basés sur l'abstraction de l'évolution darwinienne des systèmes biologiques [16]. Ils utilisent essentiellement trois opérateurs génétiques : croisement, mutation et sélection [16]. Il a été démontré que les AG présentent de nombreux avantages par rapport aux algorithmes conventionnels [62, 101], nous en citons :

1. Ils sont sans-gradient : aucune information sur le gradient ou la dérivée n'est requise ; donc, nous pouvons traiter des problèmes complexes et discontinus.
2. Ils possèdent une grande capacité d'exploration de l'espace de recherche : la nature stochastique du croisement et de la mutation permet d'explorer plus efficacement l'espace de recherche, et l'optimum global est plus accessible.
3. Ils sont parallèles : une population d'individus (i.e., solutions) essaiment dans l'espace de recherche de manière parallèle.

Les trois principaux opérateurs évolutionnaires des algorithmes génétiques sont résumés comme suit :

1. Croisement : combiner deux individus parents en échangeant des portions de leurs chromosomes, afin de produire de nouveaux individus.
2. Mutation : le changement d'une partie d'un chromosome pour générer de nouvelles caractéristiques. Par exemple, dans un codage binaire, une mutation peut être obtenue en remplaçant 0 par 1 et vice-versa. La mutation peut être appliquée sur un seul gène ou plusieurs gènes simultanément.
3. Sélection : la survie des plus aptes, ce qui signifie que les individus ayant les meilleures qualités sont gardés en passent d'une génération à l'autre (i.e., élitisme).

Le croisement peut être vu comme une recherche locale appliquée à un sous-espace [109, 110]. Par un exemple, nous considérons un problème avec un espace de recherche $\Omega = \mathbb{B}^8$ (i.e., $d = 8$). Si les individus parents sont : $\mathbf{x}_1 = [11111100]$ et $\mathbf{x}_2 = [11111111]$, alors le croisement nous donne quatre combinaisons possibles : $[11111101]$, $[11111110]$, $[11111111]$ ou $[11111100]$. Dans tous les cas, les six premiers gènes sont toujours les mêmes (i.e., $[111111]$), et cela signifie que le croisement mènera à une solution dans un sous-espace où seules les variables de décision 7 et 8 sont différentes. En d'autres termes, l'opérateur de croisement ne crée des solutions que dans le sous-espace $S = [111111] \cup \mathbb{B}^2 \subset \Omega$. Par conséquent, le croisement est un opérateur de recherche locale, bien qu'il puisse également devenir un opérateur de recherche globale si le sous-espace est suffisamment grand.

La mutation fournit un mécanisme d'exploration globale. Dans l'exemple précédent, si nous mutons l'une des solutions dans la première dimension, cela générerait une solution qui probablement ne se trouve pas dans le sous-espace S. Par exemple, pour $\mathbf{x} = [11111100] \in S$, si la valeur de son premier gène devient 0, alors nous avons une nouvelle solution $\mathbf{x}' = [01111100] \notin S$. En fait, la nouvelle solution peut être différente ou égale aux solutions existantes ou précédentes. Pour cette raison, la mutation est un opérateur de recherche globale. Toutefois, la mutation peut être locale si le taux de mutation est suffisamment faible ou les longueurs de pas sont trop petites (e.g., cas des gènes à valeurs réelles). Par conséquent, la frontière entre la recherche locale et globale peut être vague et relative. La mutation peut également prendre des formes différentes, et l'une des façons simples consiste à utiliser des mouvements stochastiques (i.e., randomisation).

Le croisement et la mutation fournissent la diversité pour de nouvelles solutions. Le croisement peut fournir de bonnes combinaisons, et sa diversité est principalement limitée dans des sous-espaces. La mutation peut fournir une meilleure diversité, bien qu'il y ait des solutions qui peuvent faire diverger la population de l'optimum global.

L'opérateur de sélection possède deux rôles : (i) choisir les meilleurs individus ; et (ii) fournir un moyen de convergence. En d'autres termes, sans sélection les AG sont dépourvus de toutes capacités de choisir ce qui est mieux (i.e., la sélection permet aux AG d'évoluer avec un objectif). Avec un mécanisme de sélection

approprié, seuls les meilleurs individus passent d'une génération à l'autre ; alors que les mauvais individus disparaissent progressivement. Par exemple, la sélection peut être un élitisme simple, où seul le meilleur individu est sélectionné. De toute évidence, d'autres mécanismes de sélection existent dans la littérature [111].

2.3 Algorithmes inspirés par la nature

Dans la littérature consacrée à l'optimisation, il existe plusieurs métaheuristiques inspirées par la nature. Dans cette section, nous analysons quelques algorithmes populaires, en termes d'opérateurs évolutionnaires, exploration et exploitation.

2.3.1 Recuit simulé

La métaheuristique la plus simple est probablement le recuit simulé [92]. Elle s'inspire des caractéristiques du procédé de recuit des métaux. A partir d'une solution courante x_i, une nouvelle solution x_j est acceptée avec la probabilité suivante :

$$P(x_i \rightarrow x_j | x(t) = x_i) = \frac{1}{Z} e^{-\frac{1}{T(t)} \max\{0, f(x_j) - f(x_i)\}} \tag{2.8}$$

où f est la fonction objective à minimiser ; et Z est un facteur de normalisation.

L'article souche de cette métaheuristique [92] a démontré comment résoudre des problèmes très difficiles. En plus, la génération de nouvelles solutions x_j à partir de la solution actuelle x_i peut dépendre de l'implémentation et du problème d'intérêt. Les générations de nouvelles solutions forment une chaîne de Markov ou marche aléatoire.

L'opérateur principal est la génération de nouvelles solutions par marches aléatoires (i.e., la randomisation agit comme un mécanisme d'exploration). La sélection est mise en œuvre en testant si une solution est améliorée ou non. Le recuit simulé ne possède pas un opérateur de croisement. De plus, sa capacité d'exploitation est relativement faible, car l'acceptation est faite en utilisant des probabilités. Dans la pratique, le recuit simulé a souvent une convergence très lente, possède une bonne capacité d'exploration et l'optimum global est toujours atteint après un grand nombre d'itérations.

2.3.2 Algorithmes génétiques

Les AG [16] forment essentiellement les fondements de l'informatique évolutionnaire moderne. Ils possèdent trois opérateurs génétiques : croisement, mutation et sélection. Le croisement permet d'exploiter l'espace de recherche et améliorer la convergence. La mutation permet d'explorer l'espace de recherche et atteindre l'optimum global. Les résultats empiriques et études théoriques suggèrent des probabilités relativement élevées pour le croisement ($[0.6, 0.95]$) et faibles pour la mutation ($[0.001, 0.05]$) : i.e., une forte exploitation et faible exploration. En pratique, la convergence des AG est souvent garantie et l'optimum global est facilement atteint. La sélection fournit un bon mécanisme pour garder les meilleures solutions : i.e., nous passons les meilleures solutions de génération à l'autre, ce qui améliore la convergence des AG. La mutation augmente la probabilité de trouver l'optimum global et évite la convergence prématurée.

2.3.3 Evolution différentielle

L'évolution différentielle [95, 112] présente une forte similitude avec la mutation dans la recherche de patrons [113] en optimisation. La mutation dans l'évolution différentielle est vue comme une recherche de patrons, généralisée dans n'importe quelle direction aléatoire.

$$\mathbf{x}_i = \mathbf{x}_r + F(\mathbf{x}_p - \mathbf{x}_q) \tag{2.9}$$

où F est le poids différentiel ($F \in [0,2]$) ; et les entiers naturels r, p, q et i sont générés par permutation aléatoire [114] ($r \neq p \neq q \neq i$).

L'évolution différentielle possède également un opérateur de croisement, contrôlé par une probabilité de croisement $C \in [0,1]$. Le croisement est réalisé de deux manières : binomial et exponentiel. La sélection est la même que celle utilisée dans les AG.

$$\mathbf{x}_i^{t+1} = \begin{cases} \mathbf{u}_i^{t+1} & , \quad \text{si } f(\mathbf{u}_i^{t+1}) \leq f(\mathbf{x}_i^{t+1}) \\ \mathbf{x}_i^t & , \quad \text{sinon} \end{cases} \tag{2.10}$$

La plupart des études se sont concentrées sur le choix de F, C, la taille de la population et la modification du schéma de mutation. Les variantes de l'évolution différentielle utilisent croisement, mutation et sélection ; les principales différences sont dans la mutation et le croisement (il existe plus de 10 variantes [115]).

2.3.4 Algorithmes de fourmis et abeilles

L'optimisation par colonie de fourmis [14, 93] imite le comportement de recherche des fourmis sociales. La phéromone est utilisée comme un moyen de communication, et sa concentration indique la qualité des solutions. Dans les problèmes d'optimisation combinatoires, les qualités des solutions (i.e., chemins) sont liées à leurs concentrations en phéromones, ce qui conduit au choix des routes ayant de fortes concentrations.

La mutation peut être vue comme la génération de chemins aléatoires. La sélection favorise le choix des plus courts chemins, en utilisant la concentration en phéromone. Le croisement n'est pas explicite. Par exemple, dans un problème de routage, une fourmi se trouvant à un nœud i choisit un autre nœud j selon la règle donnée par l'équation 2.11.

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta}}{\sum_{p=1}^{n} \tau_{ip}^{\alpha} \eta_{ip}^{\beta}} \tag{2.11}$$

où $\alpha > 0$ et $\beta > 0$ sont des paramètres d'influence ; τ_{ij} est la concentration en phéromone du segment (i,j) ; et η_{ij} est la désirabilité du même segment. La sélection est usuellement liée à certaines connaissances a priori sur l'itinéraire (e.g., $\eta_{ij} \propto \frac{1}{d_{ij}}$, avec d_{ij} est la longueur du segment (i,j)).

En optimisation par colonie d'abeilles [116], les abeilles sont divisées en trois groupes : employées, spectatrices et scoutes. La randomisation (i.e., mutation) est effectuée par les scoutes et employées. La sélection est liée aux qualités des solutions. Le croisement n'est pas explicite.

Les deux métaheuristiques utilisent mutation et sélection (basées sur la qualité des solutions). Elles possèdent une bonne capacité de recherche globale et peuvent efficacement explorer l'espace de recherche. Toutefois, la convergence est relativement lente car le croisement n'est pas explicite (i.e., la capacité d'exploitation d'un sous-espace est très limitée). Il est important de souligner que le manque de croisement est commun dans plusieurs métaheuristiques.

2.3.5 Optimisation par essaims particulaires

L'optimisation par essaims particulaires [49] est basée sur le comportement des troupeaux dans la nature (e.g., poissons et oiseaux). La vélocité d'une particule $\mathbf{v}_i$ et sa position $\mathbf{x}_i$ sont mises à jour en utilisant les équations 2.12 et 2.13, respectivement.

$$\mathbf{v}_i^{t+1} = w_i \mathbf{v}_i^t + \alpha \varepsilon_1 (\mathbf{g}^* - \mathbf{x}_i^t) + \beta \varepsilon_2 (\mathbf{x}_i^* - \mathbf{x}_i^t) \tag{2.12}$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1} \tag{2.13}$$

où w_i est l'inertie de chaque particule ; $\mathbf{x}_i^*$ est la position de la meilleure particule dans la population actuelle ; $\mathbf{g}^*$ est la position de la meilleure particule dans les populations antérieures ; ε_1 est un poids cognitif représentant la pensée d'une particule ; ε_2 est un poids social représentant la collaboration entre les particules ; α et β sont des valeurs aléatoires entre 0 et 1.

En comparant les équations 2.12 et 2.13 avec la recherche de patrons [113], nous verrons que la nouvelle position est générée par mutation ; tandis que la sélection est implicitement mise en place en utilisant les meilleures solutions $\mathbf{g}^*$ et $\mathbf{x}_i^*$. Le rôle de la solution $\mathbf{x}_i^*$ n'est pas tout à fait clair, car la solution $\mathbf{g}^*$ semble très importante pour la sélection : optimisation par essaims particulaires accélérée [49, 62, 117].

L'optimisation par essaims particulaires consiste principalement en mutation et sélection ; et il n'y a pas de croisement, ce qui donne une très haute capacité d'exploration à cette métaheuristique. L'utilisation de $\mathbf{g}^*$ est une épée à double tranchant : elle permet d'accélérer la convergence en se rapprochant de l'optimum global ; mais cela conduit souvent à une convergence prématurée vers un optimum local.

2.3.6 Algorithme des lucioles

L'algorithme des lucioles [50, 62, 118] imite le comportement lumineux des lucioles tropicales. Il s'agit d'une métaheuristique simple, flexible et facile à mettre en œuvre. Le mouvement d'une luciole i, attirée par une autre luciole j (plus lumineuse), est donné par l'équation 2.14.

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \beta_0 e^{-\gamma r_{ij}^2} (\mathbf{x}_j^t - \mathbf{x}_i^t) + \alpha \mathbf{s}_i^t \tag{2.14}$$

où le deuxième terme est l'attraction ; β_0 est un facteur d'attractivité ; r_{ij} est la distance entre les lucioles i et j ; le troisième terme est une randomisation ; α est un paramètre de randomisation ; et $\mathbf{s}_i^t$ est un vecteur de nombres aléatoires tirés d'une distribution gaussienne ou de Lévy [50, 118]. Les variantes de cette métaheuristique peuvent être trouvées dans les travaux [99, 119].

D'après l'équation 2.14, nous verrons que la mutation est utilisée pour faire la recherche locale et globale à la fois. Si $\mathbf{s}_i^t$ est tiré d'une distribution gaussienne ou de Lévy, alors nous avons une recherche globale ; en revanche, si α est très petite, alors la recherche est locale. Il n'y a pas de sélection explicite dans l'équation 2.14, car l'optimum global n'est pas utilisé, mais des comparaisons sont utilisées avant de faire des mouvements.

L'algorithme des lucioles utilise l'attraction. A la fin, la population est partitionnée en plusieurs sous-groupes. Les lucioles d'un même sous-groupe essaiment autour d'un mode local. Parmi tous les modes locaux, il existe toujours une meilleure solution qui est le véritable optimum global (cette métaheuristique est très efficace pour les problèmes multimodaux).

D'après l'équation 2.14, nous avons les observations suivantes : (i) si $\gamma = 0$ et $\alpha = 0$, alors nous avons une évolution différentielle ; (ii) si $\beta_0 = 0$, alors nous avons un recuit simulé ; (iii) si $\mathbf{x}_j^t = \mathbf{g}^*$, alors nous avons une optimisation par essaims particulaires accélérée. Par conséquent, ces métaheuristiques sont des cas particuliers de l'algorithme des lucioles.

2.3.7 Recherche de coucou

La recherche de coucou [120] est basée sur le parasitisme de couvées de certaines espèces de coucous. Elle utilise les vols de Lévy [121] à la place de simples marches aléatoires isotropes [1]. Des études récentes ont montré que cette métaheuristique est plus efficace que l'optimisation par essaims particulaires et les AG [101, 122-125]. La recherche de coucou utilise un paramètre de commutation p_a pour combiner deux types de marches aléatoires : locales et globales. Les équations 2.15 et 2.16 expriment, respectivement, ces marches aléatoires.

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \alpha_s \otimes H(p_a - \varepsilon) \otimes (\mathbf{x}_j^t - \mathbf{x}_k^t) \tag{2.15}$$

où $\mathbf{x}_j^t$ et $\mathbf{x}_k^t$ sont deux solutions choisies par permutation aléatoire ; $H(u)$ est la fonction de Heaviside [126] ; ε est un nombre aléatoire de distribution uniforme ; et α_s est la longueur du pas. L'opérateur $\otimes$ représente le produit de Hadamard [127].

$$\begin{cases} \mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \alpha L(s,\lambda) \\ L(s,\lambda) = \frac{\lambda\Gamma(\lambda)\sin(\pi\lambda/2)}{\pi} \frac{1}{s^{1+\lambda}} \quad , \quad (s \gg s_0 > 0) \end{cases} \tag{2.16}$$

où α est un facteur d'amplification de la longueur du pas (il est lié au problème considéré) ; et $\Gamma(u)$ la fonction Gamma [128].

La recherche de coucou présente deux avantages majeurs : marches aléatoires efficaces et bon équilibrage entre recherche locale et globale. Les vols de Lévy [121] sont généralement beaucoup plus efficaces que toutes les autres techniques de randomisation basées sur les marches aléatoires [62]. Par conséquent, la recherche de coucou est très efficace en recherche globale [129].

L'équation 2.16 est le recuit simulé généralisé dans le cadre des chaînes de Markov. Dans l'équation 2.15, (i) si $p_a = 1$ et $\alpha_s \in [0,1]$, alors nous avons une variante d'évolution différentielle ; (ii) si $\mathbf{x}_j^t = \mathbf{g}^*$, alors nous avons une optimisation par essaims particulaires accélérée [117]. Par conséquent, le recuit simulé, l'évolution différentielle et l'optimisation par essaims particulaires accélérée sont des cas particuliers de la recherche de coucou.

2.3.8 Algorithme des chauves-souris

L'algorithme des chauve-souris [130] s'inspire du comportement d'écholocalisation des petits chauve-souris. Chaque chauve-souris possède une vélocité $\mathbf{v}_i^t$ et à une localisation $\mathbf{x}_i^t$. Les équations 2.17, 2.18 et 2.19 expriment la manière de déplacement des chauve-souris dans l'espace de recherche.

$$f_i = f_{\min} + (f_{\max} - f_{\min})\beta \tag{2.17}$$

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + (\mathbf{x}_i^t - \mathbf{x}^*) f_i \tag{2.18}$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1} \tag{2.19}$$

où $\beta \in [0,1]$ est un nombre aléatoire tiré d'une distribution uniforme.

Le réglage des fréquences agit comme une mutation. La sélection est relativement constante, car nous utilisons l'optimum $\mathbf{x}^*$. Il n'y a pas de croisement explicite. Les variations des intensités sonores et taux d'émission des impulsions offrent une capacité de zoom automatique pour que l'exploitation devienne intensive à mesure que la recherche s'approche de l'optimum global.

1. L'isotropie caractérise l'invariance des propriétés physiques d'un milieu en fonction de la direction.

2.3.9 Recherche d'harmonie

La recherche d'harmonie est une métaheuristique inspirée par la composition des morceaux de musiques [131, 132]. Les solutions sont représentées en termes de population d'harmonies, en utilisant trois règles : (i) un musicien joue un morceau musical célèbre ; (ii) un musicien joue un morceau similaire à un morceau connu ; et (iii) un musicien compose de nouveaux morceaux. Principalement, la mutation et sélection sont utilisées ; tandis que le croisement n'est pas explicite. La première règle correspond à la sélection, et les deux dernières règles correspondent à la mutation. La mutation peut être locale ou globale. Par exemple, la deuxième règle utilise l'équation suivante :

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + b_w \varepsilon \tag{2.20}$$

où b_w est la largeur de bande de l'ajustement de la hauteur ; et ε est un nombre aléatoire compris entre -1 et 1. Il s'agit d'une marche aléatoire locale et sa longueur est contrôlée par la bande passante. Cette partie peut être considérée comme une mutation locale avec un taux variant de 0.1 à 0.3. La troisième règle est essentiellement une mutation globale. La sélection est contrôlée par la probabilité de choisir une harmonie dans la mémoire.

2.3.10 Algorithme de pollinisation des fleurs

L'algorithme de pollinisation des fleurs [133, 134] est inspiré par la pollinisation des fleurs dans la nature. Cette métaheuristique utilise les quatre règles suivantes :

1. La pollinisation croisée est considérée comme un processus de pollinisation globale, et les pollinisateurs (porteurs du pollen) se déplacent suivant un vol de Lévy [121].
2. Pour la pollinisation locale, l'autofécondation est utilisée.
3. Les pollinisateurs peuvent développer une constance florale, ce qui est équivalent à une probabilité de reproduction proportionnelle à la similitude de deux fleurs.
4. La commutation de la pollinisation locale et globale peut être contrôlée par une probabilité de commutation $p \in [0,1]$, avec un léger biais vers la pollinisation locale.

Dans la pollinisation globale, les gamètes du pollen sont transportés par les pollinisateurs sur de longues distances. La règle 1 est mathématiquement représentée par l'équation 2.21.

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \gamma L(\lambda)(\mathbf{g}^* - \mathbf{x}_i^t) \tag{2.21}$$

où γ est un facteur contrôlant la longueur du pas ; le paramètre $L(\lambda)$ correspond à la force de la pollinisation. Comme les insectes peuvent se déplacer sur de longues distances, alors nous utilisons un vol de Lévy [121] pour imiter efficacement cette caractéristique.

$$L(\lambda) \sim \frac{\lambda \Gamma(\lambda) \sin(\pi\lambda/2)}{\pi} \frac{1}{s^{1+\lambda}}, (s \gg s_0 > 0) \tag{2.22}$$

Cette étape est essentiellement une mutation globale, ce qui nous permet d'explorer efficacement l'espace de recherche. Pour la pollinisation locale, les règles 2 et 3 peuvent être représentées par l'équation 2.23.

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \varepsilon(\mathbf{x}_j^t - \mathbf{x}_k^t) \tag{2.23}$$

où $\mathbf{x}_j^t$ et $\mathbf{x}_k^t$ sont des fleurs différentes de la même espèce végétale. Nous imitons la constance florale dans un voisinage limité. Mathématiquement, si $\mathbf{x}_j^t$ et $\mathbf{x}_k^t$ proviennent de la même espèce ou sélectionnés dans la même population, alors nous avons une marche aléatoire locale, lorsque le paramètre ε est tiré d'une distribution uniforme dans l'intervalle $[0,1]$. Il s'agit d'une recombinaison locale, qui peut aider à converger dans un sous-espace.

En principe, la pollinisation peut se produire à toutes les échelles (i.e., locale et globale). Toutefois, une fleur donnée est plus susceptible d'être pollinisée par le pollen des fleurs adjacentes. Pour imiter cette caractéristique, nous utilisons une probabilité de commutation p (règle 4), pour basculer entre pollinisation globale (diversification) et pollinisation locale (intensification). La sélection est faite en choisissant les meilleures solutions et en les transmettant à la prochaine génération (nous utilisant explicitement $\mathbf{g}^*$). Il n'y a pas de croisement explicite.

2.3.11 Autres algorithmes

Il existe également de nombreuses métaheuristiques dans la littérature. Les travaux dans [99, 135] fournissent un aperçu sur les algorithmes récents. Les tables 2.1 et 2.2 énumèrent les métaheuristiques populaires (nous avons gardé les appellations englaises). En se basant sur la source d'inspiration des différentes métaheuristiques, nous trouvons principalement quatre familles :

1. Algorithmes évolutionnaires : ils sont inspirés par le processus de l'évolution naturelle (sélection, reproduction, croisement et mutation).
2. Algorithmes basés sur les essaims : ils imitent le comportement intelligent des troupeaux d'animaux sociaux.
3. Algorithmes basés sur les lois physiques et chimiques : ils imitent les lois gouvernant les sciences physiques et chimiques.
4. Algorithmes basés sur les relations humaines : ils sont basés sur les interactions entre les humains.

2.3.12 Analyse et critiques

En analysant les métaheuristiques ci-dessus, nous remarquons que la mutation et sélection sont toujours utilisées ; alors que le croisement est quasiment omis dans la majorité de ces algorithmes. Cela nous incite à poser la question suivante : quel est le rôle exact du croisement ?

Le croisement améliore généralement la convergence d'une métaheuristique vers un sous-espace ; cependant, si son taux est trop élevé, alors ceci peut conduire à une convergence prématurée. D'un autre côté, trop d'exploration par mutation peut ralentir la convergence. Le manque de croisement explicite dans de nombreux algorithmes peut expliquer leurs capacités à converger vers l'optimum global, mais le nombre d'itérations est significativement considérable par rapport à celui des métaheuristiques utilisant le croisement.

2.4 Réglage et contrôle de paramètres

Toutes les métaheuristiques possèdent des paramètres. Le réglage et contrôle de paramètres peuvent largement influencer le comportement et les performances d'un algorithme. La meilleure façon de régler et contrôler ces paramètres est un problème très difficile [205].

2.4.1 Réglage de paramètres

Pour régler $A(\Phi, \mathbf{p})$ (i.e., un algorithme A pour un problème Φ, avec un ensemble de paramètres $\mathbf{p}$) de manière à obtenir les meilleures performances, alors un réglage de paramètres (i.e., leur donner des valeurs adéquates) est nécessaire. Ainsi, ceci nous pousse à répondre à la question suivante : quels sont les

TABLE 2.1 – Métaheuristiques populaires évolutionnaires et bassées sur les essaims.

Métaheuristiques évolutionnaires	Métaheuristiques bassées sur les essaims
Genetic algorithm [136]	Particle swarm optimization [137]
Evolution strategies [138]	Ant colony optimization [14]
Evolutionary programming [84]	Artificial bee colony [139]
Genetic programming [15]	Grey wolf optimizer [140]
Differential evolution [95]	Bat algorithm [130]
Biogeography-based optimization [141]	Whale optimization algorithm [142]
Covariance matrix adaptation evolution strategy [143]	Dragonfly algorithm [144]
Quantum-inspired evolutionary algorithm [145]	Dolphin echolocation [146]
	Fruit fly optimization [147]
	Krill herd [148]
	Bird mating optimizer [149]
	Hunting search [150]
	Firefly algorithm [118]
	Dolphin partner optimization [151]
	Cuckoo search [120]
	Social spider optimization [152]
	Bee collecting pollen algorithm [153]
	Marriage in honey bees [154]
	Monkey search [155]
	Termite [156]
	Fish swarm algorithm [157]
	Grasshopper optimisation algorithm [158]
	Seagull optimization algorithm [159]
	Salp swarm algorithm [160]
	Selfish herd optimizer [161]
	Moth-flame optimization algorithm [162]
	Ant lion optimizer [163]
	Harris hawks optimization [20]
	Slime mould algorithm [164]
	Moth search algorithm [165]
	Elephant herding optimization [166]
	Earthworm optimisation algorithm [167]
	Monarch butterfly optimization algorithm [168]
	Rooted tree optimization algorithm [169]
	Tunicate swarm algorithm [170]

TABLE 2.2 – Métaheuristiques populaires basées sur les lois de la physique et les interactions humaines.

Métaheuristiques basées sur les lois de la physique	Métaheuristiques basées sur les interactions humaines
Simulated annealing [92]	Teaching learning-based optimization [171]
Thermodynamic laws [172]	Harmony search [131]
Gravitation [173, 174]	Taboo search [175]
Big bang–big crunch[176]	Group search optimizer [177]
Charged system [178]	Imperialist competitive algorithm [179]
Central force [180]	League championship algorithm [181]
Chemical reaction [182]	Colliding bodies optimization [183]
Black hole [184]	Interior search algorithm [185]
Ray [186]	Mine blast algorithm [187]
Small-world [188]	Soccer league competition algorithm [189]
Galaxy-based [190]	Seeker optimization algorithm [191]
General relativity theory [192]	Social-based algorithm [193]
Sine cosine algorithm [194]	Exchange market algorithm [195]
Multi-verse optimizer [196]	Nomadic people optimizer [197]
Inclined planes system optimization [198]	Group counseling optimization algorithm [199]
Firework algorithm [200]	
Modified inclined planes system optimization [201]	
Simplified inclined planes system optimization [202]	
Spherical search [203]	
Solar system algorithm [204]	

outils requis pour régler les paramètres d'un algorithme ? Une façon simple consiste à utiliser un meilleur algorithme B pour régler l'algorithme A. Maintenant, nous avons trois questions : (i) comment savoir si B est meilleur ? (ii) l'algorithme B est-il bien réglé ? (iii) si l'algorithme B est bien réglé, alors comment son réglage a été fait en premier lieu ? Naïvement, si nous utilisons un autre algorithme C pour régler B, alors encore une autre fois nous avons les mêmes questions, et cela peut continuer indéfiniment jusqu'à la fin d'une longue chaîne : disons un algorithme Q. En fin de compte, nous avons besoin d'un algorithme pour régler Q, ce qui revient à nouveau à la question d'origine : comment régler l'algorithme Q pour qu'il donne les meilleures performances ?

Il convient de souligner que même si nous avons les bons outils pour régler un algorithme, alors les meilleurs réglages de paramètres dépendent largement des métriques adoptées pour quantifier les performances. Idéalement, le réglage de paramètres devrait être suffisamment robuste pour gérer les (i) changements mineurs dans les valeurs des paramètres, (ii) randomisations et (iii) instances du problème [205] ; ce qui constitue un vrai défi dans la pratique : i.e., il est très difficile de satisfaire simultanément les trois contraintes. Le réglage de paramètres peut être : (i) itératif/non-itératif et (ii) mono-stage/multi-stage [205]. La signification de ces termes est auto-explicative. Parmi les méthodes de réglage nous trouvons : méthodes d'échantillonnage, méthodes de dépistage, méthodes basées-modèles et métaheuristiques. Le succès et l'efficacité de ces méthodes peuvent varier, et il n'existe donc aucune méthode bien établie pour le réglage universel de paramètres.

2.4.2 Hyper-optimisation

D'après la section précédente, il est admis que le réglage de paramètres est le processus d'optimisation d'un algorithme d'optimisation ; c'est donc un problème d'hyper-optimisation. Pour un problème d'optimisation standard sans contraintes, le but est de trouver l'optimum global f_* d'une fonction $f(\mathbf{x})$ dans un espace ayant d composantes.

Une fois que nous avons choisi un algorithme A pour résoudre ce problème d'optimisation, systémati-

quement l'algorithme trouverait une solution $f_{\min}$ qui peut être proche du vrai optimum global f_*. Cela peut nécessiter un certain nombre d'itérations t_δ pour atteindre $|f_{\min} - f_*| \leq \delta$ (δ étant une précision). De toute évidence, le nombre d'itérations réel dépendrait à la fois de la fonction objective $f(\mathbf{x})$ et des paramètres $\mathbf{p}$ de l'algorithme utilisé.

Le but principal de l'optimisation des algorithmes d'optimisation est de trouver le meilleur paramétrage $\mathbf{p}_*$, de sorte que le nombre d'itérations t_δ soit minimum. Ainsi, le réglage de paramètres est un problème d'hyper-optimisation qui peut être décrit de la manière suivante :

$$\text{Minimiser } t_\delta = A(f(\mathbf{x}), \mathbf{p}) \tag{2.24}$$

dont le minimum global est $\mathbf{p}_*$.

Idéalement, le vecteur de paramètres $\mathbf{p}_*$ devrait être suffisamment robuste. Autrement dit, pour différents types de problèmes, les légères variations dans $\mathbf{p}_*$ ne devraient pas grandement affecter les performances de A.

Si nous observons le processus de réglage de paramètres sous un autre angle, alors il est possible de le considérer comme un problème d'optimisation bi-objectives : une fonction objective $f(\mathbf{x})$ du problème Φ et le nombre d'itérations t_δ de l'algorithme A.

$$\begin{cases} \text{Minimiser } f(\mathbf{x}) \\ \text{Minimiser } t_\delta = A(f(\mathbf{x}), \mathbf{p}) \end{cases} \tag{2.25}$$

où t_δ est le nombre d'itérations nécessaires pour atteindre une certaine précision δ, de sorte que le minimum trouvé $f_{\min}$ soit suffisamment proche du vrai minimum global f_*, satisfaisant l'inégalité $|f_{\min} - f_*| \leq \delta$.

Cela signifie que pour une précision δ, il y aura un ensemble de meilleurs réglages de paramètres avec t_δ soit minimum (i.e., nous avons un front de Pareto). Ce problème d'optimisation peut être résolu par toutes les méthodes qui conviennent à l'optimisation multi-objectives. Comme δ est généralement donné, alors un moyen naturel de résoudre ce problème est d'utiliser la méthode ε-contrainte [206]. Pour $\delta \geq 0$, nous changeons le problème donné par l'équation 2.25 comme suit :

$$\begin{aligned} &\text{Minimiser } t_\delta = A(f(\mathbf{x}), \mathbf{p}) \\ &\text{sous la contraintes } f(\mathbf{x}) \leq \delta \end{aligned} \tag{2.26}$$

2.4.3 Contrôle de paramètres

Il existe également un autre problème qui est le contrôle de paramètres. Les valeurs des paramètres, après les réglages, sont souvent statiques durant les itérations. Le contrôle de paramètres permet de faire varier les valeurs des paramètres, pour que l'algorithme puisse atteindre les meilleures performances. Par ailleurs, le contrôle de paramètres est un problème d'optimisation difficile à résoudre. Dans l'algorithme des chauve-souris, un contrôle de paramètres a été utilisé et s'est avéré très efficace [130]. De même, le processus de refroidissement dans le recuit simulé [92] peut être considéré comme un contrôle de paramètres.

2.5 Discussions

Beaucoup d'algorithmes d'optimisation sont basés sur l'intelligence en essaim et utilisent des populations. Il y a des métaheuristiques qui possèdent des opérateurs évolutionnaires : croisement, mutation et sélection. Toutefois, la majorité des algorithmes d'optimisation utilisent uniquement la mutation et sélection. Quant au croisement, cet opérateur apparaît parfois de manière subtile dans certains algorithmes. Le croisement est très efficace pour exploiter l'espace de recherche, et peut souvent fournir une bonne convergence

dans un sous-espace local. Si l'optimum global se trouve dans un sous-espace donné, alors le croisement avec l'élitisme peut fortement garantir l'obtention de cet optimum global. D'un autre côté, si le sous-espace ne contient pas l'optimum global, alors il existe un risque de convergence prématurée.

L'utilisation extensive de la mutation et sélection peut généralement permettre à un algorithme stochastique d'avoir une grande capacité d'exploration. L'exploitation étant relativement faible, le taux de convergence est généralement faible par rapport à celui des méthodes traditionnelles telles que la méthode de Newton-Raphson [42, 43]. Par conséquent, la plupart des métaheuristiques peuvent généralement bien fonctionner pour des problèmes non-linéaires, y compris les optimisations relativement difficiles. Cependant, le nombre d'évaluations de fonctions peut être très élevé.

Le rôle du croisement et de la mutation dans l'exploration est plutôt subtil, tandis que la sélection comme mécanisme d'exploitation peut être simple et pourtant efficace. Cependant, nous ne savons pas toujours comment la combinaison de ces trois opérateurs peut être directement liée à l'équilibre entre exploration et exploitation (c'est toujours une question ouverte). Par exemple, dans les AG, la probabilité de croisement est très élevée (e.g., 0,95) et la probabilité de mutation est très faible (e.g., 0,01). En comparaison avec d'autres métaheuristiques, l'exploration semble faible, mais les AG se sont révélés très efficaces [62]. D'un autre côté, l'usage des vols de Lévy [121], liés à la mutation, dans la recherche de coucou [101] devrait augmenter sa capacité d'exploration, et pourtant cette métaheuristique peut converger très rapidement.

Même dans l'optimisation par essaims particulaires [49], le pourcentage des itérations dans la phase d'exploration n'est pas clair. L'utilisation du meilleur optimum global de la population actuelle peut être une arme à double tranchant : certes, l'usage du meilleur optimum global peut aider à accélérer la convergence, mais peut aussi conduit à un optimum local. Tous ces points suggèrent que nous ne connaissons pas vraiment comment parvenir à un équilibrage optimal entre exploration et exploitation.

En fait, un bon équilibrage, entre exploration et exploitation, ne peut être atteint en réunissant convenablement tous les opérateurs évolutionnaires sans réglage et contrôle adéquats de paramètres. Nous savons bel et bien que le réglage et contrôle de paramètres d'un algorithme peuvent significativement affecter ses performances. Donc, nous devons trouver les bonnes valeurs pour les paramètres au cours des itérations. Le réglage et contrôle de paramètres font toujours couler beaucoup d'encre [205]. Ces observations n'interdissent pas le développement de nouvelles métaheuristiques ; bien au contraire, les scientifiques doivent encourager cet axe de recherche en proposant des algorithmes efficaces en termes de meilleurs opérateurs évolutionnaires et meilleurs équilibrages entre exploration et exploitation.

2.6 Conclusion

Les algorithmes d'optimisation inspirés par la nature présentent des avantages remarquables par rapport aux méthodes traditionnelles. En utilisant les théories des systèmes dynamiques, l'auto-organisation et les chaînes de Markov, nous avons fourni une analyse critique de certaines métaheuristiques. L'analyse s'est concentrée, d'un côté, sur l'exploration et exploitation de l'espace de recherche et, d'un autre côté, sur les opérateurs évolutionnaires (i.e., croisement, mutation et sélection).

Nous avons constaté que la plupart des métaheuristiques utilisent la mutation et la sélection pour explorer et exploiter l'espace de recherche. Certains algorithmes utilisent également le croisement, mais la plupart ne le font pas. La mutation nous aide à faire une recherche globale (exploration). Le croisement nous aide à faire une recherche locale dans un sous-espace donné (exploitation). La sélection fournit un moyen pour favoriser les solutions prometteuses.

Il convient de noter que l'analyse précédente est consacrée aux problèmes d'optimisation continue, et nous pouvons nous attendre à ce que ces métaheuristiques soient toujours valables pour les problèmes

d'optimisation combinatoire. Il faut être prudent dans les problèmes d'optimisation combinatoire, où la notion du voisinage peut avoir une signification différente, et par conséquent le concept de sous-espace peut également être différent.

Dans le prochain chapitre, nous présentons la contribution de ce mémoire : la proposition d'une nouvelle métaheuristique basée sur le comportement de chasse des poissons archers.

Chapitre 3

Algorithme AHO : une nouvelle métaheuristique pour l'optimisation globale

Sommaire

3.1 Introduction

Les algorithmes d'approximation ont été proposés pour la première fois dans les années 1960 afin de résoudre des problèmes d'optimisation difficiles [207]. Ils sont nommés ainsi parce qu'ils génèrent des solutions presque-optimales. Ils ont été principalement utilisés pour traiter des problèmes d'optimisation ne pouvant pas être résolus efficacement à l'aide de techniques de calcul conventionnelles [208]. Le théorème de complétude de Gödel a également apporté une contribution majeure à la maturation de cette famille d'algorithmes, puisque la nécessité de résoudre les problèmes d'optimisation $\mathscr{NP}$-hard est devenue une priorité urgente [209]. Certains problèmes d'optimisation sont faciles à résoudre approximativement (i.e., la génération de solutions sous-optimales est rapide), tandis que pour d'autres cette tâche est aussi difficile que de trouver les solutions optimales [210].

Les algorithmes d'approximation utilisant des techniques probabilistes et randomisées connaissaient d'énormes progrès entre les années 1980 et 1990. Ils ont été nommés algorithmes métaheuristiques [38]. Une liste des premiers algorithmes métaheuristiques peut inclure, à titre indicatif et non exhaustif, recuit simulé [3], optimisation par colonies de fourmis [14], algorithmes évolutionnaires [211], recherche taboue [212], algorithmes mimétiques [213] et optimisation par essaims particulaires [49]. Au cours des trois dernières décennies, de nombreux algorithmes métaheuristiques ont été proposés dans la littérature, dont plusieurs ont été évalués expérimentalement et ont montré de bonnes performances lors de la résolution de problèmes d'optimisation issus du monde réel [214]. Les algorithmes métaheuristiques visent à trouver les meilleures solutions possibles, tout en garantissant que ces dernières répondent à certains critères et contraintes [215]. Le théorème No-Free-Lunch [216] prouvait l'inexistence de métaheuristiques universelles, ce qui justifie le nombre important d'algorithmes proposés dans la littérature [217].

Tous les algorithmes métaheuristiques partagent deux composantes fondamentales : exploration et exploitation de l'espace de recherche [218]. L'exploration est appelée aussi optimisation globale ou diversification. L'exploitation est appelée aussi optimisation locale ou intensification. L'exploration permet aux algorithmes métaheuristiques de découvrir de nouvelles régions de l'espace de recherche et d'éviter d'être piégés dans les optimums locaux [219]. L'exploitation permet aux algorithmes métaheuristiques de concentrer la recherche sur une certaine zone pour en trouver la meilleure solution [219]. Tout algorithme métaheuristique doit trouver le meilleur équilibre entre la diversification et l'intensification ; sinon, la qualité des solutions trouvées est compromise [220]

Une exploration excessive peut entraîner un gaspillage considérable d'efforts (l'algorithme passe d'un endroit à un autre sans se concentrer sur l'amélioration de la qualité des solutions courantes) [221]. Une exploitation excessive peut conduire l'algorithme à être piégé dans des optimums locaux et à converger prématurément [222]. Le principal inconvénient des algorithmes métaheuristiques est la sensibilité au réglage des paramètres de contrôle. De plus, la convergence vers l'optimum global n'est pas toujours garantie [223].

Nous proposons un nouvel algorithme métaheuristique basé-population pour l'optimisation globale. L'algorithme est inspiré par les comportements de tir et de saut des poissons archer dans la nature, lors de la capture des proies. Nous nommons cet algorithme : Archerfish Hunting Optimizer (AHO). Les principales caractéristiques de l'algorithme AHO sont les suivantes :

1. L'algorithme AHO possède trois paramètres de contrôle à définir : i) taille de la population ; ii) angle d'échange entre les phases d'exploration et d'exploitation ; et iii) taux d'attractivité entre le poisson archer et la proie.
2. L'algorithme AHO utilise les lois élémentaires de la physique (équation de la trajectoire d'un projectile) pour déterminer les positions des nouvelles solutions.
3. L'angle d'échange contrôle l'équilibre entre l'exploration et l'exploitation de l'espace de recherche.

La performance de l'algorithme AHO a été évaluée à l'aide du benchmark CEC 2020 pour l'optimisation

sans-contraintes. Ce benchmark contient dix fonctions de test mono-objective : une fonction unimodale, trois fonctions multimodales, trois fonctions hybrides et trois fonctions composites [224]. Les résultats obtenus ont été comparés à 12 algorithmes métaheuristiques récents et les résultats expérimentaux ont été évalués à l'aide de deux tests statistiques : i) test des rangs signés de Wilcoxon et ii) test de Friedman. Les résultats statistiques montrent que l'algorithme AHO est très compétitif par rapport aux algorithmes utilisés pour l'étude comparative.

Le reste du chapitre est organisé comme suit. La section 3.2 illustre le comportement de chasse des poissons archer dans la nature et fournit la source d'inspiration de l'algorithme AHO. La section 3.3 décrit l'algorithme métaheuristique proposé et son modèle mathématique. La section 3.4 présente les résultats statistiques obtenus et analyse l'étude comparative. La section 3.5 résume le chapitre et conclut par quelques perspectives.

3.2 Source d'inspiration

Les poissons archer forment une famille monotypique appelée : Toxotes Chatareus. Ils vivent principalement dans les mangroves du bassin indopacifique [225]. Ils possèdent l'un des comportements d'alimentation les plus complexes et passionnants : ils chassent les insectes aériens en les abattant avec des gouttelettes d'eau éjectées de leur bouche. La figure 3.1 montre la morphologie des poissons archer et leurs mécanismes de chasse. Un poisson archer utilise deux moyens pour capturer des insectes : soit il déloge la cible avec un puissant jet d'eau ou il saute sur la proie si celle-ci est assez proche [226].

FIGURE 3.1 – Mécanismes de chasse des poissons archer.

En pratique, la technique de tir est moins fatigante que le saut. Elle permet de nombreux tirs consécutifs. Cependant, la récupération de la proie est incertaine parce que d'autres poissons archer pourraient la voler [227, 228]. En sautant, un poisson archer se positionne directement sous la cible [229]. En tirant, un poisson archer prend une position plus latérale [230]. Les poissons archer éjectent des gouttelettes d'eau sur les insectes aériens dans le but de les faire tomber sur la surface de l'eau et les manger. Comme les yeux des poissons archer restent entièrement sous la surface de l'eau pendant l'observation et le tir, alors ce dernier doit faire face aux effets de réfraction entre l'air et l'eau [231]. Dans le présent chapitre, nous adoptons les principes suivants pour décrire les instructions de l'algorithme proposé (AHO).

— Les poissons archer vivent en groupes (l'algorithme AHO est une métaheuristique basée-population).
— Les poissons archer utilisent deux mécanismes de chasse : tir et saut (phases d'exploration et d'exploitation).
— Les poissons archer peuvent dérober des proies capturées par les autres (recherche coopérative et partage d'information).
— La commutation entre les comportements de saut et de tir est contrôlé par l'angle de perception (équilibre entre les phases d'exploration et d'exploitation).

3.3 Algorithme métaheuristique proposé

Nous illustrons les phases d'exploration et d'exploitation de l'algorithme AHO, inspiré par les comportements de tir et de saut des poissons archer lors de la capture des insectes. L'algorithme AHO est une méthode d'optimisation sans gradient qui pourrait résoudre tous les problèmes d'optimisation avec une formulation appropriée de la fonction objective. Nous supposons un espace de recherche de dimension d qui contient plusieurs poissons archer. La taille de la population (c'est-à-dire le nombre de poissons archer) est N et la position d'une solution candidate i à l'itération t est donnée comme suit.

$$X^{\langle i,t\rangle} = (x_1, x_2, \ldots, x_d)$$

Chaque composante du vecteur $X^{\langle i,t\rangle}$ a une plage de valeurs autorisées : i.e., $X^{\langle i,t\rangle} = (x_j) \in [x_j^{\min}, x_j^{\max}]$ (où $i \in \{1,\ldots,N\}$ et $j \in \{1,\ldots,d\}$) A l'itération $t = 0$, la position $X^{\langle i,0\rangle}$ est aléatoirement initialisée en utilisant l'équation 3.1.

$$X^{\langle i,0\rangle} = (\alpha_1 \times (x_1^{\max} - x_1^{\min}) + x_1^{\min}, \ldots, \alpha_d \times (x_d^{\max} - x_d^{\min}) + x_d^{\min}) \tag{3.1}$$

Où

$\alpha_1, \ldots, \alpha_d$: Nombres aléatoires à distribution uniforme.

La figure 3.2 illustre le comportement de tir d'un poisson archer (exploration de l'espace de recherche). Le mouvement des gouttelettes d'eau est modélisé à l'aide des équations de la trajectoire balistique d'un projectile [232]. Il est déterminé par l'accélération de la gravité (g), la vitesse de lancement (v) et l'angle de perception (θ_0), (nous supposons que les frictions sont négligées). Nous supposons que la proie (i.e., libellule) est située au sommet de la trajectoire. Lorsque l'insecte est abattu par un poisson k, il tombe verticalement sur la surface de l'eau. Quand un poisson archer détecte les vibrations initiées par la proie, il se déplace vers son emplacement en utilisant l'équation 3.2.

$$X^{\langle i,t+1\rangle} = X^{\langle i,t\rangle} + e^{-\|X_{prey}^{\langle k,t\rangle} - X^{\langle i,t\rangle}\|^2} (X_{prey}^{\langle k,t\rangle} - X^{\langle i,t\rangle}) \tag{3.2}$$

Où

$X^{\langle i,t+1\rangle}$: Prochain emplacement de la solution candidate i.

$X^{\langle i,t\rangle}$: Emplacement actuel de la solution candidate i.

$\|.\|$: Distance euclidienne.

$X_{prey}^{\langle k,t\rangle}$: Position de la proie. Elle est calculée à l'aide de l'équation 3.3.

ε : Vecteur de nombres aléatoires générés par une distribution uniforme. Il représente les effets de réfraction entre l'air et l'eau.

$X^{\langle k,t\rangle}$: Position du poisson archer k, qui a éjecté l'eau.

$$X_{prey}^{\langle k,t\rangle} = X^{\langle k,t\rangle} + (0, \ldots, \frac{v^2}{2g} \times \sin 2\theta_0, \ldots, 0) + \varepsilon \tag{3.3}$$

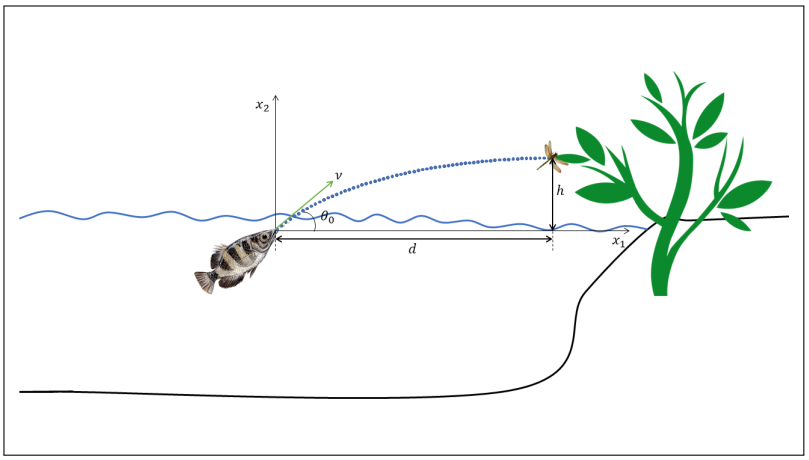


FIGURE 3.2 – Comportement de tir des poissons archer.

La position de la composante donnée par le terme $\frac{v^2}{2g} \times \sin 2\theta_0$ est un nombre aléatoire tiré dans l'ensemble $\{1,\ldots,d\}$. Pour simplifier, la fraction $\frac{v^2}{2g}$ est remplacée par la variable ω. Elle définit le taux d'attractivité entre un poisson archer et une certaine proie.

La figure 3.3 décrit le comportement de saut d'un poisson archer (exploitation de l'espace de recherche). Le poisson archer saute sur la proie et l'attrape. De même, le mouvement du poisson archer est défini par l'accélération de la gravité (g), la vitesse de lancement (v) et l'angle de perception (θ_0), (nous supposons que les frictions sont négligées). Nous supposons que la proie (i.e., libellule) est située au sommet de la trajectoire. Lorsqu'un poisson archer i décide de capturer un insecte, il se déplace vers son emplacement en utilisant l'équation 3.4.

$$X^{\langle i,t+1\rangle} = X^{\langle i,t\rangle} + e^{-\|X^{\langle i,t\rangle}_{prey} - X^{\langle i,t\rangle}\|^2}(X^{\langle i,t\rangle}_{prey} - X^{\langle i,t\rangle}) \tag{3.4}$$

Où

$X^{\langle i,t+1\rangle}$	:	Prochain emplacement de la solution candidate i.
$X^{\langle i,t\rangle}$	:	Emplacement actuel de la solution candidate i.
$\|.\|$	:	Distance euclidienne.
$X^{\langle i,t\rangle}_{prey}$	:	Position de la proie. Elle est calculée à l'aide de l'équation 3.5.
ε	:	Vecteur de nombres aléatoires générés par une distribution uniforme. Il représente les effets de réfraction entre l'air et l'eau.

$$X^{\langle i,t\rangle}_{prey} = X^{\langle i,t\rangle} + (0,\ldots,\frac{v^2}{2g}\times\sin 2\theta_0,\ldots,\frac{v^2}{2g}\times\sin^2\theta_0,\ldots,0) + \varepsilon \tag{3.5}$$

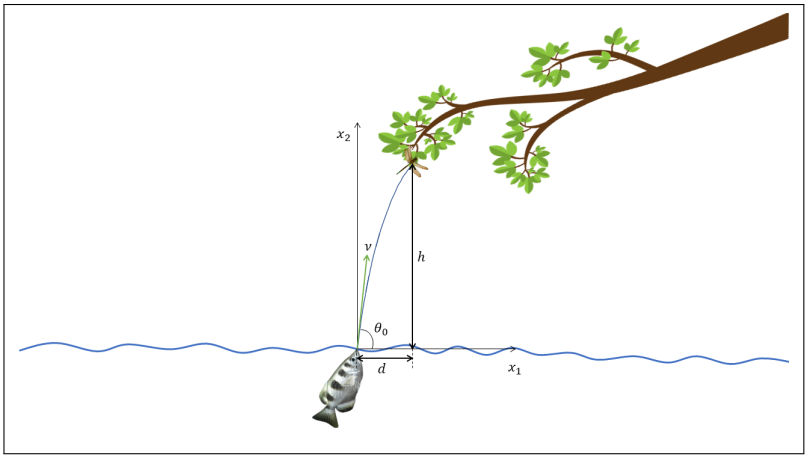


FIGURE 3.3 – Comportement de saut des poissons archer.

Les positions des composantes données par les termes $\frac{v^2}{2g}\times\sin 2\theta_0$ et $\frac{v^2}{2g}\times\sin^2\theta_0$ sont obligatoirement des nombres aléatoires distincts tirés dans l'ensemble $\{1,\ldots,d\}$. Pour simplifier, la fraction $\frac{v^2}{2g}$ est remplacée par la variable ω. Elle définit le taux d'attractivité entre un poisson archer et une certaine proie

La valeur de l'angle de perception (θ_0) garantit la permutation entre les phases d'exploration et d'exploitation. La figure 3.4 délimite les plages d'angles de perception, où l'algorithme AHO est censé explorer (zones vertes) ou exploiter (régions orange) l'espace de recherche. Ainsi, plus la valeur de θ_0 est proche de $\frac{\pi}{2}$ ou $-\frac{\pi}{2}$, plus l'algorithme AHO a tendance à exploiter l'espace de recherche et vice versa. La valeur de θ_0 est générée aléatoirement à l'aide de l'équation 3.6.

$$\theta_0 = (-1)^b \times \alpha \times \pi \tag{3.6}$$

Où

$b \sim \mathfrak{B}(0.5)$	:	Distribution de Bernoulli (probabilité de succès égale à 0,5) [233].
α	:	Nombre aléatoire uniformément distribué entre 0 et 1.
π	:	Constante d'Archimède égale à 3,14.

Pour éviter d'être piégé dans des optimums locaux, l'algorithme AHO utilise une stratégie simple. Si la position d'une solution candidate $X^{\langle i,t \rangle}$ à l'itération t n'a pas été améliorée pour un nombre fixe d'itérations (e.g., $d \times N$), alors dans ce cas cette solution est altérée en utilisant un vol de Lévy [118]. Le nouvel emplacement de $X^{\langle i,t \rangle}$ est généré en utilisant les équations 3.7 et 3.8. L'algorithme 1 illustre le pseudo-code de l'algorithme AHO.

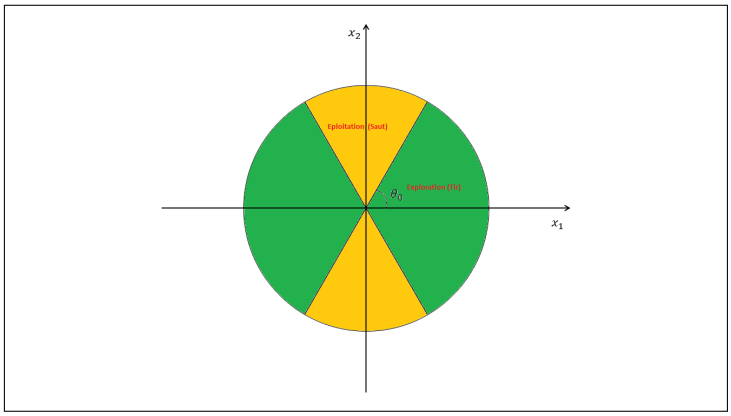


FIGURE 3.4 – Permutation entre l'exploration et l'exploitation.

$$X^{\langle i,t+1 \rangle} = X^{\langle i,t \rangle} + \alpha \left[\frac{u_1}{(v_1)^{1/\beta}}, \ldots, \frac{u_d}{(v_d)^{1/\beta}} \right] \tag{3.7}$$

$$\begin{cases} u_i \sim \mathfrak{N}(0, \sigma^2) & , \quad \sigma = \left(\frac{\Gamma(1+\beta)\sin(\frac{\pi\beta}{2})}{\Gamma(\frac{1+\beta}{2}) \times \beta \times 2^{\frac{\beta-1}{2}}} \right)^{\frac{1}{\beta}} & , \quad i \in \{1, \ldots, d\} \\ v_i \sim \mathfrak{N}(0, \acute{\sigma}^2) & , \quad \acute{\sigma} = 1 & , \quad i \in \{1, \ldots, d\} \end{cases} \tag{3.8}$$

Où

Γ	:	Fonction Gamma [128].
$\mathfrak{N}(\mu, \sigma)$	:	Distribution normale de la moyenne μ et de l'écart type σ [233].
β	:	Indice de la fonction Gamma ($\beta = 1.5$).
α	:	Nombre aléatoire uniformément distribué entre 0 et 1.

La complexité temporelle de l'algorithme AHO dépend des étapes suivantes : initialisation, évaluation de la fonction objective et mouvements des solutions candidates. La complexité de la première étape est $O(N)$. La complexité de la deuxième étape est $O(Iter_{\max} \times N \times d)$. La complexité dela troisième étape est $O(Iter_{\max} \times N \times N)$. Par conséquent, la complexité temporelle de l'algorithme AHO est $O(N \times (Iter_{\max} \times (N+d)+1))$.

3.4 Résultats expérimentaux et discussion

Le benchmark CEC 2020 pour les problèmes d'optimisation sans contraintes [224] est utilisé pour valider l'efficacité de l'algorithme AHO. Ce benchmark est composé de quatre types de fonctions de test : unimodale (UM), basique (BC), hybride (HD) et composite (CM). Les fonctions UM possèdent un seul optimum global. Elles sont utilisées pour évaluer la capacité d'exploitation de l'algorithme AHO. Les fonctions BC possèdent plusieurs optimums globaux. Elles sont utilisées pour évaluer la capacité d'exploration et d'évitement des optimums locaux de l'algorithme AHO. Les fonctions HD et CM sont obtenues à partir de l'hybridation et de la composition de plusieurs fonctions de test élémentaires. Elles sont utilisées pour tester le bon équilibrage entre l'exploration et l'exploitation de l'espace de recherche. La formulation

Input: d (the dimension of the search space).
Input: $[x_1^{\min}, x_1^{\max}], \ldots, [x_d^{\min}, x_d^{\max}]$ (the decision variables' domains).
Input: f (the objective function to be minimized).
Input: θ (the swapping angle between the exploration and exploitation phases).
Input: ω (the attractiveness rate).

for $i \leftarrow 1$ **to** N **do**
  Generate a random location $X^{\langle i,0 \rangle}$ using Equation 3.1;
end
for $t \leftarrow 1$ **to** $Iter_{\max}$ **do**
  for $i \leftarrow 1$ **to** N **do**
    $\theta_0 \leftarrow$ generate a random perceiving angle using Equation 3.6;
    `/* Shooting behavior (exploration of the search space) */`
    if $(|\theta_0| \in]0, \theta[\cup]\pi - \theta, \pi[)$ **then**
      Compute $X_{prey}^{\langle i,t \rangle}$ using Equation 3.3;
      for $j \leftarrow 1$ **to** N **do**
        if $(f(X_{prey}^{\langle i,t \rangle}) < f(X^{\langle j,t \rangle}))$ **then**
          Update the location $X^{\langle j,t \rangle}$ using Equation 3.2, and adjust its components;
        end
        else
          If the location $X^{\langle j,t \rangle}$ has not been changed for a given number of iterations. In this case, generate a new location for $X^{\langle j,t \rangle}$ using Equations 3.7 and 3.8, and adjust its components;
        end
      end
    end
    `/* Jumping behavior (exploitation of the search space) */`
    else
      Compute $X_{prey}^{\langle i,t \rangle}$ using Equation 3.5;
      if $(f(X_{prey}^{\langle i,t \rangle}) < f(X^{\langle i,t \rangle}))$ **then**
        Update the location $X^{\langle i,t \rangle}$ using Equation 3.4, and adjust its components;
      end
      else
        If the location $X^{\langle i,t \rangle}$ has not been changed for a given number of iterations. In this case, generate a new location for $X^{\langle i,t \rangle}$ using Equations 3.7 and 3.8, and adjust its components;
      end
    end
  end
end

Algorithme 1: Pseudo-code de l'algorithme AHO.

mathématique et les caractéristiques des fonctions de test du benchmark CEC 2020 sont disponibles dans [224].

Toutes les expériences ont été exécutées en utilisant le langage de programmation Java sur un poste de travail avec Windows 10 (64 bits) édition familiale. Le processeur est Intel(R) Core(TM) i7–9750H CPU@ 2.60 GHz 2.59 GHz, avec 16 Go de RAM. La dimension de l'espace de recherche (d) varie de 5 à 20 avec un pas d'incrémentation de 5. La taille de la population (N) est $\lfloor 30 \times d^{1.5} \rfloor$, où le terme $\lfloor x \rfloor$ exprime la troncature du nombre réel x. Pour chaque dimension, le nombre maximum d'itérations ($Iter_{\max}$) est égal à $\frac{50000}{N}$, $\frac{1000000}{N}$, $\frac{3000000}{N}$ et $\frac{10000000}{N}$, respectivement. La plage des valeurs autorisées pour chaque variable de décision est $[-100, 100]$. Tous les résultats sont moyennés sur 30 exécutions indépendants pour minimiser les variances. Les valeurs de l'angle de permutation (θ) sont $\frac{\pi}{12}$, $\frac{\pi}{6}$, $\frac{\pi}{4}$, $\frac{\pi}{3}$ et $\frac{5\pi}{12}$; et les valeurs du taux d'attractivité sont 0.01, 0.05, 0.25, 1.25 et 6.25. Par conséquent, nous avons 25 configurations différentes pour chaque dimension. Les tables 3.1, 3.2, 3.3 et 3.4 présentent les valeurs de l'écart-type (STD) pour chaque dimension. Il est à noter que plus la valeur de l'écart-type est proche de 0, plus le résultat est proche de l'optimum global.

Nous effectuons le test de Friedman [234] pour vérifier si le réglage de paramètres de contrôle (i.e., θ et ω) impacte la performance de l'algorithme AHO. Nous considérons chaque dimension séparément (voir les tables 3.1, 3.2, 3.3 et 3.4). Lorsque $d = 5$, nous avons 25 configurations (i.e., traitements) et 8 fonctions de test (i.e., blocs). Lorsque $d = 10$, 15 ou 20, nous avons 25 configurations (i.e., traitements) et 10 fonctions de test (i.e., blocs). La valeur de α est fixée à 0.05 et la valeur du degré de liberté (df) est fixée à 24. Nous définissons respectivement les hypothèses nulle et alternative comme suit : H_0 il n'y a pas de différence entre les 25 configurations pour chaque dimension, et H_1 il y a une différence entre les 25 configurations pour chaque dimension. La valeur critique pour $\alpha = 0.05$ et $df = 24$ est 36.4150 [235]. Nous calculons la valeur F_r en utilisant l'équation 3.9 [234], où n est le nombre de blocs, k est le nombre de traitements et $T_1, T_2, \ldots, T_k$ sont les sommes des rangs pour chaque traitement. Si la valeur F_r est supérieure à 36,4150, alors nous rejetons l'hypothèse H_0.

$$F_r = \frac{12}{nk(k+1)}(T_1^2 + T_2^2 + \ldots + T_k^2) - 3n(k+1) \tag{3.9}$$

— Pour la table 3.1, nous avons $F_r = 66.7419 > 36.4150$. Par conséquent, l'hypothèse H_0 est rejeté. La meilleure configuration est lorsque $\theta = \frac{\pi}{12}$ et $\omega = 0.01$, car son rang est le plus petit avec la valeur 57.5.
— Pour la table 3.2, nous avons $F_r = 104.2532 > 36.4150$. Par conséquent, l'hypothèse H_0 est rejetée. La meilleure configuration est lorsque $\theta = \frac{5\pi}{12}$ et $\omega = 0.01$, car son rang est le plus petit avec la valeur 68.
— Pour la table 3.3, nous avons $F_r = 125.8588 > 36.4150$. Par conséquent, l'hypothèse H_0 est rejetée. La meilleure configuration est lorsque $\theta = \frac{\pi}{3}$ et $\omega = 0.01$, car son rang est le plus petit avec la valeur 58.
— Pour la table 3.4, nous avons $F_r = 144.8797 > 36.4150$. Par conséquent, l'hypothèse H_0 est rejetée. La meilleure configuration est lorsque $\theta = \frac{5\pi}{12}$ et $\omega = 0.01$, car son rang est le plus petit avec la valeur 52.

À partir des tables 3.1, 3.2, 3.3 et 3.4, nous observons que la dimension du problème n'a pas un impact significatif sur la qualité des résultats obtenus. Par ailleurs, nous remarquons que les valeurs de l'angle de perception et du taux d'attractivité n'influencent pas grandement les performances de l'algorithme AHO. Dans toutes les configurations, nous observons que l'algorithme AHO donnent des résultats satisfaisables (les performances restent presque les mêmes pour diverses variables de décision et plusieurs valeurs de ses paramètres de contrôle). Nous pensons qu'un tel comportement est justifié par le théorème de la loi faible des grands nombres en théorie des probabilités [236]. En d'autres termes, si le nombre de générations est suffisamment grand, l'algorithme AHO tend à atteindre un équilibre équitable entre les phases d'exploration

TABLE 3.1 – Valeurs de l'écart-type pour la dimension $d = 5$.

		F_1	F_2	F_3	F_4	F_5	F_8	F_9	F_{10}
$\theta = \frac{\pi}{12}$	$\omega = 0.01$	0,00E+00	6,36E-05	0,00E+00	0,00E+00	1,94E-05	5,59E-05	2,04E-08	1,46E-03
	$\omega = 0.05$	0,00E+00	6,36E-05	0,00E+00	0,00E+00	2,77E-05	5,59E-05	1,33E-07	1,96E-03
	$\omega = 0.25$	1,78E-07	6,36E-05	0,00E+00	0,00E+00	5,20E-05	5,59E-05	4,28E-07	3,42E-03
	$\omega = 1.25$	1,52E-06	6,36E-05	3,12E-08	0,00E+00	9,38E-05	5,59E-05	3,66E-06	3,16E-03
	$\omega = 6.25$	1,16E-05	6,36E-05	1,58E-06	0,00E+00	1,66E-04	5,59E-05	1,37E-05	4,35E-03
$\theta = \frac{\pi}{6}$	$\omega = 0.01$	0,00E+00	6,36E-05	0,00E+00	0,00E+00	2,14E-05	5,59E-05	1,75E-08	1,80E-03
	$\omega = 0.05$	1,41E-08	6,36E-05	0,00E+00	0,00E+00	3,27E-05	5,59E-05	9,89E-08	2,51E-03
	$\omega = 0.25$	1,44E-07	6,36E-05	0,00E+00	0,00E+00	4,83E-05	5,59E-05	3,54E-07	2,57E-03
	$\omega = 1.25$	1,43E-06	6,36E-05	3,02E-08	0,00E+00	9,55E-05	5,59E-05	3,71E-06	4,11E-03
	$\omega = 6.25$	1,01E-05	6,36E-05	1,06E-06	0,00E+00	1,61E-04	5,59E-05	5,56E-06	4,22E-03
$\theta = \frac{\pi}{4}$	$\omega = 0.01$	0,00E+00	6,36E-05	0,00E+00	0,00E+00	1,83E-05	5,59E-05	2,26E-08	1,81E-03
	$\omega = 0.05$	1,58E-08	6,36E-05	0,00E+00	0,00E+00	2,84E-05	5,59E-05	9,87E-08	2,15E-03
	$\omega = 0.25$	8,39E-08	6,36E-05	0,00E+00	0,00E+00	4,28E-05	5,59E-05	2,78E-07	3,02E-03
	$\omega = 1.25$	1,44E-06	6,36E-05	4,33E-08	0,00E+00	1,02E-04	5,59E-05	4,43E-06	3,57E-03
	$\omega = 6.25$	1,41E-05	6,36E-05	6,71E-07	0,00E+00	1,98E-04	5,59E-05	9,97E-06	4,08E-03
$\theta = \frac{\pi}{3}$	$\omega = 0.01$	0,00E+00	6,36E-05	0,00E+00	0,00E+00	1,87E-05	5,59E-05	2,62E-08	1,64E-03
	$\omega = 0.05$	0,00E+00	6,36E-05	0,00E+00	0,00E+00	2,82E-05	5,59E-05	9,44E-08	2,57E-03
	$\omega = 0.25$	8,98E-08	6,36E-05	0,00E+00	0,00E+00	3,91E-05	5,59E-05	7,26E-07	2,60E-03
	$\omega = 1.25$	1,27E-06	6,36E-05	4,17E-08	0,00E+00	8,51E-05	5,59E-05	6,87E-07	3,32E-03
	$\omega = 6.25$	1,64E-05	6,36E-05	8,99E-07	0,00E+00	2,56E-04	5,59E-05	1,35E-05	4,48E-03
$\theta = \frac{5\pi}{12}$	$\omega = 0.01$	0,00E+00	6,36E-05	0,00E+00	0,00E+00	2,00E-05	5,59E-05	2,33E-08	1,74E-03
	$\omega = 0.05$	0,00E+00	6,36E-05	0,00E+00	0,00E+00	2,39E-05	5,59E-05	9,41E-08	2,22E-03
	$\omega = 0.25$	8,15E-08	6,36E-05	0,00E+00	0,00E+00	4,69E-05	5,59E-05	4,27E-07	2,84E-03
	$\omega = 1.25$	4,69E-07	6,36E-05	3,73E-08	0,00E+00	8,75E-05	5,59E-05	7,58E-07	3,14E-03
	$\omega = 6.25$	2,35E-05	6,36E-05	1,09E-06	0,00E+00	1,84E-04	5,59E-05	1,37E-05	4,39E-03

TABLE 3.2 – Valeurs de l'écart-type pour la dimension $d = 10$.

		F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
$\theta = \frac{\pi}{12}$	$\omega = 0.01$	4,55E-08	1,27E-04	0,00E+00	0,00E+00	3,82E-05	4,74E-05	4,44E-05	1,09E-04	0,00E+00	2,94E-03
	$\omega = 0.05$	2,71E-07	1,27E-04	0,00E+00	0,00E+00	3,82E-05	5,23E-05	5,18E-05	1,09E-04	0,00E+00	3,97E-03
	$\omega = 0.25$	7,83E-06	1,27E-04	5,77E-08	0,00E+00	3,82E-05	6,49E-05	8,36E-05	1,09E-04	0,00E+00	5,60E-03
	$\omega = 1.25$	2,08E-04	1,27E-04	2,24E-06	0,00E+00	3,87E-05	9,97E-05	1,43E-04	1,09E-04	2,80E-08	6,08E-03
	$\omega = 6.25$	5,55E-03	1,27E-04	4,68E-05	0,00E+00	4,02E-05	2,70E-04	5,38E-04	1,09E-04	7,02E-07	6,82E-03
$\theta = \frac{\pi}{6}$	$\omega = 0.01$	4,30E-08	1,27E-04	0,00E+00	0,00E+00	3,82E-05	4,51E-05	3,95E-05	1,09E-04	0,00E+00	3,31E-03
	$\omega = 0.05$	4,96E-07	1,27E-04	0,00E+00	0,00E+00	3,83E-05	5,32E-05	5,49E-05	1,09E-04	0,00E+00	3,81E-03
	$\omega = 0.25$	7,04E-06	1,27E-04	5,16E-08	0,00E+00	3,84E-05	7,53E-05	9,26E-05	1,09E-04	0,00E+00	5,15E-03
	$\omega = 1.25$	2,11E-04	1,27E-04	2,19E-06	0,00E+00	3,89E-05	1,30E-04	1,81E-04	1,09E-04	2,76E-08	5,85E-03
	$\omega = 6.25$	3,79E-03	1,27E-04	4,97E-05	0,00E+00	3,98E-05	2,58E-04	5,94E-04	1,09E-04	7,54E-07	8,15E-03
$\theta = \frac{\pi}{4}$	$\omega = 0.01$	3,68E-08	1,27E-04	0,00E+00	0,00E+00	3,82E-05	4,54E-05	3,63E-05	1,09E-04	0,00E+00	3,47E-03
	$\omega = 0.05$	5,90E-07	1,27E-04	0,00E+00	0,00E+00	3,82E-05	5,49E-05	5,30E-05	1,09E-04	0,00E+00	3,58E-03
	$\omega = 0.25$	9,16E-06	1,27E-04	5,93E-08	0,00E+00	3,84E-05	8,31E-05	8,57E-05	1,09E-04	0,00E+00	5,46E-03
	$\omega = 1.25$	2,01E-04	1,27E-04	1,83E-06	0,00E+00	3,87E-05	1,36E-04	2,09E-04	1,09E-04	3,11E-08	7,60E-03
	$\omega = 6.25$	4,04E-03	1,27E-04	3,70E-05	0,00E+00	4,06E-05	2,46E-04	5,57E-04	1,09E-04	7,41E-07	7,55E-03
$\theta = \frac{\pi}{3}$	$\omega = 0.01$	3,53E-08	1,27E-04	0,00E+00	0,00E+00	3,82E-05	4,46E-05	4,30E-05	1,09E-04	0,00E+00	3,57E-03
	$\omega = 0.05$	4,63E-07	1,27E-04	0,00E+00	0,00E+00	3,82E-05	5,68E-05	6,02E-05	1,09E-04	0,00E+00	3,79E-03
	$\omega = 0.25$	1,04E-05	1,27E-04	8,70E-08	0,00E+00	3,84E-05	7,60E-05	1,04E-04	1,09E-04	0,00E+00	6,02E-03
	$\omega = 1.25$	2,30E-04	1,27E-04	1,88E-06	0,00E+00	3,94E-05	1,24E-04	1,71E-04	1,09E-04	2,41E-08	6,68E-03
	$\omega = 6.25$	5,94E-03	1,27E-04	5,31E-05	0,00E+00	3,99E-05	2,49E-04	4,32E-04	1,09E-04	7,23E-07	8,14E-03
$\theta = \frac{5\pi}{12}$	$\omega = 0.01$	1,36E-08	1,27E-04	0,00E+00	0,00E+00	3,82E-05	4,58E-05	3,83E-05	1,09E-04	0,00E+00	3,37E-03
	$\omega = 0.05$	4,78E-07	1,27E-04	0,00E+00	0,00E+00	3,82E-05	5,23E-05	5,80E-05	1,09E-04	0,00E+00	4,61E-03
	$\omega = 0.25$	1,10E-05	1,27E-04	5,59E-08	0,00E+00	3,84E-05	7,95E-05	9,62E-05	1,09E-04	0,00E+00	4,44E-03
	$\omega = 1.25$	1,81E-04	1,27E-04	2,10E-06	0,00E+00	3,86E-05	1,40E-04	1,63E-04	1,09E-04	2,93E-08	5,76E-03
	$\omega = 6.25$	2,59E-03	1,27E-04	5,23E-05	0,00E+00	4,13E-05	2,55E-04	5,91E-04	1,09E-04	7,24E-07	8,28E-03

TABLE 3.3 – Valeurs de l'écart-type pour la dimension $d = 15$.

		F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
	$\omega = 0.01$	1,75E-07	1,91E-04	0,00E+00	0,00E+00	6,36E-05	5,61E-05	5,64E-05	1,52E-04	0,00E+00	8,56E-03
	$\omega = 0.05$	3,46E-06	1,91E-04	1,64E-08	0,00E+00	6,37E-05	6,47E-05	7,13E-05	1,52E-04	0,00E+00	1,23E-02
$\theta = \frac{\pi}{12}$	$\omega = 0.25$	5,06E-05	1,91E-04	4,61E-07	0,00E+00	6,37E-05	8,23E-05	1,34E-04	1,52E-04	1,35E-08	1,90E-02
	$\omega = 1.25$	1,15E-03	1,91E-04	1,14E-05	0,00E+00	6,43E-05	1,33E-04	2,79E-04	1,52E-04	3,17E-07	2,56E-02
	$\omega = 6.25$	2,93E-02	1,91E-04	2,31E-04	0,00E+00	8,48E-05	2,96E-04	8,09E-04	1,52E-04	6,83E-06	2,78E-02
	$\omega = 0.01$	1,20E-07	1,91E-04	0,00E+00	0,00E+00	6,36E-05	5,48E-05	5,81E-05	1,52E-04	0,00E+00	9,22E-03
	$\omega = 0.05$	2,77E-06	1,91E-04	1,20E-08	0,00E+00	6,36E-05	6,67E-05	7,26E-05	1,52E-04	0,00E+00	1,37E-02
$\theta = \frac{\pi}{6}$	$\omega = 0.25$	6,11E-05	1,91E-04	4,25E-07	0,00E+00	6,38E-05	9,27E-05	1,26E-04	1,52E-04	1,26E-08	1,94E-02
	$\omega = 1.25$	1,23E-03	1,91E-04	1,34E-05	0,00E+00	6,59E-05	1,28E-04	2,08E-04	1,52E-04	4,28E-07	2,53E-02
	$\omega = 6.25$	2,50E-02	1,92E-04	2,17E-04	1,14E-08	7,15E-05	2,83E-04	9,32E-04	1,52E-04	6,86E-06	3,11E-02
	$\omega = 0.01$	1,37E-07	1,91E-04	0,00E+00	0,00E+00	6,36E-05	5,80E-05	5,64E-05	1,52E-04	0,00E+00	8,59E-03
	$\omega = 0.05$	2,66E-06	1,91E-04	1,55E-08	0,00E+00	6,37E-05	6,62E-05	7,61E-05	1,52E-04	0,00E+00	1,33E-02
$\theta = \frac{\pi}{4}$	$\omega = 0.25$	4,35E-05	1,91E-04	3,65E-07	0,00E+00	6,38E-05	9,05E-05	1,13E-04	1,52E-04	1,24E-08	1,52E-02
	$\omega = 1.25$	1,01E-03	1,91E-04	1,13E-05	0,00E+00	6,49E-05	1,53E-04	2,20E-04	1,52E-04	3,01E-07	2,14E-02
	$\omega = 6.25$	2,69E-02	1,92E-04	3,55E-04	0,00E+00	7,49E-05	2,60E-04	7,91E-04	1,52E-04	8,24E-06	2,81E-02
	$\omega = 0.01$	8,59E-08	1,91E-04	0,00E+00	0,00E+00	6,36E-05	5,64E-05	5,49E-05	1,52E-04	0,00E+00	9,07E-03
	$\omega = 0.05$	2,14E-06	1,91E-04	1,53E-08	0,00E+00	6,37E-05	6,79E-05	6,76E-05	1,52E-04	0,00E+00	1,35E-02
$\theta = \frac{\pi}{3}$	$\omega = 0.25$	3,09E-05	1,91E-04	3,77E-07	0,00E+00	6,40E-05	8,43E-05	1,18E-04	1,52E-04	1,46E-08	1,67E-02
	$\omega = 1.25$	9,38E-04	1,91E-04	1,02E-05	0,00E+00	6,64E-05	1,44E-04	2,41E-04	1,52E-04	3,34E-07	2,51E-02
	$\omega = 6.25$	3,02E-02	1,91E-04	2,54E-04	1,48E-08	7,67E-05	3,16E-04	8,32E-04	1,52E-04	7,65E-06	2,70E-02
	$\omega = 0.01$	1,39E-07	1,91E-04	0,00E+00	0,00E+00	6,36E-05	5,60E-05	5,54E-05	1,52E-04	0,00E+00	9,71E-03
	$\omega = 0.05$	3,84E-06	1,91E-04	1,52E-08	0,00E+00	6,37E-05	6,15E-05	6,85E-05	1,52E-04	0,00E+00	1,35E-02
$\theta = \frac{5\pi}{12}$	$\omega = 0.25$	6,00E-05	1,91E-04	3,43E-07	0,00E+00	6,39E-05	8,24E-05	1,14E-04	1,52E-04	1,42E-08	1,61E-02
	$\omega = 1.25$	1,46E-03	1,91E-04	1,13E-05	0,00E+00	6,50E-05	1,58E-04	2,39E-04	1,52E-04	3,94E-07	2,26E-02
	$\omega = 6.25$	2,12E-02	1,91E-04	2,66E-04	0,00E+00	7,21E-05	3,00E-04	5,61E-04	1,52E-04	1,05E-05	2,76E-02

TABLE 3.4 – Valeurs de l'écart-type pour la dimension $d = 20$.

		F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
	$\omega = 0.01$	4,99E-07	2,55E-04	0,00E+00	0,00E+00	7,68E-05	1,02E-04	7,70E-05	2,00E-04	0,00E+00	9,84E-03
	$\omega = 0.05$	5,51E-06	2,55E-04	7,33E-08	0,00E+00	7,76E-05	1,24E-04	9,12E-05	2,00E-04	0,00E+00	1,23E-02
$\theta = \frac{\pi}{12}$	$\omega = 0.25$	1,74E-04	2,55E-04	1,94E-06	0,00E+00	8,25E-05	1,68E-04	1,62E-04	2,00E-04	2,39E-08	1,48E-02
	$\omega = 1.25$	3,85E-03	2,55E-04	4,56E-05	0,00E+00	9,08E-05	2,51E-04	3,23E-04	2,00E-04	7,21E-07	1,61E-02
	$\omega = 6.25$	9,70E-02	2,56E-04	1,18E-03	5,45E-08	1,06E-04	8,19E-04	1,06E-03	2,00E-04	1,39E-05	2,20E-02
	$\omega = 0.01$	6,77E-07	2,55E-04	0,00E+00	0,00E+00	7,67E-05	1,03E-04	6,94E-05	2,00E-04	0,00E+00	9,34E-03
	$\omega = 0.05$	7,47E-06	2,55E-04	6,52E-08	0,00E+00	7,82E-05	1,14E-04	9,49E-05	2,00E-04	0,00E+00	1,09E-02
$\theta = \frac{\pi}{6}$	$\omega = 0.25$	2,23E-04	2,55E-04	1,50E-06	0,00E+00	8,53E-05	1,94E-04	1,52E-04	2,00E-04	2,63E-08	1,48E-02
	$\omega = 1.25$	4,83E-03	2,55E-04	5,00E-05	0,00E+00	9,10E-05	3,65E-04	3,27E-04	2,00E-04	6,01E-07	1,64E-02
	$\omega = 6.25$	1,11E-01	2,56E-04	1,08E-03	3,13E-08	1,10E-04	7,06E-04	1,01E-03	2,00E-04	1,74E-05	2,21E-02
	$\omega = 0.01$	4,42E-07	2,55E-04	0,00E+00	0,00E+00	7,69E-05	1,02E-04	7,23E-05	2,00E-04	0,00E+00	9,39E-03
	$\omega = 0.05$	1,11E-05	2,55E-04	7,81E-08	0,00E+00	7,92E-05	1,20E-04	1,04E-04	2,00E-04	0,00E+00	1,06E-02
$\theta = \frac{\pi}{4}$	$\omega = 0.25$	2,21E-04	2,55E-04	1,50E-06	0,00E+00	8,31E-05	1,73E-04	1,60E-04	2,00E-04	2,89E-08	1,54E-02
	$\omega = 1.25$	3,39E-03	2,55E-04	4,15E-05	0,00E+00	8,87E-05	3,09E-04	3,79E-04	2,00E-04	7,14E-07	1,56E-02
	$\omega = 6.25$	1,37E-01	2,55E-04	1,43E-03	3,15E-08	1,17E-04	7,20E-04	1,16E-03	2,00E-04	1,68E-05	2,05E-02
	$\omega = 0.01$	4,87E-07	2,55E-04	0,00E+00	0,00E+00	7,67E-05	1,04E-04	7,21E-05	2,00E-04	0,00E+00	9,62E-03
	$\omega = 0.05$	1,33E-05	2,55E-04	5,31E-08	0,00E+00	7,86E-05	1,19E-04	9,82E-05	2,00E-04	0,00E+00	1,05E-02
$\theta = \frac{\pi}{3}$	$\omega = 0.25$	1,43E-04	2,55E-04	2,07E-06	0,00E+00	8,09E-05	1,85E-04	1,65E-04	2,00E-04	2,46E-08	1,38E-02
	$\omega = 1.25$	3,63E-03	2,55E-04	4,64E-05	0,00E+00	8,94E-05	3,54E-04	2,94E-04	2,00E-04	7,25E-07	1,70E-02
	$\omega = 6.25$	1,33E-01	2,56E-04	7,94E-04	2,71E-08	1,14E-04	7,15E-04	1,01E-03	2,00E-04	1,59E-05	2,17E-02
	$\omega = 0.01$	5,09E-07	2,55E-04	0,00E+00	0,00E+00	7,67E-05	1,00E-04	6,74E-05	2,00E-04	0,00E+00	8,83E-03
	$\omega = 0.05$	1,06E-05	2,55E-04	6,13E-08	0,00E+00	7,91E-05	1,29E-04	1,07E-04	2,00E-04	0,00E+00	1,15E-02
$\theta = \frac{5\pi}{12}$	$\omega = 0.25$	1,92E-04	2,55E-04	1,77E-06	0,00E+00	8,25E-05	1,81E-04	1,69E-04	2,00E-04	3,08E-08	1,32E-02
	$\omega = 1.25$	3,39E-03	2,55E-04	5,49E-05	0,00E+00	8,97E-05	3,12E-04	2,61E-04	2,00E-04	7,85E-07	1,53E-02
	$\omega = 6.25$	1,06E-01	2,56E-04	1,28E-03	8,83E-08	1,24E-04	9,45E-04	1,05E-03	2,00E-04	1,54E-05	1,99E-02

TABLE 3.5 – Performances de l'algorithme AHO sur les fonctions 5d, 10d, 15d et 20d (moyennées sur 30 exécutions indépendantes).

		F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
	Worst	8.54E-08	6.36E-05	0.00E+00	0.00E+00	3.70E-05	-	-	5.59E-05	4.58E-08	2.95E-03
$d = 5$,	Best	0.00E+00	6.36E-05	0.00E+00	0.00E+00	1.48E-05	-	-	5.59E-05	0.00E+00	7.61E-04
$\theta = \frac{\pi}{12}$,	Median	0.00E+00	6.36E-05	0.00E+00	0.00E+00	2.27E-05	-	-	5.59E-05	2.62E-08	1.71E-03
et $\omega = 0.01$	Mean	0.00E+00	6.36E-05	0.00E+00	0.00E+00	1.94E-05	-	-	5.59E-05	2.04E-08	1.46E-03
	Std	0.00E+00	6.36E-05	0.00E+00	0.00E+00	1.94E-05	-	-	5.59E-05	2.04E-08	1.46E-03
	Worst	7.12E-07	1.27E-04	0.00E+00	0.00E+00	3.85E-05	6.59E-05	6.15E-05	1.09E-04	0.00E+00	5.67E-03
$d = 10$,	Best	1.32E-08	1.27E-04	0.00E+00	0.00E+00	3.82E-05	4.13E-05	2.95E-05	1.09E-04	0.00E+00	2.04E-03
$\theta = \frac{5\pi}{12}$,	Median	1.33E-07	1.27E-04	0.00E+00	0.00E+00	3.82E-05	5.05E-05	4.40E-05	1.09E-04	0.00E+00	3.64E-03
et $\omega = 0.01$	Mean	1.36E-08	1.27E-04	0.00E+00	0.00E+00	3.82E-05	4.58E-05	3.83E-05	1.09E-04	0.00E+00	3.37E-03
	Std	1.36E-08	1.27E-04	0.00E+00	0.00E+00	3.82E-05	4.58E-05	3.83E-05	1.09E-04	0.00E+00	3.37E-03
	Worst	8.70E-07	1.91E-04	0.00E+00	0.00E+00	6.37E-05	6.62E-05	7.69E-05	1.52E-04	0.00E+00	1.34E-02
$d = 15$,	Best	8.20E-08	1.91E-04	0.00E+00	0.00E+00	6.36E-05	5.37E-05	5.07E-05	1.52E-04	0.00E+00	7.33E-03
$\theta = \frac{\pi}{3}$,	Median	3.05E-07	1.91E-04	0.00E+00	0.00E+00	6.36E-05	5.88E-05	6.12E-05	1.52E-04	0.00E+00	9.74E-03
et $\omega = 0.01$	Mean	8.59E-08	1.91E-04	0.00E+00	0.00E+00	6.36E-05	5.64E-05	5.49E-05	1.52E-04	0.00E+00	9.07E-03
	Std	8.59E-08	1.91E-04	0.00E+00	0.00E+00	6.36E-05	5.64E-05	5.49E-05	1.52E-04	0.00E+00	9.07E-03
	Worst	4.91E-06	2.55E-04	1.20E-08	0.00E+00	7.81E-05	1.34E-04	1.05E-04	2.00E-04	0.00E+00	1.40E-02
$d = 20$,	Best	4.16E-07	2.55E-04	0.00E+00	0.00E+00	7.65E-05	9.20E-05	5.96E-05	2.00E-04	0.00E+00	6.20E-03
$\theta = \frac{5\pi}{12}$,	Median	1.30E-06	2.55E-04	0.00E+00	0.00E+00	7.70E-05	1.14E-04	8.00E-05	2.00E-04	0.00E+00	9.90E-03
et $\omega = 0.01$	Mean	5.09E-07	2.55E-04	0.00E+00	0.00E+00	7.67E-05	1.00E-04	6.74E-05	2.00E-04	0.00E+00	8.83E-03
	Std	5.09E-07	2.55E-04	0.00E+00	0.00E+00	7.67E-05	1.00E-04	6.74E-05	2.00E-04	0.00E+00	8.83E-03

et d'exploitation de l'espace de recherche, quelle que soit la valeur de l'angle de perception.

Les résultats statistiques de l'algorithme AHO sur le benchmark CEC 2020 avec les dimensions 5, 10, 15 et 20 sont résumés dans la table 3.5. Nous présentons le min, le max, la médiane, la moyenne et l'écart-type sur 30 exécutions indépendantes pour les dix fonctions de test. Il convient de souligner que les résultats présentés dans la table 3.5 sont tirés des meilleures configurations des tables 3.1, 3.2, 3.3 et 3.4 en utilisant le test de Friedman.

Pour la fonction F_1, l'algorithme AHO obtient avec succès la solution optimale pour la dimension $d = 5$ et est très proche de l'optimum pour les dimensions $d = 10$, $d = 15$ et $d = 20$. Pour les fonctions F_2, F_5, F_6, F_7, F_8 et F_{10}, l'algorithme AHO est très proche de l'optimum pour toutes les dimensions. Pour les fonctions F_3 et F_4, l'algorithme AHO obtient la solution optimale pour toutes les dimensions. Pour la fonction F_9, l'algorithme AHO obtient avec succès la solution optimale pour les dimensions $d = 10$, $d = 15$ et $d = 20$ et est très proche de l'optimum pour la dimension $d = 5$. En conclusion, nous pouvons dire que :

— À partir de la fonction F_1, l'algorithme AHO a une bonne exploitation de l'espace de recherche.
— À partir des fonctions F_2, F_3 et F_4, l'algorithme AHO a une bonne exploration de l'espace de recherche et une bonne capacité d'évitement des optimums locaux.
— À partir des fonctions F_5, F_6, F_7, F_8, F_9 et F_{10}, l'algorithme AHO a un bon équilibrage entre les phases d'exploration et d'exploitation.

Les performances de l'algorithme AHO sont comparées à 12 algorithmes métaheuristiques bien établis que nous mentionnons ci-après. Les résultats numériques sont rapportés dans : i) les tables 3.6, 3.7, 3.8 et 3.9 pour $d = 5$; ii) les tables 3.10, 3.11, 3.12 et 3.13 pour $d = 10$; iii) les tables 3.14, 3.15, 3.16 et 3.17 pour $d = 15$; et iv) les tables 3.18, 3.19, 3.20 et 3.21 pour $d = 20$.

— Improving Cuckoo Search : Incorporating Changes for CEC 2017 and CEC 2020 Benchmark Problems (CSsin) [237].
— A Multi-Population Exploration-only Exploitation-only Hybrid on CEC-2020 Single Objective Bound Constrained Problems (MP-EEH) [238].
— Ranked Archive Differential Evolution with Selective Pressure for CEC 2020 Numerical Optimization (RASPSHADE) [239].
— Improved Multi-operator Differential Evolution Algorithm for Solving Unconstrained Problems

TABLE 3.6 – Résultats statistiques de l'algorithme AHO en comparaison aux algorithmes CSsin, MP-EEH et RASPSHADE ($d = 5$).

	AHO		CSsin			MP-EEH			RASPSHADE		
	Mean	**STD**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**
F_1	0,00E+00	0,00E+00	2,81E-07	3,01E-07	+	0,00E+00	0,00E+00	=	0,00E+00	0,00E+00	=
F_2	6,36E-05	6,36E-05	8,75E+00	2,81E+01	+	2,46E+01	3,92E+01	+	5,64E-01	1,18E+00	+
F_3	0,00E+00	0,00E+00	5,65E+00	4,74E-01	+	3,43E+00	2,13E+00	+	5,34E+00	1,72E-01	+
F_4	0,00E+00	0,00E+00	0,00E+00	0,00E+00	=	1,02E-01	8,06E-02	+	1,04E-01	3,85E-02	+
F_5	1,94E-05	1,94E-05	3,94E-04	6,52E-04	+	1,03E+01	9,90E+00	+	8,32E-02	2,12E-01	+
F_8	5,59E-05	5,59E-05	3,67E+01	4,77E+01	+	1,68E-08	5,30E-09	-	0,00E+00	0,00E+00	-
F_9	2,04E-08	2,04E-08	1,17E+01	3,25E+01	+	1,03E+02	1,83E+01	+	1,00E+02	0,00E+00	-
F_{10}	1,46E-03	1,46E-03	2,88E+02	7,00E+01	+	3,10E+02	5,42E+01	+	3,44E+02	1,18E+01	+

TABLE 3.7 – Résultats statistiques de l'algorithme AHO en comparaison aux algorithmes IMODE, DISH-XX et AGSK ($d = 5$).

	AHO		IMODE			DISH-XX			AGSK		
	Mean	**STD**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**
F_1	0,00E+00	0,00E+00	0,00E+00	0,00E+00	=	0,00E+00	0,00E+00	=	0,00E+00	0,00E+00	=
F_2	6,36E-05	6,36E-05	8,33E-02	8,89E-02	+	1,31E+01	2,82E+01	+	1,64E+01	2,58E+01	+
F_3	0,00E+00	0,00E+00	5,15E+00	0,00E+00	=	5,65E+00	6,07E-01	+	2,87E+00	2,05E+00	+
F_4	0,00E+00	0,00E+00	0,00E+00	0,00E+00	=	1,42E-02	9,57E-03	+	1,11E-01	6,05E-02	+
F_5	1,94E-05	1,94E-05	0,00E+00	0,00E+00	-	2,08E-01	2,99E-01	+	0,00E+00	0,00E+00	-
F_8	5,59E-05	5,59E-05	0,00E+00	0,00E+00	-	3,35E+00	1,83E+01	+	0,00E+00	0,00E+00	-
F_9	2,04E-08	2,04E-08	0,00E+00	0,00E+00	-	1,10E+02	4,03E+01	+	3,33E+01	4,79E+01	+
F_{10}	1,46E-03	1,46E-03	2,44E+02	1,36E+02	+	3,44E+02	1,20E+01	+	2,25E+02	1,32E+02	+

(IMODE) [240].

— DISH–XX Solving CEC2020 Single Objective Bound Constrained Numerical Optimization Benchmark (DISH-XX) [241].

— Evaluating the Performance of Adaptive Gaining-Sharing Knowledge Based Algorithm on CEC 2020 Benchmark Problems (AGSK) [242].

— Differential Evolution Algorithm for Single Objective Bound-Constrained Optimization : Algorithm j2020 (j2020) [243].

— Eigenvector Crossover in jDE100 Algorithm (jDE100e) [244].

— Large Initial Population and Neighborhood Search incoporated in LSHADE to solve CEC2020 Benchmark Problems (OLSHADE) [245].

— Multi-population Modified L-SHADE for Single Objective Bound Constrained Optimization (mpmL-SHADE) [246].

— SOMA-CL for Competition on Single Objective Bound Constrained Numerical Optimization Benchmark (SOMA-CL) [247].

— Gaining-sharing knowledge based algorithm for solving optimization problems : a novel nature inspired algorithm (GSK) [248].

Nous effectuons le test des rangs signés de Wilcoxon [249] pour établir si l'algorithme AHO est meilleur/pire que les algorithmes CSsin, MP-EEH, RASPSHADE, IMODE, DISH-XX, AGSK, j2020, jDE100e, OL-SHADE, mpmLSHADE, SOMA-CL et GSK pour toutes les dimensions. Nous considérons le test unilatéral avec $\alpha = 0.05$. Pour chaque dimension, les valeurs utilisées pour les paramètres de contrôle de l'algorithme AHO sont rapportées dans la table 3.5. Pour les algorithmes sélectionnés pour l'étude comparative, les valeurs des paramètres de contrôle sont les mêmes recommandées dans leurs travaux d'origine. Les tables 3.22, 3.23, 3.24, et 3.25 résument les résultats du test des rangs signés de Wilcoxon.

D'après les tables 3.22, 3.23, 3.24 et 3.25, l'algorithme AHO surpasse avec succès tous les algorithmes

TABLE 3.8 – Résultats statistiques de l'algorithme AHO en comparaison aux algorithmes j2020, jDE100e et OLSHADE ($d = 5$).

	AHO		j2020			jDE100e			OLSHADE		
	Mean	**STD**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**
F_1	0,00E+00	0,00E+00	0,00E+00	0,00E+00	=	0,00E+00	0,00E+00	=	0,00E+00	0,00E+00	=
F_2	6,36E-05	6,36E-05	3,23E+00	3,74E+00	+	1,34E+00	2,93E+00	+	1,88E+01	4,16E+01	+
F_3	0,00E+00	0,00E+00	3,42E+00	2,33E+00	+	4,52E+00	1,52E+00	+	1,41E+00	1,36E+00	+
F_4	0,00E+00	0,00E+00	7,68E-02	6,40E-02	+	8,67E-02	4,67E-02	+	7,27E-02	5,98E-02	+
F_5	1,94E-05	1,94E-05	1,37E-01	2,86E-01	+	5,40E-02	2,11E-01	+	4,16E-02	1,58E-01	+
F_8	5,59E-05	5,59E-05	6,28E-01	2,39E+00	+	1,20E-01	6,54E-01	+	0,00E+00	0,00E+00	-
F_9	2,04E-08	2,04E-08	2,05E+01	3,75E+01	+	9,33E+01	2,54E+01	+	0,00E+00	0,00E+00	-
F_{10}	1,46E-03	1,46E-03	1,26E+02	9,03E+01	+	2,90E+02	8,06E+01	+	1,13E+02	9,73E+01	+

TABLE 3.9 – Résultats statistiques de l'algorithme AHO en comparaison aux algorithmes mpmLSHADE, SOMA-CL et GSK ($d = 5$).

	AHO		mpmLSHADE			SOMA-CL			GSK		
	Mean	**STD**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**
F_1	0,00E+00	0,00E+00	0,00E+00	0,00E+00	=	5,29E-02	3,35E-02	+	0,00E+00	0,00E+00	=
F_2	6,36E-05	6,36E-05	1,17E-01	1,37E-01	+	6,67E+00	4,65E+00	+	1,23E+02	4,33E+01	+
F_3	0,00E+00	0,00E+00	4,70E+00	1,47E+00	+	5,74E+00	4,29E-01	+	7,44E+00	6,40E-01	+
F_4	0,00E+00	0,00E+00	9,61E-02	3,09E-02	+	2,18E-01	9,49E-02	+	3,53E-01	9,91E-02	+
F_5	1,94E-05	1,94E-05	0,00E+00	0,00E+00	-	7,62E-03	1,06E-02	+	2,92E-02	6,49E-02	+
F_8	5,59E-05	5,59E-05	3,33E+00	1,80E+01	+	2,02E-01	2,96E-01	+	6,78E-02	5,94E-02	+
F_9	2,04E-08	2,04E-08	1,03E+02	4,07E+01	+	7,09E+01	2,48E+01	+	1,08E+02	3,12E+01	+
F_{10}	1,46E-03	1,46E-03	3,41E+02	1,61E+01	+	3,17E+02	4,63E+01	+	3,47E+02	4,32E-01	+

TABLE 3.10 – Résultats statistiques de l'algorithme AHO en comparaison aux algorithmes CSsin, MP-EEH et RASPSHADE ($d = 10$).

	AHO		CSsin			MP-EEH			RASPSHADE		
	Mean	**STD**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**
F_1	1,36E-08	1,36E-08	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-
F_2	1,27E-04	1,27E-04	1,58E+01	1,21E+01	+	3,48E+01	4,61E+01	+	9,44E-01	1,32E+00	+
F_3	0,00E+00	0,00E+00	1,39E+01	1,71E+00	+	1,06E+01	3,62E+00	+	1,11E+01	3,58E-01	+
F_4	0,00E+00	0,00E+00	0,00E+00	0,00E+00	=	8,67E-02	7,24E-02	+	2,72E-01	5,73E-02	+
F_5	3,82E-05	3,82E-05	4,43E+00	1,82E+00	+	3,07E+01	2,09E+01	+	3,40E-01	1,73E-01	+
F_6	4,58E-05	4,58E-05	1,70E-01	1,20E-01	+	1,23E+00	2,08E+00	+	1,59E-01	9,39E-02	+
F_7	3,83E-05	3,83E-05	1,55E-01	9,95E-02	+	1,57E+00	1,86E+00	+	8,94E-04	9,58E-04	+
F_8	1,09E-04	1,09E-04	8,01E+01	3,18E+01	+	4,67E+01	5,07E+01	+	1,00E+02	0,00E+00	-
F_9	0,00E+00	0,00E+00	1,00E+02	1,38E-13	+	1,03E+02	1,83E+01	+	1,00E+02	0,00E+00	=
F_{10}	3,37E-03	3,37E-03	3,77E+02	7,55E+01	+	3,38E+02	1,06E+02	+	3,98E+02	6,64E-02	+

TABLE 3.11 – Résultats statistiques de l'algorithme AHO en comparaison aux algorithmes IMODE, DISH-XX et AGSK ($d = 10$).

	AHO		IMODE			DISH-XX			AGSK		
	Mean	**STD**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**
F_1	1,36E-08	1,36E-08	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-
F_2	1,27E-04	1,27E-04	4,20E+00	3,70E+00	+	1,97E+01	2,59E+01	+	2,84E+01	3,21E+01	+
F_3	0,00E+00	0,00E+00	1,21E+01	7,83E-01	+	1,14E+01	5,85E-01	+	9,93E+00	4,26E+00	+
F_4	0,00E+00	0,00E+00	0,00E+00	0,00E+00	=	2,47E-04	1,35E-03	+	5,83E-02	3,11E-02	+
F_5	3,82E-05	3,82E-05	3,88E-01	3,83E-01	+	1,24E+00	2,80E+00	+	3,18E-01	3,06E-01	+
F_6	4,58E-05	4,58E-05	9,15E-02	5,08E-02	+	6,96E-01	1,99E+00	+	1,55E-01	1,17E-01	+
F_7	3,83E-05	3,83E-05	8,54E-04	1,10E-03	+	2,40E-01	2,50E-01	+	1,54E-03	1,71E-03	+
F_8	1,09E-04	1,09E-04	2,72E+00	7,46E+00	+	1,00E+02	0,00E+00	-	1,80E+01	2,38E+01	+
F_9	0,00E+00	0,00E+00	4,11E+01	4,46E+01	+	3,07E+02	7,01E+01	+	7,63E+01	4,29E+01	+
F_{10}	3,37E-03	3,37E-03	3,98E+02	2,89E-13	-	4,07E+02	1,88E+01	+	2,98E+02	1,43E+02	+

TABLE 3.12 – Résultats statistiques de l'algorithme AHO en comparaison aux algorithmes j2020, jDE100e et OLSHADE ($d = 10$).

	AHO		j2020			jDE100e			OLSHADE		
	Mean	**STD**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**
F_1	1,36E-08	1,36E-08	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-
F_2	1,27E-04	1,27E-04	6,79E-01	1,16E+00	+	1,01E+01	1,02E+01	+	2,92E+01	1,04E+01	+
F_3	0,00E+00	0,00E+00	8,06E+00	3,88E+00	+	1,18E+01	1,83E+00	+	8,42E+00	3,23E+00	+
F_4	0,00E+00	0,00E+00	1,09E-01	9,04E-02	+	1,65E-01	4,18E-02	+	8,74E-02	7,70E-02	+
F_5	3,82E-05	3,82E-05	3,02E-01	3,13E-01	+	8,15E-01	5,75E-01	+	1,08E+00	1,91E+00	+
F_6	4,58E-05	4,58E-05	4,78E-01	2,49E-01	+	3,46E-01	2,22E-01	+	0,00E+00	0,00E+00	-
F_7	3,83E-05	3,83E-05	6,73E-02	1,25E-01	+	6,50E-03	7,87E-03	+	2,29E-01	2,63E-01	+
F_8	1,09E-04	1,09E-04	1,54E+00	4,00E+00	+	3,33E+01	4,79E+01	+	3,36E+01	2,06E+01	+
F_9	0,00E+00	0,00E+00	8,00E+01	4,07E+01	+	1,08E+02	4,19E+01	+	1,00E+02	0,00E+00	=
F_{10}	3,37E-03	3,37E-03	1,40E+02	8,12E+01	+	3,98E+02	6,75E-02	+	1,00E+02	0,00E+00	-

TABLE 3.13 – Résultats statistiques de l'algorithme AHO en comparaison aux algorithmes mpmLSHADE, SOMA-CL et GSK ($d = 10$).

	AHO		mpmLSHADE			SOMA-CL			GSK		
	Mean	**STD**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**
F_1	1,36E-08	1,36E-08	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-
F_2	1,27E-04	1,27E-04	6,81E-01	1,13E+00	+	1,47E+00	2,22E+00	+	8,19E+02	1,21E+02	+
F_3	0,00E+00	0,00E+00	1,06E+01	2,34E-01	+	1,06E+01	2,27E-01	+	2,38E+01	2,84E+00	+
F_4	0,00E+00	0,00E+00	2,84E-01	5,83E-02	+	1,52E-01	1,26E-01	+	1,46E+00	1,96E-01	+
F_5	3,82E-05	3,82E-05	1,25E-01	1,15E-01	+	5,08E-01	2,02E+00	+	2,96E+01	7,44E+00	+
F_6	4,58E-05	4,58E-05	5,09E-02	4,09E-02	+	1,72E-01	1,49E-01	+	2,69E+00	4,73E-01	+
F_7	3,83E-05	3,83E-05	1,37E-01	1,55E-01	+	6,33E-02	1,51E-01	+	7,71E-01	1,71E-01	+
F_8	1,09E-04	1,09E-04	9,67E+01	1,80E+01	+	5,71E+01	3,56E+01	+	9,71E+01	1,47E+01	+
F_9	0,00E+00	0,00E+00	2,76E+02	9,82E+01	+	9,76E+01	1,91E+01	+	2,99E+02	4,99E+01	+
F_{10}	3,37E-03	3,37E-03	4,03E+02	1,39E+01	+	4,00E+02	8,19E+00	+	3,99E+02	5,27E+00	+

TABLE 3.14 – Résultats statistiques de l'algorithme AHO en comparaison aux algorithmes CSsin, MP-EEH et RASPSHADE ($d = 15$).

	AHO		CSsin			MP-EEH			RASPSHADE		
	Mean	**STD**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**
F_1	8,59E-08	8,59E-08	3,33E+08	1,82E+09	+	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-
F_2	1,91E-04	1,91E-04	7,25E+01	5,99E+01	+	1,15E+02	8,46E+01	+	1,02E+00	1,33E+00	+
F_3	0,00E+00	0,00E+00	1,80E+01	1,25E+00	+	1,71E+01	5,03E+00	+	1,58E+01	2,29E-01	+
F_4	0,00E+00	0,00E+00	0,00E+00	0,00E+00	=	2,89E-01	1,31E-01	+	3,64E-01	4,57E-02	+
F_5	6,36E-05	6,36E-05	1,30E+01	6,19E+00	+	9,16E+01	4,80E+01	+	1,33E+00	8,38E-01	+
F_6	5,64E-05	5,64E-05	1,44E+00	2,85E+00	+	4,07E+00	5,90E+00	+	6,17E-01	2,08E-01	+
F_7	5,49E-05	5,49E-05	8,85E-01	1,96E-01	+	2,15E+01	3,89E+01	+	7,29E-01	1,24E-01	+
F_8	1,52E-04	1,52E-04	8,75E+01	2,76E+01	+	6,00E+01	4,98E+01	+	1,00E+02	0,00E+00	-
F_9	0,00E+00	0,00E+00	1,00E+02	2,30E-13	+	1,00E+02	2,19E-08	+	3,33E+02	4,08E+01	+
F_{10}	9,07E-03	9,07E-03	4,00E+02	0,00E+00	-	3,90E+02	4,03E+01	+	4,00E+02	0,00E+00	-

TABLE 3.15 – Résultats statistiques de l'algorithme AHO en comparaison aux algorithmes IMODE, DISH-XX et AGSK ($d = 15$).

	AHO		IMODE			DISH-XX			AGSK		
	Mean	**STD**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**
F_1	8,59E-08	8,59E-08	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-
F_2	1,91E-04	1,91E-04	3,14E+00	3,22E+00	+	7,13E+01	6,52E+01	+	1,85E+01	1,46E+01	+
F_3	0,00E+00	0,00E+00	1,61E+01	3,12E-01	+	1,66E+01	5,43E-01	+	1,42E+01	4,27E+00	+
F_4	0,00E+00	0,00E+00	0,00E+00	0,00E+00	=	0,00E+00	0,00E+00	=	1,42E-01	5,71E-02	+
F_5	6,36E-05	6,36E-05	7,79E+00	3,66E+00	+	2,18E+01	3,31E+01	+	6,25E+00	4,32E+00	+
F_6	5,64E-05	5,64E-05	6,92E-01	2,52E-01	+	9,40E+00	8,66E+00	+	4,02E-01	2,23E-01	+
F_7	5,49E-05	5,49E-05	5,30E-01	2,23E-01	+	6,33E-01	2,51E-01	+	2,47E-01	2,00E-01	+
F_8	1,52E-04	1,52E-04	4,18E+00	9,61E+00	+	1,00E+02	1,57E-13	-	6,85E+01	3,85E+01	+
F_9	0,00E+00	0,00E+00	9,33E+01	2,54E+01	+	3,90E+02	8,27E-01	+	9,67E+01	1,83E+01	+
F_{10}	9,07E-03	9,07E-03	4,00E+02	0,00E+00	-	4,00E+02	0,00E+00	-	4,00E+02	2,60E-13	-

TABLE 3.16 – Résultats statistiques de l'algorithme AHO en comparaison aux algorithmes j2020, jDE100e et OLSHADE ($d = 15$).

	AHO		j2020			jDE100e			OLSHADE		
	Mean	**STD**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**
F_1	8,59E-08	8,59E-08	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-
F_2	1,91E-04	1,91E-04	5,72E-02	4,32E-02	+	2,60E+00	2,52E+00	+	1,10E+02	9,21E+01	+
F_3	0,00E+00	0,00E+00	6,78E+00	7,82E+00	+	1,60E+01	4,02E-01	+	1,23E+01	6,03E+00	+
F_4	0,00E+00	0,00E+00	1,99E-01	7,47E-02	+	2,57E-01	6,92E-02	+	5,37E-01	1,42E-01	+
F_5	6,36E-05	6,36E-05	7,58E+00	7,69E+00	+	3,88E+00	2,29E+00	+	1,56E+01	4,06E+01	+
F_6	5,64E-05	5,64E-05	8,45E-01	2,09E+00	+	4,79E-01	2,12E-01	+	0,00E+00	0,00E+00	-
F_7	5,49E-05	5,49E-05	9,83E-01	2,03E+00	+	2,65E-01	1,38E-01	+	6,72E-01	1,82E-01	+
F_8	1,52E-04	1,52E-04	9,49E+00	2,74E+01	+	1,00E+02	0,00E+00	-	1,00E+02	0,00E+00	-
F_9	0,00E+00	0,00E+00	1,23E+02	5,68E+01	+	3,12E+02	1,30E+02	+	1,00E+02	0,00E+00	=
F_{10}	9,07E-03	9,07E-03	3,90E+02	5,48E+01	+	4,00E+02	0,00E+00	-	1,70E+02	4,66E+01	+

TABLE 3.17 – Résultats statistiques de l'algorithme AHO en comparaison aux algorithmes mpmLSHADE, SOMA-CL et GSK ($d = 15$).

	AHO		mpmLSHADE			SOMA-CL			GSK		
	Mean	**STD**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**
F_1	8,59E-08	8,59E-08	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-
F_2	1,91E-04	1,91E-04	1,83E-01	4,07E-01	+	1,82E+01	4,14E+01	+	1,78E+03	1,56E+02	+
F_3	0,00E+00	0,00E+00	1,56E+01	2,10E-13	+	1,61E+01	4,64E-01	+	4,79E+01	3,88E+00	+
F_4	0,00E+00	0,00E+00	3,82E-01	4,43E-02	+	7,04E-01	1,30E-01	+	3,29E+00	3,15E-01	+
F_5	6,36E-05	6,36E-05	5,40E-01	5,04E-01	+	7,38E+00	4,50E+00	+	1,01E+02	1,65E+01	+
F_6	5,64E-05	5,64E-05	7,25E-01	1,25E-01	+	6,52E+00	7,14E+00	+	4,62E+01	1,26E+01	+
F_7	5,49E-05	5,49E-05	5,43E-01	1,15E-01	+	2,69E+00	7,48E+00	+	9,51E+00	2,39E+00	+
F_8	1,52E-04	1,52E-04	1,00E+02	4,91E-13	-	8,07E+01	3,70E+01	+	1,00E+02	0,00E+00	-
F_9	0,00E+00	0,00E+00	3,90E+02	4,66E-01	+	2,30E+02	1,19E+02	+	4,13E+02	4,51E+00	+
F_{10}	9,07E-03	9,07E-03	4,00E+02	4,98E-13	-	4,00E+02	0,00E+00	-	4,00E+02	0,00E+00	-

TABLE 3.18 – Résultats statistiques de l'algorithme AHO en comparaison aux algorithmes CSsin, MP-EEH et RASPSHADE ($d = 20$).

	AHO		CSsin			MP-EEH			RASPSHADE		
	Mean	**STD**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**
F_1	5,09E-07	5,09E-07	9,33E+09	2,53E+09	+	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-
F_2	2,55E-04	2,55E-04	9,83E+01	8,33E+01	+	1,70E+02	9,42E+01	+	1,38E-01	4,54E-02	+
F_3	0,00E+00	0,00E+00	2,55E+01	2,27E+00	+	2,33E+01	6,13E+00	+	2,05E+01	1,89E-01	+
F_4	0,00E+00	0,00E+00	0,00E+00	0,00E+00	=	4,25E-01	1,41E-01	+	4,53E-01	4,18E-02	+
F_5	7,67E-05	7,67E-05	1,16E+02	6,34E+01	+	2,36E+02	7,80E+01	+	1,41E+00	1,25E+00	+
F_6	1,00E-04	1,00E-04	6,72E-01	8,22E+00	+	4,27E+01	5,23E+01	+	1,71E-01	5,74E-02	+
F_7	6,74E-05	6,74E-05	2,62E+00	2,26E+00	+	7,77E+01	6,52E+01	+	7,27E-01	2,40E-01	+
F_8	2,00E-04	2,00E-04	9,89E+01	5,59E+00	+	8,00E+01	4,07E+01	+	1,00E+02	0,00E+00	-
F_9	0,00E+00	0,00E+00	1,03E+02	1,82E+01	+	9,67E+01	1,83E+01	+	3,42E+02	3,51E+01	+
F_{10}	8,83E-03	8,83E-03	3,99E+02	2,44E-01	+	4,01E+02	3,78E+00	+	4,14E+02	1,14E-13	-

TABLE 3.19 – Résultats statistiques de l'algorithme AHO en comparaison aux algorithmes IMODE, DISH-XX et AGSK ($d = 20$).

	AHO		IMODE			DISH-XX			AGSK		
	Mean	**STD**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**
F_1	5,09E-07	5,09E-07	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-
F_2	2,55E-04	2,55E-04	5,13E-01	7,13E-01	+	8,67E+01	1,11E+02	+	9,68E-01	1,23E+00	+
F_3	0,00E+00	0,00E+00	2,05E+01	1,26E-01	+	2,13E+01	3,74E+00	+	2,04E+01	0,00E+00	=
F_4	0,00E+00	0,00E+00	0,00E+00	0,00E+00	=	2,47E-04	1,35E-03	+	1,45E-01	5,47E-02	+
F_5	7,67E-05	7,67E-05	1,09E+01	4,33E+00	+	5,63E+01	6,63E+01	+	4,50E+01	3,67E+01	+
F_6	1,00E-04	1,00E-04	3,02E-01	8,17E-02	+	1,50E+01	3,57E+01	+	1,68E-01	4,45E-02	+
F_7	6,74E-05	6,74E-05	5,24E-01	1,64E-01	+	5,09E+00	6,42E+00	+	6,81E-01	9,09E-01	+
F_8	2,00E-04	2,00E-04	8,40E+01	1,89E+01	+	1,00E+02	1,39E-13	-	9,92E+01	4,63E+00	+
F_9	0,00E+00	0,00E+00	9,67E+01	1,83E+01	+	4,05E+02	2,50E+00	+	1,00E+02	8,30E-14	+
F_{10}	8,83E-03	8,83E-03	4,00E+02	6,18E-01	+	4,14E+02	2,54E-02	+	3,99E+02	1,59E-02	+

TABLE 3.20 – Résultats statistiques de l'algorithme AHO en comparaison aux algorithmes j2020, jDE100e et OLSHADE ($d = 20$).

	AHO		**j2020**			**jDE100e**			**OLSHADE**		
	Mean	**STD**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**
F_1	5,09E-07	5,09E-07	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-
F_2	2,55E-04	2,55E-04	2,60E-02	2,47E-02	+	1,48E+00	1,50E+00	+	1,15E+00	7,62E+01	+
F_3	0,00E+00	0,00E+00	1,44E+01	9,29E+00	+	2,10E+01	4,09E-01	+	2,52E+00	7,63E+00	+
F_4	0,00E+00	0,00E+00	1,80E-01	7,84E-02	+	3,47E-01	8,04E-02	+	1,01E+00	1,22E+00	+
F_5	7,67E-05	7,67E-05	7,78E+01	5,75E+01	+	2,37E+00	8,49E-01	+	1,78E+00	4,14E+01	+
F_6	1,00E-04	1,00E-04	1,92E-01	1,01E-01	+	1,15E-01	3,31E-02	+	5,17E-01	0,00E+00	-
F_7	6,74E-05	6,74E-05	1,98E+00	4,02E+00	+	2,14E-01	1,14E-01	+	8,42E-01	1,61E-01	+
F_8	2,00E-04	2,00E-04	9,27E+01	2,21E+01	+	1,00E+02	0,00E+00	-	1,00E+00	7,01E-01	+
F_9	0,00E+00	0,00E+00	3,39E+02	1,28E+02	+	4,05E+02	1,74E+00	+	1,10E+00	3,06E+01	+
F_{10}	8,83E-03	8,83E-03	3,99E+02	4,02E-02	+	4,13E+02	2,78E+00	+	4,10E+00	6,15E+00	+

TABLE 3.21 – Résultats statistiques de l'algorithme AHO en comparaison aux algorithmes mpmLSHADE, SOMA-CL et GSK ($d = 20$).

	AHO		**mpmLSHADE**			**SOMA-CL**			**GSK**		
	Mean	**STD**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**	**Mean**	**STD**	**P**
F_1	5,09E-07	5,09E-07	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-	0,00E+00	0,00E+00	-
F_2	2,55E-04	2,55E-04	3,97E-02	2,12E-02	+	7,36E+00	2,15E+01	+	2,68E+03	1,60E+02	+
F_3	0,00E+00	0,00E+00	2,04E+01	4,67E-13	+	2,14E+01	8,06E-01	+	7,37E+01	5,25E+00	+
F_4	0,00E+00	0,00E+00	4,97E-01	4,23E-02	+	1,05E+00	1,83E-01	+	5,37E+00	4,25E-01	+
F_5	7,67E-05	7,67E-05	1,38E+00	1,45E+00	+	1,45E+02	8,01E+01	+	2,44E+02	3,97E+01	+
F_6	1,00E-04	1,00E-04	2,05E-01	4,71E-02	+	2,71E-01	8,35E-02	+	3,35E+00	2,17E+00	+
F_7	6,74E-05	6,74E-05	5,10E-01	1,19E-01	+	9,22E+00	8,93E+00	+	5,86E+01	1,09E+01	+
F_8	2,00E-04	2,00E-04	1,00E+02	7,47E-13	-	9,91E+01	3,75E+00	+	1,00E+02	0,00E+00	-
F_9	0,00E+00	0,00E+00	4,01E+02	6,68E-01	+	3,99E+02	4,54E+01	+	4,39E+02	2,95E+01	+
F_{10}	8,83E-03	8,83E-03	4,14E+02	2,74E-04	-	4,71E+02	3,23E+01	+	4,14E+02	8,87E-03	+

TABLE 3.22 – Test des rangs signés de Wilcoxon pour $d = 5$.

	k	$W+$	$W-$	$\min(W+,W-)$	Valeur critique	Résultats du test des rangs signés de Wilcoxon
AHO vs. CSsin	7	0	28	0	4	$4 > 0$, AHO surpasse CSsin
AHO vs. MP-EEH	7	1	27	1	4	$4 > 1$, AHO surpasse MP-EEH
AHO vs. RASPSHADE	7	3	25	3	4	$4 > 3$, AHO surpasse RASPSHADE
AHO vs. IMODE	5	6	9	6	1	$1 < 6$, IMODE surpasse AHO
AHO vs. DISH-XX	7	0	28	0	4	$4 > 0$, AHO surpasse DISH-XX
AHO vs. AGSK	7	3	25	3	4	$4 > 3$, AHO surpasse AGSK
AHO vs. j2020	7	0	28	0	4	$4 > 0$, AHO surpasse j2020
AHO vs. jDE100e	7	0	28	0	4	$4 > 0$, AHO surpasse jDE100e
AHO vs. OLSHADE	7	3	25	3	4	$4 > 3$, AHO surpasse OLSHADE
AHO vs. mpmLSHADE	7	1	27	1	4	$4 > 1$, AHO surpasse mpmLSHADE
AHO vs. SOMA-CL	8	0	36	0	6	$6 > 0$, AHO surpasse SOMA-CL
AHO vs. GSK	7	0	28	0	4	$4 > 0$, AHO surpasse GSK

TABLE 3.23 – Test des rangs signés de Wilcoxon pour $d = 10$.

	k	$W+$	$W-$	$\min(W+,W-)$	Valeur critique	Résultats du test des rangs signés de Wilcoxon
AHO vs. CSsin	9	2	43	2	8	$8 > 2$, AHO surpasse CSsin
AHO vs. MP-EEH	10	1	54	1	11	$11 > 1$, AHO surpasse MP-EEH
AHO vs. RASPSHADE	9	3	42	3	8	$8 > 3$, AHO surpasse RASPSHADE
AHO vs. IMODE	9	4	41	4	8	$8 > 4$, AHO surpasse IMODE
AHO vs. DISH-XX	10	3	52	3	11	$11 > 3$, AHO surpasse DISH-XX
AHO vs. AGSK	10	1	54	1	11	$11 > 1$, AHO surpasse AGSK
AHO vs. j2020	10	1	54	1	11	$11 > 1$, AHO surpasse j2020
AHO vs. jDE100e	10	1	54	1	11	$11 > 1$, AHO surpasse jDE100e
AHO vs. OLSHADE	9	6	39	6	8	$8 > 6$, AHO surpasse OLSHADE
AHO vs. mpmLSHADE	10	1	54	1	11	$11 > 1$, AHO surpasse mpmLSHADE
AHO vs. SOMA-CL	10	1	54	1	11	$11 > 1$, AHO surpasse SOMA-CL
AHO vs. GSK	10	1	54	1	11	$11 > 1$, AHO surpasse GSK

TABLE 3.24 – Test des rangs signés de Wilcoxon pour $d = 15$.

	k	$W+$	$W-$	$\min(W+,W-)$	Valeur critique	Résultats du test des rangs signés de Wilcoxon
AHO vs. CSsin	9	2	43	2	8	$8 > 2$, AHO surpasse CSsin
AHO vs. MP-EEH	10	2	53	2	11	$11 > 2$, AHO surpasse MP-EEH
AHO vs. RASPSHADE	10	6	49	6	11	$11 > 6$, AHO surpasse RASPSHADE
AHO vs. IMODE	9	3	42	3	8	$8 > 3$, AHO surpasse IMODE
AHO vs. DISH-XX	9	6	39	6	8	$8 > 6$, AHO surpasse DISH-XX
AHO vs. AGSK	10	3	52	3	11	$11 > 3$, AHO surpasse AGSK
AHO vs. j2020	10	1	54	1	11	$11 > 1$, AHO surpasse j2020
AHO vs. jDE100e	10	6	49	6	11	$11 > 6$, AHO surpasse jDE100e
AHO vs. OLSHADE	9	6	39	6	8	$8 > 6$, AHO surpasse OLSHADE
AHO vs. mpmLSHADE	10	9	46	9	11	$11 > 9$, AHO surpasse mpmLSHADE
AHO vs. SOMA-CL	10	3	52	3	11	$11 > 3$, AHO surpasse SOMA-CL
AHO vs. GSK	10	6	49	6	11	$11 > 6$, AHO surpasse GSK

TABLE 3.25 – Test des rangs signés de Wilcoxon pour $d = 20$.

	k	$W+$	$W-$	$\min(W+,W-)$	Valeur critique	Résultats du test des rangs signés de Wilcoxon
AHO vs. CSsin	9	0	45	0	8	$8 > 0$, AHO surpasse CSsin
AHO vs. MP-EEH	10	1	54	1	11	$11 > 1$, AHO surpasse MP-EEH
AHO vs. RASPSHADE	10	6	49	6	11	$11 > 6$, AHO surpasse RASPSHADE
AHO vs. IMODE	9	1	44	1	8	$8 > 1$, AHO surpasse IMODE
AHO vs. DISH-XX	10	3	52	3	11	$11 > 3$, AHO surpasse DISH-XX
AHO vs. AGSK	9	2	43	2	8	$8 > 2$, AHO surpasse AGSK
AHO vs. j2020	10	1	54	1	11	$11 > 1$, AHO surpasse j2020
AHO vs. jDE100e	10	3	52	3	11	$11 > 3$, AHO surpasse jDE100e
AHO vs. OLSHADE	10	3	52	3	11	$11 > 3$, AHO surpasse OLSHADE
AHO vs. mpmLSHADE	10	9	46	9	11	$11 > 9$, AHO surpasse mpmLSHADE
AHO vs. SOMA-CL	10	1	54	1	11	$11 > 1$, AHO surpasse SOMA-CL
AHO vs. GSK	10	4	51	4	11	$11 > 4$, AHO surpasse GSK

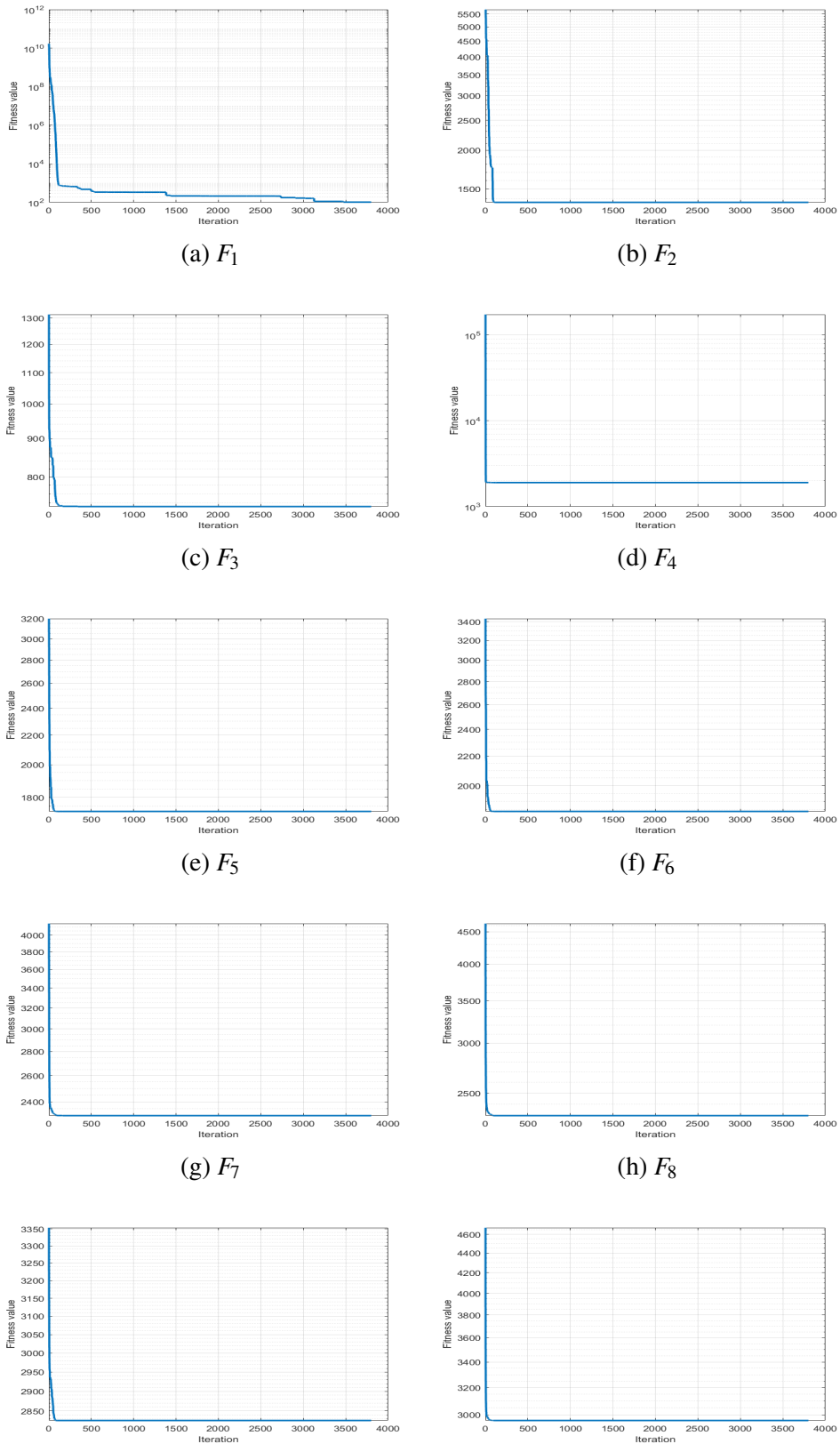
considérés pour l'étude comparative pour toutes les dimensions, à l'exception de la dimension $d = 5$ où l'algorithme AHO est surpassé par l'algorithme IMODE.

La figure 3.5 dépeint les courbes de convergence de l'algorithme AHO pour les dix fonctions de test du benchmark CEC 2020 lorsque $d = 20$ sur 30 exécutions indépendantes. Nous observons que l'exploitation de l'algorithme AHO sur la plupart des fonctions de test est dominante, conduisant à une convergence plus rapide vers l'optimum global car la phase d'exploitation est renforcée par les équations 3.4 et 3.5.

3.5 Conclusion et perspectives

Dans ce chapitre, une nouvelle métaheuristique basée-population, que nous avons baptisée Archerfish Hunting Optimizer (AHO), est introduite pour résoudre les problèmes d'optimisation sans contraintes. L'algorithme AHO est inspiré par les comportements de tir et de saut des poissons archer dans la nature lors de la chasse aux insectes aériens. Certaines équations sont proposées et décrites pour modéliser le comportement de chasse des poissons archer afin de résoudre des problèmes d'optimisation. Dix problèmes d'optimisation sans contraintes (benchmark CEC 2020) sont utilisés pour évaluer les performances de l'algorithme AHO. Les capacités d'exploration, d'exploitation et d'évitement des optimums locaux sont examinées à l'aide des fonctions de test du benchmark CEC 2020. Les résultats statistiques obtenus des tests des rangs signés de Wilcoxon et Friedman confirment que l'algorithme AHO est très compétitif par rapport aux 12 algorithmes métaheuristiques utilisés pour l'études comparative.

L'algorithme AHO est expliqué de manière simple avec des techniques d'exploration et d'exploitation élémentaires. Il est souhaitable d'introduire d'autres schémas différentiels tels que la mutation, le croisement ou le multi-essaim, ce que nous prévoyons de faire à l'avenir. De plus, nous prévoyons de développer les versions binaires et multi-objective de l'algorithme AHO.



(a) F_1 (b) F_2

(c) F_3 (d) F_4

(e) F_5 (f) F_6

(g) F_7 (h) F_8

(i) F_9 (j) F_{10}

FIGURE 3.5 – Courbes de convergence de l'algorithme AHO pour les fonctions de test 20d du Benchmark CEC 2020 (moyennées sur 30 exacutions indépendantes).

Conclusion générale et perspectives

Dans ce mémoire, nous avons présenté un nouvel algorithme métaheuristique pour l'optimisation globale continue : Archerfish Hunting Optimizer (AHO). Nous avons détaillé son fonctionnement, analysé sa complexité temporelle, étudié la sensibilité de ses paramètres de contrôle, ainsi que ses performances sur plusieurs fonctions d'optimisation.

L'algorithme AHO est inspiré par les comportements de tir et de saut des poissons archer dans la nature lors de la capture des insectes aériens. L'algorithme AHO a été lancé sur un ensemble de fonctions d'optimisation permettant ainsi de tester et valider ses performances sur différents profils de fonctions objective, et en différentes dimensions (5, 10, 15 et 20).

Dans des travaux futurs, il serait intéressant d'appliquer l'algorithme AHO sur des problèmes issus du monde réel, tels que les problèmes de traitement d'images, les problèmes d'optimisation de méthodes d'apprentissage, ou bien des problèmes en plus grandes dimensions, afin de pouvoir observer ses performances dans d'autres domaines d'application.

De plus, l'algorithme proposé dans ce mémoire est mono-objective, or en optimisation il existe de nombreux problèmes qui sont caractérisés de multi-objective, c'est-à-dire que plusieurs fonctions objective sont à considérer en même temps. Il pourrait être intéressant d'étudier des versions multi-objectives de l'algorithme AHO. Pour cela, il existe déjà des techniques se fondant notamment sur le front de Pareto pour pouvoir adapter les algorithmes du mono-objective au multi-objective.

Bibliographie

[1] N. WIENER, *The human use of human beings : Cybernetics and society*, 320. Da Capo Press, 1988 (cf. p. 2).

[2] I GUYON, « Neural networks and applications tutorial, » *Physics Reports*, t. 207, n° 3-5, p. 215-259, 1991 (cf. p. 2).

[3] P. J. VAN LAARHOVEN et E. H. AARTS, « Simulated annealing, » in *Simulated annealing : Theory and applications*, Springer, 1987, p. 7-15 (cf. p. 2, 3, 13, 17, 38).

[4] J. R. SAMPSON, *Adaptation in natural and artificial systems (John H. Holland)*, 1976 (cf. p. 2, 18).

[5] U. AICKELIN et D. DASGUPTA, « Artificial immune systems, » in *Search methodologies*, Springer, 2005, p. 375-399 (cf. p. 2).

[6] J.-T. AMENYO, « Automating DNA-based Molecular Computing via Traditional Practices of Parallel Computer, » *DNA Based Computers Two*, t. 44, p. 133, 1999 (cf. p. 2).

[7] A. STEANE, « Quantum computing, » *Reports on Progress in Physics*, t. 61, n° 2, p. 117, 1998 (cf. p. 2).

[8] G. PĂUN, « Membrane computing, » in *International Symposium on Fundamentals of Computation Theory*, Springer, 2003, p. 284-295 (cf. p. 2).

[9] S. WOLFRAM, « Statistical mechanics of cellular automata, » *Reviews of modern physics*, t. 55, n° 3, p. 601, 1983 (cf. p. 2).

[10] J. D. PINTÉR, *Global optimization in action : continuous and Lipschitz optimization : algorithms, implementations and applications*. Springer Science & Business Media, 2013, t. 6 (cf. p. 2).

[11] A. NEUMAIER et A. NEUMAIER, *Interval methods for systems of equations*. Cambridge university press, 1990, t. 37 (cf. p. 2).

[12] R. HORST et P. M. PARDALOS, *Handbook of global optimization*. Springer Science & Business Media, 2013, t. 2 (cf. p. 2).

[13] C. W. GARDINER et al., *Handbook of stochastic methods*. springer Berlin, 1985, t. 3 (cf. p. 2).

[14] M. DORIGO, G. D. CARO et L. M. GAMBARDELLA, « Ant algorithms for discrete optimization, » *Artificial life*, t. 5, n° 2, p. 137-172, 1999 (cf. p. 3, 27, 32, 38).

[15] J. R. KOZA et J. R. KOZA, *Genetic programming : on the programming of computers by means of natural selection*. MIT press, 1992, t. 1 (cf. p. 3, 19, 32).

[16] J. H. HOLLAND et al., *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992 (cf. p. 3, 25, 26).

[17] F. GLOVER, « Tabu search—part I, » *ORSA Journal on computing*, t. 1, n° 3, p. 190-206, 1989 (cf. p. 3).

[18] M. AO LEITE, B. DELINCHANT, J.-M. GUICHON et J. A. VASCONCELOS, « Simplex-based adaptive parametric model order reduction for applications in optimization, » *International Journal of Numerical Modelling : Electronic Networks, Devices and Fields*, t. 32, n° 4, e2264, 2019 (cf. p. 6).

[19] A. D. BELEGUNDU et T. R. CHANDRUPATLA, *Optimization concepts and applications in engineering*. Cambridge University Press, 2019 (cf. p. 6).

[20] A. A. HEIDARI, S. MIRJALILI, H. FARIS, I. ALJARAH, M. MAFARJA et H. CHEN, « Harris hawks optimization : Algorithm and applications, » *Future generation computer systems*, t. 97, p. 849-872, 2019 (cf. p. 6, 32).

[21] M. WATTERSON, S. LIU, K. SUN, T. SMITH et V. KUMAR, « Trajectory optimization on manifolds with applications to quadrotor systems, » *The International Journal of Robotics Research*, p. 0 278 364 919 891 775, 2020 (cf. p. 6).

[22] B. M. OZYILDIRIM et M. KIRAN, « Do optimization methods in deep learning applications matter ? » *arXiv preprint arXiv :2002.12642*, 2020 (cf. p. 6).

[23] A. SADOLLAH, M. NASIR et Z. W. GEEM, « Sustainability and Optimization : From Conceptual Fundamentals to Applications, » *Sustainability*, t. 12, n° 5, p. 2027, 2020 (cf. p. 6).

[24] T.-T. NGUYEN, L.-H. CAO, T.-A. NGUYEN et X.-P. DANG, « Multi-response optimization of the roller burnishing process in terms of energy consumption and product quality, » *Journal of Cleaner Production*, t. 245, p. 119 328, 2020 (cf. p. 6).

[25] X. LI, C. LUO, H. JI, Y. ZHUANG, H. ZHANG et V. C. LEUNG, « Energy consumption optimization for self-powered IoT networks with non-orthogonal multiple access, » *International Journal of Communication Systems*, t. 33, n° 1, e4174, 2020 (cf. p. 6).

[26] C.-B. YAN et Z. ZHENG, « Problem Formulation and Solution Methodology for Energy Consumption Optimization in Bernoulli Serial Lines, » *IEEE Transactions on Automation Science and Engineering*, 2020 (cf. p. 6).

[27] D. HE, Y. YANG, Y. CHEN, J. DENG, S. SHAN, J. LIU et X. LI, « An integrated optimization model of metro energy consumption based on regenerative energy and passenger transfer, » *Applied Energy*, t. 264, p. 114 770, 2020 (cf. p. 6).

[28] Z. MENG, J.-S. PAN et W.-m. ZHENG, « Differential evolution utilizing a handful top superior individuals with bionic bi-population structure for the enhancement of optimization performance, » *Enterprise Information Systems*, t. 14, n° 2, p. 221-242, 2020 (cf. p. 6).

[29] X. ZHAO, R. XIA, H. GU, X. KE, Y. SHI, X. CHEN, H. JIANG, H.-L. YIP et S. LIU, « Performance optimization of tandem organic solar cells at varying incident angles based on optical analysis method, » *Optics Express*, t. 28, n° 2, p. 2381-2397, 2020 (cf. p. 6).

[30] X. JIANG, F. PAN et Y. WANG, « Optimization Design of Surface Accuracy of Shape Memory Cable Net Structure, » in *Proceedings of the Seventh Asia International Symposium on Mechatronics*, Springer, 2020, p. 56-64 (cf. p. 6).

[31] R. NIE, B. HE, S. YAN et X. MA, « Optimization design method for mesh reflector antennas considering the truss deformation and thermal effects, » *Engineering Structures*, t. 208, p. 110-253, 2020 (cf. p. 6).

[32] N DELABY, S MARTIN, A BARATEAU, O HENRY, N PERICHON, R DE CREVOISIER, E CHAJON, J CASTELLI et C LAFOND, « Implementation of an optimization method for parotid gland sparing during inverse planning for head and neck cancer radiotherapy, » *Cancer/Radiothérapie*, 2020 (cf. p. 6).

[33] A. RAHIMI, M. RÖNNQVIST, L. LEBEL et J.-F. AUDY, « An optimization model for selecting wood supply contracts, » *Canadian Journal of Forest Research*, t. 50, n° 999, p. 399-412, 2020 (cf. p. 6).

[34] J. LIU, Q. WANG, C. HE, K. JAFFRÈS-RUNSER, Y. XU, Z. LI et Y. XU, « QMR : Q-learning based Multi-objective optimization Routing protocol for Flying Ad Hoc Networks, » *Computer Communications*, t. 150, p. 304-316, 2020 (cf. p. 6).

[35] S. MASKALIK, W. WU, D. BASAK, S. THAKKAR et A. SEQUEIRA, *Routing optimization for intercloud connectivity*, US Patent 10,547,540, 2020 (cf. p. 6).

[36] N. ANDREASSON, A. EVGRAFOV et M. PATRIKSSON, *An introduction to continuous optimization : foundations and fundamental algorithms*. Courier Dover Publications, 2020 (cf. p. 6).

[37] M. REZAEIAHARI et M. T. KHASAWNEH, « Simulation optimization approach for patient scheduling at destination medical centers, » *Expert Systems with Applications*, t. 140, p. 112 881, 2020 (cf. p. 6).

[38] B. EVERITT, *Introduction to optimization methods and their application in statistics*. Springer Science & Business Media, 2012 (cf. p. 6, 38).

[39] L. J. HONG et B. L. NELSON, « A brief introduction to optimization via simulation, » in *Proceedings of the 2009 Winter Simulation Conference (WSC)*, IEEE, 2009, p. 75-85 (cf. p. 6).

[40] G. R. WOOD, « The bisection method in higher dimensions, » *Mathematical programming*, t. 55, n° 1-3, p. 319-337, 1992 (cf. p. 6).

[41] A EIGER, K. SIKORSKI et F. STENGER, « A bisection method for systems of nonlinear equations, » *ACM Transactions on Mathematical Software (TOMS)*, t. 10, n° 4, p. 367-377, 1984 (cf. p. 6).

[42] T. J. YPMA, « Historical development of the Newton–Raphson method, » *SIAM review*, t. 37, n° 4, p. 531-551, 1995 (cf. p. 7, 16, 22, 35).

[43] A. BEN-ISRAEL, « A Newton-Raphson method for the solution of systems of equations, » *Journal of Mathematical analysis and applications*, t. 15, n° 2, p. 243-252, 1966 (cf. p. 7, 16, 22, 35).

[44] J. HADAMARD, *La série de Taylor et son prolongement analytique*, 12. C. Hérissey, 1901 (cf. p. 7).

[45] L. TVEDT, « Distribution of quadratic forms in normal space—application to structural reliability, » *Journal of engineering mechanics*, t. 116, n° 6, p. 1183-1197, 1990 (cf. p. 9).

[46] J.-B. HIRIART-URRUTY, J.-J. STRODIOT et V. H. NGUYEN, « Generalized Hessian matrix and second-order optimality conditions for problems withC 1, 1 data, » *Applied mathematics and optimization*, t. 11, n° 1, p. 43-56, 1984 (cf. p. 10).

[47] J. FLIEGE et B. F. SVAITER, « Steepest descent methods for multicriteria optimization, » *Mathematical Methods of Operations Research*, t. 51, n° 3, p. 479-494, 2000 (cf. p. 10).

[48] L. G. DRUMMOND et B. F. SVAITER, « A steepest descent method for vector optimization, » *Journal of computational and applied mathematics*, t. 175, n° 2, p. 395-414, 2005 (cf. p. 10).

[49] J. KENNEDY et R. EBERHART, « Particle swarm optimization, » in *Proceedings of ICNN'95-International Conference on Neural Networks*, IEEE, t. 4, 1995, p. 1942-1948 (cf. p. 13, 17, 19, 23, 28, 35, 38).

[50] X.-S. YANG, « Firefly algorithms for multimodal optimization, » in *International symposium on stochastic algorithms*, Springer, 2009, p. 169-178 (cf. p. 13, 17, 28).

[51] T. BACK, *Evolutionary algorithms in theory and practice : evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996 (cf. p. 13).

[52] K. DEB, *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, 2001, t. 16 (cf. p. 13).

[53] D. WHITLEY, « A genetic algorithm tutorial, » *Statistics and computing*, t. 4, n° 2, p. 65-85, 1994 (cf. p. 13, 17).

[54] A. DANESHMAND, F. FACCHINEI, V. KUNGURTSEV et G. SCUTARI, « Hybrid random/deterministic parallel algorithms for convex and nonconvex big data optimization, » *IEEE Transactions on Signal Processing*, t. 63, n° 15, p. 3914-3929, 2015 (cf. p. 13).

[55] A. BASHIROV, *Mathematical Analysis Fundamentals*. Elsevier Science, 2016 (cf. p. 14).

[56] P. H. WINSTON et S. A. SHELLARD, *Artificial intelligence at MIT : expanding frontiers*. MIT Press, 1990 (cf. p. 14).

[57] X. YAO et Y. LIU, « Fast Evolutionary Programming., » *Evolutionary programming*, t. 3, p. 451-460, 1996 (cf. p. 14).

[58] D. ORTIZ-BOYER, C. HERVÁS-MARTÍNEZ et N. GARCÍA-PEDRAJAS, « CIXL2 : a crossover operator for evolutionary algorithms based on population features, » *Journal of Artificial Intelligence Research*, t. 24, p. 1-48, 2005 (cf. p. 14).

[59] M. JAMIL, X.-S. YANG et H.-J. ZEPERNICK, « Test functions for global optimization : a comprehensive survey, » in *Swarm intelligence and Bio-inspired Computation*, Elsevier, 2013, p. 193-222 (cf. p. 14).

[60] C.-J. CHUNG et R. G. REYNOLDS, « CAEP : An evolution-based tool for real-valued function optimization using cultural algorithms, » *International Journal on Artificial Intelligence Tools*, t. 7, n° 03, p. 239-291, 1998 (cf. p. 14).

[61] D. H. WOLPERT et W. G. MACREADY, « No free lunch theorems for optimization, » *IEEE transactions on evolutionary computation*, t. 1, n° 1, p. 67-82, 1997 (cf. p. 15, 19).

[62] X.-S. YANG, *Nature-inspired metaheuristic algorithms*. Luniver press, 2010 (cf. p. 15-17, 25, 28, 29, 35).

[63] T. C. SERVICE, « A No Free Lunch theorem for multi-objective optimization, » *Information Processing Letters*, t. 110, n° 21, p. 917-923, 2010 (cf. p. 15).

[64] A. AUGER et O. TEYTAUD, « Continuous lunches are free plus the design of optimal optimization algorithms, » *Algorithmica*, t. 57, n° 1, p. 121-146, 2010 (cf. p. 16).

[65] J. A. MARSHALL et T. G. HINTON, « Beyond no free lunch : realistic algorithms for arbitrary problem classes, » in *IEEE Congress on Evolutionary Computation*, IEEE, 2010, p. 1-6 (cf. p. 16).

[66] J. A. NELDER et R. MEAD, « A simplex method for function minimization, » *The computer journal*, t. 7, n° 4, p. 308-313, 1965 (cf. p. 16).

[67] I. MOSER, « Hooke-jeeves revisited, » in *2009 IEEE Congress on Evolutionary Computation*, IEEE, 2009, p. 2670-2676 (cf. p. 16).

[68] N. HANSEN, « Benchmarking the Nelder-Mead downhill simplex algorithm with many local restarts, » in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference : Late Breaking Papers*, 2009, p. 2403-2408 (cf. p. 16).

[69] R. MARTÍ, M. GALLEGO, A. DUARTE et E. G. PARDO, « Heuristics and metaheuristics for the maximum diversity problem, » *Journal of Heuristics*, t. 19, n° 4, p. 591-615, 2013 (cf. p. 16).

[70] B. J. COPELAND, *The essential turing*. Clarendon Press, 2004 (cf. p. 17, 18).

[71] A. TURING, « Intelligent machinery (1948), » *B. Jack Copeland*, p. 395, 2004 (cf. p. 17, 18).

[72] K. JONG et A. DE HAVEN BRANDON, « An analysis of the behavior of a class of genetic adaptive systems, » thèse de doct., Engineering, College of - Technical Reports, 1975 (cf. p. 18).

[73] J. GENLIN, « Survey on genetic algorithm [J], » *Computer Applications and Software*, t. 2, p. 69-73, 2004 (cf. p. 18).

[74] J. ISLAM, P. M. VASANT, B. M. NEGASH, M. B. LARUCCIA et M. MYINT, « A Survey of Nature-Inspired Algorithms With Application to Well Placement Optimization, » in *Deep Learning Techniques and Optimization Strategies in Big Data Analytics*, IGI Global, 2020, p. 32-45 (cf. p. 18).

[75] R. SARKAR, D. BARMAN et N. CHOWDHURY, « A Cooperative Co-evolutionary Genetic Algorithm for Multi-Robot Path Planning Having Multiple Targets, » in *Computational Intelligence in Pattern Recognition*, Springer, 2020, p. 727-740 (cf. p. 18).

[76] J. MA, Y. LIU, S. ZANG et L. WANG, « Robot Path Planning Based on Genetic Algorithm Fused with Continuous Bezier Optimization, » *Computational Intelligence and Neuroscience*, t. 2020, 2020 (cf. p. 18).

[77] M AFIF, A GHAREB, A. SAIF, A BAKAR et O. BAZIGHIFAN, « Genetic algorithm rule based categorization method for textual data mining, » *Decision Science Letters*, t. 9, n° 1, p. 37-50, 2020 (cf. p. 18).

[78] J. LIU, Y. LIU, Y. SHI et J. LI, « Solving Resource-Constrained Project Scheduling Problem via Genetic Algorithm, » *Journal of Computing in Civil Engineering*, t. 34, n° 2, p. 04 019 055, 2020 (cf. p. 18).

[79] I. H. SIN et B. DO CHUNG, « Bi-objective optimization approach for energy aware scheduling considering electricity cost and preventive maintenance using genetic algorithm, » *Journal of Cleaner Production*, t. 244, p. 118 869, 2020 (cf. p. 18).

[80] L. R. ABREU, J. O. CUNHA, B. A. PRATA et J. M. FRAMINAN, « A genetic algorithm for scheduling open shops with sequence-dependent setup times, » *Computers & Operations Research*, t. 113, p. 104 793, 2020 (cf. p. 18).

[81] I. RECHENBERG, « Evolution strategy : Nature's way of optimization, » in *Optimization : Methods and applications, possibilities and limitations*, Springer, 1989, p. 106-126 (cf. p. 18).

[82] H.-G. BEYER et H.-P. SCHWEFEL, « Evolution strategies–A comprehensive introduction, » *Natural computing*, t. 1, n° 1, p. 3-52, 2002 (cf. p. 18).

[83] X.-S. YANG, « Review of metaheuristics and generalized evolutionary walk algorithm, » *arXiv preprint arXiv :1105.3668*, 2011 (cf. p. 18).

[84] L. J. FOGEL, A. J. OWENS et M. J. WALSH, *Artificial intelligence through simulated evolution*. Wiley, 1966 (cf. p. 18, 32).

[85] R. DECHTER, H. GEFFNER et J. Y. HALPERN, *Heuristics, Probability and Causality : A Tribute to Judea Pearl*. College Publications, 2010 (cf. p. 18).

[86] F. GLOVER et M. LAGUNA, « Tabu search, » in *Handbook of combinatorial optimization*, Springer, 1998, p. 2093-2229 (cf. p. 18).

[87] M. GEVREY, I. DIMOPOULOS et S. LEK, « Review and comparison of methods to study the contribution of variables in artificial neural network models, » *Ecological modelling*, t. 160, n° 3, p. 249-264, 2003 (cf. p. 18).

[88] P. C. DEKA et al., « Support vector machine applications in the field of hydrology : a review, » *Applied soft computing*, t. 19, p. 372-386, 2014 (cf. p. 18).

[89] Y. SINGH, P. K. BHATIA et O. SANGWAN, « A review of studies on machine learning techniques, » *International Journal of Computer Science and Security*, t. 1, n° 1, p. 70-84, 2007 (cf. p. 18).

[90] H. T. SIEGELMANN et E. D. SONTAG, « Turing computability with neural nets, » *Applied Mathematics Letters*, t. 4, n° 6, p. 77-80, 1991 (cf. p. 18).

[91] V. VAPNIK, *The nature of statistical learning theory*. Springer science & business media, 2013 (cf. p. 18).

[92] S. KIRKPATRICK, C. D. GELATT et M. P. VECCHI, « Optimization by simulated annealing, » *science*, t. 220, n° 4598, p. 671-680, 1983 (cf. p. 18, 23, 26, 33, 34).

[93] M. DORIGO, « Optimization, learning and natural algorithms, » *PhD Thesis, Politecnico di Milano*, 1992 (cf. p. 18, 27).

[94] S. MIRJALILI, J. S. DONG, A. LEWIS et A. S. SADIQ, « Particle swarm optimization : theory, literature review, and application in airfoil design, » in *Nature-inspired optimizers*, Springer, 2020, p. 167-184 (cf. p. 19).

[95] R. STORN et K. PRICE, « Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, » *Journal of global optimization*, t. 11, n° 4, p. 341-359, 1997 (cf. p. 19, 27, 32).

[96] S. YAGIZ, A. YAZITOVA et H. KARAHAN, « Application of differential evolution algorithm and comparing its performance with literature to predict rock brittleness for excavatability, » *International Journal of Mining, Reclamation and Environment*, p. 1-14, 2020 (cf. p. 19).

[97] D. H. WOLPERT et W. G. MACREADY, « Coevolutionary free lunches, » *IEEE Transactions on evolutionary computation*, t. 9, n° 6, p. 721-735, 2005 (cf. p. 19).

[98] X.-S. YANG, « Nature-inspired optimization algorithms : challenges and open problems, » *Journal of Computational Science*, p. 101 104, 2020 (cf. p. 19).

[99] J. DEL SER, E. OSABA, D. MOLINA, X.-S. YANG, S. SALCEDO-SANZ, D. CAMACHO, S. DAS, P. N. SUGANTHAN, C. A. C. COELLO et F. HERRERA, « Bio-inspired computation : Where we stand and what's next, » *Swarm and Evolutionary Computation*, t. 48, p. 220-250, 2019 (cf. p. 19, 28, 31).

[100] S. KOZIEL et X.-S. YANG, *Computational optimization, methods and algorithms*. Springer, 2011, t. 356 (cf. p. 22).

[101] X.-S. YANG, *Engineering optimization : an introduction with metaheuristic applications*. John Wiley & Sons, 2010 (cf. p. 22, 25, 29, 35).

[102] E. F. KELLER, « Organisms, machines, and thunderstorms : a history of self-organization, part two. Complexity, emergence, and stable attractors, » *Historical Studies in the Natural Sciences*, t. 39, n° 1, p. 1-31, 2009 (cf. p. 22).

[103] W. R. ASHBY, « Principles of the self-organizing system, » in *Facets of systems science*, Springer, 1991, p. 521-536 (cf. p. 22).

[104] K. M. HANSON, « Tutorial on Markov Chain Monte Carlo, » in *Workshop for Maximum Entropy and Bayesian Methods*, 2000, p. 9-13 (cf. p. 22).

[105] F. B. HILDEBRAND, *Introduction to numerical analysis*. Courier Corporation, 1987 (cf. p. 22).

[106] X.-S. YANG, *Introduction to computational mathematics*. World Scientific Publishing Company, 2014 (cf. p. 22).

[107] F. A. P. FIALHO et N SANTOS, « A general architecture for simulating complex systems CAPable of auto-organization, » *Artificial Neural Networks in Engineering (ANNIE94)*, 1994 (cf. p. 23).

[108] C. BLUM et A. ROLI, « Metaheuristics in combinatorial optimization : Overview and conceptual comparison, » *ACM computing surveys (CSUR)*, t. 35, n° 3, p. 268-308, 2003 (cf. p. 24).

[109] R. V. BELAVKIN, « Optimal measures and Markov transition kernels, » *Journal of Global Optimization*, t. 55, n° 2, p. 387-416, 2013 (cf. p. 25).

[110] ——, « On evolution of an information dynamic system and its generating operator, » *Optimization Letters*, t. 6, n° 5, p. 827-840, 2012 (cf. p. 25).

[111] J. M. KELLER, D. LIU et D. B. FOGEL, *Fundamentals of computational intelligence : Neural networks, fuzzy systems, and evolutionary computation*. John Wiley & Sons, 2016 (cf. p. 26).

[112] R. STORN, « On the usage of differential evolution for function optimization, » in *Proceedings of North American Fuzzy Information Processing*, IEEE, 1996, p. 519-523 (cf. p. 27).

[113] R. M. LEWIS et V. TORCZON, « Pattern search algorithms for bound constrained minimization, » *SIAM Journal on Optimization*, t. 9, n° 4, p. 1082-1099, 1999 (cf. p. 27, 28).

[114] R. DURSTENFELD, « Algorithm 235 : random permutation, » *Communications of the ACM*, t. 7, n° 7, p. 420, 1964 (cf. p. 27).

[115] K. PRICE, R. M. STORN et J. A. LAMPINEN, *Differential evolution : a practical approach to global optimization*. Springer Science & Business Media, 2006 (cf. p. 27).

[116] D. KARABOGA, « An idea based on honey bee swarm for numerical optimization, » Technical report-tr06, Erciyes university, engineering faculty, computer . . ., rapp. tech., 2005 (cf. p. 27).

[117] X.-S. YANG, S. DEB et S. FONG, « Accelerated particle swarm optimization and support vector machine for business optimization and applications, » in *international conference on networked digital technologies*, Springer, 2011, p. 53-66 (cf. p. 28, 29).

[118] X.-S. YANG, « Firefly algorithm, Levy flights and global optimization, » in *Research and development in intelligent systems XXVI*, Springer, 2010, p. 209-218 (cf. p. 28, 32, 43).

[119] I. FISTER, I. FISTER JR, X.-S. YANG et J. BREST, « A comprehensive review of firefly algorithms, » *Swarm and Evolutionary Computation*, t. 13, p. 34-46, 2013 (cf. p. 28).

[120] X.-S. YANG et S. DEB, « Cuckoo search via Lévy flights, » in *2009 World congress on nature & biologically inspired computing (NaBIC)*, IEEE, 2009, p. 210-214 (cf. p. 29, 32).

[121] I. PAVLYUKEVICH, « Lévy flights, non-local search and simulated annealing, » *Journal of Computational Physics*, t. 226, n° 2, p. 1830-1844, 2007 (cf. p. 29, 30, 35).

[122] A. H. GANDOMI, X.-S. YANG et A. H. ALAVI, « Cuckoo search algorithm : a metaheuristic approach to solve structural optimization problems, » *Engineering with computers*, t. 29, n° 1, p. 17-35, 2013 (cf. p. 29).

[123] A. H. GANDOMI, X.-S. YANG, S. TALATAHARI et S. DEB, « Coupled eagle strategy and differential evolution for unconstrained and constrained global optimization, » *Computers & Mathematics with Applications*, t. 63, n° 1, p. 191-200, 2012 (cf. p. 29).

[124] S WALTON, O HASSAN, K MORGAN et M. BROWN, « Modified cuckoo search : a new gradient free optimisation algorithm, » *Chaos, Solitons & Fractals*, t. 44, n° 9, p. 710-718, 2011 (cf. p. 29).

[125] X.-S. YANG et S. DEB, « Multiobjective cuckoo search for design optimization, » *Computers & Operations Research*, t. 40, n° 6, p. 1616-1624, 2013 (cf. p. 29).

[126] C. MIN et F. GIBOU, « Robust second-order accurate discretizations of the multi-dimensional Heaviside and Dirac delta functions, » *Journal of Computational Physics*, t. 227, n° 22, p. 9686-9695, 2008 (cf. p. 29).

[127] R. A. HORN, « The hadamard product, » in *Proc. Symp. Appl. Math*, t. 40, 1990, p. 87-169 (cf. p. 29).

[128] E. ARTIN, *The gamma function*. Courier Dover Publications, 2015 (cf. p. 29, 43).

[129] F. WANG, X. HE, Y. WANG et S. YANG, « Markov model and convergence analysis based on cuckoo search algorithm, » *Computer Engineering*, t. 38, n° 11, p. 180-185, 2012 (cf. p. 29).

[130] X.-S. YANG, « A new metaheuristic bat-inspired algorithm, » in *Nature inspired cooperative strategies for optimization (NICSO 2010)*, Springer, 2010, p. 65-74 (cf. p. 29, 32, 34).

[131] Z. W. GEEM, J. H. KIM et G. V. LOGANATHAN, « A new heuristic optimization algorithm : harmony search, » *simulation*, t. 76, n° 2, p. 60-68, 2001 (cf. p. 30, 33).

[132] X.-S. YANG, « Harmony search as a metaheuristic algorithm, » in *Music-inspired harmony search algorithm*, Springer, 2009, p. 1-14 (cf. p. 30).

[133] X.-S. YANG, M. KARAMANOGLU et X. HE, « Multi-objective flower algorithm for optimization, » *Procedia Computer Science*, t. 18, p. 861-868, 2013 (cf. p. 30).

[134] X.-S. YANG, « Flower pollination algorithm for global optimization, » in *International conference on unconventional computing and natural computation*, Springer, 2012, p. 240-249 (cf. p. 30).

[135] I. FISTER JR, X.-S. YANG, I. FISTER, J. BREST et D. FISTER, « A brief review of nature-inspired algorithms for optimization, » *arXiv preprint arXiv :1307.4186*, 2013 (cf. p. 31).

[136] J. H. HOLLAND, « Genetic algorithms, » *Scientific american*, t. 267, n° 1, p. 66-73, 1992 (cf. p. 32).

[137] J. KENNEDY et al., « Encyclopedia of machine learning, » *Particle swarm optimization*, p. 760-766, 2010 (cf. p. 32).

[138] H. W. BERGMANN, *Optimization : Methods and Applications, Possibilities and Limitations : Proceedings of an International Seminar Organized by Deutsche Forschungsanstalt Für Luft-und Raumfahrt (DLR), Bonn, June 1989*. Springer Science & Business Media, 2012, t. 47 (cf. p. 32).

[139] D. KARABOGA et B. BASTURK, « A powerful and efficient algorithm for numerical function optimization : artificial bee colony (ABC) algorithm, » *Journal of global optimization*, t. 39, n° 3, p. 459-471, 2007 (cf. p. 32).

[140] S. MIRJALILI, S. M. MIRJALILI et A. LEWIS, « Grey wolf optimizer, » *Advances in engineering software*, t. 69, p. 46-61, 2014 (cf. p. 32).

[141] D. SIMON, « Biogeography-based optimization, » *IEEE transactions on evolutionary computation*, t. 12, n° 6, p. 702-713, 2008 (cf. p. 32).

[142] S. MIRJALILI et A. LEWIS, « The whale optimization algorithm, » *Advances in engineering software*, t. 95, p. 51-67, 2016 (cf. p. 32).

[143] N. HANSEN, « The CMA evolution strategy : a comparing review, » in *Towards a new evolutionary computation*, Springer, 2006, p. 75-102 (cf. p. 32).

[144] S. MIRJALILI, « Dragonfly algorithm : a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems, » *Neural Computing and Applications*, t. 27, n° 4, p. 1053-1073, 2016 (cf. p. 32).

[145] H. TALBI et A. DRAA, « A new real-coded quantum-inspired evolutionary algorithm for continuous optimization, » *Applied Soft Computing*, t. 61, p. 765-791, 2017 (cf. p. 32).

[146] A KAVEH et N FARHOUDI, « A new optimization method : Dolphin echolocation, » *Advances in Engineering Software*, t. 59, p. 53-70, 2013 (cf. p. 32).

[147] W.-T. PAN, « A new fruit fly optimization algorithm : taking the financial distress model as an example, » *Knowledge-Based Systems*, t. 26, p. 69-74, 2012 (cf. p. 32).

[148] A. H. GANDOMI et A. H. ALAVI, « Krill herd : a new bio-inspired optimization algorithm, » *Communications in nonlinear science and numerical simulation*, t. 17, n° 12, p. 4831-4845, 2012 (cf. p. 32).

[149] A. ASKARZADEH et A. REZAZADEH, « A new heuristic optimization algorithm for modeling of proton exchange membrane fuel cell : bird mating optimizer, » *International Journal of Energy Research*, t. 37, n° 10, p. 1196-1204, 2013 (cf. p. 32).

[150] R OFTADEH, M. MAHJOOB et M SHARIATPANAHI, « A novel meta-heuristic optimization algorithm inspired by group hunting of animals : Hunting search, » *Computers & Mathematics with Applications*, t. 60, n° 7, p. 2087-2098, 2010 (cf. p. 32).

[151] Y. SHIQIN, J. JIANJUN et Y. GUANGXING, « A dolphin partner optimization, » in *2009 WRI Global Congress on Intelligent Systems*, IEEE, t. 1, 2009, p. 124-128 (cf. p. 32).

[152] J. JAMES et V. O. LI, « A social spider algorithm for global optimization, » *Applied Soft Computing*, t. 30, p. 614-627, 2015 (cf. p. 32).

[153] X. LU et Y. ZHOU, « A novel global convergence algorithm : bee collecting pollen algorithm, » in *International Conference on Intelligent Computing*, Springer, 2008, p. 518-525 (cf. p. 32).

[154] H. A. ABBASS, « MBO : Marriage in honey bees optimization-A haplometrosis polygynous swarming approach, » in *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, IEEE, t. 1, 2001, p. 207-214 (cf. p. 32).

[155] A. MUCHERINO et O. SEREF, « Monkey search : a novel metaheuristic search for global optimization, » in *AIP conference proceedings*, AIP, t. 953, 2007, p. 162-173 (cf. p. 32).

[156] M. ROTH, « Termite : A swarm intelligent routing algorithm for mobile wireless ad-hoc networks, » in *Stigmergic Optimization*. Springer Berlin Heidelberg, 2005 (cf. p. 32).

[157] X. LI, « A new intelligent optimization-artificial fish swarm algorithm, » *Doctor thesis, Zhejiang University of Zhejiang, China*, 2003 (cf. p. 32).

[158] S. SAREMI, S. MIRJALILI et A. LEWIS, « Grasshopper optimisation algorithm : theory and application, » *Advances in Engineering Software*, t. 105, p. 30-47, 2017 (cf. p. 32).

[159] G. DHIMAN et V. KUMAR, « Seagull optimization algorithm : Theory and its applications for large-scale industrial engineering problems, » *Knowledge-Based Systems*, t. 165, p. 169-196, 2019 (cf. p. 32).

[160] S. MIRJALILI, A. H. GANDOMI, S. Z. MIRJALILI, S. SAREMI, H. FARIS et S. M. MIRJALILI, « Salp Swarm Algorithm : A bio-inspired optimizer for engineering design problems, » *Advances in Engineering Software*, t. 114, p. 163-191, 2017 (cf. p. 32).

[161] E. CUEVAS, F. FAUSTO et A. GONZÁLEZ, « The Selfish Herd Optimizer, » in *New Advancements in Swarm Algorithms : Operators and Applications*, Springer, 2020, p. 69-109 (cf. p. 32).

[162] S. MIRJALILI, « Moth-flame optimization algorithm : A novel nature-inspired heuristic paradigm, » *Knowledge-based systems*, t. 89, p. 228-249, 2015 (cf. p. 32).

[163] ——, « The ant lion optimizer, » *Advances in engineering software*, t. 83, p. 80-98, 2015 (cf. p. 32).

[164] S. LI, H. CHEN, M. WANG, A. A. HEIDARI et S. MIRJALILI, « Slime mould algorithm : A new method for stochastic optimization, » *Future Generation Computer Systems*, 2020 (cf. p. 32).

[165] G.-G. WANG, « Moth search algorithm : a bio-inspired metaheuristic algorithm for global optimization problems, » *Memetic Computing*, t. 10, n° 2, p. 151-164, 2018 (cf. p. 32).

[166] G.-G. WANG, S. DEB et L. d. S. COELHO, « Elephant herding optimization, » in *2015 3rd International Symposium on Computational and Business Intelligence (ISCBI)*, IEEE, 2015, p. 1-5 (cf. p. 32).

[167] G.-G. WANG, S. DEB et L. D. S. COELHO, « Earthworm optimisation algorithm : a bio-inspired metaheuristic algorithm for global optimisation problems, » *International Journal of Bio-Inspired Computation*, t. 12, n° 1, p. 1-22, 2018 (cf. p. 32).

[168] G.-G. WANG, S. DEB et Z. CUI, « Monarch butterfly optimization, » *Neural computing and applications*, t. 31, n° 7, p. 1995-2014, 2019 (cf. p. 32).

[169] Y. LABBI, D. B. ATTOUS, H. A. GABBAR, B. MAHDAD et A. ZIDAN, « A new rooted tree optimization algorithm for economic dispatch with valve-point effect, » *International Journal of Electrical Power & Energy Systems*, t. 79, p. 298-311, 2016 (cf. p. 32).

[170] S. KAUR, L. K. AWASTHI, A. SANGAL et G. DHIMAN, « Tunicate Swarm Algorithm : A new bio-inspired based metaheuristic paradigm for global optimization, » *Engineering Applications of Artificial Intelligence*, t. 90, p. 103 541, 2020 (cf. p. 32).

[171] R. V. RAO, V. J. SAVSANI et D. VAKHARIA, « Teaching–learning-based optimization : a novel method for constrained mechanical design optimization problems, » *Computer-Aided Design*, t. 43, n° 3, p. 303-315, 2011 (cf. p. 33).

[172] V. ČERNỲ, « Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm, » *Journal of optimization theory and applications*, t. 45, n° 1, p. 41-51, 1985 (cf. p. 33).

[173] B. WEBSTER, J PHILIP et A BERNHARD, *Local Search Optimization Algorithm Based on Natural Principles of Gravitation, IKE'03, Las Vegas, Nevada, USA, June 2003*, 2003 (cf. p. 33).

[174] E. RASHEDI, H. NEZAMABADI-POUR et S. SARYAZDI, « GSA : a gravitational search algorithm, » *Information sciences*, t. 179, n° 13, p. 2232-2248, 2009 (cf. p. 33).

[175] D. B. FOGEL, *Artificial intelligence through simulated evolution*. Wiley-IEEE Press, 1998 (cf. p. 33).

[176] O. K. EROL et I. EKSIN, « A new optimization method : big bang–big crunch, » *Advances in Engineering Software*, t. 37, n° 2, p. 106-111, 2006 (cf. p. 33).

[177] S HE, Q. WU et J. SAUNDERS, « A novel group search optimizer inspired by animal behavioural ecology, » in *2006 IEEE international conference on evolutionary computation*, IEEE, 2006, p. 1272-1278 (cf. p. 33).

[178] A KAVEH et S TALATAHARI, « A novel heuristic optimization method : charged system search, » *Acta Mechanica*, t. 213, n° 3-4, p. 267-289, 2010 (cf. p. 33).

[179] E. ATASHPAZ-GARGARI et C. LUCAS, « Imperialist competitive algorithm : an algorithm for optimization inspired by imperialistic competition, » in *2007 IEEE congress on evolutionary computation*, IEEE, 2007, p. 4661-4667 (cf. p. 33).

[180] R. FORMATO, *Central force optimization : a new metaheuristic with applications in applied electromagnetics. Prog Electromagn Res 77 : 425–491*, 2007 (cf. p. 33).

[181] A. H. KASHAN, « League championship algorithm : a new algorithm for numerical function optimization, » in *2009 International Conference of Soft Computing and Pattern Recognition*, IEEE, 2009, p. 43-48 (cf. p. 33).

[182] B. ALATAS, « ACROA : artificial chemical reaction optimization algorithm for global optimization, » *Expert Systems with Applications*, t. 38, n° 10, p. 13 170-13 180, 2011 (cf. p. 33).

[183] A. KAVEH et V. R. MAHDAVI, « Colliding bodies optimization : a novel meta-heuristic method, » *Computers & Structures*, t. 139, p. 18-27, 2014 (cf. p. 33).

[184] A. HATAMLOU, « Black hole : A new heuristic optimization approach for data clustering, » *Information sciences*, t. 222, p. 175-184, 2013 (cf. p. 33).

[185] A. H. GANDOMI, « Interior search algorithm (ISA) : a novel approach for global optimization, » *ISA transactions*, t. 53, n° 4, p. 1168-1183, 2014 (cf. p. 33).

[186] A KAVEH et M KHAYATAZAD, « A new meta-heuristic method : ray optimization, » *Computers & structures*, t. 112, p. 283-294, 2012 (cf. p. 33).

[187] A. SADOLLAH, A. BAHREININEJAD, H. ESKANDAR et M. HAMDI, « Mine blast algorithm : A new population based algorithm for solving constrained engineering optimization problems, » *Applied Soft Computing*, t. 13, n° 5, p. 2592-2612, 2013 (cf. p. 33).

[188] H. DU, X. WU et J. ZHUANG, « Small-world optimization algorithm for function optimization, » in *International Conference on Natural Computation*, Springer, 2006, p. 264-273 (cf. p. 33).

[189] N. MOOSAVIAN et B. K. ROODSARI, « Soccer league competition algorithm : A novel meta-heuristic algorithm for optimal design of water distribution networks, » *Swarm and Evolutionary Computation*, t. 17, p. 14-24, 2014 (cf. p. 33).

[190] H. SHAH-HOSSEINI, « Principal components analysis by the galaxy-based search algorithm : a novel metaheuristic for continuous optimisation, » *International Journal of Computational Science and Engineering*, t. 6, n° 1-2, p. 132-140, 2011 (cf. p. 33).

[191] C. DAI, W. CHEN, Y. ZHU et X. ZHANG, « Seeker optimization algorithm for optimal reactive power dispatch, » *IEEE Transactions on power systems*, t. 24, n° 3, p. 1218-1231, 2009 (cf. p. 33).

[192] F. F. MOGHADDAM, R. F. MOGHADDAM et M. CHERIET, « Curved space optimization : a random search based on general relativity theory, » *arXiv preprint arXiv :1208.2214*, 2012 (cf. p. 33).

[193] F. RAMEZANI et S. LOTFI, « Social-based algorithm (SBA), » *Applied Soft Computing*, t. 13, n° 5, p. 2837-2856, 2013 (cf. p. 33).

[194] S. MIRJALILI, « SCA : a sine cosine algorithm for solving optimization problems, » *Knowledge-based systems*, t. 96, p. 120-133, 2016 (cf. p. 33).

[195] N. GHORBANI et E. BABAEI, « Exchange market algorithm, » *Applied Soft Computing*, t. 19, p. 177-187, 2014 (cf. p. 33).

[196] S. MIRJALILI, S. M. MIRJALILI et A. HATAMLOU, « Multi-verse optimizer : a nature-inspired algorithm for global optimization, » *Neural Computing and Applications*, t. 27, n° 2, p. 495-513, 2016 (cf. p. 33).

[197] S. Q. SALIH et A. A. ALSEWARI, « A new algorithm for normal and large-scale optimization problems : Nomadic People Optimizer, » *Neural Computing and Applications*, t. 32, n° 14, p. 10 359-10 386, 2020 (cf. p. 33).

[198] M. H. MOZAFFARI, H. ABDY et S. H. ZAHIRI, « IPO : an inclined planes system optimization algorithm, » *Computing and Informatics*, t. 35, n° 1, p. 222-240, 2016 (cf. p. 33).

[199] M. EITA et M. FAHMY, « Group counseling optimization, » *Applied Soft Computing*, t. 22, p. 585-604, 2014 (cf. p. 33).

[200] Y. TAN et Y. ZHU, « Fireworks algorithm for optimization, » in *International conference in swarm intelligence*, Springer, 2010, p. 355-364 (cf. p. 33).

[201] A. MOHAMMADI et S. H. ZAHIRI, « IIR model identification using a modified inclined planes system optimization algorithm, » *Artificial Intelligence Review*, t. 48, n° 2, p. 237-259, 2017 (cf. p. 33).

[202] S. MOHAMMADI-ESFAHROOD, A. MOHAMMADI et S. H. ZAHIRI, « A Simplified and Efficient Version of Inclined Planes system Optimization Algorithm, » in *2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI)*, IEEE, 2019, p. 504-509 (cf. p. 33).

[203] A. KUMAR, R. K. MISRA, D. SINGH, S. MISHRA et S. DAS, « The spherical search algorithm for bound-constrained global optimization problems, » *Applied Soft Computing*, t. 85, p. 105 734, 2019 (cf. p. 33).

[204] F. ZITOUNI, S. HAROUS et R. MAAMRI, « The Solar System Algorithm : A Novel Metaheuristic Method for Global Optimization, » *IEEE Access*, t. 9, p. 4542-4565, 2021 (cf. p. 33).

[205] A. E. EIBEN et S. K. SMIT, « Parameter tuning for configuring and analyzing evolutionary algorithms, » *Swarm and Evolutionary Computation*, t. 1, n° 1, p. 19-31, 2011 (cf. p. 31, 33, 35).

[206] K. KHALILI-DAMGHANI et M. AMIRI, « Solving binary-state multi-objective reliability redundancy allocation series-parallel problem using efficient epsilon-constraint, multi-start partial bound enumeration algorithm, and DEA, » *Reliability Engineering & System Safety*, t. 103, p. 35-44, 2012 (cf. p. 34).

[207] V. V. VAZIRANI, *Approximation algorithms*. Springer Science & Business Media, 2013 (cf. p. 38).

[208] T. GROSSMAN et A. WOOL, « Computational experience with approximation algorithms for the set covering problem, » *European Journal of Operational Research*, t. 101, n° 1, p. 81-92, 1997 (cf. p. 38).

[209] D. S. HOCHBA, « Approximation algorithms for NP-hard problems, » *ACM Sigact News*, t. 28, n° 2, p. 40-52, 1997 (cf. p. 38).

[210] P. K. AGARWAL et C. M. PROCOPIUC, « Exact and approximation algorithms for clustering, » *Algorithmica*, t. 33, n° 2, p. 201-226, 2002 (cf. p. 38).

[211] D. B. FOGEL, *Evolutionary computation : the fossil record*. Wiley-IEEE Press, 1998 (cf. p. 38).

[212] F. GLOVER, « Tabu search : A tutorial, » *Interfaces*, t. 20, n° 4, p. 74-94, 1990 (cf. p. 38).

[213] P. MOSCATO et al., « On evolution, search, optimization, genetic algorithms and martial arts : Towards memetic algorithms, » *Caltech concurrent computation program, C3P Report*, t. 826, p. 1989, 1989 (cf. p. 38).

[214] K. HUSSAIN, M. N. M. SALLEH, S. CHENG et R. NASEEM, « Common benchmark functions for metaheuristic evaluation : A review, » *JOIV : International Journal on Informatics Visualization*, t. 1, n° 4-2, p. 218-223, 2017 (cf. p. 38).

[215] A. E. EZUGWU, M. O. OLUSANYA et P. GOVENDER, « Mathematical model formulation and hybrid metaheuristic optimization approach for near-optimal blood assignment in a blood bank system, » *Expert Systems with Applications*, t. 137, p. 74-99, 2019 (cf. p. 38).

[216] Y.-C. HO et D. L. PEPYNE, « Simple explanation of the no-free-lunch theorem and its implications, » *Journal of optimization theory and applications*, t. 115, n° 3, p. 549-570, 2002 (cf. p. 38).

[217] D. MOLINA, J. POYATOS, J. DEL SER, S. GARCÍA, A. HUSSAIN et F. HERRERA, « Comprehensive Taxonomies of Nature-and Bio-inspired Optimization : Inspiration Versus Algorithmic Behavior, Critical Analysis Recommendations, » *Cognitive Computation*, t. 12, n° 5, p. 897-939, 2020 (cf. p. 38).

[218] K. HUSSAIN, M. N. M. SALLEH, S. CHENG et Y. SHI, « On the exploration and exploitation in popular swarm-based metaheuristic algorithms, » *Neural Computing and Applications*, t. 31, n° 11, p. 7665-7683, 2019 (cf. p. 38).

[219] J. XU et J. ZHANG, « Exploration-exploitation tradeoffs in metaheuristics : Survey and analysis, » in *Proceedings of the 33rd Chinese Control Conference*, IEEE, 2014, p. 8633-8638 (cf. p. 38).

[220] X.-S. YANG, S. DEB et S. FONG, « Metaheuristic algorithms : optimal balance of intensification and diversification, » *Applied Mathematics & Information Sciences*, t. 8, n° 3, p. 977, 2014 (cf. p. 38).

[221] M. N. M. SALLEH, K. HUSSAIN, S. CHENG, Y. SHI, A. MUHAMMAD, G. ULLAH et R. NASEEM, « Exploration and exploitation measurement in swarm-based metaheuristic algorithms : An empirical analysis, » in *International conference on soft computing and data mining*, Springer, 2018, p. 24-32 (cf. p. 38).

[222] B. MORALES-CASTAÑEDA, D. ZALDIVAR, E. CUEVAS, F. FAUSTO et A. RODRÍGUEZ, « A better balance in metaheuristic algorithms : Does it exist ? » *Swarm and Evolutionary Computation*, p. 100 671, 2020 (cf. p. 38).

[223] J. DRÉO, A. PÉTROWSKI, P. SIARRY et E. TAILLARD, *Metaheuristics for hard optimization : methods and case studies*. Springer Science & Business Media, 2006 (cf. p. 38).

[224] C YUE, K PRICE, P SUGANTHAN, J LIANG, M ALI, B QU, N AWAD et P BISWAS, « Problem definitions and evaluation criteria for the CEC 2020 special session and competition on single objective bound constrained numerical optimization, » *Comput. Intell. Lab., Zhengzhou Univ., Zhengzhou, China, Tech. Rep*, t. 201911, 2019 (cf. p. 39, 43, 45).

[225] K. LÜLING, « The archer fish, » *Scientific American*, t. 209, n° 1, p. 100-109, 1963 (cf. p. 40).

[226] S. ROSSEL, J. CORLIJA et S. SCHUSTER, « Predicting three-dimensional target motion : how archer fish determine where to catch their dislodged prey, » *Journal of experimental biology*, t. 205, n° 21, p. 3321-3326, 2002 (cf. p. 40).

[227] S. SCHUSTER, S. WÖHL, M. GRIEBSCH et I. KLOSTERMEIER, « Animal cognition : how archer fish learn to down rapidly moving targets, » *Current Biology*, t. 16, n° 4, p. 378-383, 2006 (cf. p. 40).

[228] S. SCHUSTER, S. ROSSEL, A. SCHMIDTMANN, I. JÄGER et J. PORALLA, « Archer fish learn to compensate for complex optical distortions to determine the absolute size of their aerial prey, » *Current Biology*, t. 14, n° 17, p. 1565-1568, 2004 (cf. p. 40).

[229] A. M. SHIH, L. MENDELSON et A. H. TECHET, « Archer fish jumping prey capture : kinematics and hydrodynamics, » *Journal of Experimental Biology*, t. 220, n° 8, p. 1411-1422, 2017 (cf. p. 40).

[230] A. VAILATI, L. ZINNATO et R. CERBINO, « How archer fish achieve a powerful impact : hydrodynamic instability of a pulsed jet in Toxotes jaculatrix, » *PLoS One*, t. 7, n° 10, e47867, 2012 (cf. p. 40).

[231] L. M. DILL, « Refraction and the spitting behavior of the archerfish (Toxotes chatareus), » *Behavioral Ecology and Sociobiology*, t. 2, n° 2, p. 169-184, 1977 (cf. p. 40).

[232] A. D. WHEELON, « Free flight of a ballistic missile, » *ARS journal*, t. 29, n° 12, p. 915-926, 1959 (cf. p. 41).

[233] H. CRAMÉR, *Random variables and probability distributions*. Cambridge University Press, 2004, t. 36 (cf. p. 42, 43).

[234] D. W. ZIMMERMAN et B. D. ZUMBO, « Relative power of the Wilcoxon test, the Friedman test, and repeated-measures ANOVA on ranks, » *The Journal of Experimental Education*, t. 62, n° 1, p. 75-86, 1993 (cf. p. 45).

[235] R. H. RIFFENBURGH, « Tables of Probability Distributions, » in *Statistics in Medicine (Second Edition)*, Academic Press, 2006 (cf. p. 45).

[236] N. ETEMADI, « An elementary proof of the strong law of large numbers, » *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, t. 55, n° 1, p. 119-122, 1981 (cf. p. 45).

[237] R. SALGOTRA, U. SINGH, S. SAHA et A. H. GANDOMI, « Improving Cuckoo Search : Incorporating Changes for CEC 2017 and CEC 2020 Benchmark Problems, » in *2020 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2020, p. 1-7 (cf. p. 48).

[238] A. BOLUFÉ-RÖHLER et S. CHEN, « A Multi-Population Exploration-only Exploitation-only Hybrid on CEC-2020 Single Objective Bound Constrained Problems, » in *2020 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2020, p. 1-8 (cf. p. 48).

[239] V. STANOVOV, S. AKHMEDOVA et E. SEMENKIN, « Ranked Archive Differential Evolution with Selective Pressure for CEC 2020 Numerical Optimization, » in *2020 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2020, p. 1-7 (cf. p. 48).

[240] K. M. SALLAM, S. M. ELSAYED, R. K. CHAKRABORTTY et M. J. RYAN, « Improved multi-operator differential evolution algorithm for solving unconstrained problems, » in *2020 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2020, p. 1-8 (cf. p. 49).

[241] A. VIKTORIN, R. SENKERIK, M. PLUHACEK, T. KADAVY et A. ZAMUDA, « DISH-XX Solving CEC2020 Single Objective Bound Constrained Numerical optimization Benchmark, » in *2020 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2020, p. 1-8 (cf. p. 49).

[242] A. W. MOHAMED, A. A. HADI, A. K. MOHAMED et N. H. AWAD, « Evaluating the Performance of Adaptive GainingSharing Knowledge Based Algorithm on CEC 2020 Benchmark Problems, » in *2020 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2020, p. 1-8 (cf. p. 49).

[243] J. BREST, M. S. MAUČEC et B. BOŠKOVIĆ, « Differential Evolution Algorithm for Single Objective Bound-Constrained Optimization : Algorithm j2020, » in *2020 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2020, p. 1-8 (cf. p. 49).

[244] P. BUJOK, P. KOLENOVSKY et V. JANISCH, « Eigenvector Crossover in jDE100 Algorithm, » in *2020 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2020, p. 1-6 (cf. p. 49).

[245] P. P. BISWAS et P. N. SUGANTHAN, « Large Initial Population and Neighborhood Search incorporated in LSHADE to solve CEC2020 Benchmark Problems, » in *2020 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2020, p. 1-7 (cf. p. 49).

[246] Y.-C. JOU, S.-Y. WANG, J.-F. YEH et T.-C. CHIANG, « Multi-population Modified L-SHADE for Single Objective Bound Constrained optimization, » in *2020 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2020, p. 1-8 (cf. p. 49).

[247] T. KADAVY, M. PLUHACEK, A. VIKTORIN et R. SENKERIK, « SOMA-CL for competition on single objective bound constrained numerical optimization benchmark : a competition entry on single objective bound constrained numerical optimization at the genetic and evolutionary computation conference (GECCO) 2020, » in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 2020, p. 9-10 (cf. p. 49).

[248] A. W. MOHAMED, A. A. HADI et A. K. MOHAMED, « Gaining-sharing knowledge based algorithm for solving optimization problems : A novel nature-inspired algorithm, » *International Journal of Machine Learning and Cybernetics*, p. 1-29, 2019 (cf. p. 49).

[249] D. REY et M. NEUHÄUSER, « Wilcoxon-signed-rank test, » in *International encyclopedia of statistical science*, Springer, Berlin, Heidelberg, 2011, p. 1658-1659 (cf. p. 49).