République Algérienne Démocratique Et Populaire
Ministère De L'enseignement Supérieur et de la Recherche Scientifique
Université BLIDA-1
Faculté De Technologie
Département Des Energies Renouvelables

Mémoire présenté pour l'obtention du diplôme du Master

Option : Conversion Photovoltaïque

Thème

Etude de l'influence des paramètres d'optimisation sur les algorithmes méta-heuristiques

Par **: BOULILA Mebarka**

AIT YAHIA Nour

Soutenu devant le jury composé par :

Madame	B. AMROUCHE	M.A.A	USDB1	Présidente
Monsieur	R. BOUKENOUI	M.A.A	USDB1	Examinateur
Monsieur	O.AIT SAHED	M.C.B	USDB1	Encadreur

Juillet 2023

ملخص

الهدف الرئيسي من هذه الأطروحة هو دراسة تأثير معلمات التحسين على الmetaheuristics (GA و ABC و PSO و DE) مع إجراء دراسة مقارنة مفصلة من خلال عدة إجراء عدة اختبارات على خمسة عشر دالة عددية معيارية تحت معلمات مختلفة. إهتممنا بقياس جودة التحسين ،سرعة التقارب ووقت التنفيذ مع زيادة ضغط الحساب عن طريق تغيير حجم السكان وعدد التكرارات القصوى وتعقيد مشكلة التحسين. أشارت النتائج التي تم الحصول عليها بوضوح إلى أن خوارزمية DE أكثر كفاءة بشكل عام من الخوارزميات الأربعة فيما يتعلق بجودة تحسينها ، و GA مقابل سرعة تقاربها ، و ABC مقابل سرعتها على الرغم من تعقيد الحساب.

الكلمات المفتاحية: التحسين غير الخطي، خوارزميات ميتاهوديولوجي(GA, ABC, PSO, DE) ، التقليل ، سرعة التقارب، وقت التنفيذ ، عشوائي.

Résumé :

L'objectif principale de ce mémoire est d'étudier l'effet des paramètres d'optimisation sur les algorithmes méta-heuristiques (GA, ABC, PSO et DE) toute en effectuant une étude comparative détaillé par plusieurs tests sur quinze fonctions numériques Benchmark sous différents paramètres. On s'est intéressé à mesurer la qualité de minimisation, la vitesse de convergence, et le temps d'exécution tout en augmentant la charge du calcul et cela en variant la taille de la population, le nombre d'itérations maximales et la complexité du problème d'optimisation. Les résultats obtenus ont clairement indiqué que l'algorithme DE est généralement plus efficace des quatre algorithmes vis à vis sa qualité d'optimisation, le GA vis-à-vis sa vitesse de convergence, et l'ABC vis-à-vis sa rapidité malgré la complexité du calcul.

Mots clés : optimisation non-linéaire, algorithmes méta-heuristiques (GA, ABC, PSO, DE), minimisation, vitesse de convergence, temps d'exécution, aléatoire.

Abstract:

The main objective of this thesis is to study the effect of optimization parameters on meta-heuristic algorithms (GA, ABC, PSO and DE) with a detailed comparative study through several tests on fifteen Benchmark numerical functions under different parameters. We were interested in measuring minimization quality, convergence speed and execution time while increasing the computational load by varying the population size, the number of maximum iterations and the complexity of the optimization problem. The results clearly show that DE is generally the most efficient of the four algorithms in terms of optimization quality, GA in terms of convergence speed, and ABC in terms of speed despite computational complexity.

Keywords: nonlinear optimization, meta-heuristic algorithms (GA, ABC, PSO, DE), minimization, convergence speed, execution time, random.

Remerciements :

Avant toute personne, nous remercions le bon Dieu de nous avoir prêté vie, santé et volonté pour achever ce modeste travail.

Nous tenons à remercier notre chef de département Monsieur M. BOUZAKI et notre chef d'option Monsieur T. DOUMAZ pour leurs aides.

Nous tenons à exprimer notre gratitude à notre encadreur Monsieur O. AIT SAHED docteur en génie électrique à l'Université de Blida1 d'avoir accepté de diriger notre travail ainsi que pour ses nombreux conseils et son soutien tout au long de cette thèse.

Encore et avec une grande fierté et honneur que nous tenons à présenter nos remerciements à tout la famille administrative du département des énergies renouvelables, à tous ces enseignants qui nous ont permis d'acquérir des connaissances.

Nos plus sincères remerciements vont au président de jury et les membres du jury Madame B. AMROUCHE et Monsieur R. BOUKENOUI pour l'honneur d'avoir accepté de juger ce travail.

Nous tenons enfin à exprimer notre reconnaissance et gratitude à toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce modeste travail ainsi qu'à toute personne qui fera l'effort de lire ce document.

Dédicaces :

Je dédie ce modeste travail aux deux êtres qui me sont les plus chères au monde : mon père ***'Abdel Waheb'*** *et ma mère* ***'Saïda'****, que dieu me les gardes (ان شاء الله).*

A mes frères et sœurs que j'aime beaucoup, et plus spécialement ma grande sœur ***'Radia'*** *et son mari* ***'Nadji'*** *qui m'ont beaucoup aidé.*

A toute ma famille.

A mes amis(es)' ***Mazigh', 'Nour', 'Nouha', 'Somia', 'Achouak', 'Fatima' et ' Nour Elhouda',*** *je me suis vraiment sentie en famille grâce à leurs conseils et leur soutien moral.*

A tous mes amis du département des Energies renouvelables.

Et à tous ceux que je connais de près ou de loin.

Touta

Dédicaces :

Je tiens tout d'abord à dédier ce travail comme expression de mon éternelle gratitude, aux personnes à qui je dois le plus en ce monde mes très chers parents,

A ma sœur et mes frères, qui m'ont soutenu et encouragé,

A ma précieuse famille,

A tous mes proches,

Ait yahia Nour

Abréviations :

a	**L'accélération.**
A*	**L'algorithme (A-star)**
ABC	**Artificial Bee Colony (la colonie d'abeilles artificielles).**
c	**Les coefficients de la fonction objective**
CR	**Taux de croisement**
CSOP	**Constraint Satisfaction and Optimization Problem**
D	**Le nombre de dimension.**
d	**L'ensemble des domaines**
DE	**Differential evolution,(Algorithme d'évolution différentielle).**
E	**L'ensemble des contraintes**
F	**Facteur d'échelle**
f	**Fonction objective**
GA	**Algorithmes Génétiques (Genetic Algorithm).**
Gbest	**La meilleure position globale**
IDA*	**Iterative Deepening A*,(approfondissement itératif)**
IR	**L'ensemble Réel.**
Iter	**Le nombre des itérations.**
LP	**Linear programming.**
m	**L'indice d'itération.**
NLP	**Non-Linear Programming.**
Np	**La taille de la population.**
NP	**Non déterministe polynomiales**
Nkeep	**membres seront conservés pour l'accouplement**
Pbest	**La meilleure position personnelle**
P_g	**La position globale.**
P_h	**La Probabilité.**

P_i	**La position personnelle**
PSO	**Particle swarm optimization, (l'optimisation par essaims particulaires).**
s	**La Solution**
S	**L'ensemble des solutions**
SA	**Simulated Annealing, (recuit simulé).**
SN	**Nombre d'abeilles employées.**
TS	**Tabu Seach, (recherche tabou).**
u_{ij}	**Le vecteur mutant**
V	**La vitesse.**
v_h	**La vitesse initiale.**
v_i	**La vitesse actuelle.**
v_{ij}	**La combinaison linéaire de trois solutions choisies au hasard**
W	**L'ensemble des solutions potentielles.**
x	**La position.**
X	**L'espace naturelle des variables.**
X_h	**La position initiale**
x_i	**La position actuelle.**
Z	**La valeur à optimiser**

Sommaire :

Liste des figures :

Liste des tableaux :

Introduction générale :

Les applications de l'optimisation sont innombrables, chaque processus peut être optimisé. Il n'existe aucune entreprise qui ne soit pas impliquée dans la résolution de problèmes d'optimisation, en effet, de nombreuses applications difficiles dans les domaines de la science et de l'industrie peuvent être formulées comme des problèmes d'optimisation. L'optimisation consiste à minimiser le temps, les coûts et les risques ou la maximisation du profit, de la qualité et de l'efficacité. Un grand nombre de problèmes d'optimisation réels dans les domaines de de la science, de l'ingénierie, de l'économie, des affaires et même les énergies renouvelables comme l'énergie solaire (optimiser la production photovoltaïque, le rendement de hétérojonctions, le dimensionnement des systèmes solaires…) sont complexes et difficiles à résoudre, Ils ne peuvent pas être résolus de manière exacte dans un délai raisonnable. L'utilisation d'algorithmes approximatifs est la principale alternative pour résoudre cette classe de problèmes. Les algorithmes approximatifs peuvent être divisés en deux catégories : les heuristiques et les méta-heuristiques. Les heuristiques sont conçues et applicables à un problème particulier. Ce mémoire traite des méta-heuristiques qui représentent des algorithmes approximatifs plus généraux. Elles peuvent être adaptées pour résoudre n'importe quel problème d'optimisation. Ces algorithmes s'inspirent souvent de phénomènes naturels, de processus évolutionnaires ou de comportements sociaux pour explorer efficacement l'espace des solutions et trouver des résultats de haute qualité. Cependant, le succès des algorithmes méta-heuristiques dépend fortement de la configuration de leurs paramètres d'optimisation. Ces paramètres, tels que la taille de la population, les taux de mutation et de croisement, nombre de dimensions, etc, ont une influence significative sur les performances des algorithmes méta-heuristiques. Par conséquent, une étude approfondie de l'effet de ces paramètres sur les performances des algorithmes méta-heuristiques est essentielle pour améliorer leur efficacité et leur applicabilité dans des contextes réels.

L'objectif de ce mémoire de fin d'études est donc d'analyser en détail l'effet des paramètres d'optimisation sur les performances des algorithmes méta-heuristiques. Nous explorerons différentes méta-heuristiques couramment utilisées, telles que les algorithmes génétiques (GA), les essaims particulaires (PSO), les colonies d'abeilles artificielle (ABC), et les algorithmes de l'évolution différentielle (DE). Nous étudierons comment la variation de ces paramètres influence la convergence, la diversité des solutions obtenues et le temps d'exécution des algorithmes méta-heuristiques.

Ce mémoire se structure en trois chapitres. Le premier chapitre fournira une revue approfondie sur l'optimisation, ses principes fondamentaux, ses types, et les différents types de résolution de ces derniers. Le deuxième chapitre abordera les méthodes de résolution des problèmes d'optimisation par méta-heuristiques. Le troisième chapitre mettra en évidence une analyse détaillée des différents paramètres d'optimisation et de leur influence sur les performances de quatre algorithmes méta-heuristiques (GA, ABC, PSO et DE) avec une évaluation comparative de leur efficacité.

Chapitre I :
Généralités sur les méthodes d'optimisation

1.1. Introduction :

L'optimisation est un domaine fondamental qui joue un rôle central dans de nombreux domaines. Ce chapitre constitue une introduction essentielle à l'étude approfondie des méthodes d'optimisation, nous allons explorer les concepts clés, les principes fondamentaux et les différentes approches utilisées pour résoudre les problèmes d'optimisation.
Nous commencerons par examiner les notions fondamentales de l'optimisation, notamment la définition du problème d'optimisation, les différentes catégories de problèmes (linéaires, non linéaires, mono-objectif, multi-objectif, avec ou sans contraintes,…), et les critères d'évaluation de la performance des solutions pour ensuite, plonger dans les méthodes d'optimisation, en mettant l'accent sur les méthodes exactes telles que la programmation linéaire, la programmation dynamique et les algorithmes de branchement. Nous étudierons également les méthodes approchées, y compris les algorithmes d'approximation, les heuristiques et les méta-heuristiques.

1.2. Définition de l'optimisation :

L'optimisation est un processus qui implique la recherche d'un ensemble de paramètres optimaux pour atteindre un objectif spécifique, en fonction de certains critères ou objectifs prédéfinis (monocritères ou multi-objectifs). L'objectif de l'optimisation est de maximiser les résultats positifs ou de minimiser les résultats négatifs, en respectant généralement des contraintes spécifiques (impératives ou indicatives).

Dans le cadre de l'optimisation, on cherche à trouver la combinaison optimale de variables, de paramètres ou de décisions qui permettra d'atteindre l'objectif souhaité de la manière la plus efficace et la plus satisfaisante possible, on définit une fonction objective, ou fonction de coût (voire plusieurs), que l'on cherche à minimiser ou à maximiser par rapport à tous les paramètres concernés, ces derniers peuvent représenter des quantités réelles, des valeurs binaires, des catégories ou des choix parmi différentes options.

L'optimisation peut être appliquée dans de nombreux domaines, tels que l'énergie solaire, l'ingénierie, l'économie, la finance, la logistique, la planification, la conception de systèmes, la gestion des ressources, la recherche opérationnelle, l'apprentissage automatique et bien d'autres encore. Elle est utilisée pour résoudre des problèmes complexes qui nécessitent une prise de décision efficace et une utilisation optimale des ressources disponibles.

Pour trouver la solution optimale, diverses méthodes d'optimisation sont utilisées, allant des approches mathématiques et analytiques aux méthodes heuristiques et évolutionnaires. L'optimisation est souvent un processus itératif qui implique la recherche, l'évaluation et l'ajustement de différentes solutions candidates jusqu'à ce qu'une solution satisfaisante soit trouvée.

- On modélise un problème réel en fonction mathématique(objectif) à maximiser ou minimiser

$$\boldsymbol{f(x^*) = y^* = min\{f(x)\}} \quad \textbf{(1.1)}$$

avec $x \in S$ et S : ensemble des contraintes [1].

1.3. Traitement d'un problème d'optimisation : [2]

En tant que scientifiques, ingénieurs et gestionnaires, nous devons toujours prendre des décisions. La prise de décision est omniprésente. Le monde devenant de plus en plus complexe et compétitif, la prise de décision doit être abordée de manière rationnelle et optimale. La prise de décision comprend les étapes suivantes (Fig. 1.1) :

1.3.1. Formuler le problème : dans cette première étape, un problème de décision est identifié. Ensuite, une première formulation du problème est faite. Cette formulation peut être imprécise. Les facteurs internes et externes et le(s) objectif(s) du problème sont décrits. De nombreux décideurs peuvent être impliqués dans la formulation du problème.

1.3.2. Modéliser le problème : dans cette étape importante, un modèle mathématique abstrait est construit pour le problème. Le modélisateur peut s'inspirer de modèles similaires dans la littérature. Cela permet de réduire le problème à des modèles d'optimisation bien étudiés. En général, les modèles que nous résolvons sont des simplifications de la réalité. Ils impliquent des approximations et parfois ils ignorent des processus complexes à représenter dans un modèle mathématique. Une question intéressante peut se poser : pourquoi résoudre exactement des problèmes d'optimisation de la vie réelle qui sont flous par nature ?

1.3.3. Optimiser le problème : une fois le problème est modélisé, la procédure de résolution génère une "bonne" solution pour le problème. La solution peut être optimale ou sous-optimale. Remarquons que nous trouvons une solution pour un modèle abstrait du problème et non pour le problème réel formulé à l'origine. Par conséquent, les performances des solutions obtenues sont indicatives lorsque le modèle est précis. Le concepteur de l'algorithme peut réutiliser des algorithmes de pointe sur des problèmes similaires ou intégrer les connaissances de cette application spécifique dans l'algorithme.

1.3.4. Mettre en application une solution : La solution obtenue est testée en pratique par le décideur et est mise en œuvre si elle est "acceptable". Certaines connaissances pratiques peuvent être introduites dans la solution à mettre en œuvre. Si la solution est inacceptable le modèle et/ou l'algorithme d'optimisation doivent être améliorés et le processus de prise de décision est répété.

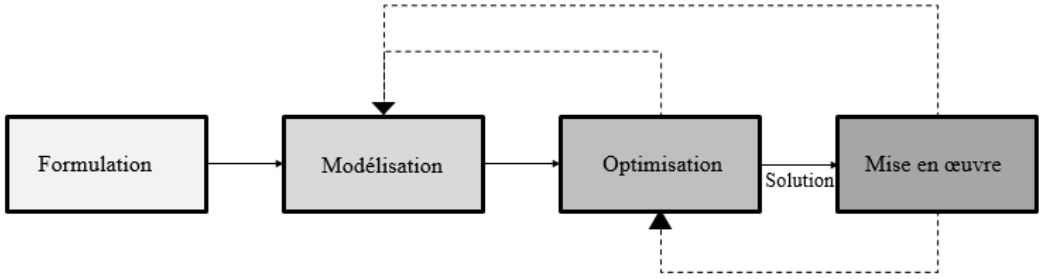

Figure (1.1) : Le processus classique de prise de décision

Dans la pratique, ce processus peut être itéré pour améliorer le modèle ou l'algorithme d'optimisation jusqu'à ce qu'une solution acceptable soit trouvée. Comme les cycles de vie dans l'ingénierie logicielle, le cycle de vie des modèles et algorithmes d'optimisation peut être linéaire, en spirale ou en cascade [2].

1.4. Quelques définitions de base :

1.4.1. Extremum d'une fonction définie sur un intervalle I :

Soit f I: $\rightarrow$ IR une fonction définie sur un intervalle I et soit a $\in$ I.

On dit que (f) admet un maximum en (a), Si, pour tout x $\in$ I, $f(x) \leq f(a)$.

On dit que (f) admet un minimum en (a), Si, pour tout x $\in$ I, $f(x) \geq f(a)$.

On parle parfois de maximum ou de minimum global de la fonction, et on dit que $f(a)$ est le maximum (ou le minimum) de (f) sur I. On dit aussi que (m) est un extremum de (f) si c'est un maximum ou un minimum.

On dit que (f) admet un maximum local (ou relatif) en (a) s'il existe un intervalle ouvert j contenant a tel que pour tout (x) appartenant au voisinage de (a) : $\forall$ x $\in$ J $\cap$ I, on a $f(x) \leq f(a)$. On définit de même un minimum local en inversant le sens de l'inégalité. Un extremum local est un maximum ou un minimum local (Figure 1.2).

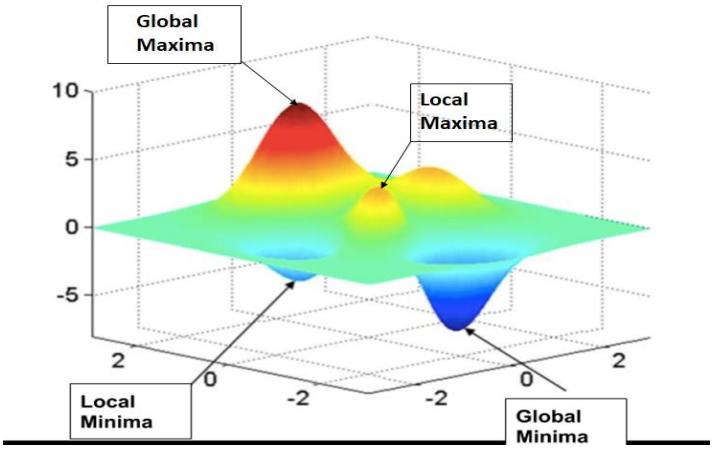

Figure (1.2) : Exemple d'un local et global minimum [3]

1.4.2. Algorithme en temps polynomial :

Un algorithme en temps polynomial est un algorithme dont le temps d'exécution est limité par une fonction polynomiale en fonction de la taille de l'entrée du problème. La complexité temporelle de l'algorithme croît de manière raisonnable lorsque la taille du problème augmente. Cela signifie que l'algorithme peut résoudre efficacement des problèmes de grande taille. En d'autres termes, Un algorithme est dit en temps polynomial s'il existe un polynôme Q(n) en n majorant T(n) pour toute donnée numérique [4].

$$\mathbf{T(n) \leq Q(n)} \qquad \mathbf{(1.2)}$$

1.4.3. Algorithme pseudo-polynomial en temps :

Un algorithme pseudo-polynomial en temps est un algorithme dont le temps d'exécution est limité par une fonction polynomiale en fonction de la taille de l'entrée et des valeurs numériques des données(P), plutôt que seulement de la taille de l'entrée. Cela signifie que l'algorithme peut sembler être en temps polynomial en fonction de la taille de l'entrée, mais en réalité, il dépend également des valeurs numériques spécifiques des données.

$$\mathbf{T(n) \leq Q\ (n, P)} \qquad \mathbf{(1.3)}$$

1.4.4. Problèmes NP-Hard « non-déterministe polynomial hard » :

Les problèmes NP-Hard sont des problèmes de décision pour lesquels il n'existe pas d'algorithme connu qui puisse les résoudre en temps polynomial déterministe. Ils sont considérés comme parmi les problèmes les plus difficiles en informatique théorique.

1.5. Les différents types de problèmes d'optimisation :

Il existe plusieurs types de problèmes d'optimisation, qui varient en fonction de leurs contraintes, de leurs objectifs et des variables impliquées, On distingue :

1.5.1. Problème d'optimisation linéaire :

Ce type de problème implique une fonction linéaire à maximiser ou à minimiser, sous des contraintes linéaires. Les contraintes et les variables sont généralement représentées par des équations linéaires.

Un problème d'optimisation linéaire est un type de problème d'optimisation où la fonction objective est une fonction linéaire que l'on cherche à minimiser ou à maximiser, tandis que les contraintes sont des inégalités ou des égalités linéaires (c'est-à-dire, du premier degré) qui doivent être satisfaites.

La forme générale d'un problème d'optimisation linéaire est la suivante : [5]

Min(Z) tel que :

$$Z = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \tag{1.4}$$

Sous les contraintes :

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq b_2 \\ \cdots \\ a_{31}x_1 + a_{32}x_2 + \cdots + a_{3n}x_n \leq b_3 \end{cases}$$

$$x_1, x_2, \ldots, x_n \geq 0$$

Où Z est la valeur à optimiser, c_1, c_2, ..., c_n sont les coefficients de la fonction objective, x_1, x_2, ..., x_n sont les variables de décision, a_{ij} sont les coefficients des contraintes, et b_1, b_2, ..., b_m sont les valeurs des contraintes [5].

I.5.2. Problème d'optimisation non linéaire :

Un problème d'optimisation non linéaire est un type de problème d'optimisation dans lequel la fonction objective ou les contraintes (ou les deux) sont non linéaires. Cela signifie que la fonction objective et/ou les contraintes peuvent contenir des termes non linéaires tels que des produits, des puissances, des fonctions trigonométriques, des fonctions exponentielles, etc.

La forme générale d'un problème d'optimisation non linéaire est la suivante : [6]

Min(Z) tel que :

$$Z = f(x_1, x_2, \dots, x_n) \tag{1.5}$$

Sous les contraintes :

$$\begin{cases} g_1(x_1, x_2, \dots, x_n) \leq 0 \\ g_2(x_1, x_2, \dots, x_n) \leq 0 \\ \dots \\ g_m(x_1, x_2, \dots, x_n) \leq 0 \end{cases}$$

$$\begin{cases} h_1(x_1, x_2, \dots, x_n) = 0 \\ h_2(x_1, x_2, \dots, x_n) = 0 \\ \dots \\ h_k(x_1, x_2, \dots, x_n) = 0 \end{cases}$$

Où Z est la valeur à optimiser, $f(x_1, x_2, ..., x_n)$ est la fonction objective non linéaire, $g_i(x_1, x_2, ..., x_n)$ sont les contraintes inégalités non linéaires, et $h_j(x_1, x_2, ..., x_n)$ sont les contraintes égalités non linéaires. Les $x_1, x_2, ..., x_n$ sont les variables de décision.

Les problèmes d'optimisation non linéaire sont généralement plus complexes à résoudre que les problèmes d'optimisation linéaire, car ils peuvent présenter des propriétés mathématiques plus compliquées et peuvent avoir plusieurs optima locaux. Des techniques d'optimisation spécifiques, telles que les méthodes de gradient, les méthodes d'optimisation sans dérivées, les méthodes de recherche locale, ou encore les algorithmes génétiques, sont utilisées pour résoudre ces problèmes [6].

1.5.3. Problème d'optimisation convexe :

Un problème d'optimisation convexe est un type de problème d'optimisation dans lequel la fonction objective et les contraintes sont convexes. La convexité est une propriété mathématique qui garantit certaines propriétés de régularité et de comportement favorable de la fonction. Dans un problème d'optimisation convexe, la fonction objective est une fonction convexe que l'on cherche à minimiser ou à maximiser, tandis que les contraintes sont des inégalités convexes ou des égalités linéaires.

Les problèmes d'optimisation convexe sont particulièrement intéressants car ils possèdent plusieurs propriétés souhaitables. Par exemple, pour tout problème d'optimisation convexe, tout minimum local est également un minimum global. De plus, les algorithmes d'optimisation pour les problèmes convexes sont généralement plus rapides et plus fiables que pour les problèmes non convexes.

I.5.4. Problème d'optimisation mono-objectif :

Un problème d'optimisation mono-objectif est un type de problème d'optimisation où l'objectif consiste à trouver la meilleure solution parmi un ensemble de possibilités en maximisant ou minimisant une seule fonction objective. Dans ce type de problème, il n'y a qu'un seul critère ou une seule mesure de performance à optimiser.

Son but est de trouver les valeurs des variables de décision qui conduisent à la meilleure valeur possible de la fonction objective tout en respectant les contraintes spécifiées. L'optimisation du problème mono-objectif peut garantir l'optimalité de la solution trouvée, mais n'en trouve qu'une seule. Dans les situations réelles, les décideurs ont généralement besoin de plusieurs alternatives [7].

1.5.5. Problème d'optimisation multi objectif :

Également appelé problème d'optimisation multicritère, est un type de problème d'optimisation où il y a plusieurs fonctions objectives à optimiser simultanément. Contrairement au problème d'optimisation mono-objectif qui cherche à optimiser une seule mesure de performance, le problème d'optimisation multi-objectif vise à trouver un ensemble de solutions qui représente le compromis entre les différents objectifs. [8]

La formulation générale d'un problème d'optimisation multi-objectif est la suivante : [2]

Min (Z) tel que :

$$\begin{cases} Z_1 = f_1(x_1, x_2, \dots, x_n) \\ Z_2 = f_2(x_1, x_2, \dots, x_n) \\ \dots \\ Z_k = f_k(x_1, x_2, \dots, x_n) \end{cases}$$

Sous les contraintes :

$$\begin{cases} g_1(x_1, x_2, \dots, x_n) \leq 0 \\ g_2(x_1, x_2, \dots, x_n) \leq 0 \\ \dots \\ g_m(x_1, x_2, \dots, x_n) \leq 0 \end{cases} \qquad (1.6)$$

$$\begin{cases} h_1(x_1, x_2, \dots, x_n) = 0 \\ h_2(x_1, x_2, \dots, x_n) = 0 \\ \dots \\ h_l(x_1, x_2, \dots, x_n) = 0 \end{cases}$$

Où Z_1, Z_2, ..., Z_k sont les fonctions objectives à minimiser ou à maximiser, f_1, f_2, ..., f_k sont les fonctions correspondantes, g_i (x_1, x_2, ..., x_n) sont les contraintes inégalités, et h_j (x_1, x_2, ..., x_n) sont les contraintes égalités. Les x_1, x_2, ..., x_n sont les variables de décision.

Le problème d'optimisation multi-objectif vise à trouver un ensemble de solutions appelé frontière PARETO [2]. Une solution est dite Pareto-optimale si on ne peut pas l'améliorer dans

un objectif sans détériorer au moins un autre objectif. La résolution de ces problèmes est complexe car il n'existe pas une unique solution optimale, mais plutôt un ensemble de solutions alternatives. Des méthodes spécifiques sont utilisées pour explorer et représenter la frontière PARETO. [2]

1.5.6. Problème d'optimisation stochastique :

Ces problèmes impliquent des variables aléatoires et des fonctions objectives probabilistes. Les techniques d'optimisation stochastique prennent en compte l'incertitude et la variabilité dans les données, les variables ou le processus lui-même. Contrairement aux problèmes d'optimisation déterministes, où les variables et les paramètres sont fixes.

La formulation générale d'un problème d'optimisation stochastique est la suivante, (minimiser ou maximiser) : [2]

Min (Z) tel que :

$$Z = f(x, \omega)$$

Sous les contraintes : (1.7)

$$\begin{cases} g(x, \omega) \leq 0 \\ h(x, \omega) = 0 \end{cases}$$

Où Z est la valeur à optimiser, f(x, ω) est la fonction objective qui dépend des variables de décision x et de la source d'incertitude ω, g(x, ω) sont les contraintes inégalités et h(x, ω) sont les contraintes égalités.

Dans un problème d'optimisation stochastique, la résolution consiste souvent à trouver une solution qui est robuste face à l'incertitude, c'est-à-dire une solution qui est performante quelle que soit la réalisation spécifique de l'incertitude. Cela peut impliquer l'utilisation de méthodes d'optimisation robuste, de techniques d'optimisation stochastique telles que la programmation stochastique, l'optimisation sous contraintes incertaines ou la simulation.

1.5.7. Problème d'optimisation combinatoire :

Ce sont des problèmes d'optimisation complexes dans lesquels la recherche de la meilleure solution consiste à trouver la meilleure combinaison parmi un ensemble fini de possibilités. Ces problèmes se caractérisent par une explosion combinatoire, ce qui signifie que le nombre total de solutions possibles augmente rapidement avec la taille du problème. Ces problèmes sont généralement difficiles à résoudre de manière exacte, car il n'est souvent pas possible d'explorer toutes les solutions de manière exhaustive en raison du temps de calcul nécessaire. Par conséquent, des algorithmes heuristiques et des méthodes d'approximation sont couramment utilisés pour trouver des solutions de bonne qualité dans un délai raisonnable [2].

1.5.8. Problème d'optimisation dynamique :

C'est un type de problème d'optimisation où les décisions à prendre évoluent dans le temps de manière séquentielle. Contrairement aux problèmes d'optimisation statique, où toutes les décisions sont prises simultanément, ils prennent en compte les interdépendances temporelles et permettent de prendre des décisions adaptatives à chaque étape en fonction de l'état du système. L'objectif est de trouver une séquence de décisions qui maximise ou minimise une fonction objective sur une période de temps donnée. Les décisions prises à chaque étape peuvent influencer l'état du système et les décisions futures.

La formulation générale d'un problème d'optimisation dynamique est la suivante : [2]

Min (Z) tel que :

$$Z = f(x_1, x_{2,} \dots, x_n, t) \tag{1.8}$$

Sous les contraintes dynamiques :

$$\begin{cases} x_1(t+1) = g_1(x_1(t), x_2(t), \dots, x_n(t)) \\ x_2(t+1) = g_2(x_1(t), x_2(t), \dots, x_n(t)) \\ \dots \\ x_n(t+1) = g_n(x_1(t), x_2(t), \dots, x_n(t)) \end{cases}$$

Où Z est la valeur à optimiser, $f(x_1, x_2, ..., x_n, t)$ est la fonction objective dépendante des variables de décision $x_1, x_2, ..., x_n$ et du temps t. Les équations $g_1, g_2, ..., g_n$ représentent les contraintes dynamiques qui décrivent l'évolution des variables de décision au fil du temps.

1.5.9. Problème d'optimisation globale :

Ces problèmes cherchent à trouver la meilleure solution parmi toutes les possibilités, sans se limiter à un sous-ensemble spécifique. Ils peuvent être particulièrement difficiles à résoudre en raison de la complexité de l'espace de recherche (vaste et comporter de nombreux optima locaux) [9].

1.5.10. Problème d'optimisation Différentiables :

C'est un type de problème d'optimisation où la fonction objective et les contraintes sont toutes différentiables. Cela signifie que les dérivées partielles de ces fonctions par rapport aux variables de décision peuvent être calculées. La différentiabilité des fonctions permet d'utiliser ces dérivées pour fournir des informations sur les directions à suivre pour améliorer la solution. En mathématiques :

Soit $f : IR^n \rightarrow IR$ une fonction continue. Si, pour tout $d \in IR^n$, la dérivée directionnelle de f dans la direction d existe, alors la fonction f est dite différentiable.

Les méthodes d'optimisation différentiables les plus couramment utilisées incluent la méthode du gradient, la méthode de Newton, la méthode de quasi-Newton et la méthode des multiplicateurs de Lagrange. Ces méthodes utilisent les dérivées des fonctions pour itérer et trouver progressivement la solution optimale.

1.5.11. Problème d'optimisation non différentiables :

C'est un type de problème d'optimisation où les dérivées partielles des fonctions par rapport aux variables ne peuvent pas être calculées ou ne sont pas définies pour certaines valeurs des variables donc les méthodes d'optimisation basées sur le calcul des dérivées ne peuvent pas être directement appliquées. Elle nécessite l'utilisation de méthodes spécifiques qui tentent d'explorer l'espace des solutions de manière plus globale, sans se fier uniquement aux informations fournies par les dérivées. Elles peuvent nécessiter davantage d'itérations et de calculs que les méthodes d'optimisation différentiables, mais elles offrent la possibilité de trouver des solutions satisfaisantes pour les problèmes non différentiables.

1.5.12. Problème d'optimisation Avec contraintes :

Constraint Satisfaction and Optimization Problem ou CSOP est un type de problème d'optimisation où les solutions possibles sont soumises à des contraintes ou des conditions à respecter. Il s'agit donc de trouver la meilleure solution possible qui satisfait toutes les contraintes du problème. Ces contraintes peuvent être des limites sur les variables de décision (impératives ou structurelles), des relations entre les variables, des équations ou des inégalités à satisfaire (indicatives ou molles).

Il existe deux types principaux de problèmes d'optimisation avec contraintes : les problèmes de programmation linéaire (LP) et les problèmes de programmation non linéaire (NLP). Dans le cas des problèmes de programmation linéaire, la fonction objective et les contraintes sont toutes linéaires, tandis que dans les problèmes de programmation non linéaire, au moins une des fonctions est non linéaire.

Un CSOP est défini par le quadruplet **(X D C *f*)** avec : [7]

- $X=\{x_1, x_2, \cdots, x_n\}$ C'est l'ensemble des variables du problème
- $D=\{D(x_1), \cdots, D(x_n)\}$ est l'ensemble des domaines
- $C=\{C_1, \cdots, C_k\}$ représente l'ensemble des contraintes

f est une fonction objective définie sur un sous-ensemble de X (à minimiser ou à maximiser).

1.5.13. Problème d'optimisation sans contraintes :

C'est un type de problème d'optimisation où il n'y a pas de contraintes à respecter. L'absence de contraintes signifie que toutes les valeurs des variables sont considérées comme faisables et acceptables. Ainsi, la recherche de la solution optimale se concentre uniquement sur la maximisation ou la minimisation de la fonction objective. Les problèmes d'optimisation sans contraintes peuvent être formulés sous forme de problèmes de programmation mathématique, ces problèmes peuvent être linéaires, non linéaires, convexes ou non convexes en fonction de la nature de la fonction cout.

- Ces catégories ne couvrent pas tous les types de problèmes d'optimisation, mais elles représentent certaines des classifications les plus courantes. Il est important de choisir les bonnes techniques et approches en fonction du type de problème d'optimisation confronté. Cela implique qu'un outil d'optimisation peut être développé pour résoudre un/plusieurs type(s) de problème ou être indépendant du problème [10].

1.6. Les méthodes de résolution de problèmes d'optimisation difficiles :

. Les méthodes de résolution liées aux problèmes d'optimisation peuvent être regroupées en deux grandes familles (figure1.3) :

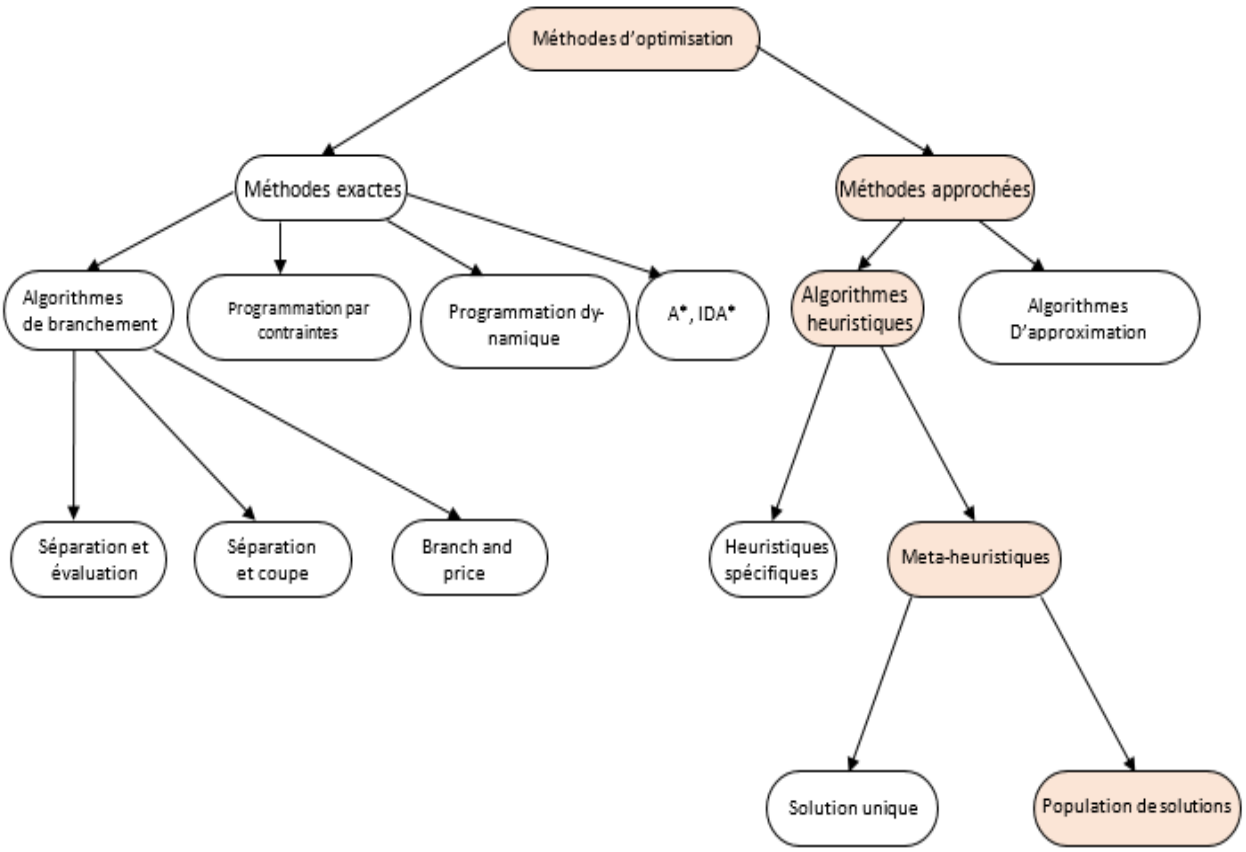

Figure (1.3) : Méthodes classiques d'optimisation [2]

1.6.1. Méthodes exactes :

Dites complètes, elles évaluent l'espace de recherche dans sa totalité en réalisant une énumération intelligente. Ils donnent la garantie de trouver la solution optimale pour une instance de taille finie dans un temps limité et permettent de prouver son optimalité. [11]

Dans la classe des méthodes exactes, on trouve les algorithmes classiques suivants : programmation dynamique, la famille d'algorithmes branch and X (branch and bound, branch and cut, branch and price), la programmation par contraintes et l'algorithme A*, IDA*.

1.6.1.1. Les algorithmes A*, IDA* : [12]

Ces algorithmes font partie de la famille d'algorithmes de recherche développés dans la communauté de l'intelligence artificielle.

L'algorithme A* (A-star) est une méthode de recherche informée qui explore un graphe à partir d'un nœud initial vers un nœud objectif en utilisant une estimation heuristique pour guider la recherche. Il maintient une file de priorité (habituellement implémentée comme un tas) pour

sélectionner les nœuds à explorer en fonction de la valeur de la fonction d'évaluation $f(n)=g(n)+h(n)$, où g(n) est le coût du chemin depuis le nœud initial jusqu'au nœud n et h(n) est une estimation heuristique du coût du chemin restant jusqu'au nœud objectif. L'algorithme choisit le nœud avec la plus petite valeur $f(n)$ pour l'exploration.

L'algorithme IDA* (Iterative Deepening A*ou approfondissement itératif) est une variante de l'algorithme A* qui utilise un processus d'exploration itérative en augmentant progressivement la profondeur maximale de recherche. Il commence avec une profondeur de recherche de 0 et exécute l'algorithme A* jusqu'à ce qu'il atteigne cette profondeur maximale. Si le nœud objectif n'a pas été trouvé, l'algorithme relance l'exploration avec une profondeur maximale plus grande. Cette procédure se répète jusqu'à ce que le nœud objectif soit trouvé.

L'avantage de l'algorithme IDA* est qu'il utilise moins de mémoire que l'algorithme A* en effectuant une recherche à profondeur limitée à chaque itération. Cependant, cela peut entraîner une exploration plus longue si la profondeur optimale du chemin est élevée [13].

1.6.1.2. La programmation dynamique :

Elle est basée sur la division récursive d'un problème en sous-problèmes plus simples. Cette procédure est basée sur le principe de BELLMAN qui dit que « la sous-politique d'une politique optimale est elle-même optimale » [14], elle stocke les résultats intermédiaires dans une table (appelée tableau de mémoïsation) pour les réutiliser ultérieurement.

Le processus de résolution d'un problème en utilisant la programmation dynamique se déroule généralement en suivant ces étapes :

- Identification de la structure des sous-problèmes
- Définition de la relation de récurrence
- Création du tableau de mémoïsation
- Résolution des sous-problèmes
- Reconstruction de la solution optimale

1.6.1.3. La programmation par contraintes :

C'est un langage construit autour des concepts de recherche arborescente et d'implications logiques. Les problèmes d'optimisation dans la programmation par contraintes sont modélisés au moyen d'un ensemble de variables liées par un ensemble de contraintes. Les variables prennent leurs valeurs sur un domaine fini d'entiers. Les contraintes peuvent avoir des formes mathématiques ou symboliques. Une description plus détaillée des techniques de programmation par contraintes se trouve à [2], section 5.3.1.

1.6.1.4. Les algorithms de branchement ou branch and X (branch and bound, branch and cut, branch and price): [15]

Ce sont des méthodes utilisées pour résoudre des problèmes d'optimisation combinatoire Développés dans la communauté de la recherche opérationnelle, ce sont des algorithmes de contraintes et de prix basés sur une énumération implicite de toutes les solutions du problème d'optimisation considéré. L'espace de recherche est exploré en construisant dynamiquement un arbre dont le nœud racine représente le problème à résoudre et l'ensemble de l'espace de recherche associé.

Le processus des algorithmes de branchement peut être décrit comme suit :

- **Initialisation :** L'algorithme commence avec une solution partielle initiale, généralement vide ou une solution triviale. Il définit également une valeur optimale courante qui est initialisée à une valeur infinie ou moins l'infini, selon le type de problème (minimisation ou maximisation).

- **Branchement :** À chaque étape, l'algorithme sélectionne une variable de décision non affectée ou une partie du problème à diviser en différentes branches. Chaque branche représente une hypothèse ou une décision différente qui peut être prise.

- **Évaluation :** Pour chaque branche, l'algorithme évalue la faisabilité de la solution partielle associée à cette branche et estime sa valeur. Si la solution partielle est déjà moins favorable que la valeur optimale courante, la branche est élaguée (pruning) et n'est pas explorée plus en détail. Sinon, l'algorithme poursuit l'exploration de cette branche.

- **Terminaison :** L'algorithme continue le processus de branchement et d'évaluation jusqu'à ce que toutes les branches aient été explorées ou jusqu'à ce qu'un critère de terminaison soit satisfait. Ce critère peut être basé sur le nombre total de branches explorées, le temps écoulé, ou d'autres considérations spécifiques au problème.

- **Mise à jour de la solution optimale :** Si une solution complète et faisable est trouvée et qu'elle est meilleure que la valeur optimale courante, la valeur optimale est mise à jour avec cette nouvelle solution.

L'avantage des algorithmes de branchement est qu'ils permettent d'explorer de manière exhaustive l'espace des solutions, garantissant ainsi de trouver la meilleure solution possible. Cependant, ces algorithmes peuvent être coûteux en termes de temps et de mémoire, surtout lorsque l'espace des solutions est de grande taille. Des techniques d'élagage intelligentes sont souvent utilisées pour réduire le nombre de branches à explorer, telles que les bornes inférieures (lower bounds) et les bornes supérieures (upper bounds).

- Les méthodes exactes peuvent être appliquées à de petites instances de problèmes difficiles, pour quelques problèmes d'optimisation NP-hard courants, l'ordre de grandeur de la taille maximale des instances que les méthodes exactes de pointe peuvent résoudre de manière optimale mais la taille de l'instance n'est pas le seul indicateur qui décrit la difficulté d'un problème, mais aussi sa structure [2].

1.6.2. Méthodes approchées :

Les méthodes incomplètes, reposent en grande partie sur l'utilisation d'heuristiques [16], [17]. Les méthodes approchées, également appelées méthodes heuristiques ou méta heuristiques, sont des techniques utilisées pour résoudre des problèmes d'optimisation lorsque des méthodes exactes ne sont pas réalisables ou trop coûteuses en termes de temps de calcul. Les méthodes approchées sont conçues pour trouver des solutions de qualité acceptable, sans garantie d'optimalité. Elles sont souvent basées sur des stratégies itératives qui explorent l'espace des solutions de manière intelligente pour se rapprocher d'une solution optimale ou de très bonne qualité. Ces méthodes sont adaptées à des problèmes difficiles et de grande taille. Dans la classe des algorithmes approchés, on peut distinguer trois sous-classes :

1.6.2.1. Les algorithmes d'approximation : [18]

Ces algorithmes sont conçus pour résoudre des problèmes d'optimisation difficiles pour lesquels il n'existe pas d'algorithme exact efficace connu. Au lieu de rechercher une solution optimale, les algorithmes d'approximation se concentrent sur la recherche d'une solution qui est proche de l'optimum en termes de qualité [19]. Ils fournissent des solutions de qualité acceptable pour des problèmes difficiles où des solutions exactes sont difficilement réalisables, pour certains problèmes NP-difficile, il est impossible de faire des approximations. De plus, ces algorithmes sont spécifiques au problème ciblé et les approximations fournies sont souvent trop éloignées de la solution optimale, ce qui freine leur utilisation pour beaucoup d'applications réelles [2].

1.6.2.2. Les heuristiques :

Les heuristiques sont des méthodes de résolution de problèmes qui utilisent des règles empiriques ou des techniques d'exploration pour guider la recherche de solutions de manière efficace. Contrairement aux méthodes exactes qui garantissent l'optimalité, les heuristiques fournissent des solutions réalisables, mais sans garantie formelle d'optimalité [20].

Quelques points importants à connaître sur les heuristiques : [7]

- Les heuristiques sont des méthodes de résolution approximatives qui fournissent des solutions qui sont généralement bonnes, mais pas nécessairement optimales.
- Elles sont basées sur des règles, des stratégies ou des techniques heuristiques spécifiques qui guident le processus de recherche de solutions.
- Les heuristiques simplifient souvent le problème d'optimisation en se concentrant sur les aspects les plus importants et en ignorant les détails moins pertinents.

- Elles peuvent être conçues pour prendre en compte des contraintes spécifiques du problème ou pour exploiter des connaissances spécialisées.
- Les heuristiques peuvent être utilisées pour résoudre une grande variété de problèmes, tels que l'ordonnancement, le routage, l'optimisation des ressources, la planification, etc.
- Elles sont souvent utilisées lorsque les méthodes exactes prennent trop de temps ou sont trop coûteuses en termes de ressources.
- Les heuristiques peuvent être itératives, c'est-à-dire qu'elles améliorent progressivement la solution en effectuant des étapes itératives d'exploration et d'optimisation.
- Elles peuvent également utiliser des techniques de recherche locale pour trouver des solutions dans des espaces de recherche complexes.
- Les heuristiques ne fournissent généralement pas de garanties mathématiques sur la qualité de la solution obtenue, mais elles sont souvent évaluées empiriquement en comparant les résultats aux solutions optimales lorsque celles-ci sont connues.
- Les heuristiques problèmes-dépendants (spécifiques) peuvent être connectés aux méta-heuristiques pour fournir une implémentation concrète aux méthodes d'optimisation abstraites, la partie abstraite étant représentée par les méta-heuristiques.

1.6.2.3. Les méta-heuristiques :

Les méta-heuristiques sont des algorithmes génériques qui peuvent être utilisés pour résoudre une grande variété de problèmes d'optimisation et d'instances différentes sans dépendre de la structure spécifique du problème. Elles peuvent être vues comme des méthodes générales de niveau supérieur (méta) utilisées pour guider la stratégie de conception d'heuristiques sous-jacentes, ces dernières étant plus flexibles et plus adaptables au problème d'optimisation à résoudre que les méthodes traditionnelles, même si elles ne garantissent pas de trouver la meilleure solution possible, mais elles cherchent à fournir des solutions de haute qualité dans des délais raisonnables.

Les méta-heuristiques utilisent des stratégies d'exploration et d'exploitation pour parcourir l'espace des solutions, exploiter les solutions prometteuses, rechercher des solutions globalement optimales et éviter de rester bloquées dans des régions sous-optimales. Elles fonctionnent par itérations, en améliorant progressivement la solution courante.

Elles peuvent être utilisées pour résoudre des problèmes d'optimisation mono-objectif ou multi-objectif, avec ou sans contraintes, linéaires ou non linéaires, convexes ou non convexes et leur performance dépend de nombreux facteurs tels que les paramètres de l'algorithme (dimensions élevées, espaces de recherche complexes, contraintes strictes, fonctions objectives non linéaires, ou une combinaison de ces facteurs), la représentation des solutions, etc.
On retrouve au niveau des méta-heuristiques les méthodes basées sur une solution unique qui manipulent et transforment une seule solution pendant le processus de recherche tandis que les

algorithmes à base de population travaillent avec une population entière, c'est à dire un ensemble de solutions qui évolue au fur et à mesure des itérations [3], [21].

1.7. L'optimisation dans les systèmes photovoltaïques :

L'optimisation joue un rôle crucial dans l'amélioration de l'efficacité et des performances des systèmes photovoltaïques (PV). Ces systèmes visent à convertir la lumière du soleil en électricité, et les techniques d'optimisation peuvent contribuer à divers aspects de leur conception, de leur fonctionnement et de leur gestion. Voici quelques exemples d'utilisation de l'optimisation mathématique dans les systèmes photovoltaïques :

1. Conception et dimensionnement du système : L'optimisation peut être utilisée pour déterminer le dimensionnement optimal des composants photovoltaïques, tels que les panneaux solaires, les onduleurs et les batteries. Il s'agit de trouver la combinaison de ces composants qui maximise la production d'énergie et minimise les coûts sur la durée de vie du système. Cela implique souvent de prendre en compte des facteurs tels que le rayonnement solaire, les profils de charge, la capacité de stockage de l'énergie et les coûts des équipements.

2. Orientation et angle d'inclinaison : L'orientation et l'angle d'inclinaison des panneaux solaires influencent considérablement leur production d'énergie. Des techniques d'optimisation peuvent être utilisées pour trouver l'orientation et l'angle d'inclinaison optimaux des panneaux afin de maximiser la production d'énergie en fonction de la situation géographique et de la période de l'année.

3. Systèmes de suivi : Les systèmes de suivi solaire qui ajustent l'angle des panneaux solaires pour suivre la trajectoire du soleil peuvent améliorer la réception de l'énergie. L'optimisation peut être utilisée pour déterminer la meilleure stratégie de suivi afin de maximiser la production d'énergie, en tenant compte de facteurs tels que l'efficacité des panneaux, la complexité du mécanisme de suivi et les coûts de maintenance.

4. Suivi du point de puissance maximale (MPPT) : Les panneaux photovoltaïques ont un point de fonctionnement optimal appelé point de puissance maximale (MPP), qui varie en fonction des conditions environnementales. Des algorithmes d'optimisation peuvent être utilisés pour suivre dynamiquement le MPP afin de s'assurer que les panneaux fonctionnent à leur plus haut niveau d'efficacité et produisent le maximum d'énergie.

5. Gestion et stockage de l'énergie : Si un système photovoltaïque comprend des composants de stockage d'énergie tels que des batteries, l'optimisation peut aider à gérer les cycles de charge et de décharge afin de minimiser les coûts, d'améliorer l'autosuffisance énergétique et d'assurer une alimentation électrique fiable.

6. Intégration au réseau : Pour les systèmes photovoltaïques connectés au réseau, l'optimisation peut aider à décider de la quantité d'énergie à consommer, à stocker ou à réinjecter dans le réseau. Cela permet d'équilibrer l'offre et la demande d'énergie, en tenant compte des réglementations et des tarifs du réseau.

7. Planification de la maintenance : L'optimisation peut être utilisée pour programmer les activités de maintenance du système photovoltaïque, telles que le nettoyage des panneaux, la réparation ou le remplacement des composants défectueux et la maintenance préventive. L'objectif est de minimiser les temps d'arrêt et les coûts de maintenance tout en maximisant la production d'énergie.

8. Conception d'un micro-réseau : Dans les zones non raccordées au réseau ou éloignées, les systèmes photovoltaïques peuvent faire partie de micro-réseaux. L'optimisation peut aider à concevoir l'agencement du micro-réseau, en optimisant l'utilisation des sources d'énergie renouvelables, des générateurs diesel et du stockage d'énergie pour répondre à la demande d'énergie tout en minimisant les coûts et l'impact sur l'environnement.

9. Analyse prédictive : Les techniques d'optimisation peuvent intégrer l'analyse prédictive et les prévisions météorologiques afin d'anticiper les variations de l'irradiation solaire et d'ajuster les paramètres du système à l'avance, améliorant ainsi la capture d'énergie et l'intégration au réseau.

10. Analyse coûts-avantages : L'optimisation permet une évaluation complète des différentes configurations du système et des stratégies opérationnelles, permettant aux décideurs d'évaluer les compromis entre l'investissement initial, les coûts opérationnels, les économies d'énergie et les avantages environnementaux.

En résumé, les techniques d'optimisation peuvent être appliquées tout au long du cycle de vie des systèmes photovoltaïques, depuis la conception et le dimensionnement jusqu'à l'exploitation et la maintenance quotidiennes, ceci afin d'obtenir un meilleur rendement énergétique, de réaliser des économies et de mieux s'intégrer à l'infrastructure énergétique existante.

1.8. Conclusion :

Ce chapitre a abordé des notions et des définitions de bases sur l’optimisation, ses différents types et les méthodes de résolution utilisées, et d’après ce qu’on a déjà cité, Il existe deux méthodes de résolution, les méthodes exactes et les méthodes approchées ou on va s’intéresser aux méthodes approchées (incomplètes) plus précisément aux méta-heuristiques dans le chapitre suivant.

Chapitre II :
Les Méta-heuristiques

2.1 Introduction :

L'introduction des méta-heuristiques fut un tournant majeur dans le domaine de l'optimisation. Elles permettent de trouver des solutions d'assez bonne qualité à cout réduit pour des problèmes qui ne trouveront pas de solutions par les méthodes classiques. Dans ce chapitre, on présente les concepts de base des algorithmes d'optimisation méta-heuristiques ainsi que certain algorithme. Les algorithmes génétiques (GA), le differencial evolution (DE), Particle swarm optimization (PSO) et Artificial bee colony (ABC) sont principalement examinés.

2.2. Classification :

Les algorithmes méta-heuristiques peuvent être classés en fonction de plusieurs critères [22] :

2.2.1. Inspirés par la nature ou non :

Les méta-heuristiques peuvent être classés selon qu'ils ont été conçus sur la base de processus naturels ou non : L'algorithme du système immunitaire artificiel est issu de la biologie ; l'optimisation par essaims de particules, abeilles et colonies de fourmis est inspirée de l'intelligence en essaim, tandis que le recuit simulé, par exemple, est inspiré de la physique.

2.2.2. Utilisation de la mémoire ou absence de mémoire :

Dans certains algorithmes, l'évolution de la recherche ne nécessite pas d'informations déjà collectées sur l'espace de recherche. Cette approche est adoptée par l'algorithme de recuit simulé. En revanche, la recherche taboue et l'OSP, par exemple, utilisent la mémoire pour sauvegarder les informations extraites dynamiquement en vue d'une exploitation ultérieure.

2.2.3. Déterministe ou stochastique :

Les algorithmes stochastiques utilisent une part d'aléatoire pendant la recherche, contrairement aux algorithmes déterministes qui utilisent des décisions déterminées. Ainsi, dans les algorithmes déterministes, le fait de partir de la même solution initiale conduira toujours à la même solution finale. Cependant, en raison du caractère aléatoire introduit dans les algorithmes stochastiques, le fait de partir de la même solution initiale conduira généralement à des solutions différentes.

2.2.4. Itératif contre gourmand :

Dans les algorithmes itératifs, la recherche commence par une ou plusieurs solutions initiales, puis ils tentent itérativement d'améliorer la ou les solutions à l'aide de certains opérateurs de recherche, contrairement aux algorithmes gourmands dans lesquels la recherche démarre à partir d'une solution vide. La solution est construite étape par étape en assignant une seule variable de décision du problème à chaque étape. La plupart des algorithmes méta-heuristiques sont itératifs [22].

2.2.5. Recherche basée sur la population versus recherche basée sur une solution unique

Les algorithmes basés sur la population travaillent avec une population de solutions. À chaque itération, toutes les solutions actuelles sont manipulées pour générer la population de solutions suivante (PSO, GA). Dans les approches basées sur une solution unique, une seule solution

évolue au cours du processus de recherche (SA, TS). Les algorithmes basés sur la population sont plus orientés vers l'exploration car ils permettent d'explorer l'espace de recherche plus efficacement, tandis que les approches basées sur une solution unique sont plus orientées vers l'exploitation car elles permettent une recherche approfondie des régions locales.

2.3. Concepts de base :

2.3.1. Population initiale :

Initialisation de la population est une étape des plus important dans la mise en œuvre des algorithmes méta-heuristiques à base de population. Sa génération se fait souvent de manière aléatoire bien qu'on peut distinguer d'autre stratégie tel que : la diversification séquentielle, la diversification parallèle et l'initialisation heuristique [22].

Celle-ci doit couvrir efficacement l'espace de recherche pour que l'algorithme puisse localiser les points de solution appropriés, l'optimum global [23] sans converger vers un optimum local (convergence prématurée).

2.3.2. Taille de la population :

Un paramètre crucial pour les méta-heuristiques à base de population. L'augmenté résulte en de meilleures solutions mais nécessite de plus grandes ressources informatiques. Il faut donc accommoder une bonne qualité des solutions à une faible capacité de calcul. Il n'existe pas de règle définitive sur la manière de choisir la taille de la population, bien que certaines règles de conception pratiques aient été extraites en fonction des algorithmes utilisés (PSO, GA, DE …etc) et de la dimension du problème.'En général, la taille de la population est choisie en fonction de la dimension du problème [24].

2.3.3. Exploitation vs exploration :

Lors de la recherche de l'optimum, les méta-heuristiques emploient deux principes :

L'exploration (recherche globale) consistant en la diversification de l'espace de recherche. Elle permet de découvrir de nouvelles régions de l'espace de recherche évitant ainsi la stagnation dans des optima locaux. Cela dit ce comportement ne favorise pas la convergence de la méta-heuristique.

L'exploitation (recherche locale) est L'intensification de l'information accumulée durant la recherche. Elle concentre la recherche sur des zones de l'espace ou la possibilité de croiser l'optimum global est plus grande. Elle agit en améliorant les meilleures solutions ou en explorant leurs voisinages dans l'attente de repéré de meilleures solutions. L'exploitation favorise la convergence de la méta-heuristique mais peut inciter à une convergence prématurée (vers un optimum local)

Le programme doit donc trouver un équilibre entre ces deux processus pour pouvoir converger vers l'optimum global de l'espace de recherche, en évitant de rester immobilisé sur un optimum local [25].

2.3.4. Critères d'arrêt :

Plusieurs critères peuvent être utilisés pour arrêter la procédure d'optimisation, on cite en [22] :

- Procédure statique : Le processus d'optimisation est limité par un nombre fixe d'itérations, limitant nombre maximal d'évaluations de la fonction objective. Cette procédure est généralement utilisée lorsque la contrainte du temps est imposée.

- Procédure adaptative : contrairement à la méthode précédente, ici la fin du processus d'optimisation n'est pas connue au préalable. Elle peut être établi avec ou lors de l'obtention d'une solution optimale satisfaisante avec une marge d'erreur prédéfinie.

2.3.5. Le nombre d'itérations :

Un nombre élevé d'itérations permet l'obtention de meilleurs résultats mais cela peut induire à la complication du calcul et au ralentissement de la vitesse d'exécution quand ce dernier est trop grand

2.4. Traitement des contraintes :

Le traitement des contraintes dans les problèmes d'optimisation est d'une importance capitale pour la conception efficace de méta-heuristiques. Pour ce fait les stratégies suivantes ont été établi [22] :

a. Stratégies de rejet :

Seules les solutions réalisables sont utilisées pendant la recherche. Si une solution infaisable est générée, elle est automatiquement rejetée.
Cette approche n'est intéressante que lorsque la majorité de l'espace de recherche est réalisable. Elles n'utilisent pas les solutions infaisables pour recueillir des informations sur les solutions optimales globales qui peuvent se trouver soit à la limite entre les solutions réalisables et infaisables, soit dans une autre région réalisable indépendante si l'ensemble admissible contient des régions discontinues.

b. Stratégies de pénalisation :

Dans ces approches, les solutions réalisables et irréalisables peuvent être prises en compte. Cependant, la fonction de coût originale sera modifiée pour inclure un nouveau terme qui pénalisera fortement les solutions irréalisables. Il s'agit des approches les plus populaires et les plus largement utilisées pour gérer les contraintes.

c. Stratégies de réparation :

Dans ces stratégies, des algorithmes heuristiques personnalisés sont utilisés sur les solutions irréalisables générées afin de les transformer en solutions réalisables. Le succès de la stratégie dépend fortement de l'efficacité de l'algorithme construit. Le problème de ces méthodes est la variante supplémentaire de calcul nécessaire pour réparer toutes les solutions infaisables générées.

d. Stratégies de préservation :

Dans ces méthodes, l'algorithme d'optimisation sera légèrement modifié afin de garantir que chaque solution générée sera toujours faisable en incorporant des connaissances spécifiques au problème. Bien entendu, l'algorithme adapté ne peut pas être utilisé pour résoudre n'importe quel problème d'optimisation donné. En outre, des solutions initiales réalisables doivent être générées pour lancer l'algorithme, ce qui peut être problématique. L'approche hybride qui combine plusieurs stratégies peut également être utilisée pour gérer les contraintes.

e. Stratégies de décodage :

Une procédure de décodage peut être considérée comme une fonction $R \rightarrow S$ qui associe à chaque représentation $r \in R$ une solution réalisable $s \in S$ dans l'espace de recherche. Cette stratégie consiste à utiliser des codages indirects. La topologie de l'espace de recherche est alors transformée à l'aide de la fonction de décodage

2.5. Les méthodes de résolutions des problèmes d'optimisation méta-heuristiques :

On présente ici quelques algorithmes méta- heuristiques, deux à base de population : le Simulated Annealing (SA) et la Tabu Search (TS), et quatre à base de population : Genetic Algorithm (GA), Differencial Evolution (DE), l'optimisation par essaims de particules (PSO) et la colonie d'abeilles artificielle (ABC).

2.5.1. Simulated Annealing (SA) :

Instauré par de S. Kirkpatrick [26]et de V. Cerny [27] en 1980, l'AS impact grandement la recherche heuristique en raison de sa simplicité et de son efficacité dans la résolution des problèmes d'optimisation combinatoire. Elle a ensuite été étendue aux problèmes d'optimisation continue [28,29,30].

Le SA repose sur les principes de la mécanique statistique selon lesquels le processus de recuit consiste à chauffer puis à refroidir lentement une substance afin d'obtenir une structure cristalline solide. La solidité de la structure dépend de la vitesse de refroidissement des métaux. Si la température initiale n'est pas suffisamment élevée ou si un refroidissement rapide est appliqué, des imperfections (états métastables) sont obtenues. Dans ce cas, le solide refroidi n'atteindra pas l'équilibre thermique à chaque température. Un refroidissement lent et prudent permet d'obtenir des cristaux solides. L'algorithme SA simule les changements d'énergie dans un système soumis à un processus de refroidissement jusqu'à ce qu'il converge vers un état d'équilibre (état gelé stable). Ce schéma a été développé en 1953 par Metropolis [31].

La fonction objective du problème est analogue à l'état énergétique du système. Une solution du problème d'optimisation correspond à un état du système. Les variables de décision associées à une solution du problème sont analogues aux positions moléculaires. L'optimum global correspond à l'état fondamental du système. La découverte d'un minimum local implique qu'un état métastable a été atteint.

Le SA est un algorithme stochastique qui permet, sous certaines conditions, la dégradation d'une solution. L'objectif est d'échapper aux optima locaux et donc de retarder la convergence. Le SA est un algorithme sans mémoire dans le sens où l'algorithme n'utilise aucune information recueillie au cours de la recherche. À partir d'une solution initiale, il procède à plusieurs

itérations. À chaque itération, un voisin aléatoire est généré. Les déplacements qui améliorent la fonction de coût sont toujours acceptés. Dans le cas contraire, le voisin est sélectionné avec une probabilité donnée qui dépend de la température actuelle et de l'importance de la dégradation E de la fonction objective. E représente la différence de valeur objective (énergie) entre la solution actuelle et la solution voisine générée. Au fur et à mesure que l'algorithme progresse, la probabilité que de tels mouvements soient acceptés diminue. Cette probabilité suit, en général, la distribution de Boltzmann.

L'algorithme utilise un paramètre de contrôle, appelé température, pour déterminer la probabilité d'accepter des solutions qui ne s'améliorent pas. À un niveau de température donné, de nombreux essais sont explorés.

Une fois qu'un état d'équilibre est atteint, la température est progressivement diminuée selon un programme de refroidissement de manière à ce que peu de solutions non améliorantes soient acceptées à la fin de la recherche. [2]

2.5.2. Tabu Search (TS) :

La méthode Tabou a été développée par Glover En 1986 puis indépendamment par Hansen en 1990 [32]. L'utilisation de la mémoire, qui stocke les informations liées au processus de recherche, représente la caractéristique particulière de la recherche Tabou.

Le TS se comporte comme un algorithme de recherche locale la plus abrupte (plus forte descente), mais il accepte des solutions non améliorantes pour échapper aux optima locaux lorsque tous les voisins sont des solutions non améliorantes. Habituellement, l'ensemble du voisinage est exploré de manière déterministe, alors que dans le SA, un voisin aléatoire est sélectionné. Lorsqu'un meilleur voisin est trouvé, il remplace la solution actuelle. Lorsqu'un optimum local est atteint, la recherche se poursuit en sélectionnant un candidat moins bon que la solution actuelle. La meilleure solution dans le voisinage est sélectionnée comme nouvelle solution courante, même si elle n'améliore pas la solution courante.

La recherche Tabou peut être considérée comme une transformation dynamique du voisinage. Cette politique peut générer des cycles, c'est-à-dire que les solutions visitées précédemment peuvent être sélectionnées à nouveau. Pour éviter les cycles, le TS écarte les voisins qui ont été visités précédemment. Il mémorise la trajectoire de recherche récente. La recherche taboue gère une mémoire des solutions ou des mouvements récemment appliqués, appelée liste taboue. Cette liste de tabous constitue la mémoire à court terme. À chaque itération de la recherche taboue, la mémoire à court terme est mise à jour. Stocker toutes les solutions visitées prend du temps et de l'espace. En effet, nous devons vérifier à chaque itération si une solution générée n'appartient pas à la liste de toutes les solutions visitées.

La liste taboue contient généralement un nombre constant de mouvements tabous. Habituellement, les attributs des mouvements sont stockés dans la liste de tabous. En introduisant le concept de caractéristiques de solution ou de caractéristiques de mouvement dans la liste Tabou, on peut perdre certaines informations sur la mémoire de recherche. Nous pouvons rejeter des solutions qui n'ont pas encore été générées.

Pour ne pas interdire l'accès à ces configurations de qualité, supérieure, on introduit certaines conditions, appelées critères d'aspiration, permettant aux solutions taboues d'être acceptées.

Les solutions voisines admissibles sont celles qui ne sont pas taboues ou qui satisfont le critère d'aspiration [2].

2.5.3. Genetic Algorithm GA :

Les algorithmes génétiques ont été développés par J. Holland dans les années 1970 (Université du Michigan, États-Unis) pour comprendre les processus adaptatifs des systèmes naturels [33] Ils ont ensuite été appliqués à l'optimisation et à l'apprentissage automatique dans les années 1980 [34] [35].

Les GA constituent une classe très populaire des algorithmes évolutionnaires. Traditionnellement, les GA sont associés à l'utilisation d'une représentation binaire, mais on trouve aujourd'hui des GA qui utilisent d'autres types de représentations. Un GA applique généralement un opérateur de croisement à deux solutions qui joue un rôle majeur, ainsi qu'un opérateur de mutation qui modifie aléatoirement le contenu individuel afin de promouvoir la diversité. Les GA utilisent une sélection probabiliste qui est à l'origine la sélection proportionnelle. Le remplacement (sélection des survivants) est générationnel, c'est-à-dire que les parents sont remplacés systématiquement par les descendants. L'opérateur de croisement est basé sur le croisement à n points ou le croisement uniforme, tandis que la mutation est un retournement de bits. Une probabilité fixe pm est appliquée à l'opérateur de mutation [2].

Le GA suit les étapes suivantes :[36] [7]

1. Initialisation de la population :

Ceci consiste en la génération d'une population initial aléatoire éparpillé sur tout l'espace de recherche. Les particules obtenues sont des vecteurs à D dimension appelées chromosomes [7]

$$\mathbf{X_{ij} = x_{min\ j} + rand_j\,[0, 1] \cdot (x_{max\ j} - x_{min\ j})}\ ,\ i \in [1, k], j \in [1, D] \qquad (2.1)$$

On calcul leur fitness conformément à la fonction objective pour avoir une évaluation primaire de la qualité des solutions.

2. La sélection naturelle :

Les chromosomes sont triés de manière décroissante en fonction de leur fitness. De la population initiale, seuls les meilleurs membres Nkeep seront conservés pour l'accouplement. Le restes des chromosomes seront écartés au profit de la nouvelle progéniture. Ce processus permettra à la population d'évoluer au fil des générations.

3. Sélection des parents :

À ce stade, deux parents sont choisis dans la population survivante Nkeep pour produire deux descendants qui contiennent des traits de chaque parent.

Pour cela, plusieurs mécanismes d'appariement sont possibles : on peut choisir les parents au hasard, utiliser la roulette wheel en fonction de leur fitness ou les apparier de haut en bas, car les chromosomes seront ordonnés en commençant par le haut de la population (conformément à la fitness) [37].

Le parent choisi sera également ajouté à la nouvelle population. Cette procédure est répétée jusqu'à ce que la nouvelle population soit entièrement régénérée.

4. Recombinaison :

Les GA utilisent deux opérateurs de variation différents : la recombinaison et la mutation. Ils permettent aux individus de changer et de s'améliorer. La recombinaison est l'analogie du GA avec la reproduction. Deux individus ou plus sont sélectionnés par l'algorithme de sélection des parents et produisent une descendance.

La manière dont la descendance est créée dépend de la détermination de l'algorithme de recombinaison.

Il existe quelques opérations générales de recombinaison de le GA qui sont utilisées par de nombreux GA différents dans des situations très variées. Ces opérations constituent de bons points de référence pour comparer les opérateurs de recombinaison plus spécifiques à un problème.

Les opérateurs généraux sont les opérateurs de croisement en un point, les opérateurs de croisement en n points et les opérateurs de croisement uniforme [38] [39] [2]. Ces opérateurs fonctionnent en prenant la première "tranche" de la première représentation parentale, puis la deuxième "tranche" de la deuxième représentation parentale et en combinant ces tranches dans la première descendance. Le même processus est répété, mais pour la deuxième "tranche" du premier parent et la première "tranche" du deuxième parent. On obtient ainsi un mélange de gènes de chaque parent dans la descendance. Une "tranche" est une partie d'une représentation.

Soit $x_f = \{xf_1, \ldots, x_{fD}\}$ et $x_m = \{xm_1, \ldots, x_{mD}\}$ les parents, la progéniture sera tel :

$$X_1 = \{xf_1, x_{f2,} \uparrow x_{m4,} x_{m5,} x_{mD}\}$$

$$X_2 = \{x_{m1}, x_{m2,} \uparrow x_{f4,} x_{f5,} x_{fD}\}$$

Cette approche ne fait qu'interchanger des variables entre les chromosomes sans en ajoutée de nouvelle ce qui peut être peut attrayant.

Une autre méthode plus intéressante est la "méthode du blending " [2] [37] [40] dans laquelle les descendants sont construits en combinant les valeurs des variables des parents comme suit :

$$X_{ij} = \beta\, x_{fi} + (1-\beta)\, x_{mi} \qquad i \in \{1 \ldots\ldots D\} \tag{2.2}$$

$$X_{ij} = \beta\, x_{mi} + (1-\beta)\, x_{fi} \qquad , i \in \{1 \ldots\ldots D\} \tag{2.3}$$

$$\text{Avec } \beta \in [0,1] \qquad [36]$$

5. Mutation :

L'autre type d'opérateur de variation, est la mutation. Les mutations sont généralement appliquées de manière statistique avec un certain taux de mutation déterminant la probabilité de mutation d'une variable.

Par exemple, un taux de mutation de 20% indique que 1/5 des variables de tous les chromosomes seront remplacées par des valeurs générées aléatoirement.[7]

Ce changement vise à créer de la diversité au sein de la population et permet à l'algorithme d'explorer d'autres régions de l'espace de recherche et d'échapper aux optima locaux

L'algorithme se poursuit en répétant les quatre phases précédentes jusqu'à ce que le critère d'arrêt soit satisfait.

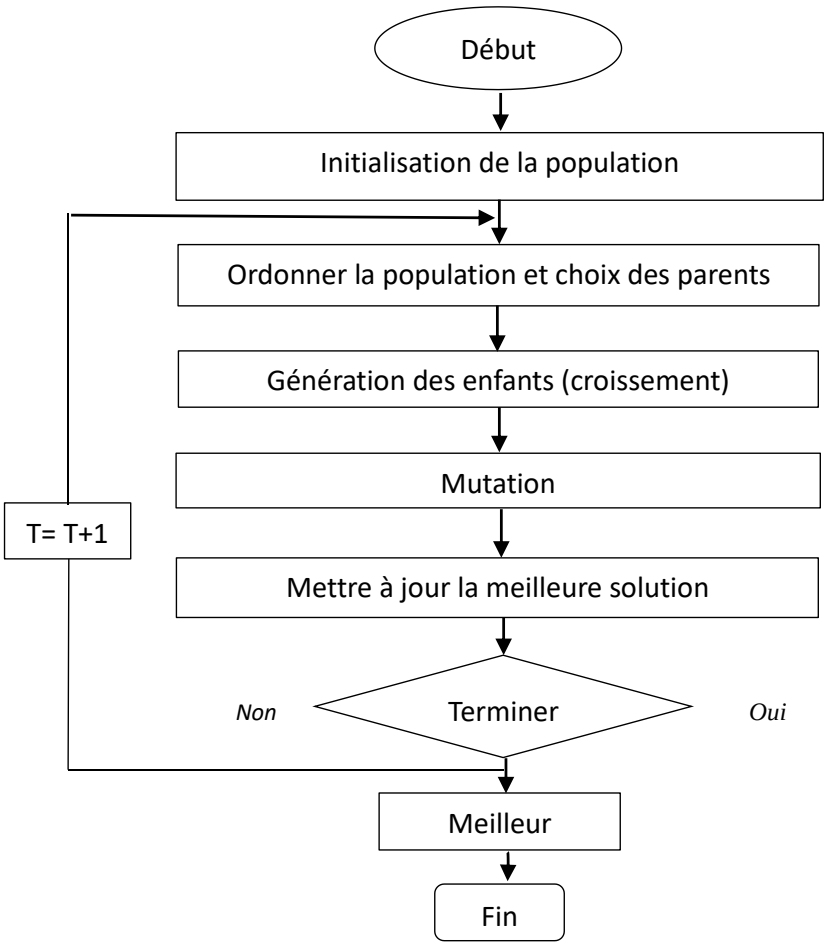

Figure (2.1) : Organigramme du GA

2.5.4. Differential Evolution (DE) :

Elaborée par K. Price en 1995, l'évolution différentielle (DE) est l'une des approches les plus réussies pour l'optimisation continue [41]. L'idée principale de DE est d'utiliser les différences vectorielles pour perturber la population de vecteurs. Cette idée a été intégrée dans un nouvel opérateur de recombinaison de deux solutions ou plus et dans un opérateur de mutation autoréférentielle pour orienter la recherche vers les bonnes solutions.

La notation DE/x/y/zest généralement utilisée pour définir une stratégie DE, où x spécifie le vecteur à muter, qui peut actuellement être rand (un vecteur de la population choisi au hasard) ou best (le vecteur de coût le plus faible de la population actuelle), y est le nombre de vecteurs de différence utilisés, et z désigne le schéma de croisement. La variante standard est le

croisement bin en raison d'expériences binomiales indépendantes. On utilisera l'algorithme DE de base : l'algorithme DE/rand/1/bin, où bin représente la variante de croisement binaire. [2]

Comme tout algorithme évolutionnaire, le DE génère une population initiale P_0 de taille k distribuée de manière aléatoire. Chaque individu est un vecteur réel x_{ij} à D dimensions. Chaque individu est codé sous la forme d'un vecteur. Chaque élément du vecteur x_{ij} est généré aléatoirement dans l'intervalle $[x_{min},x_{max}]$ chacune représentant les limites inférieure et supérieure de chaque variable. Cela suivant : [2]

$$\mathbf{x_{ij} = x_{min\ j} + rand_j\ [0, 1]\ .\ (\ x_{max\ j} - x_{min\ j}\ )} \qquad (2.1)$$

i∈ [1, k], j∈ [1, D] et $rand_j$ une variable aléatoire uniformément distribuée dans l'intervalle [0,1]

Au cours du processus d'évolution, chaque individu (i) est affiné de manière itérative. Le processus de modification comporte deux étapes :

1. **Mutation/croisement :**

Cette étape consiste à créer une solution variante en utilisant des membres de la population sélectionnée au hasard. Ensuite, une solution d'essai est créée en recombinant la solution variante avec l'individu i (étape de croisement).

L'opérateur de recombinaison DE est basé sur une combinaison linéaire dans laquelle le concept de distance joue un rôle important. On lui assigne donc un parent i et trois individus de la population r1, r2 et r3 choisis au hasard. Ou r1, r2, r3 ∈ {1, 2, …, NP} sont trois entiers différents entre eux et différents de i.

Le paramètre F représente un facteur d'échelle (F ∈ [0, 1]) tandis que le paramètre CR représente une probabilité (CR ∈ [0, 1]).

Lorsque randj [0, 1] < CR ou j = jrand, la variable associée à la descendance U_{ij} sera assigné le vecteur mutant V_{ij} consistant en la combinaison linéaire de trois solutions choisies au hasard [2].

$$\mathbf{U_{ij} = V_{ij} = x_{r3j} + F.(x_{r1j} - x_{r2j}\ )} \qquad (2.3)$$

Dans le cas contraire, la variable descendante hérite de la valeur de son parent : $\mathbf{u_{ij} = x_{ij}}$

La condition **j = jrand** est incluse pour garantir qu'au moins une variable de la descendance sera différente de son parent (par exemple, pour CR= 0). Le facteur d'échelle F contrôle l'amplification de la différence entre les individus r1 et r2 et est utilisé pour éviter la stagnation du processus de recherche [2].

2. **Remplacement :**

Cette étape exécute une procédure de sélection pour déterminer si la solution d'essai remplace l'individu i dans la population. Le remplacement est élitiste c'est-à-dire que la progéniture remplacera son parent si sa valeur objective est meilleure ou égale à celle du parent : [2]

$$x_i(t+1) = \begin{cases} ui(t+1) & si\ \ f(ui(t+1)) \le f(xi(t)) \\ xi(t) & sinon \end{cases}$$

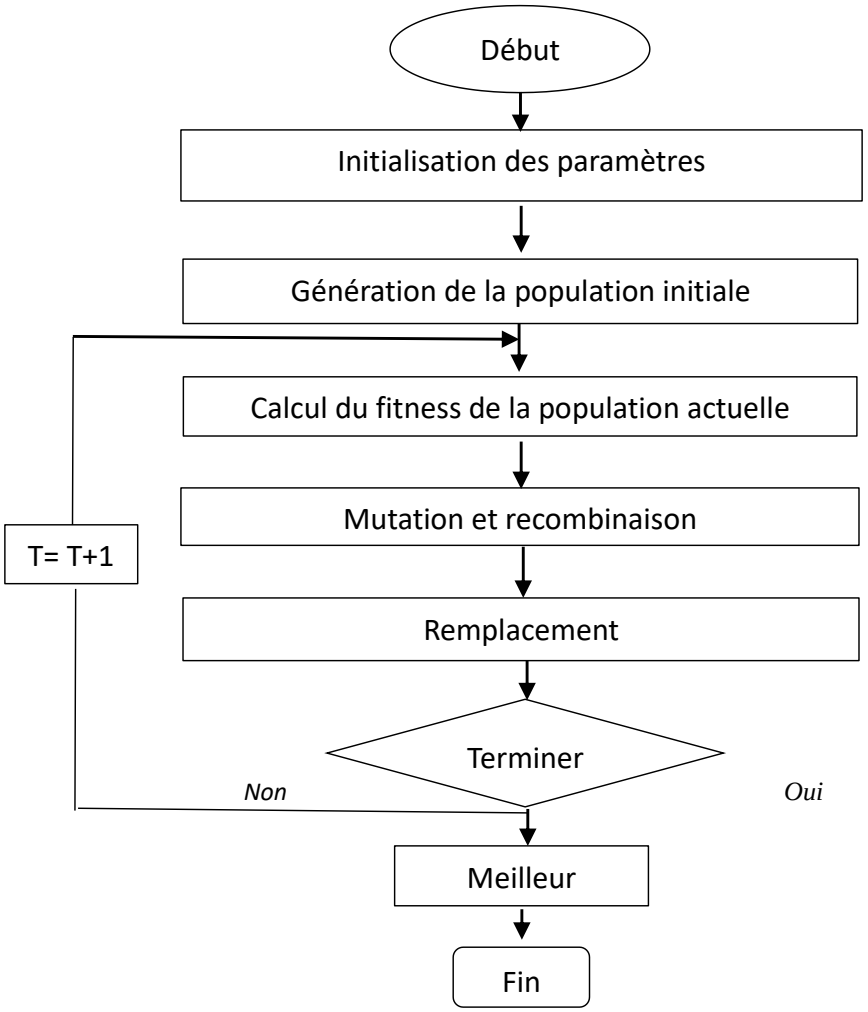

Figure (2.2) : Modèle de l'algorithme DE/rand/1/bin algorithm [2]

2.5.5. Particle Swarm Optimization (PSO):

L'optimisation par essaim de particules est une méta-heuristique stochastique basée sur la population et inspirée de l'intelligence en essaim [43]. Elle imite le comportement social d'organismes naturels tels que les volées d'oiseaux et les bancs de poissons qui cherchent à trouver un endroit où la nourriture est suffisante. En effet, dans ces essaims, un comportement coordonné utilisant des mouvements locaux émerge sans aucun contrôle central. Introduit par Kennedy et Eberhart en 1995, le PSO a été conçu au début avec succès pour les problèmes d'optimisation continue.

Dans le modèle de base, un essaim se compose de N particules volant dans un espace de recherche de dimension D. Chaque particule i est une solution candidate au problème et est représentée par le vecteur xi dans l'espace de décision. Chaque particule a sa propre position et sa propre vitesse [2].

L'optimisation tire parti de la coopération entre les particules. Le succès de certaines particules influencera le comportement de leurs homologues. Chaque particule ajuste successivement sa position xi vers l'optimum global en fonction des deux facteurs suivants : le « pesonal best position » (la meilleure position visitée par la particule elle-même) pbest, et le « global best position » (et la meilleure position visitée par l'ensemble de l'essaim) gbest [2].

Chaque particule est composée de trois vecteurs : [2]

1. Le vecteur x enregistre la position actuelle de la particule dans l'espace de recherche.
2. Le vecteur p enregistre l'emplacement de la meilleure solution trouvée jusqu'à présent par la particule.
3. Le vecteur v contenant la vitesse (direction)

Deux valeurs de fitness : Le x-fitness enregistre le fitness du vecteur x, et p-fitness recueille le fitness du vecteur P [2].

Le PSO suit les étapes suivantes : [36] [7]

1. **Initialisation :**

On commence par la génération aléatoire des positions et des vitesses initiales, l'initialisation des meilleures positions connues par chaque particule, ainsi que la détermination de la meilleure solution globale P_g. Sont suit le processus itératif.

À chaque itération, chaque particule applique les opérations suivantes :

2. **Mise à jour de la vitesse :**

La vitesse qui définit la quantité de changement qui sera appliquée à la particule est définie comme suit : [7][36]

$$\mathbf{v(m) = v(m - 1) + a(m)} \tag{2.4}$$

avec v, a, x et m représentant respectivement la vitesse, l'accélération, l'indice d'itération.

L'accélération pour i particule est donné suivant l'expression : [7][36]

$$\mathbf{a_i = \chi\,[c\,\epsilon_1(P_g - x_i) + c\,\epsilon_2(P_i - x_i)] - (1 - \chi)\,v_i} \tag{2.5}$$

Où ϵ_1 et ϵ_2 sont deux variables aléatoires dans l'intervalle **[0, 1],** et c et χ des constantes tel que **c =2.05** et χ **=0.729843788**

x_i et v_i sont respectivement la position actuelle et la vitesse de la particule *i*, Pi est la meilleure position personnelle de la particule *i*, tandis que P_g est la meilleure position globale de toutes les particules.

a_i peut aussi bien être négatives positives. Le paramètre χ garantit une vitesse décroissante pour chaque particule lorsque le nombre d'itérations augmente ce qui aide à la convergence du programme.

3. Mise à jour de la position :

Chaque particule met à jour ses coordonnées dans l'espace de décision suivant : [7][36]

$$\mathbf{x(m+1) = x(m) + v(m+1)} \qquad (2.6)$$

Elle se déplace ensuite vers sa nouvelle position.

4. Mise à jour des meilleures particules trouvées :

Chaque particule met à jour la meilleure solution locale.

Si $\boldsymbol{f(xi) < f(pbest)}$, alors $\mathbf{pi = xi}$

De plus, la meilleure solution globale de l'essaim est mise à jour :

Si $\boldsymbol{f(xi) < f(gbest)}$, alors $\mathbf{gi = xi}$

Par conséquent, à chaque itération, chaque particule modifie sa position en fonction de sa propre expérience et de celle des particules voisines.

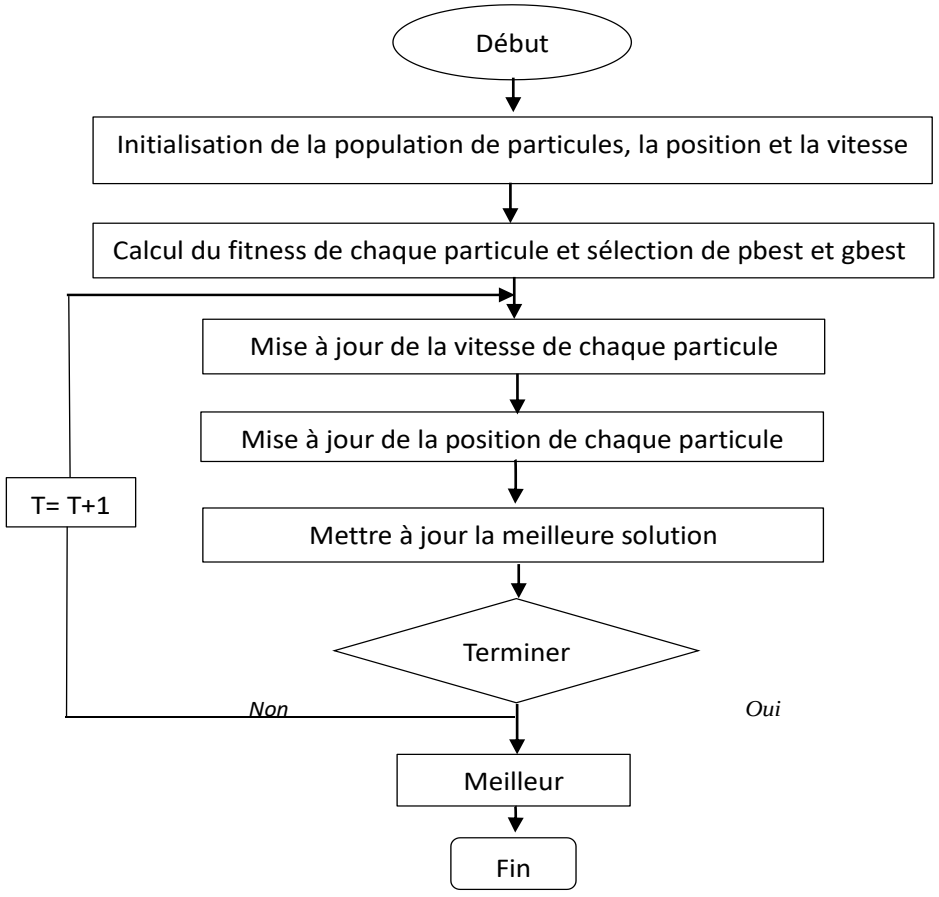

Figure (2.3) : Organigramme du PSO [43]

2.5.6 Artificial Bee Colony (ABC) :

L'ABC est un algorithme d'optimisation méta-heuristique proposé par Karaboga en 2005 [44] dans le but de résoudre le problème d'optimisation des fonctions multivariables. Il s'inspire du comportement social des essaims d'abeilles mellifères.

Il a pour composante principales :

1. **Source de nourriture :**

Elle représente une solution possible au problème d'optimisation tandis que la quantité de nectar représente son fitness correspondant.

2. **Abeilles employées :**

Représente la moitié de la population associée à des sources de nourriture qu'elles exploitent. Elles transportent ensuite l'information avec elles et la partagent avec une certaine probabilité au abeilles spectatrices.

3. **Les spectatrices (onlookers) :**

Seconde moitié de la population attendant dans la ruche et choisissant une source de nourriture par le biais des informations partagées par les abeilles employées en fonction de la qualité de cette dernière.

4. **Les éclaireuses(scouts) :**

Recherchent dans l'environnement entourant la ruche pour des nouvelles sources de nourriture pour remplacer les sources abandonnées.

2.5.6.1. Etapes de l'algorithme :

1. **Initialisation :**

Pour une population de taille Np, on a SN, le nombre d'abeilles employées, correspondant à la moitié de Np. Pour chaque abeille employée, on assigne une source de nourritures x_h de D dimension générée suivant l'expression : [7][36]

$$\mathbf{x_h = x_{min\ j} + \alpha_{h\ j}.(x_{max\ j} - x_{min\ j})} \qquad (2.7)$$

avec $j \in [1, D]$; $h \in [1, SN]$; $\alpha_{h\,j} \in [0,1]$

$x_{max\ j}$, $x_{min\ j}$ borne supérieure et inférieure de j

On évalue la qualité de chaque source en calculant le cout et le fitness de celle-ci

2. **Phase d'abeilles employées (employed bees) :**

Chaque abeille employée va chercher une nouvelle source de nourriture v_h au voisinage de l'ancienne x_h suivant l'expression suivante : [7][36]

$$\mathbf{v_{hj} = x_{hj} + \varphi_{h\,j}\,(x_{h\,j} - x_{k\,j})} \quad (2.8)$$

Ou $x_{k\,j}$ position aléatoire à D dimension contenue dans SN et différente de $x_{h\,j}$

($k \in [1,SN]$; $k \neq h$)

$\varphi_{h\,j} \in [-1,1]$ est un nombre aléatoire uniformément distribué.

Dans cette étape, rien qu'une seule dimension est mise à jour, générant ainsi notre nouvelle source dont on compare la fitness après calcul à celle de la source précédente x_h, pour savoir laquelle garder (greedy selection)

3. **Phase de sélection probabiliste :**

Lors du partage de l'information par les abeilles employées au abeilles spectatrices sur la qualité des solutions, les abeilles employées choisissent aléatoirement les sources de nourriture à exploiter suivant une probabilité ph : [7][36]

$$\mathbf{P_h} = \frac{\mathbf{1/Fitness(X_h)}}{\sum_{h=1}^{SN} \mathbf{1/Fitness(X_h)}} \quad (2.9)$$

Plus la quantité de nectar de la source de nourritures est importante plus le sera sa probabilité d'être sélectionné.

4. **Phase des abeilles spectatrices (onlookers bees) :**

Dans cette étape, chaque source de nourriture x_h subira une mise à jour conformément à l'équation précédente (2.8) produisant une nouvelle source de nourriture candidate v_h. Sont suit le greedy selection entre v_h et x_h.

5. **Phase d'abeilles éclaireuses (scout bees) :**

Si une source de nourriture x_h ne parvient pas à produire une meilleur source nourriture à son voisinage après une limite de tentatives infructueuse (limit), celle-ci est abandonnée au profit d'une nouvelle source de nourriture aléatoire généré suivant l'équation (2.7).

La limite se calcule suivant l'expression : [7][36]

$$\mathbf{Limit = SN.\ D} \quad (2.10)$$

Ce paramètre permet de conserver une certaine diversité au sein de la population ABC en générant de nouvelles sources de nourritures.

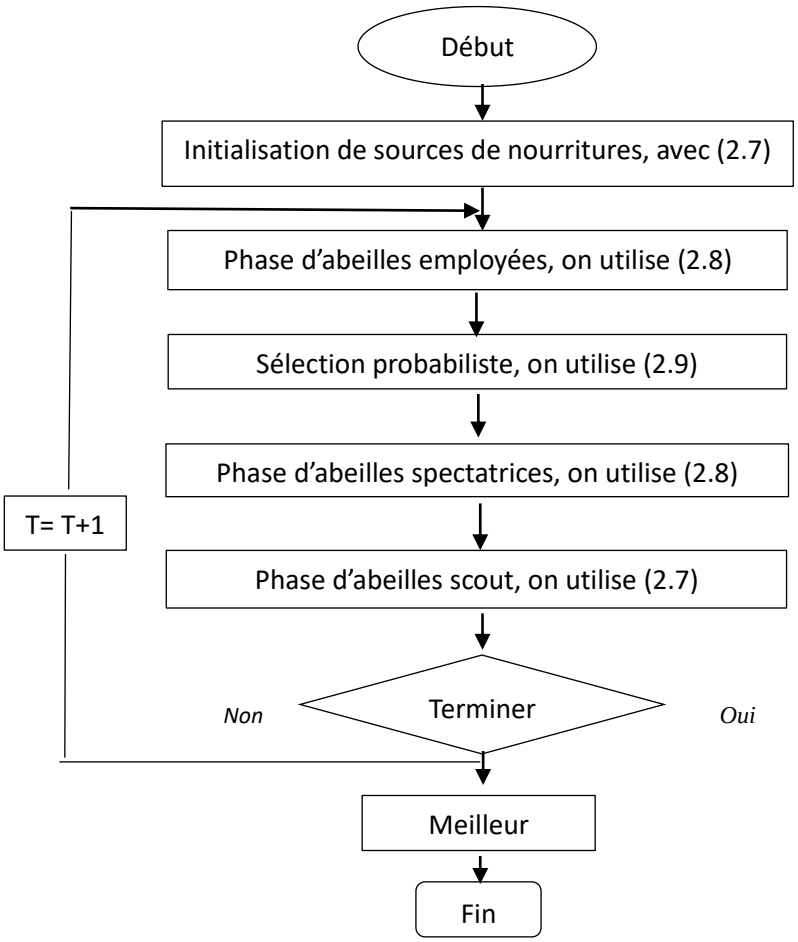

Figure (2.4) : L'organigramme d'algorithme ABC. [7]

2.7. Conclusion :

Dans ce chapitre, on a pu se familiariser avec les notions principales de l'optimisation méta-heuristiques ainsi que les détails de certains algorithmes.

On s'intéresse maintenant à quatre d'entre eux à savoir : les algorithmes génétiques (GA), l'évolution différentiel (DE), l'optimisation par essaims de particules (PSO) et la colonie d'abeilles artificielle (ABC) qui feront l'objet de notre étude, après programmation sous Matlab

Chapitre III :
Simulation et résultats

3.1. Introduction :

Dans ce chapitre, on réalise une étude sur l'impact des paramètres d'optimisation sur le bon fonctionnement des algorithmes poursuivit par une étude comparative entre les algorithmes GA, DE, PSO et ABC. Pour cela, on réalise plusieurs tests sur des fonctions benchmark dont les résultats feront l'objet d'une étude statistique.

3.2. Fonctions Benchmark :

On présente ci-dessous quinze fonctions benchmark fréquemment utilisées pour l'évaluation des algorithmes d'optimisation, qu'on utilise donc dans la cadre de notre étude. Toutes ces fonctions sont de nature différentes et intervalles différents mais possédant un minimum commun : le zéros (0).

Fonction	Espace de recherche	min
$f_1 = \sum_{i=1}^{D} x_i^2$	$[-100,100]^D$	0
$f_2 = \sum_{i=1}^{D-1} \left(100\left(x_i^2 - x_{i+1}\right)^2 + (x_i - 1)^2\right)$	$[-2.048,2.048]^D$	0
$f_3 = \sum_{i=1}^{D} \left(i x_i^4\right) + \text{random}\,0,1)$	$[-1.28,1.28]^D$	0
$f_4 = \sum_{i=1}^{D} \left(x_i^2 - 10\cos(2\pi x_i) + 10\right)$	$[-5.12,5.12]^D$	0
$f_5 = 20 + exp(1) - 20\,exp\left(-0.2\sqrt{\left(\frac{1}{D}\right)\sum_{i=1}^{D} x_i^2}\right) - exp\left(\left(\frac{1}{D}\right)\sum_{i=1}^{D} cos(2\pi x_i)\right)$	$[-32.768,32.768]^D$	0
$f_6 = \sum_{i=1}^{D} \left(\frac{x_i^2}{4000}\right) - \prod_{i=1}^{D} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600,600]^D$	0
$f_7 = \sum_{i=1}^{D} \lvert x_i\, sin(x_i) + 0.1x_i \rvert$	$[-10,10]^D$	0
$f_8 = 10^6 x_1^2 + \sum_{i=2}^{D} x_i^2$	$[-100,100]^D$	0
$f_9 = sin^2(\pi\omega_1) + \sum_{i=1}^{D-1} [(\omega_i - 1)^2(1 + 10\,sin^2(\pi\omega_i + 1))] + (\omega_D - 1)^2(1 + sin^2(2\pi\omega_D))$ $\omega_i = 1 + \left(\frac{(x_i - 1)}{4}\right)$	$[-10,10]^D$	0
$f_{10} = \left\lvert \sum_{i=1}^{D} x_i^2 - D \right\rvert^{\frac{1}{4}} + \frac{\left(0.5\sum_{i=1}^{D} x_i^2 + \sum_{i=1}^{D} x_i\right)}{D} + 0.5$	$[-100,100]^D$	0
$f_{11} = \sum_{i=1}^{D} \lvert x_i \rvert - \prod_{i=1}^{D} \lvert x_i \rvert$	$[-10,10]^D$	0
$f_{12} = \sum_{i=1}^{D} (10^6)^{\frac{i-1}{D-1}} x_i^2$	$[-100,100]^D$	0
$f_{13} = \sum_{i=1}^{D} i * x_i^2$	$[-10,10]^D$	0
$f_{14} = \sum_{i=1}^{D} x_i^2 + \left(\sum_{i=1}^{D} 0.5 \cdot i \cdot x_i\right)^2 + \left(\sum_{i=1}^{D} 0.5 \cdot i \cdot x_i\right)^4$	$[-5,10]^D$	0
$f_{15} = \sum_{i=1}^{D} \sum_{j=1}^{D} \left[\left[\frac{1}{4000}\left(100\left(x_i^2 - x_j\right)^2 + \left(1 - x_j\right)^2\right)^2\right] - cos\left(100\left(x_i^2 - x_j\right)^2 + \left(1 - x_j\right)^2\right) + 1\right]$	$[-10.24,10.24]^D$	0

Tableau (3.1) : Fonctions Benchmark

3.3. Analyse numérique :

3.3.1. Configuration expérimentale :

On réalise quatre algorithmes (GA, DE, PSO, ABC) dont le but de minimiser les fonctions benchmark vues précédemment, faisant office de fonction objective. En vue de la nature stochastique de nos algorithmes, on réalise 50 exécutions par scénario dont les résultats, subiront une étude statistique portant sur le calcul de la moyenne, la variance, la médiane, la valeur maximale et minimale de la fonction objective. On utilise le caractère gras sur fond bleu pour indiquer la meilleure solution pour un scénario donné.

3.3.2. Effet de la taille de la population et du nombre d'itérations :

Notre première analyse se porte sur l'effet de la population et du nombre d'itérations sur un algorithme d'optimisation. En utilisant le DE pour minimiser les quinze fonctions précédentes et en fixant la dimension à D=20, on fait varier nos deux paramètres suivant les scenarios indiqués dans le tableau (3.2). Les résultats de notre simulation sont compilés dans le tableau (3.3)

Scenarios	Nombre d'itérations	Taille de la population
1	1000	10
2	1000	20
3	1000	40
4	1000	100
5	100	50
6	500	50
7	1000	50
8	2000	50

Tableau (3.2) : Variation de la taille de la population et du nombre d'itérations maximal

D'après les résultats présentés dans le tableau (3.3), on remarque une forte dissemblance entre les résultats obtenus par chaque fonction, ceci peut être attribué aux particularités de chacune tel que sa distribution, sa forme, sa nature, sa complexité et son espace de recherche.

On remarque notamment une amélioration des résultats avec l'augmentation de la taille de la population (voir figure 3.1) ainsi que le nombre d'itérations et ceci pour la majorité des fonctions. On note comme exemple la fonction f_{11} dont on remarque une amélioration de la moyenne passant de **5,34E+00** pour une population de 10 à **1,77E-21** pour une population de 100 soit une optimisation de l'ordre de 10^{21}. Pareil pour le nombre d'itérations, la moyenne passe de **4,06E-01** à **2,00E-23** suite à l'incrémentation de 100 à 2000 itérations soit une progression de 10^{22}.

Néanmoins certaines fonctions échappent à ce principe et la moyenne des résultantes ne s'améliore pas forcément malgré la hausse des paramètres. Ceci s'explique par la nature aléatoire des algorithmes méta-heuristiques ce qui peut engendrer quelques solutions beaucoup trop grandes au reste des valeurs produites pouvant ainsi parasiter la moyenne des résultats.

On cite donc la fonction f_8 qui pour les scenarios 6, 7 et 8, malgré l'obtention de résultats assez bons, la moyenne ne se minimise pas pour autant. On note pour le scenario 8, une valeur maximale importante de **2,46E+00** comparée à la valeur minimale de **1,39E-71.** Cette disparité influence grandement la moyenne obtenue, expliquant donc sa détérioration.

Fonction	Population	Itérations	Moyenne	Variance	Médiane	Minimum	Maximum
f_1	10	1000	1,69E+03	1,10E+03	1,35E+03	1,57E+02	4,94E+03
	20	1000	9,38E+01	1,68E+02	2,46E+01	2,52E-01	7,77E+02
	40	1000	5,96E-01	1,72E+00	7,94E-10	2,67E-34	9,86E+00
	100	1000	**4,08E-34**	3,04E-34	3,40E-34	6,36E-35	1,53E-33
	50	100	9,29E+00	3,51E+00	8,85E+00	3,41E+00	1,96E+01
	50	500	5,39E-05	2,90E-04	9,97E-15	1,32E-15	1,99E-03
	50	1000	1,05E-05	6,32E-05	1,26E-33	8,94E-35	4,45E-04
	50	2000	**1,38E-07**	8,25E-07	1,15E-50	7,16E-73	5,74E-06
f_2	10	1000	1,23E+02	4,54E+01	1,19E+02	2,89E+01	2,59E+02
	20	1000	3,55E+01	2,09E+01	2,67E+01	1,35E+01	8,30E+01
	40	1000	1,65E+01	5,33E+00	1,58E+01	8,66E+00	5,01E+01
	100	1000	**1,508E+01**	1,21E+00	1,54E+01	1,16E+01	1,70E+01
	50	100	5,06E+01	2,34E+01	3,96E+01	2,20E+01	1,24E+02
	50	500	1,76E+01	8,77E+00	1,62E+01	8,11E+00	6,91E+01
	50	1000	1,49E+01	2,14E+00	1,53E+01	7,73E+00	1,90E+01
	50	2000	**1,47E+01**	2,29E+00	1,51E+01	4,32E+00	1,82E+01
f_3	10	1000	6,53E-01	7,43E-01	4,11E-01	4,53E-02	4,14E+00
	20	1000	2,99E-02	3,08E-02	2,07E-02	4,57E-03	1,95E-01
	40	1000	8,39E-03	2,60E-03	8,36E-03	2,65E-03	1,56E-02
	100	1000	**5,58E-03**	1,53E-03	5,31E-03	2,79E-03	9,92E-03
	50	100	9,60E-02	2,89E-02	9,50E-02	4,99E-02	1,68E-01
	50	500	1,41E-02	4,24E-03	1,41E-02	3,78E-03	2,40E-02
	50	1000	7,14E-03	1,83E-03	7,68E-03	1,01E-03	1,00E-02
	50	2000	**3,53E-03**	1,12E-03	3,65E-03	6,59E-04	6,01E-03
f_4	10	1000	5,26E+00	4,08E+00	4,21E+00	3,40E-03	1,48E+01
	20	1000	2,26E+00	2,04E+00	1,39E+00	1,09E-03	8,99E+00
	40	1000	1,25E+00	1,16E+00	1,02E+00	1,45E-02	5,44E+00
	100	1000	**5,53E-01**	7,63E-01	2,64E-01	8,25E-06	4,02E+00
	50	100	1,16E+00	8,33E-01	1,05E+00	1,18E-04	3,67E+00
	50	500	8,56E-01	7,79E-01	9,98E-01	9,05E-04	4,60E+00
	50	1000	**7,39E-01**	8,14E-01	3,67E-01	1,02E-02	4,11E+00
	50	2000	1,310E+00	1,314E+00	1,02E+00	3,92E-03	5,36E+00
f_5	10	1000	9,34E+00	1,95E+00	9,44E+00	3,27E+00	1,35E+01
	20	1000	2,27E+00	1,45E+00	2,02E+00	5,77E-02	7,00E+00
	40	1000	6,53E-02	2,29E-01	2,15E-07	6,22E-15	1,16E+00
	100	1000	**3,94E-15**	1,72E-15	2,66E-15	2,66E-15	6,22E-15
	50	100	2,02E+00	3,36E-01	2,06E+00	1,19E+00	2,68E+00
	50	500	1,48E-02	1,05E-01	2,52E-08	9,68E-09	7,41E-01
	50	1000	2,14E-04	1,47E-03	6,22E-15	2,66E-15	1,04E-02
	50	2000	**1,08E-05**	5,61E-05	2,66E-15	2,66E-15	3,68E-04
f_6	10	1000	6,40E-03	1,11E-02	8,23E-04	4,81E-09	4,75E-02
	20	1000	1,05E-03	2,46E-03	7,74E-05	2,75E-09	1,06E-02
	40	1000	5,70E-05	1,47E-04	4,68E-06	3,08E-08	7,09E-04
	100	1000	**2,82E-06**	4,32E-06	8,96E-07	3,50E-09	2,18E-05
	50	100	8,36E+00	1,83E+00	8,59E+00	4,60E+00	1,24E+01
	50	500	9,68E-03	8,40E-03	6,83E-03	9,98E-04	4,51E-02
	50	1000	2,19E-05	3,60E-05	5,19E-06	1,79E-10	1,57E-04
	50	2000	**1,11E-08**	4,76E-08	8,35E-11	1,55E-15	3,32E-07
f_7	10	1000	6,47E-01	5,46E-01	4,35E-01	3,96E-02	2,38E+00
	20	1000	6,62E-03	1,42E-02	1,67E-03	1,76E-07	8,28E-02
	40	1000	**1,84E-08**	9,06E-08	3,40E-13	3,46E-18	4,86E-07
	100	1000	1,02E-05	1,37E-05	4,69E-06	9,15E-12	6,59E-05
	50	100	3,50E-01	1,47E-01	3,37E-01	1,24E-01	8,28E-01
	50	500	1,46E-04	1,74E-04	6,79E-05	2,59E-06	7,50E-04
	50	1000	8,99E-10	3,04E-09	5,23E-13	1,55E-18	1,94E-08
	50	2000	**1,05E-21**	7,39E-21	9,38E-44	4,42E-58	5,23E-20
f_8	10	1000	8,98E+05	4,22E+06	1,94E+03	4,11E+02	2,44E+07
	20	1000	1,46E+02	2,22E+02	4,70E+01	2,73E-01	9,92E+02
	40	1000	1,02E+00	3,19E+00	6,49E-09	3,92E-34	1,49E+01
	100	1000	**1,29E-33**	8,41E-34	1,03E-33	2,96E-34	3,80E-33
	50	100	2,23E+01	8,35E+00	2,16E+01	8,50E+00	5,29E+01
	50	500	**1,01E-02**	7,09E-02	2,51E-14	6,15E-15	5,01E-01
	50	1000	2,48E-02	1,73E-01	7,03E-33	1,93E-34	1,22E+00
	50	2000	4,93E-02	3,48E-01	5,23E-41	1,39E-71	2,46E+00

f_9	10	1000	2,50E+00	1,83E+00	1,94E+00	2,83E-01	9,39E+00
	20	1000	2,36E-01	3,23E-01	9,01E-02	3,79E-05	1,70E+00
	40	1000	1,20E-03	8,07E-03	3,52E-13	1,50E-32	5,71E-02
	100	1000	**1,50E-32**	1,38E-47	1,50E-32	1,50E-32	1,50E-32
	50	100	6,26E-02	2,14E-02	5,87E-02	1,97E-02	1,15E-01
	50	500	3,66E-08	2,26E-07	6,55E-17	1,51E-17	1,59E-06
	50	1000	**1,31E-08**	8,93E-08	1,50E-32	1,50E-32	6,32E-07
	50	2000	6,58E-08	3,26E-07	1,49E-32	1,50E-32	1,73E-06
f_{10}	10	1000	1,36E+00	1,83E+00	9,03E-01	8,47E-01	1,34E+01
	20	1000	8,76E-01	6,16E-02	8,54E-01	8,46E-01	1,12E+00
	40	1000	8,53E-01	6,06E-03	8,52E-01	8,45E-01	8,73E-01
	100	1000	**8,50E-01**	3,92E-03	8,50E-01	8,44E-01	8,59E-01
	50	100	9,45E-01	4,50E-02	9,31E-01	8,82E-01	1,06E+00
	50	500	8,60E-01	7,58E-03	8,58E-01	8,49E-01	8,89E-01
	50	1000	8,56E-01	3,00E-02	8,51E-01	8,44E-01	1,06E+00
	50	2000	**8,49E-01**	4,95E-03	8,47E-01	8,44E-01	8,75E-01
f_{11}	10	1000	5,34E+00	4,19E+00	4,39E+00	4,49E-01	1,81E+01
	20	1000	1,10E-01	1,56E-01	3,54E-02	1,36E-09	6,14E-01
	40	1000	1,71E-03	8,45E-03	3,85E-21	9,96E-22	4,78E-02
	100	1000	**1,77E-21**	4,94E-22	1,74E-21	1,03E-21	3,35E-21
	50	100	4,06E-01	7,22E-02	4,02E-01	2,78E-01	5,55E-01
	50	500	5,31E-04	3,75E-03	4,51E-10	1,93E-10	2,65E-02
	50	1000	3,64E-11	2,57E-10	2,53E-21	6,51E-22	1,82E-09
	50	2000	**2,00E-23**	1,41E-22	1,08E-43	2,79E-44	9,99E-22
f_{12}	10	1000	1,59E+07	7,93E+07	1,14E-22	1,32E-131	4,41E+08
	20	1000	2,42E+01	1,71E+02	9,57E-135	1,88E-143	1,21E+03
	40	1000	4,66E-132	3,15E-131	8,63E-138	2,17E-146	2,23E-130
	100	1000	**5,25E-136**	2,27E-135	9,26E-139	4,64E-143	1,44E-134
	50	100	5,39E-07	1,86E-06	4,45E-09	1,46E-12	1,06E-05
	50	500	3,61E-63	1,65E-62	3,27E-66	2,30E-72	1,04E-61
	50	1000	3,08E-132	2,16E-131	5,36E-138	1,95E-144	1,53E-130
	50	2000	**1,66E-276**	0,00E+00	2,41E-282	1,64E-292	4,16E-275
f_{13}	10	1000	1,50E+02	9,18E+01	1,37E+02	2,93E+01	4,90E+02
	20	1000	6,04E+00	8,79E+00	1,83E+00	8,36E-03	3,59E+01
	40	1000	2,98E-03	1,14E-02	3,53E-14	4,14E-35	6,22E-02
	100	1000	**2,85E-35**	2,05E-35	2,35E-35	6,65E-36	8,69E-35
	50	100	6,72E-01	2,12E-01	6,77E-01	3,03E-01	1,18E+00
	50	500	**1,06E-05**	4,91E-05	6,42E-16	1,36E-16	3,16E-04
	50	1000	6,97E-04	4,90E-03	1,33E-34	8,43E-36	3,47E-02
	50	2000	7,42E-03	5,24E-02	9,65E-55	6,32E-74	3,70E-01
f_{14}	10	1000	2,88E+01	2,12E+01	2,23E+01	3,66E+00	1,03E+02
	20	1000	8,43E+00	5,54E+00	7,29E+00	1,60E+00	3,66E+01
	40	1000	5,57E+00	1,98E+00	5,18E+00	2,04E+00	9,84E+00
	100	1000	**3,92E+00**	1,10E+00	3,84E+00	1,65E+00	6,16E+00
	50	100	1,64E+02	3,07E+01	1,68E+02	7,81E+01	2,38E+02
	50	500	4,06E+01	1,49E+01	4,02E+01	1,46E+01	7,78E+01
	50	1000	5,23E+00	2,04E+00	4,84E+00	1,64E+00	1,20E+01
	50	2000	**1,34E-02**	1,18E-02	1,09E-02	9,98E-04	6,16E-02
f_{15}	10	1000	5,74E+07	1,90E+08	8,80E+06	7,98E+04	1,31E+09
	20	1000	3,38E+05	9,50E+05	8,97E+03	7,39E+01	5,87E+06
	40	1000	1,57E+02	2,88E+02	2,95E+01	8,69E-05	1,65E+03
	100	1000	**2,35E+01**	3,14E+01	9,11E+00	2,14E-08	1,39E+02
	50	100	4,16E+03	1,12E+03	3,95E+03	2,82E+03	1,10E+04
	50	500	1,33E+02	3,12E+02	8,81E+00	3,74E-12	1,47E+03
	50	1000	**6,25E+01**	1,06E+02	1,46E+01	6,52E-06	4,12E+02
	50	2000	8,39E+01	1,36E+02	4,31E+01	7,56E-12	8,02E+02

Tableau (3.3) : Résultats de la variation de la taille de population et du nombre d'itérations sur le DE

Pour une meilleure observation de l'effet de la taille de la population, on réalise la figure suivante :

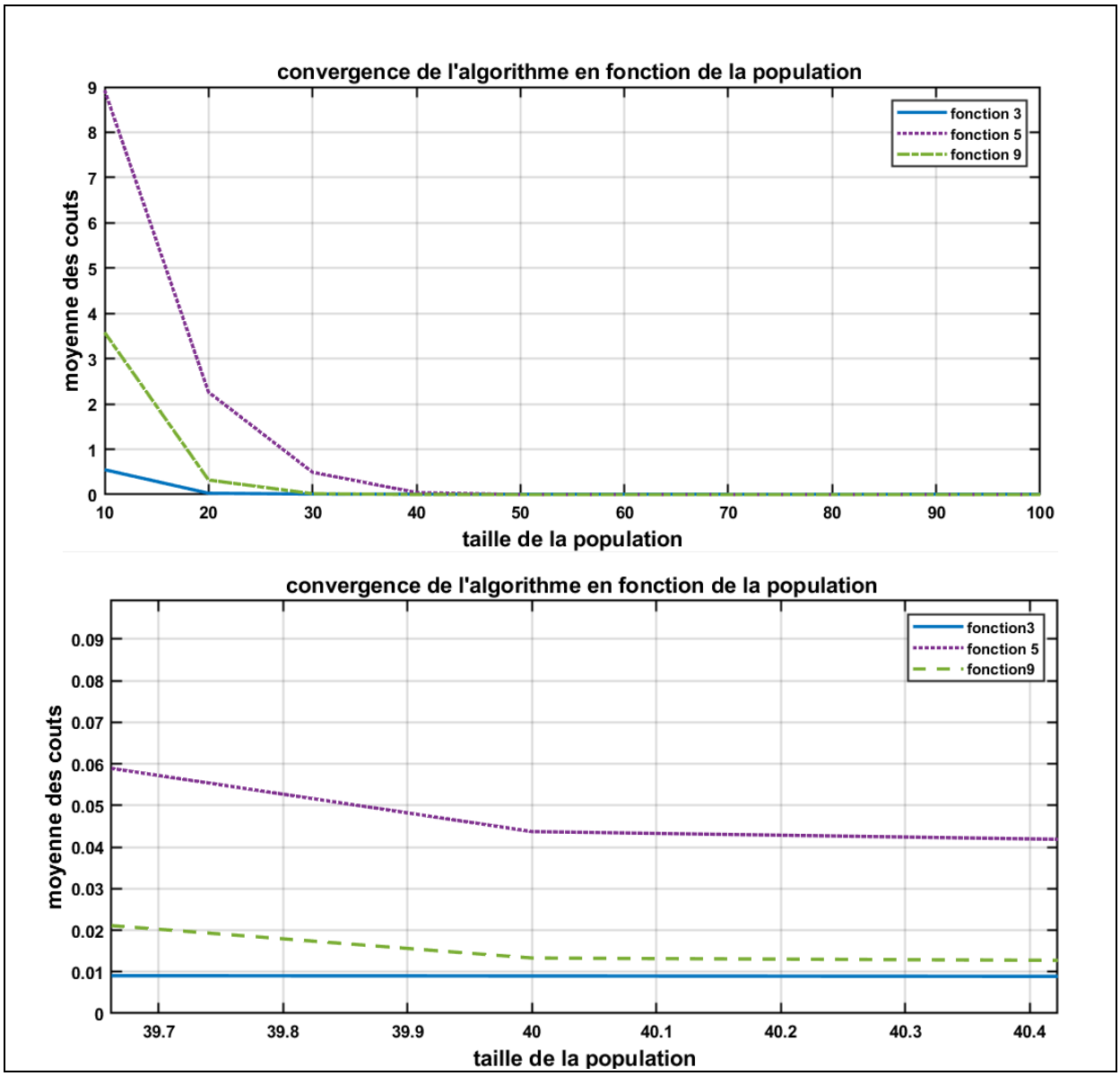

Figure (3.1) : Effet de la taille de la population sur la convergence de l'algorithme DE pour les fonctions f_3, f_5 et f_9

3.4.3. Effet de la dimension :

Toujours en utilisant le DE, on procède à la seconde analyse portant sur l'effet de la dimension. Pour ce fait, on fixe la taille de la population à 50, ainsi que le nombre d'itérations à 500 et on réalise un test pour chacune des dimensions D = {5, 10, 30, 100} sur dix fonctions benchmark à titre d'exemple, on choisit parmi le tableau (3,1) les fonctions suivantes : $f_1, f_2, f_3, f_8, f_9, f_{10}, f_{11}, f_{12}, f_{13}$ et f_{14}.

On obtient les résultats décris dans le tableau (3,4)

Fonction	Dimension	Moyenne	Variance	Médiane	Minimum	Maximum
f_1	5	**1,41E-57**	4,90E-57	1,72E-58	5,94E-61	2,73E-56
	10	3,98E-08	2,82E-07	4,20E-29	1,79E-30	1,99E-06
	30	2,30E-01	1,15E+00	8,61E-10	2,25E-10	7,44E+00
	100	6,89E+00	1,43E+01	1,19E+00	9,85E-02	7,05E+01
f_2	5	**5,59E-01**	4,90E-01	5,03E-01	2,75E-04	2,30E+00
	10	5,12E+00	1,63E+00	5,74E+00	2,58E-01	9,08E+00
	30	2,83E+01	1,03E+01	2,64E+01	1,86E+01	7,96E+01
	100	1,78E+02	5,04E+01	1,70E+02	1,03E+02	3,03E+02
f_3	5	**9,80E-04**	5,41E-04	9,58E-04	1,89E-04	2,70E-03
	10	4,01E-03	1,66E-03	3,83E-03	1,35E-03	8,32E-03
	30	2,76E-02	6,50E-03	2,60E-02	1,56E-02	4,31E-02
	100	2,39E-01	3,08E-02	2,37E-01	1,81E-01	3,03E-01
f_8	5	**2,54E-45**	1,80E-44	4,35E-57	5,09E-59	1,27E-43
	10	2,22E-26	1,54E-25	2,39E-28	3,52E-29	1,09E-24
	30	1,05E+00	7,37E+00	1,68E-09	5,01E-10	5,21E+01
	100	9,54E+00	1,73E+01	2,71E+00	1,49E-01	7,71E+01
f_9	5	**1,50E-32**	1,38E-47	1,50E-32	1,49E-32	1,51E-32
	10	1,30E-19	9,21E-19	8,95E-32	1,50E-32	6,51E-18
	30	2,68E-04	1,71E-03	7,97E-12	2,68E-12	1,21E-02
	100	7,39E-02	1,55E-01	3,81E-03	1,76E-03	5,52E-01
f_{10}	5	**7,00E-01**	9,76E-03	7,00E-01	6,78E-01	7,23E-01
	10	7,84E-01	5,35E-03	7,83E-01	7,75E-01	7,99E-01
	30	9,02E-01	2,58E-02	8,97E-01	8,81E-01	1,06E+00
	100	9,83E-01	2,89E-02	9,76E-01	9,43E-01	1,08E+00
f_{11}	5	**1,48E-32**	1,97E-32	9,20E-33	8,75E-34	1,27E-31
	10	9,23E-18	4,65E-18	8,62E-18	2,11E-18	1,97E-17
	30	2,20E-05	1,53E-04	3,41E-07	1,94E-07	1,08E-03
	100	6,56E-02	2,12E-02	6,06E-02	4,35E-02	1,91E-01
f_{12}	5	**9,52E-100**	3,41E-99	2,01E-104	2,43E-109	1,72E-98
	10	3,78E-76	1,33E-75	1,28E-79	8,60E-85	7,60E-75
	30	2,05E-59	1,15E-58	1,42E-62	2,18E-69	8,11E-58
	100	5,62E-52	2,83E-51	1,02E-55	4,82E-59	1,80E-50
f_{13}	5	**3,02E-59**	1,09E-58	2,24E-60	2,35E-62	7,52E-58
	10	4,50E-30	7,40E-30	1,61E-30	4,96E-32	3,80E-29
	30	4,88E-02	3,14E-01	1,16E-10	2,66E-11	2,22E+00
	100	7,36E+00	1,59E+01	1,90E-01	4,30E-02	7,24E+01
f_{14}	5	**1,69E-11**	8,84E-11	2,32E-19	1,07E-21	5,66E-10
	10	4,31E-02	4,37E-02	2,93E-02	2,98E-03	2,13E-01
	30	1,83E+02	3,31E+01	1,85E+02	1,17E+02	2,60E+02
	100	1,62E+03	1,42E+02	1,64E+03	1,30E+03	1,96E+03

Tableau (3.4) : Effet de la dimension sur l'algorithme d'optimisation (DE)

D'après le tableau (3,4) on remarque un comportement unanime pour toutes les fonctions utilisées, consistant en une détérioration des résultats obtenues avec l'augmentation de la dimension.

On cite comme exemple la fonction 13 qui perd 59 ordres de grandeur en passant d'une moyenne de **3,02E-59** pour une dimension de 5, à **7,36E+00** pour une dimension de 100.

- Nos observations sur le comportement du DE face à l'augmentation de la taille de la population, le nombre d'itérations et la dimension est applicable pour le reste de nos algorithmes. Il est aussi important de noter que la hausse des paramètres nécessite un plus grand besoin de ressources de calcul se traduisant par un temps d'exécution plus long.

3.4.3. Effet du paramètre de mutation sur le GA :

On procède ensuite sur une étude de l'effet du paramètre de mutation sur l'algorithme génétique (GA). En fixant la taille de la population, le nombre d'itérations et la dimension à 50, 500 et 20 chacune, on test l'algorithme sur les cinq fonctions f_1, f_2, f_3, f_4 et f_5 pour les pourcentages de mutation suivant : {0, 10, 20, 50, 100}

Le tableau (3.5) représente les résultats de l'effet du paramètre de mutation sur les performances de l'algorithme génétique (GA) pour les quatre scenarios suivant : { 0, 0.1 , 0.2 , 0.5 et 1 }

Fonction	Taux de mutation	Moyenne	Variance	Médiane	Minimum	Maximum
f_1	0	5,64E+03	1,08E+04	4,07E+02	4,29E-04	3,94E+04
	0,1	**3,80E+00**	1,06E+01	3,65E-01	1,10E-03	6,89E+01
	0,2	5,64E+00	1,20E+01	8,32E-01	4,21E-02	7,13E+01
	0,5	2,62E+03	1,02E+03	2,68E+03	7,32E+02	4,94E+03
	1	1,98E+04	2,36E+03	1,99E+04	1,32E+04	2,36E+04
f_2	0	5,19E+02	1,03E+03	6,20E+01	1,75E+00	4,52E+03
	0,1	1,13E+01	8,72E+00	1,86E+01	3,31E-03	1,99E+01
	0,2	**1,08E+01**	7,91E+00	1,12E+01	5,03E-04	2,33E+01
	0,5	1,67E+02	5,25E+01	1,63E+02	5,87E+01	2,84E+02
	1	1,24E+03	2,21E+02	1,23E+03	5,69E+02	1,69E+03
f_3	0	2,97E+00	5,65E+00	1,96E-01	3,57E-05	2,56E+01
	0,1	**1,81E-03**	1,49E-03	1,43E-03	1,36E-04	7,53E-03
	0,2	1,96E-02	1,42E-02	1,75E-02	1,60E-05	6,95E-02
	0,5	1,00E+00	3,41E-01	1,02E+00	2,59E-01	1,74E+00
	1	1,10E+01	2,02E+00	1,11E+01	5,18E+00	1,53E+01
f_4	0	4,15E-01	5,66E-02	5,76E-07	1,23E+00	3,18E-01
	0,1	4,33E-05	1,39E-06	2,84E-14	1,94E-04	1,60E-05
	0,2	3,62E-05	5,36E-06	9,43E-09	1,86E-04	1,96E-05
	0,5	6,03E-05	5,77E-06	2,76E-08	3,92E-04	2,50E-05
	1	**2,84E-05**	2,59E-06	3,03E-11	1,44E-04	1,56E-05
f_5	0	1,06E+01	6,28E+00	1,11E+01	5,88E-03	1,97E+01
	0,1	**1,11E+00**	1,14E+00	5,37E-01	1,49E-02	3,62E+00
	0,2	8,77E-01	9,15E-01	6,27E-01	9,73E-04	3,57E+00
	0,5	1,23E+01	1,35E+00	1,28E+01	9,07E+00	1,42E+01
	1	1,91E+01	3,24E-01	1,92E+01	1,83E+01	1,97E+01

Tableau (3.5) : Effet de la variation du taux de mutation sur le GA

D’après le tableau (3.5) on constate que :

Lorsque le taux de mutation est trop faible, entre [0, 0.5], toutes les fonctions testées ont des moyennes élevées donc elles ne s’améliorent pas (n’approchent pas vers l’optimum), car la diversité génétique au sein de la population diminue, ce qui peut conduire à une convergence prématurée vers des régions locales de l'espace des solutions. Cela limite la capacité de l'algorithme à explorer de manière exhaustive l'espace de recherche et peut entraîner une stagnation des performances.

En revanche, un taux de mutation trop élevé, entre [0.5, 1], peut entraîner une exploration excessive de l'espace de recherche et une perte de convergence car les nouvelles solutions générées par mutation peuvent être trop éloignées des solutions déjà présentes dans la population, ce qui peut rendre difficile l'amélioration progressive des individus et ralentir la convergence de l'algorithme, le cas des fonctions f_1, f_2, f_3, f_4 et f_5.

Pour la fonction 4 qui s’est améliorée pour un taux de mutation égal à 1, cela s’explique par le fait que certains problèmes peuvent bénéficier d'une mutation plus agressive pour sortir des minimums locaux, tandis que d'autres peuvent nécessiter une mutation plus conservatrice pour préserver les bonnes solutions déjà présentes dans la population, le cas des fonctions f_1, f_2, f_3 et f_5 qui s’améliorent lorsque le taux de mutation entre [0.1, 0.2].

D’après ces résultats, on peut déduire que ce paramètre peut varier en fonction de la nature du problème d'optimisation et de sa complexité, donc trouver un taux de mutation adéquat favorise l'équilibre entre l'exploration et l'exploitation, ce qui permet d'obtenir de bonnes performances d'optimisation.

3.4.5. Comparaison entre algorithmes :

On finalise notre travail par une étude comparative de nos quatre algorithmes. Pour ce, on réalise une analyse préliminaire de nos algorithmes consistant en l’observation des résultats statistiques de nos quinze fonctions pour les paramètres de dimension, itérations et population fixes à 20, 500 et 50 chacun

Nous sommes également intéressés par l'enregistrement de la vitesse de convergence de chaque algorithme (testé sur quatre fonctions) ainsi qu’à la relève de leurs temps d’exécutions et cela pour divers scénarios

Le tableau (3.6) représente les résultats préliminaires de la simulation de nos quatre algorithmes sur les quinze fonctions benchmark. Les meilleurs résultats (solution minimal) sont donnés en gras sur un fond bleu alors que les plus mauvaises moyennes sont soulignées :

Le symbole (+) dans le tableau (3.6) représente le nombre de cas ou chaque l’algorithme obtient les meilleures solutions

Le symbole (-) dans le tableau (3.6) représente le nombre de cas ou chaque algorithme obtient les moins bonnes solutions

f	Moyenne				Variance				Médiane			
	GA	DE	PSO	ABC	GA	DE	PSO	ABC	GA	DE	PSO	ABC
f_1	3,26E+00	1,98E-02	**1,18E-12**	9,60E-08	1,08E+01	9,75E-02	**1,05E-23**	1,61E-14	4,97E-01	**9,51E-15**	2,91E-13	4,92E-08
f_2	1,37E+01	1,53E+01	3,11E+01	**1,28E+01**	7,65E+00	**2,11E+00**	2,25E+03	7,37E+00	1,86E+01	1,58E+01	1,40E+01	**1,33E+01**
f_3	**1,74E-03**	1,36E-02	6,38E-02	1,34E-01	1,74E-03	3,74E-03	1,44E-01	**1,25E-03**	**1,19E-03**	1,35E-02	8,97E-03	1,35E-01
f_4	**8,79E-06**	8,83E-01	4,04E+01	6,16E-05	1,62E-05	8,76E-01	4,33E+02	**1,65E-08**	**1,88E-06**	7,06E-01	3,53E+01	5,08E-06
f_5	8,44E-01	1,89E-02	1,32E-01	**3,25E-03**	9,79E-01	1,24E-01	1,61E-01	**6,59E-06**	5,38E-01	**2,07E-08**	3,36E-07	2,46E-03
f_6	2,39E-01	**2,46E-03**	2,27E-02	5,63E-03	3,20E-01	1,72E-02	3,80E-04	**6,68E-05**	8,65E-02	**1,82E-13**	1,72E-02	1,36E-03
f_7	6,40E-02	**1,48E-04**	7,10E-01	3,61E-03	1,52E-01	1,96E-04	2,70E+00	**8,60E-06**	1,41E-02	7,64E-05	**1,44E-07**	2,64E-03
f_8	7,73E+02	1,56E-02	2,29E+05	**2,15E-06**	1,57E+03	9,35E-02	9,35E+11	**3,17E-11**	8,46E+01	**1,82E-14**	7,27E+04	4,21E-07
f_9	1,42E-02	**9,20E-10**	1,19E+00	1,96E-09	2,91E-02	6,49E-09	4,02E+00	**9,57E-18**	1,36E-03	**7,17E-17**	4,54E-01	7,54E-10
f_{10}	1,18E+00	**8,61E-01**	9,09E-01	9,04E-01	1,23E-01	8,66E-03	5,19E-03	**2,03E-03**	1,20E+00	**8,59E-01**	8,84E-01	8,94E-01
f_{11}	4,74E-01	**4,85E-10**	4,60E+00	5,66E-05	5,03E-01	**1,89E-10**	4,17E+01	5,92E-10	3,30E-01	**4,50E-10**	3,57E-07	5,38E-05
f_{12}	3,27E+01	**1,03E-61**	1,63E-44	3,02E-02	6,41E+01	6,89E-61	**9,51E-87**	9,95E-03	3,14E+00	**3,49E-66**	2,40E-48	1,76E-03
f_{13}	1,92E-01	9,68E-06	2,20E+01	**5,44E-09**	4,05E-01	3,80E-05	2,16E+03	**3,00E-17**	3,76E-02	**8,03E-16**	5,74E-14	3,39E-09
f_{14}	**3,47E-02**	3,99E+01	3,12E+01	1,52E+02	**5,44E-02**	1,14E+01	1,35E+03	7,32E+02	**1,53E-02**	3,92E+01	2,55E+01	1,55E+02
f_{15}	3,78E+02	**1,21E+02**	2,39E+10	2,59E+04	7,23E+02	**2,35E+02**	1,38E+20	6,36E+07	1,34E+02	**1,42E+01**	2,23E+10	2,68E+04
+	3	**7**	1	4								
-	5	**0**	8	2								

Tableau (3.6) : Résultats comparatifs de la qualité de convergence des fonctions benchmark

Chaque algorithme réagis différemment pour chaque fonction, il peut donner les meilleurs résultats pour l'une et les plus mauvais dans l'autre, on cite comme exemple l'ABC qui donne la meilleure moyenne pour la fonction f : **2,15E-06** et la plus mauvaise pour la fonction: **1,52E+02**

On note donc que pour ce cas précis, l'algorithme le plus efficace le plus souvent est le DE, résultant en une meilleure optimisation comparer au trois autres, sept fois sur quinze, en l'occurrence les fonctions f_6, f_7, f_9, f_{10}, f_{11}, f_{12} et f_{15} et n'obtient en aucun de nos cas la moins bonne solution.

Il est suivi par l'ABC avec quatre résultats préférables aux autres soit les fonctions f_2, f_4, f_8 et f_{13} et résulte en les plus mauvaises solutions deux fois en la fonction f_3 et la fonction f_{14}.

Vien ensuite le GA qui obtient trois fois la meilleure solution en les fonctions f_3, f_4 et f_{14} et cinq fois la plus mauvaise, les fonctions f_1, f_5, f_6, f_{10} et f_{12}.

Le moins bon pour ce cas d'étude est donc le PSO qui non seulement n'a la meilleure moyenne que pour la fonction 1, il a les moins bons résultats pour huit fonctions : f_2, f_4, f_7, f_8, f_9, f_{11}, f_{13} et f_{15}

3.4.5.1.Vitesse de convergence :

Dans le but de comprendre le comportement de nos algorithmes, on poursuit l'étude sur quatre fonctions (f_1, f_3 , f_5, f_{13}). On obtient les résultats compilés dans le tableau (3.8) ainsi que les graphes de la vitesse de convergence présentés par les figures (3,1), (3,2), (3,3) et (3,4).

Scénarios	Dimension	Population (Np)	Itérations(iter)
I	20	50	500
II	100	50	500
III	100	50	1000
IV	100	100	500
V	100	100	1000

Tableau (3.7) : Scénarios de l'étude de la vitesse de convergence

f	D	Np	Iter	Moyenne				Variance				Médiane			
				ABC	PSO	DE	GA	ABC	PSO	DE	GA	ABC	PSO	DE	GA
f_1	20	50	500	9,60E-08	**1,18E-12**	1,98E-02	3,26E+00	1,61E-14	**1,05E-23**	9,75E-02	1,08E+01	4,92E-08	2,91E-13	**9,51E-15**	4,97E-01
	100	50	500	8,70E+03	1,07E+04	**6,54E+00**	1,54E+04	3,39E+06	7,68E+07	**1,31E+01**	3,36E+03	8,62E+03	1,05E+04	**1,21E+00**	1,58E+04
	100	50	1000	8,12E+01	1,20E+04	**1,40E+01**	1,32E+04	1,58E+03	9,01E+07	**3,48E+01**	2,49E+03	7,04E+01	1,00E+04	**3,22E-01**	1,32E+04
	100	100	500	7,57E+03	6,77E+03	**8,96E-02**	1,20E+04	2,03E+06	5,08E+07	**1,26E-02**	3,10E+03	7,59E+03	1,00E+04	**8,91E-02**	1,17E+04
	100	100	1000	5,86E+01	6,80E+03	**2,74E-08**	1,05E+04	4,63E+02	5,49E+07	**5,34E-09**	2,50E+03	5,81E+01	1,00E+04	**2,70E-08**	1,05E+04
f_3	20	50	500	1,34E-01	6,38E-02	1,36E-02	**1,74E-03**	**1,25E-03**	1,44E-01	3,74E-03	1,74E-03	1,35E-01	8,97E-03	1,35E-02	**1,19E-03**
	100	50	500	1,44E+01	1,57E+02	**2,69E-01**	3,38E+01	2,67E+01	6,19E+03	**1,90E-01**	1,18E+01	1,39E+01	1,65E+02	**2,25E-01**	3,29E+01
	100	50	1000	3,00E+00	1,50E+02	**1,09E-01**	2,66E+01	1,96E-01	1,08E+04	**2,38E-02**	9,31E+00	3,00E+00	1,23E+02	**1,04E-01**	2,57E+01
	100	100	500	1,16E+01	1,46E+02	**1,89E-01**	2,48E+01	1,40E+01	1,01E+04	**2,22E-02**	8,54E+00	1,09E+01	1,12E+02	**1,87E-01**	2,51E+01
	100	100	1000	2,64E+00	1,68E+02	**7,93E-02**	2,10E+01	1,14E-01	8,90E+03	**8,66E-03**	8,04E+00	2,66E+00	1,63E+02	**7,88E-02**	2,18E+01
f_5	20	50	500	**3,25E-03**	1,32E-01	1,89E-02	8,44E-01	**6,59E-06**	1,61E-01	1,24E-01	9,79E-01	2,46E-03	3,36E-07	**2,07E-08**	5,38E-01
	100	50	500	1,28E+01	1,40E+01	**2,52E-01**	1,23E+01	2,79E-01	7,64E+00	**2,61E-01**	1,07E+00	1,29E+01	1,41E+01	**1,95E-01**	1,25E+01
	100	50	1000	4,70E+00	1,34E+01	**3,65E-01**	1,18E+01	**8,78E-02**	9,92E+00	4,11E-01	8,55E-01	4,68E+00	1,35E+01	**3,01E-01**	1,20E+01
	100	100	500	1,22E+01	1,05E+01	**4,41E-02**	1,122E+01	3,10E-01	1,77E+01	**2,45E-02**	1,06E+00	1,22E+01	1,16E+01	**4,04E-02**	1,13E+01
	100	100	1000	4,42E+00	8,91E+00	**4,13E-03**	1,121E+01	5,12E-02	1,79E+01	**2,90E-02**	8,55E-01	4,45E+00	1,11E+01	**2,01E-05**	1,14E+01
f_{13}	20	50	500	**5,44E-09**	2,20E+01	9,68E-06	1,92E-01	**3,00E-17**	2,16E+03	3,80E-05	4,05E-01	3,39E-09	5,74E-14	**8,03E-16**	3,76E-02
	100	50	500	4,03E+03	8,63E+03	**6,32E+00**	6,05E+03	6,20E+05	2,33E+07	**1,02E+01**	1,38E+03	4,16E+03	7,51E+03	**1,66E+00**	6,17E+03
	100	50	1000	3,93E+01	8,66E+03	**2,58E+00**	5,59E+03	3,29E+02	2,21E+07	**1,15E+01**	1,16E+03	3,68E+01	8,17E+03	**1,93E-01**	5,57E+03
	100	100	500	3,51E+03	6,10E+03	**3,81E-02**	5,72E+03	5,51E+05	1,41E+07	**5,70E-03**	1,23E+03	3,43E+03	5,05E+03	**3,80E-02**	5,86E+03
	100	100	1000	2,40E+01	6,94E+03	**1,53E-05**	4,68E+03	1,17E+02	2,23E+07	**1,08E-04**	1,01E+03	2,42E+01	5,35E+03	**1,18E-08**	4,80E+03

Tableau (3.8) : Résultats de la convergence de quatre fonctions benchmark pour l'ABC, PSO, DE et GA

Toutes les fonctions choisis trouvent leur optimum grâce à des algorithmes différents du DE pour leur configuration initiale (à dimension restreinte : D=20) : le PSO pour f_1, le GA pour f_3 et l'ABC pour f_5 et f_{13}

Quand on augmente la dimension à 100, le DE devient l'algorithme le plus efficace et donne systématiquement les meilleurs résultats.

L'ABC voit une amélioration lors de la hausse du nombre d'itérations et de la taille de la population comme aperçu dans la fonction f_{13} dont la moyenne passe de **4,03E+03** pour une population de 50 et un nombre d'itérations de 500 à **2,40E+01** pour une population égale à 100 et itérations égales à 1000.

En observant le GA, on retient que pour une dimension aussi importante, l'amélioration des résultats n'est pas toujours à l'ordre, où est souvent très minime.

Notant pour le GA, la fonction f_5 passant de **1,23E+01** à **1,121E+01** pour une population et itérations allant de 50 et 500 à 100 et 1000.

Ceci peut aussi s'appliquer pour le PSO pour certain cas tel que la fonction f_{13}.

Pour une meilleure compréhension du comportement de nos algorithmes, on réalise les graphismes illustrés dans les figures (3.2), (3.3), (3.4) et (3.5).

Chacune représente la vitesse de convergence des fonctions f_1, f_2, f_3 et f_4 successivement pour les quatre algorithmes pour trois scenarios :

a : scenario I (D=20 ; iter=500 ; population=50)

b :scenario III (D=100 ; iter=500 ; population=50)

c : scénario V(D=100 ; iter=1000 ; population=100)

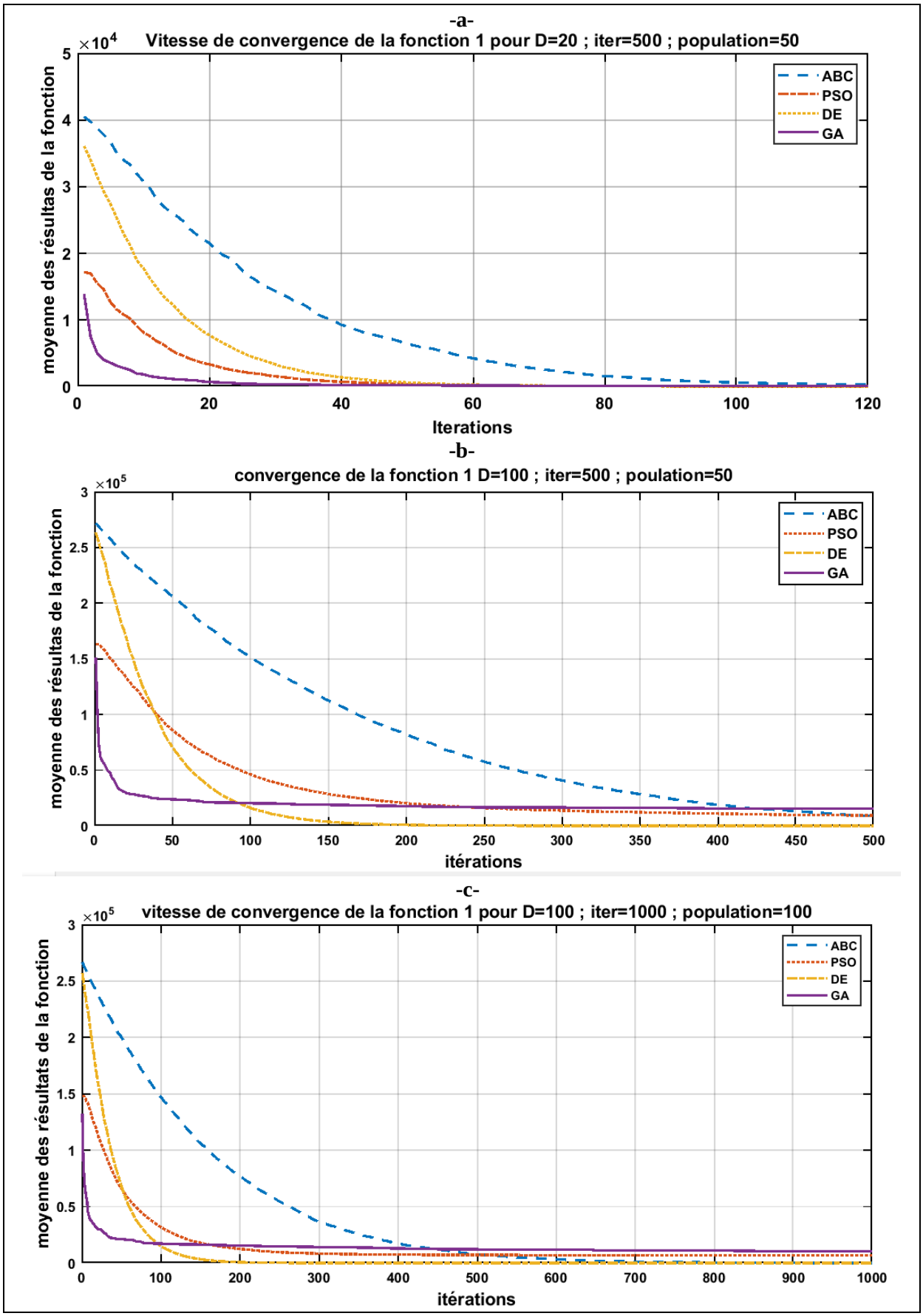

Figure (3.2) : Vitesse de convergence de la fonction 1 pour différents scénarios (a ; b ; c)

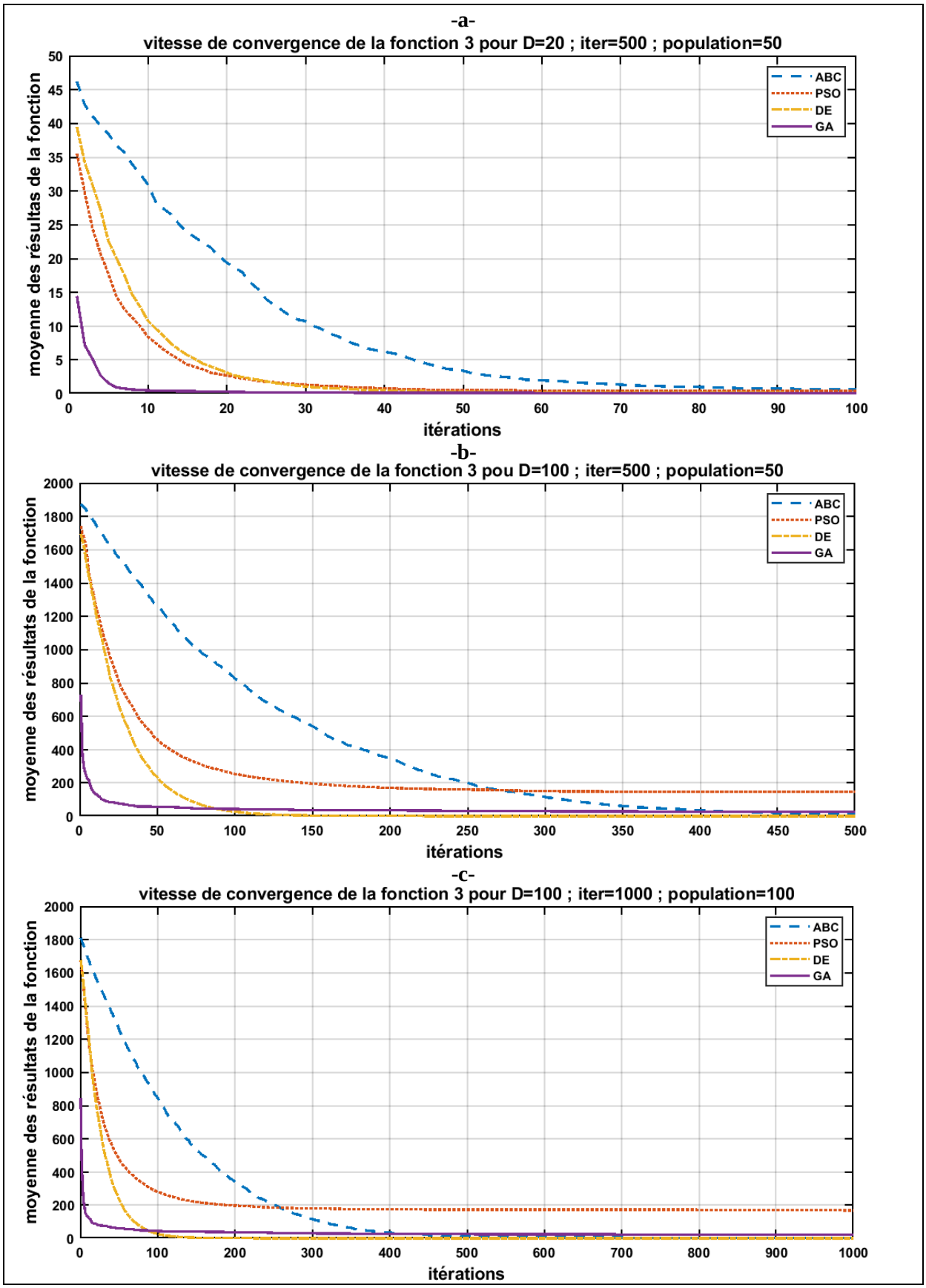

***Figure* (3.3) :** Vitesse de convergence de la fonction 3 pour différents scénarios (a ; b ; c)

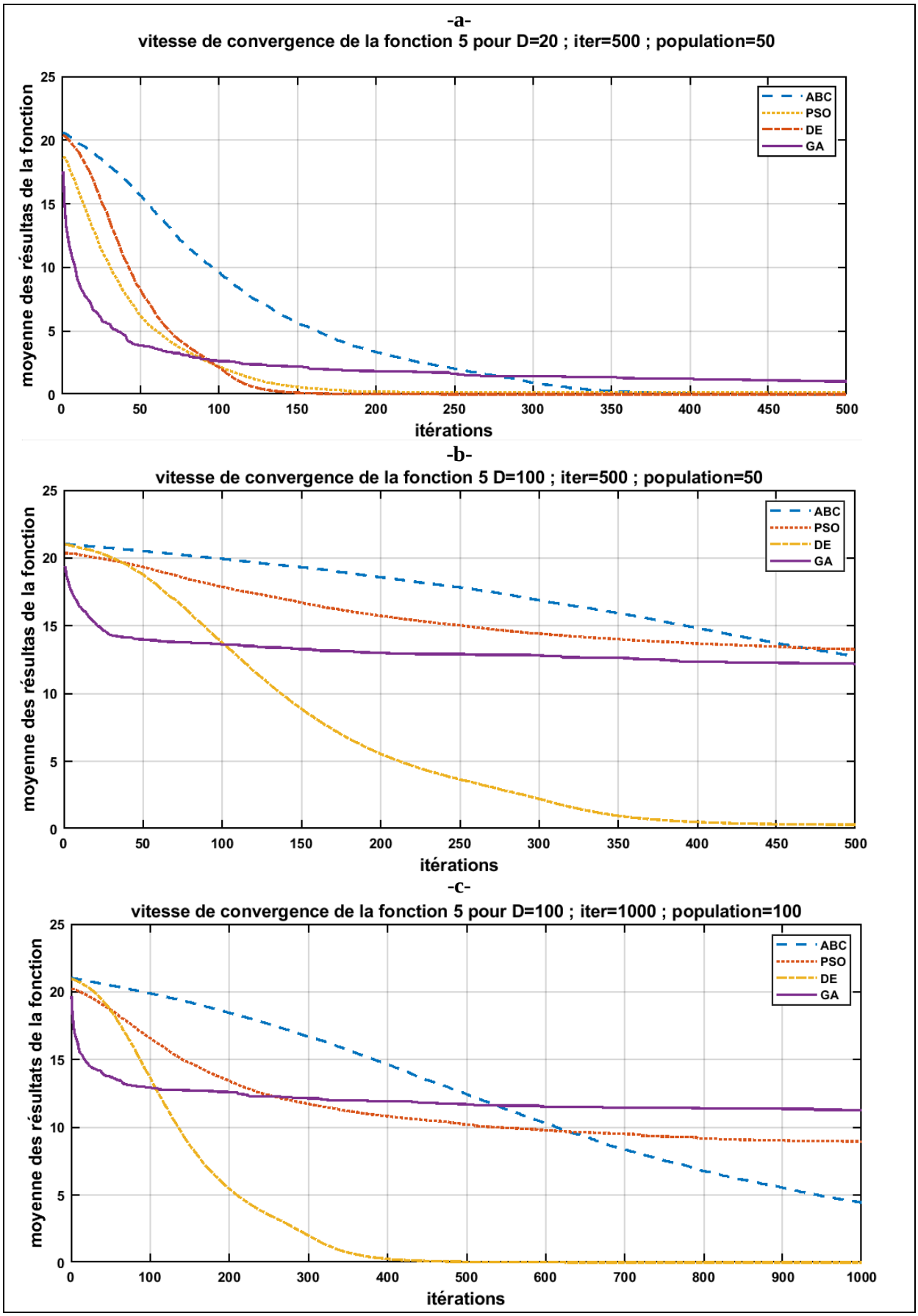

Figure (3.4) : Vitesse de convergence de la fonction 5 pour différents scénarios (a ; b ; c)

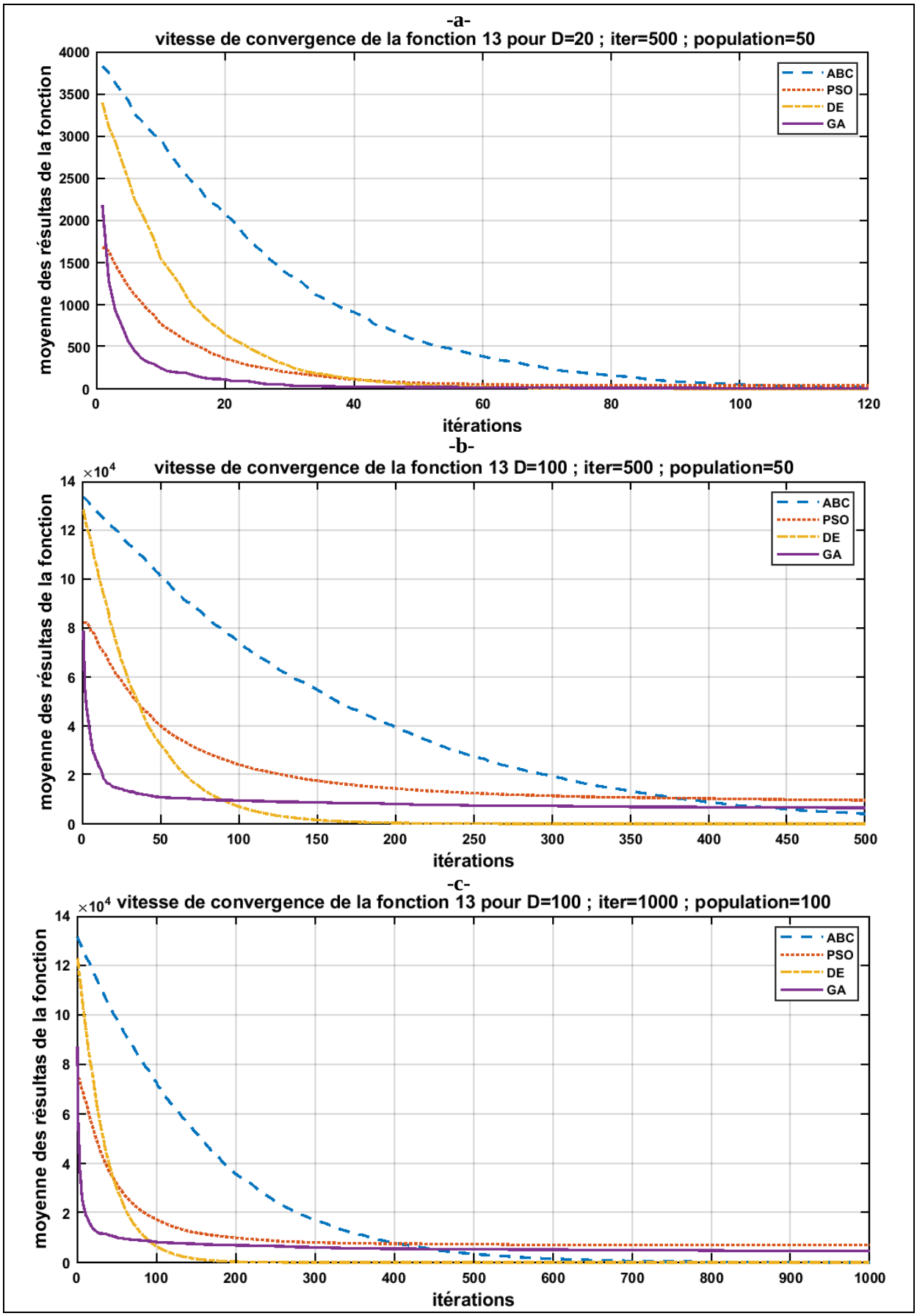

Figure (3.5) : Vitesse de convergence de la fonction 13 pour différents scénarios (a ; b ; c)

On observe, d'après les graphes des figures (3.2), (3.3), (3.4) et (3.5), que le GA a la plus rapide convergence des quatre algorithmes. Celle-ci se fait en deux temps, une convergence très rapide en premier suivit d'une stagnation et d'une vitesse pratiquement fixe. On cite la fonction f_1 scenario -a-, la fonction converge très rapidement sur l'intervalle d'itérations [0 ; 2.4] puis est quasi constant sur le reste. Le GA donnera donc de meilleurs résultats comparés aux autres algorithmes pour un nombre d'itérations restreint.

Dans le scénario -a- de chaque figure pour une dimension plus ou moins faible (D=20), le GA parvient à donner des résultats acceptables pour chaque fonction comparer à quand la dimension est très grande (D=100), Le GA converge prématurément et reste stagné dans des solutions sous-optimales, on cite le scénario -b- de la fonction f_1 où le GA reste fixe dans des valeurs approximant les 10^4.

Le manque d'amélioration entre le scénario -b- (itérations=500) et -c- (itérations=1000) est aussi très clair dans toutes les figures, donc l'incrémentation du nombre d'itérations ou de la population à but d'améliorer la solution est inutile.

Contrairement au GA, le reste des algorithmes ont une convergence assez stable avec une allure plus ou moins similaire l'une de l'autre. On note quand même une différence de vitesse, ou le DE est le plus rapide des trois et surpasse même le GA quand ce dernier est stagné. La qualité de sa convergence et aussi très visible comparé aux autres. Il est donc clair que pour les problèmes à plusieurs dimensions le DE est le seul à donner de bons résultats.

Quant à l'ABC et le PSO, l'ABC et généralement plus lent dans le scénario -a- par contre à partir du scénario -b-, bien que le PSO est plus rapide que lui au début, les aspects s'inversent après un certain nombre d'itérations où le PSO finit par décélérer avec l'ABC qui continue au même rythme.

Les graphes illustrent bien l'amélioration continuelle de l'ABC avec le nombre d'itérations, ainsi que les cas de stagnation du PSO après l'atteinte d'une itération limite. La figure (3, 3) est un exemple, vue qu'après l'atteinte de 100 itérations l'algorithme cesse de s'améliorer.

L'ABC donne donc les seconds meilleurs résultats pour ce cas d'étude, mais nécessite un nombre très élevé d'itérations, on remarque dans la figure (3,3) -c- après 1000 itérations, l'algorithme est encore bien loin de l'optimum. Ceci étant la hausse du nombre d'itérations et de la population bien que majoritairement utile pour l'amélioration des solutions, reste parfois contraignante quand elle est trop importante.

3.4.5.2.Temps d'exécution :

On finalise la comparaison de nos algorithmes par la mesure du temps d'exécution de chacun. On choisit pour ce test la fonction f_3

On obtient les résultats du tableau suivant :

Les cas	Scenarios	D	Np	iter	GA (ms)	ABC (ms)	PSO (ms)	DE (ms)
Cas standard	1	20	50	500	7,70E+02	**2,17E+02**	2,31E+02	4,51E+02
Variation de la dimension	2	30	50	500	1,08E+03	**2,55E+02**	3,15E+02	5,95E+02
	3	50	50	500	2,55E+03	**3,67E+02**	5,30E+02	1,58E+03
	4	100	50	500	6,50E+03	**9,14E+02**	1,11E+03	1,92E+03
Variation de la taille de la population	5	20	30	500	4,32E+02	**1,34E+02**	1,41E+02	2,78E+02
	6	20	80	500	1,12E+03	**3,54E+02**	3,69E+02	8,12E+02
	7	20	100	500	2,67E+03	**4,37E+02**	4,68E+02	9,11E+02
Variation du nombre d'itérations	8	20	50	1000	1,35E+03	**4,15E+02**	4,38E+02	8,75E+02
	9	20	50	2000	5,79E+03	**1,72E+03**	9,18E+02	1,92E+03
	10	20	50	3000	8,42E+03	**1,53E+03**	2,63E+03	5,62E+03
Variation de Np et iter	11	20	80	1000	4,45E+03	1,25E+03	**7,52E+02**	1,57E+03

Tableau (3.9) : Temps d'exécution de chaque algorithme pour la fonction 3 en millisecondes

Le tableau représente le temps moyen de cinquante exécutions de chaque algorithme en millisecondes prenant l'exemple de la fonction f_3 sous 11 scenarios bien choisis pour comprendre l'effet de la complexité des paramètres d'optimisation sur le temps d'exécutions de notre quatre algorithmes méta-heuristiques.

Tout d'abord, on remarque une forte augmentation du temps d'exécution avec la hausse des paramètres (itérations, population et dimension) due au nombre important de ressources de calcul consommés.

Ce tableau montre que l'ABC est plus rapide dans la grande majorité des scenarios avec un temps d'exécution minimal de : **1,34E+02 millisecondes** donc 0,1 secondes, suivie par le PSO qui est plus rapide que le DE dans tous les scenarios et c'est le meilleur dans le scenario 11 où on a augmenté le nombre d'itérations et la taille de la population en même temps.

On remarque aussi que le GA est le plus lent dans tous les scenarios avec un maximum de temps d'exécution dans le scenario 10 : 8,42E+03 ms donc 8 seconds de plus par rapport à l'ABC qui ne dure que **1,53E+03 ms** seulement dans le même scenario.

On constate donc que : l'augmentation des paramètres d'optimisation tel que le nombre d'itérations et la taille de la population bien qu'essentiel pour l'obtention d'une meilleure qualité de solutions se fait au détriment du temps d'exécution, chose qui est indésirable durant la résolution de nos problèmes. Un compromis doit donc être établie de façon à obtenir des résultats acceptables dans les meilleurs délais.

3.5. Conclusion :

Ce chapitre se porte sur l'effet des paramètres d'optimisation sur le comportement des algorithmes méta-heuristiques ainsi qu'une étude comparative des algorithmes GA, DE, PSO et ABC. L'analyse s'effectue en plusieurs tests sur des fonctions benchmark dont les résultats font l'objet d'une étude statistique.

On constate donc l'influence de la taille de la population et du nombre d'itération sur l'amélioration de la solution et celle de la hausse de la dimension sur la baisse de celle-ci. On remarque aussi l'effet du paramètre de mutation sur le GA : trop petit ou trop grand l'algorithme converge prématurément.

La comparaison entre algorithme démontre la supériorité du DE par rapport aux autres programmes, sachant qu'il obtient les meilleures solutions sur le plus de fonctions et a de loin la meilleure convergence lors de la hausse de la dimension. L'ABC le suit avec les seconds meilleurs résultats ainsi que le meilleur temps d'exécution.

Conclusion générale

Les travaux présentés dans cette étude portent sur la résolution de problème d'optimisation difficile (NP-hard) par les méthodes de résolution approchées (approximatives) dites méta-heuristiques. On traite donc l'effet de quelques paramètres d'optimisation (nombre d'itération, taille de population, dimension, pourcentage de mutation) sur quatre algorithmes à savoir le GA, le DE, le PSO et l'ABC afin de cerner la meilleure configuration permettant l'obtention de solutions de bonne qualité en un temps raisonnable. Les tests se font sur quinze fonctions benchmark, dont les résultats comprennent différentes variables statistiques dont la moyenne des couts, la vitesse de convergence et le temps d'exécution.

Les résultats obtenus ont montré que la hausse du nombre d'itération ainsi que la taille de population améliorent généralement la qualité de la solution mais souvent au détriment de la vitesse d'exécution de l'algorithme. Ce principe ne s'applique pas au GA qui d'après nos observations sur la vitesse de convergence, s'améliore pour un nombre limité d'itération avant de stagner, il est donc meilleur pour des problèmes nécessitant un nombre restreint d'itération.

L'augmentation de la dimension par contre résulte toujours en la détérioration de la qualité de la solution et à un ralentissement de l'algorithme. Pour les problèmes à plusieurs grandeurs, on privilégiera donc le plus performent de nos algorithmes à savoir le DE. Bien que pour des dimension minime l'ABC et le GA peuvent être plus adapté pour certaines fonctions, le DE l'emporte toujours quand celle-ci augmente. Si on privilégie la minimisation du temps d'exécution, on préfèrera l'ABC, le plus rapide de nos algorithmes

Concernant le pourcentage de mutation du GA, il est clair que les pourcentages de 10% et 20% sont les meilleur pour la plupart des cas, ajustant le tôt d'exploration et d'exploitation de l'algorithme

Les paramètres d'optimisation influencent grandement les performances des algorithmes méta-heuristiques, le choit de leur configuration dépend de l'algorithme choisit et du problème à résoudre.

Les méta-heuristiques comportent plusieurs autres questions qu'on vise traiter dans nos prochaines études tel que la réalisation d'autres algorithmes (SA, TS, …) , l'étude des caractéristiques des fonctions pour mieux comprendre leur affinité pour certains algorithmes, l'étude des mécanismes d'un algorithme en détail (exploration, exploitation, …) ainsi que l'application des méta-heuristiques dans des problèmes réels (MPPT pour partial shading , …).

Références :

[1] R.S. Anderssen, Global optimization, ´edit´e par R.S. Anderssen, L.S. Jennings et D.M. Ryan. Optimization, Univ. of Queensland Press, St Lucia, 1972.

[2] E.-G. Talbi, Metaheuristics: from design to implementation vol. 74: John Wiley & Sons, 2009.

[3] M. Bierlaire, Introduction à l'optimisation différentiable: PPUR presses polytechniques, 2006.

[4] H. Alice Yalaoui, FaroukYalaoui et Lionel Amodeo, "méthodes et techniques " , Université de Technologie de Troyes,2012.

[5] J.-B. Hiriat-Urruty, Les mathématiques du mieux faire. *Vol. 1 : Premiers pas en optimisation.* Collection Opuscules, Editions ELLIPSES (décembre 2007), 144 pages

[6] C. Solnon, "Contributions à la résolution pratique de problèmes combinatoires," Université Lyon 1, 2005

[7] C. Karaali et Z. Sellam, Master 2 en énergies renouvelables option conversion photovoltaïque "Etude comparative entre les algorithmes Méta-heuristiques", Université de Blida 1, 2022.

[8] S. B. Mena, "Introduction aux méthodes multicritères d'aide à la décision," Unité de Mathématique. Faculté universitaire des Sciences agronomiques de Gembloux BASE, 2000.

[9] Berthiau, G., & Siarry, P. (2001). État de l'art des méthodes "d'optimisation globale". *RAIRO - Operations Research*

[10] E. Matagne, "Cours elec 2311: Physique interne des convertisseurs électromécaniques. Milieux magnétiques composites. Notes de cours de l'Université Catholique de Louvains, Belgique".

[11] J. Puchinger and G. R. Raidl, "Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification," in International workconference on the interplay between natural and artificial computation, 2005, pp. 41- 53.

[12] R. Korf, Depth-first iterative-deepening: An optimal admissible tree search, Artificial Intelligence, 27(1):97–109, 1985.

[13] S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. Prentice-Hall, Third Edition, 1995.

[14] R. Bellman. Dynamic Programming. Princeton University Press, Princeton, NJ, 1957.

[15] A. H. Land and A. G. Doig, "An automatic method for solving discrete programming problems," in 50 Years of Integer Programming 1958-2008, ed: Springer, 2010, pp. 105-132.

[16] J. W. S. Emile Aarts and Jan K. Lenstra, Local Search in Combinatorial Optimization. New York, NY, USA, 1997.

[17] J.-K. Hao, P. Galinier, and M. Habib, "Méta-heuristiques pour l'optimisation combinatoire et l'affectation sous contraintes," Revue d'intelligence artificielle, vol. 13, pp. 283-324, 1999.

[18] N. Schabanel, Algorithmes d'approximation et Algorithmes randomisés, CIRM , 2003.

[19] D. S. Hochbaum. Approximation Algorithms for NP-Hard Problems. International Thomson Publishing, 1996.

[20] A. A. T.A.J. Nicholson, "Optimization in Industry : Industrial applications.," ed. London Business School series., 1971.

[21] B. BELIN, "Conception interactive d'environnement urbains durables à base de résolution de contraintes," Thèse de Doctorat, École doctorale : Sciences et technologies 68 de l'information, et mathématiques, l'Université de Nantes Angers Le Mans, france, 2014

[22] : P. Augerat, J. M. Belenguer, E. Benavent, A. Corberan, and D. Naddef. Separating capacity constraints in the CVRP using tabu search. European Journal of Operational Research, 106(2):546–557, 1998.

[23] P. R. Cohen. Empirical Methods for Artificial Intelligence. MIT Press, Cambridge, 1995.

[24]: J. P. Cohoon, W. N. Martin, and D. S. Richards. Genetic algorithms and punctuated equilibria. In H.-P. Schwefel and R. Manner, editors, Parallel Problem Solving from Nature, Dortmund, Germany, Oct 1990, LNCS Vol. 496. Springer, 1990, pp. 134–141.

[25] : A. Djihen, H.Abir .Commande MPPT Pour les systémes photovoltaiques en utilisant l'optimisation par essaim des particules PSO. Université SaâdDahlab, Blida-1. Faculté de Technologie. 2021

[26]: S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. Science, 220(4598):671–680, 1983.

[27]: V. Cerny. A thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of Optimization Theory and Applications, 45:41–51, 1985.

[28]: A. Dekkers and E. Aarts. Global optimization and simulated annealing. Mathematical Programming, 50:367–393, 1991

[29]: M. Locatelli. Simulated annealing algorithms for continuous global optimization: Convergence conditions. Journal of Optimization Theory and Applications, 29(1):87–102, 2000.

[30]: L. Ozdamar and M. Demirhan. Experiments with new stochastic global optimization search techniques. Computers and Operations Research, 27(9):841–865, 2000.

[31]: N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. Journal of Chemical Physics, 21:1087–1092, 1953.

[32]: F. Glover. Tabu search: Part I. ORSA Journal on Computing, 1(3):190–206, 1989.

[33]: J. H. Holland. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI, 1975

[34]: D. E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 1989.

[35]: K. A. De Jong. Genetic algorithms: A 10 year perspective. In International Conference on Genetic Algorithms, 1985, pp. 169–177.

[36] O. Ait Sahed. FUZZY PREDICTIVE CONTROL USING META-HEURISTIC ALGORITHMS.thése de Doctorat. Université de Blida 1. 2015

[37]: Haupt, R.L. and S.E. Haupt, Practical Genetic Algorithms. 2004: John Wiley & Sons, Inc

[38]: Paul CH Chu. A genetic algorithm approach for combinatorial optimisation problems. 1997.

[39]: Agoston E Eiben, James E Smith, et al. Introduction to evolutionary computing, volume 53. Springer, 2003.

[40]: Reeves, C.R., Genetic Algorithms, in Handbook of Metaheuristics, M. Gendreau and J.-Y. Potvin, Editors. 2010, Springer US. p. 109-139

[41]: K. V. Price, R. M. Storn, and J. A. Lampinen. Differential evolution: A Practical Approach to Global Optimization. Springer, 2006.

[42]: J. Kennedy and R. C. Eberhart. Swarm Intelligence. Morgan Kaufmann, San Francisco, CA, 2001.

[43] Mahamad Nabab Alam. Particle Swarm Optimization: Algorithm and its Codes in MATLAB. Department of Electrical Engineering, Indian Institute of Technology, Roorkee-247667, India.2016

[44] 119. Karaboga, D., An idea based on honey bee swarm for numerical optimization. 2005, Erciyes University: Kayseri/Turkey.