Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche

researchportal.unamur.be

University of Namur

THESIS / THÈSE

MASTER EN SCIENCES MATHÉMATIQUES À FINALITÉ SPÉCIALISÉE EN DATA SCIENCE

Amélioration des algorithmes d'optimisation par essaim de particules

TONELLI, Auban

Award date:
2022

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 29. Apr. 2024

Université de Namur

Faculté des Sciences

Amélioration des algorithmes d'optimisation par essaim de particules

Mémoire présenté pour l'obtention du grade académique de master en Sciences Mathématiques, à finalité didactique

Auban Tonelli

Promoteur
Pr. Timoteo Carletti

16 août 2022

Remerciements

Ce mémoire est la finalité de 5 années d'études que je n'aurais sans doute pas réussi à traverser sans le soutien de mes différentes sphères relationnelles.

Je tiens donc tout d'abord à remercier Monsieur Carletti, mon promoteur, pour sa disponibilité, ses conseils, ses suggestions et ses remarques avisées lors de l'écriture de ce mémoire.

Je tiens ensuite à remercier l'ensemble du corps enseignant et assistant de la faculté de mathématique de l'UNamur. Chacune et chacun m'aura, d'une certaine façon, permis de grandir et de m'épanouir pendant mes études. Je tiens à remercier plus particulièrement Monsieur Füzfa qui, par son rôle de sensei durant les cours de judo de l'université, a su me transmettre des valeurs de travail et de persévérance qui m'ont aidées à avancer.

Merci à mes parents, à mon frère et à mes 4 soeurs qui ont toujours été là pour me soutenir quand j'en avais besoin.

Merci enfin à mes amis de l'université et à mes coéquipiers du tennis de table pour leur soutien et les moments de détente qu'ils ont pu m'offrir.

Résumé

Le sujet central de ce mémoire est un algorithme d'optimisation faisant partie de la classe des méthodes heuristiques : l'algorithme d'optimisation par essaim de particules (algorithme PSO). Dans cet algorithme, les particules évoluent au moyen d'une mise à jour de leur vitesse et position influencées par leur direction courante, leur meilleure position connue ainsi que la meilleure position connue dans leur voisinage.

L'objectif de ce mémoire est la recherche d'améliorations de l'algorithme PSO. Pour cela, nous n'avons pas considéré les potentielles améliorations du voisinage des particules mais nous avons développé des méthodes centrées sur la mise à jour des vitesses et positions. Les méthodes développées sont de deux types. L'une des méthodes a pour principe de choisir plus intelligemment la mise à jour des vitesses en se basant sur l'algorithme du recuit simulé. Les autres méthodes sont basées sur une optimisation de la taille du vecteur vitesse utilisé pour la mise à jour des positions.

Après avoir développé nos nouvelles méthodes, nous les avons testées sur un ensemble de fonctions de test pour les comparer aux versions classiques de l'algorithme PSO. Ces tests nous ont alors permis de déterminer que l'optimisation de la taille du vecteur vitesse ne permet pas d'améliorer l'algorithme PSO. Cependant, nous avons aussi pu voir que le choix des vitesses utilisant un recuit simulé est une bonne option d'amélioration de l'algorithme PSO.

Mots clés : Essaim de particules / Heuristique / Optimisation / Profil de performance

Abstract

The subject of this master thesis is an optimization algorithm. The latest is a heuristic method named "Particle swarn optimization" (PSO algorithm). This algorithm works as follows : The particules evolve according to two factors : their speed and position. Those are updated taking into account their current direction, their best-known position and the best-known position in their neighbourhood.

The purpose of this thesis is to upgrade the PSO algorithm. We did not consider working on a different version of the neighbourhood of the particles. However, we developped other updates of speed and positions. We implemented two different types of methods : the first one aims to choose a wiser upgrade of the speed using the simulated annealing algorithm. The other aim to optimize the length of the speed vector used to update the positions.

Once we implemented our new methods, we tested them on a set of functions to compare them to the usual versions of the PSO algorithm. Those tests enabled us to conclude that optimizing the length of the speed vector doesn't improve the PSO algorithm. However, we could also notice that chosing the speed using a simulated annealing algorithm is a potential good upgrade to the PSO algorithm.

Keywords : Particle swarn / heuristic / optimization / Performance profile

Table des matières

Introduction **1**

1 Les méthodes heuristiques pour l'optimisation **2**
1.1 L'optimisation . . . 2
1.2 L'optimisation "difficile" . . . 3
1.3 Les méthodes heuristiques . . . 4
1.3.1 Les algorithmes génétiques . . . 4
1.3.2 Le recuit simulé . . . 5
1.3.3 Critères de sélection d'un algorithme . . . 6

2 L'algorithme PSO **7**
2.1 L'algorithme PSO standard . . . 7
2.1.1 Description de l'algorithme . . . 8
2.1.2 Notations et description formelle de l'algorithme . . . 9
2.1.3 Initialisation de l'essaim . . . 9
2.1.4 Mise à jour des vitesses et des positions . . . 11
2.1.5 La condition d'arrêt . . . 12
2.1.6 Test de l'algorithme PSO standard . . . 13
2.2 les modifications de l'algorithme . . . 14
2.2.1 Le paramètre V_{max} . . . 14
2.2.2 Stratégies de confinement . . . 15
2.2.3 Le coefficient de constriction . . . 15
2.2.4 Le coefficient d'inertie . . . 16
2.3 Le voisinage . . . 17
2.3.1 Les topologies statiques . . . 17
2.3.2 La Topologie aléatoire adaptative . . . 20
2.3.3 Vers une évolution de l'algorithme . . . 20

3 Modifications de l'algorithme PSO **22**
3.1 Recuit simulé dans la mise à jour des vitesses et positions . . . 22
3.2 Recherche locale dans la mise à jour des vitesses . . . 25
3.2.1 Méthode de recherche tabou . . . 26
3.2.2 Recherche linéaire exacte . . . 26

4 Cas-tests utilisés **28**

5 Outils d'analyse des performances **34**
5.1 Profils de performance . . . 35
5.2 Analyse par cas-test . . . 37

6 Étude numérique des algorithmes **38**
6.1 Algorithme PSO utilisé à titre comparatif 38
6.2 Algorithme inspiré du recuit simulé 42
6.3 Algorithme avec recherche tabou 45
6.4 Algorithme avec Recherche linéaire exacte 48

7 Comparaison des différentes options et résultats **50**

Conclusion et perspectives **54**

Bibliographie **56**

Annexe **58**

Introduction

. Dans notre société actuelle, il arrive régulièrement que nous soyons confrontés à des problèmes pouvant être modélisés mathématiquement par des problèmes d'optimisation. En effet, il n'est pas rare de se retrouver dans une situation où nous voulons acheter un ensemble d'objets à moindre prix et il est encore moins rare d'essayer de choisir le chemin le plus court pour se rendre quelque part. Tous les problèmes de ce type peuvent alors être résolus à l'aide d'optimisation. Pour cela, les problèmes sont représentés par une fonction qu'il nous faut alors minimiser ou maximiser.

Une façon courante de résoudre ces problèmes est l'utilisation de méthodes dites exactes. Ces méthodes sont toutes des méthodes qui ont été pensées pour fournir à coup sûr une solution exacte aux problèmes d'optimisation considérés. Cependant, il arrive que les problèmes à traiter ne possèdent pas de formulation précise ou qu'ils soient trop compliqués à traiter, ce qui met en échec les méthodes exactes.

Pour palier à cela, il existe d'autres méthodes qui peuvent être utilisées. Ces méthodes sont basées sur un principe itératif, souvent inspiré d'un phénomène naturel, qui va permettre, sans certitude de trouver la solution exacte, de se rapprocher pas à pas d'une solution adaptée au problème. Ces méthodes sont appelées méthodes heuristiques.

Ce sont ces méthodes heuristiques qui vont faire l'objet de ce mémoire. En particulier, dans ce mémoire, nous tenterons d'améliorer l'une d'entre elles : l'algorithme d'optimisation par essaim de particules.

Pour ce faire, dans le chapitre 1, nous commencerons ce mémoire en définissant plus en détail les problèmes d'optimisation que nous considérerons, ainsi que la classe des méthodes heuristiques. Dans le chapitre 2, nous ferons alors une présentation détaillée de l'algorithme d'optimisation par essaim de particules. Nous en présenterons la version classique ainsi que des variantes souvent utilisées. Nous poursuivrons alors, dans le chapitre 3, avec une présentation des différentes améliorations que nous envisageons pour l'algorithme.

La fin de ce mémoire sera consacrée à une étude numérique des algorithmes développés. Pour cela, nous commencerons, dans le chapitre 4, par une présentation de l'ensemble des fonctions de test qui nous ont permis d'utiliser nos algorithmes. Le chapitre 5 sera ensuite consacré à une définition des outils qui nous ont permis de tester, de manière globale ainsi que sur chaque fonction de test prise individuellement, nos algorithmes.

Le chapitre 6, quant à lui, nous permettra de définir les meilleures versions de chacune des améliorations envisagées pour l'algorithme. Nous utiliserons les outils développés dans le chapitre 5 pour comparer les variantes de chaque amélioration entre elles et ainsi faire émerger des candidats à une comparaison finale. Nous terminerons enfin avec le chapitre 7 dans lequel nous comparerons tous les candidats retenus avec une version classique de l'algorithme PSO. Nous pourrons ainsi déterminer si les méthodes développées dans ce mémoire peuvent être retenues comme des améliorations de l'algorithme PSO.

Chapitre 1

Les méthodes heuristiques pour l'optimisation

Dans ce premier chapitre, nous définirons la notion de problème d'optimisation, nous présenterons la classe des méthodes heuristiques permettant d'offrir des solution à ces problèmes et nous illustrerons ces méthodes au moyen des algorithmes génétiques et de l'algorithme du recuit simulé.

1.1 L'optimisation

De nos jours, de nombreux ingénieurs et décideurs doivent faire face à divers problèmes dont la complexité ne cesse de croître. Pour la pluspart, ces problèmes peuvent se formaliser au moyen de problèmes d'optimisation.

Ces problèmes d'optimisation sont généralement caractérisés par une fonction objectif f qu'il faut maximiser ou minimiser suivant un ensemble de contraintes d'égalité ou d'inégalité. Cette fonction objectif s'exprime à l'aide d'un nombre fixe de variables existantes sur des domaines définis par le problème. L'ensemble de toutes les solutions possibles du problème d'optimisation est appelé espace de recherche. Cet espace possède une dimension égale au nombre de variables du problème. Ainsi, les problèmes d'optimisation sont entièrement définis par leur fonction objectif, leur espace de recherche et leurs contraintes.

1.1.0.1 Définition du problème d'optimisation

Nous considérons un problème d'optimisation mono objectif, c'est à dire un problème ne possédant qu'une fonction objectif, défini sur un domaine continu. Le problème est défini par les informations suivantes :

- R, un sous espace compact de $\mathbb{R}^n$ où n est la dimension du problème. Le compact R est notre espace de recherche, il est délimité par un hypercube de vecteurs de positions minimale et maximale min_R et max_R.

- $\Omega = C_e \bigcup C_i$ où C_e est un ensemble de contraintes d'égalité et C_i est un ensemble de contraintes d'inégalité. Toute solution existant dans R et vérifiant les contraintes de Ω fait partie de l'ensemble admissible. Toute solution n'appartenant pas à l'ensemble admissible devra être rejetée.

- f la fonction objectif du problème.

Ces trois informations constituent notre problème d'optimisation et sont reprises par le triplet $O = (R, \Omega, f)$. La définition formelle de la fonction objectif qui en découle est

$$\begin{array}{cccc} f & : & R \subset \mathbb{R}^n & \rightarrow & \mathbb{R} \\ & & x & \mapsto & f(x). \end{array}$$

Dans le cas de la recherche d'un minimum global de f, si nous supposons que ce minimum x^* existe, il devra vérifier

$$\forall\ c_e \in C_e,\ \forall\ c_i \in C_i,\ \forall\ x \in \mathbb{R}$$
$$\begin{cases} f(x^*) \ < \ f(x) \\ c_e(x^*) = 0 \\ c_i(x^*) \leq 0 \end{cases}$$

En d'autres termes,

$$\begin{aligned} x^* &= \underset{x \in \mathbb{R}^n}{argmin}(f(x)) \\ s.c\ c_e(x) &= 0,\ \forall\ c_e \in C_e \\ c_i(x) &\leq 0,\ \forall\ c_i \in C_i \end{aligned}$$

1.2 L'optimisation "difficile"

Pour certains problèmes vérifiant des caractéristiques comme la convexité, la continuité ou la dérivabilité, il existe des méthodes d'optimisation globale, dont la convergence a été démontrée, permettant d'obtenir une solution exacte du problème en un temps "acceptable" [2]. Mais pour d'autres problèmes, il est possible que f ne soit pas strictement convexe ou bien que f ne soit pas continue ou dérivable. Il arrive également que la fonction objectif soit perturbée par un bruit ou que le calcul de sa dérivée ne soit pas possible. Par exemple, il n'est pas possible de dériver une fonction notant les chemins d'un réseau. Ce second type de problème d'optimisation fait partie de la classe des problèmes d'optimisation difficile.

Selon [6], la classe des problèmes d'optimisation difficile est mal définie et pour mieux l'appréhender, il est nécessaire de faire la distinction entre deux types de problèmes d'optimisation : les problèmes d'optimisation "discrète" et les problèmes d'optimisation "continue".

Dans le groupe des problèmes d'optimisation à variables discrètes, nous retrouvons par exemple le problème bien connu dit du voyageur de commerce. Pour ce problème, il faut déterminer le chemin minimal que doit emprunter un "voyageur de commerce" de sorte qu'il passe une et une seule fois par toutes ses différentes destinations en retournant finalement à son point de départ. Pour les problèmes d'optimisation à variables continues, l'exemple fourni dans [6] est le problème de la recherche des valeurs à affecter aux paramètres d'un modèle numérique de processus, de sorte que le modèle reproduise un comportement réel observé.

Parmi les problèmes d'optimisation discrète et continue, deux catégories de problèmes peuvent être qualifiés de problèmes d'optimisation difficile [6]. La première catégorie reprend les problèmes d'optimisation discrète qu'aucun algorithme exact polynomial ne permet de

résoudre. Cette catégorie reprend en particulier les problèmes "NP-difficiles". La seconde catégorie reprend les problèmes d'optimisation à variables continues pour lesquels aucun algorithme ne permet de trouver un optimum global en un nombre fini de calcul.

Dans le but d'obtenir une solution aux problèmes d'optimisation difficile, le domaine de l'optimisation a vu émerger une classe d'optimisation stochastique : les méthodes heuristiques.

1.3 Les méthodes heuristiques

Comme nous l'avons évoqué précédemment, en optimisation, il est possible que le calcul de la dérivée de la fonction objectif considérée soit très compliqué voire impossible. Il arrive également que, pour certaines fonctions objectif, des solutions exactes existent mais que le problème soit si compliqué qu'il n'y aurait pas assez d'une vie pour trouver cette solution.

Les méthodes heuristiques sont des méthodes utilisées comme solution aux problèmes d'optimisation difficile. Il s'agit de méthodes d'optimisation globale, ne recourant pas aux calcul du gradient de la fonction objectif, qui permettent de trouver des solutions acceptables en un temps raisonnable. A l'inverse des méthodes d'optimisation classiques, aucun théorème ne garantit l'optimalité des solutions trouvées. En revanche, pour trouver ces solutions optimales, les méthodes classiques nécessitent des conditions initiales suffisamment proches d'un extremum pour pouvoir espérer l'atteindre. Les méthodes heuristiques quant à elles, ne souffrent pas de cette contrainte et permettent de trouver des extremums locaux pour des conditions initiales quelconques. De plus, les méthodes heuristiques peuvent être lancées plusieurs fois sur un même problème avec des conditions initiales différentes pour obtenir une liste de diverses solutions.

Une autre particularité des méthodes heuristique est la source de leur inspiration. En effet ces méthodes trouvent généralement leur inspiration dans le monde réel. Pour illustrer cela, nous allons présenter brièvement les algorithmes génétiques et l'algorithme du recuit simulé.

1.3.1 Les algorithmes génétiques

La classe des algorithmes génétiques présentée dans [3] trouve son inspiration dans l'évolution darwinienne. Ces algorithmes itèrent sur plusieurs générations d'individus, représentant chacun une solution potentielle, en faisant évoluer ce groupe au moyen d'une sélection naturelle, de croisements et de mutations.

Dans cet algorithme, un individu représente une solution potentielle du problème d'optimisation décrit précédemment. De ce fait, une génération d'individu est donc constituée de plusieurs solutions potentielles. Au cours de l'algorithme, la population initiale,ou première génération, va évoluer à l'aide de 3 mécanismes : la sélection naturelle, qui consiste en la conservation des meilleurs individu de la population, les croisement, qui ont pour principe de combiner plusieurs individu entre eux pour donner naissance à de nouvelles solutions potentielles ainsi que les mutations, qui sont des modifications aléatoires de certains individus permettant plus d'exploration.

Les trois mécanismes que nous venons de décrire vont permettre à la population d'évoluer vers la solution du problème d'optimisation en générant des individus de plus en plus adaptés. Pour un nombre de générations suffisant, la population verra généralement émerger un individu très bien adapté au problème d'optimisation qui fera office de solution. La figure 1.1 illustre le fonctionnement basique de ces algorithmes.

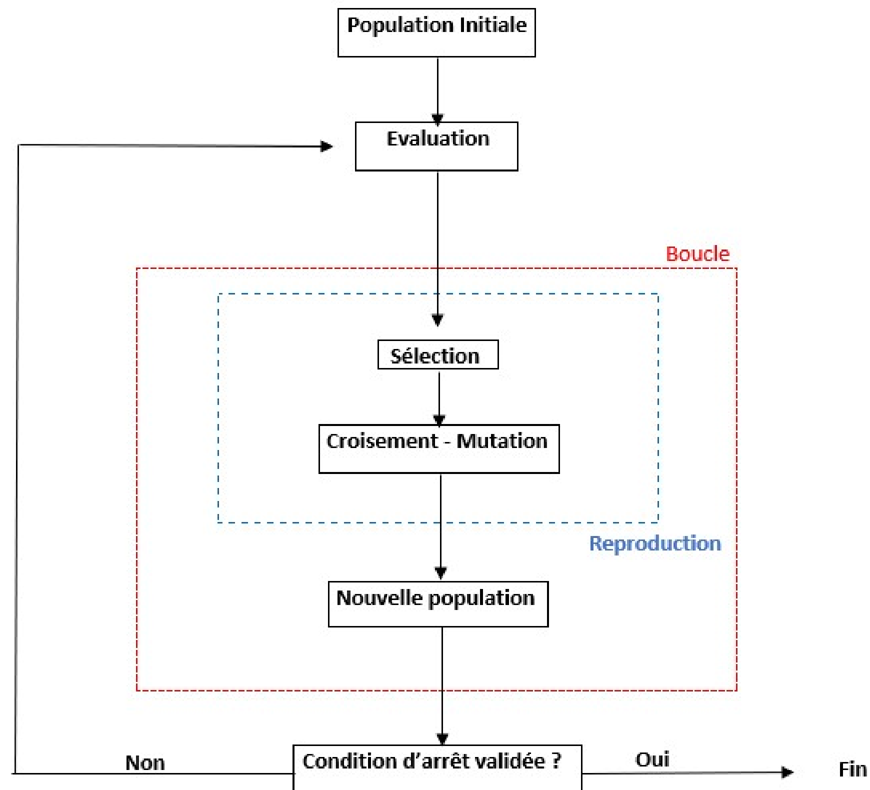

FIGURE 1.1 – Algorithme génétique de base.

1.3.2 Le recuit simulé

L'algorithme du recuit simulé est un algorithme développé par Kirkpatrick et al [19], qui s'inspire directement d'un processus de métallurgie nommé recuit. En métallurgie, le recuit est le fait de chauffer un métal à des températures élevées avant de le refroidir lentement. Il est utilisé pour rendre les matériaux plus malléables.

Pour faire évoluer sa solution, l'algorithme du recuit simulé va utiliser l'algorithme de Metropolis [22] qui consiste en une règle de validation d'une évaluation de la fonction objectif f en un nouveau point de l'espace de recherche. Le principe de cette règle de sélection est de valider et conserver le nouveau point de l'espace de recherche si il fait mieux que le précédent et de tout de même l'accepter, suivant une certaine probabilité, si il fait moins bien. Le fait d'accepter un "mauvais" point de l'espace de recherche permet dans les faits de favoriser l'exploration de l'espace de recherche. L'algorithme est représenté en pseudo code à la figure 1.2.
Un élément remarquable de l'algorithme de Metropolis est la probabilité de sélection d'un mauvais candidat. A l'itération i, cette probabilité de sélection vaut $e^{\frac{f(x_i)-f(x_{i-1})}{T}}$ et est directement dépendante de la température. Dans l'algorithme du recuit simulé, la température T décroît à mesure que l'algorithme progresse. Cela signifie que la probabilité de sélection d'un candidat non optimal va être de plus en plus faible à mesure que l'algorithme se rapproche de la solution. L'utilisation de cette probabilité constitue donc un bon compromis entre exploitation et exploration. En effet, en début d'algorithme, la sélection d'un point "moins performant" peut potentiellement permettre de sortir d'un minimum local, l'exploration est privilégiée. Tandis que, en fin d'algorithme, comme la probabilité de sélection d'un moins bon candidat est faible, l'algorithme n'explore pas de nouvelles pistes, l'exploitation

Initialiser un point de départ x_0 et une température T
pour i=1 à n **faire**
  tant que x_i n'est pas accepté **faire**
    si $f(x_i) \leq f(x_{i-1})$ **alors**
      accepter x_i
    fin
    si $f(x_i) > f(x_{i-1})$ **alors**
      accepter x_i avec la probabilité $e^{\frac{f(x_i)-f(x_{i-1})}{T}}$
    fin
  fin
fin

FIGURE 1.2 – Algorithme de Metropolis (illustration tirée de [15]).

est privilégiée.

1.3.3 Critères de sélection d'un algorithme

En pratique, pour un problème d'optimisation donné, il existe une multitude d'algorithmes différents pouvant résoudre ce problème. C'est pourquoi, il est important de poser des critères qui nous permettent d'être objectif dans le choix d'un algorithme. D'abord, il est important que l'algorithme utilisé soit performant, c'est à dire qu'il doit pouvoir trouver la solution du problème avec précision. Ensuite, le temps de calcul est également à prendre en compte. Si un algorithme trouve précisément la solution d'un problème mais que cela lui prend plusieurs années, il ne sera pas très utile. Enfin, nous pouvons aussi considérer un critère de polyvalence. Pour ce critère nous privilégierons des algorithmes qui peuvent fonctionner avec une large gamme de fonctions objectif différentes.

Dans la suite du mémoire, nous allons nous pencher plus en détail sur une autre méthode heuristique : l'optimisation par essaim de particules.

Chapitre 2

L'algorithme PSO

Ce chapitre présentera l'algorithme d'optimisation par essaim de particules (PSO) qui fait l'objet de ce mémoire. Dans un premier temps, nous commencerons par en présenter une version dite standard basée sur [4]. Dans un second temps, nous présenterons certaines variantes intéressantes de l'algorithme PSO standard.

2.1 L'algorithme PSO standard

l'algorithme d'optimisation par essaim de particules (PSO) trouve son inspiration dans des phénomènes tels que les bancs de poissons ou les vols d'oiseaux. Il s'agit d'une méthode d'optimisation globale qui ne va pas toujours trouver l'optimum mais qui fournira une très bonne solution en un temps acceptable. Développé par James Kennedy et Russel Eberhart [9], l'algorithme PSO est directement inspiré du code Boïds que Craig Raynold a mis au point en 1986 [24]. Ce code permet de simuler des vols d'oiseaux au moyen de 3 contraintes sur les agents de l'essaim : la séparation, la cohésion et l'alignement.

Ainsi, à l'intérieur d'un essaim, la séparation empêche les agents de se percuter, la cohésion force les agents à rester proches les uns des autres, de sorte que l'essaim ne se disperse pas et l'alignement conduit les agents à voler dans la même direction. Une représentation de ces trois principes se trouve sur la figure 2.1. De plus, si ces règles sont suffisantes pour la représentation d'un vol d'oiseaux, elles ne le sont cependant pas pour l'optimisation d'un problème. En effet, un essaim représenté par ces trois principes ne poursuit aucun objectif et ne permettra pas de trouver de solutions.

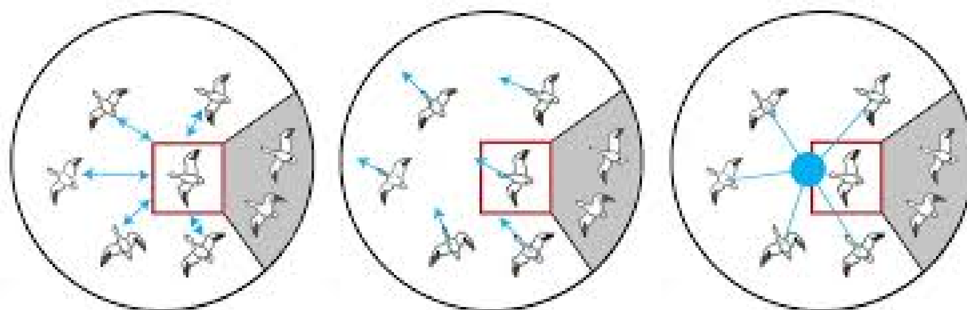

Figure 2.1 – Illustration du code Boïds : les principes de séparation, d'alignement et de cohésion y sont représentés de gauche à droite (illustration tirée de [25]).

Dans les sections suivantes, nous allons détailler le fonctionnement de la PSO. Nous commencerons par en faire une description informelle servant à donner l'intuition du fonctionnement de l'algorithme, suivie d'une description plus rigoureuse.

2.1.1 Description de l'algorithme

Dans cette sous section, nous présentons la version standard de l'algorithme PSO basée sur [4].

Le principe de la PSO est le suivant : Pour un espace de recherche et une fonction de fitness associée, nous recherchons le point de cet espace qui possède la meilleure fitness (la plus élevée dans le cas d'un problème de maximisation et la moins élevée pour un problème de minimisation). Nous recherchons donc l'optimum global de la fonction de fitness. Pour rechercher cet optimum global, la PSO va utiliser des agents appelés "particules" qui vont explorer l'espace de recherche. L'ensemble de toutes ces particules est appelé "essaim" et est caractérisé par une topologie. Cette topologie est un ensemble de liens entre les particules qui permettra de définir quelle particule informe quelle autre particule. Par la suite, pour une particule donnée, nous utiliserons le terme voisinage de la particule pour parler de l'ensemble des particules qui l'informent.

De plus, chaque particule de l'essaim va être caractérisée par les informations suivantes :

- Une position dans l'espace de recherche du problème à optimiser.
- Une valeur de fitness associée à la position, ce qui fait d'une particule une solution potentielle.
- Une vitesse définie sur un sous espace de l'espace de recherche qui sera utilisée à chaque itération pour calculer la nouvelle position de la particule.
- Une mémoire contenant la meilleure position atteinte par la particule au cours de sa vie.
- Une mémoire des informations données par le voisinage de la particule. Grâce à cela, chaque particule va définir la meilleure particule de son voisinage. Il s'agit de la particule du voisinage ayant connu la meilleure valeur de fitness au cours de sa vie. Nous pouvons dénoter la meilleure particule d'un voisinage par le terme de "meilleur ami" .

A l'aide de ces informations, l'algorithme commencera donc par l'initialisation des positions et des vitesses de chaque particule. Après quoi, pour chaque particule, l'algorithme va itérer, de façon séquentielle, particule par particule, sur les instructions suivantes jusqu'à la vérification d'un critère d'arrêt fixé.En premier lieu l'algorithme va évaluer la fitness pour la position courante de la particule, qu'il comparera à la meilleure valeur de fitness atteinte par la particule au cours de sa vie. Si la nouvelle position de la particule fait mieux que la meilleure position en mémoire, l'algorithme assignera la position courante de la particule comme nouvelle "meilleure position". Dans le cas contraire, il conservera la valeur de la "meilleure position". Après cela, l'algorithme réévaluera les positions des particules du voisinage pour redéfinir un nouveau "meilleur ami" si besoin. La boucle se termine par la mise à jour de la vitesse et de la position de la particule considérée.

2.1.2 Notations et description formelle de l'algorithme

Avant toute chose, nous allons ici poser les nouvelles variables et notations que nous utiliserons dans la suite de cette section. Notons que nous reprenons les notations que nous avons déjà utilisées dans les sections précédentes.

- R est notre espace de recherche de dimension D.
- f est la fonction de fitness, nous supposons que nous en cherchons le minimum.
- t est le numéro du pas temporel courant.
- x_k^t est la position de la particule k au temps t, la position possède D coordonnées.
- v_k^t est la vitesse de la particule k au temps t, il s'agit d'un vecteur à D composantes.
- p_k^t est la meilleure position connue de la vie de la particule k au temps t, elle possède D coordonnées.
- t_k^t est la meilleure position trouvée dans le voisinage de la particule k au temps t, elle possède D coordonnées.

Une illustration schématique de l'algotithme PSO utilisant ces notations est visible sur la figure 2.2. Nous pouvons constater sur cette figure que l'algorithme comporte trois étapes majeures que sont l'initialisation des vitesses et des positions des particules, la mise à jour de ces vitesses et de ces positions ainsi que le test du critère d'arrêt de la boucle de l'algorithme. Dans les sections suivantes, nous allons faire la lumière sur chacune de ces étapes.

2.1.3 Initialisation de l'essaim

L'algorithme PSO étant une méthode d'optimisation à population, le choix de la population initiale est crucial pour le bon fonctionnement de l'algorithme. C'est pourquoi il est important de sélectionner un échantillon initial uniformément réparti couvrant au mieux l'espace de recherche tout en respectant les bornes imposées par ce dernier.

Dans la version originale de l'algorithme PSO, les vitesses initiales sont prises aléatoirement pour chaque particule. Cependant, cette option rendait possible le fait d'obtenir des vitesses bien trop élevées et non adaptées au problème considéré, ce qui pouvait empêché sa résolution. L'algorithme a donc évolué.

Une première technique d'initialisation des vitesses, qui a été considérée dans les versions des années 2006 et 2007 de l'algorithme PSO, est donnée par l'équation (2.1) où $U(R_{min}, R_{max})$ est un nombre aléatoire tiré suivant une loi uniforme dans l'intervalle $[R_{min}, R_{max}]$ et où v_k^0 et x_k^0 désignent la vitesse et la position initiale de la particule.

$$\forall k \in [1, D],\ v_k^0 = \frac{U(R_{min}, R_{max}) - x_k^0}{2}. \tag{2.1}$$

Cette première approche qui utilise la taille du domaine n'est pas fondamentalement mauvaise. Cependant, en 2008 , il a été montré que plusieurs particules peuvent immédiatement quitter le domaine si sa dimension est élevée [17]. Ce constat a mené à l'équation (2.2) utilisée dans la version de l'algorithme de 2011. Cette initialisation des vitesses étant meilleure, nous nous baserons sur cette dernière pour la suite.

$$\forall k \in [1, D],\ v_k = U(R_{min} - x_k^0, R_{max} - x_k^0). \tag{2.2}$$

La figure 2.3 illustre les deux possibilités pour l'initialisation des vitesses des particules.

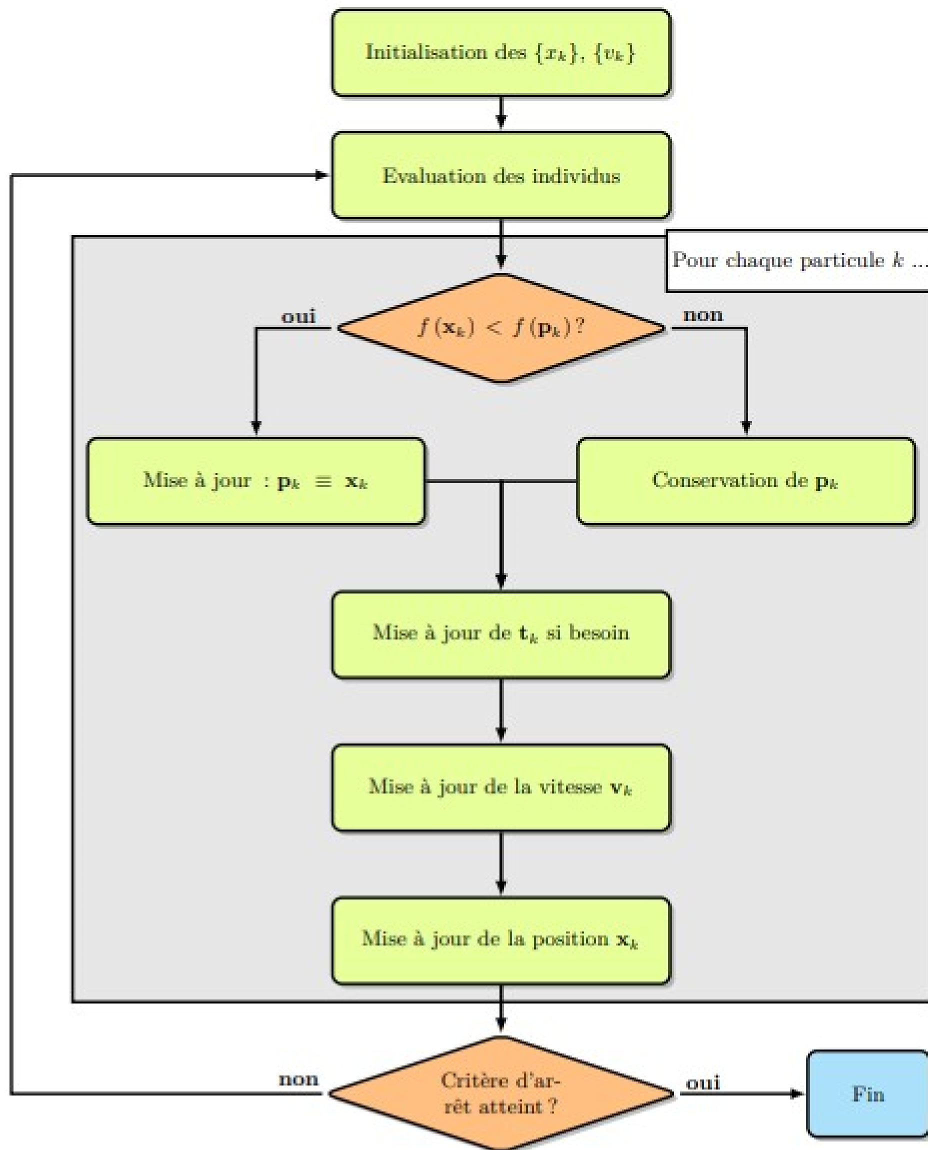

FIGURE 2.2 – Algorithme PSO standard (illustration tirée de [25]).

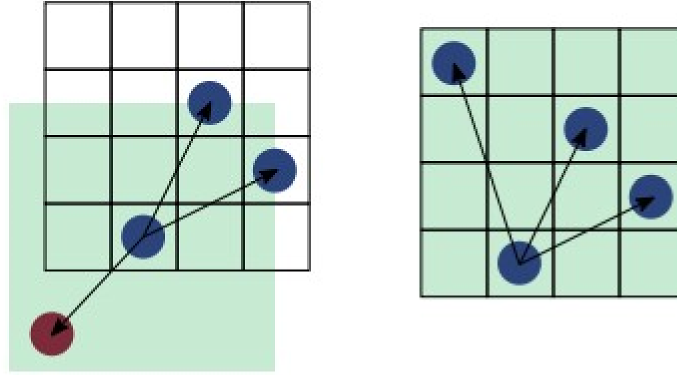

FIGURE 2.3 – Représentations des manières d'initialiser les vitesses des particules. Le carré vert représente l'étendue dans laquelle le vecteur vitesse initiale peut se trouver. Dans la partie de gauche, la vitesse est bornée par les bornes du domaine multipliées par une constante. Dans la partie de droite la position de la particule est prise en compte pour le calcul des bornes de la vitesse. (illustration tirée de [25]).

2.1.4 Mise à jour des vitesses et des positions

L'étape de mise à jour des positions des particules de l'essaim constitue le coeur de la PSO. Dans cette étape, à l'itération t, la position de chaque particule de l'essaim va être mise à jour suivant trois composantes.

- Une composante d'**inertie** qui représente la tendance qu'ont les particules à suivre la direction courante de leur déplacement.

- Une composante **cognitive** qui va représenter le fait que les particules tendent à se diriger vers leur meilleure position connue p_k.

- Une composante **sociale** représentant la tendance qu'ont les particules à se déplacer vers la meilleure position connue de leur voisinage.

L'idée de la mise à jour de la position d'une particule suivant les 3 composantes citées plus haut est représentée à la figure 2.4.

Plus formellement, l'étape de mise à jour des positions s'effectuera au moyen des équations (2.3) et (2.4).

$$x_k^{t+1} = x_k^t + \alpha v_k^t. \tag{2.3}$$

$$v_k^{t+1} = \omega v_k^t + r_1 c_1 (x_k^t - p_k^t) + r_2 c_2 (x_k^t - t_k^t). \tag{2.4}$$

Dans ces deux équations nous pouvons observer différents paramètres qui influenceront la qualité de l'algorithme et qu'il sera nécessaire de discuter. Dans l'équation (2.3) nous avons α le taux d'apprentissage de l'essaim, tandis que dans l'équation (2.4), nous pouvons constater la présence des paramètres ω, c_1 et c_2. ω est une constante, appelée coefficient d'inertie tandis que c_1 et c_2 sont deux constantes appelés coefficients d'accélération. Dans l'équation (2.4), nous pouvons également observer la présence des termes r_1 et r_2 qui sont deux nombres aléatoires tirés dans l'intervalle [0,1] suivant une loi uniforme. Le tirage de r_1 et r_2 est répété pour chaque particule et pour chaque itération de la boucle de l'algorithme.

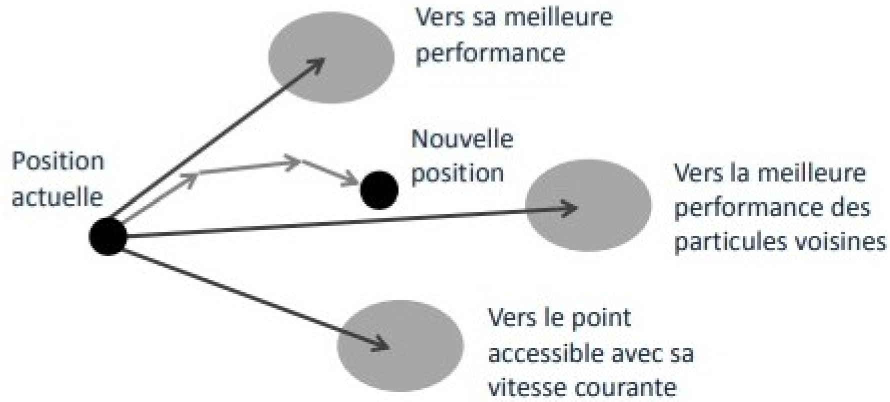

FIGURE 2.4 – Déplacement d'une particule suivant les trois composantes. La composante d'inertie est le point accessible avec la vitesse courante. La composante sociale est la meilleure performance des particules voisines. La composante cognitive est la meilleure performance de la particule (illustration tirée de [15]) .

De plus, nous pouvons constater que, dans l'équation (2.4), nous retrouvons bien les trois composantes annoncées au début de cette sous section qui influencent le déplacement.

- ωv_k^t joue le rôle de la composante d'inertie avec ω qui va permettre de quantifier l'importance de la direction du déplacement courant.

- $r_1 c_1 (x_k^t - p_k^t)$ est la composante cognitive où c_1 va donc quantifier l'importance de p_k sur le déplacement.

- $r_2 c_2 (x_k^t - t_k^t)$ est donc la composante sociale et c_2 va quantifier l'importance de t_k sur le déplacement.

La figure 2.5 illustre l'étape de mise à jour de la vitesse d'une particule. Une fois que les particules se seront déplacées, les nouvelles valeurs de p_k et t_k seront recalculées pour chaque particule suivant les équations (2.5) et (2.6).

$$p_k^{t+1} = \begin{cases} p_k^t, \ \mathrm{si} f(x_k^{t+1}) \geq p_k^t \\ x_k^{t+1}, \ \mathrm{sinon} \end{cases} \tag{2.5}$$

$$t_k^{t+1} = argmin_{p_k} f(p_k^{t+1}), \ 1 \leq k \leq D. \tag{2.6}$$

2.1.5 La condition d'arrêt

La condition d'arrêt de la boucle de l'algorithme est cruciale. En effet, l'algorithme PSO faisant partie de la classe des méthodes heuristiques, il cherche à fournir une solution qui ne sera pas forcément exacte mais qui se doit d'être "acceptable". C'est bien l'utilisation d'une bonne condition d'arrêt qui va garantir la qualité de la solution fournie à la fin de l'algorithme.

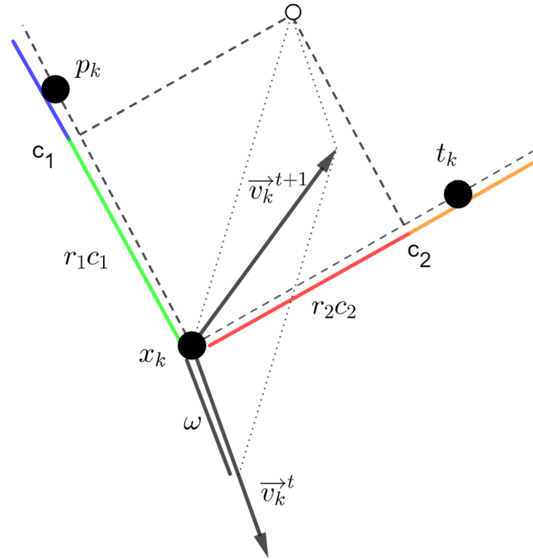

FIGURE 2.5 – Mise à jour de la vitesse d'une particule. La particule est représentée en noir au point x_k, la mémoire de sa meilleure position est représentée au point p_k et son "meilleur ami" est représenté au point t_k. Les vecteurs vitesse sont représentés pour les pas temporels t et $t+1$.

Pour l'algorithme PSO standard, deux critères d'arrêt sont utilisés. Dans les cas où la meilleure position de l'espace de recherche est connue, il est naturel d'utiliser une erreur absolue fixée comme seuil d'admissibilité. L'algorithme va alors simplement évaluer, à chaque étape, la différence en valeur absolue entre la fitness du point optimal et la meilleure fitness connue à l'itération courante. L'algorithme se stoppera à l'instant où cette différence sera inférieure à l'erreur posée. Ces cas de figure sont en pratique peu courants car connaître la position optimale revient à connaître la solution du problème d'optimisation à l'avance.

Le deuxième critère d'arrêt est donc le critère qui est le plus utilisé, il s'agit simplement d'un nombre maximal d'itérations posé à l'avance. La difficulté de ce critère réside dans le fait de trouver un nombre d'itérations suffisamment élevé pour laisser à l'algorithme le temps de trouver une solution "acceptable" et suffisamment petit de sorte que le temps de calcul reste "raisonnable". Dans la version standard de l'algorithme PSO, le nombre maximal d'itérations est équivalent à la taille de l'essaim qui est fixe.

2.1.6 Test de l'algorithme PSO standard

A titre de validation de l'algorithme PSO standard, nous avons codé un algorithme PSO standard tel que présenté dans les sous-sections précédentes et nous l'avons utilisé pour obtenir le minimum de la fonction de Rosenbrock. Cette fonction s'exprime suivant l'équation

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2. \quad (2.7)$$

Cette fonction positive atteint 0 qui est sa valeur minimale quand $x = 1$ et $y = 1$. La figure 2.6 nous montre l'évolution de la fitness, c'est à dire de la valeur de la fonction de Rosenbrock, au cours des itérations de l'algorithme PSO. Pour ce test, nous avons pris un essaim de 20

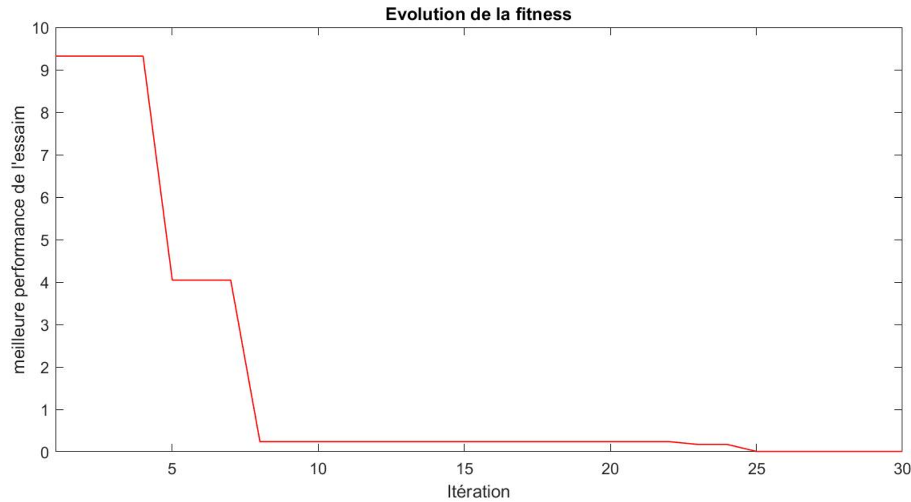

FIGURE 2.6 – Calcul du minimum de la fonction de Rosenbrock par l'algorithme PSO standard. La précision est fixée à 10^{-5}.

particules avec une topologie à 3 liens d'information par particule. Le voisinage de chaque particule est donc constitué de 3 particules sélectionnée aléatoirement parmi l'essaim.

Nous pouvons constater sur la figure 2.6 que la valeur minimale de 0 a été atteinte en un peu plus de 25 itérations, ce qui a pris très peu de temps. Fort de ce test, nous pouvons supposer que la version standard de l'algorithme PSO fonctionne comme attendu.

2.2 les modifications de l'algorithme

Dans cette section, nous présentons diverses modifications et variantes de l'algorithme PSO.

2.2.1 Le paramètre V_{max}

L'algorithme PSO décrit dans la section précédente comporte un problème. Si nous laissons les vitesses des particules telles quelles, il est possible que la vitesse de certaines particules devienne trop importante et conduise les particules en question à sortir de l'espace de recherche. Une solution à ce problème, proposée dans [14], consiste en l'introduction d'une vitesse seuil que nous noterons V_{max}.

L'introduction de ce paramètre va permettre de limiter la vitesse des particules pour ainsi empêcher les problèmes de vitesses trop importantes. Notons que le paramètre V_{max} ne va pas limiter les positions atteignables par les particules, il va simplement limiter la distance maximale qu'une particule peut parcourir en une itération. Ce paramètre va donc également avoir une influence sur le temps de convergence de l'algorithme. En effet, pour un même problème d'optimisation, si nous diminuons la distance maximale que peut parcourir une particule en une itération, le nombre d'itération nécessaire pour arriver à l'optimum global sera naturellement plus grand.

2.2.2 Stratégies de confinement

Outre l'apport du paramètre V_{max}, une autre manière de résoudre les problèmes de vitesses est l'utilisation d'une stratégie de confinement.

L'objectif de ces stratégies est de ramener à l'intérieur du domaine de recherche toutes particules qui pourraient en sortir. Pour cela il existe également plusieurs façon de s'y prendre.

- Une première possibilité serait de laisser les particules qui s'échappent en dehors du domaine de recherche mais de ne plus en évaluer la fitness. Ainsi elles seront ignorées et elles n'influenceront plus les autres particules. En agissant de la sorte, le nombre de particules de l'essaim peut alors diminuer au cours des itérations.

- Une autre façon de procéder serait de stopper, à la frontière du domaine, les particules qui s'apprêteraient à s'échapper en annulant leur vitesse. Les particules dont la vitesse a été annulée resteraient alors bloquée à la frontière du domaine jusqu'à la fin de l'exécution de l'algorithme.

- Une dernière façon de procéder est de faire rebondir les particules sur la frontière du domaine. Pour cela, il faut multiplier les vitesses des particules en question par des coefficients pris aléatoirement dans l'intervalle $[-1, 0]$.

La deuxième option correspond aux versions de 2006 et 2007 de l'algorithme PSO tandis que la troisième option est utilisée dans la version de 2011 de l'algorithme présentée dans [4]. Cette stratégie de confinement se traduit par les équations (2.8) et (2.9).

$$\text{if } (x_k^{t+1} < R_{min}) \begin{cases} x_k^{t+1} = R_{min} \\ v_k^{t+1} = -0.5 v_k^{t+1} \end{cases} \tag{2.8}$$

$$\text{if } (x_k^{t+1} > R_{max}) \begin{cases} x_k^{t+1} = R_{max} \\ v_k^{t+1} = -0.5 v_k^{t+1} \end{cases} \tag{2.9}$$

Les deux premières stratégies de confinement présentent cependant des problèmes potentiels. En effet, dans le cas de la première stratégie, il n'est pas impossible que l'ensemble de l'essaim s'échappe du domaine, ce qui empêcherait la résolution du problème d'optimisation. De même, dans le cas de la seconde stratégie, il est possible que l'ensemble des particules se retrouvent bloqué aux frontières du domaine, ce qui nuirait à la résolution du problème.

Pour cette raison, nous privilégierons donc la stratégie de confinement présentée aux équations (2.8) et (2.9).

2.2.3 Le coefficient de constriction

Tout comme l'utilisation d'un paramètre V_{max}, de nombreuses autres méthodes de contrôle des particules de l'essaim ont été développées. Nous allons ici nous attarder un peu plus sur l'une d'entre elles qui est le coefficient de constriction.

Dans [5], Clerc et Kennedy ont montré que l'utilisation d'un coefficient de constriction noté χ, identique pour chaque particule, permet de se passer du paramètre V_{max} tout en ayant un bon contrôle sur la divergence des particules de l'essaim. La variante de l'algorithme

PSO qui utilise ce coefficient est connue sous le nom de *canonical* PSO. Dans cette variante, l'équation (2.4) de mise à jour des vitesses est modifiée au profit de l'équation

$$v_k^{t+1} = \chi(v_k^t + r_1\phi_1(p_k^t - x_k^t) + r_2\phi_2(t_k^t - x_k^t)), \tag{2.10}$$

avec

$$\chi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}}, \tag{2.11}$$

où

$$\phi = \phi_1 + \phi_2,\ \phi > 4. \tag{2.12}$$

A la suite d'une batterie de test, Clerc et Kennedy ont déterminé des valeurs optimales pour ϕ_1 et ϕ_2. Les valeurs utilisées sont généralement $\phi = 4,1$ avec $\phi_1 = \phi_2$. Cela donne un coefficient de constriction $\chi = 0.7298844$.

Cependant, il a été montré dans [12] que le coefficient de constriction ne permettait pas toujours à lui seul, d'empêcher que les particules ne quittent le domaine. Selon les études de Shi et Eberhart, en plus de l'utilisation d'un coefficient d'inertie, fixer un paramètre $V_{max} = \frac{R_{max} - R_{min}}{2}$, permet de résoudre le problème en améliorant globalement les performances de l'algorithme.

2.2.4 Le coefficient d'inertie

Le coefficient d'inertie ω de l'équation (2.4) a été introduit dans l'algorithme PSO par Shi et Eberhart (dans [11]). Ce coefficient permet de définir l'importance de la direction courante de la particule sur son déplacement futur. Le coefficient ω permet aussi de trouver un équilibre entre la recherche locale et la recherche globale. En effet, une grande valeur de ω va privilégier la recherche globale tandis qu'une plus petite favorisera la recherche locale. Notons aussi que ce coefficient est identique pour toutes les particules de l'essaim.

L'impact qu'a ce coefficient sur la qualité de la PSO a naturellement aidé à son développement et il existe donc plusieurs manières de le définir. Un exemple serait le coefficient d'inertie dynamique défini par Shi et Eberhart dans [11]. L'algorithme débute avec $\omega = 0,9$ et cette valeur descend linéairement vers $0,4$ au cours des itérations. Cette version du coefficient ω est régie par l'équation

$$\omega = \omega_{min} + (\omega_{max} - \omega_{min})\frac{t}{t_{max}}. \tag{2.13}$$

dans laquelle t_{max} est le nombre d'itération maximal et où ω_{min} et ω_{max} sont les valeurs minimales et maximales de ω prises dans l'interval $[0,1]$.

Un autre exemple pour le choix du paramètre ω est un choix proposé par Eberhart et Shi dans [13]. Pour cet exemple, ω est sélectionné aléatoirement dans l'interval [0.5,1] suivant une loi uniforme.

En pratique, il n'existe pas de choix de coefficient parfait. Le choix optimal des paramètres de l'algorithme est bien souvent dépendant du problème que nous cherchons à résoudre. Il faut donc le sélectionner suivant la nature du problème.

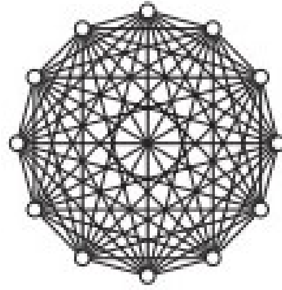

FIGURE 2.7 – Représentation de la topologie *Gbest* (illustration tirée de [15]).

2.3 Le voisinage

Comme nous avons pu le constater dans les sections précédentes, le concept de voisinage est capital pour le bon fonctionnement de l'algorithme PSO. Quand nous parlons de voisinage, il nous vient directement à l'esprit l'idée de définir un voisinage géographique qui définirait les particules voisines à l'aide d'une distance. Cependant, l'emploi d'un voisinage géographique peut être coûteux en termes de calculs numériques dans le cas d'espace de recherche à haute dimension.

Dans la suite de cette section nous allons détailler le deuxième type de voisinage employé qui est le voisinage social. Dans ce type de voisinage, aucune notion de distance n'est utilisée, les particules voisines sont définies au moyen d'un graphe topologique qui va assigner les particules informatrices d'une particule donnée. Pour pouvoir définir ce voisinage, nous allons devoir utiliser le concept de topologie.

Dans le cadre de l'algorithme PSO, une topologie constitue un ensemble de liens entre les particules de l'essaim qui vont définir un réseau de communication entre les particules (représentable au moyen d'un graphe topologique). Pour rappel, pour une particule donnée, nous utilisons le terme voisinage pour parler de l'ensemble des particules qui sont en lien avec elle via la topologie. Ce voisinage constitue l'ensemble des particules informatrices de la particule que nous considérons.

En terme de voisinage social, il existe une multitude de topologies utilisables et qui permettent d'obtenir des solution de bonne qualité. Ces topologies se divisent en deux catégories : les topologies dynamiques et les topologies statiques.

2.3.1 Les topologies statiques

Cette première classe de topologies est qualifiée par le terme statique car il s'agit de topologies dans lesquelles le réseau de communication posé à l'avance ne change pas en cours d'algorithme. Ces topologies ont été les premières topologies utilisées dans l'histoire de l'algorithme PSO. Parmi elles, nous retrouvons la topologie *Gbest*, la topologie utilisée dans la version original de l'algorithme. Dans cette topologie dite globale, toutes les particules de l'essaim sont en lien les unes avec les autres. Pour cette topologie, le voisinage de chaque particule est donc l'intégralité de l'essaim et le "meilleur ami" d'une particule est donc la meilleure particule de tout l'essaim, t_k était alors noté par G_{best} pour *Global Best*, ce qui a inspiré le nom de la topologie. Cette topologie est illustrée par son graphe topologique sur la figure 2.7.

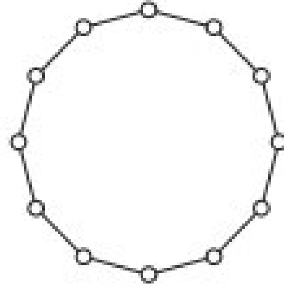

Figure 2.8 – Représentation de la topologie *Lbest* (illustration tirée de [15]).

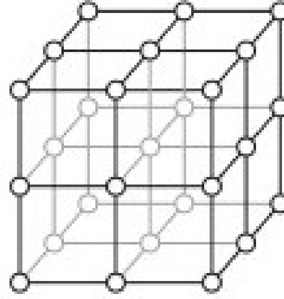

Figure 2.9 – Représentation de la topologie *Von Neuman* (illustration tirée de [15]).

Cependant, l'utilisation de cette topologie ne permettait pas une exploration suffisante de l'espace de recherche. A cause de cela, l'algorithme se retrouvait trop souvent bloqué dans des optima locaux. De nombreuses autres topologies ont donc été développées dans le but d'améliorer les performances de l'algorithme.

La seconde topologie que nous présentons est la topologie *Lbest*. Il s'agit d'une topologie dite en anneau dans laquelle un voisinage restreint est défini pour chaque particule, le terme *Lbest* est donc utilisé pour *Local Best* et fait référence à la meilleure particule d'un voisinage qui n'est plus meilleure globalement mais localement. Dans l'histoire de l'algorithme PSO, cette topologie a été utilisée pour améliorer la convergence de la *version globale* grâce à son réseau à faible porté de communication. La figure 2.8 représente le graphe topologique d'une topologie en anneau.

Pour terminer avec les topologies statiques, nous pouvons parler de trois autres topologies populaires que sont les topologies *Von Neuman*, *Wheel* et *Four-clusters*.

La topologie *Von Neuman* étudiée dans [18] est un réseau sociale, schématisé par un rectangle, dans lequel chaque particule (excepté les particules positionnées sur les bords du rectangle) est en communication avec quatre particules voisines (au dessus, en dessous, à gauche et à droite). Tout comme la topologie en anneau, cette topologie a une faible portée de communication et permet d'éviter les problèmes de la topologie globale en ralentissant la propagation de l'information à travers l'essaim. Notons que comme il s'agit d'un voisinage sociale, les particules "sur les bords du rectangle", informées par 3 autres particules ne verront pas leur position changer au sein du réseau, même si leur position géographique change. Cette topologie est illustrée à la figure 2.9.

La topologie *Wheel* étudiée dans [18] peut être schématisée par une étoile avec une particule principale appelée *focale* en son centre. Toutes les informations vont donc transiter par cette particule centrale qui va les utiliser pour modifier sa propre trajectoire. L'information est

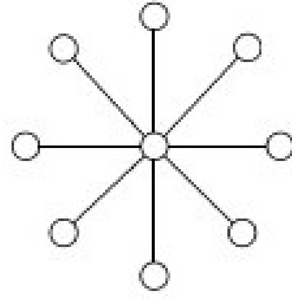

FIGURE 2.10 – Représentation de la topologie *Wheel* (illustration tirée de [15]).

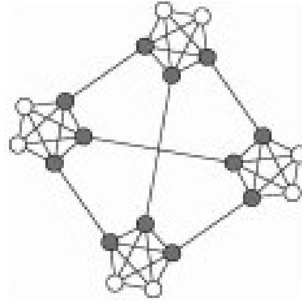

FIGURE 2.11 – Représentation de la topologie *Four-clusters* (illustration tirée de [15]).

relayée aux autres particules si elle permet une amélioration de la fitness. Comme pour les deux topologies précédentes, le réseau d'information de la topologie va ralentir la propagation de l'information, ce qui empêche la stagnation dans certains minima locaux et améliore les performances de l'algorithme. Comme pour la topologie Von Neuman, la position géographique des particules n'est pas considérée, une particule va conserver sa place dans le réseau tout au long de l'exécution de l'algorithme. La topologie est représentée à la figure 2.10.

Enfin, la topologie *Four-clusters* étudiée dans [21] est une topologie qui divise l'essaim en quatre groupes. Chacun de ces groupes relié selon un graphe topologique global et dans ces groupes certaines particules auront la possibilité de communiquer avec une seule autre particule d'une autre communauté. Ces arrêtes entre les particules de différents groupes font office de passerelle. Dans cette topologie, le chemin le plus long allant d'une particule à une autre comporte trois arrêtes. Cela signifie qu'il ne faut que trois itérations de l'algorithme pour que l'information globale soit transmise à l'intégralité des particules. Comme pour les topologies précédentes, la position géographique des particules n'est pas considérée et la topologie ne défini qu'un réseau d'information. La topologie est représentée sur la figure 2.11.

2.3.1.1 Les topologies dynamiques

Comme nous avons pu le constater dans la sous section précédente, de nombreuses topologies statiques ont été développées dans le but d'améliorer les performances de la topologie *Gbest*. Cependant, même si toutes les topologies que nous avons vues constituent chacune une amélioration de l'algorithme par rapport à la topologie *Gbest*, elles ne permettent pas d'éviter à coup sûr la stagnation dans des minima locaux.

Les topologies dynamiques sont des topologies développées pour résoudre ce problème. Elles

ont la particularité de changer la forme de leur graphe topologique au cours des itérations. C'est cette forme changeante de la topologie qui va permettre d'éviter les minima locaux. En effet, en cas de stagnation dans un minimum local, changer la forme de la topologie peut permettre aux particules de l'essaim d'obtenir une information supplémentaire rendant possible la sortie de ce minimum.

La topologie dynamique que nous allons présenter est une topologie qui va définir aléatoirement les liens entre les particules. Cette topologie porte le nom de *adaptive random topology* ou topologie aléatoire adaptive.

2.3.2 La Topologie aléatoire adaptative

Dans cette topologie présentée dans [4], chaque particule de l'essaim s'informe elle-même et va informer un nombre aléatoire d'autres particules. Cependant, un paramètre K souvent posé à 3, va limiter le nombre de particules informées. Cela signifie que chaque particule va informer au plus $K + 1$ particules choisies aléatoirement dans l'ensemble des particules. Comme le choix des particules informées est aléatoire et que rien ne limite le nombre de particules informatrices d'une particule donnée, cela signifie également que chaque particule peut être informée par au plus D particules. Cependant, comme le montre la Figure 2.12, la distribution du nombre de particules informatrices n'est pas uniforme et le nombre de particules informatrices est en moyenne K.

La particularité de la topologie aléatoire adaptative est qu'elle va redéfinir le voisinage des particules en cas de stagnation. Cela signifie que la réafectation des voisinages s'opère si la solution fournie par l'algorithme ne s'est pas améliorée en un nombre donné d'itérations consécutives.

L'utilisation d'une telle topologie permet un partage efficace de l'information entre les particules tout en évitant les problèmes liés aux topologies fortes comme la topologie globale. Elle ne va donc pas subir le défaut des topologies statiques qui peuvent parfois contraindre les particules à travailler en circuit fermé et conduire l'algorithme à stagner dans un minimum local. De plus, par rapport à des topologies telles que la topologie en anneau où des stagnations peuvent parfois survenir, l'étape de réaffectation des voisinages permet à la topologie aléatoire adaptative de potentiellement obtenir une nouvelle information et ainsi s'extraire du minimum local.

2.3.3 Vers une évolution de l'algorithme

Maintenant que les bases concernant l'algorithme PSO ont été posées, nous allons chercher à faire progresser l'algorithme. Pour cela, une fois que nous aurons présenté les fonctions de test qui nous serviront à tester des algorithmes, nous commencerons par développer les différentes variantes d'algorithme PSO que nous avons considéré. Nous présenterons ensuite les outils d'analyse de performance qui nous seront utiles. Après quoi, nous utiliserons ces outils pour déterminer la version de l'algorithme PSO qui nous servira comme base de comparaison. Nous nous servirons aussi de ces outils pour déterminer les meilleures versions de ne nouveaux algorithmes. Nous pourrons alors terminer par une confrontation de l'ensemble de nos algorithmes.

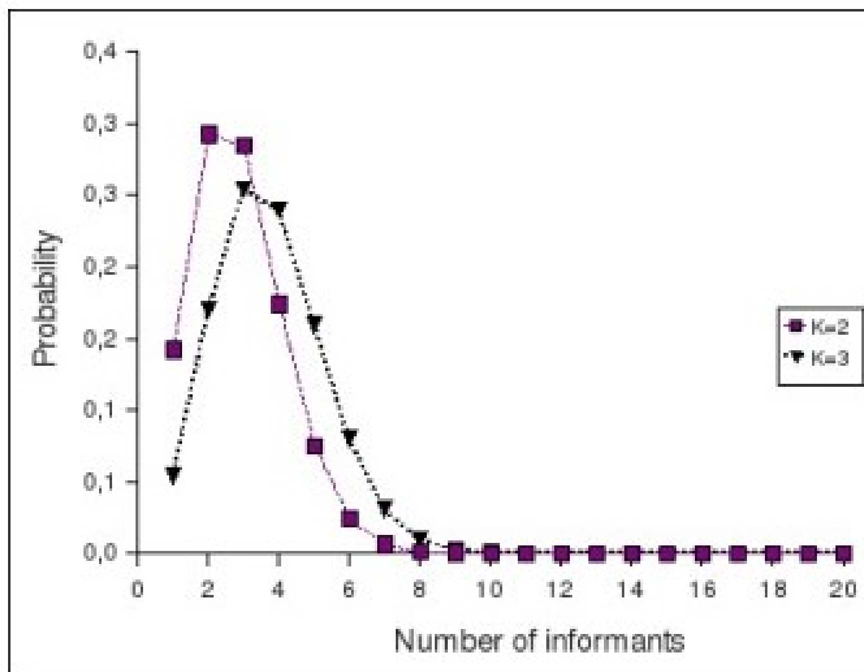

FIGURE 2.12 – Distribution du nombre de particules informatrices d'une autre particule dans la topologie aléatoire adaptative. (illustration tirée de [4]).

Chapitre 3

Modifications de l'algorithme PSO

Ce chapitre présentera notre apport à l'algorithme PSO. Nous y présenterons les différentes modifications que nous y avons apportées. Ces modifications sont celles que nous chercherons à comparer avec les versions déjà existantes de l'algorithme PSO.

3.1 Recuit simulé dans la mise à jour des vitesses et positions

Dans cette section, nous allons présenter la première modification que nous allons apporter à l'algorithme PSO. La modification que nous voulons introduire dans l'algorithme se situe au niveau de la mise à jour des vitesses et des positions. Dans le chapitre 2, nous avions décrit l'étape de mise à jour des vitesses et positions de l'algorithme PSO standard. Pour rappel, en posant

— x_k^t la position de la particule au temps t

— v_k^t la vitesse de la particule au temps t

— p_k^t la meilleure position connue de la particule au temps t

— t_k^t la meilleure position connue du voisinage de la particule au temps t

La mise à jour de la vitesse et de la position d'une particule s'effectue de la manière suivante

$$v_k^{t+1} = \omega v_k^t + r_1 c_1 (x_k^t - p_k^t) + r_2 c_2 (x_k^t - t_k^t), \tag{3.1}$$

$$x_k^{t+1} = x_k^t + \alpha v_k^t, \tag{3.2}$$

où ω est le coefficient d'inertie, α est le taux d'apprentissage de l'essaim. Dans cette étape de mise à jour, comme représenté sur la Figure 2.5, la nouvelle vitesse est influencée par 3 éléments : son inertie, sa meilleure position connue et la meilleure position connue de son voisinage.

La variante de l'algorithme que nous avons créée consiste en une modification de la mise à jour des vitesses et des positions inspirée de l'algorithme du recuit simulé. En effet, comme pour l'algorithme du recuit simulé, deux paramètres sont utilisés dans notre variante : un paramètre T représentant une température ainsi qu'un paramètre $COOL$ faisant office de facteur de refroidissement et appartenant à l'intervalle [0,1[. Au début de l'algorithme, T

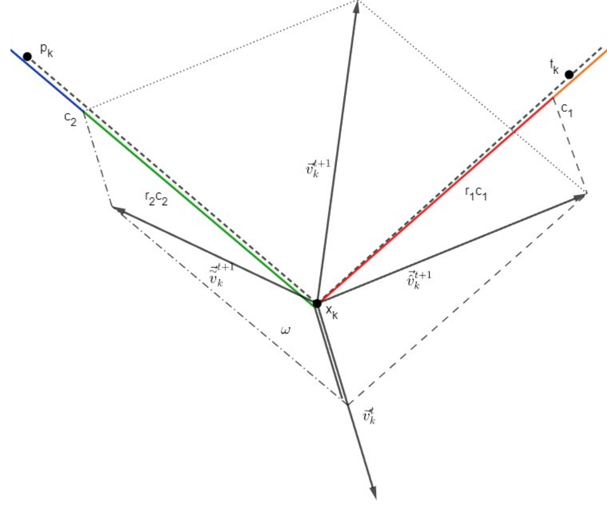

FIGURE 3.1 – Les différentes possibilités de mise à jour de la vitesse dans la variante de l'algorithme PSO basée sur l'algorithme du recuit simulé.

sera fixé à une valeur de départ que nous devrons déterminer et il sera mis à jour à chaque itération en étant multiplié par $COOL$.

$$T = T.COOL \tag{3.3}$$

Dans notre variante, à chaque itération, nous calculons 3 potentielles vitesses et les 3 potentielles positions associées. Le premier couple de vitesse et position est le calcul effectué dans la version standard de l'algorithme PSO, visible dans les équations (3.1) et (3.2). Dans le deuxième couple de vitesse et position, la vitesse est calculée en ne prenant en compte que la meilleure position de la particule, cette mise à jour est réalisée par les équations (3.4) et (3.5).

$$\hat{v}_k^{t+1} = \omega v_k^t + r_1 c_1 (x_k^t - p_k^t), \tag{3.4}$$

$$\hat{x}_k^{t+1} = \hat{x}_k^t + \alpha \hat{v}_k^t \tag{3.5}$$

Le dernier couple de vitesse et position ne prend, lui, en compte que la meilleure position connue du voisinage de la particule. Il correspond aux équations (3.6) et (3.7).

$$\widetilde{v}_k^{t+1} = \omega v_k^t + r_1 c_1 (x_k^t - p_k^t), \tag{3.6}$$

$$\widetilde{x}_k^{t+1} = \widetilde{x}_k^t + \alpha \widetilde{v}_k^t, \tag{3.7}$$

Les différentes possibilités de mise à jour des vitesses sont illustrées sur la Figure 3.1.

Le principe de notre variante est alors de faire un choix entre ces trois possibilités suivant une probabilité qui sera similaire à celle que l'ont peut retrouver dans l'algorithme du recuit simulé. Pour cela, nous définissons d'abord 3 valeurs de la fonction objectif : f_t, la valeur de la fonction objectif évaluée en la position "classique" de la particule, $\hat{f}_{t+1}$, la valeur de la fonction objectif calculée avec la position de l'équation (3.5) et $\widetilde{f}_{t+1}$, la valeur de la fonction objectif calculée avec la position de l'équation (3.7).

$$f_t = f(x_k^t)$$
$$\hat{f}_{t+1} = f(\hat{x}_k^{t+1})$$

$$\tilde{f}_{t+1} = f(\tilde{x}_k^{t+1})$$

Une fois ces valeurs définies, pour déterminer la nouvelle position qui sera conservée, il y a deux cas de figure. Dans le premier cas, si l'une des deux nouvelles positions calculée aux équation (3.5) et (3.7) fait mieux que la position "classique", nous conservons alors la meilleure des deux. Ce cas est représenté par les équations (3.8) et (3.9).

$$\text{if } \min(\hat{f}_{t+1}, \tilde{f}_{t+1}) < f_t \tag{3.8}$$

$$x_k^{t+1} = argmin(f(\hat{x}_k^{t+1}), f(\tilde{x}_k^{t+1})) \tag{3.9}$$

Dans le deuxième cas, alors, nous changerons quand même la nouvelle position suivant une probabilité exponentielle négative. Sans perdre de généralité, admettons que la position qui utilise uniquement la meilleure position connue de la particule est la moins bonne des 3 possibilités, il en découle que

$$f_t < \hat{f}_{t+1} < \tilde{f}_{t+1}. \tag{3.10}$$

Nous pouvons alors définir deux exponentielles dépendantes des valeurs calculées de la fonction objectif en nos différentes positions : $e^{-\frac{\hat{f}_{t+1}-f_t}{T}}$ et $e^{-\frac{\tilde{f}_{t+1}-f_t}{T}}$, et nous savons alors par l'équation (3.10) que

$$e^{-\frac{\hat{f}_{t+1}-f_t}{T}} > e^{-\frac{\tilde{f}_{t+1}-f_t}{T}}. \tag{3.11}$$

Pour mettre en oeuvre cette probabilité de choix, nous définissons, à chaque itération, un certain r suivant une loi uniforme sur l'intervalle [0,1] et nous avons 3 cas de figure possibles.

Dans le premier cas, si $e^{-\frac{\tilde{f}_{t+1}-f_t}{T}} > r$, alors nous conserverons la position qui utilise uniquement la meilleure position connue du voisinage de la particule. Ce cas de figure correspond aux équations (3.12) et (3.13).

$$\text{if } e^{-\frac{\tilde{f}_{t+1}-f_t}{T}} > r \tag{3.12}$$

$$x_k^{t+1} = \tilde{x}_k^{t+1} \tag{3.13}$$

Dans le second, si $e^{-\frac{\hat{f}_{t+1}-f_t}{T}} > r > e^{-\frac{\tilde{f}_{t+1}-f_t}{T}}$, alors nous conserverons la position qui utilise uniquement la meilleure position connue de la particule. Ce cas de figure correspond aux équations (3.14) et (3.15).

$$\text{if } e^{-\frac{\hat{f}_{t+1}-f_t}{T}} > r > e^{-\frac{\tilde{f}_{t+1}-f_t}{T}} \tag{3.14}$$

$$x_k^{t+1} = \hat{x}_k^{t+1} \tag{3.15}$$

Dans le dernier cas où $r > e^{-\frac{\hat{f}_{t+1}-f_t}{T}}$, nous ne changeons pas de position et nous gardons la position telle qu'elle aurait été classiquement calculée. Ce cas de figure correspond aux équations (3.16) et (3.17).

$$\text{if } r > e^{-\frac{\hat{f}_{t+1}-f_t}{T}} \tag{3.16}$$

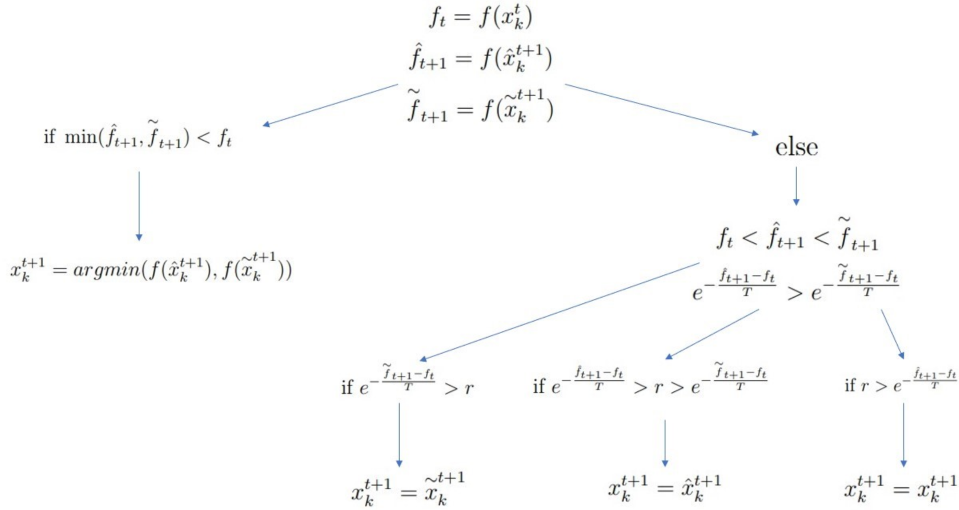

FIGURE 3.2 – Arbre décisionnel du choix de la nouvelle position dans l'algorithme PSO inspiré du recuit simulé.

$$x_k^{t+1} = x_k^{t+1} \tag{3.17}$$

Un schéma des différents choix de positions en fonction des différents cas de figure que nous venons d'expliquer est visible sur la Figure 3.2.

Il pourrait paraître étrange de quand même sélectionner une nouvelle position qui fait moins bien que la position classique. Cependant, tout l'intérêt de cette méthode repose dans les paramètres T et $COOL$. En effet, comme $COOL < 1$, la valeur de T va décroître à chaque itération. De ce fait, les exponentielles $e^{-\frac{\hat{f}_{t+1}-f_t}{T}}$ et $e^{-\frac{\widetilde{f}_{t+1}-f_t}{T}}$ vont également décroître à chaque itération. Cela signifie alors que, en début d'exécution, les chances de "se tromper" de position sont les plus élevées mais que plus nous avançons dans les itérations, moins il y a de chances que r soit inférieur à l'une des deux exponentielles. Cette méthode, tout comme le fait le recuit simulé, va donc potentiellement "se tromper" de position en début d'exécution et va donc ainsi favoriser l'exploration de l'espace de recherche. Mais à mesure que l'algorithme se rapproche de la solution, les chances de se tromper deviendront de plus en plus infimes, ce qui va favoriser l'exploitation de l'espace de recherche.

3.2 Recherche locale dans la mise à jour des vitesses

Dans cette section nous allons présenter une autre variante de l'algorithme PSO standard que nous avons développée. Comme pour la version précédente qui était inspirée du recuit simulé, la modification que nous allons apporter se situe également au niveau du calcul des vitesses et des positions. Cependant, si la méthode inspirée du recuit simulé était basée sur le fait d'obtenir différentes vitesses et d'en choisir une, cette méthode-ci va toujours conserver une vitesse calculée selon l'équation (3.1).

Dans l'algorithme PSO, la position est recalculée suivant l'équation 3.2 où α est le taux d'apprentissage de l'essaim. Cette mise à jour consiste en le fait de chercher une meilleure

position dans la direction de la vitesse calculée par l'équation (3.1) mais l'ampleur du pas réalisé dans cette direction est aléatoire puisqu'elle dépend de r_1 et r_2 sont des nombres générés aléatoirement suivant une loi uniforme dans l'intervalle [0,1].

Le principe de cette nouvelle méthode va alors être d'optimiser le déplacement dans la direction de la vitesse qui aura été calculée. Pour cela, une fois la vitesse calculée pour la particule, nous effectuerons une recherche locale dans la direction trouvée pour garder la meilleure des positions de cette direction. Nous n'avons cependant pas accès à cette meilleure position facilement et selon les fonctions objectifs, il peut parfois être impossible de calculer la meilleure position dans la direction de la vitesse.

La méthode que nous avons développée est alors la suivante : une fois la vitesse obtenue, nous la normalisons de sorte à obtenir un vecteur de norme 1. Optimiser la fonction objectif dans la direction de la vitesse revient alors à trouver la valeur de α qui minimise $f(x_k^{t+1} + \alpha \frac{v_k^{t+1}}{||v_k^{t+1}||})$.

3.2.1 Méthode de recherche tabou

Pour la minimisation de $f(x_k^{t+1} + \alpha \frac{v_k^{t+1}}{||v_k^{t+1}||})$ selon α, nous avons envisagé deux possibilités. Pour notre première méthode, nous procédons d'une manière inspirée de la recherche tabou dans laquelle nous définissons un nombre d'essais permis à l'algorithme pour définir aléatoirement un nouveau α qui fait mieux que la nouvelle position que nous aurions obtenue classiquement.

L'intervalle dans lequel nous cherchons alpha est $[-2 * ||v_k^{t+1}||, 2 * ||v_k^{t+1}||]$. Cela nous permet d'avoir un intervalle de recherche proportionnel à la norme de la vitesse calculée. Au bout du nombre d'essais donné, si une amélioration a été trouvée, nous réitérons ce processus, si aucune nouvelle amélioration n'a été trouvée, nous conservons la meilleure position qui a été obtenue. Notons que cette position peut être la position "classique" dans le cas ou aucune amélioration n'est jamais trouvée.

Dans cette version de l'algorithme PSO, un paramètre important a déterminer est le nombre d'essais autorisé à l'algorithme pour trouver une meilleure position. Nous discuterons le choix de ce paramètre dans les chapitres suivants.

3.2.2 Recherche linéaire exacte

Notre première méthode de recherche locale utilise une méthode heuristique pour le problème de minimisation de la fonction $f(x_k^{t+1} + \alpha \frac{v_k^{t+1}}{||v_k^{t+1}||})$, elle nous permet de trouver des valeurs de α qui nous amènent à de meilleures positions mais elle ne nous permet pas de pouvoir trouver de façon certaine le meilleur α.

La seconde méthode est donc une méthode qui va rendre cela possible. La méthode que nous avons choisi d'utiliser est une méthode de recherche linéaire exacte utilisant la direction de descente de la plus forte pente. Étant donné que le problème de minimisation que nous devons résoudre n'est qu'un problème de minimisation à une dimension, notre algorithme de recherche linéaire exacte prend alors la forme suivante :

Nous commençons par définir un nombre maximum d'itérations (que nous avons posé à 30 pour limiter les évaluations de fonction objectif), une tolérance (fixée à 10^{-5}) ainsi qu'une

valeur initiale de α_{init}. Pour partir de la position qui aurait été sélectionnée dans le cas standard, nous prenons

$$\alpha_{init} = ||v_k^{t+1}||.$$

Comme nous travaillons avec une direction de plus forte pente, nous prenons également $d = -f'(\alpha_{init})$ comme direction de descente initiale. Nous rentrons ensuite dans la boucle de notre recherche linéaire dans laquelle nous allons effectuer différentes opérations jusqu'à ce qu'une condition d'arrêt soit vérifiée.

A l'itération k, dans la boucle de l'algorithme, nous commençons par calculer la dérivée première et seconde de la fonction $f(\alpha) = f(x_k^{t+1} + \alpha \frac{v_k^{t+1}}{||v_k^{t+1}||})$. Comme nous ne connaissons pas l'expression générale des dérivées première et seconde de cette fonction de α, nous en effectuons une approximation en utilisant les formules suivante :

$$f'(\alpha_k) = \frac{f(\alpha_k + h) - f(\alpha_k - h)}{2h}$$

$$f''(\alpha_k) = \frac{f(\alpha_k + h) + f(\alpha_k - h) - 2f(\alpha_k)}{h^2}$$

pour h = 10^{-5}. Nous calculons ensuite, à l'aide des dérivées première et seconde, un paramètre β qui va définir l'ampleur du déplacement dans la direction de descente. Le calcul de β est le suivant :

$$\beta = \frac{(f'(\alpha_k))^2}{f'(\alpha_k) * f''(\alpha_k) * f'(\alpha_k)} = \frac{1}{f''(\alpha_k)}.$$

Nous mettons ensuite à jour α suivant

$$\alpha_{k+1} = \alpha_k + \beta * d.$$

Une itération de la boucle se termine enfin par le calcul de la nouvelle direction de descente $d = -f'(\alpha_{k+1})$. Nous sortons de la boucle dans le cas où $|f'(\alpha_{k+1})| < 10^{-5}$ ou bien après 30 itérations.

Cet algorithme est le dernier des algorithmes que nous avons essayé de développer dans le cadre de ce mémoire. Dans les chapitres suivants, nous présenterons les fonctions de test et les outils qui vont nous permettre d'analyser l'efficacité de nos différents algorithmes.

Chapitre 4

Cas-tests utilisés

Pour pouvoir éprouver la validité d'un algorithme, il faut le mettre à l'épreuve sur certaines fonctions de référence appelées "cas-test". Dans ce chapitre, nous présenterons donc les différents cas-tests, tirés de [26], que nous utiliserons pour tester les modifications que nous avons apporté à l'algorithme PSO. Pour chaque cas-test, nous présenterons son expression analytique, la valeur de sa solution globale et nous fournirons une représentation de la fonction. Nous pouvons aussi déjà noter que notre corpus de cas-tests ne contient que des fonctions objectif sans contraintes.

Rosenbrock

L'expression analytique de la fonction Rosenbrock est

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2) + (x_i - 1)^2],$$

pour $x \in [-30, 30]^n$ où n est le nombre de variables. La fonction possède un minimum global situé en $x_i^* = 1, i = 1, ..., n$ et la valeur de la fonction en ce minimum est 0. La fonction est unimodale pour $n < 4$ mais elle devient multimodale pour $n \geq 4$.

La figure 4.1 présente une représentation de la fonction Rosenbrock pour $n = 2$ ainsi que les courbes de niveau associées. Un zoom autour du minimum gloal est visible sur la Figure 4.2.

Ackley

L'expression analytique de la fonction Ackley est

$$f(x) = -20\ exp\Big(-0,2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\Big) - exp\Big(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\Big) + 20 + e,$$

pour $x \in [-32, 32]^n$ où n est le nombre de variables. La fonction possède un minimum global situé en $x_i^* = 0, i = 1, ..., n$ et la valeur de la fonction en ce minimum est 0.

La figure 4.3 présente une représentation de la fonction Ackley pour $n = 2$ ainsi que les courbes de niveau associées.

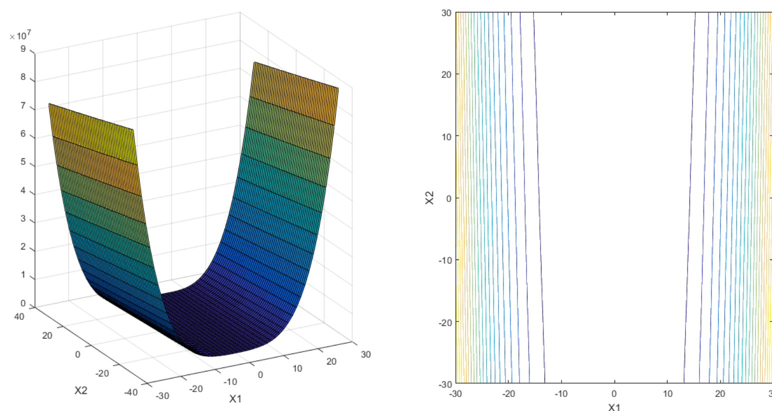

FIGURE 4.1 – Représentation et courbes de niveau de la fonction Rosenbrock.

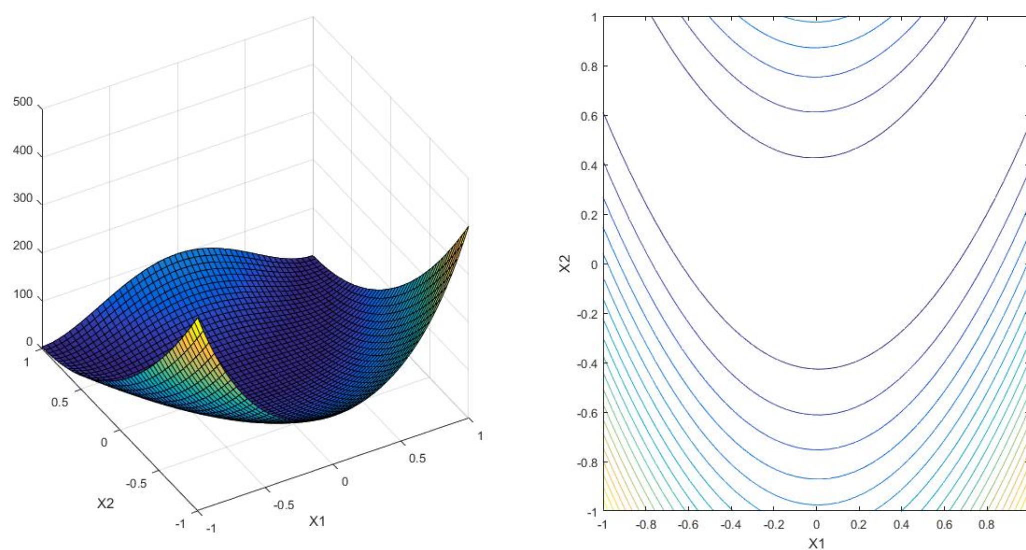

FIGURE 4.2 – Représentation et courbes de niveau de la fonction Rosenbrock autour du minimum globale.

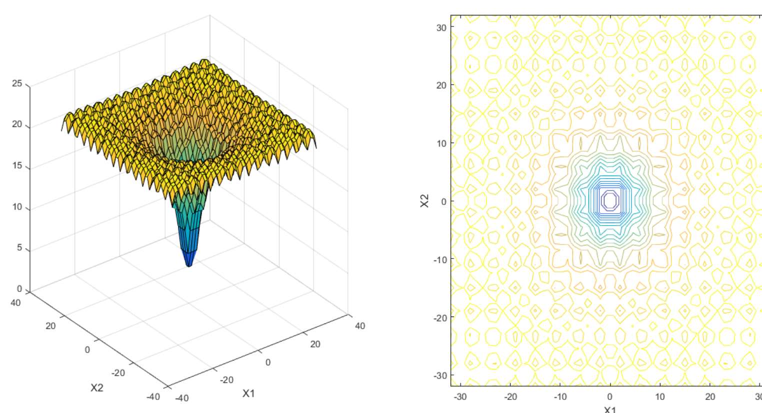

FIGURE 4.3 – Représentation et courbes de niveau de la fonction Ackley.

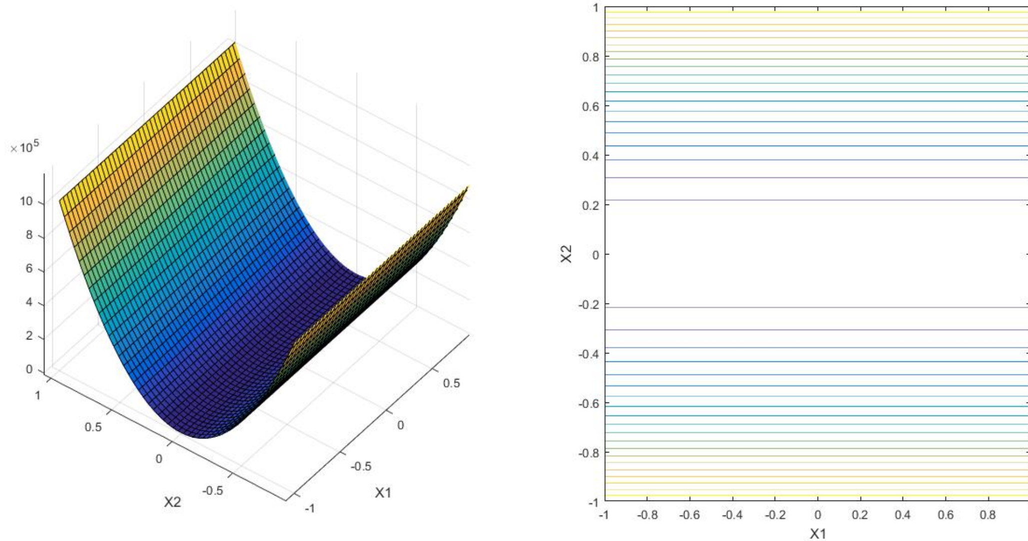

FIGURE 4.4 – Représentation et courbes de niveau de la fonction Elliptique.

Elliptique

L'expression analytique de la fonction Elliptique est

$$f(x) = \sum_{i=1}^{n} 10^{6\frac{i-1}{n-1}} x_i^2,$$

pour $x \in [-100, 100]^n$ où n est le nombre de variables. La fonction possède un minimum global situé en $x_i^* = 0, i = 1, ..., n$ et la valeur de la fonction en ce minimum est 0.
La figure 4.4 présente une représentation de la fonction Elliptique pour $n = 2$ ainsi que les courbes de niveau associées.

Rastrigin

L'expression analytique de la fonction Rastrigin est

$$f(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10],$$

pour $x \in [-5.12, 5.12]^n$ où n est le nombre de variables. La fonction possède un minimum global situé en $x_i^* = 0, i = 1, ..., n$ et la valeur de la fonction en ce minimum est 0. La fonction est également multimodale.
La figure 4.5 présente une représentation de la fonction Rastrigin pour $n = 2$ ainsi que les courbes de niveau associées.

Schwefel 12

L'expression analytique de la fonction Schwefel 12 est

$$f(x) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j^2),$$

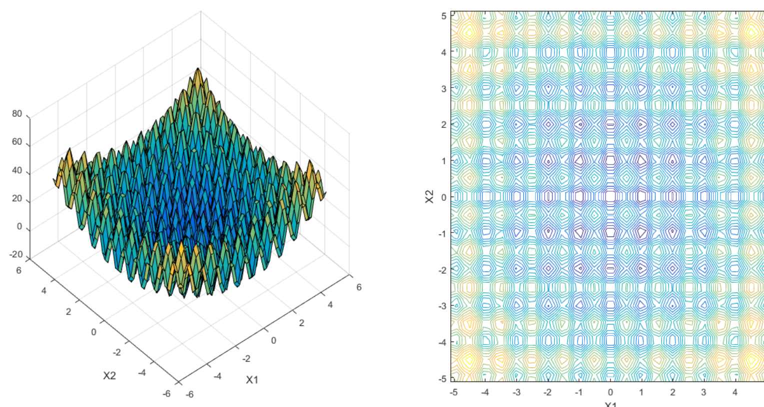

FIGURE 4.5 – Représentation et courbes de niveau de la fonction Rastrigin.

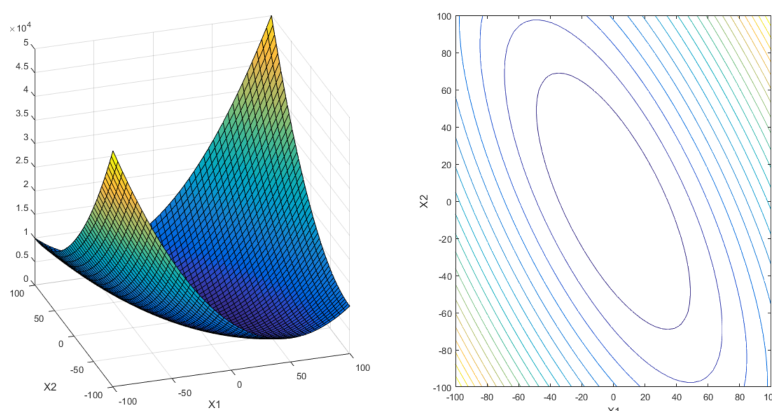

FIGURE 4.6 – Représentation et courbes de niveau de la fonction Schwefel 12.

pour $x \in [-100, 100]^n$ où n est le nombre de variables. La fonction possède un minimum global situé en $x_i^* = 0, i = 1, ..., n$ et la valeur de la fonction en ce minimum est 0.
La figure 4.6 présente une représentation de la fonction Schwefel 12 pour $n = 2$ ainsi que les courbes de niveau associées.

Schwefel 2.21

L'expression analytique de la fonction Schwefel 2.21 est

$$f(x) = \max_i \{|x_i|, 1 \leq i \leq n\},$$

pour $x \in [-100, 100]^n$ où n est le nombre de variables. La fonction possède un minimum global situé en $x_i^* = 0, i = 1, ..., n$ et la valeur de la fonction en ce minimum est 0.
La figure 4.7 présente une représentation de la fonction Schwefel 2.21 pour $n = 2$ ainsi que les courbes de niveau associées.

Schwefel 2.22

L'expression analytique de la fonction Schwefel 2.22 est

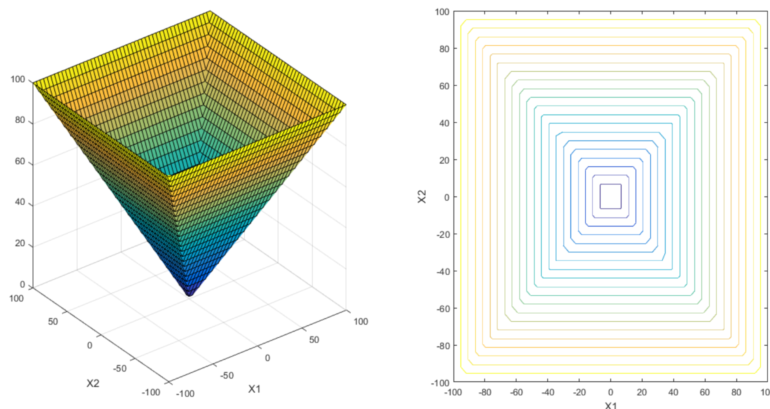

FIGURE 4.7 – Représentation et courbes de niveau de la fonction Schwefel 2.21.

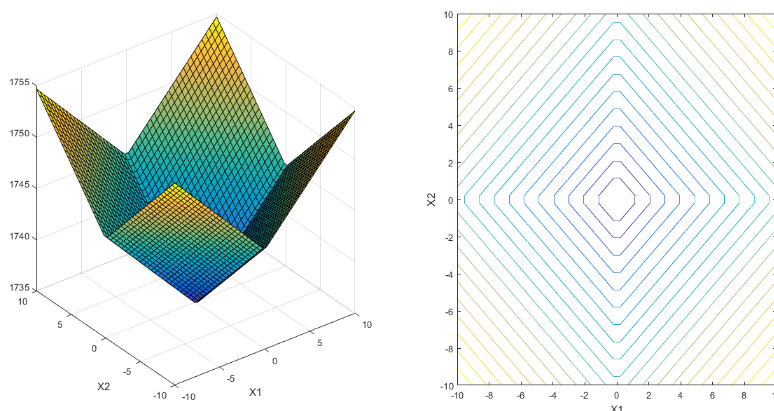

FIGURE 4.8 – Représentation et courbes de niveau de la fonction Schwefel 2.22.

$$f(x) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|,$$

pour $x \in [-10, 10]^n$ où n est le nombre de variables. La fonction possède un minimum global situé en $x_i^* = 0, i = 1, ..., n$ et la valeur de la fonction en ce minimum est 0.
La figure 4.8 présente une représentation de la fonction Schwefel 2.22 pour $n = 2$ ainsi que les courbes de niveau associées.

Griewank

L'expression analytique de la fonction Griewank est

$$f(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1,$$

pour $x \in [-600, 600]^n$ où n est le nombre de variables. La fonction possède un minimum global situé en $x_i^* = 0, i = 1, ..., n$ et la valeur de la fonction en ce minimum est 0. La fonction est également multimodale.
La figure 4.9 présente une représentation de la fonction Griewank pour $n = 2$ ainsi que les courbes de niveau associées.

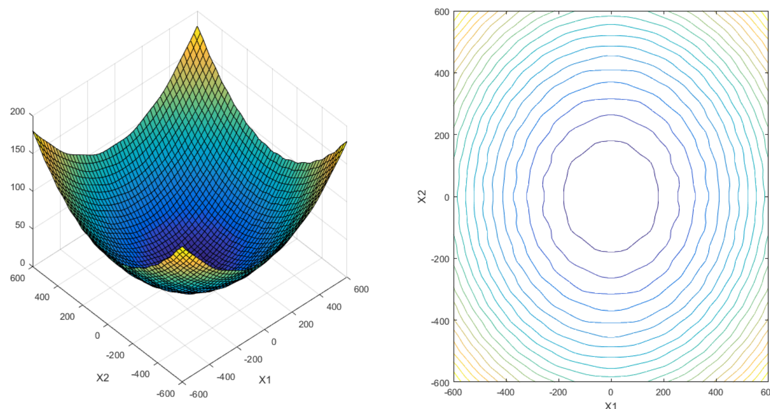

FIGURE 4.9 – Représentation et courbes de niveau de la fonction Griewank.

Comme nous l'avons mentionné en début de chapitre, ces fonctions, que nous avons sélectionnées pour avoir un échantillon varié de cas-tests, vont constituer le corpus de cas-tests qui nous servira à tester les différentes variantes de l'algorithme PSO que nous avons développé. Cependant, la capacité d'un algorithme à pouvoir trouver l'optimum global de nos différents cas-tests n'est pas une condition suffisante pour pouvoir mener une comparaison objective. C'est pourquoi, dans la section suivante, nous développons les méthodes qui vont nous permettre de comparer nos algorithmes entre eux.

Chapitre 5

Outils d'analyse des performances

Dans ce chapitre, nous présentons les outils qui vont nous permettre de comparer les différentes variantes de l'algorithme PSO que nous avons développé.

Nous avons utilisé deux méthodes différentes pour comparer nos algorithmes. La première méthode que nous présentons est une méthode qui va évaluer la performance globale des algorithmes, appliquée à un ensemble de problèmes d'optimisation que nous appelons cas-tests. La seconde méthode est une méthode qui nous permettra de comparer les algorithmes cas-test par cas-test. Avant de développer ces méthodes, rappelons que les problèmes d'optimisation que nous considérons sont ceux définis dans le chapitre 1 tels que les fonctions objectifs sont de la forme

$$\begin{array}{ccccc} f & : & R \subset \mathbb{R}^n & \to & \mathbb{R} \\ & & x & \mapsto & f(x), \end{array} \tag{5.1}$$

avec un minimum globale x^* qui vérifie

$$\forall\ c_e \in C_e,\ \forall\ c_i \in C_i,\ \forall\ x \in \mathbb{R} \tag{5.2}$$

$$\begin{cases} f(x^*) \ < \ f(x) \\ c_e(x^*) = 0 \\ c_i(x^*) \leq 0 \end{cases}$$

Précisons que dans le cadre de notre tentative d'amélioration de l'algorithme PSO, nous nous sommes placé dans une situation d'optimisation sans contraintes. En effet, les différents cas-tests du chapitre précédent ne contiennent aucune contrainte d'égalité ou d'inégalité. De ce fait, la valeur de chacun de nos cas-tests étant connue en x^*, pour déterminer si un algorithme a pu résoudre un cas-test donné, nous avons fixé une tolérance ϵ sur la solution et nous considérons alors qu'un algorithme a réussi à résoudre un problème d'optimisation donné si elle a réussi à fournir une solution $\widetilde{x}$ telle que

$$err_f(\widetilde{x}) \leq \epsilon, \tag{5.3}$$

où $err_f(\widetilde{x})$ est l'erreur absolue entre la fonction objectif évaluée en $\widetilde{x}$ et la fonction objectif évaluée en x^*. Pour toute solution $\widetilde{x}$ proposée par un algorithme, cette erreur est donc définie comme

$$err_f(\widetilde{x}) = |f(\widetilde{x}) - f(x^*)|.$$

Comme nous l'avons déjà évoqué dans le chapitre 1, dans notre mémoire, les éléments qui nous permettront d'analyser les performances d'un algorithme sont

- La capacité ;
- L'efficacité ;
- La polyvalence ;

Le critère de capacité relève de la capacité d'un algorithme à résoudre un cas-test donné en trouvant une solution vérifiant (5.3). Pour le critère d'efficacité, nous considérerons qu'un algorithme est plus efficace qu'un autre si il parvient à résoudre le cas-test considéré pour un moindre nombre d'évaluations de la fonction f. Enfin, la polyvalence est la capacité d'un algorithme à résoudre une large variété de cas-tests. Dans notre cas, nous donnerons la plus haute mesure de polyvalence aux algorithmes capables de résoudre tous les cas-tests présentés dans le chapitre précédent.

Nous allons maintenant présenter la méthode d'analyse des performances globales des algorithmes que nous utiliserons.

5.1 Profils de performance

La notion de performance globale a été explorée et travaillée par Dolan et Moré dans [8]. La méthode qu'ils ont développée est une méthode permettant de comparer la capacité de différents algorithmes à résoudre un ensemble de problèmes d'optimisation suivant un budget donné. Nous parlerons de cette méthode comme étant la méthode de profils de performance. Le développement du profil de performance pour un cas similaire au nôtre a déjà été réalisé dans [23], les choix de développement que nous avons fais en sont donc inspirés.

Pour développer cette méthode, nous noterons C comme étant l'ensemble des problèmes d'optimisation à résoudre, soit l'ensemble de nos cas-tests. Nous noterons également A comme étant l'ensemble des algorithmes que nous voulons évaluer. En ce qui concerne le budget donné, dans [8], Dolan et Moré définissent ce budget comme étant le temps d'exécution de l'algorithme. Cependant, le temps de calcul n'est pas quelque chose que nous pouvons évaluer avec précision, sans oublier le fait que le temps de calcul peut aussi être influencé par la machine qui exécute l'algorithme. C'est pourquoi, dans un souci de contrôle et d'objectivité, nous ne définirons pas ce budget comme un temps d'exécution mais plutôt comme le nombre d'évaluation de la fonction f effectué pendant l'exécution de l'algorithme. Notons que ce choix de budget ne devrait pas porter atteinte à la méthode puisque la méthode de Dolan et Moré calcule la performance globale d'un algorithme pour un budget donné. Nous pourrons alors simplement définir un nombre maximum d'évaluation de la fonction objectif.

Nous pouvons aussi noter que, suivant les fonctions considérées, une évaluation de la fonction objectif peut être plus ou moins coûteuse. Par exemple, si nous considérons une fonction à 10 variables et une fonction à 5 variables, évaluer 10 fois la première fonction a un coût, en terme de temps de calcul, plus élevé que celui des 10 évaluations de la seconde fonction. Fixer une limite du nombre d'évaluations serait alors inéquitable car cela reviendrait à laisser plus de temps à la première fonction. Dans notre cas, la dimension de toutes les fonctions étant la même, nos fonctions ont un temps d'évaluation sensiblement égal, ce qui nous permet de définir le nombre d'évaluation de fonction comme budget.

Sur base des différents éléments que nous venons de poser, le principe de la méthode de profils de performance est de comparer les proportions de cas-tests résolus par chaque algorithme en fonction d'un budget donné.

Dès lors, selon [8], plusieurs notions sont nécessaires pour construire la méthode de Dolan et Moré. Pour commencer, il est utile de connaître le budget alloué aux algorithmes de A pour résoudre les cas-tests de C. Pour ce faire, nous définissons ce budget, pour tout algorithme $a \in A$ et pour tout cas-test $c \in C$, comme étant le nombre médian d'évaluations nécessaire pour que l'algorithme a résolve le cas-test c. Nous notons ce budget $t_{c,a}$. Dans notre cas, comme nous l'avons précisé en début de chapitre, résoudre un cas-test revient à trouver une valeur qui diffère au plus d' ϵ du minimum global du cas-test. $t_{c,a}$ est donc dépendant de ϵ et pour empêcher que cette dépendance n'influence nos résultats, nous avons choisi de travailler avec un ϵ fixé à 10^{-5} pour tous les algorithmes.

Ce choix de budget est basé sur le fait que les algorithmes que nous allons tester sont des algorithmes stochastiques. Cela signifie donc qu'ils sont soumis au hasard et que les résultats fournis diffèrent d'une exécution à l'autre. Nous allons donc devoir exécuter plusieurs fois les algorithmes sur chaque cas-test et prendre la moyenne ou la médiane des nombres d'évaluation de la fonction objectif du cas-test est un choix que nous pensons raisonnable. De plus, une exécution de l'algorithme a n'arriverait pas à résoudre le cas-test c, nous fixerons $t_{c,a}$ à l'infini. Nous avons choisi de prendre la médiane plutôt que la moyenne pour éviter d'avoir un résultat trop influencé par les valeurs extrêmes d'exécutions particulièrement bonnes ou mauvaises.

A l'aide de ce budget, Dolan et Moré définissent alors le ratio de performance pour a selon le cas-test c comme étant le quotient entre le budget de l'algorithme a relatif au cas test c et le plus petit des budget observé parmi les différents algorithmes, pour la résolution du cas-test c :

$$r_{c,a} = \frac{t_{c,a}}{\min\limits_{\tilde{a} \in A} t_{c,\tilde{a}}}. \tag{5.4}$$

Comme le ratio de performance compare le budget d'un algorithme avec celui du meilleur algorithme, dans (5.4) le dénominateur sera toujours inférieur ou égal au numérateur. Cela implique que les valeurs de $r_{c,a}$ seront toujours comprises dans l'intervalle $[1, +\infty[$ de sorte que plus le ratio est proche de 1, plus il est bon. En particulier, dans le cas où $r_{c,a} = 1$, l'algorithme a est alors celui qui demande le plus petit budget pour la résolution du cas-test c.

Dolan et Moré ont finalement utilisé ce ratio de performance pour définir la dernière notion de leur modèle : le profil de performance. Pour un algorithme a, le profil de performance est donné par

$$\rho_a(\tau) = \frac{1}{|C|}|K_{\tau,a}|, \tag{5.5}$$

où,

$$K_{\tau,a} = \{\tilde{c} \in C : r_{\tilde{c},a} \leq \tau\},$$

et où τ est un nombre réel compris dans l'intervalle $[1, +\infty[$. $K_{\tau,a}$ est donc l'ensemble des cas-tests qui ont obtenu un ratio de performance d'au plus τ avec l'algorithme a. Par exemple, si nous fixons $\tau = 2$ seuls les cas-tests ayant un ratio de performance d'au plus 2, pour l'algorithme a, seront repris dans l'ensemble $K_{\tau,a}$. Au regard de la définition du ratio de performance, pour chaque cas-test c, τ peut alors aussi être vu comme la proportion du plus petit budget nécessaire à la résolution de c qui est autorisée pour la résolution de c.

Par exemple, si nous considérons un cas-tests dont le plus petit nombre médian d'évaluation qui a été nécessaire à sa résolution est de 20, fixer $\tau = 2$ revient un nombre médian de 40 évaluations pour résoudre le cas-test.

Cette définition du profil de performance montre donc que $\rho_a(\tau)$ livre la proportion de cas-tests résolus par a suivant la limite de budget τ. Cela signifie aussi que la valeur du profil de performance est comprise entre 0 et 1. Il est alors intéressant d'avoir une valeur de profil de performance proche de 1. En effet, pour un algorithme a donné et pour un τ fixé, plus le profil de performance est proche de 1, plus la proportion de cas-tests résolus par a, pour la limite de budget τ, est grande. En particulier, si nous observons une valeur de 1, cela signifie que l'algorithme a aura alors réussi à optimiser l'intégralité des cas-tests de C dans la limite de budget fixée.

En pratique, pour utiliser ce profil de performance, nous calculerons la valeur du profil de performance en fonction d'une limite sur le budget, τ que nous ferons varier entre 1 et 5. Cela nous permettra d'observer l'évolution de la proportion de cas résolus pour une limite sur le budget qui évolue. La valeur maximale pour τ est une valeur choisie arbitrairement qui est suffisante pour observer de grandes variations du profil de performance.

5.2 Analyse par cas-test

Le profil de performance est un outil qui nous permettra d'étudier la performance globale de nos algorithmes, que ce soit en termes de capacité, d'efficacité ou encore de polyvalence. Cependant, dans certains cas, il pourrait être intéressant d'observer le comportement des algorithmes sur les cas-tests pris individuellement. Les outils d'analyse par cas-test que nous utiliserons ne pourront plus évaluer nos algorithmes en terme de polyvalence mais ils donneront de l'information sur la capacité à résoudre un cas-test ainsi que sur l'efficacité des algorithmes. Nous utiliserons donc un outil pour chacun de ces deux critères. L'algorithme PSO étant une méthode heuristique qui utilise l'aléatoire, les méthodes d'analyse par cas-test vont alors consister en une synthèse d'informations reprises de plusieurs exécutions de l'algorithme sur un cas-test particulier.

Le premier outil que nous utiliserons va nous permettre d'étudier la capacité d'un algorithme à résoudre un cas-test. Pour cela, nous utiliserons simplement un pourcentage du taux de résolution du cas-test considéré. Rappelons que nous considérons le cas-test résolu si l'algorithme a su trouver $\widetilde{x}$ qui vérifie (5.3).

Le second outil que nous utiliserons est un graphique sur lequel nous afficherons, calculées sur base des différentes exécutions de l'algorithme, en abscisse, la médiane du nombre total d'évaluations de la fonction objectif, en ordonnée, la médiane des valeurs de fitness obtenues par l'algorithme. Cet outil nous permettra d'évaluer l'efficacité des algorithmes à résoudre le cas-test. En effet, ces graphiques nous permettront de voir la manière dont les algorithmes se rapprochent de l'optimum. Nous pourrons observer si ils sont plus où moins rapides, ou bien s'ils stagnent par moment sur une valeur de la fonction objectif. Pour utiliser ces outils, même si nous avons obtenu des résultats pour l'ensemble de nos fonctions objectif, nous ne présenterons les résultats que pour une seule de nos fonctions. Les autres résultats seront visibles en annexe.

Chapitre 6

Étude numérique des algorithmes

Dans ce chapitre, nous utiliserons les outils développés dans le chapitre précédent pour déterminer les meilleurs versions des modifications que nous avons apportées à l'algorithme PSO. Ces modifications sont celles développées dans le chapitre 3. Tous les résultats de ce chapitre ont été obtenus avec les fonctions prises en dimension 5 et pour une précision ϵ fixée à 10^{-5}.

6.1 Algorithme PSO utilisé à titre comparatif

Dans le chapitre 2, nous avons décrit l'algorithme PSO standard et ses variantes, mais pour pouvoir améliorer l'algorithme, nous avons besoin de définir une version de l'algorithme PSO qui pourra nous servir de base pour des comparaisons.

Tout d'abord, en ce qui concerne les différentes versions de l'algorithme, nous avons choisi de ne pas utiliser le coefficient de constriction mais plutôt de travailler avec la stratégie de confinement basée sur le rebond des particules à la frontière du domaine. Ensuite, au niveau de la topologie, nous avons choisi d'utiliser la topologie aléatoire adaptative qui était identifiée comme la plus performante dans nos références. De plus, comme il a été montré dans [25], le meilleur nombre de stagnations de l'algorithme avant une redéfinition du voisinage est de 2, nous redéfinirons donc aléatoirement le voisinage des particules si l'algorithme n'a pas trouvé de meilleure solution en 2 itérations. Enfin, en ce qui concerne les différents paramètres de l'algorithme, nous travaillerons avec un essaim de 20 particules, un coefficient d'inertie, ω, égal à 0,738 ainsi que des coefficients c_1 et c_2 tout deux égaux à 1,51.

Le dernier paramètre qu'il nous reste à choisir est le nombre maximum de particules informées. Nous avons indiqué dans le chapitre 2 que le nombre de liens dans la topologie était habituellement fixé à 3 mais maintenant que nous avons présenté des outils permettant d'analyser les performances d'un algorithme, nous pouvons tester l'algorithme PSO avec différents nombres de liens dans la topologie et sélectionner celui qui sera le plus performant.

C'est pourquoi, nous avons appliqué un profil de performance à 5 algorithmes PSO classiques dont nous avons fait varier le nombre de liens de la topologie. Les nombres de liens que nous avons testés sont 1,2,3,5 et 10 liens dans la topologie pour un essaim de 20 particules.

Le profil de performance que nous avons obtenu est visible à la sur la Figure 6.1. Pour obtenir cette figure, nous avons effectué 50 exécutions sur chaque cas-test dont nous avons fixé la dimension à 5. En abscisse, nous pouvons y observer la variation de la limite sur

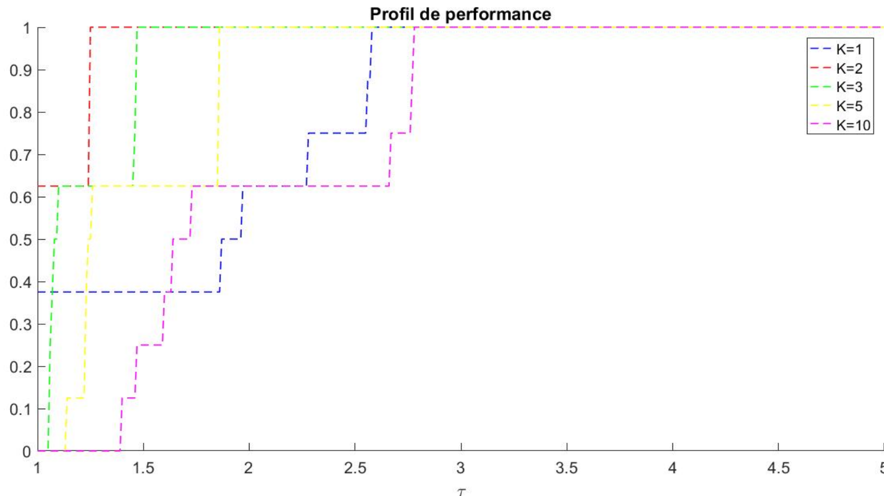

FIGURE 6.1 – Graphique des profils de performance obtenus pour différentes variantes de l'algorithme PSO standard appliqué sur l'ensemble de cas-tests du chapitre 4.

le budget τ tandis qu'en ordonnée, nous pouvons observer le profil de performance $\rho_a(\tau)$ correspondant. Si nous regardons les valeur du profil de performance les plus élevées pour $\tau = 1$, nous observerons les algorithmes qui auront été les plus performants pour résoudre certains cas-tests. Tandis que si l'ordonnée est nulle pour un algorithme avec $\tau = 1$, cela signifie que l'algorithme n'a été le plus performant pour aucun des cas-tests considérés. Ainsi, sur la Figure 6.1, nous pouvons observer que pour un budget minimal, l'algorithme le plus performant est celui utilisant 2 liens dans la topologie. Celui-ci est suivi de près par l'algorithme qui utilise 1 lien tandis que que les 3 autres algorithmes ont un profil de performance nul. En ce qui concerne l'évolution du profil de performance en fonction du budget, nous pouvons constater que l'algorithme à 2 liens dans la topologie est celui qui atteint la valeur de 1 le plus vite. Cela signifie qu'il est l'algorithme qui parvient à résoudre l'ensemble des cas-tests pour le plus petit budget. Au niveau des autres algorithmes, même si ils parviennent à atteindre eux aussi la valeur de 1, ils demandent cependant une limite sur le budget plus élevée et sont donc moins efficaces. Au vu de cette figure, l'algorithme qui est le plus performant serait alors l'algorithme avec 2 liens dans la topologie.

Pour compléter l'analyse du profil de performance de Dolan et Moré, nous avons créé un tableau reprenant les résultats quantitatifs correspondants au graphique de la Figure 6.1. Dans ce tableau, chaque ligne correspond à un des algorithmes testé tandis que nous reprenons 3 informations dans les colonnes du tableau. La première est $\rho_a(1)$, il s'agit de la valeur du profil de performance correspond au budget minimal, pour $\tau = 1$. Cette colonne nous permettra de comparer la performance des algorithmes à budget minimal. La seconde information est $\rho_a(\tau_{max})$ où τ_{max} est la limite sur le budget la plus haute que nous avons utilisé dans nos tests (ici $\tau_{max} = 5$). Cette colonne nous permet alors de voir la valeur de profil de performance qu'est capable d'atteindre l'algorithme testé pour un budget maximal. La dernière information est τ_{best} qui correspond à la valeur de τ la plus petite pour laquelle l'algorithme a atteint $\rho_a(\tau_{max})$. Cette colonne montre alors le plus petit budget à partir duquel les algorithmes sont capables d'atteindre leur plus haute valeur de profil de performance.

La Table 6.1 est la table associée à la Figure 6.1 reprenant les informations que nous venons de décrire. Nous pouvons y observer qu'à budget minimal, l'algorithme avec $K = 2$ est bien

algorithme	$\rho_a(1)$	$\rho_a(\tau_{max})$	τ_{best}
$K = 1$	0.375	1	2.62
$K = 2$	0.625	1	1.25
$K = 3$	0	1	1.47
$K = 5$	0	1	1.87
$K = 10$	0	1	2.78

TABLE 6.1 – Résultats quantitatifs des profils de performance correspondants au graphique de la Figure 6.1.

le meilleur puisqu'il a le profil de performance le plus élevé des algorithmes testés (0.625). Ensuite, nous pouvons observer dans la $3^{ème}$ colonne du tableau que tous les algorithmes parviennent à résoudre l'ensemble des cas-tests si on leur donne le budget maximal. Enfin, au niveau de la dernière colonne, nous pouvons constater que l'algorithme pour $K = 2$ est celui qui parvient à résoudre l'ensemble des cas-tests pour le plus petit budget parmi les différents algorithmes. Il est suivi à une différence de 0.22 près par l'algorithme avec 3 liens dans la topologie. Cependant comme l'algorithme pour $K = 3$ ne résout aucun cas-test à budget minimal, c'est bien l'algorithme à 2 liens dans topologie qui est globalement le plus performant. Pour avoir plus d'informations sur le comportement des algorithmes pour nos cas-tests pris individuellement, nous avons également utilisé l'outil de comparaison par cas-test expliqué dans le chapitre précédent. Les résultats que nous avons obtenus sont visibles sur la Figure 6.2. Ces résultats sont à regarder en parallèle avec la Table 6.2 qui reprend les résultats quantitatifs associés au graphique présenté. Les informations reprises dans ce tableau sont le taux de réussite dans la résolution du cas-test, la meilleure médiane de fitness obtenue, notée $fitness_{min}$, ainsi que la médiane du nombre d'évaluations de la fonction pour laquelle la meilleure médiane de fitness a été obtenue, nous la notons $eval_{best}$. Les résultats ont été obtenus pour 50 évaluations de la fonction objectif.

Au regard de la Figure 6.2, nous pouvons constater que, à l'exception de l'algorithme à 1 lien dans la topologie, les différentes variantes ont un comportement similaire. En effet, les courbes ont toutes la même allure et seule la courbe associée à $K = 1$ va stagner sans atteindre notre précision ϵ. En ce qui concerne la Table 6.2, nous pouvons y observer que l'algorithme qui obtient les meilleurs résultats est celui qui utilise un maximum de 3 liens dans la topologie. En effet, son taux de réussite est de 100%, sa meilleure médiane de fitness est inférieure à ϵ et il est l'algorithme dont $eval_{best}$ est le plus bas. Cependant, Les résultats quantitatifs de l'algorithme à 2 liens maximum dans la topologie sont assez proches de ceux de l'algorithme associé à $K = 3$ (tout comme ceux associés à K = 5 et K = 10) et ces légères différences ne sont pas nécessairement une marque de performance de l'algorithme. En effet, notre algorithme est soumis à l'aléatoire, il est donc possible que ces différences soit dues à de très bonnes ou de très mauvaises exécutions.

Nous conserverons donc l'algorithme associé à $K = 2$, car ses résultats sur la fonction Schwefel 2.22 sont tout de même très bons et surtout, parce qu'il est l'algorithme identifié comme le plus globalement performant par notre profil de performance.
Pour la suite de ce mémoire, tous les algorithmes que nous développerons seront également testés avec les variantes et paramètres de l'algorithme décrit dans cette section. De plus, lorsque nous devrons comparer nos algorithmes à l'algorithme PSO standard, nous utiliserons donc l'algorithme PSO de cette section qui utilisera 2 liens maximum dans la topologie. Enfin, Pour les prochains tests à venir dans ce chapitre, nous utiliserons la même

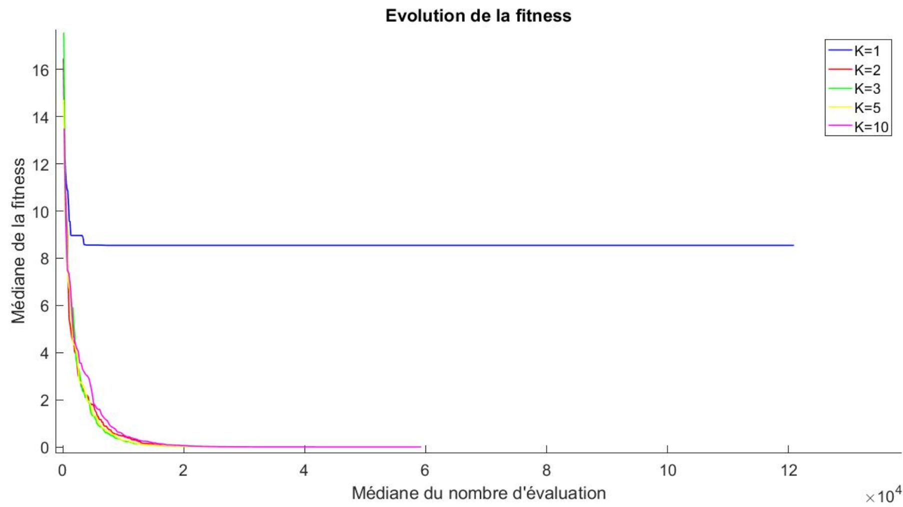

FIGURE 6.2 – Évolution de la valeur de fitness associée à la fonction Schwefel 2.22. Les courbes correspondent aux résultats obtenus pour les algorithmes à 1,2,3,5 et 10 liens maximum dans la topologie aléatoire adaptative.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$K = 1$	0	8,5426	120879
$K = 2$	0.96	5.1523*10^{-6}	52198
$K = 3$	1	8.4656*10^{-6}	47500
$K = 5$	1	4.4789*10^{-6}	52373
$K = 10$	0.92	8.5620*10^{-6}	59260

TABLE 6.2 – Résultats quantitatifs associés au graphique de la Figure 6.2

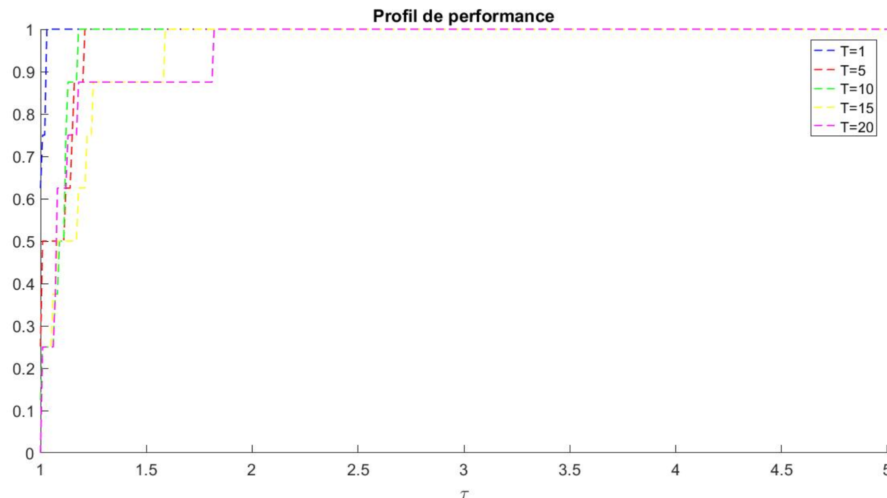

Figure 6.3 – Graphique des profils de performance obtenus pour différentes variantes de l'algorithme PSO avec recuit simulé appliqué sur l'ensemble de cas-tests du chapitre 4. Variation de la Température T initiale.

méthodologie que celle utilisée pour ce test.

6.2 Algorithme inspiré du recuit simulé

Comme nous l'avons précisé précédemment, la variante inspirée du recuit simulée que nous avons développé utilise 2 paramètres T et $COOL$. Nous allons maintenant utiliser le profil de performance de Dolan et Moré pour déterminer les valeurs de ces paramètres qui nous donneront les meilleurs performances globales.

Dans un premier temps, nous avons cherché à déterminer la meilleure valeur initiale du paramètre T. Notons que, dans l'algorithme du recuit simulé, le paramètre T est généralement adapté aux fonctions objectifs. Chercher un T qui serait le plus performant pour un ensemble de cas-test n'est donc pas utile. Cependant, comme nous utilisons le profil de performance de Dolan et Moré pour comparer nos algorithmes, nous allons tout de même chercher, dans notre cas, le T qui fournira les meilleurs résultats. Cela nous permettra ensuite de travailler avec ce "meilleur" T pour définir le paramètre $COOL$.

Pour cela nous avons fixé $COOL$ à 0.99, la valeur habituellement conseillée dans l'algorithme du recuit simulé, et nous avons réalisé un profil de performance pour 5 algorithmes avec des valeurs de T différentes. Les valeurs de T que nous avons testé sont $T = 1$, $T = 5$, $T = 10$, $T = 15$ et $T = 20$. Le profil de performance obtenu est visible sur la Figure 6.3, il a été obtenu pour 50 exécutions de chaque algorithme sur l'ensemble des cas-tests de la section 4 dont nous avons fixé la dimension à 5.

Sur cette figure, nous pouvons observer que à budget minimal, l'algorithme qui résout le plus de cas-tests est celui associé à $T = 1$ et il est également celui qui parvient à résoudre l'ensemble des cas-tests pour la plus petite limite sur le budget. Ce serait donc cette variante qui serait la plus globalement performante.
La Table 6.3 quant elle, reprend les résultats quantitatifs issus du profil de performance de la Figure 6.3. Nous pouvons observer dans ce tableau que l'algorithme associé à $T = 1$ est bien celui qui est le plus intéressant à budget minimal, puisqu'il parvient à résoudre

algorithme	$\rho_a(1)$	$\rho_a(\tau_{max})$	τ_{best}
$T = 1$	0.625	1	1.03
$T = 5$	0.25	1	1.21
$T = 10$	0.125	1	1.18
$T = 15$	0	1	1.59
$T = 20$	0	1	1.82

Table 6.3 – Résultats quantitatifs des profils de performance correspondants au graphique de la Figure 6.3.

plus de la moitié des cas-tests. Nous pouvons également constater que touts les algorithmes parviennent à résoudre l'ensemble des cas-tests à budget maximal mais que c'est bien l'algorithme associé à $T = 1$ qui demande la plus petite limite sur le budget pour y parvenir, avec une limite τ de seulement 1.03 contre 1.18 pour le second meilleur algorithme. Cela nous montre bien que c'est l'algorithme associé à $T = 1$ qui est le plus globalement performant de ceux que nous avons testé. Nous conserveront donc 1 comme valeur initiale de T. Notons que dans le cas de T, utiliser nos outils d'analyse par cas-test n'a pas de sens puisque le meilleur T devrait être différent pour chaque fonction.
Maintenant que nous avons déterminer la meilleure valeur de T, nous allons utiliser cette dernière pour réévaluer la valeur de $COOL$. Sur la Figure 6.4 se trouve le profil de performance que nous avons réalisé pour 5 algorithmes liée à des valeurs différentes du paramètre $COOL$ et utilisant $T = 1$. Nous avons effectué 50 exécutions de chaque algorithme sur l'ensemble des cas-tests de la section 4 pour tester les valeurs de $COOL$ suivante : $COOL = 0.1$, $COOL = 0.3$, $COOL = 0.5$, $COOL = 0.75$, et $COOL = 0.99$.

Sur la Figure 6.4, nous pouvons observer que 0.75 semble être la meilleure valeur du paramètre $COOL$. En effet, c'est l'algorithme utilisant cette valeur qui parvient à atteindre 1 en premier et donc c'est l'algorithme qui arrive à résoudre l'ensemble des cas-tests pour la plus petite limite sur le budget. De façon surprenante, nous pouvons aussi constater que dans notre cas, la valeur 0.99 semble être la moins robuste. Effectivement l'algorithme qui utilise $COOL = 0.99$ demande une limite sur le budget plus élevée que tout les autres algorithmes pour résoudre l'ensemble des cas-tests.

Pour compléter cette analyse, nous pouvons observer la Table 6.4 qui reprend les résultats quantitatifs des profils de performance que nous venons d'observer. Nous pouvons d'abord y constater que à budget minimal ce sont les algorithmes liés à $COOL = 0.1$, $COOL = 0.5$ et $COOL = 0.75$ qui sont les plus performant avec une valeur de profil de performance de 0.125, pour $COOL = 0.1$, 0.25 pour $COOL = 0.5$ et 0.625, soit plus de la moitié, des cas-tests résolu, pour $COOL = 0.75$. Ensuite nous pouvons observer que tous les algorithmes testés parviennent à résoudre l'ensemble des cas-tests à budget maximal. Pour finir, nous pouvons voir que la plus petite limite sur le budget demandée pour la résolution de l'ensemble des cas-tests est demandée par l'algorithme qui utilise $COOL = 0.75$. Nous pouvons aussi observer dans la dernière colonne que l'algorithme utilisant $COOL = 0.99$ est celui qui demande la plus grande limite sur le budget pour la résolution des cas-tests. Comme pour les tests concernant l'algorithme PSO standard, nous présentons également les résultats des tests réalisés sur la fonction Schwefel 2.22 prise individuellement pour 50 exécutions de chaque algorithme. Ces résultats sont visible sur la Figure 6.5 couplée à la Table 6.5. Sur la Figure 6.5, nous pouvons observer que l'allure de toute les coure est identique, les différences dans le comportement des différents algorithmes devraient donc être minime. Cela se confirme

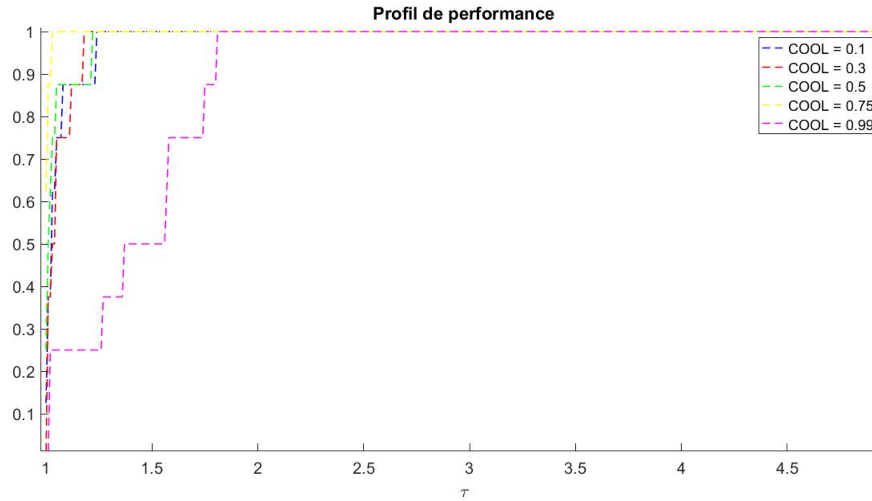

Figure 6.4 – Graphique des profils de performance obtenus pour différentes variantes de l'algorithme PSO avec recuit simulé appliqué sur l'ensemble de cas-tests du chapitre 4. Variation du facteur de refroidissement $COOL$.

algorithme	$\rho_a(1)$	$\rho_a(\tau_{max})$	meilleur τ
$COOL = 0.1$	0.125	1	1.24
$COOL = 0.3$	0	1	1.18
$COOL = 0.5$	0.25	1	1.22
$COOL = 0.75$	0.625	1	1.03
$COOL = 0.99$	0	1	1.81

Table 6.4 – Résultats quantitatifs des profils de performance correspondants au graphique de la Figure 6.4.

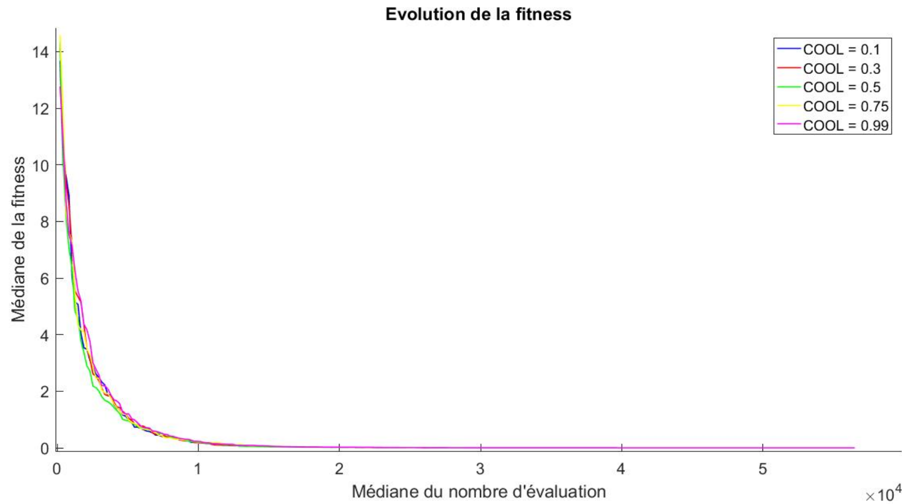

FIGURE 6.5 – Évolution de la valeur de fitness associée à la fonction Schwefel 2.22. Les courbes correspondent aux résultats obtenus pour les algorithmes inspirés du recuit simulé associés aux valeurs de $COOl$ suivantes : 0.1,0.3,0.5,0.75 et 0.99.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$COOL = 0.1$	1	8,5426*10^{-6}	52276
$COOL = 0.3$	1	2.1423*10^{-6}	51293
$COOL = 0.5$	1	6.6548*10^{-6}	53786
$COOL = 0.75$	1	8.5789*10^{-6}	54426
$COOL = 0.99$	0.96	5.3256*10^{-6}	56513

TABLE 6.5 – Résultats quantitatifs associés au graphique de la Figure 6.5

au niveau de la Table 6.5 dans laquelle nous pouvons observer que tous les algorithmes ont une valeur $fitness_{min}$ de l'ordre de 10^{-6} et une valeur $eval_{best}$ de l'ordre de $5 * 10^5$. En ce qui concerne les taux de réussite, nous pouvons observer que seul l'algorithme associé à $COOL = 0.99$ a eu des échecs lors de la résolution du cas-test avec un taux de réussite de 96%. Cela confirme bien le fait fait que l'algorithme associé à $COOL = 0.99$ est le moins performant des 5. Le taux de réussite et les résultats des 4 autres variantes étant similaires, nous nous baserons alors sur le profil de performance pour choisir la meilleure valeur de $COOL$. Nous conserverons donc la valeur de 0.75 pour le paramètre $COOL$.
De cette analyse il ressort alors que la version de notre algorithme la plus globalement performante est celle qui utilise $T = 1$ et $COOL = 0.75$. C'est donc cette version que nous utiliseront lors de notre comparaison finale.

6.3 Algorithme avec recherche tabou

Dans cette section, nous avons utilisé le profil de performance de Dolan et Moré ainsi que nos outils d'analyse par cas-test pour déterminer le nombre d'essais que nous allons autoriser à l'algorithme pendant l'étape de recherche locale. Nous avons réalisé des tests pour des versions de l'algorithme associées à des nombre d'essais différents. Les nombres d'essais que nous avons testés sont 3,5,7,10 et 15. Les résultats du profil de performance que nous présentons dans cette section ont été obtenu pour 50 exécutions par algorithme sur

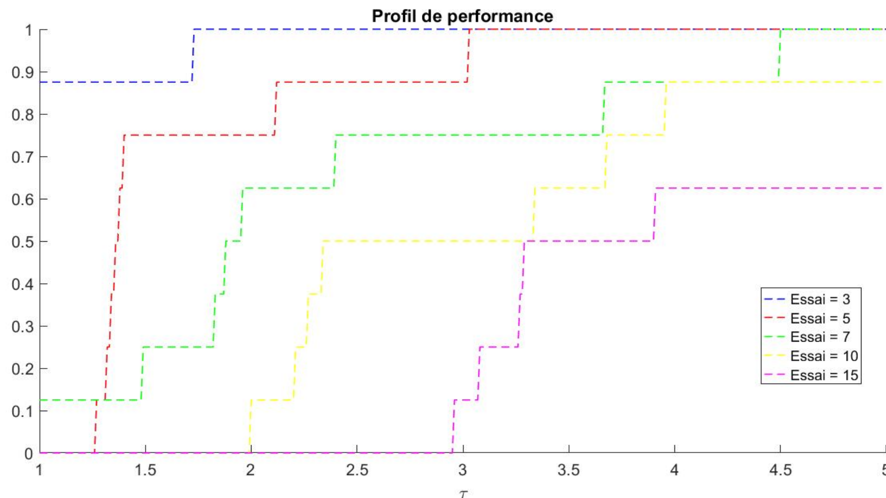

Figure 6.6 – Graphique des profils de performance obtenus pour différentes variantes de l'algorithme PSO, avec une recherche locale basée sur une recherche tabou, appliquée sur l'ensemble de cas-tests du chapitre 4. Variation du nombre d'essais d'amélioration.

algorithme	$\rho_a(1)$	$\rho_a(\tau_{max})$	meilleur τ
$Essai = 3$	0.875	1	1.73
$Essai = 5$	0	1	3.03
$Essai = 7$	0.125	1	4.50
$Essai = 10$	0	0.875	3.96
$Essai = 15$	0	0.625	3.91

Table 6.6 – Résultats quantitatifs des profils de performance correspondants au graphique de la Figure 6.6.

chacun des cas-tests du chapitre 4 dont nous avons pris la dimension à 5.

Le profil de performance que nous avons obtenu est visible sur la Figure 6.6. Nous pouvons y observer que l'algorithme qui fixe le nombre d'essais à 3 semble être le meilleur. En effet, à budget minimal, il est celui qui est capable de résoudre le plus de cas-tests. De plus, il s'agit également de l'algorithme qui atteint 1 en premier et qui demande donc la limite sur le budget la plus petite pour la résolution de l'ensemble des cas-tests. Ce profil de performance nous permet aussi de voir que la performance de l'algorithme semble diminuer à mesure que nous autorisons des essais. Cela est certainement dû au fait que plus nous autorisons d'essai d'amélioration, plus l'algorithme a besoin d'évaluations de la fonction objectif, et donc de budget.
La Table 6.6 reprend les résultats quantitatifs liés à la Figure 6.6. Nous pouvons y observer qu'à budget minimal, seuls les algorithmes à 3 et 7 essais parviennent à résoudre des cas-tests et que l'algorithme à 3 essais a un très bon profil de performance égal à 0.875. Dans la troisième colonne, nous pouvons constater que seuls les algorithme à 3, 5 et 7 essais parviennent à résoudre l'ensemble des cas-tests pour une limite sur le budget maximale. Enfin, nous pouvons aussi constater que l'algorithme à 3 essais ne demande qu'une limite τ de 1.73 pour la résolution des cas-tests. Le second meilleur algorithme pour ce critère, quant à lui, demande une limite τ de 3.03 soit presque 2 fois plus.

De façon similaire aux autres tests de ce chapitre, nous présentons maintenant les résultats

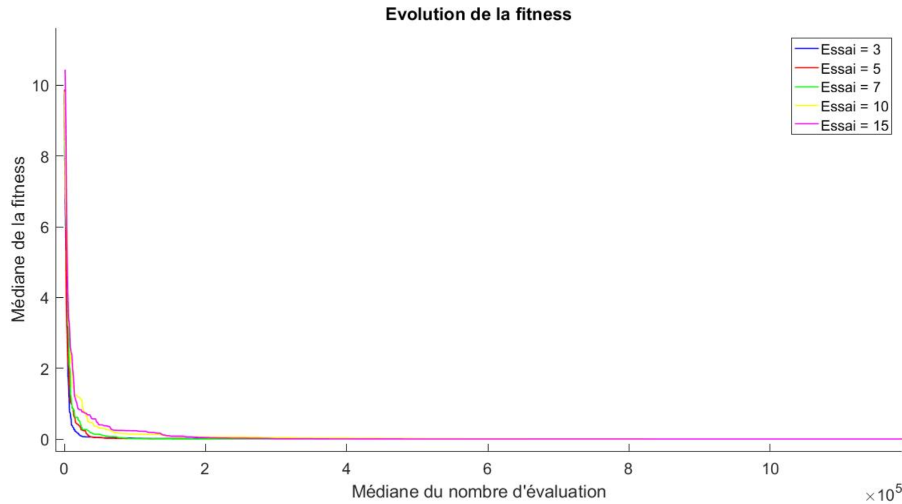

FIGURE 6.7 – Évolution de la valeur de fitness associée à la fonction Schwefel 2.22. Les courbes correspondent aux résultats obtenus pour des algorithmes utilisant une recherche locale, basée sur une recherche tabou, utilisant 3, 5, 7, 0 et 15 essai lors de la recherche d'une amélioration de la position des particules .

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$Essai = 3$	0.15	0.0051	3.3690*10^5
$Essai = 5$	0.3	1.0277*10^{-4}	481288
$Essai = 7$	0.4	1.6444*10^{-5}	6.2298*10^5
$Essai = 10$	0.3	0.0024	8.3437*10^5
$Essai = 15$	0.4	3.0528*10^{-5}	1.1875*10^6

TABLE 6.7 – Résultats quantitatifs associés au graphique de la Figure 6.7

que nous avons obtenus pour 50 exécutions des différents algorithmes sur le cas-test Schwefel 2.22. Ces résultats sont visibles sur la Figure 6.7 et dans la Table 6.7. Au niveau de la Figure 6.7, nous pouvons constater que plus nous ajoutons d'essais dans la recherche d'une meilleure position des particules, moins la courbe descend en début d'algorithme. Cette observation confirme donc l'information du profil de performance qui donnait des algorithmes de moins en moins bons avec l'ajout d'essais. Au vu de la figure 6.7, nous estimons alors que ce phénomène est dû à la vitesse de convergence des algorithmes vers la solution : plus nous autorisons d'essais, plus l'algorithme fera d'évaluations de la fonction objectif et plus lentement il convergera. En ce qui concerne les résultats quantitatifs de la Table 6.7, nous pouvons remarquer que l'ordre de grandeur de $eval_{best}$ est de 10^5 pour tous les algorithmes à l'exception de l'algorithme qui autorise 15 essais. Toujours à ce niveau nous pouvons également constater que $eval_{best}$ augmente avec le nombre d'essais, ce qui confirme l'observation graphique. L'algorithme à 3 essais est donc le meilleur pour ce critère. Au niveau de la valeur $fitness_{min}$, les meilleurs candidats sont les algorithmes à 3 et 10 essais avec une valeur de l'ordre de 10^{-3}. Enfin, en ce qui concerne le taux de réussite des algorithmes à 3 et 10 essais, ils sont respectivement de 15 et 30%. Pour ce critère (pour le cas-test Schwefel 2.22) l'algorithme à 10 essais est donc le meilleur de nos deux candidats. Pour départager les deux algorithmes, nous allons nous fier au profil de performance qui nous avait indiqué que l'algorithme à 3 essais est le plus globalement performant.
Nous retenons de cette analyse que pour cette variante de l'algorithme PSO, la version la

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$RL1$	0	0.1487	$2.9370*10^6$
$RL2$	0.9	$9.3894*10^{-4}$	$3.2*10^5$

Table 6.8 – Résultats quantitatifs associés au graphique de la Figure 6.8

plus performante est celle qui autorise 3 essais. C'est donc cette version que nous retiendrons.

6.4 Algorithme avec Recherche linéaire exacte

Dans cette section, nous discutons d'un problème rencontré avec notre version de l'algorithme PSO qui utilise une recherche linéaire exacte et nous présentons la solution que nous avons envisagée pour le résoudre.

Dans notre algorithme PSO qui utilise une recherche linéaire exacte dans l'étape de recherche locale pour le choix de vitesse, nous nous sommes aperçu lors de nos tests que l'algorithme, bien que rapide en début d'exécution, finissait très souvent par bloquer à une valeur de la fonction objectif sans pouvoir en sortir. Pour résoudre ce problème, nous avons donc pensé à une combinaison entre la recherche linéaire exacte et l'utilisation de la mise à jour standard des vitesses qui permettrait de débloquer l'algorithme en cas de stagnation.

En pratique, nous avons simplement fixé un nombre d'itérations égal à 30 qui représente le nombre d'itérations consécutives durant lesquelles nous autorisons l'algorithme à ne pas fournir d'amélioration de la meilleure solution qu'il a pu atteindre. Tant que ce nombre d'itérations consécutives sans amélioration n'est pas rencontré, nous utilisons la recherche linéaire exacte pour la mise à jour de la vitesse. Mais à l'instant où les 30 itérations consécutives sans amélioration sont atteintes, nous estimons alors que l'algorithme est en train de stagner et nous mettons alors à jour les vitesses suivant la méthode classique. Nous espérons ainsi que l'algorithme aura assez progressé en utilisant la recherche linéaire exacte et nous lui permettons de trouver le minimum global grâce à la mise à jour standard.

Pour confirmer l'efficacité de notre modification, nous avons utilisé nos outils d'analyse par cas-tests. Les résultats que nous avons obtenus pour 50 exécutions des deux algorithmes sur la fonction Schwefel 2.22 sont visibles sur la Figure 6.8 et dans la Table 6.8. Au niveau des courbes de la Figure 6.8, nous pouvons observer que la courbe bleue, associée à l'algorithme sans modification, va bel et bien stagner sans atteindre le minimum global, tandis que la courbe rouge associée à notre modification va suivre la courbe bleue avant de descendre vers le minimum global de la fonction objectif au moment ou la courbe bleue commence à bloquer. Cela nous laisse penser que notre ajout a bien l'effet escompté sur l'algorithme. La Table 6.8, quant à elle, nous montre que la modification que nous avons apportée est bien une amélioration de la méthode avec recherche linéaire exacte puisqu'elle est gagnante à tous les niveaux. En effet, nous gagnons un ordre de grandeur sur $eval_{best}$, la valeur $fitness_{min}$ était de l'ordre de 10^{-1} et est maintenant de l'ordre de 10^{-4} et l'algorithme qui ne résolvait jamais le cas-test Schwefel 2.22 a maintenant un taux de réussite de 90%.
Pour notre comparaison finale, nous utiliserons donc la modification décrite dans cette section pour améliorer la méthode avec recherche linéaire exacte. C'est à dire que nous autoriserons l'algorithme à stagner pendant 30 itérations consécutives avant d'utiliser la mise à jour classique des vitesses.

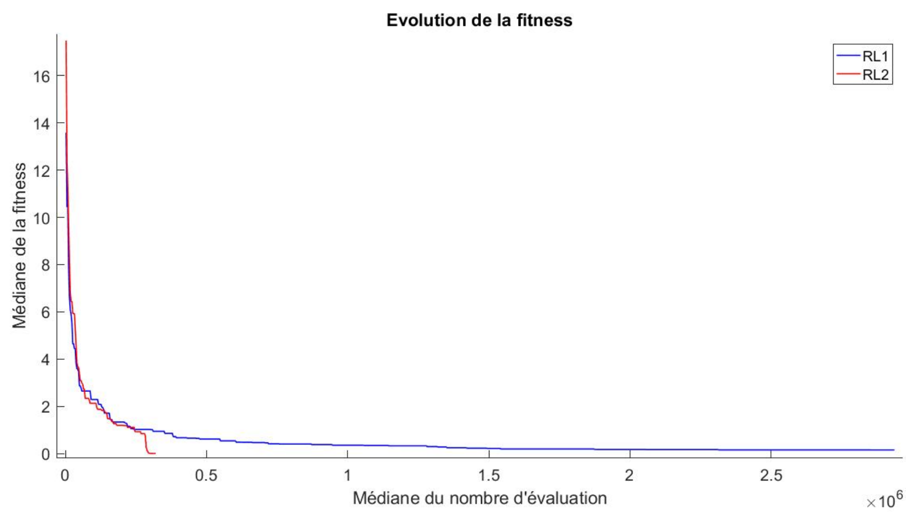

Figure 6.8 – Évolution de la valeur de fitness associée à la fonction Schwefel 2.22. La courbe RL1 correspond à la méthode dans laquelle seule la recherche linéaire exacte est utilisée tandis que la courbe RL2 correspond à la méthode dans laquelle nous recalculons les vitesse de façon classique en cas de stagnation de l'algorithme.

Chapitre 7

Comparaison des différentes options et résultats

Dans ce chapitre, nous allons maintenant chercher à déterminer parmi les nouvelles variantes de l'algorithme PSO que nous avons développées, quelle est la version de l'algorithme PSO qui est la plus performante. Nous regarderons aussi si nos algorithmes peuvent être considérés comme des améliorations de l'algorithme PSO classique.

Pour rappel, les algorithme en lice sont : l'algorithme PSO classique à 2 liens maximum dans la topologie, l'algorithme qui utilise une recherche locale de type recherche tabou pour la mise à jour des vitesses avec 3 essais autorisés pour obtenir une amélioration de la position, l'algorithme qui utilise une recherche locale de type recherche linéaire exacte pour la mise à jour des vitesse ainsi que l'algorithme inspiré du recuit simulé avec $T = 1$ et $COOL = 0.75$.Pour comparer ces différents algorithmes entre eux, nous avons de nouveau utilisé un profil de performance. Pour cela, nous avons exécuté chaque algorithme 50 fois sur chaque cas-test. Le profil de performance que nous avons obtenu est visible sur la Figure 7.1.

Sur cette figure, nous pouvons observer une nette séparation entre les différentes courbes. Nous pouvons constater la courbe de l'algorithme inspiré du recuit simulé est au dessus des 3 autres courbes. Cette dernière est suivie par la courbe associée à l'algorithme standard tandis que les courbes associées aux méthodes de recherche locale s'entremêlent en dessous des deux première courbes. Cette observation laisse à penser que l'algorithme inspiré du recuit simulé est le plus globalement performant. En effet, en $\tau = 1$ il a déjà une valeur de profil de performance supérieure à 0.7 et il atteint également la valeur de 1 en premier. L'algorithme classique serait alors le deuxième algorithme le plus globalement performant puisqu'il démarre avec un profil de performance supérieur à 0.2 et qu'il atteint tout de même 1, signifiant qu'il va finir par résoudre l'ensemble des cas-tests avant le budget maximal autorisé. En ce qui concerne les algorithmes utilisant une recherche locale, il semblerait qu'ils soient les moins performants du lot. En effet, à budget minimal, leur profil de performance est nul tandis qu'à budget maximal, ils atteignent un profil de performance inférieur à 0.6. La Table 7.1 reprend les résultats quantitatifs du graphique de profil de performance que nous venons d'analyser. Nous pouvons y observer que, à budget minimal, le profil de performance de l'algorithme avec recuit simulé est le plus élevé, avec une valeur de 0.75 contre 0.25 pour l'algorithme PSO classique et 0 pour les algorithmes avec recherche locale. Si nous regardons la $3^{ème}$ colonne, nous pouvons constater que seuls les algorithmes classique et avec recuit simulé parviennent à atteindre la valeur de 1 et donc à résoudre l'ensemble des cas-tests pour un des budgets considérés. Enfin, nous pouvons aussi voir que, pour la

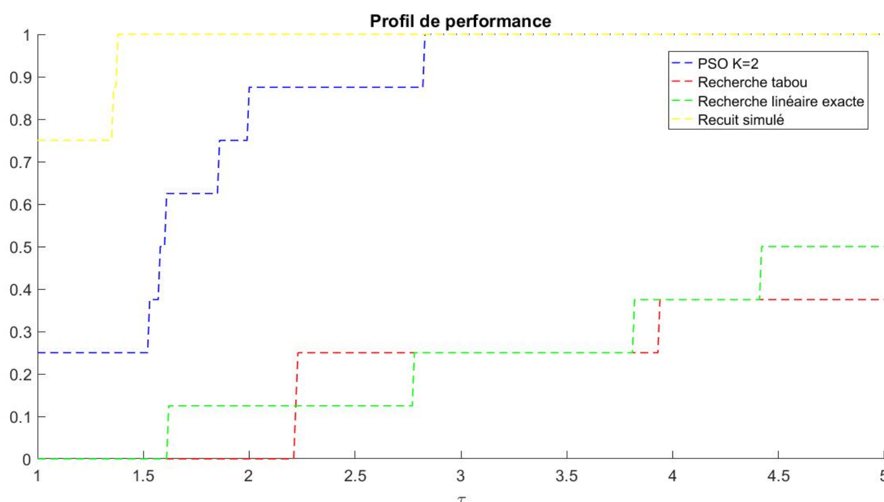

FIGURE 7.1 – Graphique des profils de performance obtenus pour les différentes variantes de l'algorithme PSO retenues, appliqué sur l'ensemble de cas-tests du chapitre 4.

algorithme	$\rho_a(1)$	$\rho_a(\tau_{max})$	meilleur τ
PSO classique	0.25	1	2.83
PSO recherche tabou	0	0.375	3.94
PSO recherche linéaire exacte	0	0.5	4.42
PSO recuit simulé	0.75	1	1.38

TABLE 7.1 – Résultats quantitatifs des profils de performance correspondants au graphique de la Figure 7.1.

résolution de l'ensemble des cas-tests, l'algorithme avec recuit simulé ne demande qu'une limite maximum sur le budget de 1.38 contre 2.83 pour l'algorithme classique, ce qui est une limite plus de 2 fois plus petite.

Nous avons également utilisé nos outils d'analyse par cas-test pour cette comparaison finale. Les résultats obtenus pour le cas-test Schwefel 2.22 sont visibles sur la Figure 7.2 et dans la Table 7.2. Sur la figure 7.2, dans un premier temps, avant la médiane du nombre d'évaluations de $0.5 * 10^5$, nous pouvons observer que l'algorithme le plus rapide est celui qui utilise le recuit simulé, suivi par l'algorithme classique, l'algorithme avec recherche tabou et enfin, l'algorithme avec recherche linéaire exacte. Dans un second temps, nous pouvons voir que l'algorithme qui s'arrête en premier est l'algorithme avec recuit simulé, suivi par l'algorithme classique et les algorithmes avec recherche locale.Dans la Table 7.2, au niveau des taux de réussite, nous pouvons constater que seuls l'algorithme classique et l'algorithme avec recuit simulé n'ont rencontrés aucun échec dans la résolution du cas-test. Parmi les algorithmes avec recherche locale, le meilleur à ce niveau est l'algorithme avec recherche linéaire exacte avec 98% de réussites contre 14% seulement pour l'algorithme avec recherche tabou. Enfin, au niveau des valeurs $fitness_{min}$ et $eval_{best}$, les algorithmes les plus performants restent les algorithmes classique et avec recuit simulé puisqu'ils atteignent des valeurs $fitness_{min}$ inférieures à 10^{-6} pour des $eval_{best}$ de l'ordre de 10^4. Les algorithmes avec recherche locale quant à eux ont tous deux besoin de plus d'évaluations de fonction objectif (de l'ordre de 10^5 pour les deux algorithmes) pour atteindre leur valeur $fitness_{min}$. De plus, dans le cas de la recherche tabou cette valeur est moins précise que l'algorithme PSO classique.

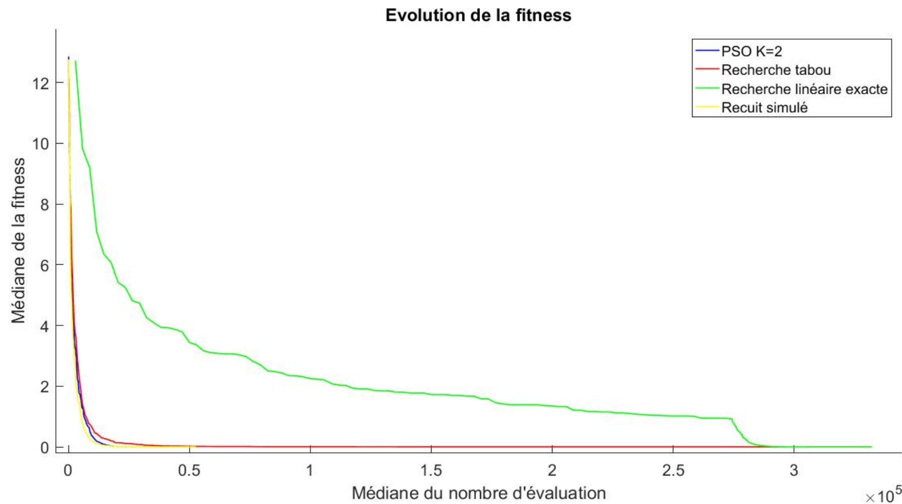

FIGURE 7.2 – Évolution de la valeur de fitness associée à la fonction Schwefel 2.22. Les différentes courbes correspondent aux algorithmes testés dans cette section : L'algorithme PSO classique à 3 liens maximum dans la topologie, l'algorithme avec une méthode de recherche locale de type recherche tabou, l'algorithme avec recherche locale de type recherche linéaire exacte et l'algorithme avec une méthode type recuit simulé dans la mise à jour des vitesses.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$PSO\ K = 2$	1	8.7167*10^{-6}	45190
Recherche tabou	0.14	7.3087*10^{-4}	331743
Recherche linéaire exacte	0.98	9.2215*10^{-6}	3.1214*10^{5}
Recuit simulé	1	9.0448*10^{-6}	5.2736*10^{4}

TABLE 7.2 – Résultats quantitatifs associés au graphique de la Figure 7.2

De cette analyse, il ressort alors que l'algorithme le plus performant de manière globale et sur le cas-test Schwefel 2.22 est l'algorithme qui utilise le recuit simulé. Il constitue donc naturellement une amélioration de l'algorithme classique puisque nous avons pu constater qu'il fait mieux que l'algorithme classique à tous niveaux.En ce qui concerne les algorithmes avec recherche locale, au niveau des deux méthodes de comparaison, nous pouvons affirmer que l'algorithme avec recherche linéaire exacte est le plus performant des deux. Cependant, lorsque nous les comparons à l'algorithme standard, nous nous rendons compte qu'ils ne font pas mieux que l'algorithme standard. Les résultats sur le cas-test Schwefel 2.22 ont montré que les algorithmes avec recherche locale peuvent tout de même fournir une solution de bonne qualité, à condition d'avoir un budget suffisant. Les algorithmes demandaient en effet 10 fois plus de budget pour atteindre la valeur $fitness_{min}$ de la Table 7.2. Le côté trop coûteux des algorithmes avec recherche locale se ressent aussi au niveau du profil de performance de la Figure 7.1 dans lequel les deux algorithmes sont les moins globalement performants.

Interprétation des résultats

Au regard des résultats que nous avons pu obtenir, il est clair que l'algorithme qui est le plus performant est celui qui utilise le recuit simulé dans la mise à jour des vitesses. Cela s'explique par le fait que le choix réfléchi, que nous avons ajouté dans la mise à jour des vitesses, permet à l'algorithme d'améliorer ses résultats pour un budget raisonnable. En effet, notre modification ne demande que 2 évaluations de fonction supplémentaire à chaque itération. Ces 2 évaluations sont donc vite rentabilisées si l'algorithme se rapproche plus vite du minimum global. Nous pouvons alors considérer l'algorithme avec recuit simulé comme une réelle amélioration de l'algorithme PSO.

En ce qui concerne les algorithme qui utilisent une recherche locale, nous pensions qu'une optimisation de la taille du nouveau vecteur vitesse aurait pu significativement améliorer les performances de l'algorithme. Cependant, les résultats obtenus nous forcent à constater qu'il n'en est rien. Nous pensons que cela s'explique par le fait que ces optimisations de la taille du vecteur vitesse sont trop coûteuses en terme d'évaluation de fonction. De plus, nous pensons aussi que l'optimisation systématique de la taille du vecteur vitesse n'offre finalement pas un bon compromis entre l'exploitation et l'exploration. Dans le chapitre 6, nous avons de fait pu observer des phénomènes de stagnation dans des optima locaux.Nous serions donc d'avis de dire que ces deux algorithmes, bien qu'ils puissent permettre d'obtenir une solution, ne constituent pas de réelles améliorations de l'algorithme classique.

Conclusion et perspectives

Dans le cadre de ce mémoire sur l'algorithme d'optimisation par essaim de particules, nous avons commencé par aborder le concept de méthodes heuristiques que nous avons illustré avec les algorithmes génétiques et avec l'algorithme du recuit simulé. Nous avons également pu présenter l'algorithme d'optimisation par essaim de particules que nous désirions améliorer. Nous en avons présenté une version standard ainsi que certaines variantes couramment utilisées.

Nous avons ensuite pu expliciter l'ensemble des méthodes que nous considérions comme améliorations potentielles de l'algorithme PSO. Dans nos améliorations, nous avons envisagé une méthode qui se sert du recuit simulé dans l'étape de mise à jour des vitesses des particules ainsi qu'une méthode basée sur une optimisation de la taille du vecteur vitesse servant à la mise à jour de la position. Pour calculer ce meilleur vecteur vitesse, nous avons envisagé l'utilisation d'une heuristique de type recherche tabou ainsi que l'utilisation d'une recherche linéaire exacte.

Pour comparer nos méthodes entre elles, nous avons commencé par définir une version de l'algorithme PSO comme base de comparaison et nous avons sélectionné les meilleurs paramètres dans nos nouvelles méthodes. Ces choix ont tous été effectués sur base d'un profil de performance de Dolan et Moré ou sur base d'une étude numérique sur nos cas-tests pris individuellement. De ces tests numériques, nous avons pu obtenir 3 candidats au titre d'amélioration de l'algorithme PSO. Les algorithmes en lice étaient l'algorithme qui utilise une recherche locale de type recherche tabou pour la mise à jour des vitesse avec 3 essais autorisés pour obtenir une amélioration de la position, l'algorithme qui utilise une recherche locale de type recherche linéaire exacte pour la mise à jour des vitesse (avec un retour à la méthode classique en cas de stagnation), ainsi que l'algorithme inspiré du recuit simulé avec $T = 1$ et $COOL = 0.75$.

Nous avons finalement pu comparer nos candidats et nous en avons tiré que l'algorithme le plus globalement performant est l'algorithme qui utilise le recuit simulé. Nous pensons donc qu'il est bel et bien une amélioration de l'algorithme classique. Les deux autres algorithmes quant à eux n'ont pas fourni de résultats suffisamment probants pour qu'ils soient considérés comme des améliorations de l'algorithme classique.

Précisons tout de même que les résultats de ce mémoire ne sont pas à prendre à titre de vérité absolue. En effet, l'algorithme PSO est une méthode basée sur l'aléatoire. Que ce soit dans le choix de nos candidats ou dans notre comparaison finale, nous avons donc très bien pu observer de très chanceuses ou malchanceuses exécutions des algorithmes. D'autant plus que nous n'avons travaillé qu'en exécutant 50 fois les algorithmes sur chaque cas-test. Une première perspective d'amélioration de ce mémoire serait alors d'effectuer des batteries de tests a très grande échelle pour minimiser l'influence de l'aléatoire dans les résultats.

Une autre limite de notre étude est le choix que nous avons dû effectuer pour l'algorithme

PSO que nous avons pris comme base de développement. Il serait également intéressant de tester les méthodes que nous avons développées pour d'autres variantes de l'algorithme PSO.

Comme perspectives d'amélioration de notre mémoire, nous pensons tout de même qu'il n'est pas intéressant de chercher plus loin au niveau des améliorations basées sur la recherche locale pour l'optimisation de la taille du vecteur vitesse. En effet, nous pensons que ces méthodes seront toujours trop coûteuses pour fournir des améliorations trop peu significatives. En revanche, chercher à trouver des méthodes qui choisissent intelligemment les vitesses, nous semble être une voie porteuse pour mener à d'autres améliorations. Nous n'avons, dans ce mémoire, développé qu'une méthode de ce type, il pourrait donc y en avoir d'autres plus intéressantes encore. Une autre option pouvant mener à des améliorations de l'algorithme serait de chercher des optimisations de l'algorithme au niveau de la topologie, possibilité que nous n'avons pas considérée dans ce mémoire.

Nous terminerons finalement ce mémoire en posant les questions suivantes :

A partir de quel moment peut-on considérer qu'une méthode heuristique est aboutie ?

Dans quelle mesure les algorithmes plusieurs fois modifiés gardent-t-ils leur nature initiale ?

Bibliographie

[1] S. Arif, C. Abdelghani, and A. Hellal. Optimisation par essaim de particules appliquée à l'écoulement optimal de puissance réactive. https ://www.researchgate.net/publication/251012942 , 2007.

[2] G. Berthiau and P. Siarry. Etat de l'art des méthodes d'optimisation globale. *RAIRO Operations Research*, pages 329–365, 2001.

[3] C. Blum and A. Roli. Metaheuristics in combinatorial optimization : Overviewand conceptual comparison. *ACM Computing Surveys*, 35 :268–308, 2003.

[4] M. Clerc. Standard particle swarm optimisation from 2006 to 2011. *hal-00764996*, 2012.

[5] M. Clerc and J. Kennedy. The particle swarm—explosion, stability, and convergence in a multidimensional complex space. *IEEE transactions on evolutionary computation*, 6 :58–73, 2002.

[6] M. Clerc and P. Siarry. Une nouvelle métaheuristique pour l'optimisation difficile : la méthode des essaims particulaires. http ://www.j3ea.org, 2004.

[7] M. Clerc, M. Zambrano-Bigiarini, and R. Rojas. Standard particle swarm optimisation 2011 at cec-2013 : A baseline for future pso improvements. *IEEE Congress on Evolutionary Computation*, pages 2337–2344, 2013.

[8] E.D Dolan and J.J Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2) :201–213, 2002.

[9] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theorys. *IEEE Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[10] R. Eberhart and J. Kennedy. Particle swarm optimization. *IEEE*, pages 1942–1948, 1995.

[11] R. Eberhart and Y. Shi. Empirical study of particle swarm optimization. *IEEE*, 1999.

[12] R. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. *IEEE*, 2000.

[13] R. Eberhart and Y. Shi. Tracking and optimizing dynamic systems with particle swarms. *IEEE*, 2001.

[14] R. Eberhart, P. Simpson, and R. Dobbins. Computational intelligence pc tools. *AP Professional*, 1996.

[15] A. El Dor. Perfectionnement des algorithmes d'optimisation par essaim particulaire : applications en segmentation d'images et en électronique. *Université Paris-Est*, 2012.

[16] A. El Moubarek Bouzid. Optimisation par la méthode des essaims particulaires d'une fonction trigonométrique. https ://www.researchgate.net/publication/331354576 , 2010.

[17] S. Helwig and R. Wanka. Theoretical analysis of initial particle swarm behavior. *PPSN*, page 889–898, 2008.

[18] J. Kennedy and R. Mendes. Population structure and particle swarm performance. *IEEE*, 2002.

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

[20] W. B. Langdon and R. Poli. Evolving problems to learn about particle swarm optimizers and other search algorithms. *IEEE transactions on evolutionary computation*, 11 :561–578, 2007.

[21] R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm : Simpler, maybe better. *IEEE*, 2004.

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

[23] S. Pool Marquez. Adaptation de la « one fifth success rule » pour le réglage de la taille de la population dans l'algorithme génétique du logiciel minamo. *Universite de Namur*, 2021.

[24] C. W. Reynolds. Flocks, herds, and schools : A distributed behavioral model. *Computer Graphics*, 12 :25–34, 1987.

[25] N. Roy. Mise au point d'essaims de particules proactifs au moyen de la logique floue. *Universite de Namur*, 2019.

[26] Liu Y. Yao, X. and G. Lin. Evolutionary programming made faster. *IEEE Transactions on evolutionary computation*, 3(2) :82–102, 1999.

[27] M. Zemzami, N. Elhami, A. Makhloufi, M. Itmi, and N. Hmina. Application d'un modèle parallèle de la méthode pso au problème de transport d'électricité. *ISTE Science Publishing*, 2016.

Annexe

Cas-test Rosenbrock

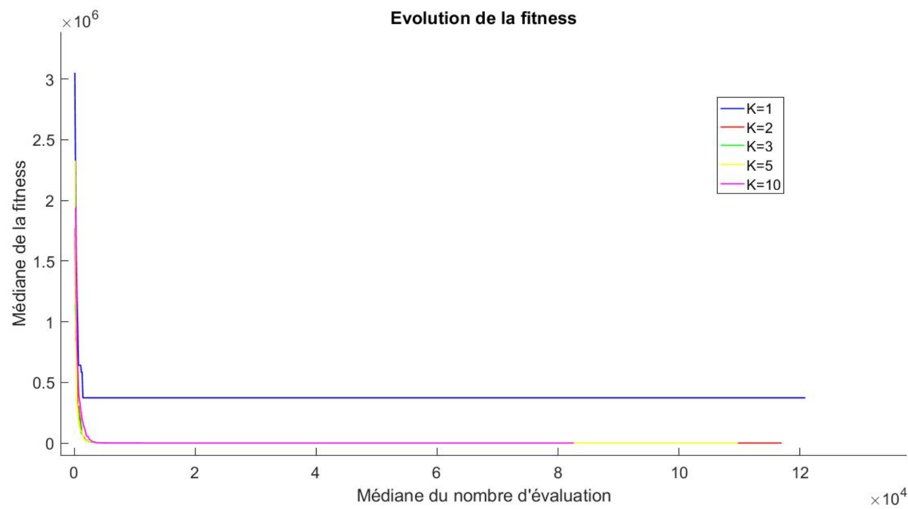

FIGURE 7.3 – Évolution de la valeur de fitness associée à la fonction Rosenbrock. Test des algorithmes avec différents nombres de liens maximum autorisé dans la topologie.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$K = 1$	0	$3.7247*10^5$	120879
$K = 2$	0.68	$\mathcal{O}(10^{-6})$	52743
$K = 3$	0.80	$\mathcal{O}(10^{-6})$	44278
$K = 5$	0.60	$\mathcal{O}(10^{-6})$	49720
$K = 10$	0.76	$\mathcal{O}(10^{-6})$	88232

TABLE 7.3 – Résultats quantitatifs associés au graphique de la Figure 7.3

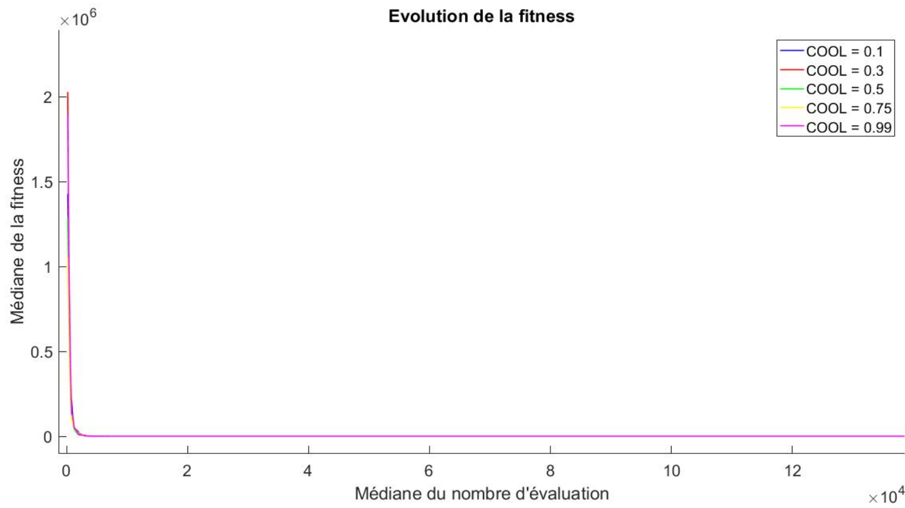

Figure 7.4 – Évolution de la valeur de fitness associée à la fonction Rosenbrock. Test des différentes valeurs du paramètre $COOL$.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$COOL = 0.1$	0.68	$\mathcal{O}(10^{-6})$	112477
$COOL = 0.3$	0.76	$\mathcal{O}(10^{-6})$	122253
$COOL = 0.5$	0.72	$\mathcal{O}(10^{-6})$	129840
$COOL = 0.75$	0.84	$\mathcal{O}(10^{-6})$	105059
$COOL = 0.99$	0.76	$\mathcal{O}(10^{-6})$	138521

Table 7.4 – Résultats quantitatifs associés au graphique de la Figure 7.4

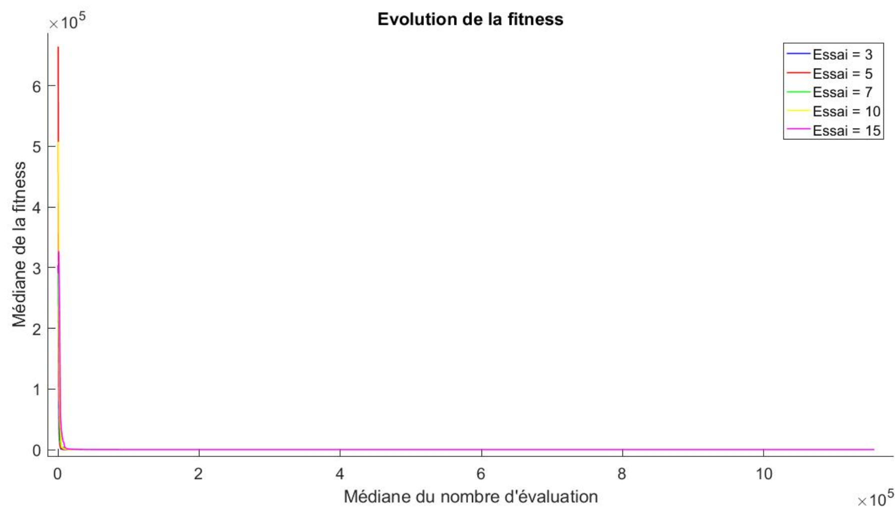

Figure 7.5 – Évolution de la valeur de fitness associée à la fonction Rosenbrock. Test de différents nombres d'essais dans la recherche tabou.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$Essai = 3$	0	0.9275	$3.36*10^5$
$Essai = 5$	0.1	1.0882	$4.76*10^5$
$Essai = 7$	0	5.7497	614108
$Essai = 10$	0	4.2410	$8.28*10^5$
$Essai = 15$	0	4.0542	1156902

TABLE 7.5 – Résultats quantitatifs associés au graphique de la Figure 7.5

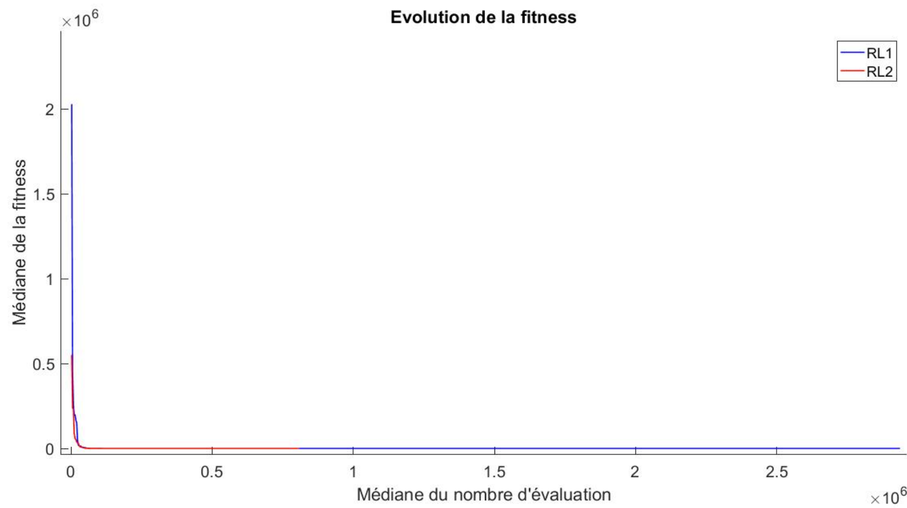

FIGURE 7.6 – Évolution de la valeur de fitness associée à la fonction Rosenbrock. Test des algorithmes avec recherche linéaire exacte. RL1 est la version ou nous n'utilisons que la recherche linéaire exacte. RL2 est la version ou nous recalculons classiquement les vitesses en cas de stagnation.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
RL 1	0	$3.8983*10^{-4}$	2937053
RL 2	0.4	$3.9308*10^{-4}$	809657

TABLE 7.6 – Résultats quantitatifs associés au graphique de la Figure 7.6

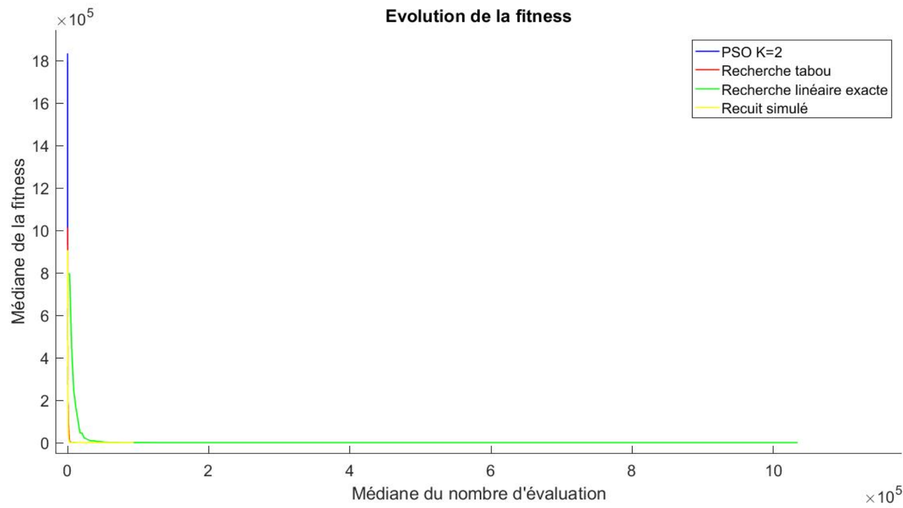

Figure 7.7 – Évolution de la valeur de fitness associée à la fonction Rosenrock. Test des différents algorithmes du chapitre 7.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$PSO\ K=2$	0	0.3060	149801
Recherche tabou	0.1	1.8777	3.2765*10^5
Recherche linéaire exacte	0.76	$\mathcal{O}(10^{-6})$	1034662
Recuit simulé	0.8	$\mathcal{O}(10^{-6})$	93543

Table 7.7 – Résultats quantitatifs associés au graphique de la Figure 7.7

Cas-test Ackley

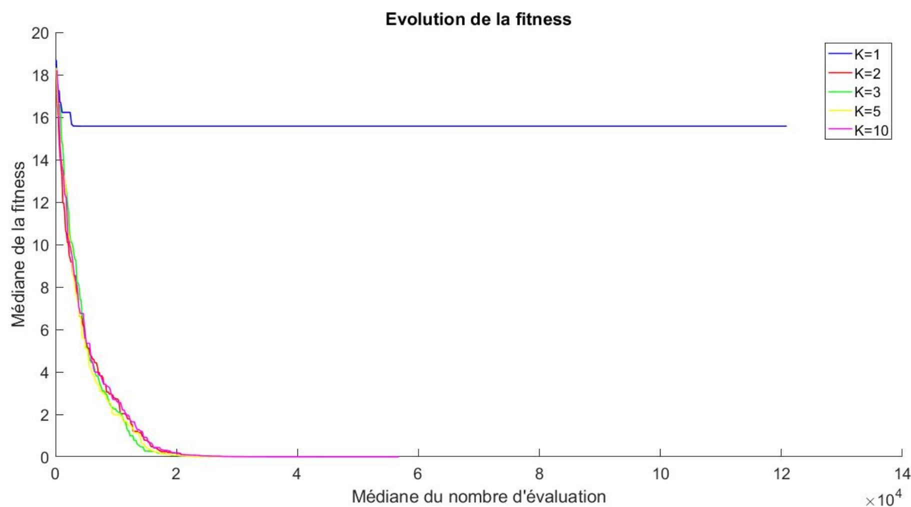

Figure 7.8 – Évolution de la valeur de fitness associée à la fonction Ackley. Test des algorithmes avec différents nombres de liens maximum autorisé dans la topologie.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$K = 1$	0	15.5754	120879
$K = 2$	0.76	$\mathcal{O}(10^{-6})$	52755
$K = 3$	0.92	$\mathcal{O}(10^{-6})$	49973
$K = 5$	0.96	$\mathcal{O}(10^{-6})$	50860
$K = 10$	0.96	$\mathcal{O}(10^{-6})$	56711

TABLE 7.8 – Résultats quantitatifs associés au graphique de la Figure 7.8

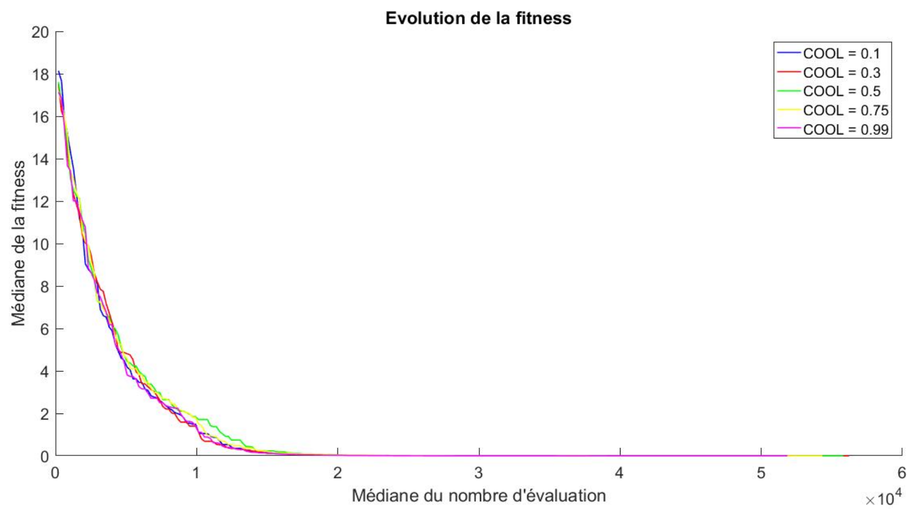

FIGURE 7.9 – Évolution de la valeur de fitness associée à la fonction Ackley. Test des différentes valeurs du paramètre $COOL$.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$COOL = 0.1$	0.96	$\mathcal{O}(10^{-6})$	54681
$COOL = 0.3$	1	$\mathcal{O}(10^{-6})$	56232
$COOL = 0.5$	0.88	$\mathcal{O}(10^{-6})$	55812
$COOL = 0.75$	0.88	$\mathcal{O}(10^{-6})$	54339
$COOL = 0.99$	0.92	$\mathcal{O}(10^{-6})$	51852

TABLE 7.9 – Résultats quantitatifs associés au graphique de la Figure 7.9

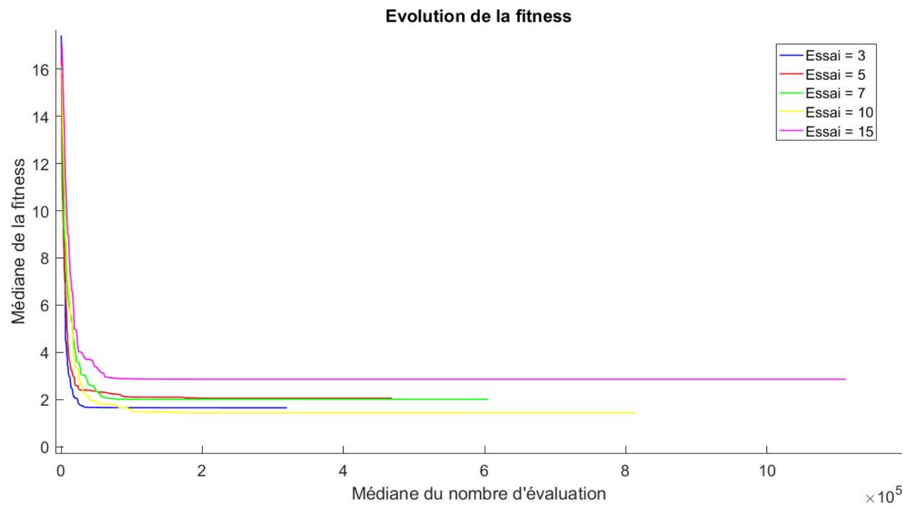

Figure 7.10 – Évolution de la valeur de fitness associée à la fonction Ackley. Test de différents nombres d'essais dans la recherche tabou.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$Essai = 3$	0	1.6465	$3.13*10^5$
$Essai = 5$	0.3	2.0481	$4.68*10^5$
$Essai = 7$	0.1	2.0092	$6.04*10^5$
$Essai = 10$	0	1.4332	$8.14*10^5$
$Essai = 15$	0	2.8568	1111611

Table 7.10 – Résultats quantitatifs associés au graphique de la Figure 7.10

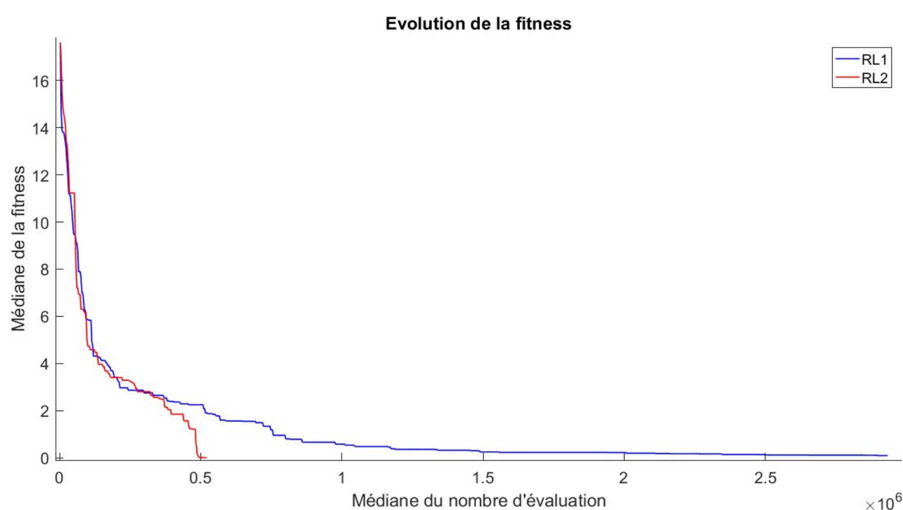

Figure 7.11 – Évolution de la valeur de fitness associée à la fonction Ackley. Test des algorithmes avec recherche linéaire exacte. RL1 est la version ou nous n'utilisons que la recherche linéaire exacte. RL2 est la version ou nous recalculons classiquement les vitesses en cas de stagnation.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
RL 1	0	0.0870	2933551
RL 2	1	$\mathcal{O}(10^{-6})$	520913

TABLE 7.11 – Résultats quantitatifs associés au graphique de la Figure 7.11

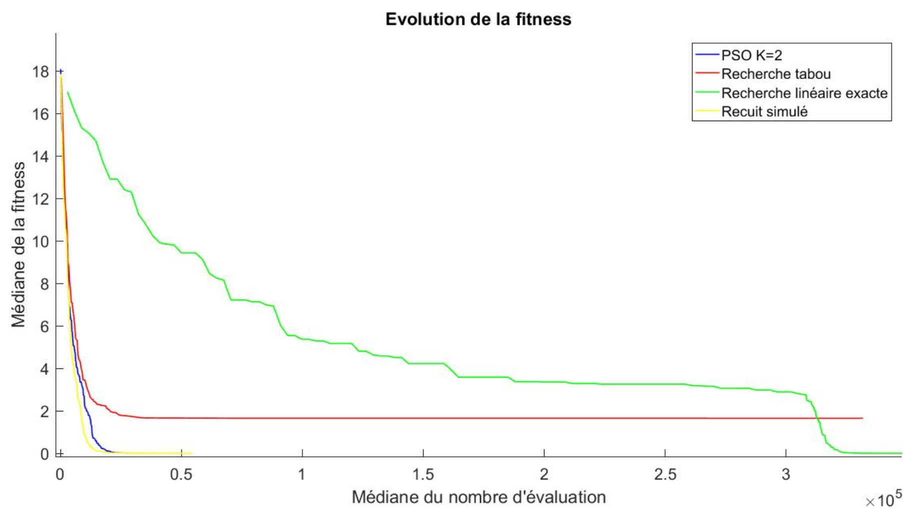

FIGURE 7.12 – Évolution de la valeur de fitness associée à la fonction Ackley. Test des différents algorithmes du chapitre 7.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$PSO\ K = 2$	1	$\mathcal{O}(10^{-6})$	49969
Recherche tabou	0.1	1.6464	331722
Recherche linéaire exacte	1	$\mathcal{O}(10^{-6})$	349800
Recuit simulé	1	$\mathcal{O}(10^{-6})$	54652

TABLE 7.12 – Résultats quantitatifs associés au graphique de la Figure 7.12

Cas-test Elliptique

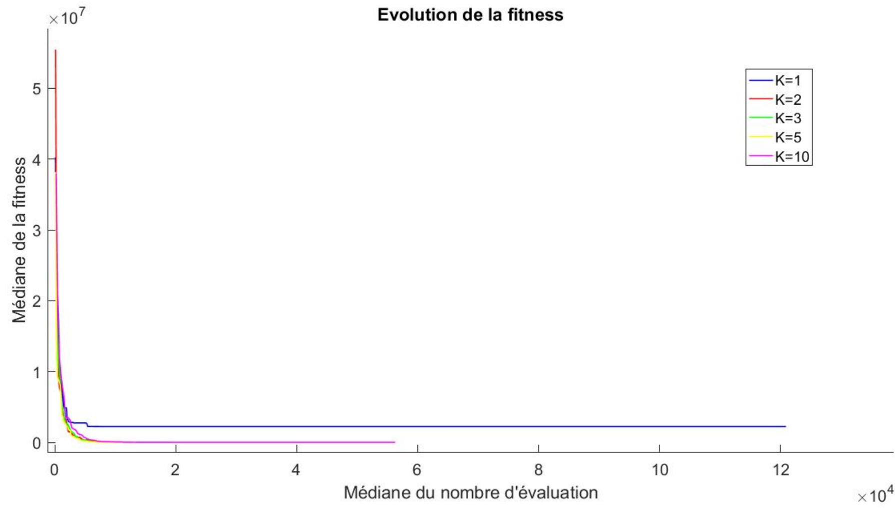

Figure 7.13 – Évolution de la valeur de fitness associée à la fonction Elliptique. Test des algorithmes avec différents nombres de liens maximum autorisé dans la topologie.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$K = 1$	0	$2.2226{*}10^6$	120879
$K = 2$	1	$\mathcal{O}(10^{-6})$	50367
$K = 3$	1	$\mathcal{O}(10^{-6})$	45230
$K = 5$	1	$\mathcal{O}(10^{-6})$	46760
$K = 10$	1	$\mathcal{O}(10^{-6})$	56285

Table 7.13 – Résultats quantitatifs associés au graphique de la Figure 7.13

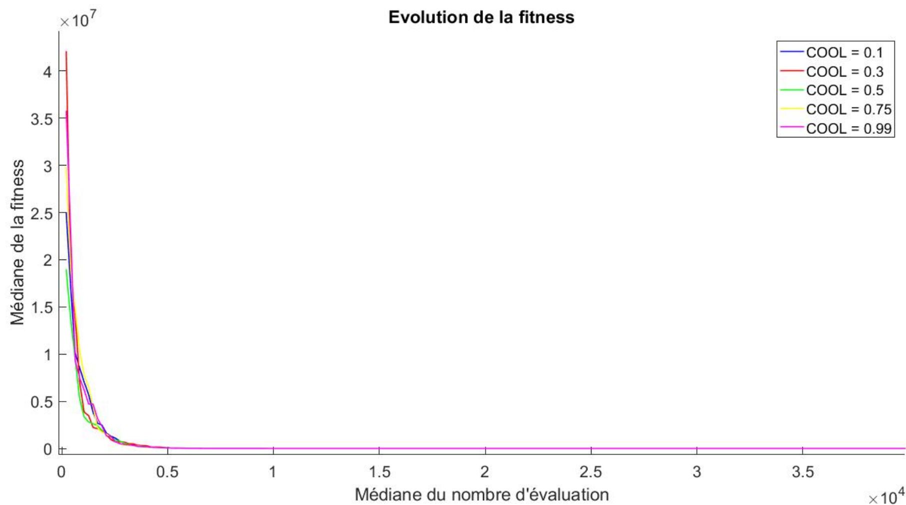

Figure 7.14 – Évolution de la valeur de fitness associée à la fonction Elliptique. Test des différentes valeurs du paramètre $COOL$.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$COOL = 0.1$	1	$\mathcal{O}(10^{-6})$	39003
$COOL = 0.3$	1	$\mathcal{O}(10^{-6})$	38950
$COOL = 0.5$	1	$\mathcal{O}(10^{-6})$	39479
$COOL = 0.75$	1	$\mathcal{O}(10^{-6})$	37965
$COOL = 0.99$	1	$\mathcal{O}(10^{-6})$	39868

TABLE 7.14 – Résultats quantitatifs associés au graphique de la Figure 7.14

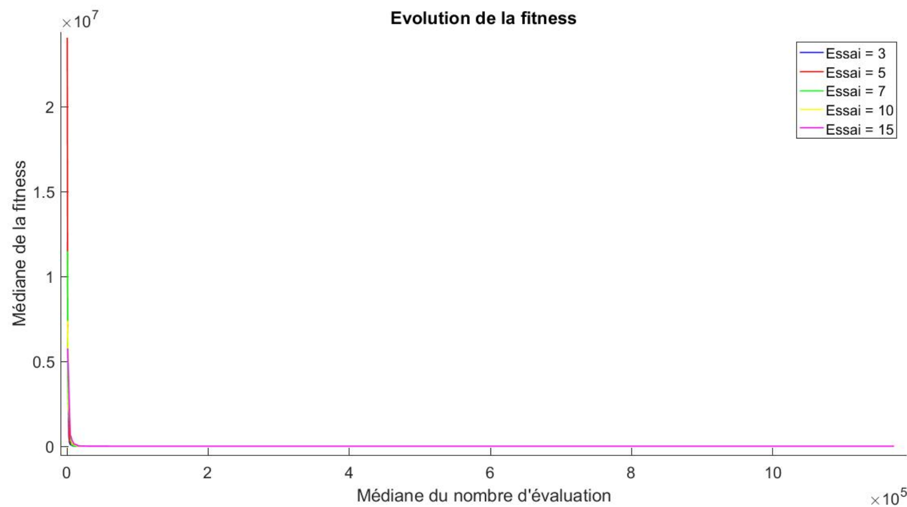

FIGURE 7.15 – Évolution de la valeur de fitness associée à la fonction Elliptique. Test de différents nombres d'essais dans la recherche tabou.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$Essai = 3$	0.32	0.0848	317860
$Essai = 5$	0.3	0.0224	476302
$Essai = 7$	0.16	2.1076	$6.19*10^5$
$Essai = 10$	0.1	8.6293	$8.29*10^5$
$Essai = 15$	0	28.7983	1172127

TABLE 7.15 – Résultats quantitatifs associés au graphique de la Figure 7.15

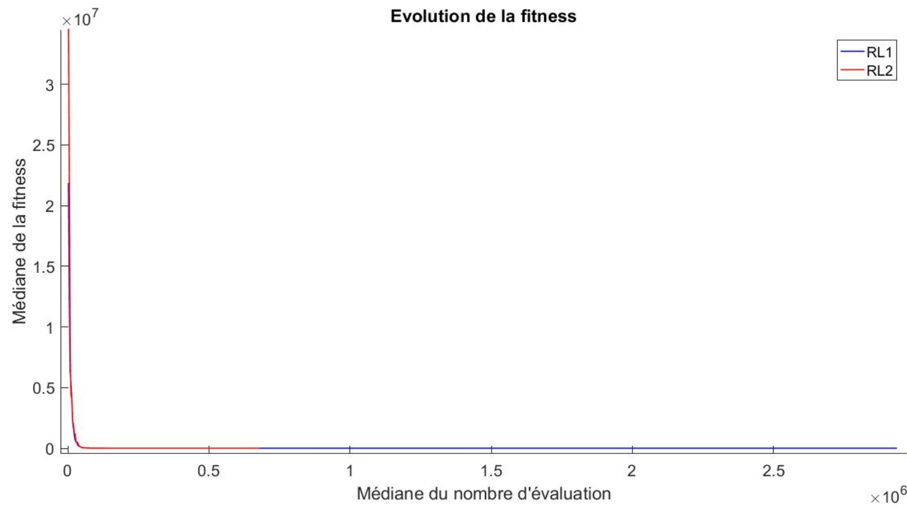

Figure 7.16 – Évolution de la valeur de fitness associée à la fonction Elliptique. Test des algorithmes avec recherche linéaire exacte. RL1 est la version ou nous n'utilisons que la recherche linéaire exacte. RL2 est la version ou nous recalculons classiquement les vitesses en cas de stagnation.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
RL 1	0	$9.6608{*}10^{-4}$	$2.9372{*}10^{6}$
RL 2	1	$\mathcal{O}(10^{-6})$	$6.7906{*}10^{5}$

Table 7.16 – Résultats quantitatifs associés au graphique de la Figure 7.16

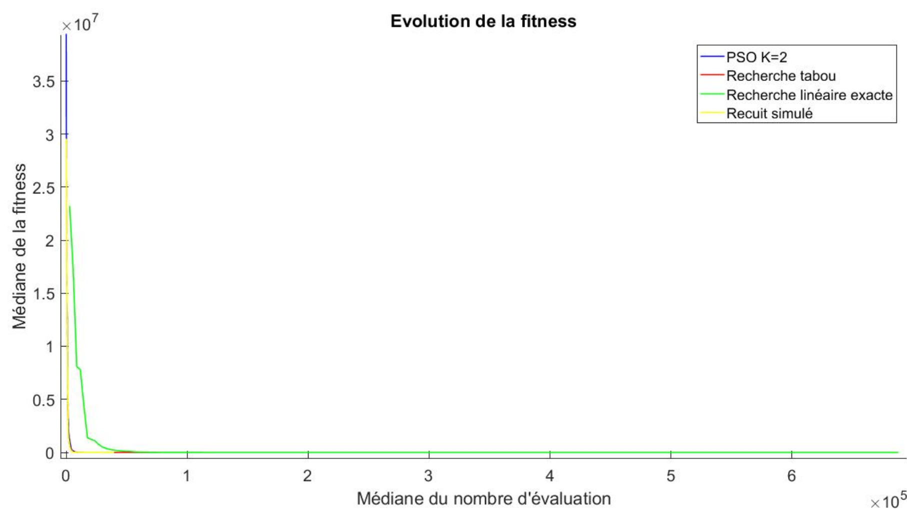

Figure 7.17 – Évolution de la valeur de fitness associée à la fonction Elliptique. Test des différents algorithmes du chapitre 7.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$PSO\ K = 2$	1	$\mathcal{O}(10^{-6})$	45390
Recherche tabou	0.2	0.0035	3.3096*10^5
Recherche linéaire exacte	1	$\mathcal{O}(10^{-6})$	6.8793*10^5
Recuit simulé	1	$\mathcal{O}(10^{-6})$	3.9656*10^4

TABLE 7.17 – Résultats quantitatifs associés au graphique de la Figure 7.17

Cas-test Rastrigin

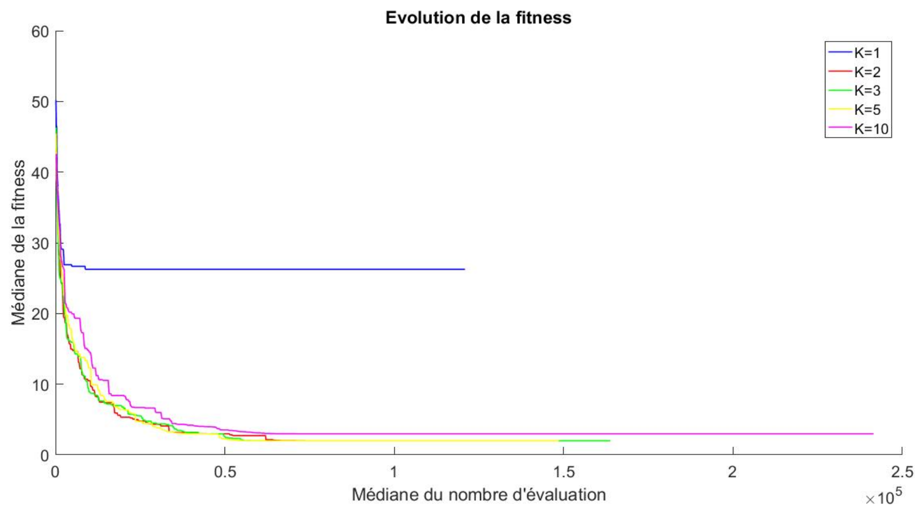

FIGURE 7.18 – Évolution de la valeur de fitness associée à la fonction Rastrigin. Test des algorithmes avec différents nombres de liens maximum autorisé dans la topologie.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$K = 1$	0	26.2480	120879
$K = 2$	0.12	1.9899	148965
$K = 3$	0.08	1.9899	163782
$K = 5$	0.04	1.9899	148559
$K = 10$	0.04	2.9849	241470

TABLE 7.18 – Résultats quantitatifs associés au graphique de la Figure 7.18

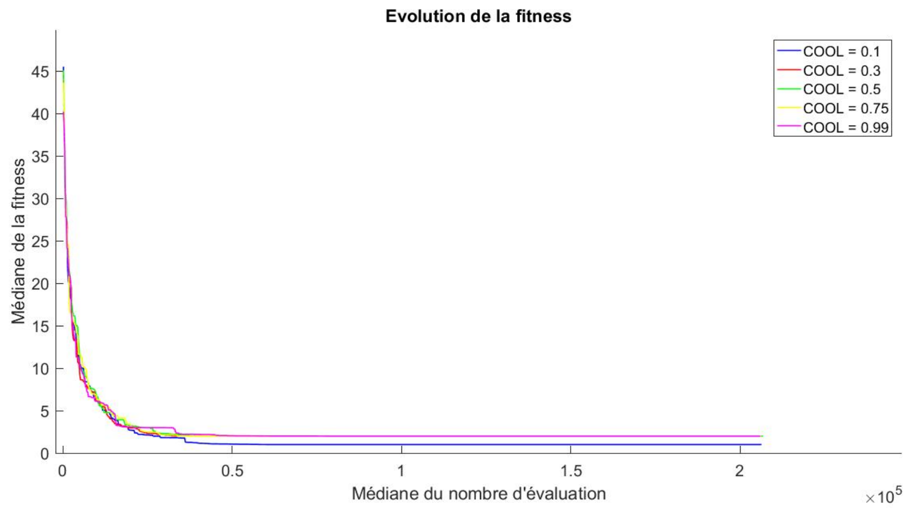

Figure 7.19 – Évolution de la valeur de fitness associée à la fonction Rastrigin. Test des différentes valeurs du paramètre $COOL$.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$COOL = 0.1$	0.12	0.9950	206255
$COOL = 0.3$	0.08	1.9899	205755
$COOL = 0.5$	0.08	1.9899	206691
$COOL = 0.75$	0.08	1.9899	206156
$COOL = 0.99$	0.04	1.9899	205939

Table 7.19 – Résultats quantitatifs associés au graphique de la Figure 7.19

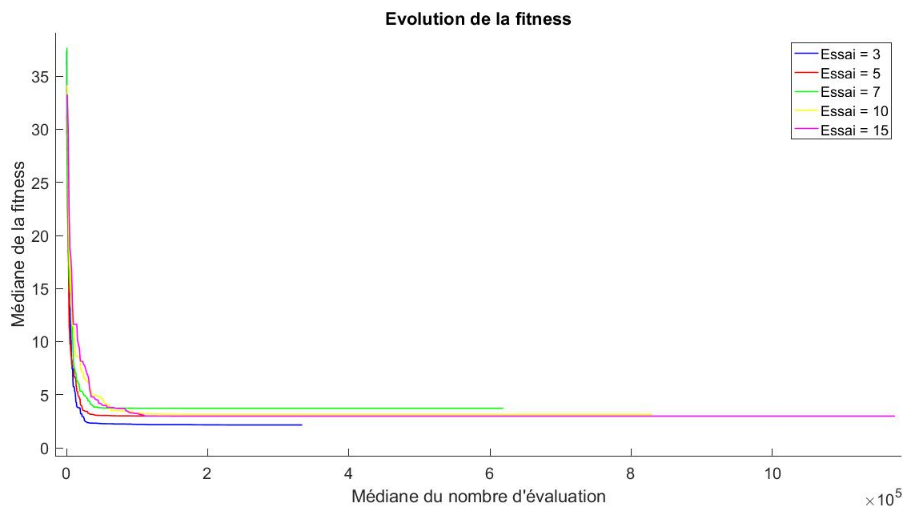

Figure 7.20 – Évolution de la valeur de fitness associée à la fonction Rastrigin. Test de différents nombres d'essais dans la recherche tabou.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$Essai = 3$	0	2.1469	$3.34*10^5$
$Essai = 5$	0	2.9849	473362
$Essai = 7$	0	3.7182	$6.19*10^5$
$Essai = 10$	0	3.1467	$8.30*10^5$
$Essai = 15$	0	2.9870	1174047

TABLE 7.20 – Résultats quantitatifs associés au graphique de la Figure 7.20

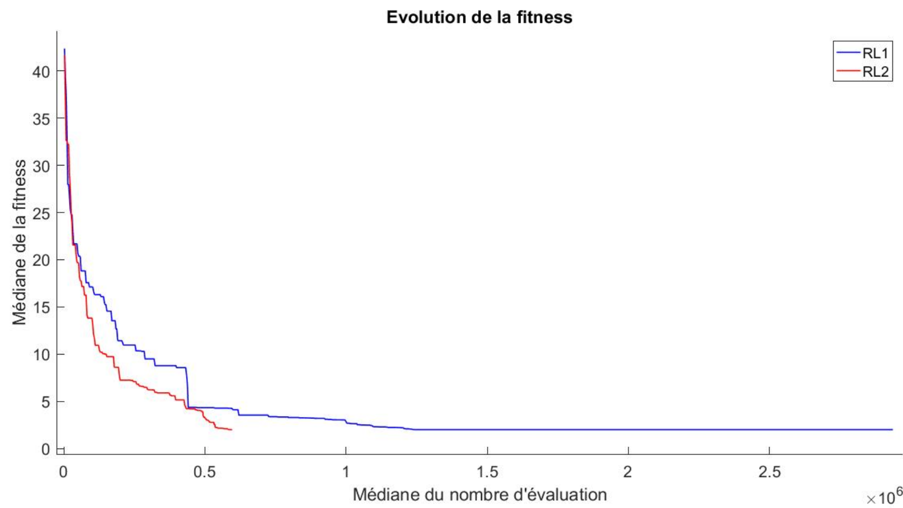

FIGURE 7.21 – Évolution de la valeur de fitness associée à la fonction Rastrigin. Test des algorithmes avec recherche linéaire exacte. RL1 est la version ou nous n'utilisons que la recherche linéaire exacte. RL2 est la version ou nous recalculons classiquement les vitesses en cas de stagnation.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
RL 1	0	1.9900	$2.9371*10^6$
RL 2	0	1.9899	596659

TABLE 7.21 – Résultats quantitatifs associés au graphique de la Figure 7.21

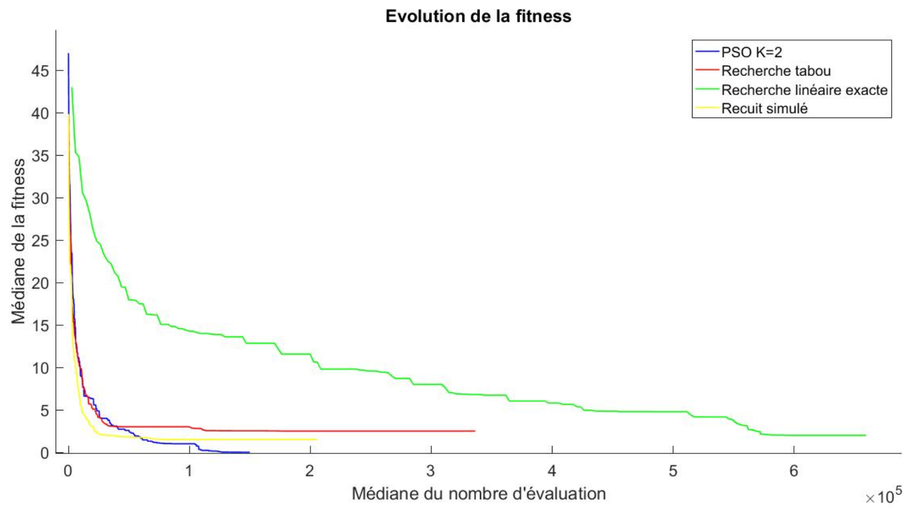

Figure 7.22 – Évolution de la valeur de fitness associée à la fonction Rastrigin. Test des différents algorithmes du chapitre 7.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$PSO\ K=2$	0.46	$3.2794*10^{-5}$	150011
Recherche tabou	0	2.4980	336348
Recherche linéaire exacte	0.06	1.9899	$5.5951*10^5$
Recuit simulé	0.06	1.4924	$2.0567*10^5$

Table 7.22 – Résultats quantitatifs associés au graphique de la Figure 7.22

Cas-test Schwefel 12

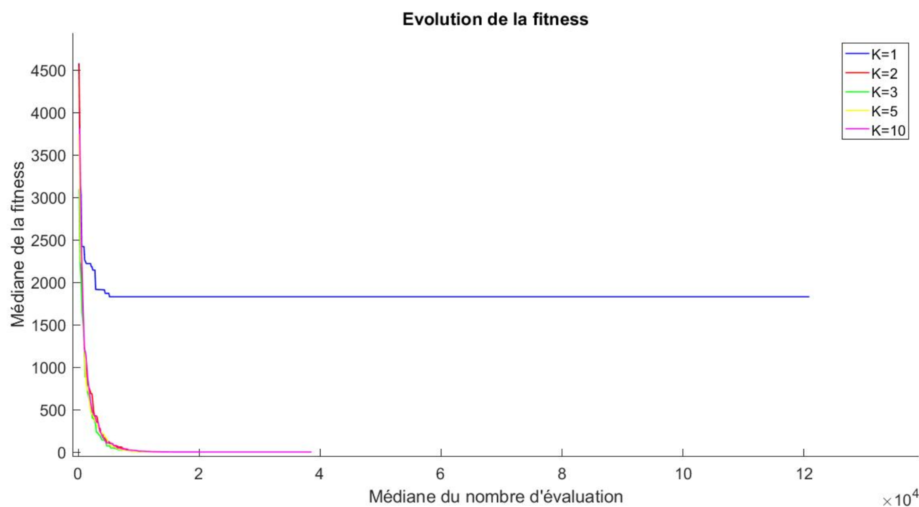

Figure 7.23 – Évolution de la valeur de fitness associée à la fonction Schwefel 12. Test des algorithmes avec différents nombres de liens maximum autorisé dans la topologie.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$K = 1$	0	$1.8278*10^3$	120879
$K = 2$	1	$\mathcal{O}(10^{-6})$	33568
$K = 3$	1	$\mathcal{O}(10^{-6})$	31540
$K = 5$	1	$\mathcal{O}(10^{-6})$	36600
$K = 10$	1	$\mathcal{O}(10^{-6})$	38587

TABLE 7.23 – Résultats quantitatifs associés au graphique de la Figure 7.23

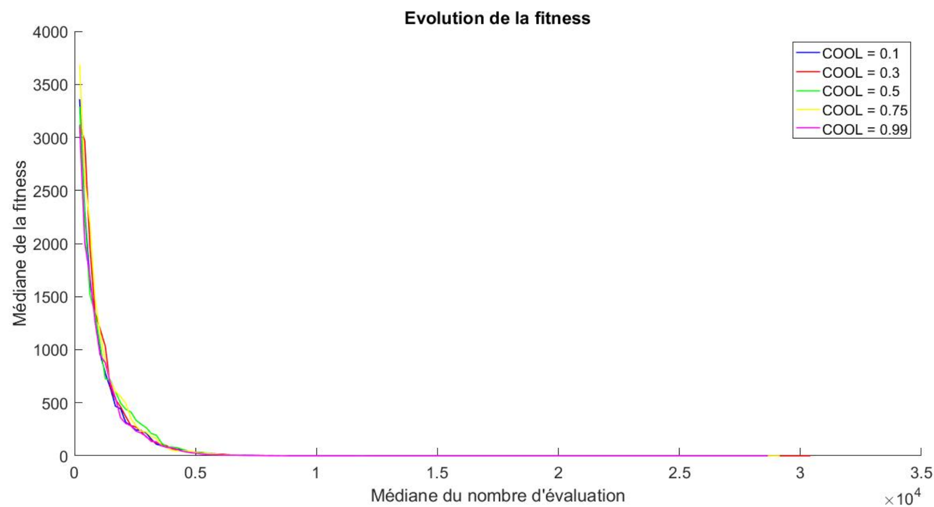

FIGURE 7.24 – Évolution de la valeur de fitness associée à la fonction Schwefel 12. Test des différentes valeurs du paramètre $COOL$.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$COOL = 0.1$	1	$\mathcal{O}(10^{-6})$	29950
$COOL = 0.3$	1	$\mathcal{O}(10^{-6})$	30404
$COOL = 0.5$	1	$\mathcal{O}(10^{-6})$	28775
$COOL = 0.75$	1	$\mathcal{O}(10^{-6})$	29143
$COOL = 0.99$	1	$\mathcal{O}(10^{-6})$	28656

TABLE 7.24 – Résultats quantitatifs associés au graphique de la Figure 7.24

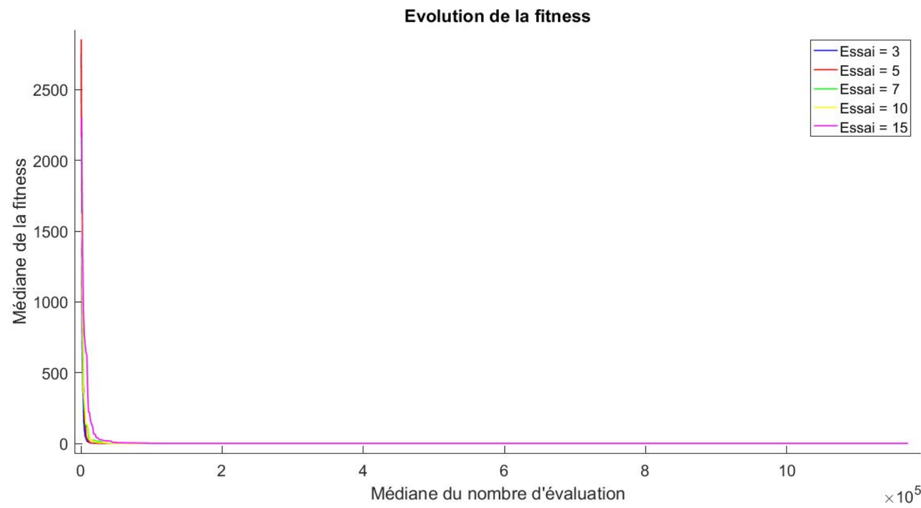

Figure 7.25 – Évolution de la valeur de fitness associée à la fonction Schwefel 12. Test de différents nombres d'essais dans la recherche tabou.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$Essai = 3$	0.5	1.2182*10^{-5}	2.0520*10^5
$Essai = 5$	0.34	0.0037	474917
$Essai = 7$	0	0.0154	618996
$Essai = 10$	0.2	0.0058	8.3031*10^5
$Essai = 15$	0	0.0835	1.1718*10^5

Table 7.25 – Résultats quantitatifs associés au graphique de la Figure 7.25

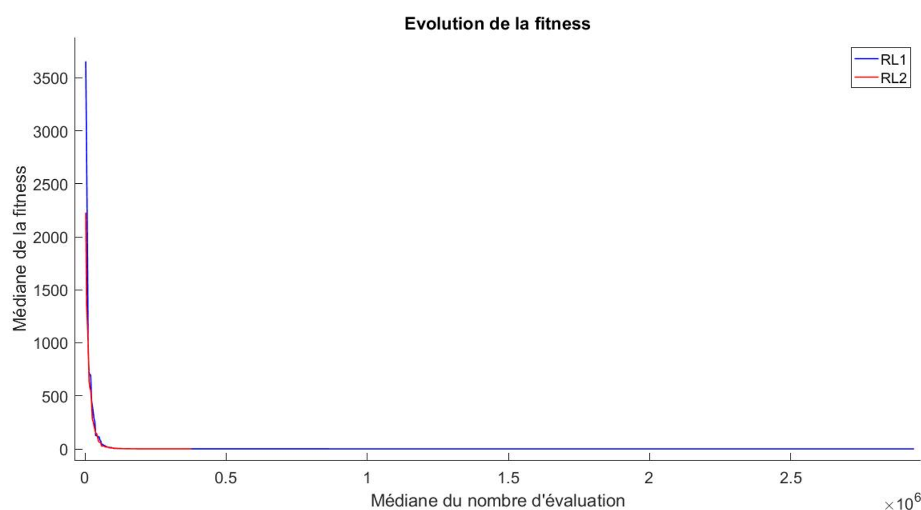

Figure 7.26 – Évolution de la valeur de fitness associée à la fonction Schwefel 12. Test des algorithmes avec recherche linéaire exacte. RL1 est la version ou nous n'utilisons que la recherche linéaire exacte. RL2 est la version ou nous recalculons classiquement les vitesses en cas de stagnation.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
RL 1	0	0.0119	2.9369*10^6
RL 2	1	$\mathcal{O}(10^{-6})$	377658

TABLE 7.26 – Résultats quantitatifs associés au graphique de la Figure 7.26

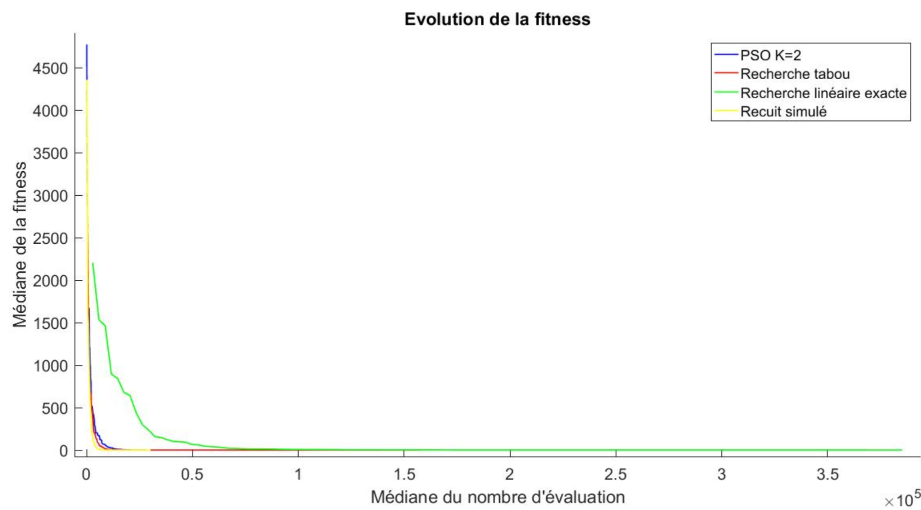

FIGURE 7.27 – Évolution de la valeur de fitness associée à la fonction Schwefel 12. Test des différents algorithmes du chapitre 7.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$PSO\ K = 2$	1	$\mathcal{O}(10^{-6})$	5.6520*10^4
Recherche tabou	0.64	$\mathcal{O}(10^{-6})$	157255
Recherche linéaire exacte	1	$\mathcal{O}(10^{-6})$	3.8523*10^5
Recuit simulé	1	$\mathcal{O}(10^{-6})$	3.02877*10^4

TABLE 7.27 – Résultats quantitatifs associés au graphique de la Figure 7.27

Cas-test Schwefel 2.21

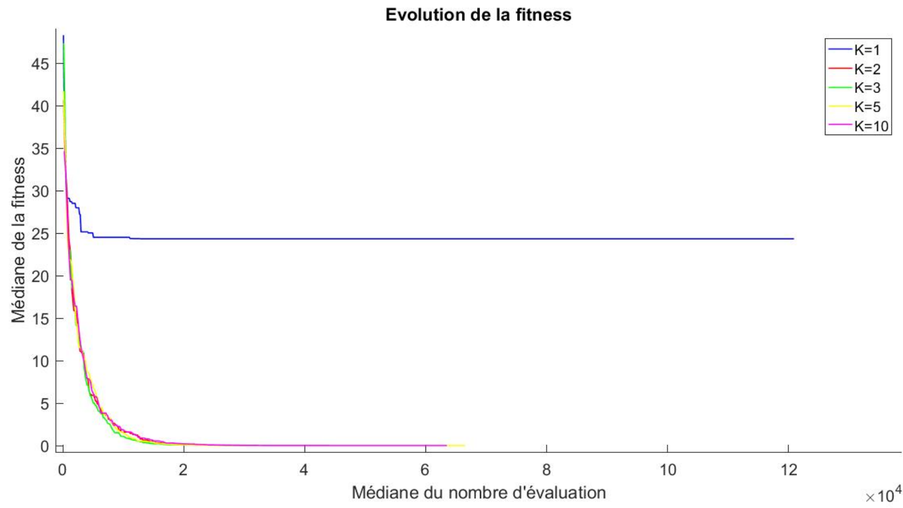

Figure 7.28 – Évolution de la valeur de fitness associée à la fonction Schwefel 2.21 . Test des algorithmes avec différents nombres de liens maximum autorisé dans la topologie.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$K = 1$	0	24.3190	120879
$K = 2$	0.92	$\mathcal{O}(10^{-6})$	62698
$K = 3$	1	$\mathcal{O}(10^{-6})$	51578
$K = 5$	0.96	$\mathcal{O}(10^{-6})$	66533
$K = 10$	0.88	$\mathcal{O}(10^{-6})$	63521

Table 7.28 – Résultats quantitatifs associés au graphique de la Figure 7.28

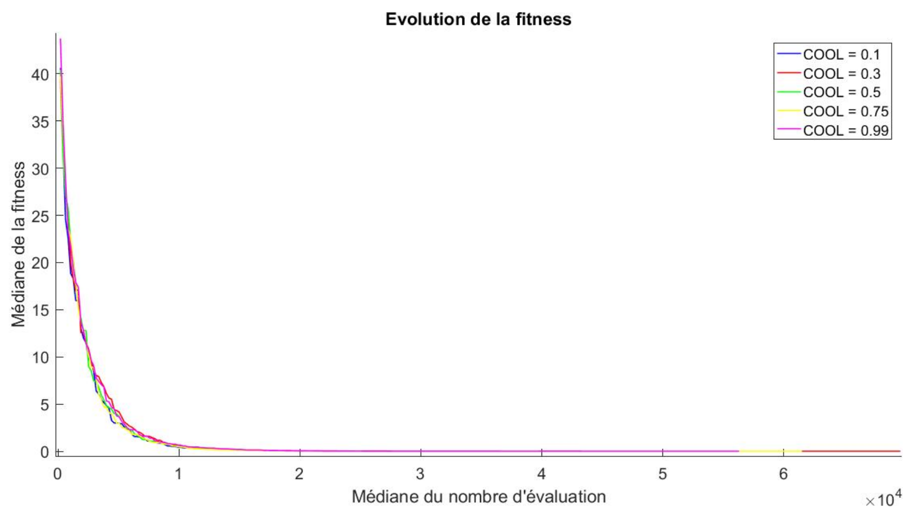

Figure 7.29 – Évolution de la valeur de fitness associée à la fonction Schwefel 2.21. Test des différentes valeurs du paramètre $COOL$.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$COOL = 0.1$	0.96	$\mathcal{O}(10^{-6})$	58409
$COOL = 0.3$	0.88	$\mathcal{O}(10^{-6})$	69633
$COOL = 0.5$	0.96	$\mathcal{O}(10^{-6})$	57305
$COOL = 0.75$	1	$\mathcal{O}(10^{-6})$	61495
$COOL = 0.99$	1	$\mathcal{O}(10^{-6})$	56286

TABLE 7.29 – Résultats quantitatifs associés au graphique de la Figure 7.29

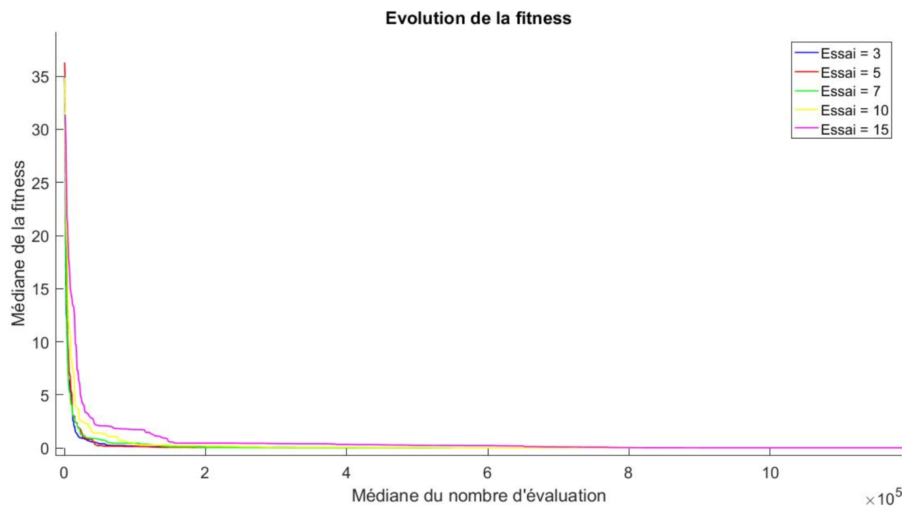

FIGURE 7.30 – Évolution de la valeur de fitness associée à la fonction Schwefel 2.21. Test de différents nombres d'essais dans la recherche tabou.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$Essai = 3$	0.1	0.0068	329365
$Essai = 5$	0	0.0232	486658
$Essai = 7$	0.1	0.0017	623565
$Essai = 10$	0	0.0033	847362
$Essai = 15$	0	0.0051	$1.1978*10^6$

TABLE 7.30 – Résultats quantitatifs associés au graphique de la Figure 7.30

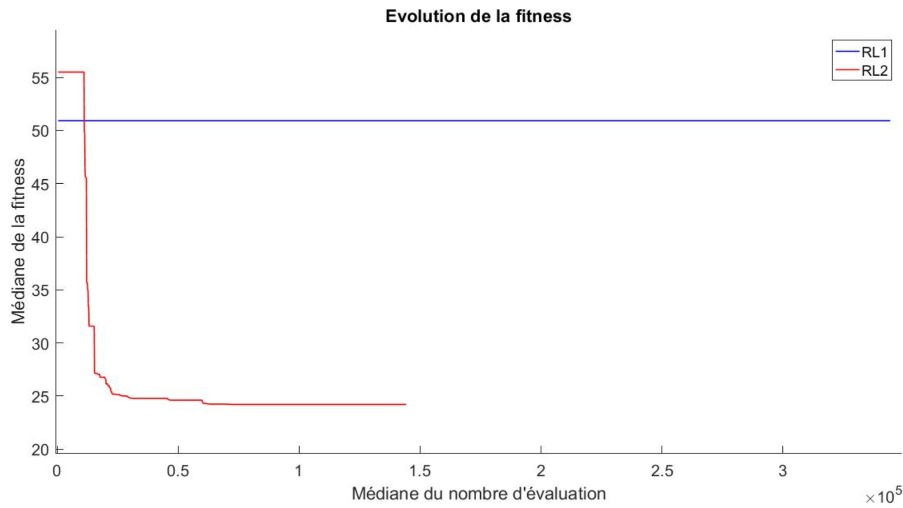

Figure 7.31 – Évolution de la valeur de fitness associée à la fonction Schwefel 2.21. Test des algorithmes avec recherche linéaire exacte. RL1 est la version ou nous n'utilisons que la recherche linéaire exacte. RL2 est la version ou nous recalculons classiquement les vitesses en cas de stagnation.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
RL 1	0	50.9254	$3.4430*10^5$
RL 2	0.3	24.1907	$1.4427*10^5$

Table 7.31 – Résultats quantitatifs associés au graphique de la Figure 7.31

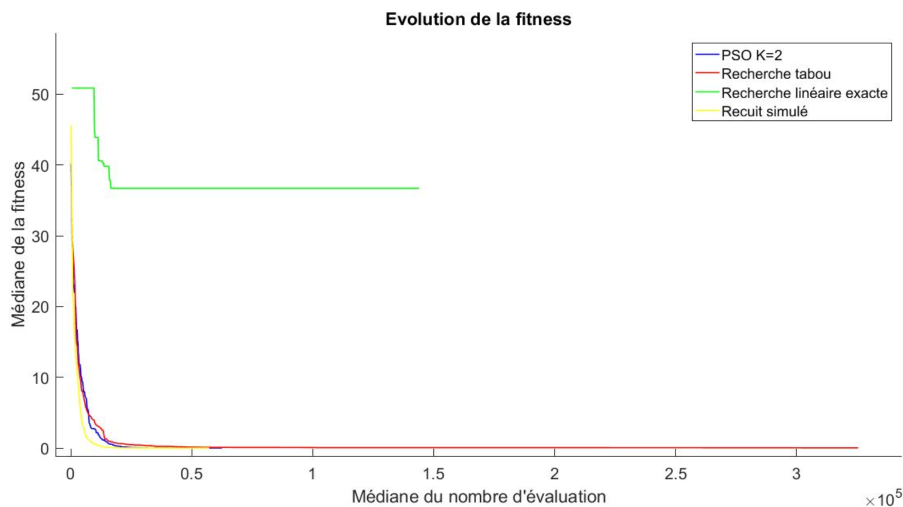

Figure 7.32 – Évolution de la valeur de fitness associée à la fonction Schwefel 2.21. Test des différents algorithmes du chapitre 7.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$PSO\ K = 2$	1	$\mathcal{O}(10^{-6})$	$6.2506*10^4$
Recherche tabou	0	0.0232	$3.2544*10^5$
Recherche linéaire exacte	0.3	36.6984	144055
Recuit simulé	0.94	$\mathcal{O}(10^{-6})$	$5.7224*10^4$

TABLE 7.32 – Résultats quantitatifs associés au graphique de la Figure 7.32

Cas-test Griewank

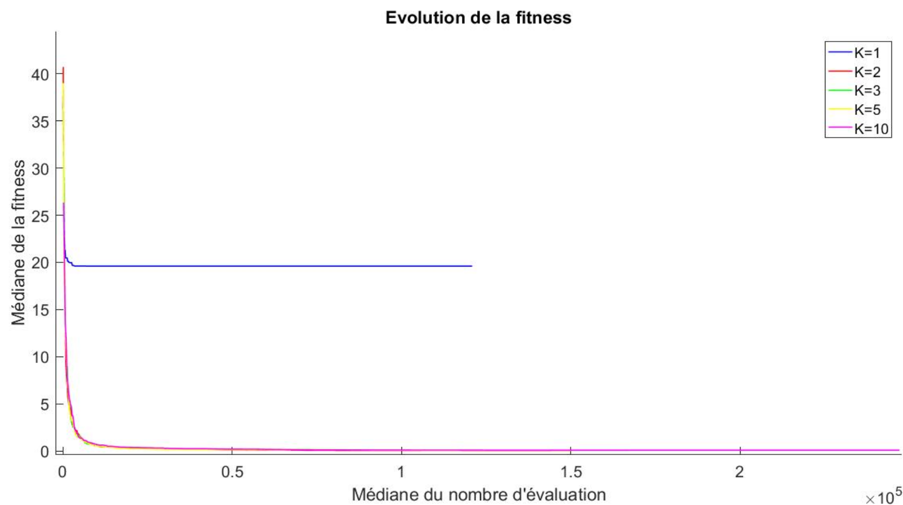

FIGURE 7.33 – Évolution de la valeur de fitness associée à la fonction Griewank. Test des algorithmes avec différents nombres de liens maximum autorisé dans la topologie.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$K = 1$	0	19.5997	120879
$K = 2$	0.04	0.0370	150104
$K = 3$	0	0.0616	167208
$K = 5$	0	0.0508	149889
$K = 10$	0	0.0690	247025

TABLE 7.33 – Résultats quantitatifs associés au graphique de la Figure 7.33

FIGURE 7.34 – Évolution de la valeur de fitness associée à la fonction Griewank. Test des différentes valeurs du paramètre $COOL$.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$COOL = 0.1$	0	0.0566	208298
$COOL = 0.3$	0	0.0518	208411
$COOL = 0.5$	0	0.0566	208375
$COOL = 0.75$	0	0.0468	208613
$COOL = 0.99$	0	0.0468	208522

TABLE 7.34 – Résultats quantitatifs associés au graphique de la Figure 7.34

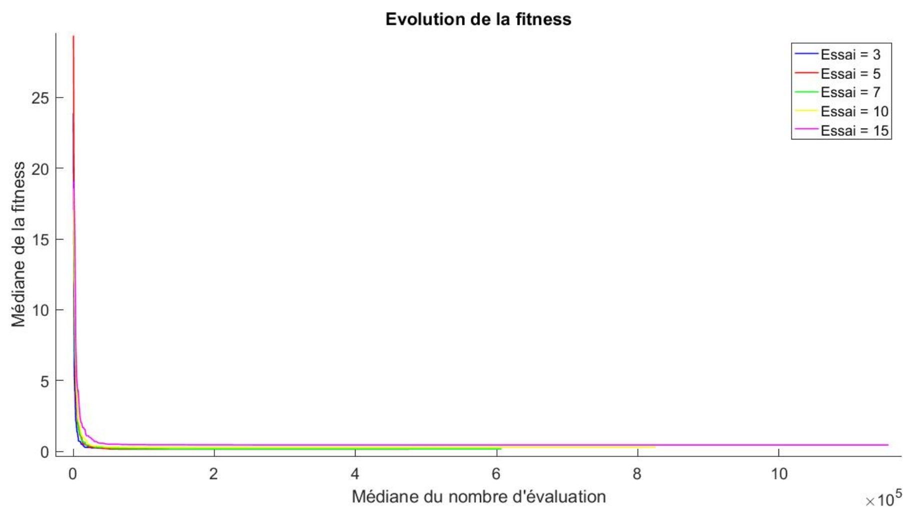

FIGURE 7.35 – Évolution de la valeur de fitness associée à la fonction Griewank. Test de différents nombres d'essais dans la recherche tabou.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$Essai = 3$	0	0.1578	336314
$Essai = 5$	0	0.1450	$4.7702*10^5$
$Essai = 7$	0.	0.1614	606722
$Essai = 10$	0	0.2849	825283
$Essai = 15$	0	0.4430	$1.1554*10^6$

TABLE 7.35 – Résultats quantitatifs associés au graphique de la Figure 7.35

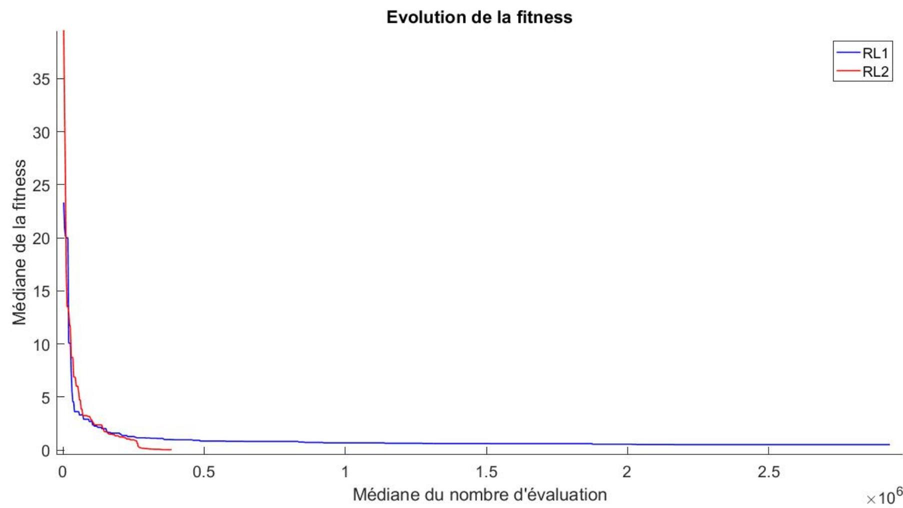

FIGURE 7.36 – Évolution de la valeur de fitness associée à la fonction Griewank. Test des algorithmes avec recherche linéaire exacte. RL1 est la version ou nous n'utilisons que la recherche linéaire exacte. RL2 est la version ou nous recalculons classiquement les vitesses en cas de stagnation.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
RL 1	0	0.5060	$2.9296*10^6$
RL 2	0	0.0357	$3.8523*10^5$

TABLE 7.36 – Résultats quantitatifs associés au graphique de la Figure 7.36

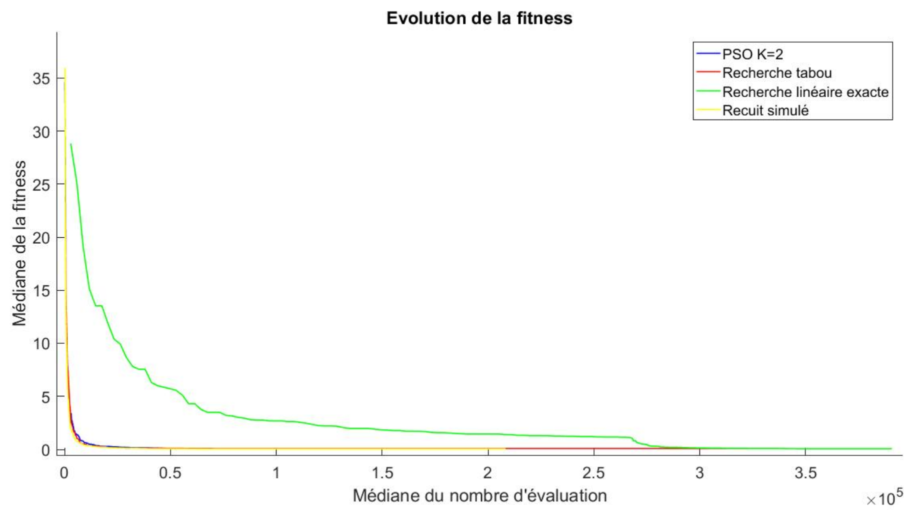

Figure 7.37 – Évolution de la valeur de fitness associée à la fonction Griewank. Test des différents algorithmes du chapitre 7.

algorithme	Taux de réussite	$fitness_{min}$	$eval_{best}$
$PSO\ K = 2$	0	0.0485	150543
Recherche tabou	0	0.0847	336283
Recherche linéaire exacte	0	0.0592	390725
Recuit simulé	0	0.0518	$2.0837*10^5$

Table 7.37 – Résultats quantitatifs associés au graphique de la Figure 7.37