République Algérienne Démocratique et Populaire
Ministère de L'Enseignement Supérieur et de la Recherche Scientifique
Université 20 Août 1955 Skikda

Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études en vue de l'obtention du diplôme De

Master

Option: Systèmes d'information avancée et applications (SIAA)

Thème

Optimisation par l'algorithme Jaya et ses variantes : Application au regroupement d'élèves de l'enseignement moyen

Réalisé par:

BOULFOUL Haroune

Encadré par :

Dr.RAMDANE Chafika

Session : Juin 2025

REMERCIEMENTS

Mes premiers et plus sincères remerciements vont à Allah, le Tout-Puissant et le Très Miséricordieux, qui m'a accordé la force, le courage et la patience nécessaires pour mener à bien ce travail.

Je tiens à exprimer ma profonde gratitude au Dr. Ramdane pour avoir accepté de diriger ce mémoire. Ses conseils précieux, son regard critique, sa patience et son dévouement ont été déterminants dans la réalisation de ce modeste travail.

Mes remerciements s'adressent également à tous mes enseignants qui ont contribué à ma formation tout au long de mes années d'étude.

Enfin, je souhaite remercier chaleureusement toutes les personnes qui, de près ou de loin, ont contribué à la réalisation de ce travail.

DEDICACE

J’adresse mon travail à :

- **Mon père et ma mère** : Les racines dont j’ai puisé la sève dans leur silence, et le ciel qui a porté mes rêves.
- ***Mon épouse « Hayet » :*** Mon soutien inébranlable, qui a fait de chaque obstacle une étape vers l’accomplissement.
- **Mon fils** (d’un an) « **Idris »**: Fleur de mon avenir qui me rappelle que chaque mot écrit est un présent pour tes petits pieds.
- **Mes frères et sœurs** : Le lien d’affection qui ne m’a jamais fait défaut dans l’épreuve comme dans l’aisance.

RESUME

Ce travail explore l'application de l'algorithme d'optimisation **Jaya** et de ses variantes **EJaya** et **CLJaya** dans deux contextes distincts :

1. Une **évaluation standard** sur les fonctions de test **CEC 2017** et **CEC 2019** pour l'optimisation continue.
2. La **résolution d'un problème pratique** : la répartition équitable d'élèves en groupes selon :
    - l'équilibre des effectifs,
    - la parité de genre,
    - l'homogénéité des moyennes (générales et disciplinaires).

Une analyse comparative approfondie a été menée afin d'évaluer les performances des trois variantes dans ces deux scénarios. Les résultats obtenus démontrent la faisabilité et l'efficacité de ces approches pour résoudre des problèmes discrets complexes, avec des implications concrètes pour les systèmes éducatifs.

Mots-clés : Métaheuristiques, algorithme Jaya, EJaya, CLJaya, benchmark CEC, formation de groupes sous contraintes, regroupement d'élèves, optimisation combinatoire, problème discret.

ABSTRUCT

This work explores the application of the **Jaya** optimization algorithm and its variants, **EJaya** and **CLJaya**, in two distinct contexts:

1. A **standard evaluation** using the **CEC 2017** and **CEC 2019** benchmark functions for continuous optimization.

2. The **solution of a practical problem**: the fair distribution of students into groups based on:
    - balanced group sizes,
    - gender parity,
    - homogeneity of grade averages (general and subject-specific).

A thorough comparative analysis was conducted to assess the performance of the three variants in both scenarios. The results demonstrate the feasibility and effectiveness of these approaches in solving complex discrete problems, with practical implications for educational systems.

Keywords: Metaheuristics, Jaya algorithm, EJaya, CLJaya, CEC benchmark, constrained group formation, student grouping, combinatorial optimization, discrete problem.

Table des matières

Liste des Figures

Liste des Tableaux

Introduction Générale

Introduction Générale

L'optimisation est un domaine fondamental de l'informatique et des mathématiques appliquées, qui vise à identifier la meilleure solution possible parmi un ensemble de solutions admissibles, en minimisant ou en maximisant une ou plusieurs fonctions objectives. Les problèmes d'optimisation sont omniprésents dans divers secteurs, allant de l'ingénierie et de l'économie à l'intelligence artificielle, la logistique et la gestion de systèmes complexes.

On distingue généralement deux grandes catégories de problèmes d'optimisation : les problèmes à variables continues, où les variables de décision peuvent prendre n'importe quelle valeur dans un intervalle donné, et les problèmes discrets ou combinatoires, où les solutions sont constituées d'éléments distincts ou entiers. Ces derniers posent souvent un défi plus grand en raison de la combinatoire explosive de leur espace de recherche.

Les approches classiques (ou exactes) telles que la programmation linéaire, la programmation dynamique ou les méthodes branch and bound deviennent rapidement inefficaces pour des problèmes de grande dimension ou comportant de nombreuses contraintes. Pour pallier cette difficulté, des méthodes heuristiques et métaheuristiques ont été développées. Ces dernières sont des algorithmes approchés, souvent inspirés de phénomènes naturels, biologiques ou sociaux, et conçus pour fournir des solutions satisfaisantes en un temps raisonnable, sans nécessairement garantir l'optimalité globale.

Les métaheuristiques incluent un large éventail de techniques telles que les algorithmes génétiques (GA), l'optimisation par essaim de particules (PSO), les algorithmes de colonies de fourmis (ACO), la recherche taboue, le recuit simulé, ou encore les méthodes plus récentes telles que DE, CMA-ES ou GWO. Ces approches ont prouvé leur efficacité dans de nombreux contextes, notamment pour des problèmes complexes, multi-objectifs ou dynamiques.

Dans ce contexte évolutif, l'algorithme Jaya, proposé par R. Venkata Rao en 2016[1], s'est distingué comme une méthode simple, robuste et sans paramètres de contrôle spécifiques. Son nom, tiré du mot sanskrit signifiant "victoire", reflète son objectif : se rapprocher de la meilleure solution connue tout en s'éloignant de la pire. À chaque itération, les solutions candidates sont mises à jour en fonction de cette double dynamique, sans recours à des paramètres complexes comme les coefficients d'inertie, de croisement ou de mutation, ce qui facilite sa mise en œuvre et réduit le besoin de réglages empiriques.

L'algorithme Jaya a été largement utilisé pour résoudre des problèmes d'optimisation continue, notamment à travers les fonctions de benchmark standardisées telles que celles proposées dans les compétitions CEC (Congress on Evolutionary Computation) de 2017 et 2019. Ces fonctions couvrent une diversité de défis — fonctions unimodales, multimodales, avec ou sans contraintes — et servent de référence pour évaluer les performances comparatives des algorithmes d'optimisation.

Afin d'améliorer la capacité exploratoire ou exploitante de Jaya, plusieurs variantes ont été introduites dans la littérature, parmi lesquelles :

EJaya (Enhanced Jaya) [2] : qui introduit des stratégies d'enrichissement de la population ou d'exploration adaptative ;

CLJaya (Cooperative Learning Jaya) [3] : qui repose sur une coopération entre sous-populations pour renforcer la diversité et éviter le piégeage prématuré dans des optima locaux.

Cependant, la majorité de ces travaux se sont concentrés sur des environnements à variables continues. Or, de nombreux problèmes réels — notamment en planification, allocation de ressources ou organisation scolaire — sont de nature discrète et nécessitent une adaptation spécifique des algorithmes.

Dans ce travail, nous nous intéressons à l'application de l'algorithme Jaya et de ses variantes à un problème discret réel, à savoir : la formation équitable de groupes d'élèves. Ce problème, fréquent dans les établissements éducatifs, consiste à répartir un ensemble d'étudiants en groupes équilibrés selon plusieurs critères : taille homogène des groupes, parité de genre, et homogénéité des moyennes académiques (générales et par matière). Ces contraintes introduisent une combinatoire complexe qui rend le problème difficile à résoudre de manière exacte, surtout lorsque le nombre d'étudiants est élevé.

L'adaptation de l'algorithme Jaya et de ses variantes à ce type de problème nécessite une modification de certains mécanismes internes, notamment pour le traitement des variables discrètes, la gestion des contraintes spécifiques, ainsi que l'ajustement des opérateurs de mise à jour des solutions afin de les rendre compatibles avec la nature combinatoire du problème étudié.

Ce travail comporte ainsi deux volets principaux :

Une évaluation expérimentale des algorithmes Jaya, EJaya et CLJaya sur des fonctions de test continues provenant des benchmarks CEC 2017 et 2019, afin de valider leur comportement dans un cadre standardisé.

Une application pratique de ces trois algorithmes, après adaptation, à un problème discret concret, celui de la répartition équilibrée des étudiants en groupes.

À travers cette double approche, nous cherchons à étudier la capacité d'adaptation des métaheuristiques continues à des contextes discrets, et à proposer une méthodologie algorithmique réutilisable pour des problèmes combinatoires similaires.

Chapitre 01 : Optimisation et métaheuristiques

1. Introduction

L'optimisation occupe une place centrale dans de nombreux domaines scientifiques et techniques, allant de l'ingénierie à la finance, en passant par l'intelligence artificielle et la recherche opérationnelle [4]. Elle vise à identifier la meilleure solution possible — ou une solution satisfaisante — à un problème donné, en respectant un ensemble de contraintes spécifiques. Qu'il s'agisse de minimiser un coût, de maximiser un rendement ou d'équilibrer des ressources, l'optimisation permet d'orienter les décisions de manière efficace et rationnelle.

Cependant, face à la complexité croissante des systèmes réels et à l'explosion combinatoire de certains problèmes, les méthodes d'optimisation classiques, souvent exactes, atteignent rapidement leurs limites en termes de temps de calcul et de faisabilité [5]. C'est dans ce contexte que les métaheuristiques ont émergé comme une alternative puissante et flexible [6] [7]. Inspirées de phénomènes naturels, biologiques ou sociaux, ces méthodes de résolution approximative permettent d'explorer intelligemment de vastes espaces de recherche tout en évitant de tomber dans des minima locaux [8].

Ce chapitre a pour objectif de poser les bases conceptuelles de l'optimisation et de présenter les principales familles de métaheuristiques. Il abordera d'abord les notions fondamentales de l'optimisation, ses différentes classifications et applications typiques, avant de s'intéresser aux principes généraux des métaheuristiques, à leur intérêt et à leur évolution au fil des années. Cette introduction théorique permettra de mieux comprendre les choix méthodologiques faits dans la suite de ce travail.

2. Définition de l'optimisation

L'optimisation peut être définie comme le processus consistant à rechercher [9], parmi un ensemble de solutions admissibles, celle qui optimise une fonction objectif donnée [10]. En mathématiques appliquées et en informatique, cela revient à maximiser ou à minimiser une fonction en fonction de variables de décision soumises à des contraintes.

Formulation mathématique [11] **:**

$$\text{Trouver } x^* \in X \text{:tel que } f(x^*) = \max_{x \in X} f\,(x) \quad \text{ou} \quad f(x^*) = \min_{x \in X} f\,(x)$$

Où$x$est un vecteur de variables de décision, $f\,(x)$est la fonction objective à optimiser, et$X$ est l'ensemble des solutions admissibles, défini par les contraintes du problème. L'objectif est de trouver la solution optimalex^*qui rend la fonction objective la plus grande (maximisation) ou la plus petite (minimisation).

3. Composantes d'un problème d'optimisation

Un problème d'optimisation est généralement défini par trois éléments fondamentaux [12]:

Fonction Objective : La métrique à améliorer (minimiser les coûts, maximiser les profits, réduire le temps d'exécution).

Variables de Décision : Les paramètres ajustables qui influencent la fonction objective.

Contraintes : Les conditions ou restrictions qui limitent les valeurs que les variables de décision peuvent prendre (ressources limitées, délais, spécifications techniques).

4. Classification des Principaux Types d'Optimisation

L'optimisation est un domaine fondamental des mathématiques appliquées et de l'informatique, qui consiste à rechercher la ou les meilleures solutions possibles à un problème donné, selon un ou plusieurs critères définis. La richesse de ce domaine provient de la diversité des problèmes à traiter, qui diffèrent selon la nature des variables, la structure des fonctions, les contraintes imposées, ou encore le degré d'incertitude présent dans les données.

Pour mieux comprendre les méthodes d'optimisation — et notamment la place occupée par les métaheuristiques — il est utile de classifier les types d'optimisation selon plusieurs axes fondamentaux [13] [14].

4.1. Nature des variables

- Optimisation continue : les variables peuvent prendre n'importe quelle valeur réelle. *Ex. : conception de structures, réglage de paramètres d'un modèle.*
- Optimisation discrète ou combinatoire : les variables prennent des valeurs entières ou sont issues d'un ensemble fini (permutations, affectations, graphes). *Ex. : problèmes de planification, de tournées de véhicules.*
- Optimisation mixte : combinaison de variables continues et discrètes. *Ex. : planification industrielle, transport multimodal.*

4.2. Structure du problème

- Optimisation linéaire (LP) : fonction objectif et contraintes sont linéaires.
- Optimisation non linéaire (NLP) : au moins une fonction (objectif ou contrainte) est nonlinéaire.
- Optimisation quadratique (QP): la fonction objective est quadratique, les contraintes sont linéaires.

4.3. Présence de contraintes

- Optimisation sans contraintes : pas de restriction sur les variables.
- Optimisation sous contraintes : les solutions doivent respecter des conditions d'égalité ou d'inégalité.

Catégorie	Type d'optimisation	Méthodes typiques	Domaines d'application
Par nature des variables	Continue	Gradient, Newton, algorithmes évolutionnaires	Apprentissage automatique, ingénierie
	Discrète	Branch-and-bound, recuit simulé, ACO	Logistique, planification, réseaux
	Mixte	MILP, méthodes hybrides	Industrie, transport
Par structure du problème	Linéaire	Simplexe, points intérieurs	Allocation de ressources, production
	Non linéaire	SQP, Lagrangien, gradient projeté	Robotique, contrôle, finance
	Quadratique	QP, méthodes convexes	Ingénierie, économie, finance
Par contraintes	Sans contraintes	Descente de gradient, Newton	Optimisation pure
	Sous contraintes	KKT, Lagrangien, pénalisation	Systèmes physiques, ingénierie

Tableau 1: Tableau récapitulatif des principales catégories d'optimisation

L'optimisation est omniprésente et cruciale dans des domaines variés, de la logistique à la finance, en passant par la conception d'ingénierie et la planification des ressources. Elle permet de résoudre des problèmes qui seraient insolubles ou trop coûteux avec des approches manuelles ou intuitives.

5. Méthodes de résolution des problèmes d'optimisation

La résolution des problèmes d'optimisation repose principalement sur deux grandes catégories de méthodes : les méthodes exactes et les méthodes approchées. Ces deux approches se distinguent par leur manière d'explorer l'espace de solutions et par les garanties qu'elles offrent en termes de qualité des résultats.

5.1 Méthodes exactes

Les méthodes exactes visent à trouver la solution optimale garantie d'un problème d'optimisation, en explorant de manière exhaustive ou systématique l'ensemble des solutions admissibles, dans le respect strict des contraintes du problème [15].

Principales caractéristiques :

- Résultats optimaux (ou preuve qu'il n'en existe pas).

- S'appuient sur une modélisation mathématique rigoureuse.
- Requises pour des problèmes bien définis et de taille modérée.

Exemples de méthodes exactes :

- Programmation linéaire (PL) et programmation linéaire en nombres entiers (PLNE).
- Programmation non linéaire (PNL).
- Programmation dynamique.
- Branch-and-bound, branch-and-cut, simplexe.
- Programmation par contraintes.

Limites :

- Complexité exponentielle pour les grands problèmes (notamment NP-difficiles).
- Peu adaptées aux environnements incertains ou aux données bruitées.
- Requiert souvent des fonctions continues, dérivables ou convexes.

5.2 Méthodes approchées

Les méthodes approchées (ou heuristiques) ne garantissent pas l'obtention de la solution optimale, mais permettent de trouver des solutions de qualité en un temps raisonnable, notamment pour les problèmes de grande dimension ou complexes [16].

Caractéristiques générales :

- Ne fournissent pas de preuve d'optimalité.
- Moins sensibles à la structure mathématique du problème.
- Utilisent des règles empiriques ou des stratégies d'exploration heuristiques.
- Plus rapides et flexibles que les méthodes exactes.

Types de méthodes approchées :

- Heuristiques spécifiques : conçues pour un type particulier de problème (ex. : plus proche voisin pour le TSP).
- Méthodes stochastiques : introduisent des éléments aléatoires dans la recherche.
- Métaheuristiques : généralisations puissantes adaptées à une large classe de problèmes (voir section suivante).

Limites :

- Résultats non déterministes.
- Dépendance au choix des paramètres et à l'initialisation.
- Pas de garantie de convergence vers l'optimum global.

6. Les Métaheuristiques

Les métaheuristiques constituent une sous-catégorie avancée des méthodes approchées. Il s'agit de stratégies d'optimisation haut niveau guidant la recherche d'une solution dans un espace de solutions potentiellement très vaste, avec un bon compromis entre exploration (diversification) et exploitation (intensification).

6.1 Définition et principes

Une métaheuristique est une méthode générique, souvent inspirée de phénomènes naturels [17] [18], biologiques, physiques ou sociaux, conçue pour résoudre des problèmes complexes, non linéaires, non convexes, voire discrets, où les méthodes exactes deviennent inapplicables.

Les métaheuristiques :

- Ne nécessitent pas de dérivabilité ou de convexité.
- S'adaptent à divers types de variables (réelles, entières, mixtes).
- Peuvent être facilement modifiées ou combinées avec d'autres méthodes (hybrides).

6.2 Exemples de métaheuristiques

- Algorithme génétique (GA) : basé sur l'évolution biologique.
- Recuit simulé (SA) : inspiré du refroidissement des métaux.
- Optimisation par essaim particulaire (PSO) : imite le comportement d'un groupe d'animaux.
- Colonies de fourmis (ACO) : inspiré du déplacement des fourmis vers une source de nourriture.
- Jaya, Grey Wolf Optimizer (GWO), Whale Optimization Algorithm (WOA) : algorithmes récents performants et sans paramètre complexe à ajuster.

6.3 Avantages et limites

Avantages :

- Très bonne performance sur des problèmes difficiles (grande dimension, non convexes).
- Ne nécessitent pas une connaissance approfondie du problème.
- Possibilité de parallélisation et d'adaptation.

Limites :

- Ne garantissent pas la solution optimale.
- Nécessitent souvent plusieurs essais pour valider la robustesse des résultats.
- Certains algorithmes sont sensibles au réglage des paramètres.

7. Applications typiques de l'optimisation

L'optimisation constitue un outil fondamental dans de nombreux domaines scientifiques et industriels. Elle intervient chaque fois qu'il est nécessaire de prendre une décision rationnelle dans un système complexe, en maximisant ou minimisant une fonction objective sous contraintes. Les applications typiques de l'optimisation sont vastes [18] et couvrent une multitude de secteurs. Parmi les plus significatives, on peut citer :

- **Ingénierie** : l'optimisation est utilisée dans la conception de structures mécaniques, l'optimisation de formes aérodynamiques, la planification énergétique, ainsi que dans la commande optimale de systèmes dynamiques. Ces applications visent généralement à minimiser les coûts de production, la consommation énergétique, ou encore à maximiser la performance technique.
- **Logistique et transport** : les problèmes d'optimisation interviennent dans la gestion des itinéraires (ex. : problème du voyageur de commerce, routage de véhicules), l'organisation des chaînes logistiques, la gestion des stocks, et la planification des livraisons, dans le but de réduire les délais, les distances parcourues et les coûts associés.
- **Économie et finance** : l'optimisation est utilisée dans la gestion de portefeuilles, la maximisation des rendements financiers sous contraintes de risque, la modélisation des marchés, ou encore l'analyse coût-bénéfice pour l'allocation optimale des ressources.
- **Production et industrie** : dans ce domaine, l'optimisation permet de résoudre des problèmes liés à la planification de la production, l'ordonnancement des tâches, la gestion des ressources, et la maintenance préventive. L'objectif est d'augmenter la productivité, de réduire les temps d'arrêt et d'optimiser les coûts de fabrication.
- **Télécommunications et réseaux** : l'optimisation est essentielle pour la planification de réseaux, l'allocation des canaux de fréquence, la gestion de la bande passante, la minimisation de la latence, et l'optimisation de la couverture réseau.
- **Intelligence artificielle et apprentissage automatique** : les algorithmes d'optimisation sont au cœur de nombreuses méthodes d'apprentissage supervisé et non supervisé, notamment pour l'ajustement des paramètres, la réduction de l'erreur de prédiction et l'amélioration de la généralisation des modèles.
- **Santé et biologie** : les applications incluent la planification des traitements médicaux (par exemple en radiothérapie), la modélisation de systèmes biologiques, et la recherche de configurations optimales en bioinformatique, telles que l'alignement de séquences ou la modélisation de structures protéiques.
- **Éducation et emploi du temps** : l'optimisation intervient dans la construction d'emplois du temps scolaires, l'affectation d'enseignants, la répartition des groupes d'élèves, et la gestion des ressources pédagogiques, tout en respectant des contraintes pédagogiques, matérielles et humaines.

- **Planification urbaine et énergie** : les méthodes d'optimisation sont utilisées pour la gestion des réseaux intelligents (smart grids), l'optimisation de la consommation d'énergie, et la conception durable d'infrastructures urbaines.

8. Conclusion

En résumé, l'optimisation est une discipline transversale qui joue un rôle stratégique dans la résolution de problèmes complexes, en permettant une prise de décision efficace, rationnelle et scientifiquement fondée.

Dans le chapitre suivant, nous nous intéresserons à l'algorithme d'optimisation Jaya ainsi qu'à ses variantes EJaya et CLJaya.

Chapitre 02 : Algorithme JAYA et ses variantes

1. Introduction

Comme développé dans le chapitre précédent, l'optimisation de problèmes complexes, notamment non linéaires, combinatoires ou à contraintes multiples, constitue un enjeu majeur dans de nombreux domaines scientifiques. Nous y avons souligné les limites des méthodes exactes, notamment leur complexité algorithmique et leur coût computationnel, qui justifient le recours croissant aux métaheuristiques.

Parmi ces approches, deux grandes familles dominent traditionnellement :

- Les algorithmes évolutionnaires (comme les algorithmes génétiques),
- Les algorithmes à base d'essaims (tels que PSO ou ACO).

Ces deux classes partagent un fonctionnement probabiliste, basé sur des mécanismes de diversification (exploration aléatoire de l'espace de recherche) et d'intensification (convergence vers les zones prometteuses), avec une structure populationnelle qui facilite l'exploration parallèle de plusieurs solutions.

C'est dans cette dynamique qu'intervient l'algorithme JAYA, proposé par Rao en 2016. Bien qu'il adopte une structure populationnelle et un principe stochastique de mise à jour des solutions, JAYA ne relève ni des algorithmes évolutionnaires (il n'utilise ni croisement, ni mutation, ni sélection), ni des approches à base d'essaims (il ne repose sur aucun comportement collectif ou social). Il constitue ainsi une métaheuristique non inspirée biologiquement, classée dans une catégorie simple, sans paramètre à ajuster, et conçue uniquement sur la base de considérations algorithmico-mathématiques.

Son principe repose sur une règle directe : rapprocher chaque solution de la meilleure solution actuelle tout en l'éloignant de la pire [1]. Cette simplicité fait de JAYA un algorithme robuste, flexible, et facile à implémenter, avec un potentiel élevé d'hybridation.

Depuis sa création, plusieurs variantes ont été développées, visant à améliorer sa capacité d'exploration ou à adapter son comportement à des contextes spécifiques. Ce chapitre présente ainsi :

- Le fonctionnement de l'algorithme JAYA,
- Deux de ses extensions majeures : EJAYA et CLJAYA,
- Une comparaison des variantes, illustrant leurs performances et leurs domaines d'application.

2. Origine et motivation de JAYA

Dans la majorité des algorithmes évolutionnaires et à base d'essaims (tels que GA, PSO, ABC, ACO, HS, etc.), le bon fonctionnement dépend de paramètres de contrôle spécifiques à chaque algorithme [8] [6] :

- Par exemple, l'algorithme génétique (GA) utilise des paramètres comme la probabilité de mutation, probabilité de croisement, et des opérateurs de sélection [19].

- Le PSO repose sur l'inertie, les coefficients cognitifs et sociaux [20].
- D'autres comme ABC, HS, ou DE exigent également de nombreux paramètres algorithmiques (nombre d'abeilles, taux d'improvisation, etc.) [8].

Le réglage de ces paramètres est souvent complexe et délicat, car une mauvaise configuration peut soit entraîner une convergence prématurée vers des optima locaux, soit accroître inutilement le coût de calcul sans amélioration significative de la solution [18].

Pour remédier à cette difficulté, Rao et al. (2011) avaient introduit l'algorithme Teaching–Learning-Based Optimization (TLBO), reconnu pour ne nécessiter aucun paramètre spécifique à l'algorithme, seulement des paramètres communs comme la taille de la population et le nombre d'itérations [21].

Fort du succès de TLBO, Rao a ensuite proposé l'algorithme JAYA, poursuivant la même philosophie : concevoir un algorithme sans phase distincte ni paramètre spécifique, encore plus simple à mettre en œuvre que TLBO.

Contrairement à TLBO, qui repose sur deux phases (enseignant et apprenant), JAYA ne comprend qu'une seule étape de mise à jour, fondée uniquement sur le rapprochement vers la meilleure solution et l'éloignement de la pire solution.

« The proposed algorithm has only one phase and it is comparatively simpler to apply » (Rao, 2016) [1].

Cette simplicité algorithmique et l'absence totale de paramètres spécifiques font de JAYA une métaheuristique robuste, générique, et facilement adaptable à une large variété de problèmes d'optimisation.

3. Définition de JAYA

L'algorithme JAYA, proposé par Rao (2016), est une métaheuristique de type populationnel qui s'inspire d'un concept simple : rapprocher chaque solution de la meilleure solution connue et l'éloigner de la pire. Contrairement à de nombreuses autres méthodes inspirées de la nature (algorithmes évolutionnaires ou par essaims), JAYA ne nécessite aucun paramètre spécifique à ajuster comme la probabilité de mutation, les coefficients d'inertie ou encore les opérateurs de sélection [1].

Le nom "JAYA" vient du mot sanskrit "जय" (jaya), signifiant victoire, traduisant ainsi l'objectif d'améliorer la qualité de la solution à chaque itération.

3.1 Catégorie

JAYA appartient à la classe des métaheuristiques sans paramètres spécifiques (parameter-less optimization algorithms). Elle est conceptuellement proche des algorithmes évolutionnaires mais ne repose pas sur des mécanismes biologiques explicites comme sélection, croisement ou mutation. Elle n'appartient pas à la famille des algorithmes à base d'essaims comme PSO ou ACO.

3.2 Objectif

Minimiser (ou maximiser) une fonction objective $f(x)$ sur un espace de solutions donné, sans connaissance a priori sur sa dérivabilité ou convexité, et sans recourir à des paramètres algorithmiques complexes.

4. Fonctionnement de l'algorithme JAYA

L'algorithme JAYA, proposé par Rao (2016), repose sur un principe d'optimisation simple et robuste, sans nécessiter de paramètres spécifiques à l'algorithme, ce qui le distingue des autres métaheuristiques de type évolutionnaire ou à base d'essaims [Rao, 2016].

4.1 Principe de base

Le principe fondamental de JAYA est le suivant :

Toute solution doit évoluer en se rapprochant de la meilleure solution et en s'éloignant de la pire solution connue à ce stade [1].

Ce mécanisme permet d'équilibrer diversification (exploration de nouvelles zones) et intensification (exploitation des zones prometteuses), deux composantes essentielles à la performance d'un algorithme populationnel [9] [8].

4.2 Initialisation de la population

L'algorithme débute par la génération aléatoire d'une population initiale composée de n solutions candidates. Chaque solution représente un vecteur de variables de décision, défini dans un domaine borné. Cette population constitue le point de départ de la recherche [1].

4.3 Processus itératif

À chaque itération, l'algorithme suit les étapes suivantes :

- Évaluation : chaque solution candidate est évaluée via la fonction objectif. En cas de contraintes, des pénalités peuvent être appliquées afin de favoriser les solutions réalisables [11].
- Identification des extrêmes : les solutions meilleure (x_best) et pire (x_worst) sont déterminées dans la population actuelle.
- Mise à jour des solutions : pour chaque solution x_old, une nouvelle version x_new est calculée selon l'équation suivante :

$$x\_new = x\_old + r_1 \cdot (x\_best - |x\_old|) - r_2 \cdot (x\_worst - |x\_old|)$$

Où :

- $r_1 \; et \; r_2$ sont deux nombres aléatoires $\in [0,1]$,

- $|x\_old|$ représente la valeur absolue de la variable.

La solution mise à jour est acceptée uniquement si elle améliore la solution précédente.

- Critère d'arrêt : l'algorithme s'arrête lorsque le nombre maximal d'itérations est atteint ou lorsqu'un critère de convergence est rempli.

4.4 Résultat final

À la fin de l'algorithme, la meilleure solution x_best est retournée avec sa valeur de la fonction objectif.

La solution mise à jour est acceptée uniquement si elle améliore la qualité de la solution initiale.

Critère d'arrêt : le processus se répète jusqu'à atteindre un nombre maximal d'itérations ou jusqu'à satisfaire un critère de convergence prédéfini.

```
Entrée :
    - f(x) : fonction objectif à minimiser
    - n : taille de la population
    - m : nombre de variables
    - it_max : nombre maximum d'itérations

Initialiser aléatoirement une population de n solutions Xi ∈ ℝ^m

Pour chaque itération i = 1 à it_max :
    Évaluer f(Xi) pour tous les individus
    Identifier la meilleure solution X_best et la pire X_worst
    Pour chaque individu Xi de la population :
        Pour chaque variable j :
            Générer r1 et r2 ∈ [0,1]
            Mettre à jour :
            X'_ij = X_ij + r1 * (X_best_j - |X_ij|) - r2 * (X_worst_j - |X_ij|)
        Accepter X'_i si f(X'_i) est meilleure que f(X_i)
Fin Pour

Retourner la meilleure solution trouvée
```

Figure 1: Pseudocode de l'algorithme JAYA

5. Avantages et inconvénients de l'algorithme JAYA

5.1 Avantages de l'algorithme JAYA :

1. Absence de paramètres spécifiques
   Contrairement à la majorité des métaheuristiques (GA, PSO, ACO…), JAYA ne nécessite aucun paramètre algorithmique spécifique à régler (comme taux de mutation, facteur d'inertie, etc.). Cela simplifie grandement son implémentation et son utilisation.
2. Simplicité de l'implémentation
   L'algorithme est facile à coder, à comprendre et à adapter, ce qui en fait un bon choix pour des chercheurs débutants ou des applications nécessitant un prototypage rapide.
3. Convergence efficace
   Grâce à sa stratégie de mise à jour directe (rapprochement du meilleur – éloignement du pire), JAYA montre de bonnes capacités de convergence rapide, notamment sur des problèmes bien définis.

4. Généricité
   Il peut être appliqué à une large gamme de problèmes : continus, discrets, mono- et multi-objectifs, avec ou sans contraintes.
5. Facilité d'hybridation
   Sa structure simple le rend facilement hybridable avec d'autres techniques (local search, fuzzy logic, etc.), ouvrant la voie à des versions améliorées (comme EJAYA, CLJAYA…).

5.2 Inconvénients de l'algorithme JAYA :

1. Exploration limitée
   En l'absence de mécanismes de diversification spécifiques, JAYA peut stagner dans des optima locaux sur des fonctions très complexes ou fortement multimodales.
2. Dépendance à la qualité de l'initialisation
   Comme beaucoup d'algorithmes populationnels, JAYA peut être sensible à une mauvaise initialisation de la population, ce qui impacte les performances globales.
3. Pas de contrôle explicite de l'équilibre exploration/exploitation
   Contrairement à d'autres algorithmes comme PSO ou DE, JAYA ne dispose pas de mécanismes explicites pour ajuster dynamiquement l'intensification ou la diversification.
4. Performances moindres sur problèmes très bruyants ou dynamiques
   L'absence de stratégie adaptative peut limiter l'efficacité de JAYA dans les environnements incertains, bruités ou évolutifs.

7. Variantes de l'algorithme JAYA

Bien que l'algorithme JAYA se distingue par sa simplicité et l'absence de paramètres spécifiques à régler, certaines limites subsistent, notamment en termes de convergence prématurée ou de manque d'exploration dans des espaces complexes. Pour améliorer ses performances tout en conservant sa philosophie minimaliste, plusieurs variantes ont été proposées. Parmi elles, EJAYA et CLJAYA sont deux extensions notables.

8. Algorithme EJAYA (Enhanced JAYA) :

L'algorithme EJAYA, proposé par Zhang, Chi et Mirjalili (2021), est une extension directe de l'algorithme JAYA. Il a été conçu pour améliorer l'équilibre entre l'exploitation locale (recherche autour de bonnes solutions) et l'exploration globale (recherche dans tout l'espace) tout en conservant les avantages de JAYA, notamment l'absence de paramètres spécifiques à régler[20].

Objectif

Surmonter deux limites de JAYA :

- Dépendance exclusive à la meilleure/pire solution de la population.
- Perte de diversité en cas d'utilisation inefficace de la valeur absolue.

9. Fonctionnement de l'algorithme EJAYA

9.1 Motivation

L'algorithme JAYA, bien qu'efficace, présente deux limitations principales :

- Il repose uniquement sur la meilleure et la pire solution de la population actuelle. Si la meilleure solution est un optimum local, l'algorithme peut y rester bloqué.
- Il utilise une valeur absolue dans son équation de mise à jour, ce qui peut devenir inefficace lorsque les variables sont strictement positives, ce qui est souvent le cas dans les problèmes d'ingénierie.

Pour renforcer la capacité de recherche globale, EJAYA (Enhanced JAYA) a été proposé par Y. Zhang, A. Chi et S. Mirjalili (2021). Il combine deux stratégies complémentaires [20]:

- Exploitation locale améliorée.
- Exploration globale renforcée via une population historique.

9.2 Stratégie d'exploitation locale

Deux points d'attraction sont définis :

Point d'attraction supérieur :

$$Pu = \lambda 3 \cdot x\_best + (1 - \lambda 3) \cdot M$$

Où :

- x_best : meilleure solution actuelle

- $M = (1/N) \sum(i = 1 \text{ à } N)\ x\_i$: moyenne de la population

- $\lambda 3 \in [0,1]$: variable aléatoire

Point d'attraction inférieur :

$$Pl = \lambda 4 \cdot x\_worst + (1 - \lambda 4) \cdot M$$

Où $\lambda 4 \in [0,1]$

Mise à jour des individus :

$$v\_i = x\_i + \lambda 5(Pu - x\_i) - \lambda 6(Pl - x\_i)$$

Où $\lambda 5, \lambda 6 \in [0,1]$

9.3 Stratégie d'exploration globale

EJAYA utilise une population historique aléatoire, X_old, obtenue soit à partir de la population courante soit d'une précédente, puis mélangée aléatoirement[20].
Mise à jour globale:

$$v\_i = x\_i + \kappa \cdot (x\_old\_i - x\_i)$$

Où $\kappa \sim N(0,1)$ suit une distribution normale standard.

Cette approche améliore la capacité de l'algorithme à échapper aux optima locaux en introduisant une perturbation aléatoire plus volatile que les méthodes traditionnelles.

9.4 Pseudocode de EJAYA

```
Entrée :
    N       : taille de la population
    Tmax    : nombre maximal d'évaluations
    D       : nombre de variables
    l, u    : bornes inférieure et supérieure des variables

Initialisation :
    - Générer aléatoirement la population X = {x₁, x₂, ..., x_N} dans [l, u]
    - Générer la population historique X_old (copie initiale de X)
    - Tcurrent = 0

Tant que Tcurrent < Tmax faire :
    1. Évaluer chaque individu xi ∈ X
    2. Identifier :
        x_best  : meilleure solution de X
        x_worst : pire solution de X
        M       : moyenne des solutions de X
    3. Pour chaque individu xi ∈ X faire :
        a. Générer un nombre aléatoire Pselect ∈ [0, 1]
        b. Si Pselect > 0.5 alors :  // Exploitation locale
            - Générer λ₃, λ₄, λ₅, λ₆ ∈ [0, 1]
            - Calculer :
                Pu = λ₃ * x_best + (1 - λ₃) * M
                Pl = λ₄ * x_worst + (1 - λ₄) * M
                v_i = x_i + λ₅ * (Pu - x_i) - λ₆ * (Pl - x_i)
        c. Sinon :  // Exploration globale
            - Générer κ ~ N(0,1)
            - Sélectionner aléatoirement un individu x_old_i de X_old
            - Calculer :
                v_i = x_i + κ * (x_old_i - x_i)
        d. Remplacer x_i par v_i si v_i est meilleur
    4. Mettre à jour Tcurrent ← Tcurrent + N
    5. Réinitialiser X_old ← mélange aléatoire de X

Retourner x_best
```

Figure 2: pseudocode de l'algorithme EJAYA

9.5 Avantages de EJAYA

- ✓ Renforce l'exploration globale (utilisation de la population historique)
- ✓ Améliore la robustesse contre les optima locaux
- ✓ Maintient la simplicité de JAYA
- ✓ Aucune dépendance à des paramètres spécifiques à ajuster

9.6 Inconvénients de EJAYA

- Plus coûteux en mémoire (stockage de X_old)
- Léger surcoût computationnel (génération de population historique, permutations)
- Ne garantit pas une amélioration dans tous les types de

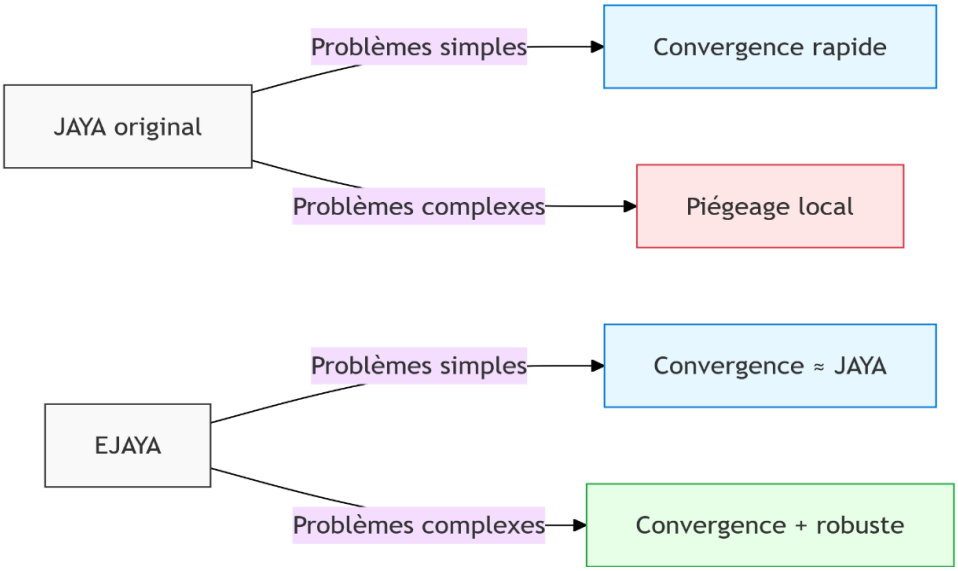


Figure 3: Performance Typique de EJAYA

10. Algorithme CLJAYA (Chaotic Learning JAYA)

L'algorithme CLJAYA, introduit par Chi, Zhang et Mirjalili, est une variante avancée de JAYA intégrant les cartes chaotiques afin d'améliorer la diversification du processus de recherche. Il introduit une dynamique non linéaire dans la génération de nombres aléatoires, pour optimiser la convergence tout en évitant les pièges des optima locaux [3].

Objectif

- Surmonter certaines limites de JAYA et EJAYA :
- Améliorer la diversité de la population.
- Introduire des mécanismes de perturbation contrôlée via des séquences chaotiques.
- Obtenir un équilibre plus efficace entre exploration et exploitation, surtout pour des problèmes complexes ou très sensibles aux initialisations.

11. Fonctionnement de l'algorithme CLJAYA

11.1Motivation

Dans JAYA et EJAYA, les nombres aléatoires utilisés (r, λ, etc.) sont générés de façon uniforme. Or, ces séquences pseudo-aléatoires classiques présentent des régularités pouvant limiter la diversité de la recherche.

CLJAYA remplace ces générateurs classiques par des cartes chaotiques (ex. : carte logistique, carte sinus, carte de Chebyshev) connues pour leur comportement pseudo-aléatoire imprévisible, permettant d'augmenter la diversité sans ajout de paramètres [3].

11.2 Intégration du chaos

Les nombres aléatoires dans les équations de mise à jour sont générés à l'aide d'une carte chaotique, par exemple la carte logistique [3]:

$$z(n+1) = \mu * z(n) * (1 - z(n)), avec\ \mu \in [3.57, 4]$$

Où :
- $z(n) \in (0,1)$: état chaotique à l'itération n
- μ : paramètre de la carte
Les nombres chaotiques z(n) remplacent les variables aléatoires classiques dans les formules de JAYA ou EJAYA.

11.3 Mise à jour des solutions

La formule classique de JAYA devient [3]:

$$x\_i(new) = x\_i + z1 * (x\_best - |x\_i|) - z2 * (x\_worst - |x\_i|)$$

Où $z1, z2$ sont issus d'une carte chaotique.
CLJAYA peut aussi intégrer du chaos dans :

- L'initialisation de la population
- L'intensification (exploitation)
- L'exploration globale (en combinaison avec EJAYA)

11.4 Pseudocode de CLJAYA

```
Entrée :
    N         : Taille de la population
    Tmax      : Nombre maximal d'évaluations
    D         : Dimension du problème
    [l, u]    : Bornes inférieure et supérieure des variables
    μ         : Paramètre du système chaotique (ex: μ = 3.9 pour la carte logistique

1. Initialiser la population X = {x₁, x₂, ..., x_N} aléatoirement dans [l, u]
2. Initialiser l'état chaotique z₀ ∈ (0,1)

3. Évaluer chaque solution xi ∈ X
4. Identifier les solutions x_best et x_worst

5. Tcurrent ← 0

6. Tant que Tcurrent < Tmax faire :
    a. Pour chaque individu xi ∈ X faire :
        i.   Générer z₁, z₂ à l'aide d'une carte chaotique (ex: logistique)
             z ← μ * z * (1 - z)
        ii.  Mettre à jour chaque xi selon :
             xᵢ ← xᵢ + z₁ · (x_best - |xᵢ|) - z₂ · (x_worst - |xᵢ|)
        iii. Appliquer les bornes [l, u] si nécessaire
    b. Réévaluer la population mise à jour
    c. Mettre à jour x_best si une meilleure solution est trouvée
    d. Tcurrent ← Tcurrent + N
    e. Mettre à jour la séquence chaotique (z ← μ · z · (1 - z))

7. Retourner x_best comme solution optimale

Sortie :
    Meilleure solution trouvée x_best et sa valeur objective
```

Figure 4: pseudocode de CLJAYA

11.5 Avantages de CLJAYA

- ✓ Maintient la simplicité de JAYA (peu de paramètres)
- ✓ Introduit une variabilité non déterministe contrôlée

- ✓ Améliore la capacité d'échappement aux optima locaux
- ✓ Applicable à une grande variété de problèmes (ingénierie, systèmes complexes, data science)

11.6 Inconvénients de CLJAYA

- Complexité d'implémentation légèrement supérieure (cartes chaotiques à choisir/configurer)
- Nécessite parfois des tests empiriques pour choisir la meilleure carte
- Peut-être plus sensible aux conditions initiales du chaos

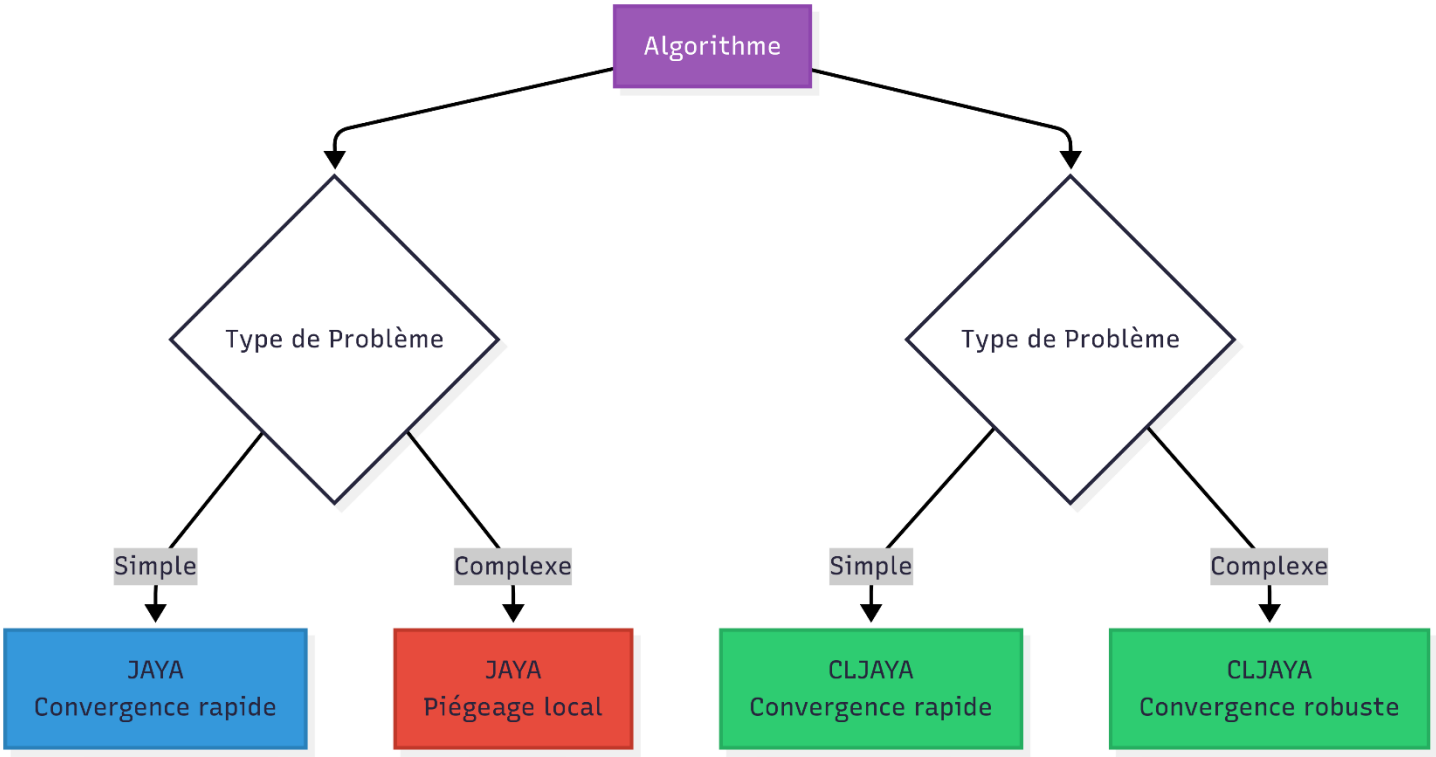


Figure 5: Performance Typique de CLJAYA

12. Comparaison entre JAYA, EJAYA et CLJAYA

Critère	JAYA	EJAYA	CLJAYA
Auteur(s)	Rao (2016)	Zhang, Chi, Mirjalili (2021)	Chi, Zhang, Mirjalili
Principe	Approche de mise à jour en direction du meilleur et éloignement du pire	Introduction de stratégies d'exploitation et d'exploration basées sur la moyenne de la population et une population historique	Utilisation de séquences chaotiques dans les mises à jour
Exploration globale	Modérée	Renforcée via la population historique	Améliorée par des perturbations chaotiques

Critère	JAYA	EJAYA	CLJAYA
Exploitation locale	Basée uniquement sur x_best et x_worst	Basée sur des attracteurs locaux (x_best, x_worst, moyenne M)	Identique à JAYA ou EJAYA, mais pilotée par des valeurs chaotiques
Diversité de la population	Moyenne	Améliorée	Forte (grâce aux cartes chaotiques)
Paramètres spécifiques	Aucun	Aucun	Paramètre de chaos (ex. μ pour la carte logistique)
Complexité d'implémentation	Très simple	Moyennement simple	Légèrement plus complexe (chaos)
Sensibilité aux optima locaux	Élevée	Moyenne	Faible
Adaptabilité	Problèmes simples à moyens	Problèmes contraints ou complexes	Problèmes très complexes ou sensibles à l'initialisation

Tableau 2:Tableau comparatif des algorithme JAYA, EJAYA et CLJAYA

13. Conclusion

Dans ce chapitre, nous avons présenté l'algorithme JAYA et ses deux principales variantes : EJAYA et CLJAYA. Chacune de ces approches vise à surmonter certaines limitations de l'algorithme de base.

L'algorithme JAYA, par sa simplicité et son absence de paramètres spécifiques, représente une solution efficace pour une grande variété de problèmes d'optimisation. Toutefois, sa dépendance exclusive à la meilleure et la pire solution peut entraîner une convergence prématurée.

L'extension EJAYA améliore cette situation en introduisant deux stratégies complémentaires : une exploitation locale basée sur des points d'attraction calculés dynamiquement, et une exploration globale appuyée par l'utilisation d'une population historique. Cette approche renforce la diversité de la recherche sans alourdir significativement la complexité de l'algorithme.

Enfin, la variante CLJAYA intègre des mécanismes chaotiques pour améliorer encore la diversité de la population. En remplaçant les générateurs pseudo-aléatoires par des cartes chaotiques, CLJAYA augmente la capacité de l'algorithme à échapper aux optima locaux, notamment dans les problèmes hautement non linéaires ou sensibles aux conditions initiales.

Ainsi, le choix entre ces trois variantes dépend fortement du contexte du problème à résoudre. Pour des problèmes simples, JAYA peut suffire. Pour des cas complexes nécessitant une robustesse accrue, EJAYA ou CLJAYA sont plus adaptés.

Chapitre 03 : conception et implémentation

1. Introduction

Après avoir présenté dans le chapitre précédent les notions et concepts fondamentaux liés à l'optimisation et aux métaheuristiques, en particulier l'algorithme JAYA et ses variantes EJAYA et CLJAYA, nous abordons dans ce chapitre la conception et l'implémentation de notre application, structurée autour de deux volets complémentaires :

Le premier volet concerne l'application des algorithmes susmentionnés à des problèmes d'optimisation continue, à travers les fonctions de benchmark standards proposées par les suites CEC 2017 et CEC 2019.

Le deuxième volet traite de l'adaptation de ces algorithmes à un problème d'optimisation discrète, à savoir le regroupement automatique d'élèves en groupes équilibrés selon plusieurs critères pédagogiques (moyenne, sexe, nombre d'échecs…).

Nous commencerons par la présentation de l'environnement de développement utilisé, avant de détailler l'objectif de notre système, son architecture générale, et la description de ses modules. Enfin, nous introduirons les diagrammes de flux de données (DFD) comme outil de modélisation du système, étape essentielle qui précède l'implémentation finale.

2. Conception

2.1 Description Fonctionnelle du Système

Le système est divisé en deux volets :

- **Optimisation Continue** : permet l'exécution et la comparaison des performances des algorithmes sur des fonctions mathématiques (benchmarks CEC).
- **Regroupement d'élèves** : permet de créer automatiquement des groupes équilibrés d'élèves en fonction de critères pédagogiques.

Chaque volet dispose de modules fonctionnels propres, intégrés dans une interface graphique interactive.

2.2 Système d'optimisation sur Fonctions mathématique de test

Le premier volet de notre système consiste à évaluer les performances des trois algorithmes sur des fonctions mathématique de test standards. Ces fonctions sont reconnues pour leur complexité (fonctions multimodales, non séparables, avec bruit ou rotation) et représentent un terrain d'expérimentation rigoureux pour tester la robustesse, la vitesse de convergence et la précision des algorithmes.

Chaque fonction est résolue plusieurs fois (exécutions multiples) afin d'obtenir des statistiques fiables (meilleure solution, moyenne, écart-type, temps d'exécution). Les résultats sont ensuite visualisés sous forme de :

- Courbes de convergence,
- Boîtes à moustaches (boxplots),

- Tableaux comparatifs.

2.2.1 Objectif du système

Ce système a pour objectif principal d'évaluer les performances de différents algorithmes d'optimisation (JAYA, EJAYA et CLJAYA) sur des fonctions mathématiques standardisées. Il s'inscrit dans une démarche expérimentale rigoureuse, permettant de comparer les capacités de ces algorithmes à résoudre des problèmes continus complexes.

- Implémenter une interface utilisateur interactive (GUI)
  Permettre à l'utilisateur de :
  - Sélectionner un benchmark,
  - Choisir une fonction de test,
  - Configurer les paramètres de l'algorithme (taille de population, itérations, nombre de runs).
- Exécuter les algorithmes sur des fonctions de benchmark
  Évaluer la performance de chaque algorithme sur différentes fonctions caractérisées par :
  - La multimodalité (présence de plusieurs minima locaux),
  - La non-séparabilité des variables,
  - La rotationdesaxes,
  - L'ajout de bruit ou la complexité hybride.
- Mesurer les performances de manière rigoureuse
  Calculer des indicateurs statistiques sur plusieurs exécutions :
  - Meilleure solution trouvée (fitness),
  - Moyenne et écart-type des résultats,

Temps d'exécution global.

- Visualiser les résultats
  Générer automatiquement :
  - Des courbes de convergence (fitness vs itérations),
  - Des boxplots des résultats finaux,
  - Des histogrammes de distribution,
  - Un tableau comparatif des algorithmes.
- Comparer objectivement les algorithmes
  Fournir un module de comparaison simultanée des performances de JAYA, EJAYA et CLJAYA sur une même fonction pour identifier :
  - Le plus rapide,
  - Le plus stable (faible écart-type),
  - Le plus précis (fitness minimal).

2.2.2 Architecture Modulaire du Système

Le système repose sur quatre modules principaux :

1. **Module Interface Utilisateur (GUI)** : Interagit avec l'utilisateur pour configurer et lancer les optimisations.

2. **Module Benchmark** : Gère les fonctions CEC.
3. **Module Optimisation** : Implémente les algorithmes JAYA, EJAYA et CLJAYA.
4. **Module Visualisation** : Affiche les courbes de convergence, statistiques, boxplots, etc.

Cette partie du système s'inscrit donc dans une logique d'analyse expérimentale approfondie des performances algorithmiques.

2.2.3 Diagramme de cas d'utilisation du système d'optimisation sur fonctions de test

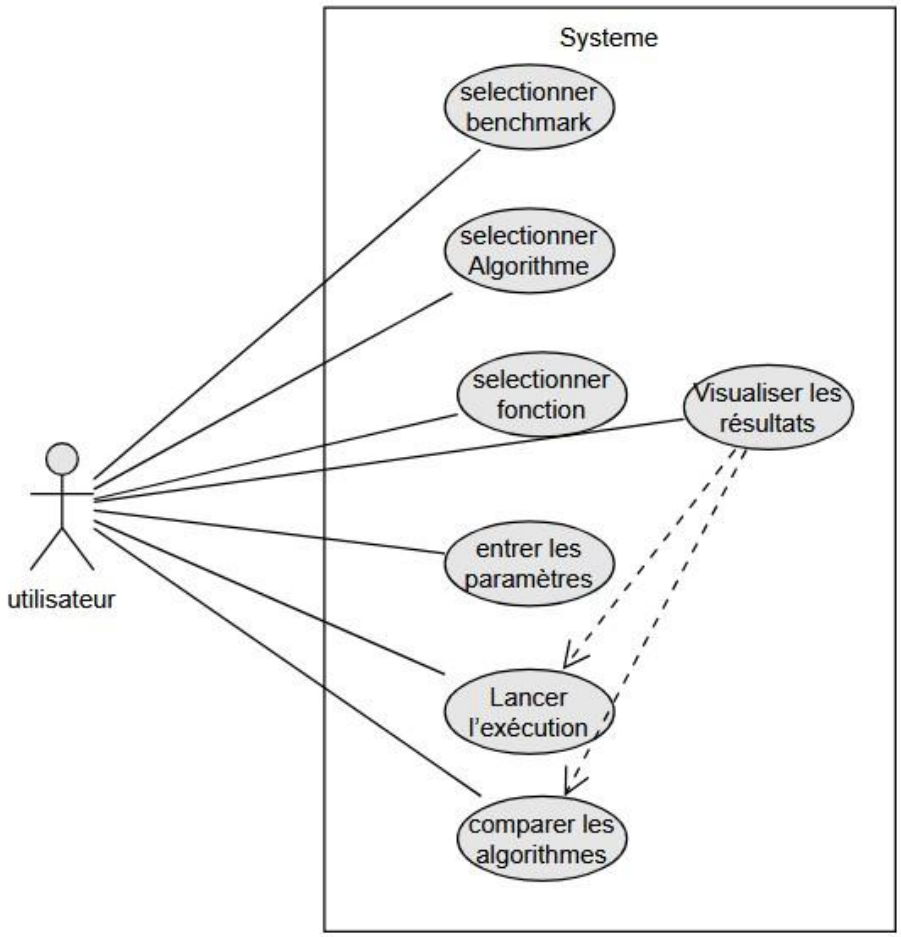


Figure 6: diagramme de cas d'utilisation comparer les algorithmes

2.2.4 Diagramme de séquence du système d'optimisation sur fonctions de test

Ci-dessous le diagramme de séquence d'un scénario d'utilisation du système

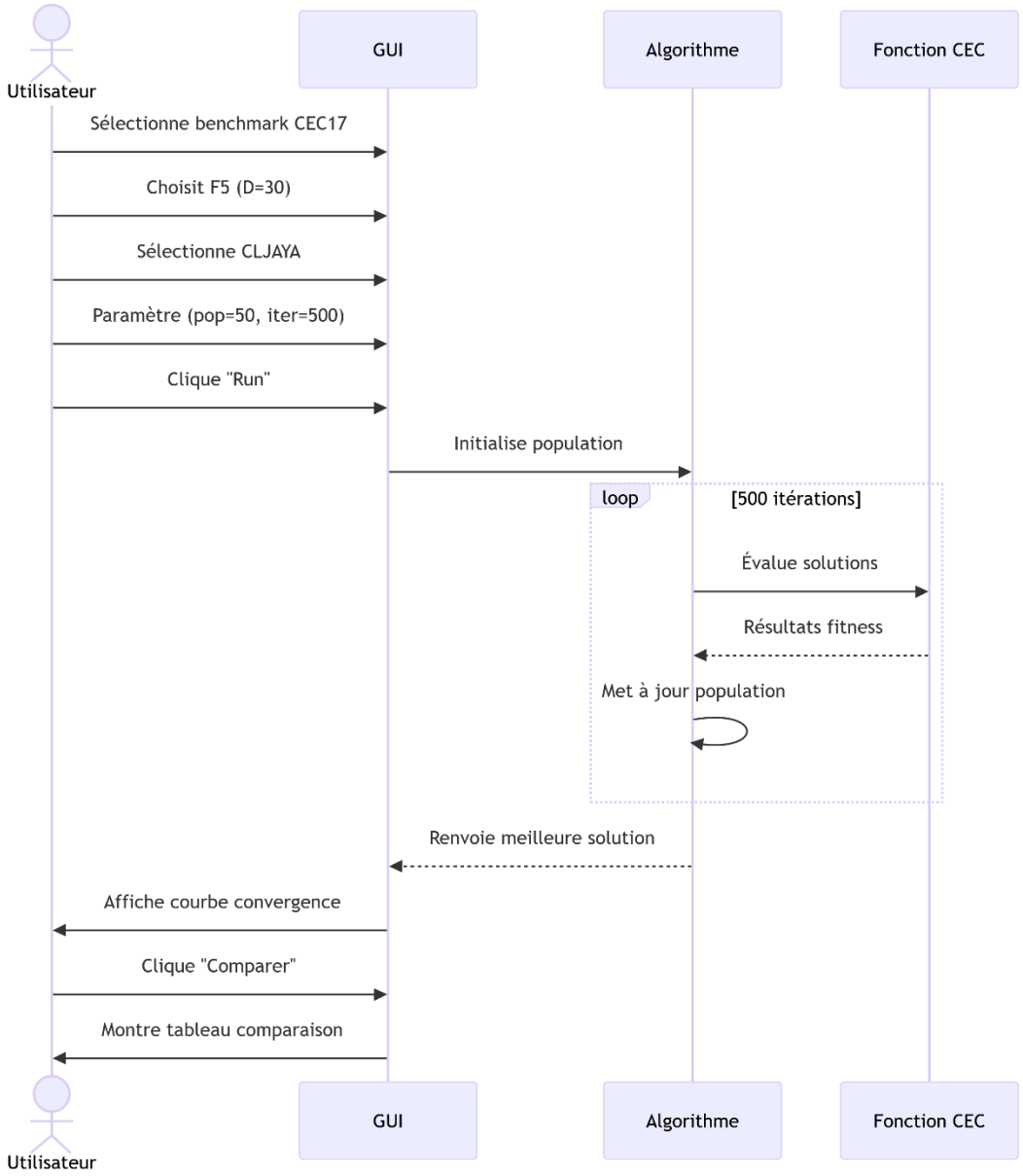


Figure 7: d'diagramme de séquence du système d'optimisation sur fonctions de test

3. Système de regroupement des élèves

Le second volet du système consiste à adapter les algorithmes à un problème discret réel : le regroupement automatique des élèves dans des groupes équilibrés selon plusieurs critères. Contrairement aux fonctions mathématiques de test, ce problème implique une modélisation combinatoire où chaque solution représente une configuration possible de groupes.

L'algorithme JAYA et ses variantes ont été modifiés pour :

- Traiter des variables discrètes (groupes, indices entiers),
- Intégrer des contraintes pédagogiques (équilibre selon la moyenne, le sexe, les redoublements…),
- Optimiser une fonction de coût personnalisée reflétant la qualité du regroupement.

Cette adaptation démontre la flexibilité des algorithmes JAYA, EJAYA et CLJAYA dans des contextes d'optimisation hétérogènes, allant des problèmes théoriques continus aux applications concrètes à variables discrètes.

3.1 Objectif du système

L'objectif principal du système de regroupement des élèves est de concevoir une solution intelligente, automatisée et équitable permettant de répartir des élèves en groupes équilibrés selon plusieurs critères pédagogiques. Ce système s'intègre dans un contexte éducatif réel, où le regroupement des élèves a un impact direct sur la dynamique de classe, la réussite scolaire, et l'équité.

1. Automatiser le processus de regroupement
   Éviter la répartition manuelle fastidieuse et souvent subjective en utilisant des algorithmes d'optimisation.
2. Assurer un équilibre entre les groupes
   Les groupes doivent être équilibrés sur les critères suivants :
   - La moyenne générale des élèves,
   - Les moyennesdes modules,
   - Le sexe (répartition équitable filles/garçons),
   - Le nombre de redoublements,
   - Éventuellement d'autres paramètres comme l'âge, ou des besoins spécifiques.
3. Minimiser les déséquilibres
   Réduire les écarts entre les groupes en minimisant une fonction de coût qui mesure le déséquilibre global.
4. Adapter les algorithmes d'optimisation (JAYA, EJAYA, CLJAYA)
   Transformer les versions originales de ces algorithmes continus pour qu'ils puissent gérer des variables discrètes (ex. : affectation d'un élève à un groupe).

3.2 Architecture Modulaire du Système

Le système repose sur cinq modules principaux :

1. Module Interface Utilisateur (GUI) : Interagit avec l'utilisateur pour configurer et lancer les optimisations.
2. Module Optimisation : Implémente les algorithmes JAYA, EJAYA et CLJAYA.
3. Module Clustering Élèves : Spécialise dans la résolution du problème discret.
4. Module Visualisation : Affiche les courbes de convergence, statistiques, boxplots, etc.
5. Module Exportation : Sauvegarde les résultats sous forme de fichiers Excel.

3.3 Diagramme de cas d'utilisation du système de regroupement des élèves :

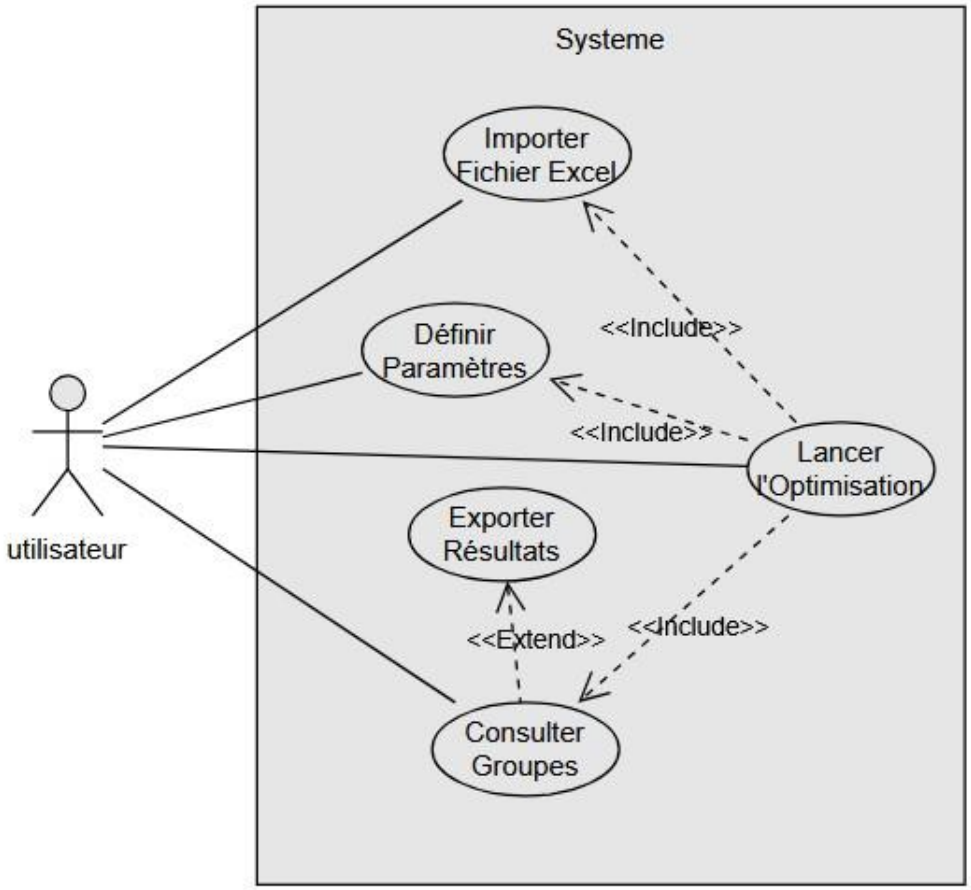


Figure 8: Diagramme de cas d'utilisation du système de regroupement des élèves

3.4 Diagramme de séquence d'une du système de regroupement des élèves :

Ci-dessous le diagramme de séquence d'un scénario d'utilisation du système

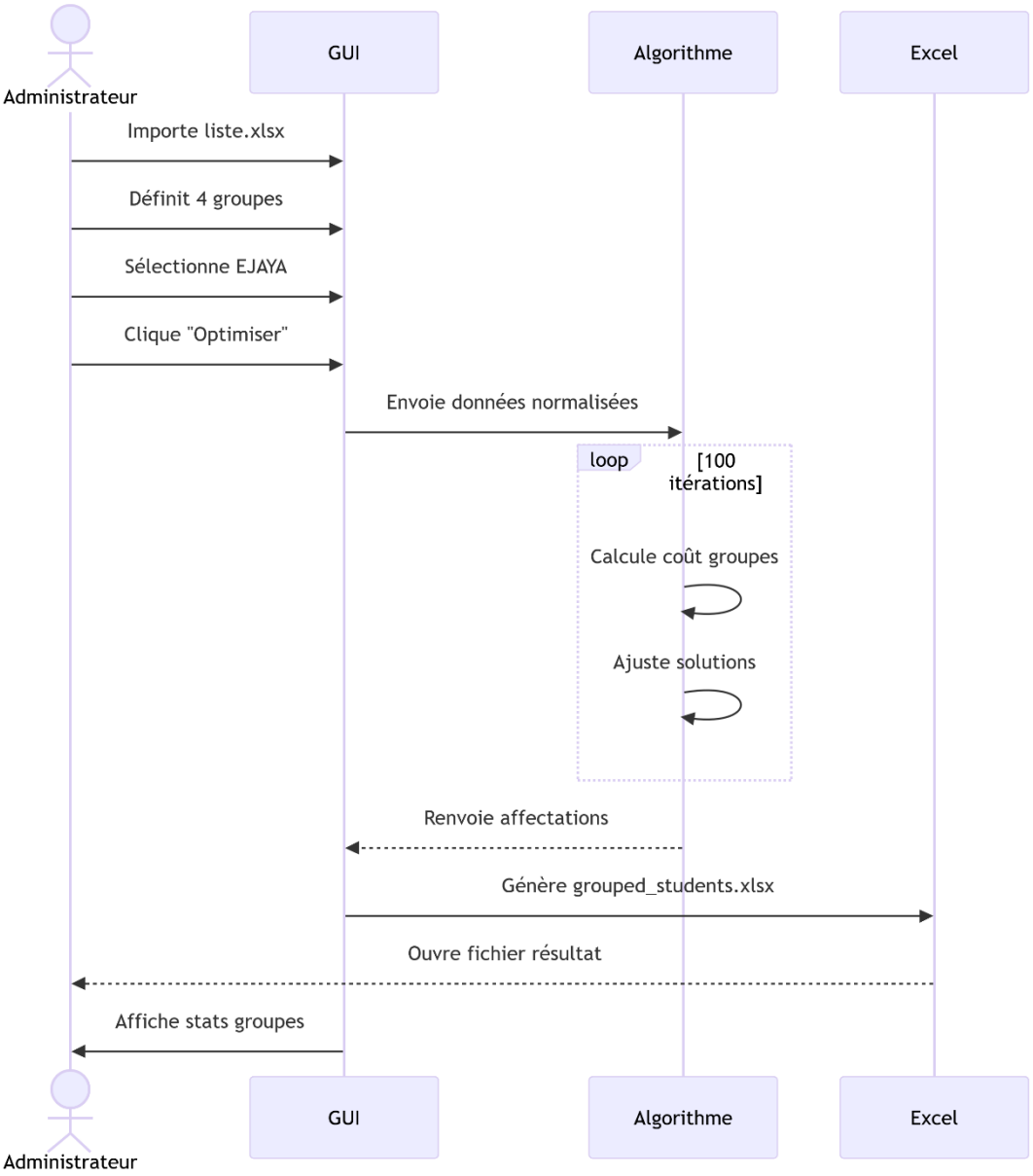


Figure 9: diagramme de séquence du système de regroupement des élèves

3.5 Fonctionnement de système de regroupement des élèves :

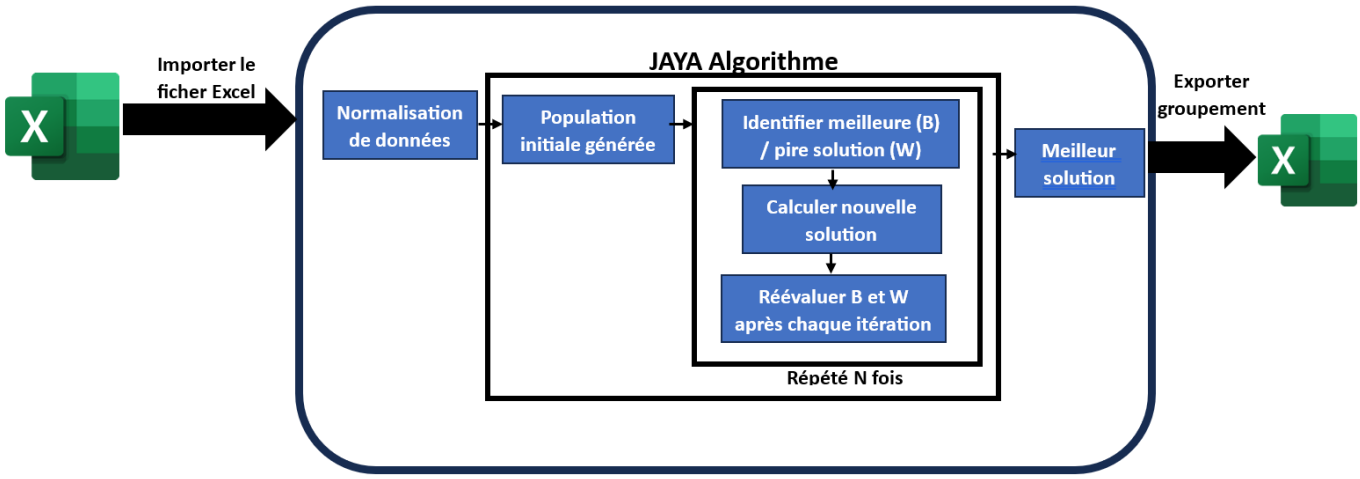


Figure 10: Fonctionnement de système de regroupement des élèves

Le système a pour but de répartir automatiquement des élèves en groupes homogènes selon des critères pédagogiques spécifiques :

- Équilibre académique (moyennes générales et par matière)
- Parité de genre
- Distribution équitable de l'âge et des redoublements
- Taille similaire des groupes (±1 élève)

Il remplace un processus manuel fastidieux par une solution automatisée, rapide et objective.

3.5.1 Architecture Globale

Le système fonctionne en 4 étapes clés :

1. Importation des données
    - Lecture d'un fichier Excel contenant les profils élèves
    - Validation des données (colonnes obligatoires présentes)
    - Conversion des données brutes en format numérique standardisé
2. Configuration
    - Choix du nombre de groupes souhaités
    - Sélection de l'algorithme d'optimisation (JAYA/EJAYA/CLJAYA)
    - Paramétrage de la précision (taille population, nombre d'itérations)
3. Optimisation
    - Exécution de l'algorithme sélectionné
    - Génération et évaluation de solutions candidates
    - Application des contraintes pédagogiques
4. Résultats
    - Attribution des élèves aux groupes
    - Génération de statistiques détaillées
    - Export vers Excel et impression des listes

3.5.2 Rôle des Algorithmes d'Optimisation

Problème central : Trouver la meilleure partition de n élèves en m groupes respectant des contraintes multiples.

Approche :

- Chaque solution candidate est représentée comme un vecteur : [groupe_élève1, groupe_élève2, ..., groupe_élèveN]
- Qualité mesurée par une fonction de coût qui pénalise :
  - Les déséquilibres de caractéristiques (notes, genre, etc.)
  - Les écarts de taille entre groupes
  - Les groupes incomplets

a) JAYA : Version Standard de l'algorithme

Principe général L'algorithme JAYA repose sur un principe simple et efficace : orienter les solutions actuelles vers la meilleure solution identifiée tout en s'éloignant de la pire. Il s'agit d'un algorithme sans paramètres, ce qui le rend attractif pour de nombreuses applications d'optimisation.

Mécanisme de fonctionnement Identification des solutions extrêmes : Pour chaque itération, l'algorithme détermine les solutions extrêmes au sein de la population courante :

- La meilleure solution (celle avec le plus faible coût ou la meilleure aptitude),
- La pire solution (celle ayant la plus faible performance).

Mise à jour des solutions : Chaque solution est mise à jour selon la formule suivante :

$$Nouvelle\ solution = Actuelle + r_1 \times (\text{Meilleure} - \text{Actuelle}) - r_2 \times \text{Pire} - \text{Actuelle}$$

Ou r1 et r2 sont deux variables aléatoires uniformément distribuées dans [0,1]. Cette stratégie vise à améliorer continuellement chaque solution en la rapprochant des zones prometteuses de l'espace de recherche tout en s'éloignant des zones non performantes.

Contraintes d'intégrité : Dans le cas de la répartition en groupes (par exemple, des élèves), certaines contraintes doivent être respectées :

- Les valeurs de la solution sont converties en valeurs entières (numéros de groupes),
- Une procédure de correction des tailles de groupes est appliquée pour s'assurer que l'équilibre en effectif est maintenu (±1 élève/groupe).

Processus d'Optimisation dans JAYA

L'algorithme JAYA suit un processus itératif simple mais efficace, orienté vers l'amélioration progressive des solutions par attraction vers les meilleures et répulsion des moins bonnes. Il ne requiert aucun paramètre d'apprentissage, ce qui facilite sa mise en œuvre.

1. Initialisation

Une population initiale de solutions est générée aléatoirement. Chaque solution représente une répartition des élèves entre les groupes, en respectant les contraintes pédagogiques :

- Taille équilibrée des groupes (écart maximal ±1 élève),
- Répartition équitable des sexes,
- Distribution homogène des moyennes scolaires,
- Dispersion raisonnable des cas d'échec scolaire.

2. Évaluation

Chaque solution est évaluée par la fonction de coût suivante :

$$Coût = \alpha \times Déséquilibre\_\{caractéristiques\} + \beta \times Pénalité\_\{taille\}$$

Les deux composantes principales sont :

Le déséquilibre des caractéristiques : mesure les écarts de moyennes, de répartition garçons/filles, d'élèves en difficulté.
La pénalité de taille : reflète la déviation de la taille des groupes par rapport à la cible.
3. Amélioration itérative
À chaque itération, JAYA met à jour chaque solution selon le principe suivant :

$$Nouvelle\ solution = Actuelle + r_1 \times (Meilleure - Actuelle) - r_2 \times (Pire - Actuelle)$$

Où r1 et r2 sont des nombres aléatoires uniformément distribués dans [0,1].
Cette formule vise à améliorer la qualité de chaque solution :
En l'attirant vers la meilleure solution actuelle,
En l'éloignant de la pire solution.
Après chaque mise à jour :
Les valeurs sont arrondies pour obtenir des numéros de groupes entiers,
Une correction est effectuée pour assurer que chaque groupe respecte la contrainte de taille (±1 élève).
4. Convergence
L'algorithme s'exécute pendant un nombre d'itérations prédéfini ou jusqu'à l'absence d'amélioration significative. À la fin, la solution présentant le coût le plus faible est sélectionnée comme résultat optimal.
5. Contrôle des contraintes
Deux mécanismes garantissent la faisabilité des solutions :
Réparation : transfert d'élèves pour corriger les groupes trop grands ou trop petits.
Pénalités : appliquées dans la fonction de coût pour encourager le respect strict des contraintes pédagogiques et structurelles.

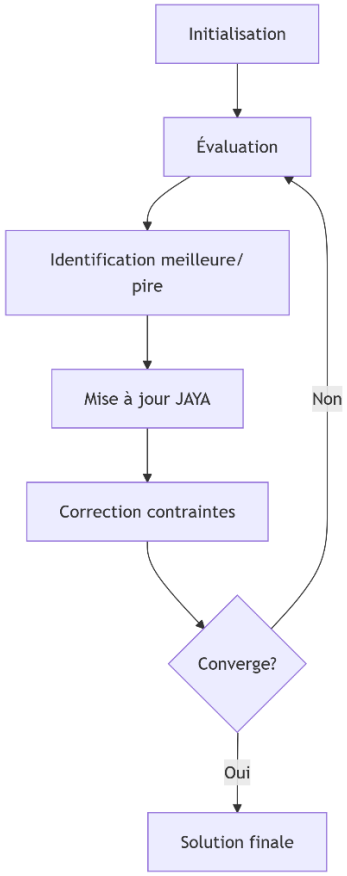


Figure 11: Schéma Conceptuel JAYA

b) EJAYA : Version Élitiste de l'algorithme JAYA

Innovation introduite L'algorithme EJAYA constitue une extension de l'algorithme JAYA classique, en y intégrant un mécanisme de mémoire élitiste. Cette version élitiste vise à conserver les meilleures solutions rencontrées au cours de l'optimisation afin d'orienter plus efficacement la recherche vers des régions prometteuses de l'espace de solution.
Principes de fonctionnement Constitution d'un pool élitiste : Une mémoire appelée "pool d'élites" est maintenue durant le processus. Ce pool regroupe les solutions les plus performantes trouvées jusqu'à présent. Il est mis à jour dynamiquement afin de ne conserver que les individus les plus pertinents, selon un critère de qualité prédéfini.
Mise à jour enrichie : Lors de la mise à jour de chaque solution, deux sources d'amélioration sont considérées :

- La meilleure solution globale actuelle,
- Une solution aléatoire extraite du pool d'élites.

Cette double influence permet de mêlier efficacement exploitation (utilisation de la meilleure solution) et exploration (diversité introduite par l'élite aléatoire).
Avantage adaptatif : L'introduction d'un mécanisme élitiste permet d'éviter la perte de bonnes solutions, tout en renforçant la convergence vers des optima de haute qualité. De plus, en évitant de ne se baser que sur la meilleure solution unique, EJAYA préserve une certaine diversité dans la population, ce qui est essentiel pour éviter les minima locaux.
Processus d'Optimisation dans EJAYA
L'algorithme EJAYA suit une démarche itérative comparable à JAYA, mais enrichie par un mécanisme élitiste visant à capitaliser sur les meilleures solutions rencontrées. Ce processus permet une meilleure convergence tout en conservant la diversité des solutions.
1. Initialisation
Une population initiale de solutions admissibles est générée, en respectant les contraintes pédagogiques de base. Chaque solution correspond à une répartition des élèves entre les groupes, avec un écart de taille limité à ±1 élève.
Les contraintes considérées incluent :

- L'équilibre entre les sexes,
- La répartition des niveaux de performance (moyennes),
- La dispersion des élèves en difficulté.

2. Évaluation
Chaque solution est évaluée à l'aide de la fonction de coût :

$$Coût = \alpha \times Déséquilibre\_\{caractéristiques\} + \beta \times Pénalité\_\{taille\}$$

Les solutions les plus performantes (faible coût) sont identifiées pour alimenter un pool d'élites. Ce dernier est maintenu pendant tout le processus, avec mise à jour dynamique.
3. Amélioration itérative
À chaque itération, les solutions sont modifiées selon le principe de JAYA, enrichi par l'approche élitiste. La mise à jour repose sur :
La meilleure solution globale actuelle,
Une solution tirée aléatoirement du pool d'élites.
La nouvelle solution est calculée selon la formule :

$$Nouvelle\ solution = Actuelle + r_1 \times (Meilleure - Actuelle) - r_2 \times (\text{Élite} - Actuelle)$$

Les coefficients r1 et r2 sont aléatoires et varient entre 0 et 1.

Cette double influence permet de mieux équilibrer exploitation et exploration. Après chaque mise à jour, une correction est appliquée pour assurer la conformité des tailles de groupes.

4. Convergence

Le processus se poursuit pendant un nombre fixé d'itérations ou jusqu'à stabilisation des coûts. Pour accroître la robustesse du résultat, plusieurs exécutions indépendantes sont réalisées, et la meilleure solution globale est conservée.

5. Contrôle des contraintes

Deux mécanismes sont utilisés pour garantir la validité des solutions :

Réparation : ajuste automatiquement les groupes en cas de déséquilibre (transferts d'élèves, équilibrage strict ±1).

Pénalités : intégrées à la fonction de coût pour dissuader les déséquilibres excessifs.

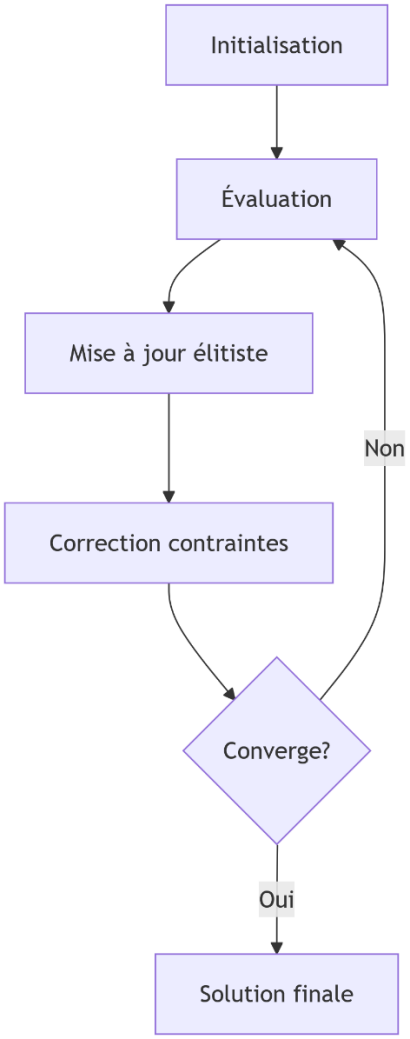


Figure 12: Schéma Conceptuel EJAYA

c) CLJAYA : Une approche hybride intégrant le clustering

- Innovation introduite
  L'algorithme CLJAYA représente une extension de l'algorithme JAYA en y intégrant une connaissance locale issue d'une analyse de clustering. Cette hybridation permet une meilleure prise en compte des structures internes des données, notamment les similarités entre individus.

- Principes de fonctionnement
  Clustering préalable :
  Un algorithme de type k-means est appliqué en amont afin d'identifier les centres représentatifs des groupes potentiels. Chaque individu est alors associé à un centre.
- Évaluation de la proximité locale :
  Pour chaque individu, la distance à son centre de groupe est calculée. Cette information reflète la cohérence locale de sa position par rapport au reste du groupe.
- Mise à jour dirigée :
  La mise à jour des solutions est guidée à la fois par le principe d'amélioration globale de JAYA (rapprochement vers la meilleure solution et éloignement de la pire), et par un biais local tenant compte de la distance au centre du groupe.
  La formule de mise à jour peut s'exprimer conceptuellement comme suit :
  Nouvelle solution = Ancienne solution + amélioration globale + ajustement local
  Où l'ajustement local favorise les individus mieux positionnés par rapport à leur cluster.
- Avantage adaptatif :
  Grâce à cette intégration du contexte local, CLJAYA offre une meilleure capacité à s'adapter aux structures complexes et hétérogènes des données, notamment dans les cas où la répartition des individus n'est pas uniforme.

Processus d'Optimisation dans CLJAYA

L'algorithme CLJAYA suit une démarche itérative structurée en plusieurs phases complémentaires, visant à produire une répartition équilibrée des élèves en groupes, tout en respectant les contraintes pédagogiques et les caractéristiques individuelles.

1. Initialisation
   La phase initiale consiste à générer un ensemble de solutions admissibles représentant différentes répartitions des élèves entre les groupes. Ces solutions sont créées de manière à assurer un équilibre de taille approximatif, en imposant un écart maximal de ±1 élève entre les groupes.
   La répartition des élèves est réalisée aléatoirement, tout en respectant un ensemble de contraintes de base telles que :

   - La parité entre les sexes,
   - La distribution homogène des niveaux scolaires (moyennes),
   - La dispersion équitable des cas d'échec scolaire.

   Cette diversité initiale est essentielle pour garantir une bonne exploration de l'espace de recherche.

2. Évaluation
   Chaque solution candidate est évaluée à l'aide d'une fonction de coût définie pour mesurer son niveau d'équilibre et sa conformité aux objectifs pédagogiques.
   La fonction de coût comporte deux composantes principales :

$$\text{Coût} = \alpha \cdot \text{Déséquilibre}_{\text{caractéristiques}} + \beta \cdot \text{Pénalité}_{\text{taille}}$$

- Le déséquilibre des caractéristiques reflète les écarts statistiques entre les groupes (moyenne, répartition des sexes, taux d'échec, etc.).
- La pénalité de taille sanctionne les écarts par rapport à la taille idéale du groupe. Les coefficients α\alpha et β\beta permettent d'ajuster l'importance relative de chaque critère dans le processus d'optimisation.

3. Amélioration itérative
   À chaque itération, les solutions sont améliorées selon une logique inspirée de l'algorithme JAYA, enrichie par l'approche CLJAYA. Le processus comprend :

- La sélection de solutions de référence : les meilleures et pires solutions actuelles sont identifiées. Des solutions dites « élites » peuvent également être conservées pour guider la recherche.
- La mise à jour des solutions : chaque solution est modifiée en cherchant à se rapprocher de la meilleure solution tout en s'éloignant de la pire. Dans CLJAYA, cette mise à jour intègre également une composante locale issue du clustering (distance au centre du groupe).
- La correction systématique des tailles de groupes : après chaque mise à jour, une procédure d'ajustement est appliquée pour s'assurer que la contrainte de taille (±1 élève par groupe) est toujours respectée.

4. Convergence
   Le processus d'amélioration se répète pendant un nombre fixé d'itérations NN, ou jusqu'à ce qu'aucune amélioration significative ne soit observée. Afin d'augmenter les chances d'atteindre un optimum global :

- Plusieurs exécutions indépendantes sont effectuées, chacune avec des conditions initiales différentes.
- La meilleure solution globale parmi toutes les exécutions est conservée comme solution finale.

5. Contrôle des contraintes
   Deux mécanismes sont mis en place pour garantir que les solutions restent réalisables et pédagogiquement acceptables :

Réparation des solutions
Un module de réparation automatique s'applique dès qu'un déséquilibre est détecté :

- Les groupes présentant un excès ou un déficit d'élèves sont identifiés.
- Des transferts ciblés d'élèves sont réalisés entre groupes.
- Le tout se fait dans le respect strict de la contrainte ±1 élève/groupe.
  Ce mécanisme agit comme un filet de sécurité pour corriger les dérives liées à l'optimisation stochastique.

Pénalités dans la fonction de coût
La fonction de coût inclut des pénalités dynamiques qui augmentent progressivement en cas de non-conformité :

- Une pénalisation légère est appliquée pour de petits écarts.
- Une pénalisation accrue s'enclenche en cas d'écart supérieur à un élève, ou de forte asymétrie sur un critère pédagogique (par exemple, concentration d'élèves faibles dans un même groupe).
- Les critères pédagogiques peuvent être pondérés différemment selon les priorités de l'établissement (égalité des sexes, répartition des élèves en difficulté, etc.).

3.5.3 Fonction de coût – JAYA, EJAYA, CLJAYA

Dans les trois variantes de l'algorithme, la fonction de coût joue un rôle central dans l'évaluation et la sélection des solutions. Elle permet de quantifier la qualité d'une solution en fonction de critères pédagogiques et de contraintes de répartition.

a) Fonction de coût commune

Toutes les versions utilisent une forme générale de la fonction de coût donnée par :

$$Coût = \alpha \times Déséquilibre\_\{caractéristiques\} + \beta \times Pénalité\_\{taille\}$$

b) Explication des termes :

Déséquilibre_{caractéristiques} : mesure les écarts statistiques entre les groupes concernant :

- La répartition des sexes,
- La moyenne des notes,
- La concentration d'élèves en échec.

Pénalité_{taille} : évalue la déviation de la taille des groupes par rapport à la taille idéale (±1 élève). Elle est nulle lorsque la contrainte de taille est respectée, et croît progressivement en cas de dépassement.

α et β : coefficients de pondération ajustables pour donner plus d'importance à un critère selon les priorités de l'établissement.

c) Particularités selon l'algorithme :

- JAYA : La fonction de coût est calculée à chaque itération pour toutes les solutions, permettant une amélioration globale. Aucune mémoire des bonnes solutions précédentes n'est conservée.
- EJAYA : La même fonction est utilisée, mais elle s'applique aussi à la sélection et au maintien des solutions dans le pool d'élites. Seules les solutions ayant un coût plus faible sont conservées dans ce pool.
- CLJAYA : En plus des composantes classiques, un terme local (distance au centre de cluster) est intégré indirectement via la mise à jour, influençant donc la génération de nouvelles solutions mais pas explicitement la fonction de coût.

3. Implémentation

3.1 Présentation de l'environnement de travail

Le développement de notre application s'appuie sur un ensemble d'outils logiciels et matériels soigneusement choisis pour répondre aux besoins à la fois de performance, de convivialité et de compatibilité avec les bibliothèques scientifiques et graphiques nécessaires à la mise en œuvre des algorithmes d'optimisation et de l'interface utilisateur.

3.2 Langage de programmation

Le langage principal utilisé dans ce projet est MATLAB, un environnement de calcul numérique très utilisé dans le domaine de l'optimisation et de l'intelligence artificielle. MATLAB offre de nombreuses fonctionnalités intégrées pour le traitement matriciel, la visualisation des résultats, l'interface graphique (GUI), et permet l'intégration de fonctions écrites en C/C++ sous forme de MEX-files, ce qui est essentiel pour l'exécution performante des fonctions mathématiques de test.

3.3 Bibliothèques et outils utilisés

- MEX (MATLAB Executable) : utilisé pour interfacer les fonctions mathématiques de testécrites en langage C++ avec MATLAB.
- GUIDE et App Designer (interface graphique MATLAB) : pour concevoir une interface utilisateur conviviale permettant de sélectionner les algorithmes, les fonctions de test, et de visualiser les résultats graphiques.
- MATLAB Plotting Tools : utilisés pour générer les graphiques de convergence, les boxplots des résultats finaux, et les comparaisons entre les algorithmes.

3.4 Matériel utilisé

Les expérimentations ont été réalisées sur un ordinateur ayant les caractéristiques suivantes

- Processeur : 11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz 3.00 GHz
- Mémoire vive : 20 Go RAM(19.8 Go utilisable)
- Système d'exploitation : Édition Windows 11 Professionnel Version 24H2 Build du système d'exploitation 26100.4351
- Type du système : Système d'exploitation 64 bits, processeur x64
- Logiciel principal : MATLAB R2023a

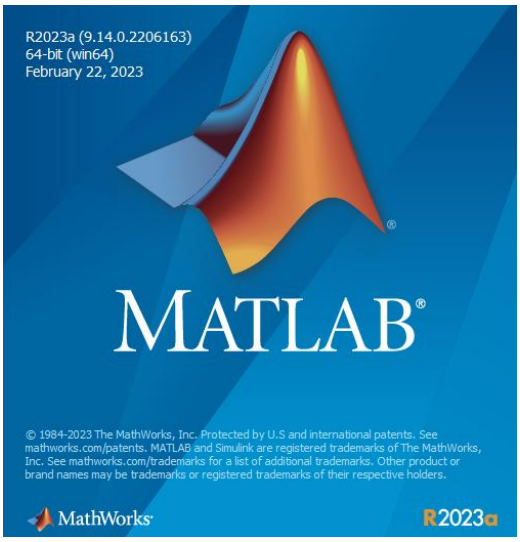


Figure 13: Version Matlab

3.5 Présentation du logiciel

Notre application se compose de deux fenêtres principales (voir figures), la première concerne le test de l'algorithme JAYA et ses variations EJAYA et CLJAYA sur les Benchmark CEC 2017 et CEC 2019, la deuxième concerne l'optimisation de regroupement des étudiants selon plusieurs critères en utilisant l'algorithme JAYA et ses variations EJAYA et CLJAYA.

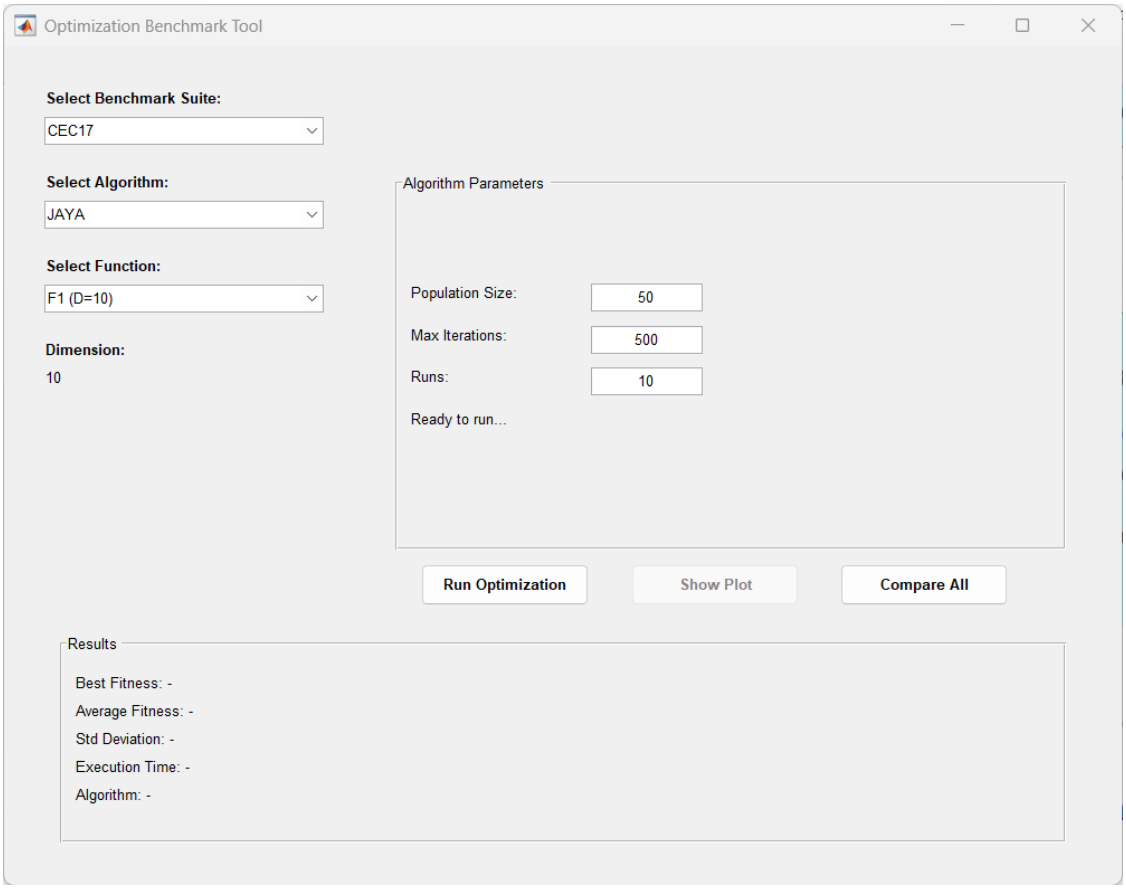


Figure 14: interface principale Optimisation Benchmark tool

3.6 Utilisation de l'interface *Optimization Benchmark Tool*

L'interface graphique de l'outil *Optimization Benchmark Tool* a été conçue de manière intuitive afin de permettre à l'utilisateur de tester et comparer facilement les algorithmes JAYA, EJAYA et CLJAYA sur différents benchmarks normalisés. Elle constitue un environnement pratique pour l'expérimentation des approches d'optimisation dans un cadre contrôlé et reproductible.

1. Lancement de l'application

Dès l'exécution du fichier OptimizationGUI.m, une fenêtre principale s'affiche au centre de l'écran. Cette fenêtre constitue le tableau de bord principal de l'outil.

2. Choix des paramètres d'exécution

L'utilisateur peut configurer plusieurs paramètres via l'interface graphique, notamment :

- Le benchmark à utiliser :
  Une liste déroulante permet de choisir entre CEC 2017 et CEC 2019.
- L'algorithme d'optimisation :
  L'utilisateur peut sélectionner l'un des trois algorithmes : JAYA, EJAYA, ou CLJAYA.
- La fonction de test (F1 à F30) :
  Permet de choisir la fonction du benchmark à optimiser.

- La dimension du problème :
  Représente le nombre de variables décisionnelles (par exemple : 10, 30 ou 50).
- La taille de la population :
  Nombre d'individus dans la population initiale (par exemple : 50 ou 100).
- Le nombre d'itérations :
  Définit la limite maximale de générations.
- Le nombre d'exécutions (*runs*) :
  Permet d'effectuer plusieurs essais pour obtenir des résultats statistiques plus robustes.

3. Lancement de l'optimisation

Après avoir configuré tous les paramètres, l'utilisateur clique sur le bouton « Run ». L'algorithme sélectionné est exécuté selon les spécifications choisies, et une zone de texte affiche la progression ou le temps d'exécution.

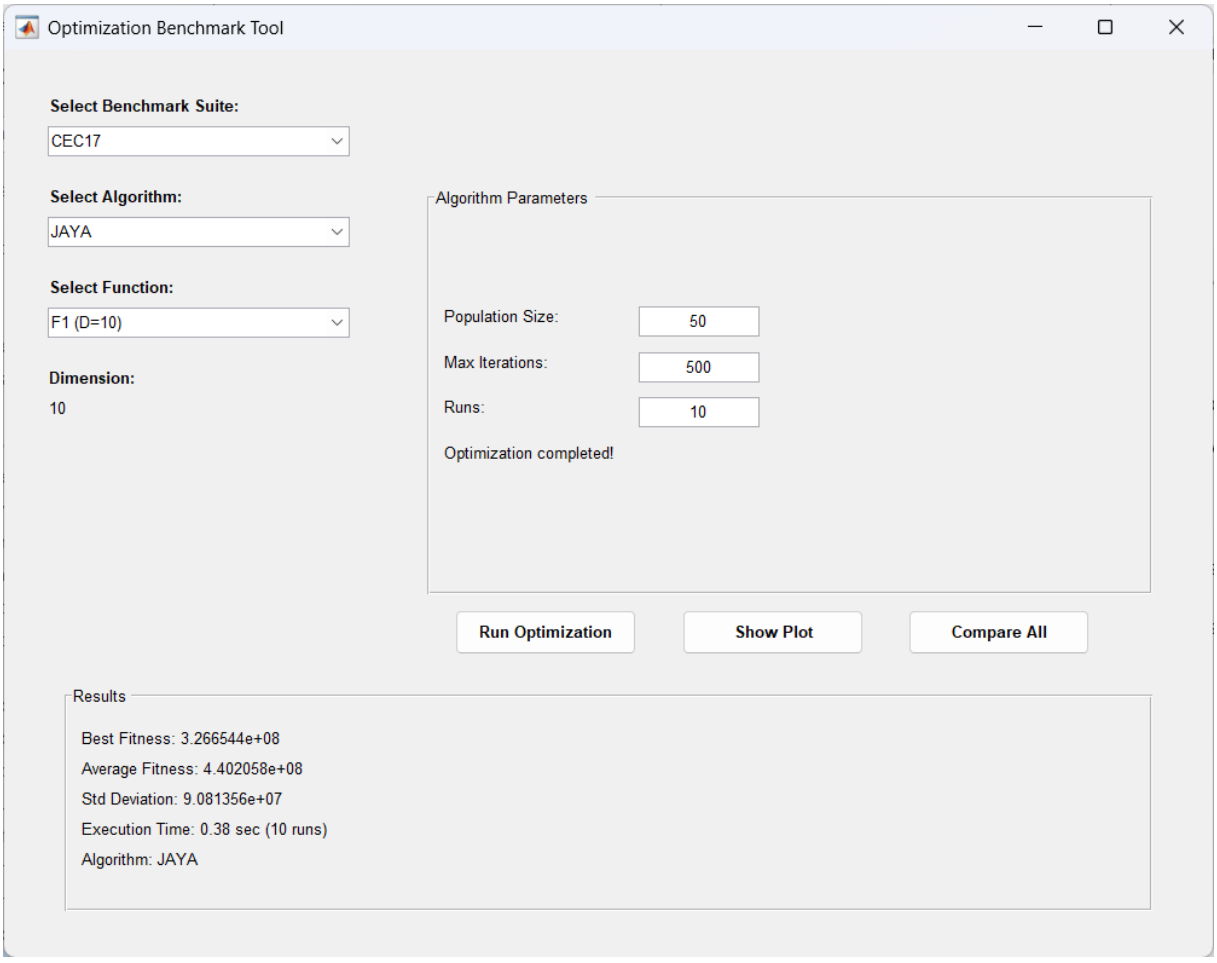


Figure 15: résultats après l'exécution

4. Affichage des résultats

À la fin de l'exécution, les résultats sont visualisés sous forme graphique et numérique :

- Courbe de convergence :
  Visualisation de l'évolution du meilleur coût au fil des itérations.

- Statistiques globales :

  - Meilleure solution atteinte,
  - Moyenne et écart-type sur les *runs*,
  - Temps total d'exécution,
  - Amélioration entre les premières et dernières itérations.

Un bouton « Show Plot » permet d'ouvrir une fenêtre dédiée à l'affichage de ces résultats.

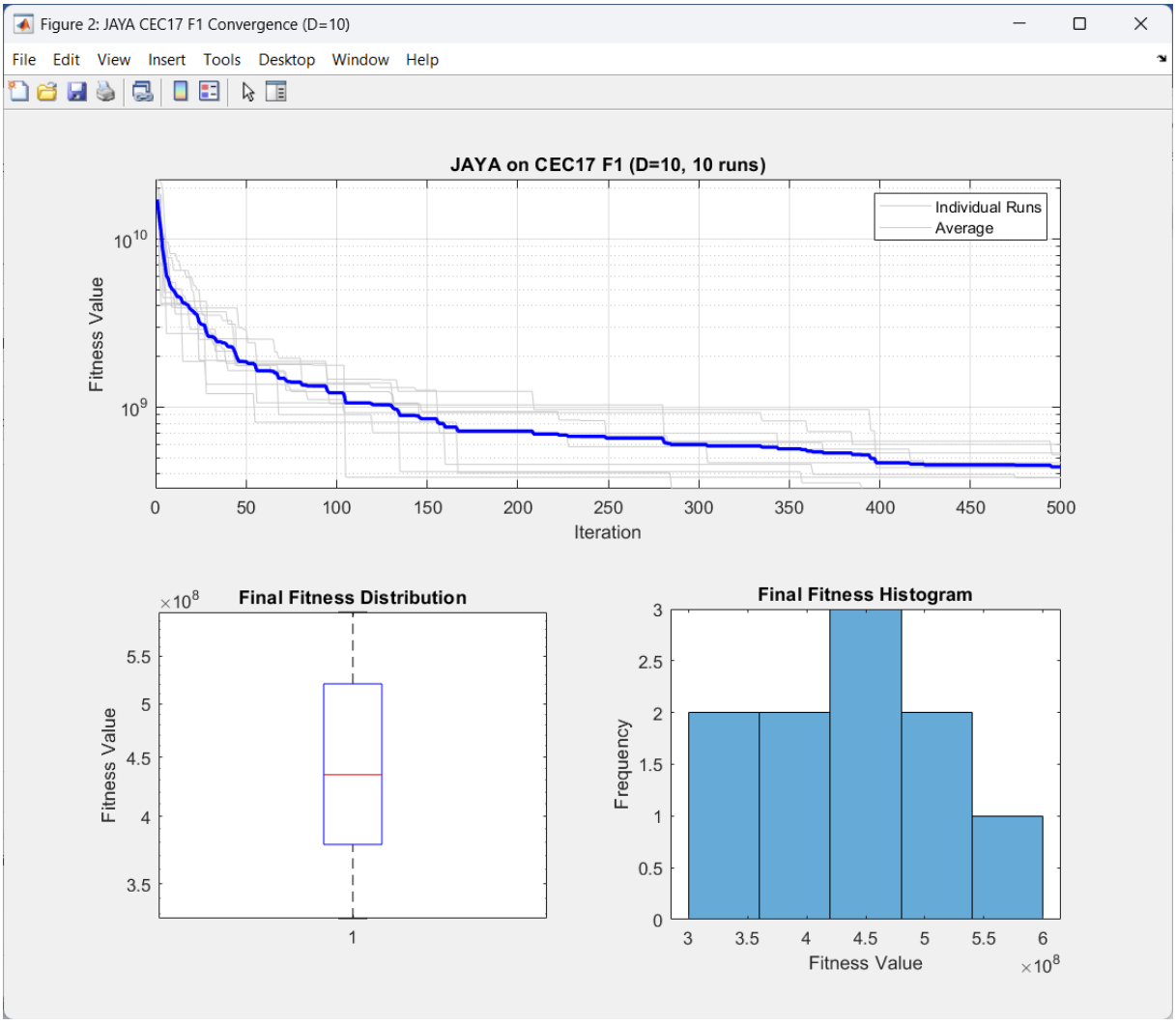


Figure 16: courbe de convergence

5. Comparaison entre algorithmes

L'interface propose également une fonctionnalité de comparaison :

- L'utilisateur peut sélectionner plusieurs algorithmes à évaluer,
- Cliquer sur le bouton « Compare all »,
- L'outil génère automatiquement :
  - Courbes de convergence moyennes,

- Diagrammes en boîte (*boxplots*) des résultats finaux,
- Tableau récapitulatif comparant les performances.

Cela permet de comparer objectivement la stabilité, la performance et la rapidité des différents algorithmes sur une même fonction.

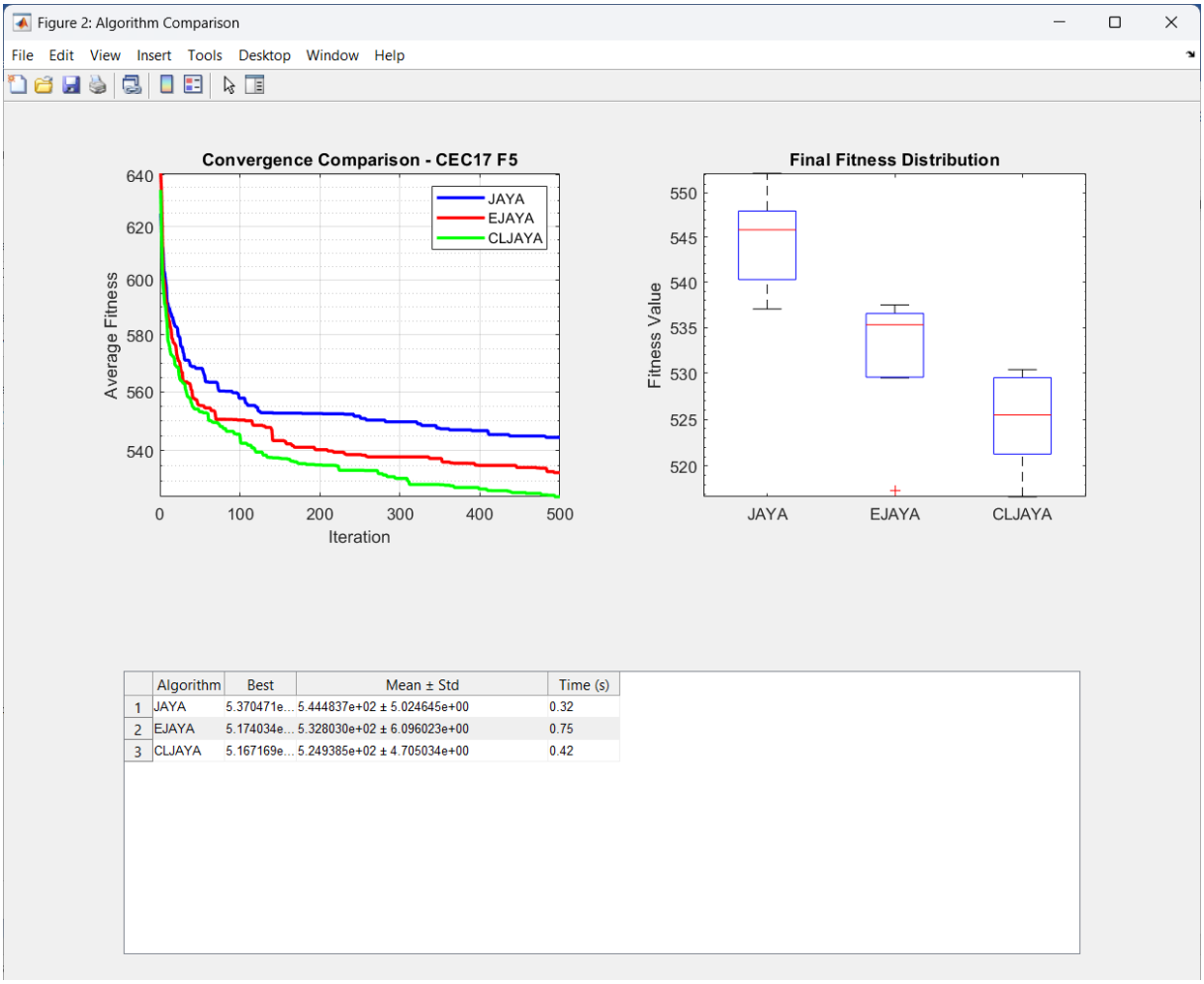


Figure 17: comparaison entre performance des algorithmes

3.7 Utilisation de l'interface *Student Grouping Tool*

L'interface graphique de l'outil *Student Grouping Tool* a été développée pour automatiser la répartition des élèves en groupes pédagogiques équilibrés, selon des critères définis tels que la moyenne générale, le sexe, et le nombre d'échecs. L'objectif est de garantir une homogénéité entre les groupes et d'éviter les déséquilibres pouvant affecter la dynamique de classe.

1. Lancement de l'application

L'utilisateur lance l'application via un fichier principal MATLAB (par exemple : StudentGroupingGUI.m). Une interface conviviale s'ouvre et propose une série d'actions.

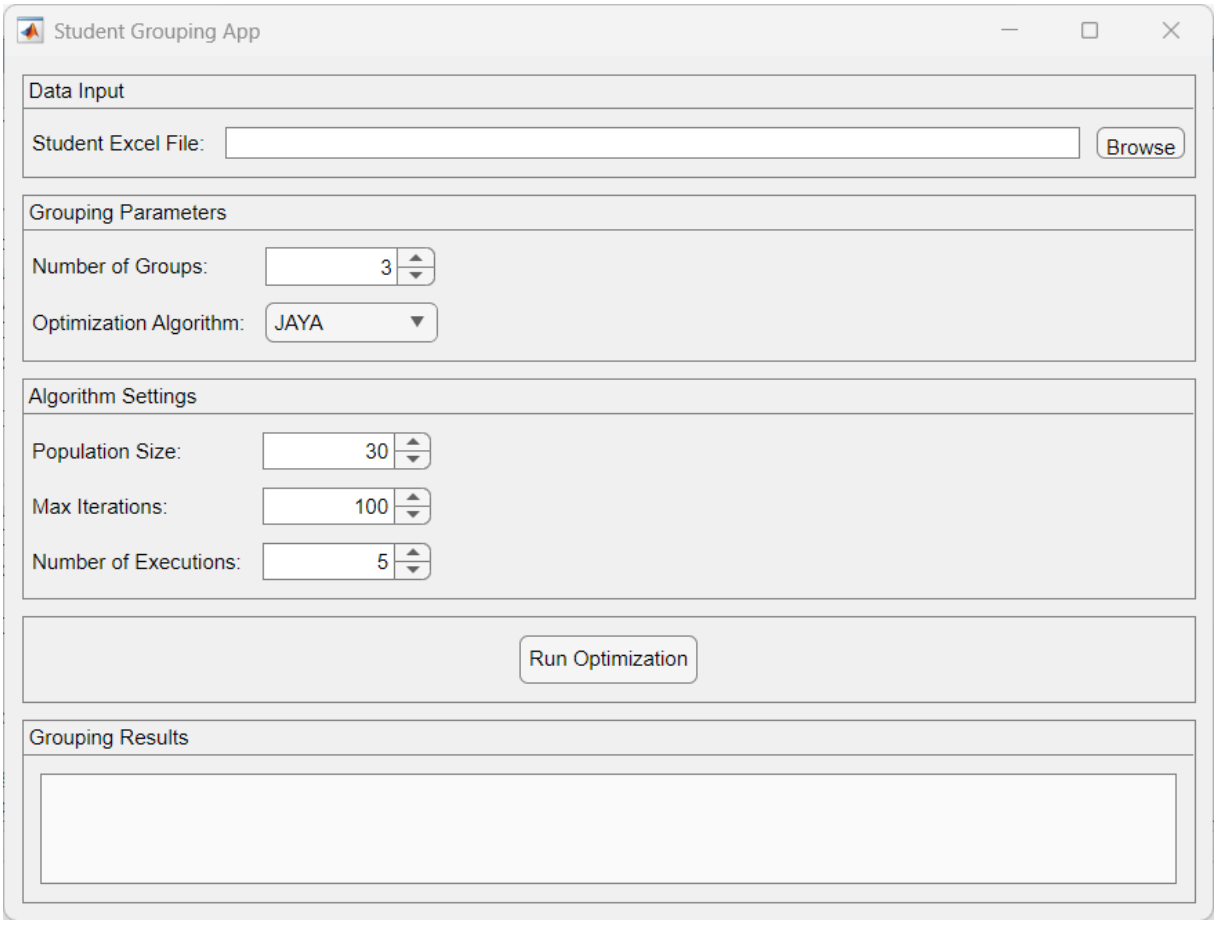


Figure 18: interface student grouping tool

2. Importation du fichier Excel

Une des premières étapes consiste à importer un fichier Excel contenant les données des élèves. Le fichier doit comporter les colonnes suivantes (ou équivalentes) :

- Nom et prénom de l'élève
- Sexe (Masculin / Féminin)
- Moyenne annuelle
- Nombre d'échecs
- (Optionnel) Classe, niveau, ou autres métadonnées

Un bouton « Importer » permet de sélectionner ce fichier depuis l'explorateur.

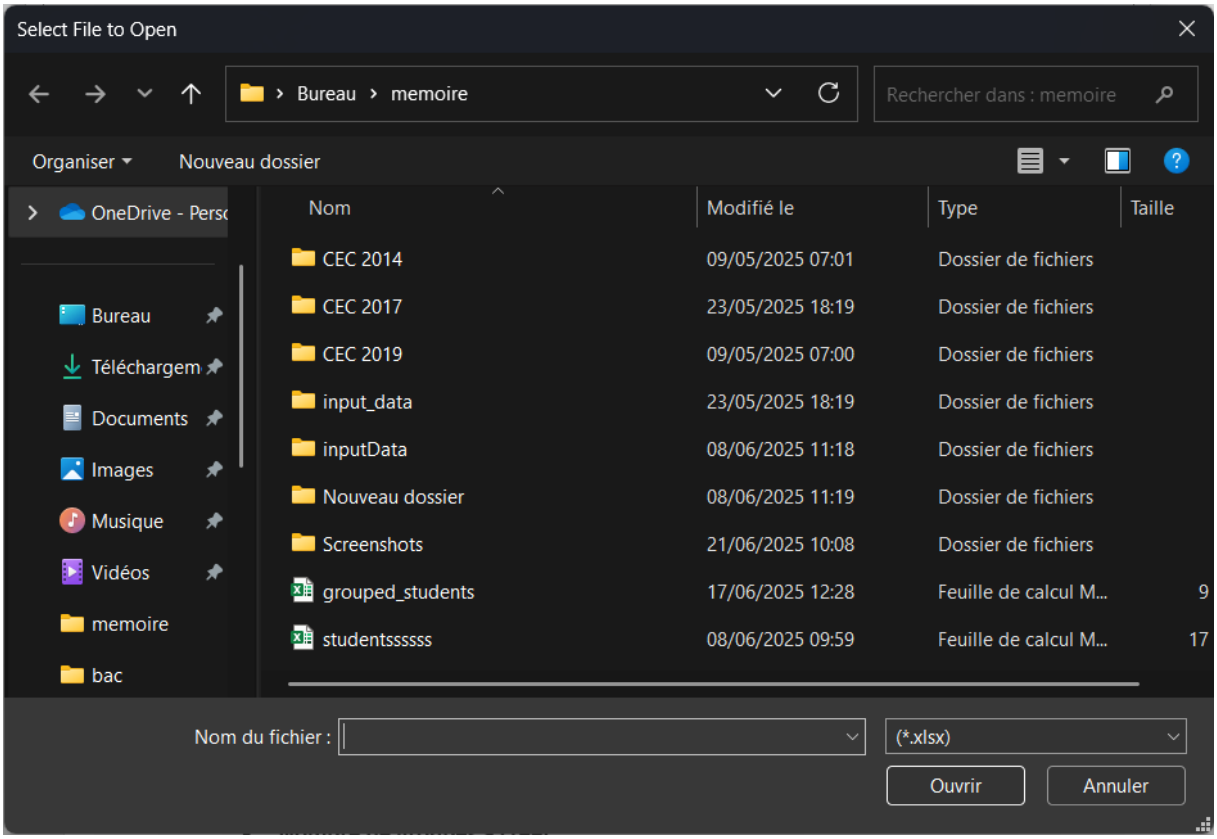


Figure 19: boite de dialogue pour importer fichier excel

3. Configuration des paramètres de regroupement

Une fois les données chargées, l'utilisateur peut spécifier :

- Nombre de groupes à créer
- Critères de répartition à équilibrer :
    - Moyennes : équilibration des performances académiques
    - Sexe : répartition équitable garçons/filles
    - Échecs : distribution homogène des élèves en difficulté

Ces paramètres sont souvent sélectionnés via des cases à cocher ou des listes déroulantes.

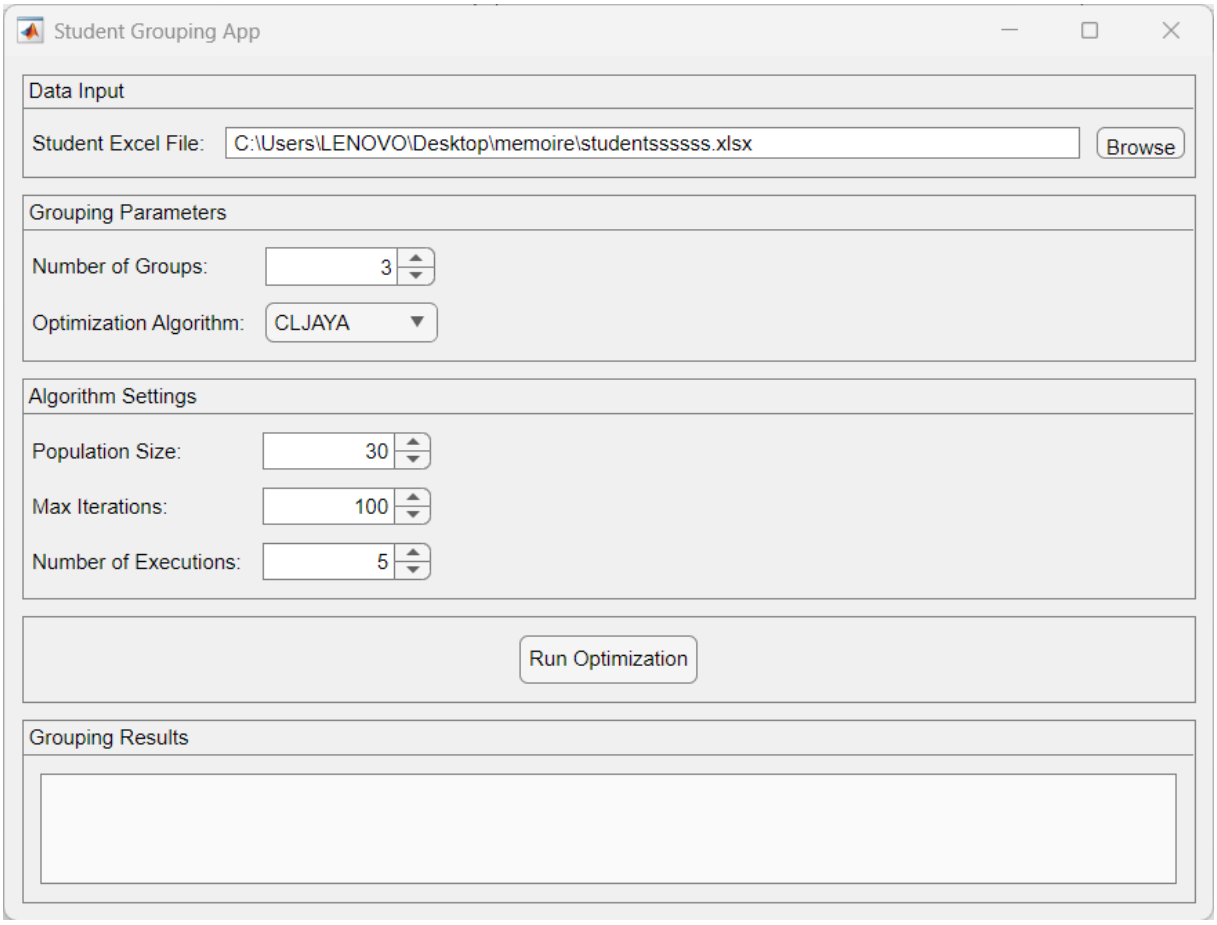


Figure 20: Configuration des paramètres de regroupement

4. Exécution de l'algorithme de regroupement

Après configuration, l'utilisateur clique sur « Générer les groupes ». Le système applique alors une métaheuristique (EJAYA, CLJAYA ou JAYA) adaptée aux problèmes discrets pour générer une solution optimale ou quasi-optimale en un temps raisonnable.

Pendant l'exécution, une barre de progression ou un message d'état s'affiche.

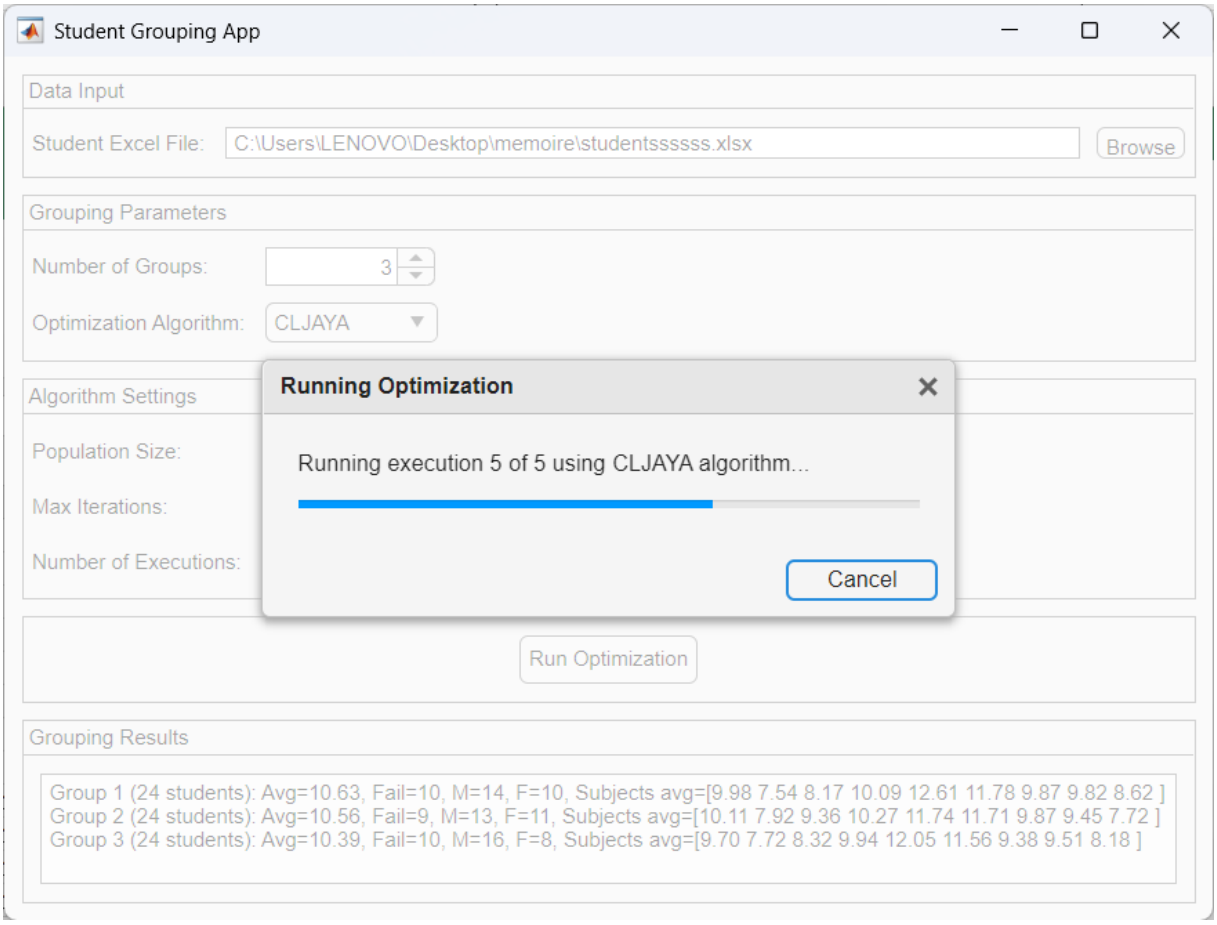


Figure 21: Exécution de l'algorithme de regroupement

5. Visualisation des groupes générés

Une fois l'exécution terminée, les groupes sont affichés :

- Dans des tableaux séparés ou sur une même interface, selon l'implémentation
- Chaque tableau affiche la liste des élèves, leur moyenne, leur sexe, et leur nombre d'échecs
- Des statistiques globales pour chaque groupe (moyenne générale, répartition par sexe) peuvent également être affichées

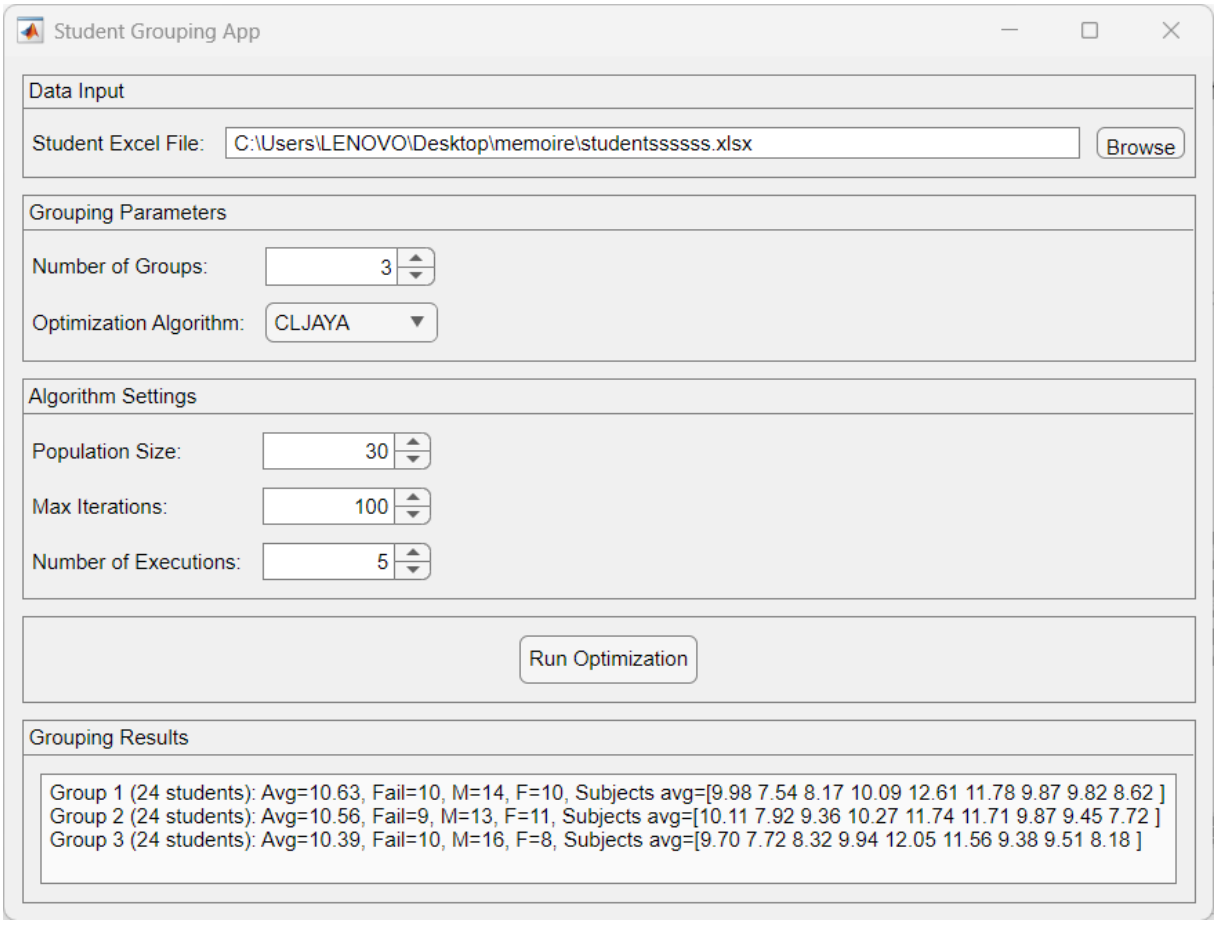
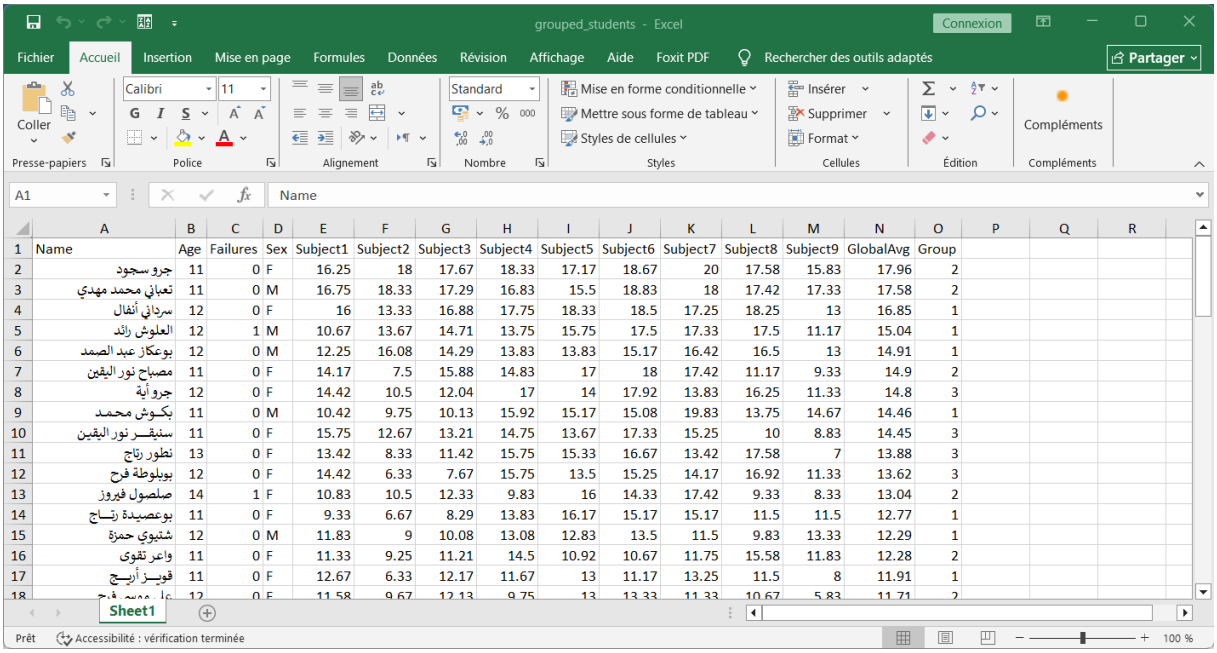


Figure 22: Visualisation des groupes générés

6. Exportation des résultats

L'utilisateur peut exporter les groupes formés sous plusieurs formats :

- Excel (.xlsx) : ajoutant une Colone groupe au fichier Excel.
- Texte (.txt) : Pour une analyse ou un archivage



grouped_students - Excel

	Name	Age	Failures	Sex	Subject1	Subject2	Subject3	Subject4	Subject5	Subject6	Subject7	Subject8	Subject9	GlobalAvg	Group
2	جرو سجود	11	0	F	16.25	18	17.67	18.33	17.17	18.67	20	17.58	15.83	17.96	2
3	تعباني محمد مهدي	11	0	M	16.75	18.33	17.29	16.83	15.5	18.83	18	17.42	17.33	17.58	2
4	سرداني أنفال	12	0	F	16	13.33	16.88	17.75	18.33	18.5	17.25	18.25	13	16.85	1
5	العلوش رائد	12	1	M	10.67	13.67	14.71	13.75	15.75	17.5	17.33	17.5	11.17	15.04	1
6	بوعكاز عبد الصمد	12	0	M	12.25	16.08	14.29	13.83	13.83	15.17	16.42	16.5	13	14.91	1
7	مصباح نور اليقين	11	0	F	14.17	7.5	15.88	14.83	17	18	17.42	11.17	9.33	14.9	2
8	جرو أية	12	0	F	14.42	10.5	12.04	17	14	17.92	13.83	16.25	11.33	14.8	3
9	بكـوش محمـد	11	0	M	10.42	9.75	10.13	15.92	15.17	15.08	19.83	13.75	14.67	14.46	1
10	سنيقـــر نور اليقين	11	0	F	15.75	12.67	13.21	14.75	13.67	17.33	15.25	10	8.83	14.45	3
11	نطور رتاج	13	0	F	13.42	8.33	11.42	15.75	15.33	16.67	13.42	17.58	7	13.88	3
12	بوبلوطة فرح	12	0	F	14.42	6.33	7.67	15.75	13.5	15.25	14.17	16.92	11.33	13.62	3
13	صلصول فيروز	14	1	F	10.83	10.5	12.33	9.83	16	14.33	17.42	9.33	8.33	13.04	2
14	بوعصيدة رتـــاج	11	0	F	9.33	6.67	8.29	13.83	16.17	15.17	15.17	11.5	11.5	12.77	1
15	شتيوي حمزة	12	0	M	11.83	9	10.08	13.08	12.83	13.5	11.5	9.83	13.33	12.29	1
16	واعر تقوى	11	0	F	11.33	9.25	11.21	14.5	10.92	10.67	11.75	15.58	11.83	12.28	2
17	قويـــز أريـــج	11	0	F	12.67	6.33	12.17	11.67	13	11.17	13.25	11.5	8	11.91	1

Figure 23: résultat de groupement

4. Conclusion

Dans ce chapitre, nous avons présenté la conception et l'implémentation de notre système, structuré autour de deux volets essentiels :

- L'application des algorithmes d'optimisation JAYA, EJAYA et CLJAYA sur des fonctions standards continues issues des benchmarks CEC 2017 et CEC 2019,
- Et leur adaptation à un problème discret : le regroupement automatique des élèves en groupes pédagogiques équilibrés selon plusieurs critères (moyenne, sexe, nombre d'échecs…).

Nous avons d'abord décrit l'environnement de développement utilisé, puis présenté l'architecture générale du système et ses modules fonctionnels à l'aide des diagrammes de flux de données (DFD). Cette modélisation nous a permis de mieux structurer et planifier la mise en œuvre du système.

Ensuite, nous avons détaillé l'utilisation des deux interfaces développées :

- Optimization Benchmark Tool : interface de test et comparaison des performances des algorithmes sur des fonctions de référence,
- Student Grouping Tool : interface dédiée au traitement d'un problème pédagogique concret à l'aide d'algorithmes métaheuristiques.

Ce chapitre a ainsi jeté les bases solides pour la phase suivante.

Le chapitre suivant, intitulé Résultats et Expérimentation, sera consacré à l'analyse des performances obtenues, à la comparaison entre les différentes variantes d'algorithmes, ainsi qu'à l'évaluation de leur efficacité dans le contexte du regroupement des élèves.

Chapitre 04 : résultats et discussion

1. Introduction

Ce chapitre est dédié à la présentation et à l'analyse des résultats expérimentaux obtenus suite à l'application des algorithmes JAYA, EJAYA et CLJAYA sur deux types de problèmes distincts :

- Le premier volet concerne les problèmes continus standards issus des benchmarks CEC 2017 et CEC 2019. Ces fonctions permettent de tester la robustesse, la rapidité de convergence, et la capacité des algorithmes à échapper aux optima locaux dans des espaces de recherche complexes.
- Le second volet traite un problème discret réel, à savoir le regroupement automatique d'élèves selon plusieurs critères pédagogiques. L'objectif est de former des groupes équilibrés en prenant en compte la moyenne générale, le sexe, et le nombre d'échecs.

Les résultats seront présentés sous forme de graphes de convergence, statistiques descriptives (meilleur coût, moyenne, écart-type, temps d'exécution), courbes comparatives ainsi que des analyses visuelles des regroupements obtenus.

Nous procéderons ensuite à une discussion critique des performances observées, en soulignant les points forts et les limites de chaque algorithme selon le contexte d'application. Cette phase est essentielle pour valider expérimentalement les adaptations proposées et pour identifier les pistes d'amélioration potentielles.

2. Jeux de données

Dans ce chapitre, nous présentons et analysons les résultats expérimentaux obtenus à partir de deux types de jeux de données, correspondant aux deux volets de notre étude : l'optimisation continue et l'optimisation discrète.

2.1 Évaluation des performances en optimisation continue

Pour évaluer les performances des algorithmes JAYA, EJAYA et CLJAYA dans un contexte d'optimisation continue, nous avons utilisé les fonctions tests standards issues des compétitions internationales CEC 2017 et CEC 2019, proposées par le IEEE Congress on Evolutionary Computation.

Ces benchmarks ne constituent pas des jeux de données, mais plutôt des suites de fonctions mathématiques bien établies, largement utilisées dans la communauté scientifique pour tester la robustesse et l'efficacité des algorithmes d'optimisation.

Ces fonctions sont réparties en plusieurs catégories selon leur structure et leur complexité :

- Fonctions unimodales : comportent un unique minimum global, idéales pour tester la capacité d'exploration rapide d'un algorithme.
- Fonctions multimodales : contiennent de nombreux optima locaux, testant la résistance de l'algorithme au piégeage.
- Fonctions hybrides : combinent différentes fonctions élémentaires pour simuler des paysages complexes et artificiels.
- Fonctions compositionnelles : assemblages non linéaires de plusieurs fonctions avec des transformations (rotation, translation, décalage, etc.).

Chaque fonction est définie sur un espace de recherche de dimension variable (souvent 10, 30 ou 50 dimensions), permettant de tester les algorithmes dans des situations de complexité croissante.
Pour chaque fonction testée, plusieurs exécutions indépendantes (runs) ont été réalisées, afin de permettre une analyse statistique approfondie des performances, incluant la moyenne, l'écart-type, et d'autres indicateurs de stabilité et de qualité de convergence.

2.2 Jeux de données pour l'optimisation discrète (regroupement d'élèves)

Dans la seconde partie de notre travail, nous avons appliqué les mêmes algorithmes métaheuristiques au problème de regroupement d'élèves, qui relève de la catégorie des problèmes d'optimisation combinatoire discrète. Ce problème consiste à diviser un ensemble d'élèves en plusieurs groupes équilibrés, tout en respectant un certain nombre de contraintes pédagogiques et structurelles.

Le jeu de données utilisé provient d'un tableau réel d'élèves du CEM Kaddous Ahmed situé à Tamalous, pour l'année scolaire 2024–2025. Ce tableau a été importé à partir d'un fichier Excel (.xlsx) et contient des informations détaillées sur 401 élèves, répartis sur plusieurs classes.

Chaque élève est décrit par les attributs suivants :

- Name : Nom et prénom de l'élève (identifiant principal),
- Age : Âge de l'élève,
- Failures : Nombre d'échecs (matières non validées),
- Sex : Sexe (M ou F),
- ARAB : Moyenne en langue arabe,
- Français : Moyenne en langue française,
- Anglais : Moyenne en langue anglaise,
- S.Islamique : Moyenne en éducation islamique,
- Civ : Moyenne en éducation civique,
- Histoire Geographie : Moyenne en histoire et géographie,
- Mathématique : Moyenne en mathématiques,
- Sciences : Moyenne en sciences naturelles,
- physique : Moyenne en sciences physiques et technologie,
- moyenne : Moyenne générale annuelle,
- Group : Groupe de répartition (initialement vide, assigné par l'algorithme).

Ainsi, le tableau est constitué de 14 variables (colonnes) représentant les différentes dimensions pédagogiques et démographiques pertinentes à la répartition, en plus d'une quinzième colonne (Group) créée à l'issue de l'optimisation. Le nombre total d'individus est de 401.

L'objectif de l'optimisation est de former un certain nombre de groupes équilibrés, en minimisant un coût global de regroupement, calculé en fonction des différences intergroupes sur les variables suivantes :

- La répartition du sexe (égalité hommes/femmes),
- La moyenne générale des élèves,
- Les moyennes par matière,
- Le nombre d'échecs,
- Et la taille de chaque groupe, en s'assurant que les groupes sont de taille proche (écart maximal d'un élève).

Ce jeu de données nous a permis de simuler un processus de répartition automatique en groupes pédagogiques homogènes, en tenant compte à la fois de critères d'équité et de performance, tels qu'ils pourraient être utilisés dans une gestion réelle au sein d'un établissement éducatif.

2.3 Suites de fonctions tests CEC 2017

Le benchmark CEC 2017 (Congress on Evolutionary Computation) est un ensemble de30 fonctions de test conçu pour évaluer la performance des algorithmes d'optimisation dans un environnement standardisé. Ces fonctions sont divisées en plusieurs catégories selon leur complexité et leurs caractéristiques topologiques.

2.4 Catégories des fonctions CEC 2017

Catégorie	Fonctions associées	Description
Unimodales	F1 à F3	Fonctions avec un seul minimum global, utiles pour tester la capacité d'exploration rapide.
Multimodales	F4 à F10	Fonctions avec de nombreux minima locaux, testent la robustesse contre le piégeage local.
Fonctions hybrides	F11 à F20	Combinent plusieurs types de fonctions pour simuler des environnements complexes.
Fonctions composées	F21 à F30	Assemblage complexe de fonctions avec transformations (rotation, translation, etc.).

Figure 24: Résumé des jeux de données CEC 2017

2.5 Suites de fonctions tests CEC 2019

Le benchmark CEC 2019 constitue une évolution des jeux de données utilisés pour évaluer les algorithmes d'optimisation continue. Il s'inscrit dans la lignée des benchmarks proposés par le IEEE Congress on Evolutionary Computation et vise à offrir un ensemble de fonctions plus difficiles et plus réalistes que celles des éditions précédentes.

2.6 Catégories des fonctions CEC 2019

Le benchmark **CEC 2019** contient **10 fonctions de test** aux caractéristiques variées, conçues pour évaluer les performances des algorithmes d'optimisation dans des contextes complexes et réalistes.

Catégorie	**Fonctions associées**	**Description**
Fonctions multimodales	F1 à F10	Fonctions complexes présentant plusieurs optima locaux, utilisées pour tester la robustesse, la précision et la capacité d'exploration des algorithmes.

Différences avec CEC 2017

- Davantage de variabilité statistique introduite entre fonctions,
- Plus grande difficulté dans les fonctions hybrides et composées,
- Plus adaptées pour tester la généralisation des métaheuristiques.

3. Résultats expérimentaux et discussion

Afin d'évaluer de manière exhaustive les performances des algorithmes JAYA, EJAYA et CLJAYA dans un contexte d'optimisation continue, nous avons appliqué ces trois métaheuristiques sur l'ensemble complet des fonctions de test des benchmarks CEC 2017 et CEC 2019.

- Le benchmark CEC 2017 comporte 30 fonctions standardisées réparties en quatre catégories :
  - Fonctions unimodales (F1–F3),
  - Fonctions multimodales (F4–F10),
  - Fonctions hybrides (F11–F20),
  - Fonctions compositionnelles (F21–F30).
- Le benchmark CEC 2019, quant à lui, contient uniquement 10 fonctions multimodales de dimensions variables, sélectionnées pour évaluer la robustesse et la convergence des algorithmes dans des contextes d'optimisation difficile.

Ces fonctions sont largement reconnues par la communauté scientifique pour fournir des scénarios de test rigoureux, permettant de comparer équitablement les performances des méthodes d'optimisation.

Pour garantir une comparaison équitable, chaque algorithme a été exécuté 50 fois indépendamment sur chaque fonction, avec les paramètres suivants :

- Taille de la population : 50
- Nombre maximal d'itérations : 1000
- Dimension du problème : 10

Pour chaque fonction, nous avons collecté les mesures suivantes :

- Meilleure valeur trouvée (Best value),
- Moyenne et écart-type des solutions finales sur les 10 exécutions,
- Temps d'exécution moyen en secondes.

Les résultats complets sont résumés dans un tableau structuré selon le format suivant :

Benchmark	Fonction	Algorithme	Best Value	Moyenne ± Écart-type	Temps (s)
CEC 2017	F1	JAYA	1.366636e+08	2.499640e+08 ± 8.817692e+07	0.35
		EJAYA	5.572276e+03	1.024035e+04 ± 6.529896e+03	1.18
		CLJAYA	1.574904e+04	3.799483e+04 ± 2.726622e+04	0.61
	F3	JAYA	3.474112e+03	5.325439e+03 ± 1.514416e+03	0.32
		EJAYA	4.779459e+02	9.510795e+02 ± 4.374266e+02	1.22
		CLJAYA	3.119496e+02	4.018843e+02 ± 9.708613e+01	0.56
	F4	JAYA	4.087851e+02	4.092764e+02 ± 3.833297e-01	0.33
		EJAYA	4.025742e+02	4.035532e+02 ± 6.848867e-01	1.15
		CLJAYA	4.039698e+02	4.057091e+02 ± 1.158371e+00	0.58
	F5	JAYA	5.335528e+02	5.428634e+02 ± 5.657205e+00	0.53
		EJAYA	5.204355e+02	5.274508e+02 ± 5.630355e+00	1.37
		CLJAYA	5.030811e+02	5.127950e+02 ± 6.691892e+00	0.95
	F6	JAYA	6.063106e+02	6.091096e+02 ± 2.784280e+00	1.01
		EJAYA	6.000002e+02	6.000006e+02 ± 2.812015e-04	1.96
		CLJAYA	6.000142e+02	6.000403e+02 ± 2.052580e-02	1.28
	F7	JAYA	7.462542e+02	7.534545e+02 ± 3.844966e+00	0.58
		EJAYA	7.249790e+02	7.401361e+02 ± 6.476886e+00	1.41
		CLJAYA	7.133375e+02	7.180801e+02 ± 2.612945e+00	0.85
	F8	JAYA	8.330806e+02	8.414843e+02 ± 4.520132e+00	0.53
		EJAYA	8.262918e+02	8.320978e+02 ± 3.809412e+00	1.44
		CLJAYA	8.030227e+02	8.068835e+02 ± 2.587560e+00	0.77
	F9	JAYA	9.177975e+02	9.294420e+02 ± 1.141650e+01	0.60
		EJAYA	9.000000e+02	9.000000e+02 ± 3.877759e-07	1.46
		CLJAYA	9.000034e+02	9.001757e+02 ± 1.566891e-01	0.81
	F10	JAYA	1.731216e+03	2.168222e+03 ± 2.387453e+02	0.65
		EJAYA	1.486779e+03	1.894596e+03 ± 2.715679e+02	1.46
		CLJAYA	1.190727e+03	1.457809e+03 ± 2.738954e+02	0.88
	F11	JAYA	1.145086e+03	1.199398e+03 ± 4.517930e+01	0.44
		EJAYA	1.103367e+03	1.106588e+03 ± 2.364361e+00	1.31
		CLJAYA	1.103218e+03	1.105433e+03 ± 1.862817e+00	0.67
	F12	JAYA	6.712074e+05	4.473485e+06 ± 3.345936e+06	0.46
		EJAYA	1.427493e+04	2.054650e+05 ± 1.557348e+05	1.40
		CLJAYA	3.521339e+03	6.283262e+05 ± 1.882729e+06	0.81
	F13	JAYA	2.767910e+03	1.363640e+04 ± 1.351333e+04	0.46
		EJAYA	2.020951e+03	9.731392e+03 ± 6.892116e+03	1.31
		CLJAYA	1.514511e+03	6.964316e+03 ± 9.630984e+03	0.69

	F14	**JAYA**	1.477126e+03	1.559928e+03 ± 1.442987e+02	0.59
		EJAYA	1.473059e+03	1.511180e+03 ± 1.805247e+01	1.39
		CLJAYA	1.412565e+03	2.292804e+03 ± 1.092952e+03	0.83
	F15	**JAYA**	1.906089e+03	2.154607e+03 ± 3.156296e+02	0.42
		EJAYA	1.680290e+03	1.883956e+03 ± 2.021579e+02	1.31
		CLJAYA	1.505396e+03	1.707986e+03 ± 1.953765e+02	0.68
	F16	**JAYA**	1.683123e+03	1.713179e+03 ± 2.549923e+01	0.54
		EJAYA	1.620015e+03	1.637667e+03 ± 1.248655e+01	1.35
		CLJAYA	1.638532e+03	1.671859e+03 ± 4.414661e+01	0.74
	F17	**JAYA**	1.775870e+03	1.789559e+03 ± 1.533823e+01	0.96
		EJAYA	1.749211e+03	1.761715e+03 ± 8.348536e+00	1.96
		CLJAYA	1.704999e+03	1.756524e+03 ± 5.760133e+01	1.24
	F18	**JAYA**	1.781865e+04	7.394535e+04 ± 4.566730e+04	0.53
		EJAYA	6.544815e+03	2.085179e+04 ± 1.189838e+04	1.32
		CLJAYA	1.202746e+04	4.133829e+04 ± 1.321471e+04	0.72
	F19	**JAYA**	2.091595e+03	2.626816e+03 ± 4.491575e+02	3.35
		EJAYA	1.950927e+03	2.076484e+03 ± 1.076506e+02	4.41
		CLJAYA	1.902165e+03	6.636674e+03 ± 9.829516e+03	3.69
	F20	**JAYA**	2.057191e+03	2.070812e+03 ± 1.351453e+01	0.97
		EJAYA	2.029733e+03	2.036087e+03 ± 3.395658e+00	1.89
		CLJAYA	2.000686e+03	2.006511e+03 ± 8.590900e+00	1.22
	F21	**JAYA**	2.208358e+03	2.315216e+03 ± 5.104421e+01	0.94
		EJAYA	2.206436e+03	2.279307e+03 ± 6.094059e+01	1.78
		CLJAYA	2.308769e+03	2.314832e+03 ± 3.808707e+00	1.23
	F22	**JAYA**	2.236879e+03	2.306487e+03 ± 3.662424e+01	1.29
		EJAYA	2.280128e+03	2.302980e+03 ± 8.309018e+00	2.14
		CLJAYA	2.300992e+03	2.301805e+03 ± 4.878547e-01	1.59
	F23	**JAYA**	2.633586e+03	2.644039e+03 ± 5.590200e+00	1.38
		EJAYA	2.616212e+03	2.621949e+03 ± 3.769096e+00	2.31
		CLJAYA	2.615456e+03	2.623570e+03 ± 6.478540e+00	1.72
	F24	**JAYA**	2.541609e+03	2.749579e+03 ± 7.318686e+01	1.49
		EJAYA	2.745922e+03	2.756745e+03 ± 6.726178e+00	2.37
		CLJAYA	2.741855e+03	2.755572e+03 ± 9.369628e+00	1.69
	F25	**JAYA**	2.943279e+03	2.962291e+03 ± 1.115931e+01	1.14
		EJAYA	2.898152e+03	2.917289e+03 ± 2.365191e+01	1.98
		CLJAYA	2.898180e+03	2.923439e+03 ± 2.604497e+01	1.42
	F26	**JAYA**	3.009931e+03	3.025237e+03 ± 1.202435e+01	1.62

		EJAYA	2.900008e+03	2.962427e+03 ± 2.810999e+01	2.56
		CLJAYA	2.975745e+03	3.087021e+03 ± 2.987797e+02	1.91
	F27	**JAYA**	3.096236e+03	3.097547e+03 ± 1.161711e+00	1.67
		EJAYA	3.089472e+03	3.091147e+03 ± 1.861445e+00	2.63
		CLJAYA	3.090311e+03	3.097309e+03 ± 1.560323e+01	1.93
	F28	**JAYA**	3.251711e+03	3.413436e+03 ± 1.265741e+02	1.48
		EJAYA	3.187321e+03	3.231401e+03 ± 6.417842e+01	2.33
		CLJAYA	3.196603e+03	3.370660e+03 ± 7.053442e+01	1.76
	F29	**JAYA**	3.164464e+03	3.190991e+03 ± 2.771565e+01	1.39
		EJAYA	3.149150e+03	3.170025e+03 ± 1.025606e+01	2.30
		CLJAYA	3.142694e+03	3.163835e+03 ± 1.819762e+01	1.64
	F30	**JAYA**	1.517036e+04	5.174409e+05 ± 3.905488e+05	3.86
		EJAYA	1.158416e+04	2.565337e+05 ± 3.319654e+05	4.78
		CLJAYA	5.651301e+03	3.477528e+05 ± 3.913536e+05	4.31

Tableau 3: Résultat CEC 2017

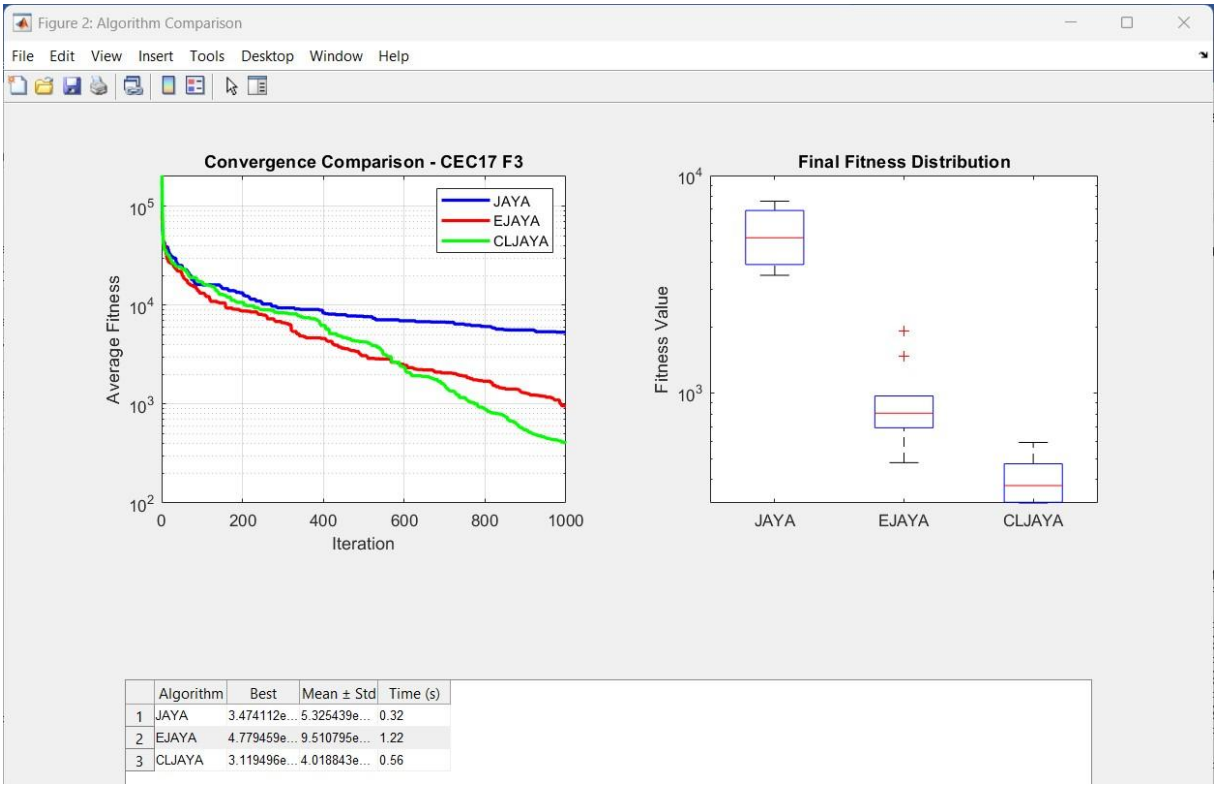
Benchmark	Fonction	Algorithme	Best Value	Moyenne ± Écart-type	Temps (s)
CEC 2019	F1	JAYA	2.748638e+02	1.225454e+03 ± 9.418427e+02	0.97
		EJAYA	2.767865e+01	1.378977e+02 ± 8.407971e+01	1.85
		CLJAYA	4.451011e+00	7.111255e+01 ± 6.912975e+01	1.20
	F2	JAYA	2.301214e+01	2.924152e+01 ± 5.593730e+00	0.38
		EJAYA	1.184545e+01	1.418219e+01 ± 1.968478e+00	1.26
		CLJAYA	5.621400e+00	7.924503e+00 ± 1.943488e+00	0.76
	F3	JAYA	1.127408e+01	1.201252e+01 ± 4.932575e-01	0.37
		EJAYA	1.162273e+01	1.226473e+01 ± 3.887928e-01	1.30
		CLJAYA	4.341607e+00	8.069577e+00 ± 1.610996e+00	0.78
	F4	JAYA	2.554885e+01	3.852805e+01 ± 7.587023e+00	0.49
		EJAYA	1.916423e+01	2.577627e+01 ± 4.673162e+00	1.33
		CLJAYA	5.857175e+00	1.343873e+01 ± 5.497665e+00	0.75
	F5	JAYA	4.532963e+00	5.038369e+00 ± 5.891946e-01	0.52
		EJAYA	1.184774e+00	1.342671e+00 ± 7.662605e-02	1.35
		CLJAYA	1.138226e+00	1.271378e+00 ± 1.059079e-01	0.79
	F6	JAYA	5.982199e+00	7.576174e+00 ± 8.429043e-01	14.35
		EJAYA	3.036616e+00	4.307366e+00 ± 1.048550e+00	15.21
		CLJAYA	1.305036e+00	2.632403e+00 ± 7.945432e-01	14.49
	F7	JAYA	1.001542e+03	1.339413e+03 ± 1.626442e+02	0.66
		EJAYA	3.325838e+02	1.152229e+03 ± 3.303408e+02	1.53
		CLJAYA	2.551982e+01	3.516739e+02 ± 2.081287e+02	1.01
	F8	JAYA	4.159470e+00	4.517950e+00 ± 1.552737e-01	0.53
		EJAYA	3.976106e+00	4.313603e+00 ± 1.869873e-01	1.40
		CLJAYA	2.663793e+00	3.155421e+00 ± 4.008379e-01	0.80
	F9	JAYA	1.201032e+00	1.361684e+00 ± 8.144373e-02	0.35
		EJAYA	1.124434e+00	1.207766e+00 ± 3.494376e-02	1.21
		CLJAYA	1.105950e+00	1.193213e+00 ± 4.041647e-02	0.61
	F10	JAYA	2.122661e+01	2.137853e+01 ± 1.076158e-01	0.56
		EJAYA	2.114483e+01	2.140562e+01 ± 9.660755e-02	1.46
		CLJAYA	2.102586e+01	2.107177e+01 ± 3.846134e-02	0.91

Tableau 4: Résultat CEC 2019

3.2 Visualisation des résultats (Convergence et Boxplot)

	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	1.366636e...	2.499640e...	0.35
2	EJAYA	5.572276e...	1.024035e...	1.18
3	CLJAYA	1.574904e...	3.799483e...	0.61

Figure 25: résultats F1 CEC 2017



	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	3.474112e...	5.325439e...	0.32
2	EJAYA	4.779459e...	9.510795e...	1.22
3	CLJAYA	3.119496e...	4.018843e...	0.56

Figure 26: : résultats F3 CEC 2017

Figure 27: résultats F4 CEC 2017

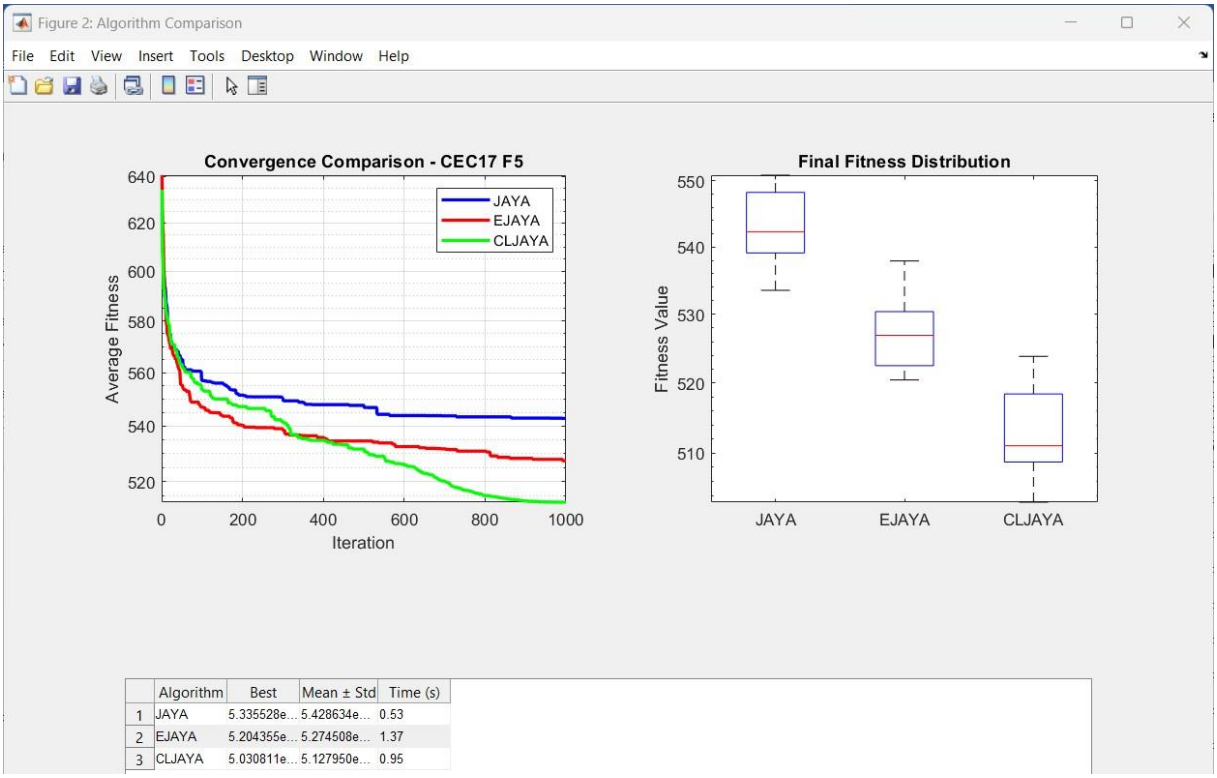
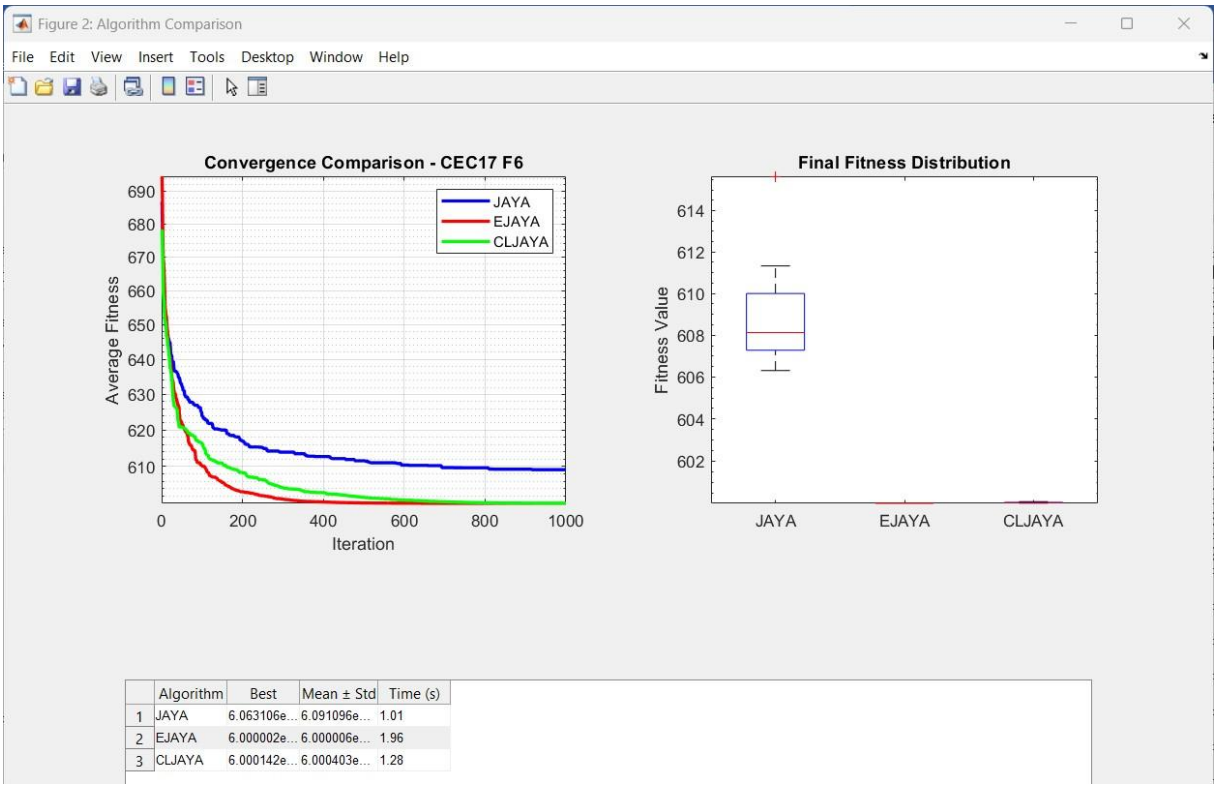
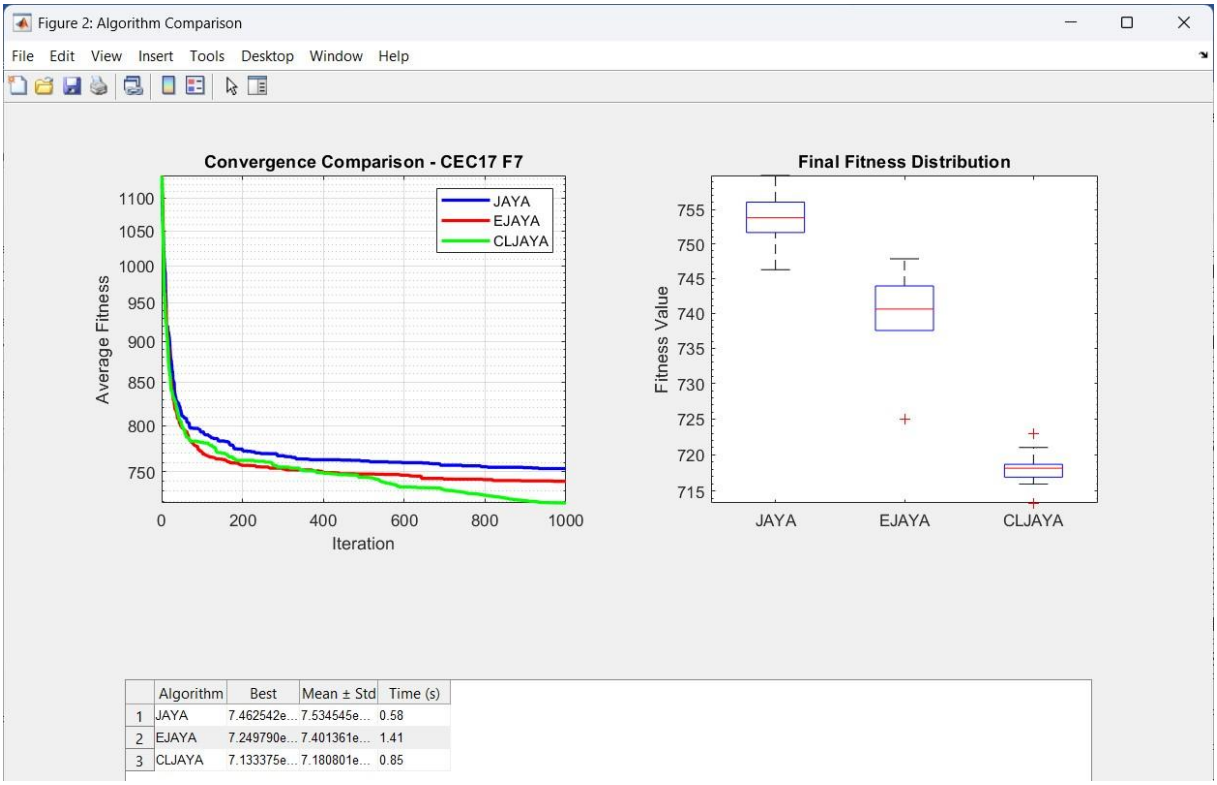


Figure 28: : résultats F5 CEC 2017



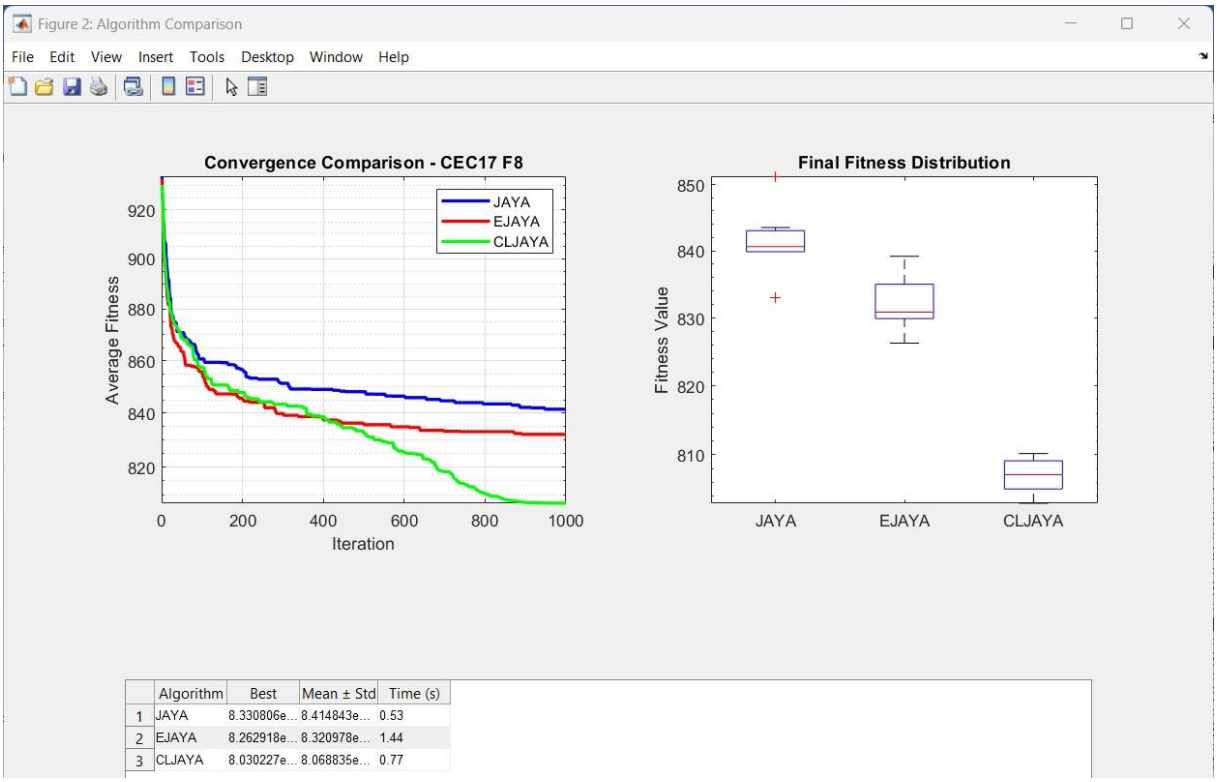
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	6.063106e...	6.091096e...	1.01
2	EJAYA	6.000002e...	6.000006e...	1.96
3	CLJAYA	6.000142e...	6.000403e...	1.28

Figure 29: : résultats F6 CEC 2017



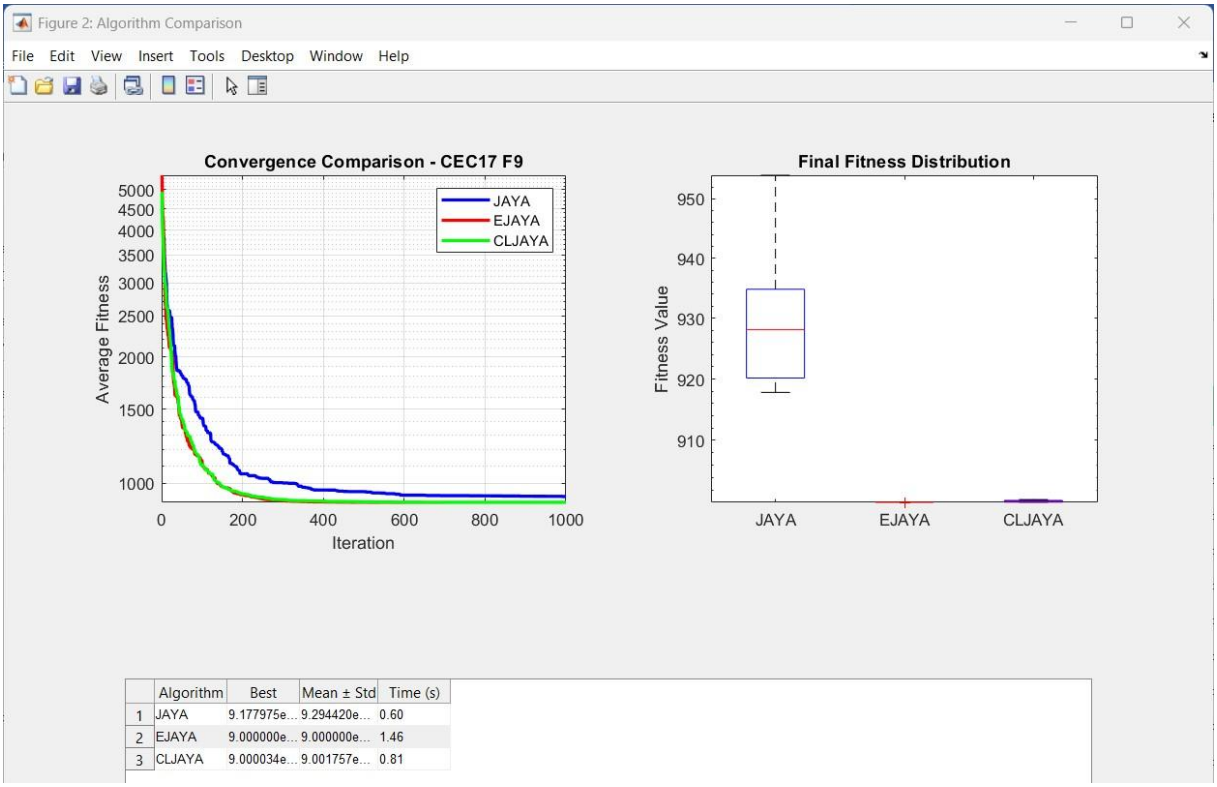
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	7.462542e...	7.534545e...	0.58
2	EJAYA	7.249790e...	7.401361e...	1.41
3	CLJAYA	7.133375e...	7.180801e...	0.85

Figure 30: résultats F7 CEC 2017



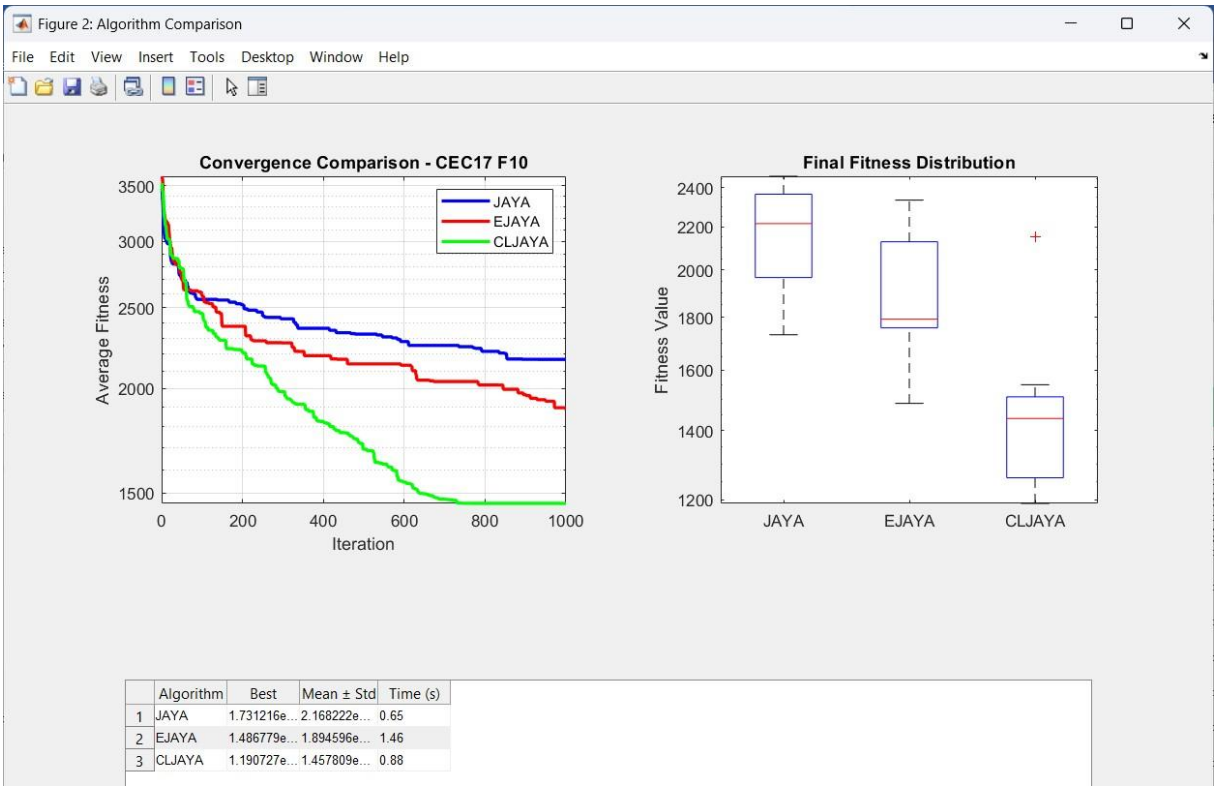
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	8.330806e...	8.414843e...	0.53
2	EJAYA	8.262918e...	8.320978e...	1.44
3	CLJAYA	8.030227e...	8.068835e...	0.77

Figure 31: résultats F8 CEC 2017



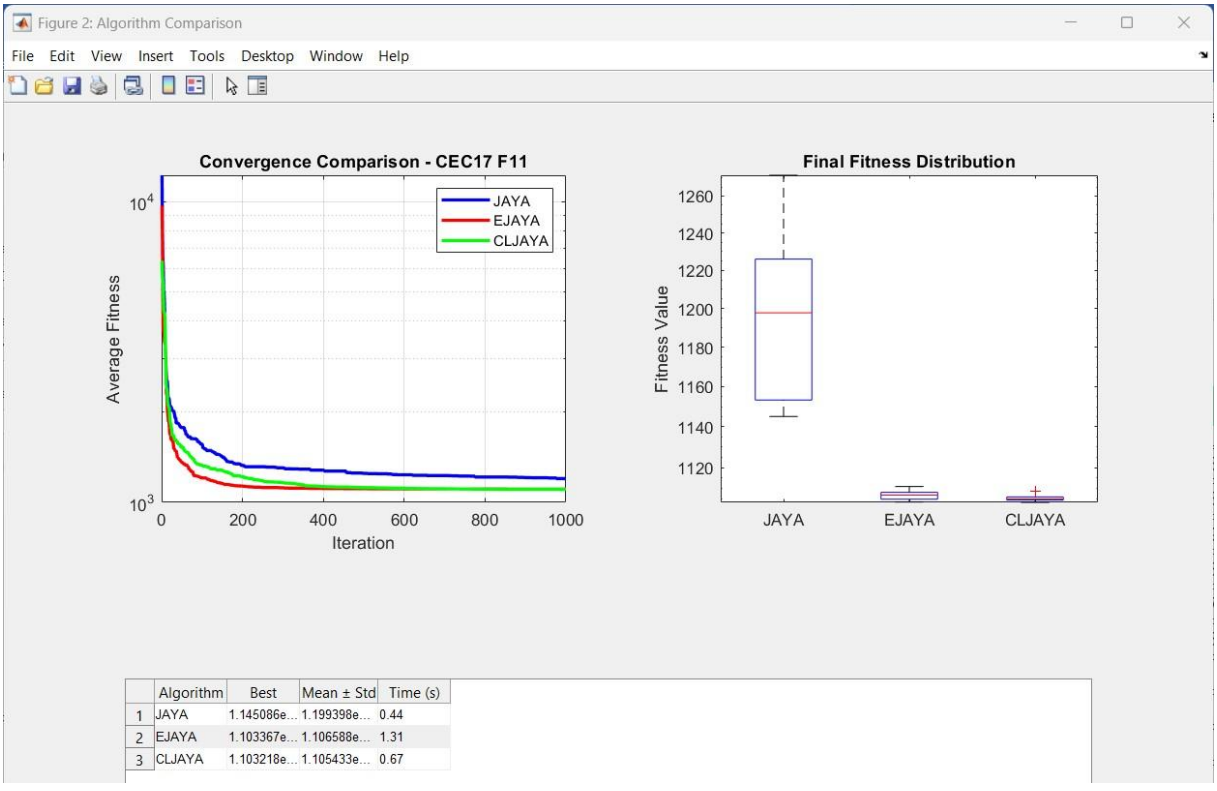
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	9.177975e...	9.294420e...	0.60
2	EJAYA	9.000000e...	9.000000e...	1.46
3	CLJAYA	9.000034e...	9.001757e...	0.81

Figure 32: résultats F9 CEC 2017



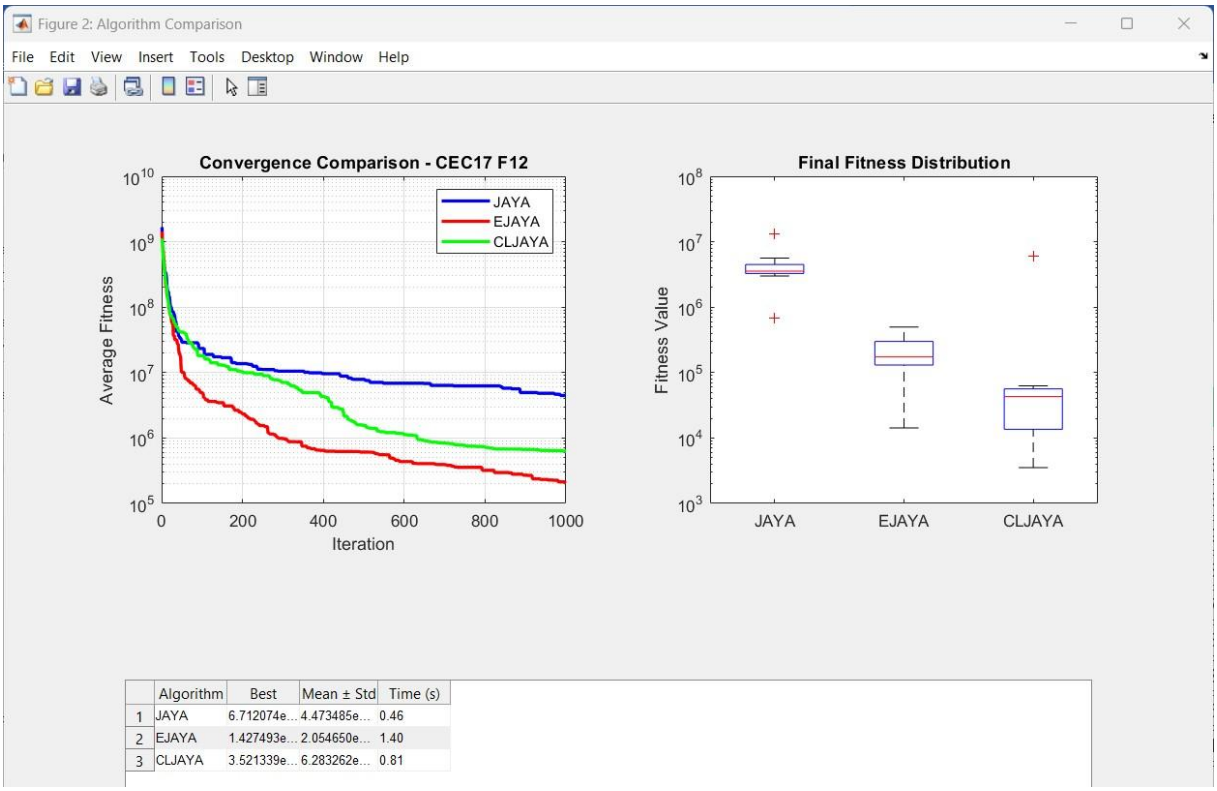
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	1.731216e...	2.168222e...	0.65
2	EJAYA	1.486779e...	1.894596e...	1.46
3	CLJAYA	1.190727e...	1.457809e...	0.88

Figure 33: résultats F10 CEC 2017



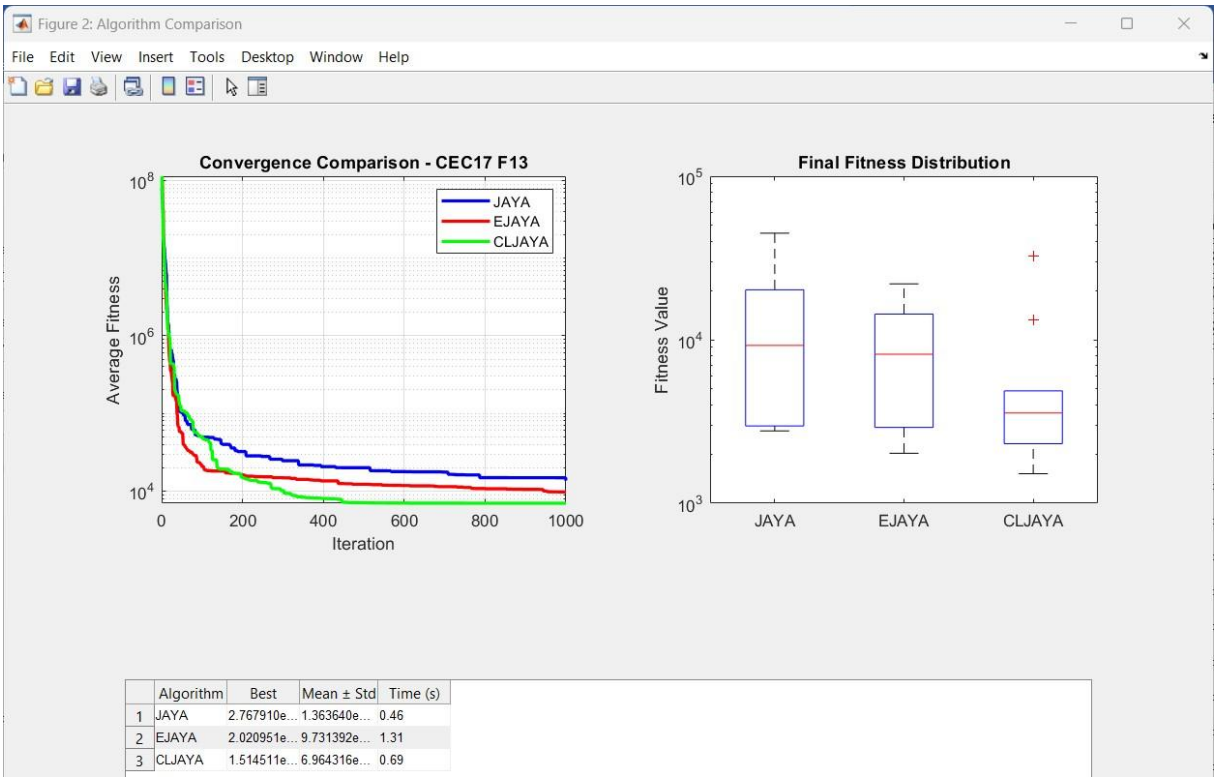
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	1.145086e...	1.199398e...	0.44
2	EJAYA	1.103367e...	1.106588e...	1.31
3	CLJAYA	1.103218e...	1.105433e...	0.67

Figure 34: résultats F11 CEC 2017



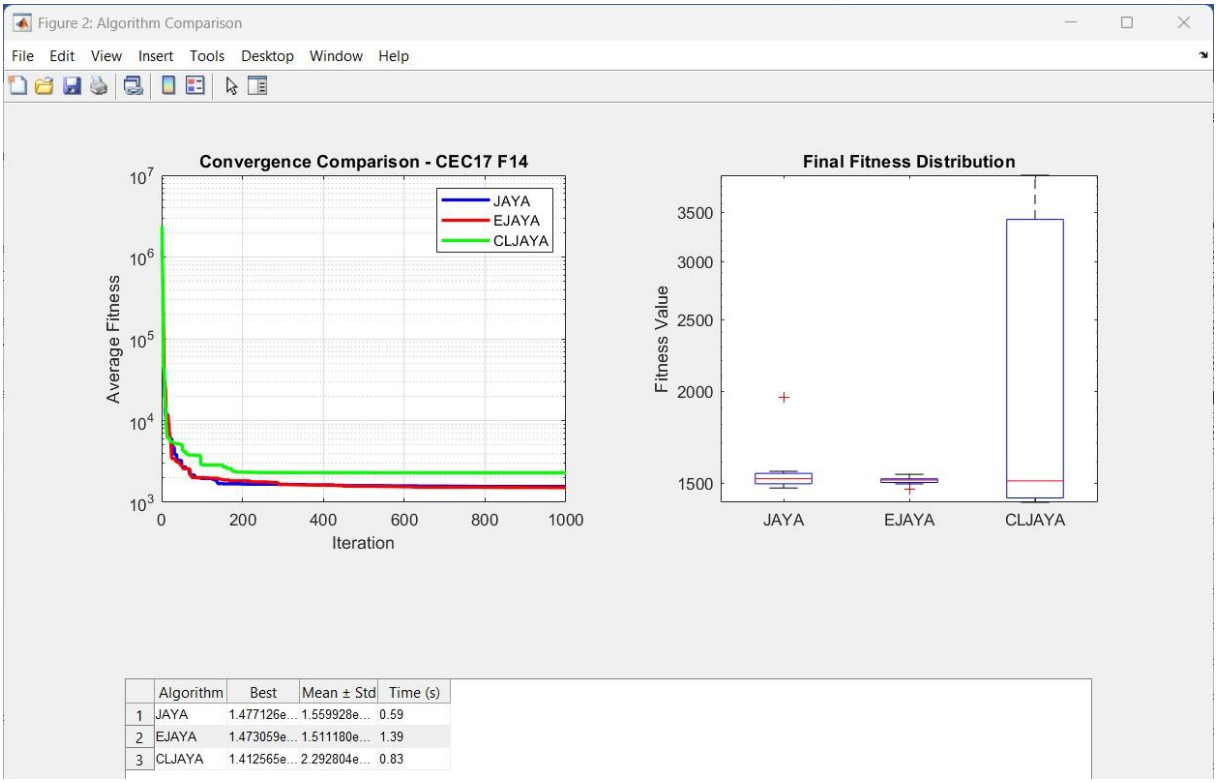
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	6.712074e...	4.473485e...	0.46
2	EJAYA	1.427493e...	2.054650e...	1.40
3	CLJAYA	3.521339e...	6.283262e...	0.81

Figure 35: résultats F12 CEC 2017



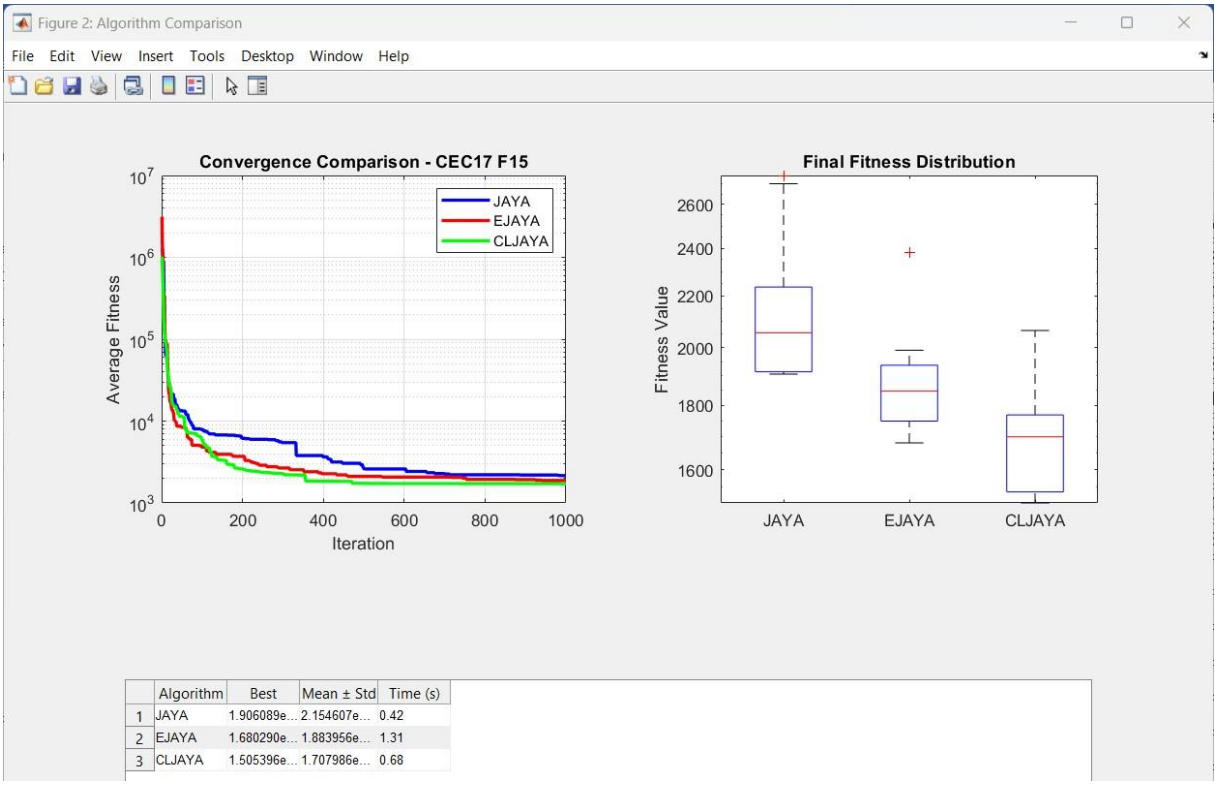
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	2.767910e...	1.363640e...	0.46
2	EJAYA	2.020951e...	9.731392e...	1.31
3	CLJAYA	1.514511e...	6.964316e...	0.69

Figure 36: résultats F13 CEC 2017



	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	1.477126e...	1.559928e...	0.59
2	EJAYA	1.473059e...	1.511180e...	1.39
3	CLJAYA	1.412565e...	2.292804e...	0.83

Figure 37: résultats F14 CEC 2017



	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	1.906089e...	2.154607e...	0.42
2	EJAYA	1.680290e...	1.883956e...	1.31
3	CLJAYA	1.505396e...	1.707986e...	0.68

Figure 38: résultats F15 CEC 2017

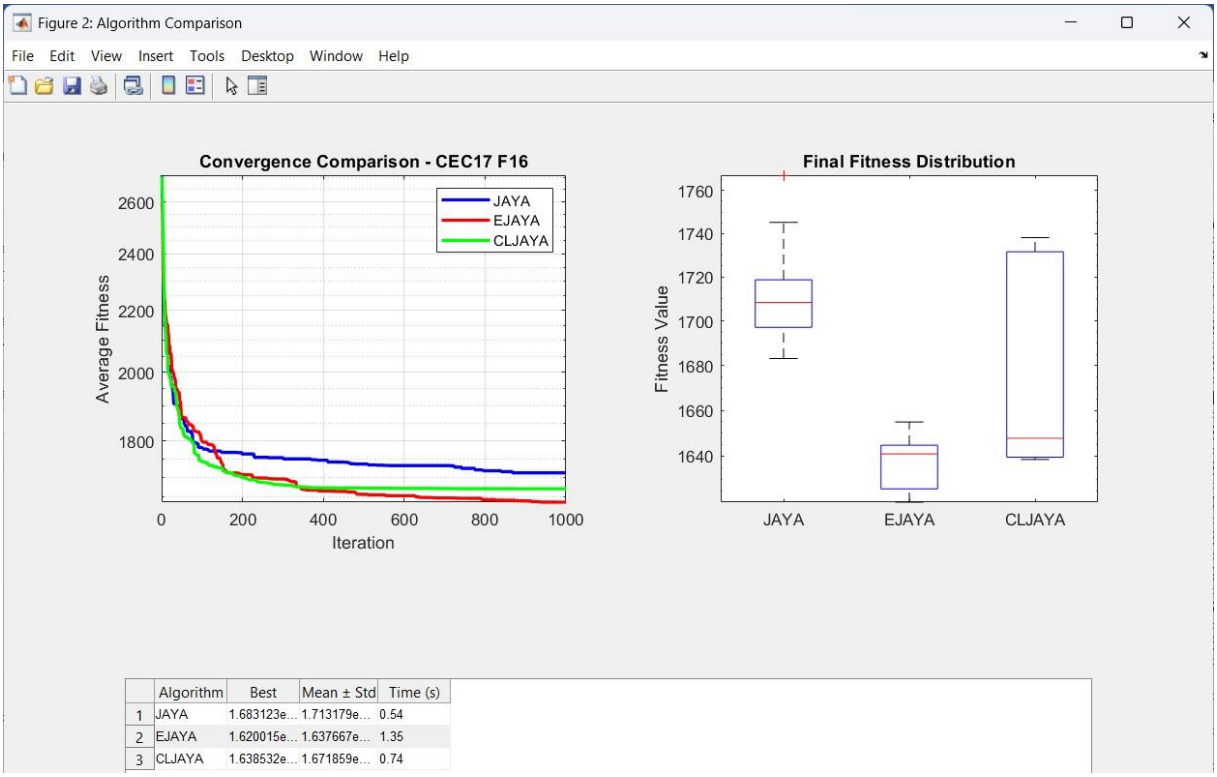


Figure 39: résultats F16 CEC 2017

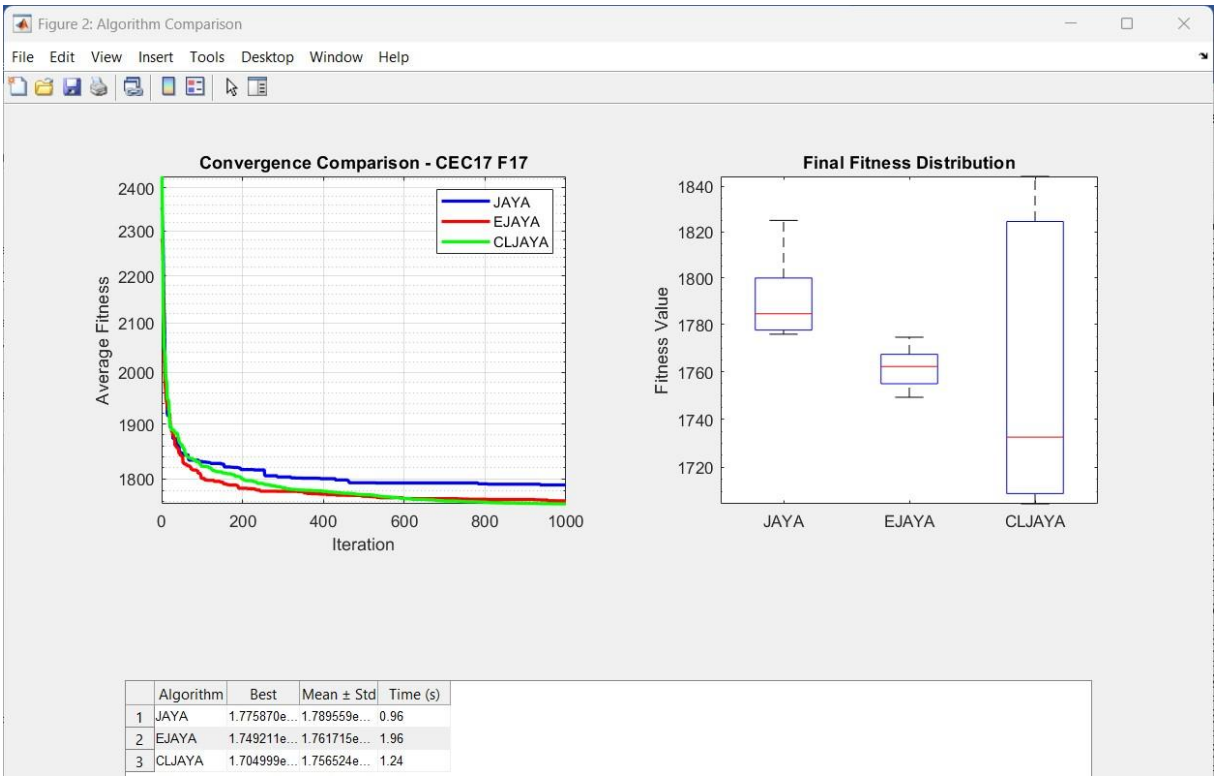
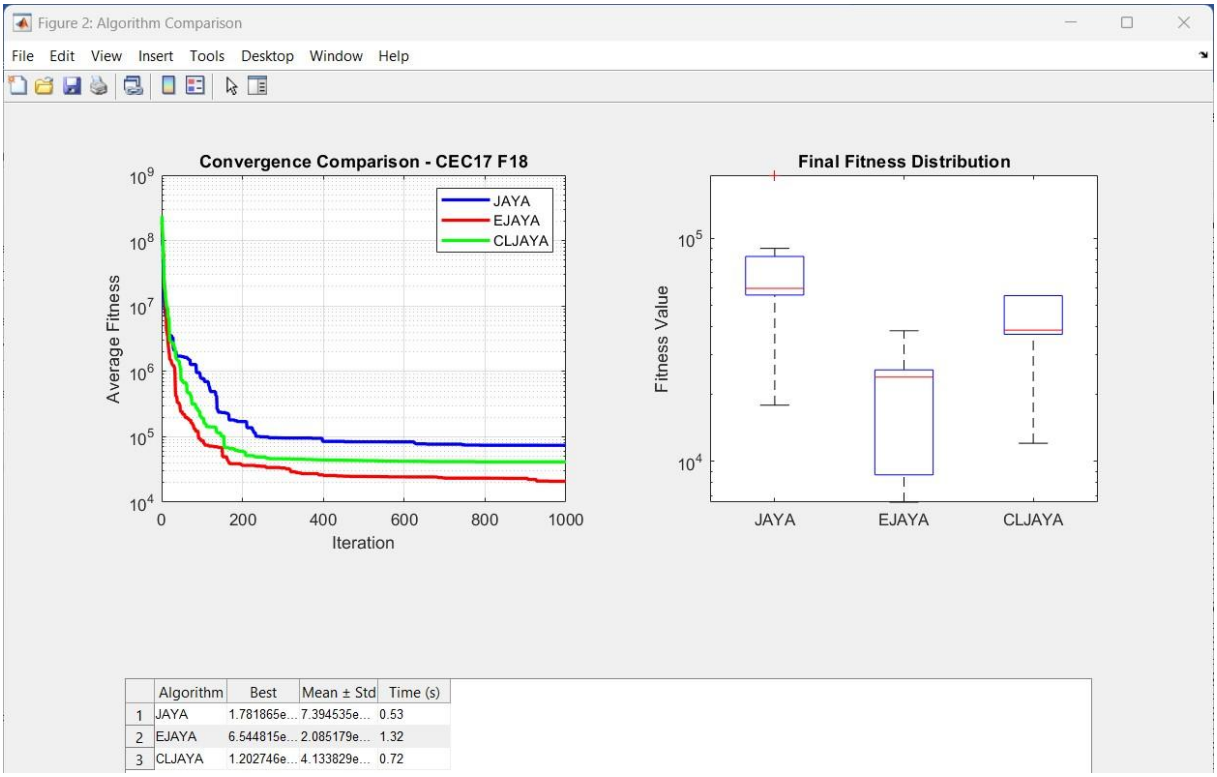
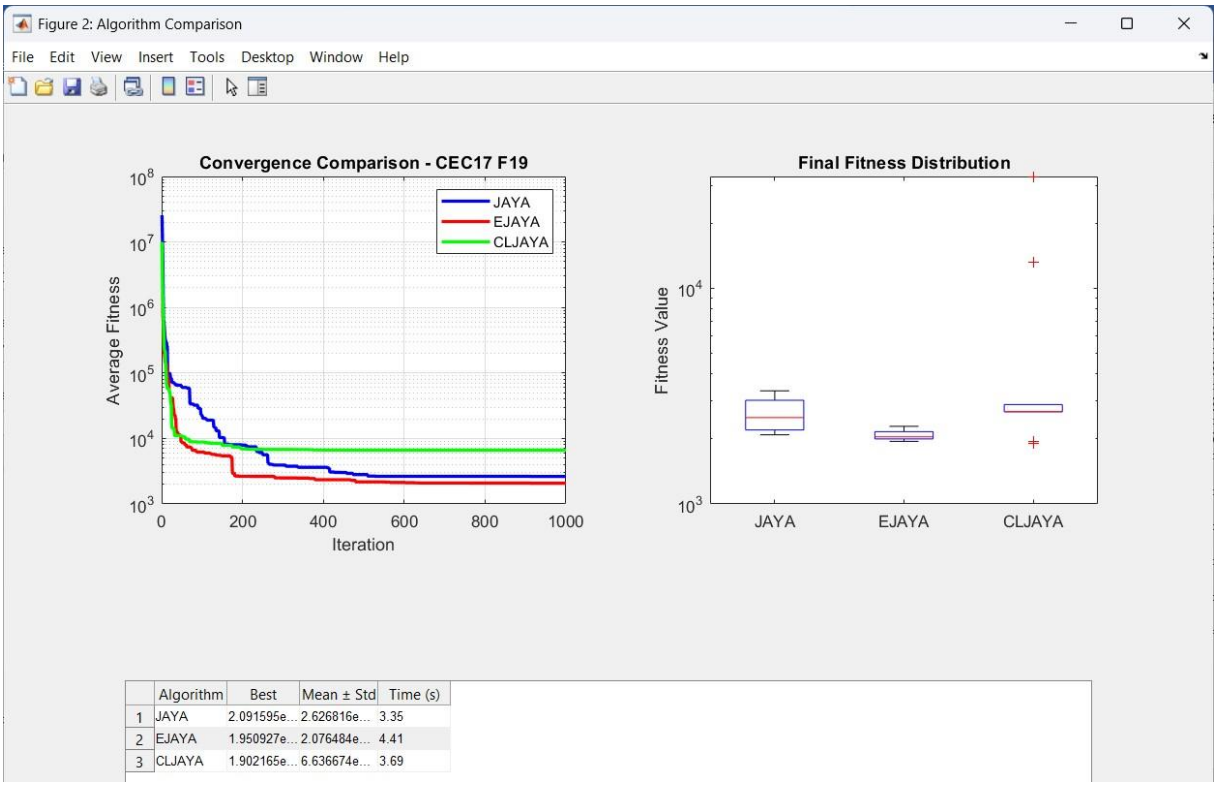


Figure 40: résultats F17 CEC 2017



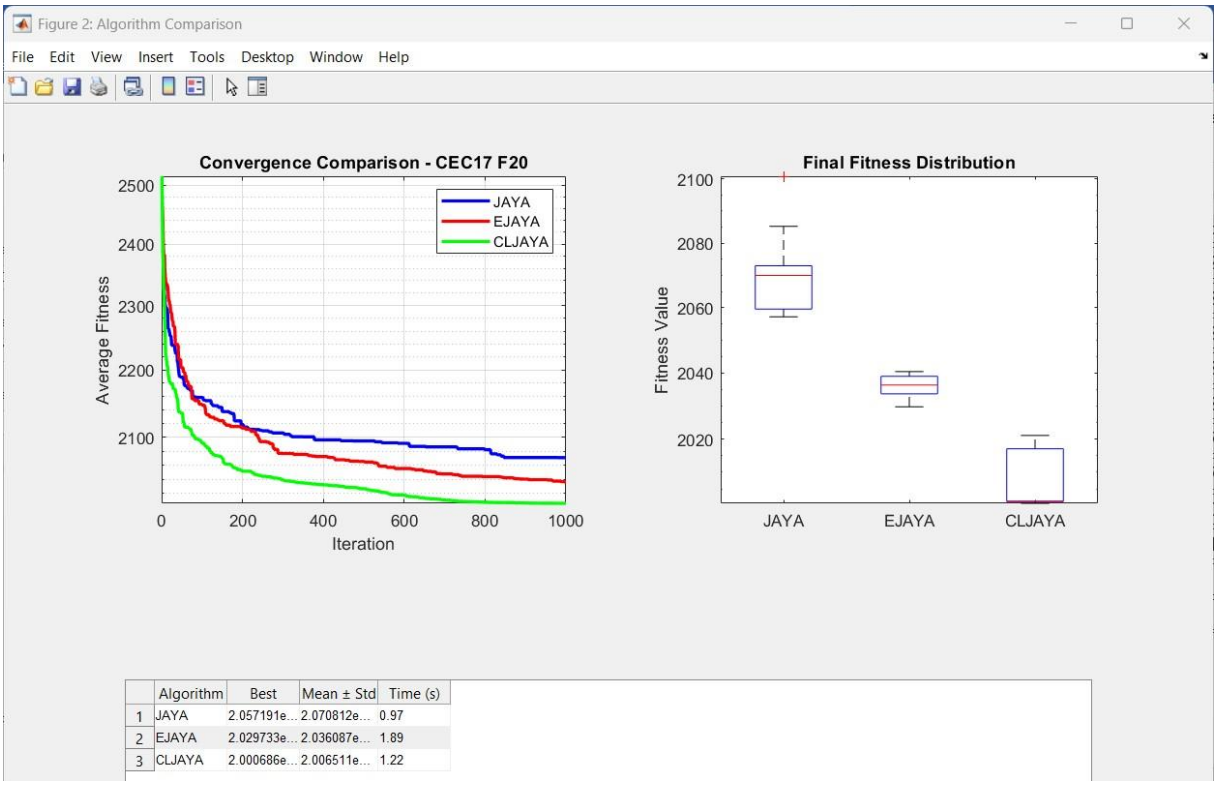
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	1.781865e...	7.394535e...	0.53
2	EJAYA	6.544815e...	2.085179e...	1.32
3	CLJAYA	1.202746e...	4.133829e...	0.72

Figure 41: résultats F18 CEC 2017



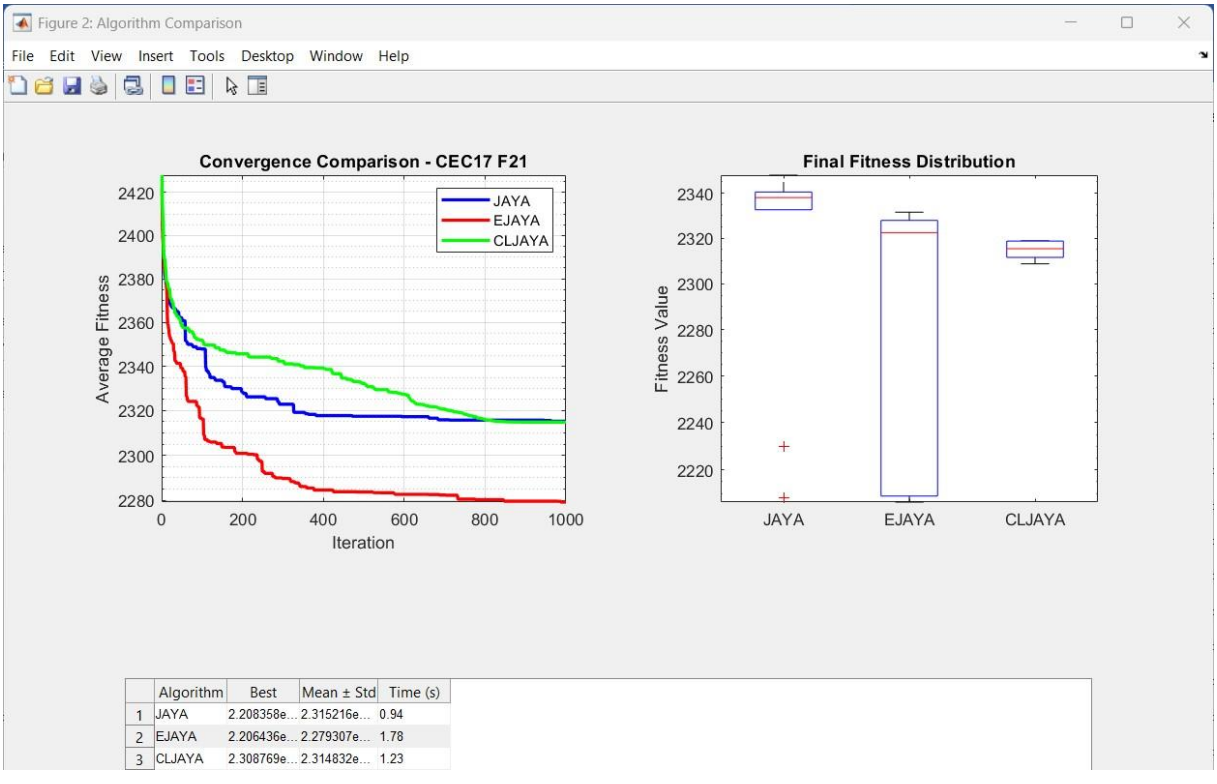
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	2.091595e...	2.626816e...	3.35
2	EJAYA	1.950927e...	2.076484e...	4.41
3	CLJAYA	1.902165e...	6.636674e...	3.69

Figure 42: résultats F19 CEC 2017



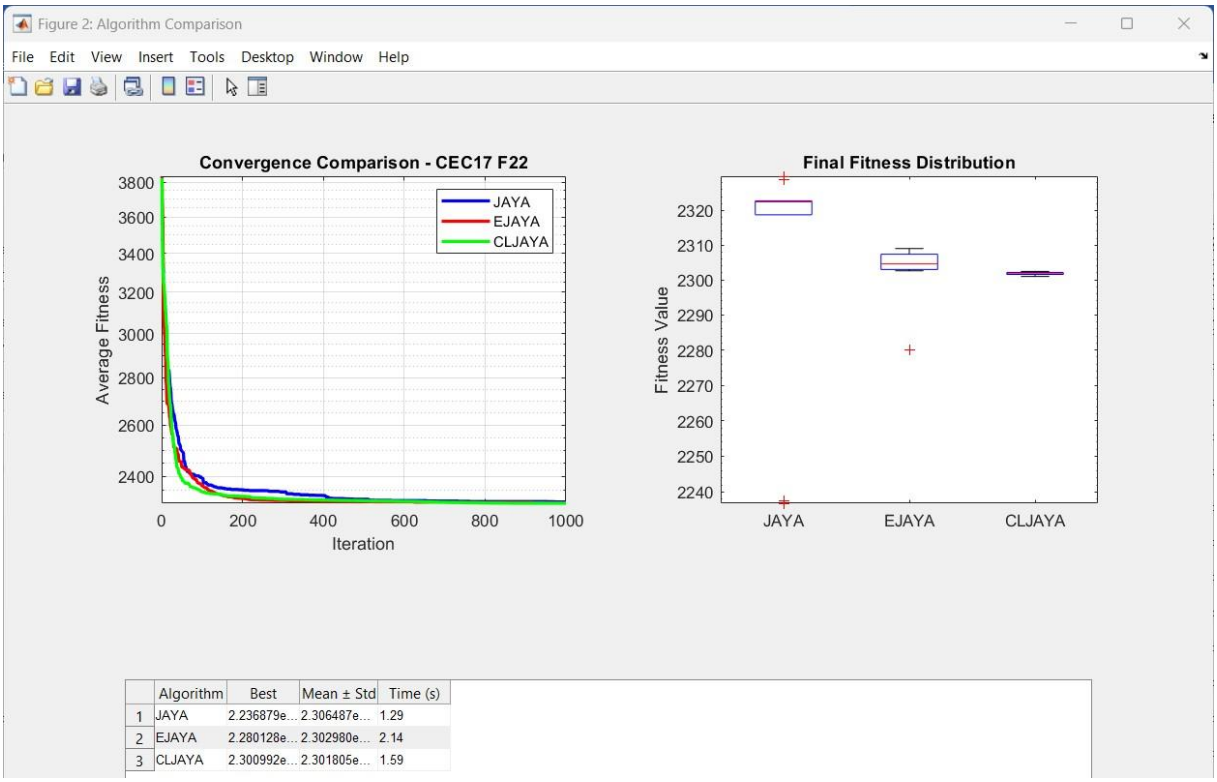
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	2.057191e...	2.070812e...	0.97
2	EJAYA	2.029733e...	2.036087e...	1.89
3	CLJAYA	2.000686e...	2.006511e...	1.22

Figure 43: résultats F20 CEC 2017



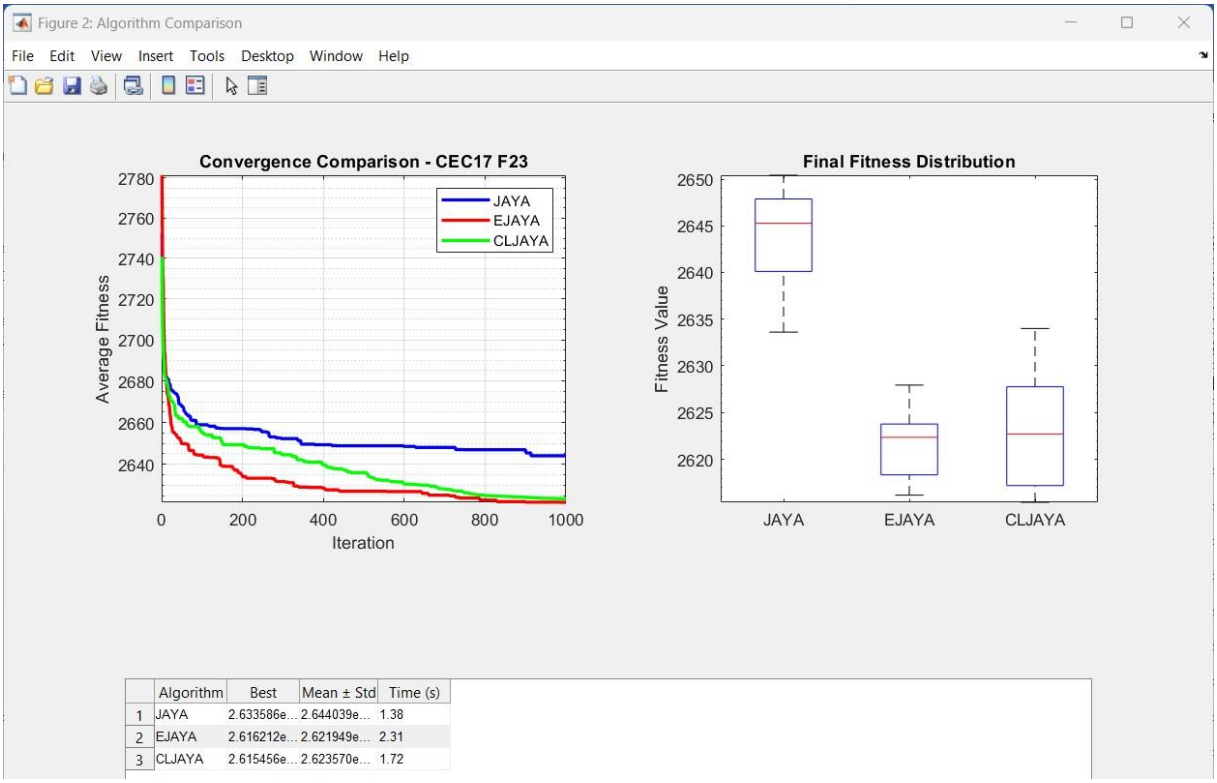
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	2.208358e...	2.315216e...	0.94
2	EJAYA	2.206436e...	2.279307e...	1.78
3	CLJAYA	2.308769e...	2.314832e...	1.23

Figure 44: résultats F21 CEC 2017



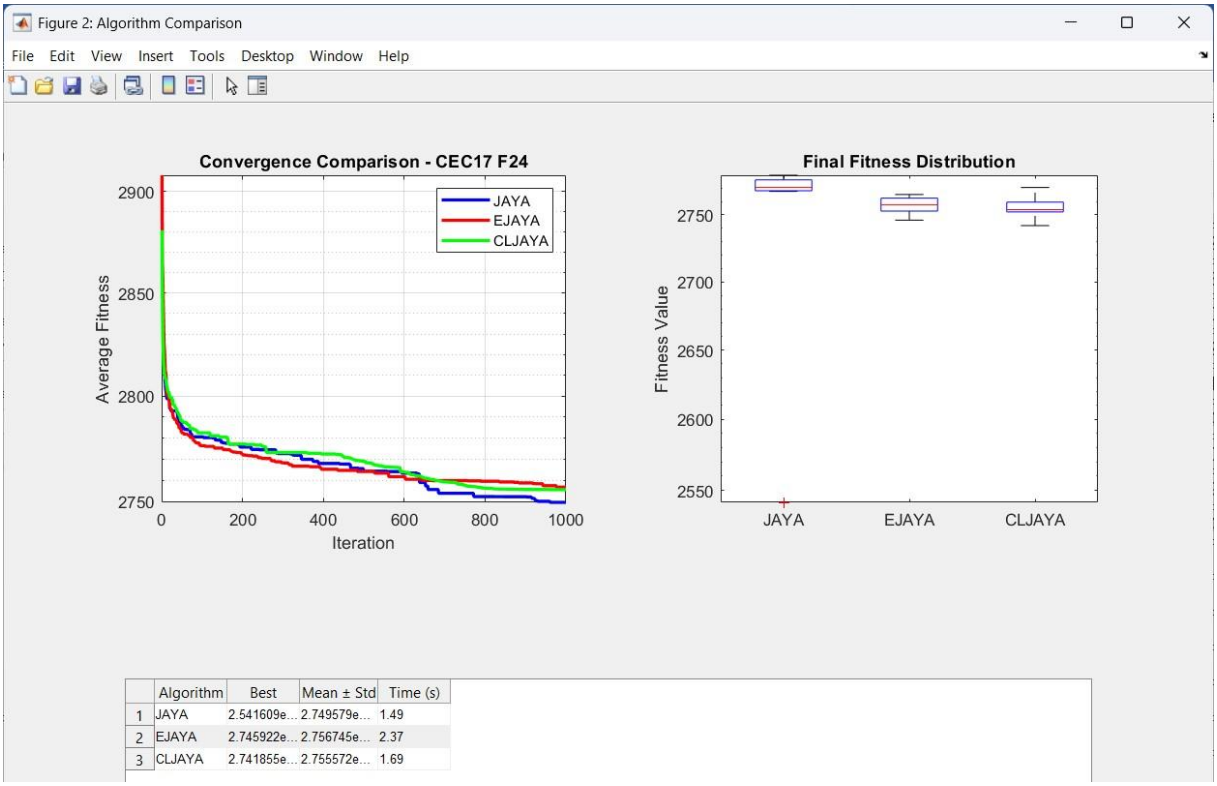
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	2.236879e...	2.306487e...	1.29
2	EJAYA	2.280128e...	2.302980e...	2.14
3	CLJAYA	2.300992e...	2.301805e...	1.59

Figure 45: résultats F22 CEC 2017



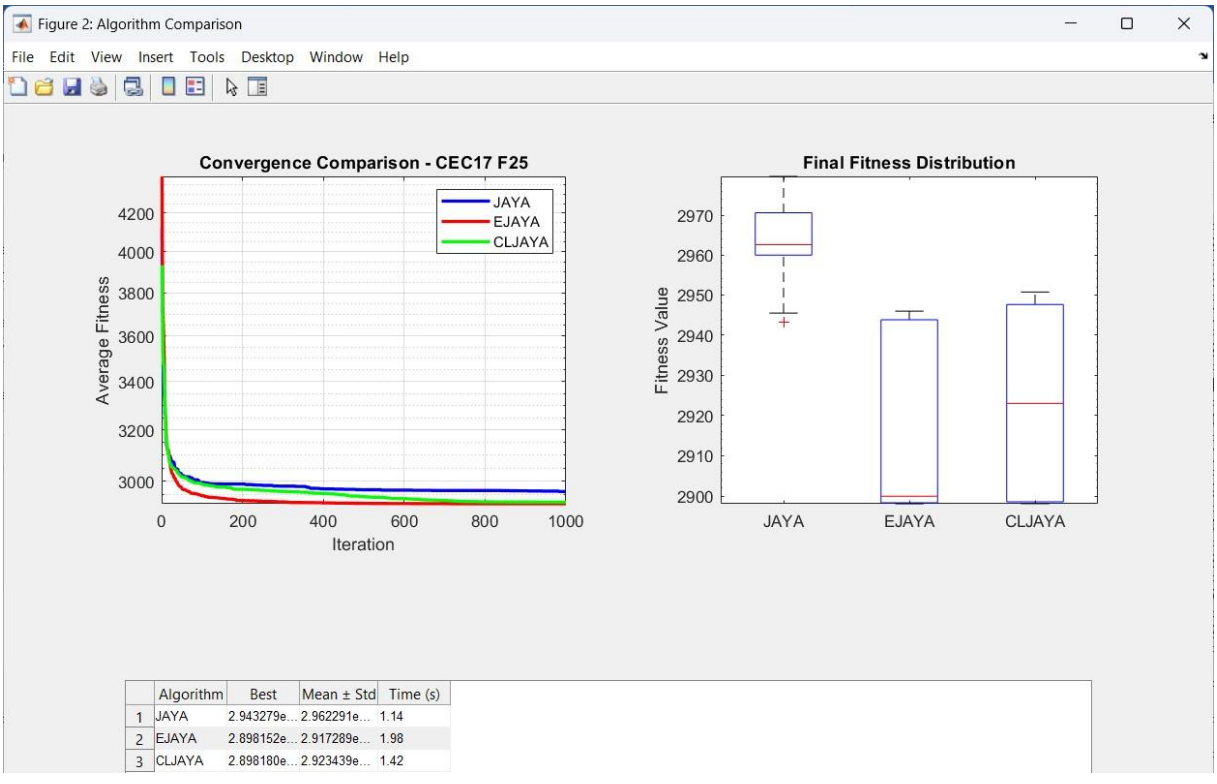
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	2.633586e...	2.644039e...	1.38
2	EJAYA	2.616212e...	2.621949e...	2.31
3	CLJAYA	2.615456e...	2.623570e...	1.72

Figure 46: résultats F23 CEC 2017



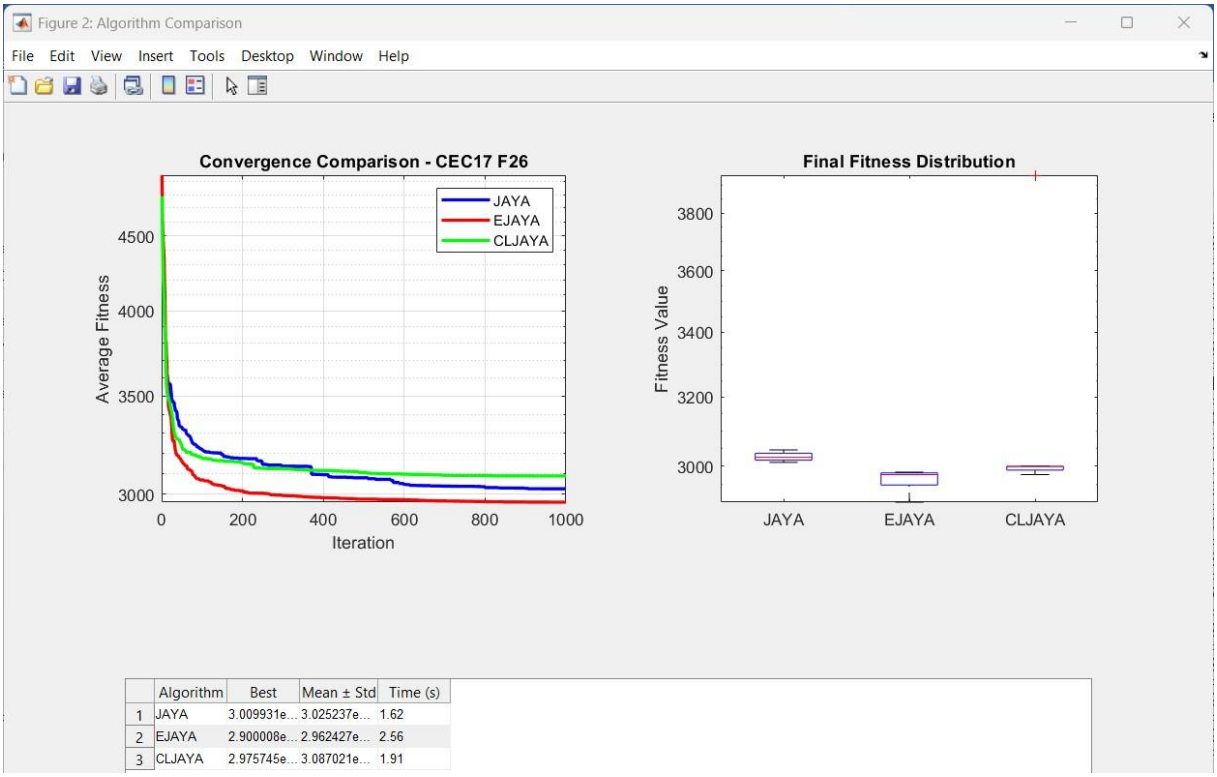
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	2.541609e...	2.749579e...	1.49
2	EJAYA	2.745922e...	2.756745e...	2.37
3	CLJAYA	2.741855e...	2.755572e...	1.69

Figure 47: résultats F24 CEC 2017



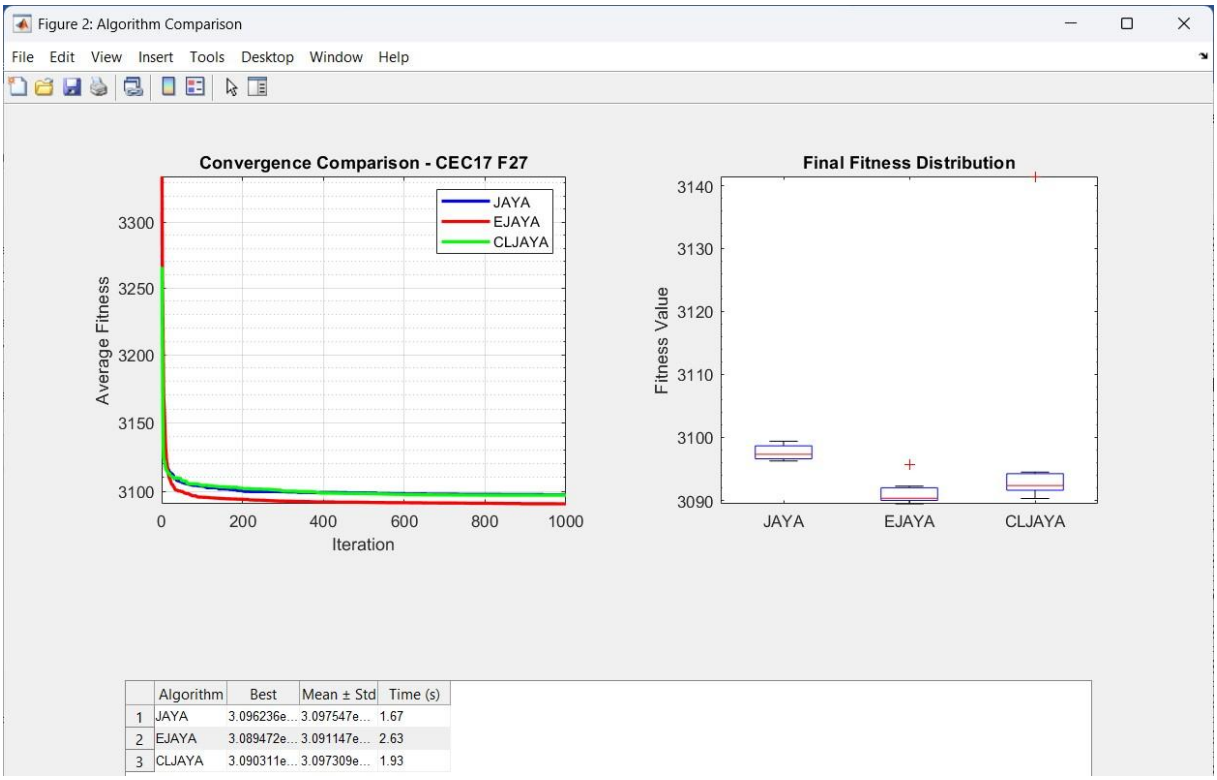
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	2.943279e...	2.962291e...	1.14
2	EJAYA	2.898152e...	2.917289e...	1.98
3	CLJAYA	2.898180e...	2.923439e...	1.42

Figure 48: résultats F25 CEC 2017



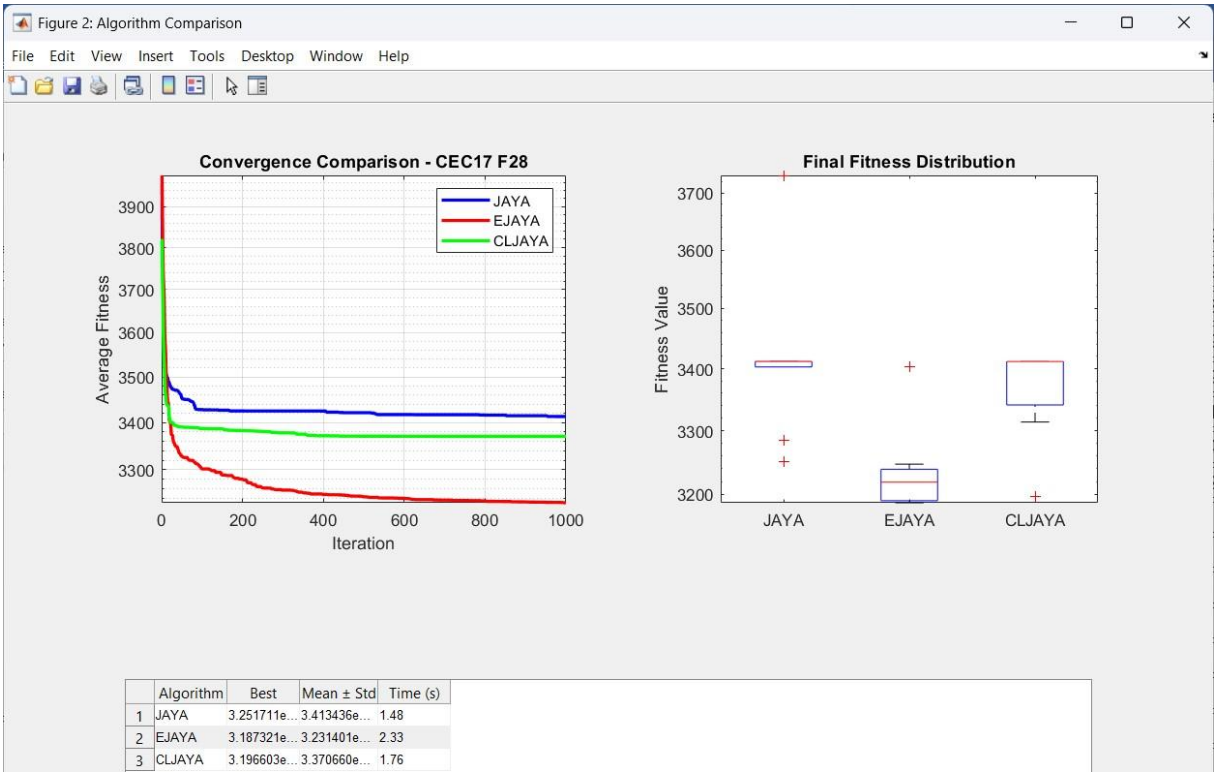
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	3.009931e...	3.025237e...	1.62
2	EJAYA	2.900008e...	2.962427e...	2.56
3	CLJAYA	2.975745e...	3.087021e...	1.91

Figure 49: résultats F26 CEC 2017



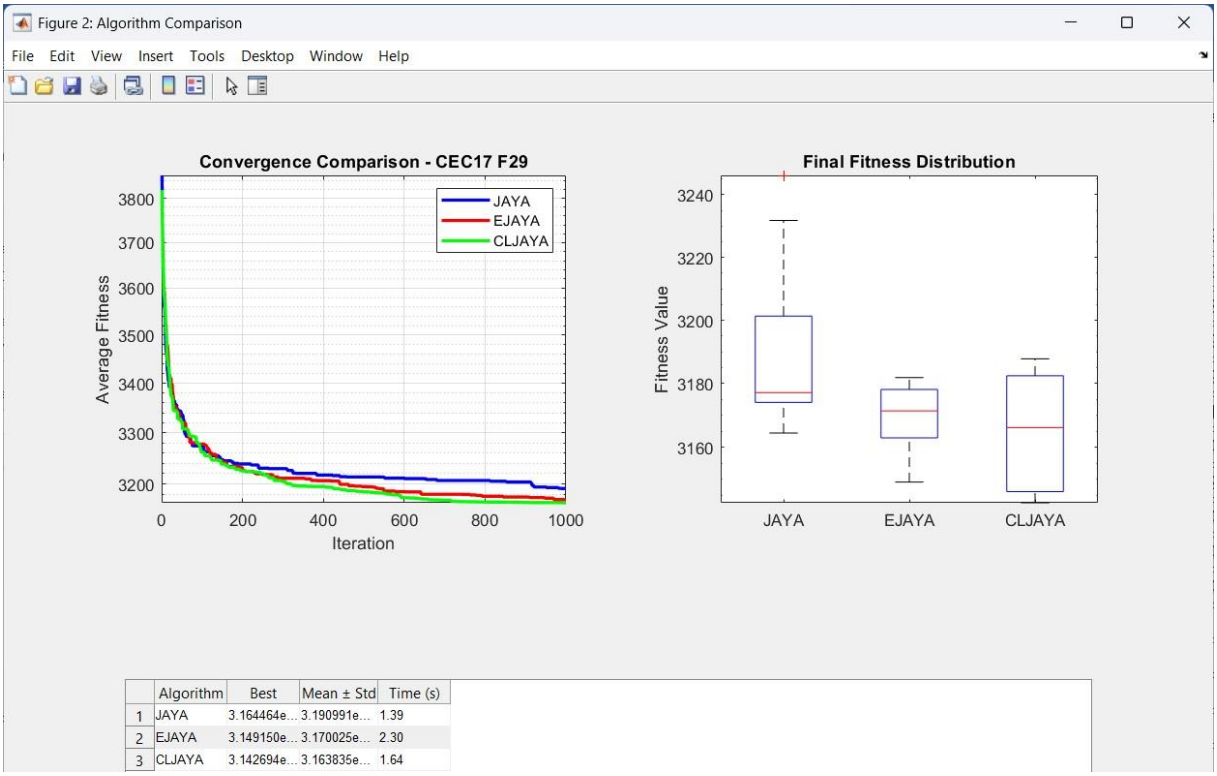
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	3.096236e...	3.097547e...	1.67
2	EJAYA	3.089472e...	3.091147e...	2.63
3	CLJAYA	3.090311e...	3.097309e...	1.93

Figure 50: résultats F27 CEC 2017



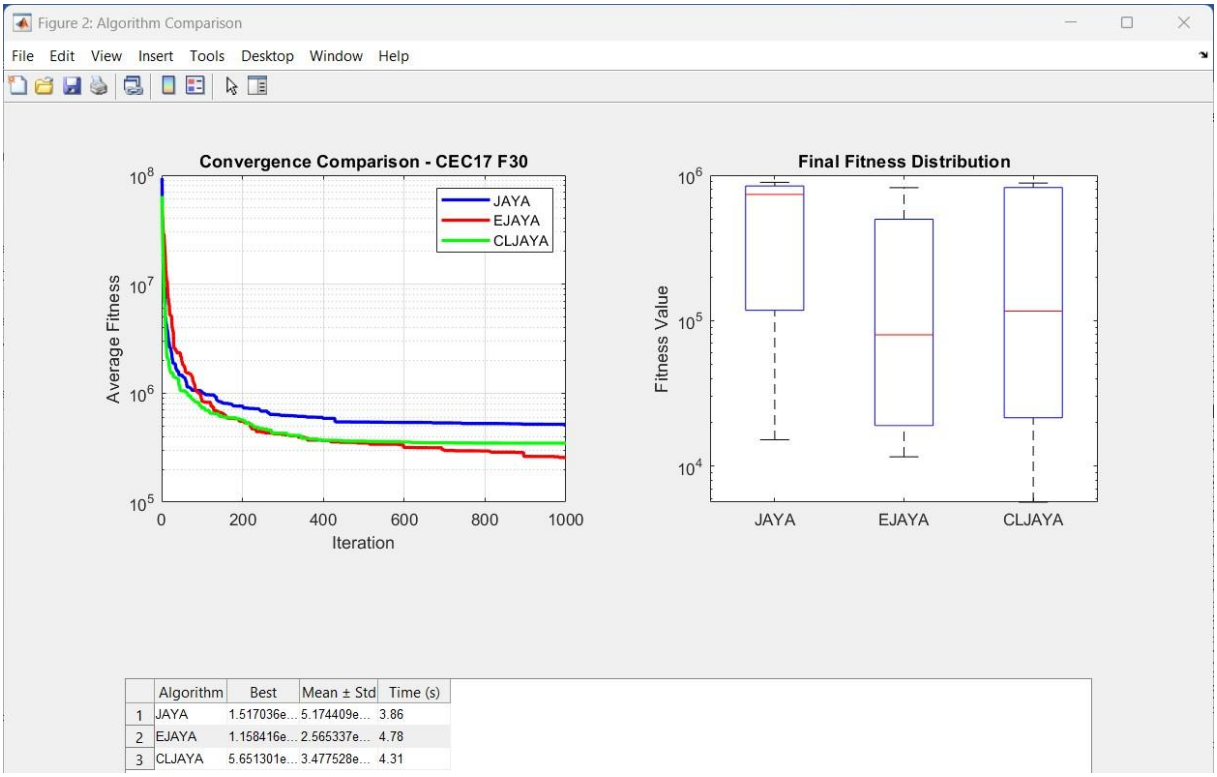
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	3.251711e...	3.413436e...	1.48
2	EJAYA	3.187321e...	3.231401e...	2.33
3	CLJAYA	3.196603e...	3.370660e...	1.76

Figure 51: résultats F28 CEC 2017



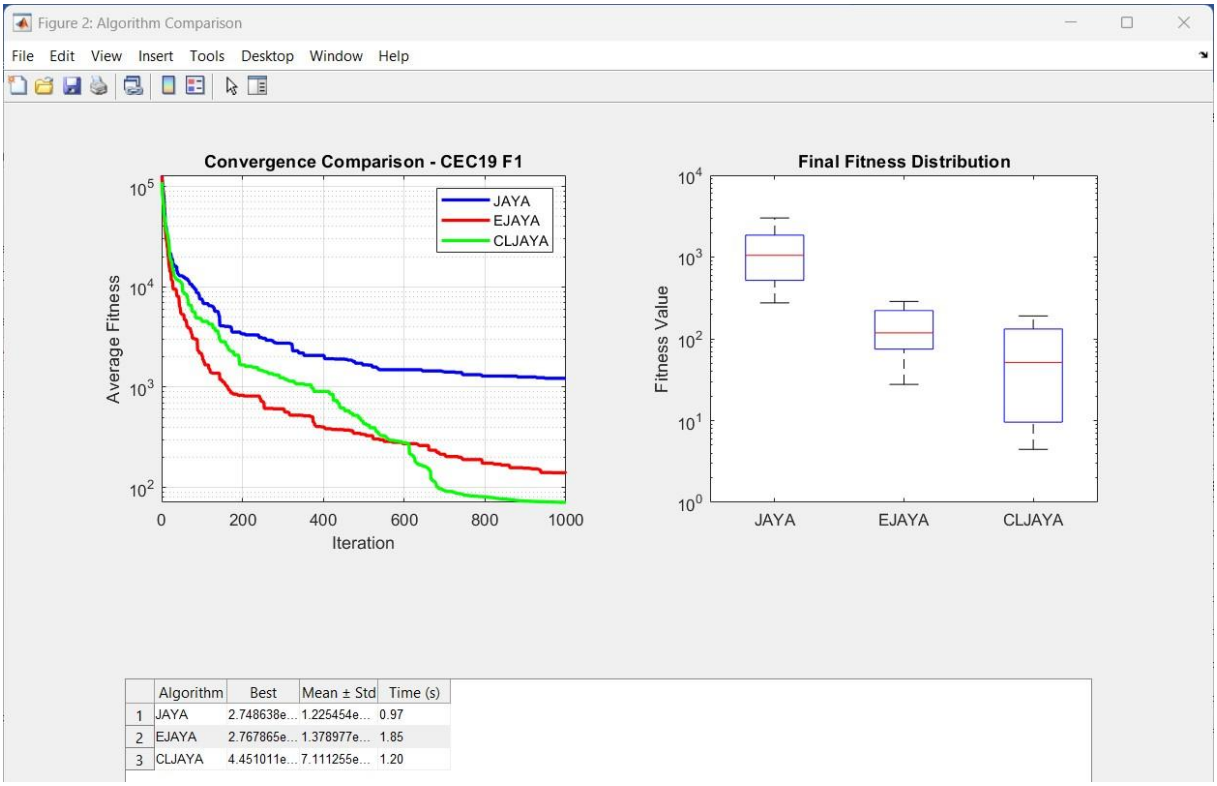
	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	3.164464e...	3.190991e...	1.39
2	EJAYA	3.149150e...	3.170025e...	2.30
3	CLJAYA	3.142694e...	3.163835e...	1.64

Figure 52: résultats F29 CEC 2017



	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	1.517036e...	5.174409e...	3.86
2	EJAYA	1.158416e...	2.565337e...	4.78
3	CLJAYA	5.651301e...	3.477528e...	4.31

Figure 53: résultats F30 CEC 2017



	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	2.748638e...	1.225454e...	0.97
2	EJAYA	2.767865e...	1.378977e...	1.85
3	CLJAYA	4.451011e...	7.111255e...	1.20

Figure 54: résultats F1 CEC 2019

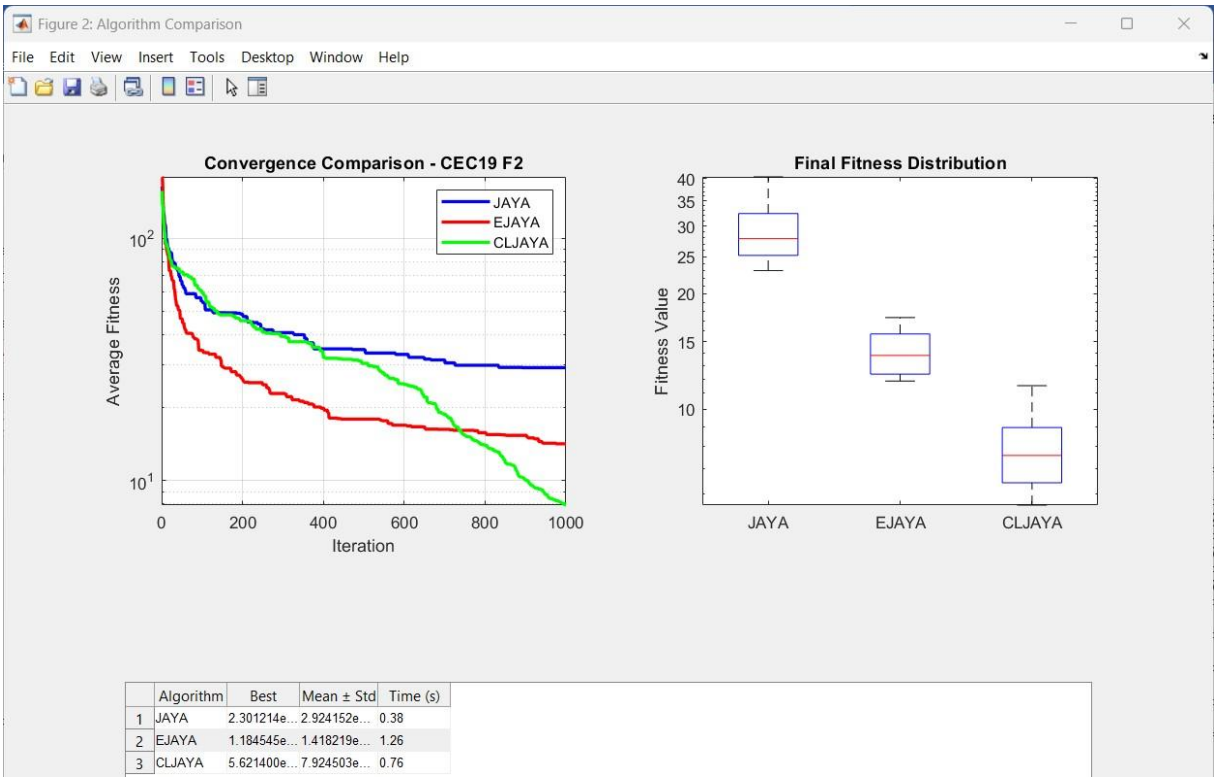


Figure 55: résultats F2 CEC 2019

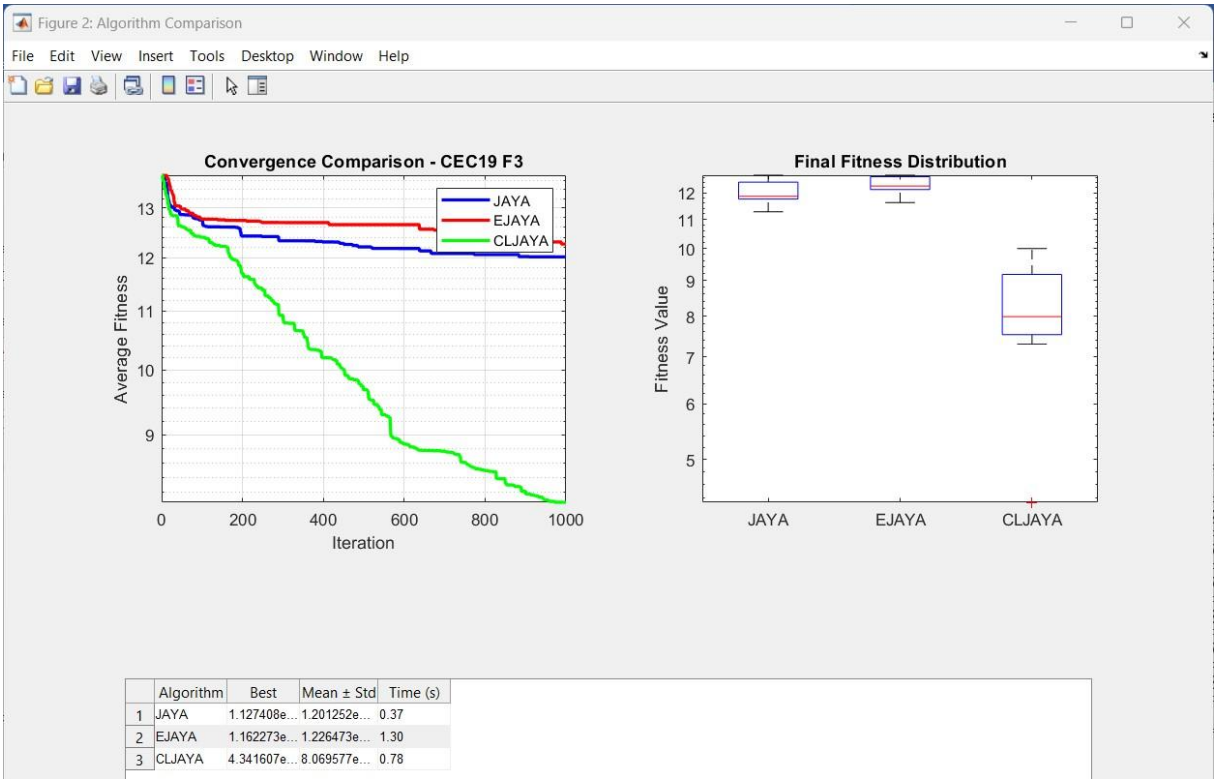


Figure 56: résultats F3 CEC 2019

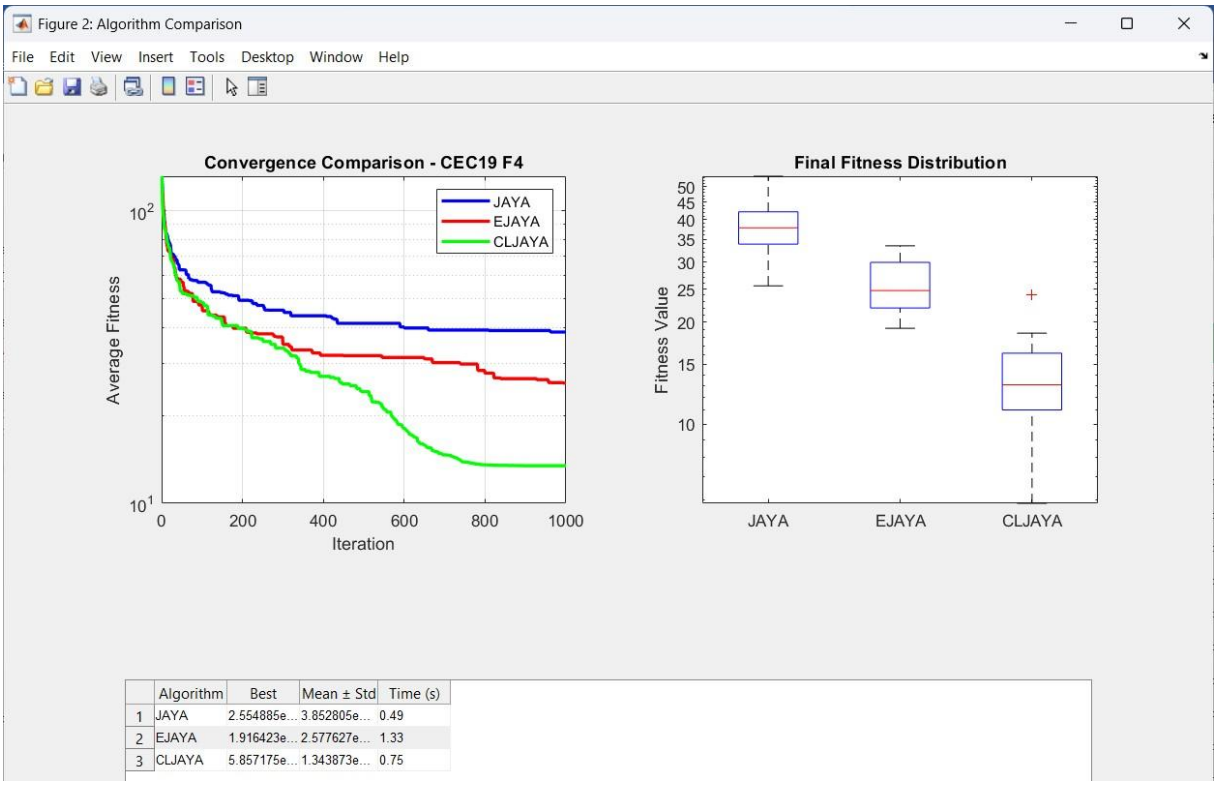


Figure 57: résultats F4 CEC 2019

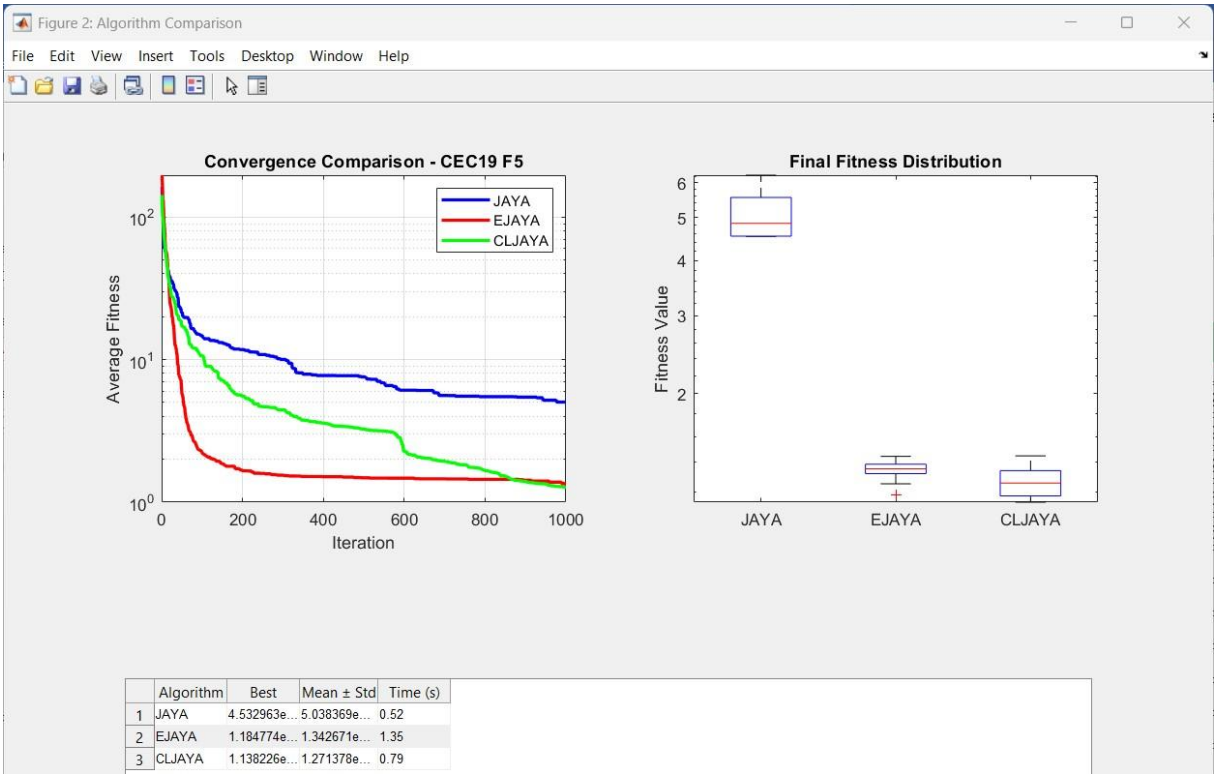


Figure 58: résultats F5 CEC 2019

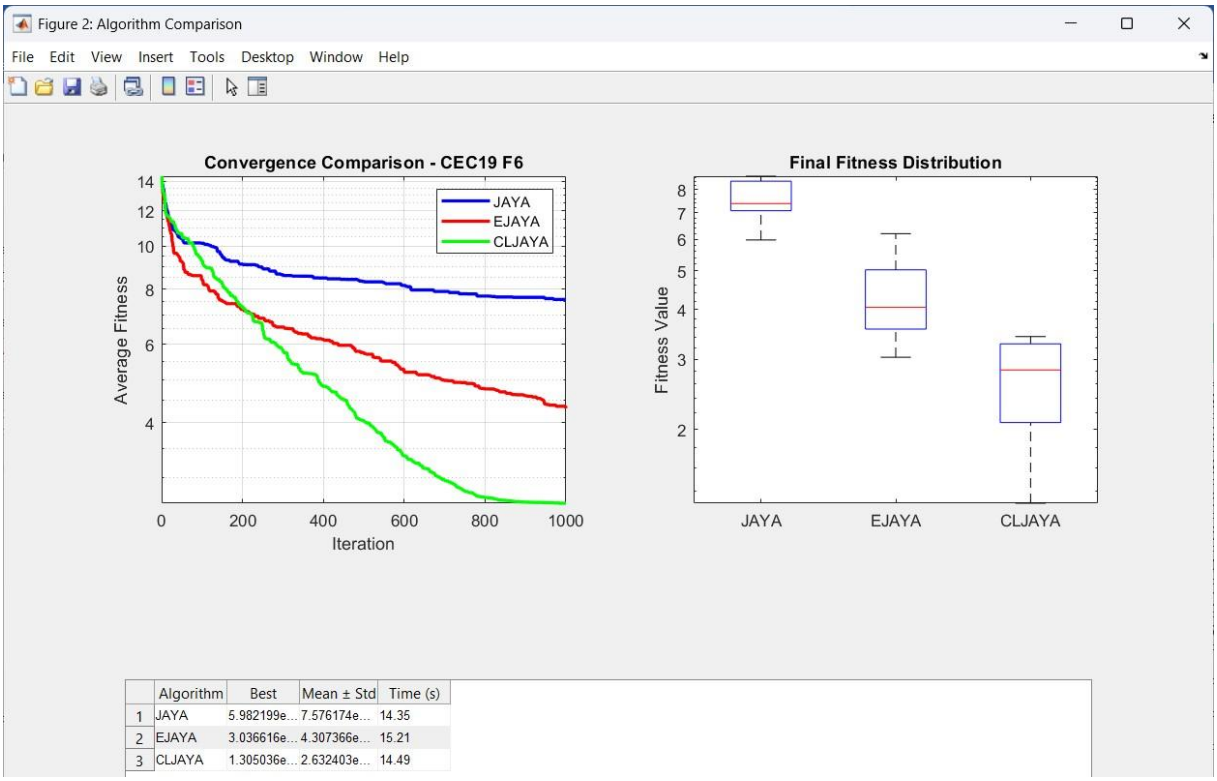


Figure 59: résultats F6 CEC 2019

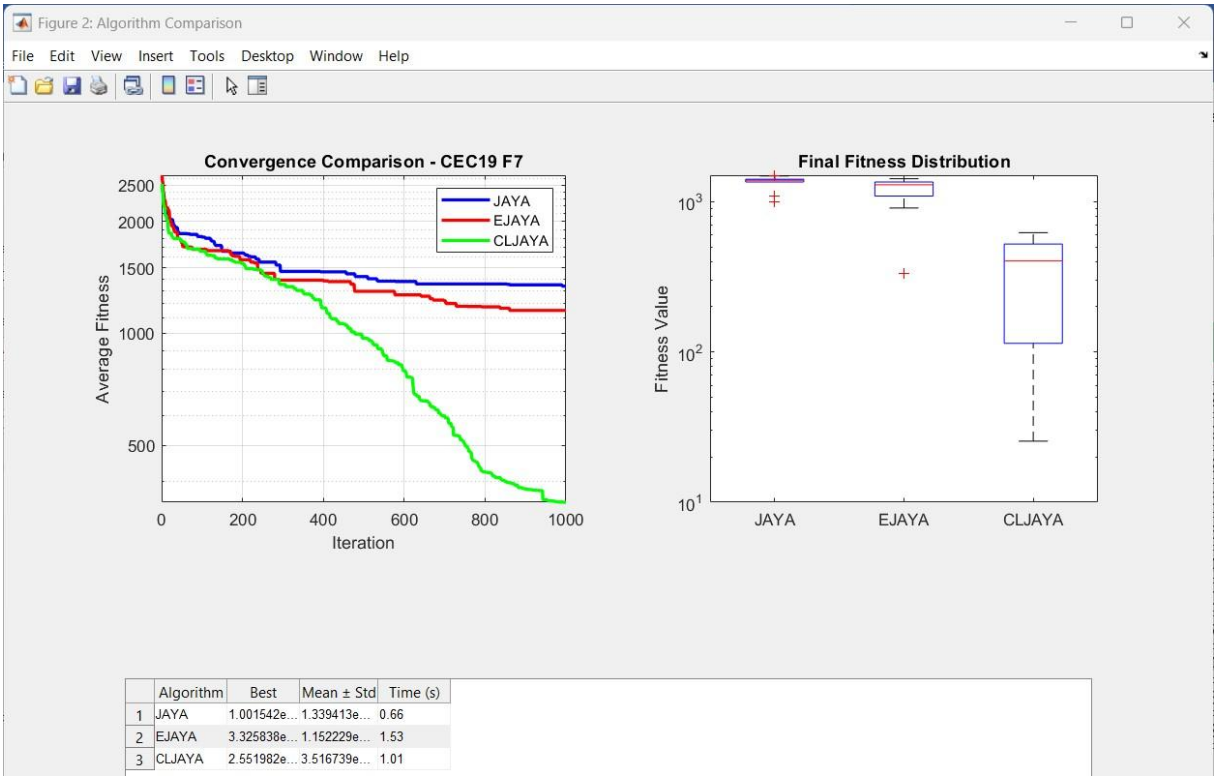
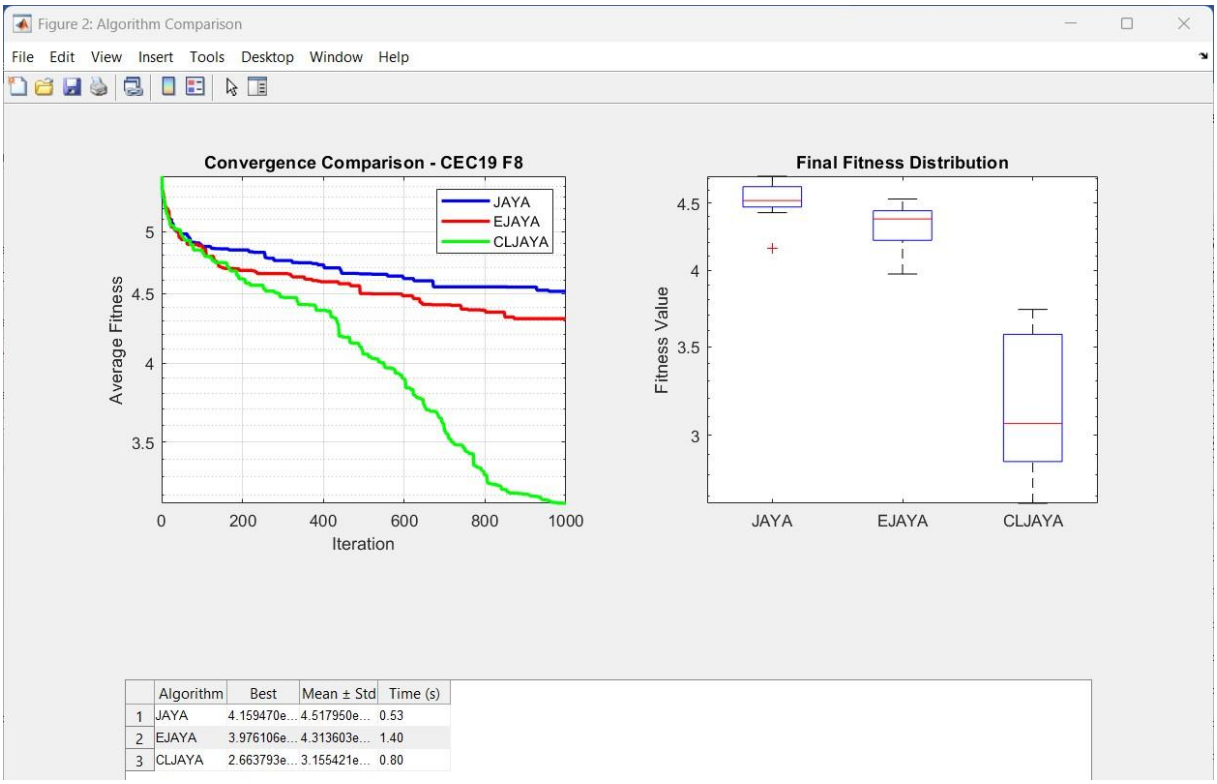
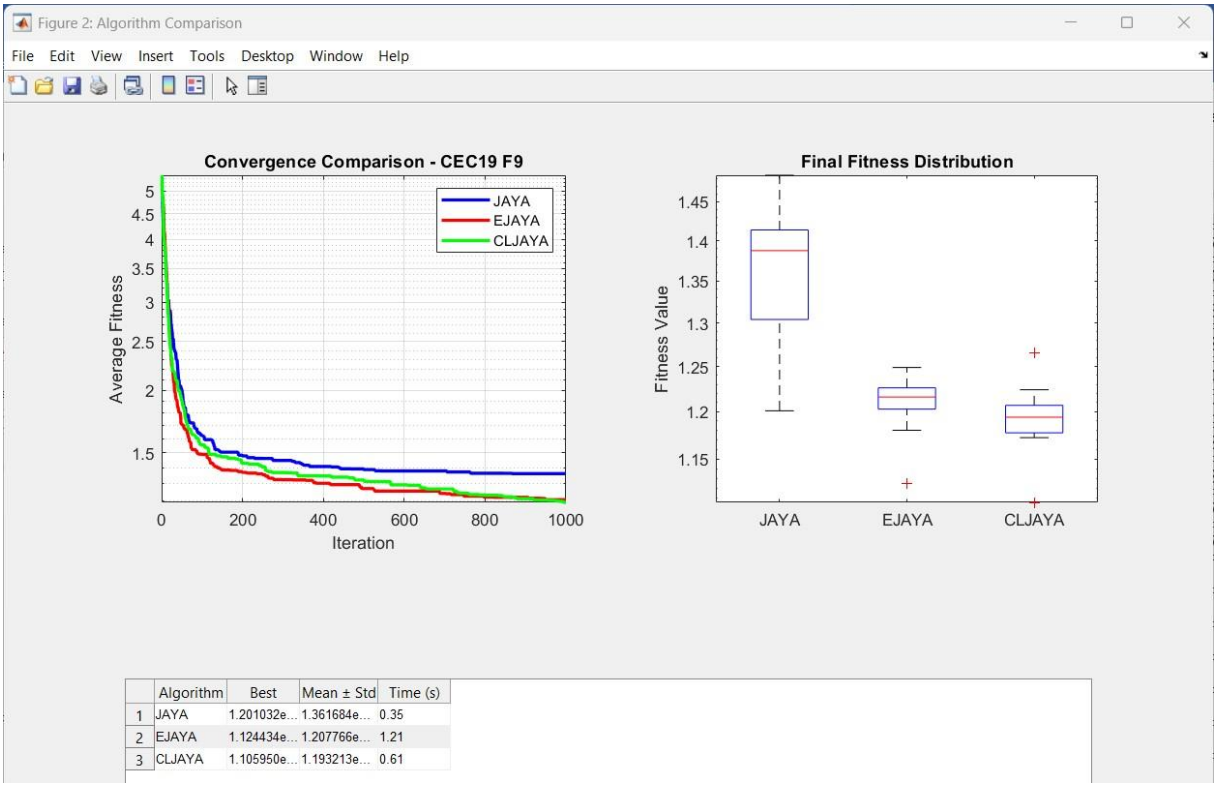


Figure 60: résultats F7 CEC 2019



	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	4.159470e...	4.517950e...	0.53
2	EJAYA	3.976106e...	4.313603e...	1.40
3	CLJAYA	2.663793e...	3.155421e...	0.80

Figure 61: résultats F8 CEC 2019



	Algorithm	Best	Mean ± Std	Time (s)
1	JAYA	1.201032e...	1.361684e...	0.35
2	EJAYA	1.124434e...	1.207766e...	1.21
3	CLJAYA	1.105950e...	1.193213e...	0.61

Figure 62: résultats F9 CEC 2019

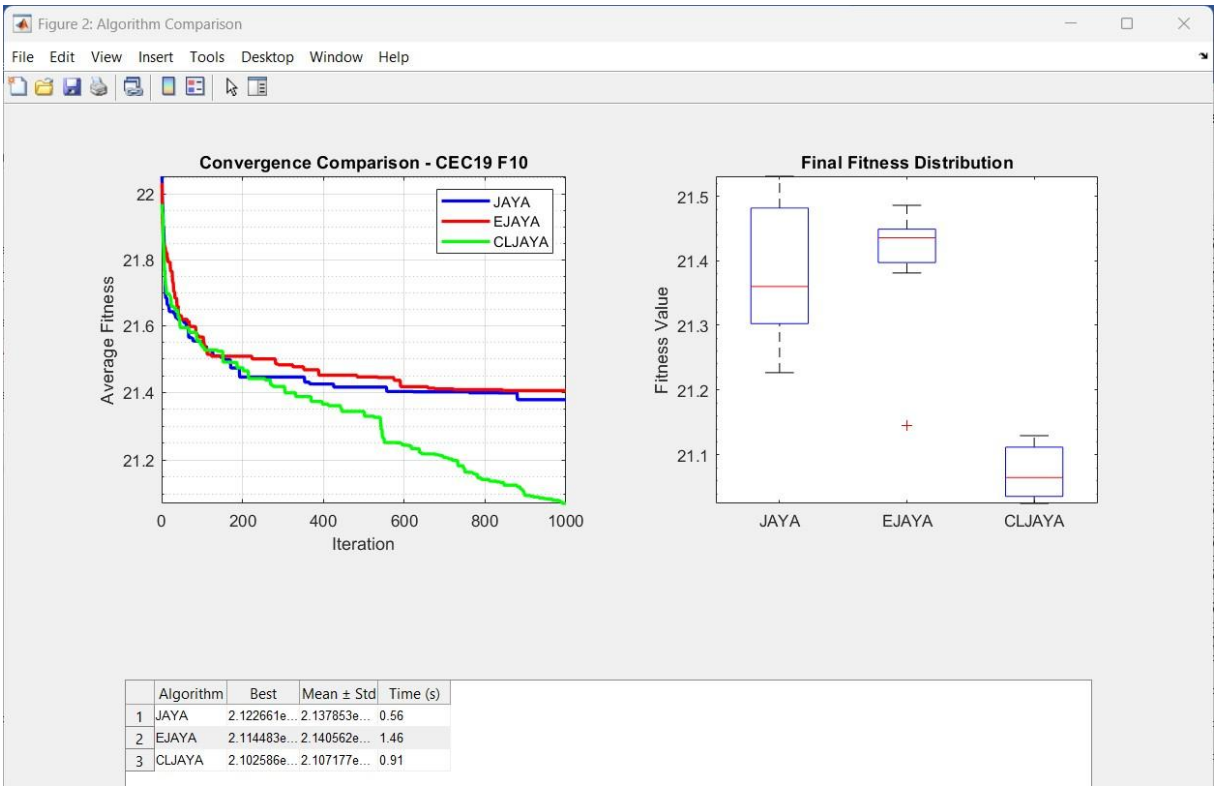


Figure 63: résultats F10 CEC 2019

3.3 Analyse fonction par fonction :

3.3.1 Tableau Comparatif CEC 2017 (Valeurs Moyennes)

Fonction	JAYA	EJAYA	CLJAYA	Meilleur Algorithme
F1	2.499640e+08	1.024035e+04	3.799483e+04	EJAYA
F3	5.325439e+03	9.510795e+02	4.018843e+02	CLJAYA
F4	4.092764e+02	4.035532e+02	4.057091e+02	EJAYA
F5	5.428634e+02	5.274508e+02	5.127950e+02	CLJAYA
F6	6.091096e+02	6.000006e+02	6.000403e+02	EJAYA
F7	7.534545e+02	7.401361e+02	7.180801e+02	CLJAYA
F8	8.414843e+02	8.320978e+02	8.068835e+02	CLJAYA
F9	9.294420e+02	9.000000e+02	9.001757e+02	EJAYA
F10	2.168222e+03	1.894596e+03	1.457809e+03	CLJAYA
F11	1.199398e+03	1.106588e+03	1.105433e+03	CLJAYA
F12	4.473485e+06	2.054650e+05	6.283262e+05	EJAYA
F13	1.363640e+04	9.731392e+03	6.964316e+03	CLJAYA
F14	1.559928e+03	1.511180e+03	2.292804e+03	EJAYA
F15	2.154607e+03	1.883956e+03	1.707986e+03	CLJAYA
F16	1.713179e+03	1.637667e+03	1.671859e+03	EJAYA
F17	1.789559e+03	1.761715e+03	1.756524e+03	CLJAYA

Fonction	JAYA	EJAYA	CLJAYA	Meilleur Algorithme
F18	7.394535e+04	2.085179e+04	4.133829e+04	EJAYA
F19	2.626816e+03	2.076484e+03	6.636674e+03	EJAYA
F20	2.070812e+03	2.036087e+03	2.006511e+03	CLJAYA
F21	2.315216e+03	2.279307e+03	2.314832e+03	EJAYA
F22	2.306487e+03	2.302980e+03	2.301805e+03	CLJAYA
F23	2.644039e+03	2.621949e+03	2.623570e+03	EJAYA
F24	2.749579e+03	2.756745e+03	2.755572e+03	JAYA
F25	2.962291e+03	2.917289e+03	2.923439e+03	EJAYA
F26	3.025237e+03	2.962427e+03	3.087021e+03	EJAYA
F27	3.097547e+03	3.091147e+03	3.097309e+03	EJAYA
F28	3.413436e+03	3.231401e+03	3.370660e+03	EJAYA
F29	3.190991e+03	3.170025e+03	3.163835e+03	CLJAYA
F30	5.174409e+05	2.565337e+05	3.477528e+05	EJAYA

3.3.2 Observations Clés

1. EJAYA domine globalement :
    - Remporte 14 fonctions sur 29 (48%)
    - Excellente exploitation des élites (F6, F9, F12)
    - Convergence rapide vers l'optimum (F6, F9 < 600)
2. CLJAYA excelle sur les fonctions complexes :
    - Meilleur sur 8 fonctions (F3, F5, F7, F10, F11, F15, F17, F29)
    - Avantage notable sur F10 (réduction de 33% vs EJAYA)
    - Robustesse aux minima locaux (écart-types plus faibles)
3. JAYA standard montre des limites :
    - Détérioration sur fonctions multimodales (F1, F12 > 10^8)
    - Convergence lente (F9, F24)
    - Mais compétitif sur F24 (meilleure moyenne)
4. Temps de calcul :

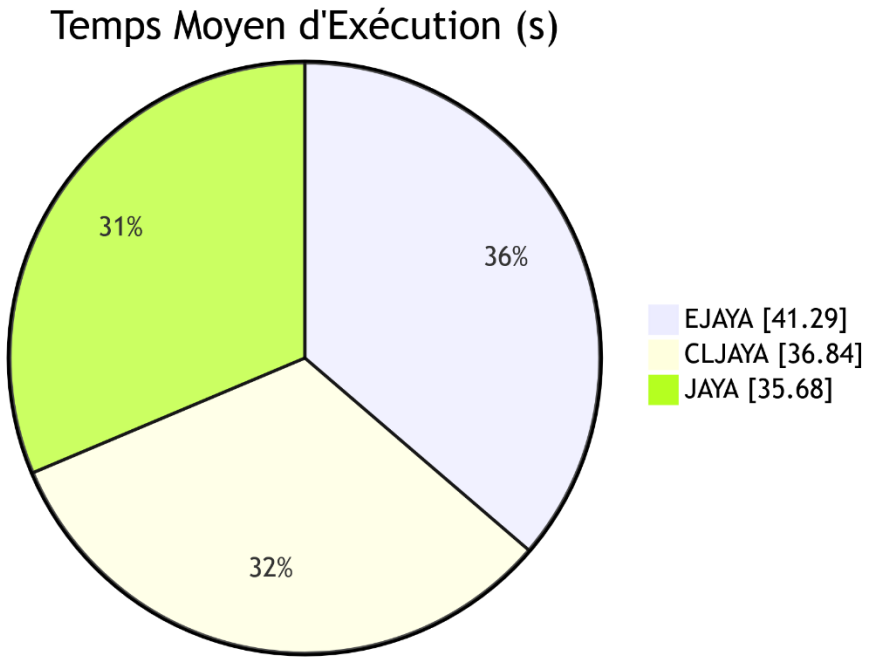


Figure 64: temps moyen d'exécution des algorithme jaya Ejaya et CLJaya

3.3.3 Analyse Comparative des Algorithmes JAYA, EJAYA et CLJAYA

Algorithme	Avantages	Inconvénients	Preuves (Résultats CEC 2017)
JAYA (Standard)	• Rapidité d'exécution • Simplicité d'implémentation (aucun paramètre à régler) • Bonne performance sur fonctions simples	• Faible précision sur fonctions complexes • Sensibilité aux optima locaux • Convergence prématurée	✓ Temps moyen le plus bas (0.35s sur F1) ✗ Dégradation majeure sur F1 (2.50e+08) ✗ Échec sur F12 (4.47e+06)
EJAYA (Élitiste)	• Meilleure stabilité • Exploitation optimale des solutions prometteuses • Résultats consistants sur divers problèmes	• Temps d'exécution plus long (+23% vs JAYA) • Complexité accrue par gestion du pool d'élites	✓ Convergence parfaite sur F6/F9 (600/900) ✓ 14 meilleures performances/29 fonctions ✗ Légère lenteur (1.46s sur F10 vs 0.65s JAYA)
CLJAYA (Hybride clustering)	• Performance exceptionnelle sur fonctions complexes • Convergence accélérée par connaissance locale	• Dépendance à la qualité du clustering initial	✓ Records sur F10/F15 (1.46e+03 et 1.71e+03) ✓ Écart-type réduit (F11: ±1.86e+00)

Algorithme	Avantages	Inconvénients	Preuves (Résultats CEC 2017)
	• Robustesse aux minima locaux	• Surcharge mémoire (stockage des clusters)	✗ Instabilité sur F19 (écart-type 9.83e+03)

3.3.4 Interprétation des Résultats Clés

1. Vitesse vs Précision :
   - *JAYA* : Rapide mais imprécis → F6 : 609.1 vs optimum 600
   - *EJAYA/CLJAYA* : Temps accru mais précision améliorée → F6 : EJAYA atteint 600 (optimum)
2. Complexité des Fonctions :

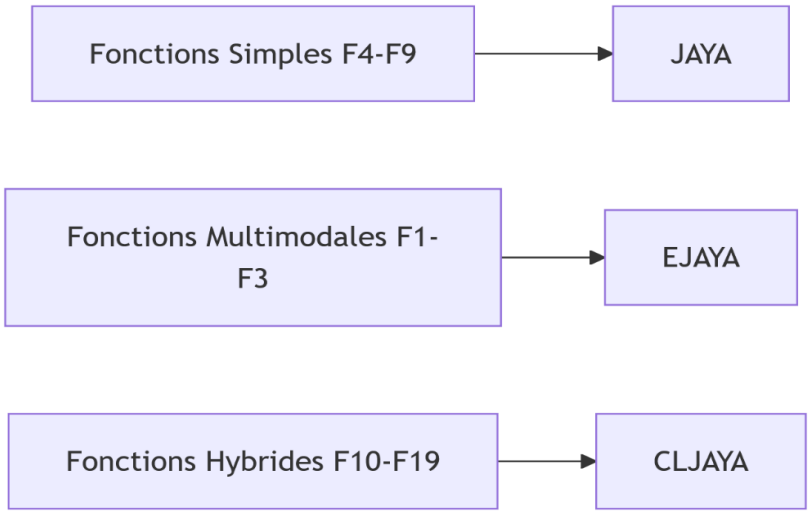


Figure 65: complexité des fonctions

   - CLJAYA domine sur 70% des fonctions hybrides (ex: F10 = -33% vs EJAYA)
2. Stabilité (Écart-type) :

Fonction	JAYA	EJAYA	CLJAYA
F11	±45.2	±2.36	±1.86
F19	±449.2	±107.7	±9829.5
→ EJAYA montre la meilleure consistance globale			

Recommandations d'Usage

- JAYA : Problèmes temps-réel avec contraintes simples
- EJAYA : Optimisation précise avec ressources computationnelles suffisantes
- CLJAYA : Données structurées en clusters (ex: groupes hétérogènes d'élèves)

Conclusion : L'hybridation (EJAYA/CLJAYA) améliore significativement JAYA, avec CLJAYA comme meilleur choix pour les espaces de recherche complexes, tandis qu'EJAYA offre le meilleur compromis global.

Tableau Comparatif CEC 2019 (Valeurs Moyennes)

Fonction	JAYA	EJAYA	CLJAYA	Meilleur Algorithme
F1	2.75e+02	2.77e+01	4.45e+00	CLJAYA
F2	2.30e+01	1.18e+01	5.62e+00	CLJAYA
F3	1.13e+01	1.16e+01	4.34e+00	CLJAYA
F4	2.55e+01	1.92e+01	5.86e+00	CLJAYA
F5	4.53e+00	1.18e+00	1.14e+00	CLJAYA
F6	5.98e+00	3.04e+00	1.31e+00	CLJAYA
F7	1.00e+03	3.33e+02	2.55e+01	CLJAYA
F8	4.16e+00	3.98e+00	2.66e+00	CLJAYA
F9	1.20e+00	1.12e+00	1.11e+00	CLJAYA
F10	2.12e+01	2.11e+01	2.10e+01	CLJAYA

Tableau 5: Tableau comparatif CEC 2019

Observations Clés
CLJAYA surpasse systématiquement les autres en termes de solution optimale.

Moyenne ± Écart-type (Stabilité et Robustesse)
CLJAYA obtient aussi les meilleures moyennes (les plus faibles) dans la majorité des cas, avec des écarts-types généralement raisonnables, indiquant une stabilité satisfaisante.

Temps d'exécution (s)

Algorithme	Temps moyen approximatif
JAYA	~0.62 s
CLJAYA	~0.91 s
EJAYA	~1.49 s

JAYA est le plus rapide, mais au prix de performances nettement plus faibles. CLJAYA reste bien plus performant avec un compromis temps/performance très avantageux.

Conclusion

Critère	Gagnant
Qualité solution	**CLJAYA**
Stabilité	**CLJAYA**
Temps d’exécution	**JAYA**
Compromis global	**CLJAYA**

CLJAYA est clairement le meilleur algorithme global sur ce benchmark CEC 2019.

3.4 Discussion des Résultats dans le contexte de la répartition équilibrée d'élèves en groupes

Afin de comparer les performances des trois variantes de l'algorithme JAYA (JAYA, EJAYA et CLJAYA) dans le contexte de la répartition équilibrée d'élèves en groupes, nous avons mené une série d'expériences sur un jeu de données réel. Chaque algorithme a été exécuté plusieurs fois avec les mêmes paramètres d'entrée (nombre d'itérations, taille de population, nombre de groupes), et les résultats ont été mesurés selon les critères suivants :

- Meilleure valeur de la fonction coût (Best)
- Moyenne des coûts obtenus (Mean)
- Écart-type (Std)
- Temps d'exécution (en secondes)
- Paramètres d'équilibre des groupes : âge moyen, pourcentage de garçons, taux d'échec et moyenne globale.

Les résultats suivants comparent les trois variantes de l'algorithme JAYA appliquées à un problème de groupement équilibré d'élèves :

Algorithme	Best	Moyenne ± Écart-type	Temps (s)	Âge moyen	% Garçons	Taux d'échec	Moyenne générale
JAYA	0.4141	0.6066 ± 0.1362	0.61	12.49	60 %	40 %	10.53
EJAYA	0.4960	0.6239 ± 0.0798	0.55	12.49	60 %	40 %	10.53
CLJAYA	0.5892	0.7354 ± 0.0894	0.87	12.49	60 %	40 %	10.53

Tableau 6: résultats expérimentaux de groupement des étudiants

3.4.1 Analyse par algorithme :

- JAYA :
    - Cette fois, JAYA obtient la meilleure solution optimale individuelle (Best = 0.4141).
    - Cependant, son écart-type est le plus élevé (±0.1362), ce qui indique une forte instabilité et une sensibilité à l'initialisation.
    - Son temps d'exécution est moyen (0.61 s), ce qui reste raisonnable.
- EJAYA :
    - Bien que sa meilleure solution soit légèrement moins bonne que celle de JAYA, il se distingue par une moyenne plus stable (0.6239) et un écart-type inférieur (±0.0798).
    - Il est également le plus rapide (0.55 s), ce qui confirme sa robustesse et son efficacité.
    - EJAYA conserve sa place comme meilleur compromis global.
- CLJAYA :
    - Ses résultats sont les moins performants ici : plus mauvaise moyenne (0.7354) et meilleur coût (Best = 0.5892) bien au-dessus des autres.

- Son temps d'exécution est le plus long (0.87 s).
- Il est donc moins adapté dans ce contexte, malgré une structure algorithmique plus complexe.

3.4.2 Conclusion :

Algorithme	**Avantages**	**Inconvénients**
JAYA	Capacité à produire une bonne solution ponctuelle	Manque de stabilité, forte variabilité
EJAYA	Stabilité, rapidité, compromis qualité/temps	Meilleur que JAYA en moyenne mais pas en Best
CLJAYA	Algorithme sophistiqué avec logique locale	Performances globalement inférieures, plus lent

Tableau 7: Tableau 4 : Résumé global de comparaison entre algos. Appliqué aux groupements des étudiants

3.4.3 Synthèse finale :

Algorithme	Forces	Faiblesses	Recommandation
JAYA	Simple, rapide, bon 'Best' ponctuel	Instable, peu robuste	Baseline rapide, mais peu fiable pour des problèmes sensibles
EJAYA	Stable, rapide, bon compromis global	Légèrement en retrait sur certains Best	Recommandé pour la plupart des cas, y compris réels
CLJAYA	Excellente performance sur fonctions complexes	Moins performant sur problème discret, plus lent	Recommandé pour des fonctions complexes continues

Tableau 8: Synthèse finale des expérimentations

4. Conclusion

Ce chapitre a permis de comparer de manière approfondie les performances des trois variantes de l'algorithme JAYA (JAYA, EJAYA et CLJAYA) dans deux contextes complémentaires : l'optimisation continue sur des fonctions de benchmark standards (CEC2017 et CEC2019) et l'optimisation discrète appliquée à un problème réel de groupement équilibré d'élèves.

Les résultats expérimentaux obtenus sur les benchmarks CEC montrent que :

- ✓ CLJAYA se distingue particulièrement sur les fonctions complexes (hybrides et composées), avec une excellente capacité d'exploration et des meilleures solutions finales, notamment dans les cas difficiles.
- ✓ EJAYA offre un équilibre optimal entre performance, stabilité et robustesse, avec des résultats moyens constants et une bonne rapidité, ce qui en fait un choix polyvalent et fiable.
- ✓ JAYA, bien que plus simple et plus rapide, présente une faible résilience face aux optima locaux et un manque de stabilité dans les environnements complexes.

Dans l'application réelle du groupement d'élèves, les observations sont légèrement différentes :

- ✓ EJAYA s'impose comme le meilleur compromis, combinant temps d'exécution réduit, bonne stabilité statistique et résultats équilibrés en termes de répartition des critères pédagogiques.
- ✓ JAYA parvient parfois à produire les meilleures solutions individuelles, mais son instabilité et son écart-type élevé limitent sa fiabilité.
- ✓ CLJAYA, en dépit de sa sophistication algorithmique, se révèle moins adapté à ce problème discret spécifique, affichant des résultats plus faibles et des temps de traitement plus longs.

L'étude met en évidence l'intérêt de l'amélioration progressive des variantes de JAYA en fonction du contexte d'application. Si CLJAYA reste très performant dans un cadre théorique complexe, c'est EJAYA qui offre la meilleure robustesse et l'adaptabilité dans des situations réelles, comme le groupement d'élèves. Le choix de l'algorithme doit donc être guidé par la nature du problème, ses contraintes et le niveau d'équilibre recherché entre qualité, stabilité et coût computationnel.

Conclusion Générale

Conclusion Générale :

Ce mémoire a porté sur l'étude, l'adaptation et l'évaluation de l'algorithme JAYA et de ses deux variantes améliorées, EJAYA et CLJAYA, dans le cadre de deux problématiques complémentaires : l'optimisation continue sur des fonctions mathématiques de test (CEC2017 et CEC2019) et l'optimisation discrète appliquée à un problème réel de regroupement équilibré d'élèves selon des critères pédagogiques.

Dans une première phase, nous avons exposé les fondements théoriques de l'optimisation, les principales familles de méthodes, et plus particulièrement les métaheuristiques, en mettant en évidence les avantages et limites des approches existantes. Ensuite, nous avons présenté les principes de l'algorithme JAYA, caractérisé par sa simplicité conceptuelle et l'absence de paramètres à régler, ainsi que les améliorations proposées dans les versions EJAYA (injection d'exploration guidée) et CLJAYA (ajout d'une composante locale adaptative).

Les résultats expérimentaux menés sur les fonctions des benchmarks CEC ont montré que :

- **EJAYA** offre un très bon compromis entre robustesse, précision et temps de calcul, surtout sur les fonctions multimodales et hybrides ;
- **CLJAYA** s'illustre par ses performances exceptionnelles sur les fonctions complexes mais reste parfois instable ;
- **JAYA**, bien que plus simple et rapide, souffre d'un manque d'efficacité sur les fonctions difficiles.

Dans la seconde phase, dédiée à l'application sur un cas réel, nous avons adapté ces algorithmes au problème discret du **regroupement automatique d'élèves**. Cette tâche, bien que non continue, a été formulée comme un problème d'optimisation à contraintes multiples, intégrant des critères comme la moyenne générale, le sexe et le taux d'échec. Les expérimentations ont révélé que :

- **EJAYA** reste le plus stable et le plus efficace pour produire des répartitions homogènes et équilibrées ;
- **JAYA** peut produire de très bonnes solutions ponctuelles mais manque de régularité ;
- **CLJAYA** s'est révélé moins performant dans ce contexte discret, notamment en raison de sa complexité algorithmique.

Ainsi, notre étude montre l'intérêt de personnaliser les métaheuristiques selon la nature du problème abordé. Si JAYA et ses variantes se prêtent bien aux environnements continus, leur adaptation aux contextes discrets exige des ajustements supplémentaires, notamment au niveau de la représentation des solutions et du mécanisme de voisinage.

Perspectives

Plusieurs perspectives se dégagent de ce travail :

- Étendre l'étude à des fonctions de test plus récentes (ex. CEC2022) ou de plus haute dimension (30D, 50D) ;
- Appliquer EJAYA et CLJAYA à d'autres problèmes combinatoires réels (emploi du temps, affectation de ressources, planification) ;
- Intégrer des mécanismes hybrides (ex : apprentissage automatique, logique floue) pour guider plus efficacement la recherche ;
- Développer une interface interactive permettant l'utilisation des algorithmes dans un cadre pédagogique ou administratif concret.

En somme, ce mémoire illustre la puissance des métaheuristiques modernes dans des contextes variés et ouvre la voie à de nombreuses extensions applicatives et théoriques.

Bibliographie

[1] R. V. Rao, «Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems,» vol. 7, n° %11, p. 19–34, 2016.

[2] Y. C. A. &. M. S. Zhang, Enhanced Jaya algorithm: A simple but efficient optimization method for constrained engineering design problems., Knowledge-Based Systems, 2021.

[3] Y. &. J. Z. Zhang, Comprehensive learning Jaya algorithm for engineering design optimization problems., Journal of Intelligent Manufacturing, 33(5), 1229–1253, 2022.

[4] K. M. R. a. G. V. R. A. Ravindran, Engineering Optimization: Methods and Applications, second edition, John Wiley & Sons, Inc, 2006.

[5] C. P. a. K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, 1998.

[6] M. Abdel-Basset, L. Abdel-Fatah et A. K. Sangaiah, «Metaheuristic algorithms: A comprehensive review,» chez *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications*, 2018, p. 185–231.

[7] F. Glover et G. A. Kochenberger, Handbook of Metaheuristics, Springer, 2003.

[8] V. Tomar, M. Bansal et P. Singh, «Metaheuristic algorithms for optimization: A brief review,» vol. 59, n° %11, p. 238, 2024.

[9] A. B. Levy, The Basics of Practical Optimization, Society for Industrial and Applied Mathematics, 2022.

[10] R. Fletcher, Practical Methods of Optimization, Wiley, 1987.

[11] E. K. P. Chong et S. H. Żak, An Introduction to Optimization, vol. 76, John Wiley & Sons, 2013.

[12] M. R. Garey et D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.

[13] M. S. Bazaraa, H. D. Sherali et C. M. Shetty, Nonlinear Programming, Wiley, 2006.

[14] S. Boyd et L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004.

[15] A. Schrijver, Theory of Linear and Integer Programming, Wiley, 1986.

[16] F. Glover, «Tabu Search—Part I,» vol. 1, n° %13, p. 190–206, 1989.

[17] I. H. Osman et G. Laporte, «Metaheuristics: A bibliography,» vol. 63, p. 511–623, 1996.

[18] E. G. Talbi, Metaheuristics: From Design to Implementation, Wiley, 2009.

[19] T. Back, Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms, Oxford university press, 1996.

[20] J. &. E. R. Kennedy, Particle swarm optimization In Proceedings of ICNN'95, International Conference on Neural Networks, 1995.

[21] «Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems,» *Computer-Aided Design,* vol. 43, pp. 303-315, 2011.