

EvoMapX: An explainable framework for metaheuristic optimization algorithms

Bilal H. Abed-Alguni 

Department of Computer Sciences, Yarmouk University, Irbid, Jordan

ARTICLE INFO

Keywords:

Explainable AI
Explainable optimization
Interpretable optimization
Population evolution graph
Operator attribution matrix
Search dynamics visualization
Convergence driver score
Evolutionary pathway mapping
CEC 2021

ABSTRACT

Population-based optimization algorithms (POAs) are widely adopted solutions for NP-hard and complex high-dimensional optimization problems. However, their internal dynamics often remain opaque, limiting trust and insight into how solutions evolve. This paper introduces EvoMapX, a novel explainable framework designed to interpret the internal dynamics of population-based optimization algorithms. EvoMapX includes three interpretable structures to visualize evolutionary optimization dynamics: the Operator Attribution Matrix (OAM) quantifies the contribution of specific operators over iterations; the Population Evolution Graph (PEG) traces the ancestry and transformation of candidate solutions; and the Convergence Driver Score (CDS) identifies which operators drive convergence, helping interpret why the algorithm improved. EvoMapX was evaluated across four POAs on the CEC 2021 test suite in order to demonstrate how it reveals meaningful textual and graphical insights into algorithm behavior. EvoMapX paves the way for interpretable metaheuristic optimization. The source code of EvoMapX is available at <https://github.com/Bilal20252025/EvoMapX>

1. Introduction

Metaheuristic optimization algorithms, such as Genetic Algorithm (GA) (Katoch et al., 2021), Differential Evolution (DE) (Chakraborty et al., 2023), Particle Swarm Optimization (PSO) (Gad, 2022), and Cuckoo Search (CS) (Guerrero-Luis et al., 2021), are highly effective methods for solving complex and NP-hard problems in domains ranging from feature selection (Abed-alguni & Al-Jarah, 2024; Agrawal et al., 2021; Al-qudah et al., 2024) and scheduling (Abed-alguni & Paul, 2022; Hosseini et al., 2024; Prity et al., 2023) to hyperparameter tuning (Bischl et al., 2023; Daviran et al., 2025). Metaheuristic optimization algorithms, especially population-based optimization algorithms (POAs), are attractive due to their efficiency, ease of use, and global search capabilities. However, POAs are “black box” methods, meaning that their internal behaviors are largely opaque (Corrente et al., 2024; Wang & Chen, 2024). This lack of transparency obstructs interpretability, trust, and informed decision-making, especially in high-stakes or safety-critical applications like healthcare and financial applications.

Due to the increased demand for transparency in AI systems, a new field has emerged: Explainable Artificial intelligence (XAI). XAI techniques aim to offer explanations in human language for the decisions, predictions, or actions carried out by AI systems. So far, many advances have been achieved in explaining deep learning and predictive models with these techniques: SHAP (SHapley Additive exPlanations) (Lundberg & Lee, 2017), LIME (Local Interpretable Model-Agnostic Explanations) (Ribeiro et al., 2016), and Grad-CAM (Gradient-weighted Class Activation Mapping) (Selvaraju et al., 2020). In contrast, little attention has been paid to the interpretability of metaheuristic optimization processes. Existing research studies in XAI for optimization techniques are either domain-specific or limited in scope. The fields of XAI and optimization actually lack comprehensive tools to trace and quantify the internal dynamics of POAs.

Recent research studies have begun to explore the application of XAI in optimization. They integrated tools like SHAP, LIME, and decision trees with evolutionary algorithms and swarm intelligence to improve transparency in scheduling, parameter tuning, and design optimization tasks (Dubey, 2024; Parkavi et al., 2024; Wang & Chen, 2024; Zhou et al., 2024). However, these methods remain mostly post hoc and external to the search process. They offer explanations of outputs rather than the dynamics of convergence. Visual methods such as fitness landscape analysis (Prager & Trautmann, 2024) and ancestry trees (Črepinšek et al., 2011) explain behavior but do not link fitness improvements to operator-level decisions across time. This highlights the need for process-level, real-time interpretability frameworks in metaheuristics, a gap that this paper aims to fill.

In order to bridge this gap, we propose EvoMapX, a novel explainable framework designed to enhance the transparency and interpretability of population-based metaheuristics. EvoMapX includes three interpretable structures: the Operator Attribution Matrix (OAM) for quantifying the contribution of specific operators (e.g., mutation,

E-mail address: Bilal.h@yu.edu.jo

<https://doi.org/10.1016/j.eswa.2025.129514>

Received 28 May 2025; Received in revised form 30 July 2025; Accepted 24 August 2025

Available online 15 September 2025

0957-4174/© 2025 Elsevier Ltd. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

crossover) over iterations; the Population Evolution Graph (PEG) for tracing the ancestry and transformation of candidate solutions; and the Convergence Driver Score (CDS) for identifying which operators drive convergence, helping interpret why the algorithm improved. Unlike formal convergence proofs, which address algorithm behavior in the limit, EvoMapX is designed to explain the practical mechanisms of convergence over finite runs, which help users understand not just whether an algorithm converges, but how and why it does so. Therefore, EvoMapX complements, rather than replaces, theoretical convergence.

Four well-known POAs (GA, DE, PSO, and CS) from the CEC 2021 benchmark suite are used to assess EvoMapX. Through both graphical and textual insights, the results show how EvoMapX can expose operator efficacy, identify stagnation periods, explain convergence behavior, and direct the improvement of optimization algorithms.

In this work, we focus on making two critical properties of POAs explainable: operator effectiveness and convergence dynamics. These properties are central to algorithm performance but are typically hidden within the iterative and stochastic nature of POAs. Operator effectiveness measures the fitness improvements of specific operators in an optimization algorithm (e.g., crossover, mutation, velocity update). On the other hand, convergence dynamics describe how search progresses, stagnates, or escapes local optima over time. EvoMapX targets these properties to make algorithm behavior transparent, diagnosable, and controllable in a structured and extensible manner. The key contributions of this paper are as follows:

- We introduce EvoMapX, the first integrated explainability framework tailored to population-based metaheuristics.
- We define and implement three interpretable structures (OAM, PEG, CDS) to dissect and visualize search dynamics.
- We conduct comprehensive experiments across CEC 2021 benchmark functions to validate EvoMapX's ability to generate interpretable explanations into operator effectiveness, convergence dynamics, and evolutionary pathways.
- We generate textual explanations using rule-based interpretation modules in order to bridge the gap between visual analytics and human-readable diagnostics.

EvoMapX makes the internal dynamics of metaheuristics transparent and interpretable, which helps academics and practitioners to better understand, trust, and improve optimization algorithms. Although many prior studies (Kora & Yadlapalli, 2017; Manazir & Raza, 2021; Osaba et al., 2014) have compared the statistical effects of individual operators (e.g. crossover and mutation in genetic algorithms), these analyses are often static, algorithm-specific, or limited to ablation-style comparisons. In contrast, EvoMapX provides a unified and extensible framework that enables real-time, per-iteration tracking of operator influence across multiple population-based algorithms. By combining OAM with the CDS and PEGs, EvoMapX reveals not just which operators are effective, but how they influence convergence pathways and fitness propagation over time. Unlike most studies that aim to improve optimization performance through algorithm design (Alawad et al., 2023, 2025), this work focuses on enhancing interpretability in POAs, which is a comparatively underexplored and increasingly relevant aspect, particularly in applications where transparency and algorithmic trust are critical.

The remainder of the document is organized into five sections: [Section 2](#) provides a thorough analysis of related work on XAI methods in the optimization field. [Section 3](#) outlines the fundamental ideas and methods relevant to this EvoMapX. [Section 4](#) presents the proposed EvoMapX framework in detail. [Section 5](#) reports and analyzes the experimental results of applying EvoMapX to four widely used POAs on the CEC 2021 benchmark functions. Finally, [Section 6](#) summarizes the main conclusions and outlines potential study avenues.

2. Related work

The review of the related work of XAI methods for optimization is organized into two parts: [Section 2.1](#) reviews how the standard XAI techniques (e.g., SHAP, LIME) were adapted to optimization tasks like scheduling, hyperparameter tuning, and feature selection, while [Section 2.2](#) focuses on techniques specifically designed for optimization tasks.

2.1. General XAI methods and their implementation in optimization

XAI techniques such as SHAP and LIME are being increasingly utilized in optimization algorithms to explain their black-box behaviors. These techniques provide insights into surrogate models and decision boundaries and explain feature importance with respect to optimization tasks such as hyperparameter tuning, scheduling, feature selection, or metaheuristic search. This section provides a review of recent research studies that discussed the implementation of common XAI methods in the optimization field.

Zhou et al. (2024) investigated the application of XAI methods in the processes of evolutionary computation (EC) to improve interpretability and transparency. They argued that while the EC methodologies such as genetic algorithms and swarm intelligence are very powerful, their black-box nature does not foster trust and confidence in those applying them for critical applications. Therefore, they proposed a framework for XAI in EC that includes techniques such as visualizing fitness landscapes, surrogate modeling for approximating solutions, performing feature importance analysis with SHAP or LIME, and rule extraction to try to decipher the decision-making process of evolution.

Wang and Chen (Wang & Chen, 2024) combined three common post-hoc explanation techniques (SHAP, LIME, and decision trees) and visual explanations to demonstrate how key GA decisions (e.g., selection, crossover, and mutation methods) can be made transparent. The authors stress the importance of feature importance analysis to display which scheduling constraints (e.g., deadlines, resource limits) exert the most influence on GA-generated solutions, alongside rule extraction to distill the processes of GAs into human-understandable logic. Furthermore, the authors used dynamic convergence visualization to help the users to track optimization paths, and they used counterfactual explanations to illustrate how minor changes in inputs could alter scheduling outcomes. The study emphasizes that XAI not only improves trust in GA-driven scheduling but also aids in debugging biases (e.g., premature convergence) and refining heuristic rules for better performance.

Dubey (Dubey, 2024) integrated XAI techniques with evolutionary algorithms (EAs) to enhance the interpretability of optimized engineering designs. He used decision trees, SHAP, and LIME to extract human-understandable rules from high-dimensional candidate solutions to explain why certain design parameters outperform others in their performance. The authors argue that their hybrid approach has interpreted Pareto-optimal trade-offs, feature importances, or counterfactual scenarios, explaining and justifying the EA's black-box decisions so engineers can trust them and improve upon them. In this sense, XAI promises real-time explanations of search behavior, sensitivity analyzes of design variables, and post hoc justification of optimal configurations in bridging automated optimization into human-centric decision-making in engineering.

Parkavi et al. (2024) proposed a decision support framework that combines chaotic particle swarm optimization (CPSO) with XAI techniques to improve personalized learning recommendations. The CPSO algorithm optimizes the learning path in a flexible manner by adapting to student performance data, while SHAP and LIME are integrated into the system for transparency in decision making. These XAI components are intended to give educators and learners insight into why particular recommendations, such as adjusting the style of the course or lowering the difficulty levels, are made through importance of features and contextual influences. The hybridization ensures optimization

efficiency using CPSO and interpretability using XAI, addressing AI-driven educational systems' "black-box" issue. The study illustrates how the usage of XAI improves trust and usability in adaptive learning contexts by making algorithmic decisions interpretable for stakeholders.

Quach et al. (2025) used a well-known visual XAI method called Gradient-weighted Class Activation Mapping (Grad-CAM) to interpret and assess Bayesian Optimization (BO) behavior when tuning deep learning models. Grad-CAM visually highlights which regions of input data most influence the optimization process, thereby making BO's hyperparameter search more transparent. The technique illustrates via heatmaps how various hyperparameter settings (e.g., learning rates and batch sizes) affect feature importance in the trained neural network and enhance users' understanding of why BO favors one configuration over others. The method instills confidence in BO's judgment as well as serves as a diagnostic tool for failures in optimization (like convergence problems or biased sampling) by correlating hyperparameter choices with model behavior. The research showed that interpretability through XAI-BO is not at the expense of optimization performance, thus furthering the applicability of black-box Bayesian methods to yield actionable insights for deep learning practitioners.

2.2. Specialized XAI methods for optimization

Optimization problems, such as high-dimensional search spaces and dynamic objectives, demand specialized XAI approaches. XAI methods such as search space visualization, attention-based trajectory analysis, and symbolic rule extraction provide targeted insights beyond generic interpretability tools. This section explores XAI methods designed explicitly for the unique demands of optimization.

Lotif (2014) developed a visual XAI approach called Self-Organizing Maps (SOMs), which provide interpretability for metaheuristic search dynamics in multi-objective optimization. SOMs map fitness landscapes and track the evolution of the solution, making the optimization process more transparent (Pareto fronts, population distributions, and search trajectories). SOMs clarify why algorithms like NSGA-II succeed or struggle in specific scenarios. SOMs show exploitation trade-offs, convergence patterns, and biases and provide actionable insights for algorithm design and parameter tuning. SOM helps users trust and refine metaheuristics by transforming opaque search processes into interpretable visual processes.

Hein et al. (2017) employed the Particle Swarm Optimization (PSO) method to develop transparent fuzzy rule-based policies for Reinforcement Learning (RL). The objective was to address the non-transparent aspect of traditional deep RL techniques by optimizing the fuzzy rule parameters through PSO. The method produces human-interpretable IF-THEN rules that clearly relate states to actions so that they can be audited and understood by users. The evolution of the policy is explained by visualizing the optimization path in the PSO, and interpretability metrics (e.g., number of rules, overlap analysis) are developed to balance transparency with performance. This therefore opens the road to using metaheuristics for essentially enshrining explainability in RL through co-optimization of interpretable structures with their performances.

Fyvie et al. (2025) developed an XAI method for population-based metaheuristics such as genetic algorithms. The method investigates the search trajectory of metaheuristics using Principal Component Analysis (PCA). It projects the multi-dimensional space of a candidate solution into lower-dimensional subspaces that extract features to explain algorithm behavior, such as changes in population diversity and variable importance. This comparison provided the establishment of PCA-derived measures (angle-based) contrasting with conventional entropic divergence to demonstrate how trajectory mining can recognize important learning steps and problem structures. In addition, surrogate models were used to validate the variable importance ranking against PCA results, showing good agreement. The study illustrated how such techniques can fill the gap between black-box optimization and explainable

AI. Users would understand and be able to have trust in solutions generated by metaheuristics.

Beeramoole et al. (2022) designed a multi-objective optimization framework to estimate discrete choice models by considering multiple objectives such as predictive accuracy and interpretability. The proposed framework enhances the transparency of the model by optimizing not only the statistical performance of the outcome but also the simplicity and explainability of the models involved. This ensures that decision-makers understand and trust the outputs. The whole multi-objective optimization scheme, often based on evolutionary algorithms, generates a Pareto front of solutions in which complexity is traded off against accuracy, thus allowing the user to select models that are capable of being both high performing and interpretable. Notable contributions to XAI in this research include interpretation metric integration (such as model sparsity or rule-based clarity) as optimization objectives and the trade-off visualization to help users make well-informed decisions; this approach fills the gap that is left between black-box optimization and explainable decision-making. Such understanding of choice behavior is one of the primary factors that makes these fields, especially in the context of both transportation and market research, different from many others.

Fitness landscapes refer to the graphical representations of the population of candidate solutions in POAs (Zou et al., 2022). These are the maps in an optimization problem of all the feasible solutions with their respective fitness values in the search domain. Prager and Trautmann (Prager & Trautmann, 2024) implemented an analytical Python library, Pflacco, to conduct fitness landscape analysis. In Pflacco, peaks, valleys, and plateaus characterize the optima and challenges of algorithms. Pflacco is an approach to numerically describing landscapes (modality, ruggedness, gradient homogeneity, etc.) to assess problem difficulty and metaheuristic behavior using these landscapes. In this way, raw fitness landscapes are transformed into interpretable metrics to help select the appropriate algorithm, best parameter settings, and performance prediction. The tool integrates with optimization frameworks, aligning problem structure and algorithmic response and improving explainability in evolutionary computation.

Liefoghe et al. (2021) investigated how fitness landscape analysis can predict the performance of multi-objective optimization algorithms (e.g., NSGA-II, MOEA/D) on interpolated continuous problems. The authors performed the extraction of landscape indicators (modality, ruggedness, gradient behavior, etc.) targeting the development of a machine learning-based recommendation system that can automatically decide on the best-performing algorithm according to the problem at topic. The study shows that landscape features have a strong correlation with algorithm performance, which indeed supports the applicability of data-driven algorithm selection to enhance optimization efficiency. Under this new XAI, interpretable landscape metrics justify why some algorithms perform better than others, thus improving the automation and transparency of multi-objective optimization.

Ancestry Trees is a graph-based representation that traces the lineage of solutions in evolutionary algorithms across generations, where nodes represent candidate solutions and edges denote genetic operations (e.g., crossover, mutation). Črepinšek et al. (2011) suggested generating ancestry trees for evolutionary algorithms and then analyzing them to quantify the trade-off between exploration search and exploitation search through metrics like branching depth (indicating exploration) and convergence clusters (reflecting exploitation). The work focused on how ancestry trees give interpretability into the dynamics of evolutionary algorithms, revealing how selection pressure and genetic operators affect search behavior. This helps bridge computational evolution with interpretability, enabling users to independently diagnose the performance of an algorithm and subsequently adjust parameters with full awareness.

ExTea (Huang et al., 2024) is an XAI model inspired by evolutionary algorithms such as the genetic algorithms. It aims at enhancing interpretability in black-box time series. It works by modeling

explanations as evolving individuals within a pyramidal optimization structure. The approach effectively overcomes important shortcomings of the current time series XAI methods (SBXAI, MCXAI, TS-MULE) by employing a three-tier evolutionary process where candidate explanations progress through the newborn, tested, and elite stages, with the fitness function incorporating time series-specific characteristics to ensure temporally meaningful interpretations. ExTea was evaluated on six benchmark datasets utilizing four distinct target models, where ExTea achieved stronger performance than the state-of-the-art time-series XAI methods. This was specially noted on the generation of multiple, optimized explanations while preserving the fidelity of the target model and automatically fine-tuning hyperparameters associated with explanations. Overall, ExTea not only improves transparency in complex temporal models but does so far more effectively than any previous method, without sacrificing predictive performance.

IOH-Xplainer (van Stein et al., 2024) is an XAI method for iterative optimization heuristics (IOHs). IOH-Xplainer addresses the shortcomings of traditional performance comparisons that often lack interpretability. It uses landscape analysis and XAI techniques to decompose algorithm performance into interpretable components, such as problem landscape characteristics (e.g., modality, ruggedness) and optimization dynamics (e.g., convergence behavior, exploration-exploitation trade-offs). IOH-Xplainer uses both visual analytics (such as trajectory plots and feature importance rankings) and surrogate modelling to link algorithm success or failure to specific problem features or heuristic design choices. This leads to insights beyond aggregate metrics (e.g., runtime, quality of solution), which facilitates actionable enhancements in algorithmic development. Experiments on both synthetic and real-world benchmark problems show that IOH-Xplainer identifies the strengths/weaknesses of popular heuristics (e.g., genetic algorithms) across different problem conditions, filling the gap between empirical evaluation and explainability aspects in the optimization context.

Recently, several research studies in POAs have proposed interesting hybrid optimization algorithms for solving real-world optimization fields. For example, orthogonal learning was introduced to enhance Brain Storm Optimization (Ma et al., 2020), and localized decision variable analysis has been used for large-scale multi-objective problems (Ma et al., 2021). Other research studies have explored incentive mechanisms in federated learning (Zeng et al., 2022), hybrid genetic programming for ontology alignment (Xue et al., 2025), and adaptive reference vector strategies for many-objective optimization (Li et al., 2024). All these methods focus on improving algorithms or adapting to specific areas, while EvoMapX instead aims to clearly explain how algorithms work, how operators behave, and what leads to their success.

Some existing frameworks provide partial interpretability of optimization behavior. For example, behavioral landscape analysis (BLA) (Nikolij et al., 2025) characterizes the structure of search trajectories or algorithm footprints in terms of performance regions or attractor types. Although these methods provide useful insights into global dynamics, they do not trace operator-level impact, parent-offspring lineage, or convergence phases at the resolution provided by EvoMapX. Unlike purely statistical footprinting (Arrieta et al., 2020), EvoMapX builds interpretable structures (OAM, PEG, CDS) tied to operators and fitness transitions. This enables fine-grained attribution and post-hoc textual generation. A formal comparison with such methods is a promising avenue for future research.

The XAI methods listed in Table 1 mainly rely on post hoc interpretability tools such as SHAP, LIME, Grad-CAM, or decision trees to analyze surrogate models or optimization outputs. In contrast, EvoMapX provides an intrinsic explanation mechanism that operates on the basis of the optimization process itself. Instead of retrospectively interpreting the results, EvoMapX continuously records the internal dynamics of metaheuristic algorithms through the construction of OAMs and PEGs. These tools enable real-time tracking of operator influence, exploration-exploitation behavior, and convergence trends via dedicated metrics like CDS. Therefore, EvoMapX differs fundamentally from the methods listed

in Table 1 by offering algorithm-specific, process-level causal explanations that go beyond the importance of features or the transparency of models.

2.3. Research gaps

Although there was a significant progress in XAI for optimization, there are critical limitations on the related work (Table 1):

- Most current approaches (e.g., SHAP Lundberg & Lee, 2017, LIME Ribeiro et al., 2016) provide retrospective analyses rather than real-time insights into optimization dynamics. This means that there is a significant limitation in the utility of these methods for adaptive algorithm control.
- The existing XAI methods in general rely on surrogate models or rule extraction. This means that they separate interpretability from the optimization process.
- The visualization-based interpretability techniques (e.g., fitness landscape analysis Prager & Trautmann, 2024) are unsuitable for high-dimensional spaces. In addition, feature importance methods face computational expensive.
- Few XAI approaches quantify how specific search operators (e.g., crossover strategies) control convergence patterns. This represents a critical matter because convergence behaviour plays a critical role in metaheuristic performance.
- Current XAI methods typically provide single-point explanations rather than capturing the evolutionary dynamics of optimization processes across generations.
- The XAI field lacks standardized metrics for assessing explanation quality in optimization contexts.

2.4. Summary of related work

POAs have been extensively studied, with varying levels of emphasis on interpretability and performance. Table 1 presents a comparative summary of representative approaches. Notably, most prior work emphasizes performance improvement through algorithmic modifications (e.g., adaptive mechanisms, hybrid strategies) rather than interpreting how existing algorithms behave. While these studies often report strong benchmark results, they rarely provide insights into the internal search dynamics, operator roles, or failure modes during optimization.

For example, behavioral landscape analysis and trajectory network methods provide valuable qualitative insights but often lack quantitative structures for operator-level attribution. Other works focus on sensitivity analysis or convergence trends but do not generalize across algorithms or include structured visual diagnostics. Moreover, few of these methods support real-time analysis or decision-level explainability.

In contrast, EvoMapX is designed to offer algorithm-agnostic interpretability without altering the algorithm's behavior. It includes interpretable structures and metrics (OAM, PEG, and CDS) that are scalable and informative. Although EvoMapX does not aim to directly improve optimization performance, it supports performance diagnosis, parameter tuning, and comparative analysis. Therefore, it fills an underexplored gap in the literature.

3. Background information

This section is in two parts. The first part describes popular XAI methods, while the second part explores popular POAs (Genetic algorithms, Differential evolution, particle swarm optimization, and Cuckoo search) employed for explainability assessment in the experiments.

3.1. Popular XAI methods

3.1.1. SHAP

SHAP (SHapley Additive exPlanations) (Lundberg & Lee, 2017) is a popular XAI method that explains the outputs of a machine

Table 1
Comparison of XAI methods applied to optimization tasks.

Study	Optimization Method	XAI Technique(s)	Main Contribution
General XAI Methods in Optimization			
Zhou et al. (2024)	Genetic Algorithms, Swarm Intelligence	SHAP, LIME, Fitness Landscapes, Rule Extraction	Framework combining surrogate modeling and rule extraction to improve interpretability
Wang and Chen (2024)	Genetic Algorithm	SHAP, LIME, Decision Trees, Counterfactuals	Feature importance and convergence visualization to explain GA for scheduling
Dubey (2024)	Evolutionary Algorithms	SHAP, LIME, Decision Trees	Feature sensitivity and rule-based interpretation of Pareto-optimal trade-offs
Parkavi et al. (2024)	Chaotic PSO	SHAP, LIME	Enhanced transparency in educational recommendations using XAI + optimization
Quach et al. (2025)	Bayesian Optimization	Grad-CAM	Heatmap visualization to show hyperparameter impacts on model behavior
Specialized XAI Methods for Optimization			
Lotif (2014)	NSGA-II, MOEAs	Self-Organizing Maps (SOMs)	Visualized Pareto fronts and search dynamics
Hein et al. (2017)	PSO	Fuzzy Rule Extraction	Co-optimized fuzzy rule sets with interpretable RL behavior
Fyvie et al. (2025)	Genetic Algorithms	PCA Trajectory Analysis, Surrogate Models	Learning behavior explained via PCA-based trajectory mining
Beeramoole et al. (2022)	Evolutionary Algorithms	Multi-objective Explanation Trade-offs	Pareto-based balancing of model accuracy and interpretability
Prager and Trautmann (2024)	Evolutionary Methods	Fitness Landscape Metrics	Landscape-informed algorithm guidance using 'pflacco' metrics
Liefooghe et al. (2021)	NSGA-II, MOEA/D	Landscape Indicators	Predictive performance modeling via landscape-derived features
Črepinšek et al. (2011)	Evolutionary Algorithms	Ancestry Trees	Traced solution lineages to analyze exploration vs. exploitation
Huang et al. (2024)	Genetic Algorithm-inspired	Evolutionary Explanation Generator	Optimized time-series explanations via evolutionary operators
van Stein et al. (2024)	Iterative Heuristics	Surrogate Models, Visual Analytics	Explained optimizer behavior via decomposition and visual inspection

learning model by computing the contribution of each feature using concepts from cooperative game theory, specifically the Shapley value. For a model $f(x)$, the SHAP value ϕ_i for the feature i is:

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|! (|F| - |S| - 1)!}{|F|!} [f(S \cup \{i\}) - f(S)]$$

where F is the set of all features, S is a subset of features not containing i , and $f(S)$ denotes the model output when only features in S are present.

SHAP fairly distributes the “payout” (prediction) among features based on their marginal contributions across all possible feature coalitions.

3.1.2. LIME

LIME (Local Interpretable Model-Agnostic Explanations) (Ribeiro et al., 2016) is a well-known model-agnostic XAI method that explains a model's prediction by approximating it locally around a specific instance x with a simple, interpretable model $g(x)$, typically a linear regression. Formally, LIME solves:

$$\arg \min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g)$$

where f is the original complex model, g is an interpretable model from a class G (e.g., sparse linear models), π_x is a proximity measure between x and other samples, $\mathcal{L}$ is a loss function measuring how closely g approximates f near x , and $\Omega(g)$ is a regularization term to ensure interpretability. LIME perturbs the input x , observes the output changes, and fits an easy-to-understand surrogate model around x .

3.1.3. Grad-CAM

Grad-CAM (Gradient-weighted Class Activation Mapping) (Selvaraju et al., 2020) is a visual explanation method designed for convolutional neural networks (CNNs). It provides visual explanation by highlighting the regions of an input image that highly affected the decision process in CNNs. For an arbitrary target class score y^c , Grad-CAM computes the gradient of y^c with respect to the feature maps A^k of a convolutional layer:

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}$$

where α_k^c is the importance weight for feature map k , and Z is a normalization factor (usually the number of pixels in A^k).

The final Grad-CAM heatmap $L_{\text{Grad-CAM}}^c$ is:

$$L_{\text{Grad-CAM}}^c = \text{ReLU} \left(\sum_k \alpha_k^c A^k \right)$$

Grad-CAM makes coarse localization maps that indicate which regions of interest are located in the input with the gradients of the target output flowing into the last convolutional layer for that output.

3.2. Population-based optimization algorithms

This section discusses four popular POAs (Genetic Algorithm, Differential Evolution, Cuckoo Search, Particle Swarm Optimization), with a focus on their core operators. For clarity, the primary operators of each algorithm are highlighted in **bold** throughout the discussion.

3.2.1. Genetic algorithm

The Genetic Algorithm (GA) (Katoch et al., 2021) is a POA inspired by the principles of natural selection and genetics. It iteratively optimizes a population of candidate solutions mainly using four operations: selection, crossover, mutation, and replacement operations.

In each generation t , the GA algorithm goes through four main phases (**Selection**, **Crossover**, **Mutation**, and **Replacement**). In the **Selection** phase of GA, the parent solutions are selected according to a selection probability based on fitness values. The probability of selecting a solution increases with an increase in the quality of the fitness value. In the **Crossover** phase, the genes of the parent pairs are joined to create their offspring. For example, in single-point crossover:

$$\mathbf{x}_k = \text{Crossover}(\mathbf{x}_i, \mathbf{x}_j) \quad (1)$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are two parent candidate solutions selected randomly from the population of solutions, and $\mathbf{x}_k$ is the child candidate solution.

In the **Mutation** phase, small random changes are applied to offspring solutions:

$$\mathbf{x}'_k = \text{Mutate}(\mathbf{x}_k) \quad (2)$$

Finally, in the **Replacement** phase, the new population is formed by selecting among parents and offspring.

GA mimics variable evolutionary pressures and selects superior solutions. A key success factor for GA is that it maintains diversification through crossover and mutation.

3.2.2. Differential evolution

Differential Evolution (DE) (Chakraborty et al., 2023) is a simple but powerful evolutionary algorithm that optimizes a population using vector differences for perturbation.

Given a target candidate solution $\mathbf{x}'_t$ at generation t , DE creates a potentially improved candidate solution $\mathbf{u}'_t$ through three operators.

First, using the **Mutation** operator, a mutant candidate solution is generated by perturbing target candidate solutions from the population (basic DE/rand/1 strategy):

$$\mathbf{v}'_i = \mathbf{x}'_{r1} + F(\mathbf{x}'_{r2} - \mathbf{x}'_{r3}), \quad (3)$$

where $r1, r2$, and $r3$ are random and distinct indices, and $F > 0$ is the scaling factor.

Second, using the **Crossover** operator, the mutant candidate solution ($\mathbf{v}'_i$) is mixed with the target candidate solution ($\mathbf{x}_i$) to create a trial candidate solution ($\mathbf{u}_i$):

$$\mathbf{u}'_{i,j} = \begin{cases} \mathbf{v}'_{i,j} & \text{if } \text{rand}_j(0, 1) \leq CR \text{ or } j = j_{\text{rand}} \\ \mathbf{x}'_{i,j} & \text{otherwise} \end{cases}, \quad (4)$$

where $CR \in [0, 1]$ is the crossover rate.

Third, using the **Selection** operator, the trial candidate solution ($\mathbf{u}_i$) is selected to replace the target candidate solution ($\mathbf{x}_i$) in the next generation, if $f(\mathbf{u}'_i) \leq f(\mathbf{x}'_i)$ for minimization problems:

$$\mathbf{x}^{t+1}_i = \begin{cases} \mathbf{u}'_i & \text{if } f(\mathbf{u}'_i) \leq f(\mathbf{x}'_i) \\ \mathbf{x}'_i & \text{otherwise} \end{cases} \quad (5)$$

DE uses the scaled difference of randomly selected population members to explore the search space effectively.

3.2.3. Particle swarm optimization

The Particle Swarm Optimization (PSO) method (Gad, 2022) is a POA technique inspired by the swarming behavior of birds flocking and fish schools. Each potential solution, termed as a particle, iteratively updates its velocity and position in the search space by taking into account its own past information as well as that of the neighbors.

At iteration t , the velocity $\mathbf{v}'_i$ and position $\mathbf{x}'_i$ of the i -th particle are updated according to:

$$\mathbf{v}^{t+1}_i = \underbrace{w \mathbf{v}^t_i}_{\text{Inertia}} + \underbrace{c_1 r_1 (\mathbf{p}_i - \mathbf{x}^t_i)}_{\text{Cognitive}} + \underbrace{c_2 r_2 (\mathbf{g} - \mathbf{x}^t_i)}_{\text{Social}} \quad (6)$$

$$\mathbf{x}^{t+1}_i = \mathbf{x}^t_i + \mathbf{v}^{t+1}_i \quad (7)$$

where $w \in [0, 1]$ is the inertia weight, controlling the influence of the previous velocity on the current motion, c_1 and $c_2 > 0$ are acceleration coefficients determining the strength of attraction toward the personal and global best positions, r_1 and r_2 are random numbers between 0 and 1, $\mathbf{p}_i$ is the cognitive best (i.e., the best position discovered by particle i so far), and $\mathbf{g}$ is the social best (i.e., the best position found by the entire swarm).

The three components in the PSO velocity update equation influence the particle's movements in PSO. The **inertia component** ($w \mathbf{v}^t_i$) preserves a fraction of the previous velocity, encouraging continued exploration in the current direction and helping maintain momentum across iterations. The **cognitive component** ($c_1 r_1 (\mathbf{p}_i - \mathbf{x}^t_i)$) directs the particle movement in the direction of its own historically best-known position. This aims to stimulate self-adaptive behavior and individual learning. In contrast, the **social component** ($c_2 r_2 (\mathbf{g} - \mathbf{x}^t_i)$) guides the particle movement toward the best global solution. This enables information sharing and collective intelligence. These three components work simultaneously to balance global exploration with local exploitation. This makes PSO a flexible and efficient optimizer across a variety of complex landscapes.

The PSO technique can tackle a large number of complex optimization problems because the three components of its velocity update equation allow efficient exploration and exploitation of the search space. However, the optimal values for w, c_1, c_2 must be carefully determined to achieve optimal convergence behavior.

3.2.4. Cuckoo search

The Cuckoo Search (CS) algorithm, as described by Guerrero-Luis et al. (2021), is a POA inspired by the deceptive parasitic lifestyle of certain cuckoo species. These cuckoos lay their eggs into the nests of other host birds. CS is an amalgamation of this biological metaphor and a global search strategy based on Lévy flights, which work effectively to appropriately balance exploration with exploitation.

The core of CS is the generation of new candidate solutions (nests) through **Lévy flights**, which are random walks characterized by heavy-tailed step size distributions. The position of a new solution $\mathbf{x}^{t+1}_i$ is typically updated from the current solution $\mathbf{x}^t_i$ as:

$$\mathbf{x}^{t+1}_i = \mathbf{x}^t_i + \alpha \otimes \text{Lévy}(\lambda), \quad (8)$$

where $\alpha > 0$ is the step size scaling factor, and $\text{Lévy}(\lambda)$ denotes a step vector sampled from a Lévy distribution with index $\lambda \in (1, 3]$. The mechanism, therefore, allows CS to make long jumps in distance which enable effective exploration of search space and allows it to escape from local optima.

A newly generated solution replaces an existing one if it yields a better fitness value, modeling a **successful replacement** of a host egg by a cuckoo's egg. In contrast, if the new solution does not improve the existing one, it represents an **unsuccessful attempt**, and the original solution is retained. To simulate the host bird's ability to discover and abandon alien eggs, a fraction $p_a \in [0, 1]$ of the worst nests is replaced by randomly generated solutions at each iteration. This **abandoned nest** strategy introduces fresh genetic material into the population, enhances diversity, and prevents premature convergence.

Random initialization is used at the beginning of the algorithm to generate the initial population of nests uniformly across the search space and ensures unbiased sampling of the solution domain. The balance between directed Lévy-based updates and stochastic nest replacement makes CS converge robustly across different optimization problems.

4. Methodology: The EvoMapX framework

EvoMapX is a novel XAI approach for analyzing and interpreting the behavior of population-based metaheuristic algorithms. It works

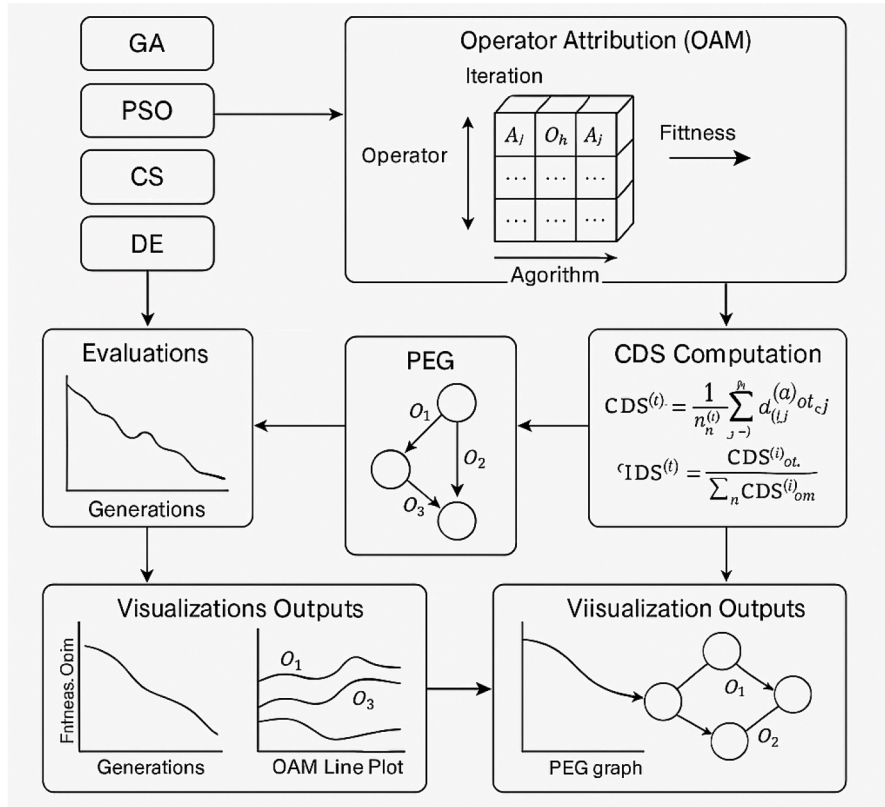


Fig. 1. Overview of the EvoMapX framework. The process begins with logging operator-level events from population-based optimization algorithms. These are used to construct three interpretable structures: OAM, PEG, and CDS. These outputs enable operator-level attribution, convergence tracing, and narrative generation to explain algorithm behavior.

by extracting, attributing, and visualizing the contributions of algorithmic operators to search progression over time. EvoMapX includes three explainable components: the Operator Attribution Matrix (OAM), the Population Evolution Graph (PEG), and the Convergence Driver Score (CDS). The diagram in Fig. 1 illustrates the analysis of multiple meta-heuristic algorithms (e.g., GA, PSO, CS, DE), the construction of PEG, OAM, and CDS. EvoMapX enables interpretability by capturing the contribution of each operator across algorithms to the convergence process.

Together, the components of EvoMapX help dissect the internal dynamics of optimization algorithms and answer questions such as: “Which operators contribute most to convergence?”, “How do strategies interact over time?”, and “What characterizes stagnation phases?”

EvoMapX (Fig. 2) does not affect the POA algorithm’s optimization process. It works in an independent layer recording operator event, fitness changes, and parent-child relationships without influencing algorithm execution or operations. The recorded data is stored externally and processed post hoc. Thus, EvoMapX does not create a feedback loop that could alter algorithm behavior.

4.1. Operator attribution matrix (OAM)

The OAM is a structured record of the impact of each search operator over iterations. For each operator O_k applied at generation t , the matrix records the associated fitness gain, defined as:

$$\Delta f_{O_k}^{(t)} = f(\mathbf{x}_{\text{parent}}^{(t)}) - f(\mathbf{x}_{\text{child}}^{(t)}) \quad (9)$$

where $\mathbf{x}_{\text{parent}}^{(t)}$ is the input solution (parent solution) to operator O_k at generation t , and $\mathbf{x}_{\text{child}}^{(t)}$ is the output solution (child solution) produced by O_k . The fitness change $\Delta f_{O_k}^{(t)}$ measures the improvement in objective value due to O_k , where a positive Δf indicates a fitness improvement, while a negative Δf indicates a fitness deterioration.

This value is stored in the OAM for each operator, algorithm, and generation, resulting in a 3D structure indexed by algorithm A_i , operator O_k , and iteration t :

$$\text{OAM}_{A_i, O_k}^{(t)} = \left\{ \Delta f_{O_k, j}^{(t)} \right\}_{j=1}^{n_{\text{applications}}} \quad (10)$$

4.2. Population evolution graph (PEG)

The PEG is a directed graph $G = (V, E)$, where nodes V represent individuals in the population and edges E capture parent-child relationships. Each edge is annotated with the operator responsible for the transition:

$$E_{ij} = (\mathbf{x}_i \xrightarrow{O_k} \mathbf{x}_j) \quad (11)$$

This graph provides a structural view of the population’s lineage and evolutionary flow, making it possible to trace how specific operators or individuals contribute to convergence.

The structure of PEG may appear similar to Search Trajectory Networks (STNs) (Ochoa et al., 2021); however, it has a different purpose, and it is constructed in a different way. STNs typically model transitions between solution states in the decision or objective space. It captures how a single agent or algorithm moves through the landscape. In contrast, PEGs in EvoMapX are explicitly designed to trace the *lineage of individuals* within a population over time. Each edge in the PEG represents a parent-offspring relationship, and nodes retain their fitness and operator attribution. This allows the PEG to encode not only the ancestry of solutions but also the influence of operators on inherited fitness improvements. This means that PEGs are more useful for studying how groups of solutions come together and how different operators affect them in population-based algorithms, instead of tracking the specific path of one search.

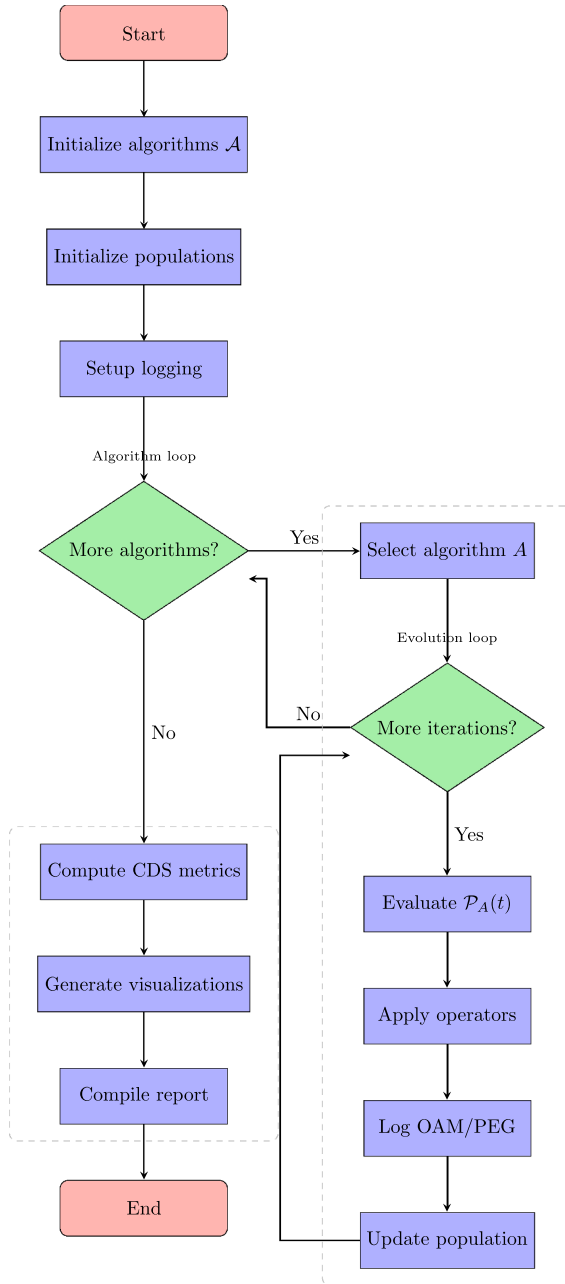


Fig. 2. Flowchart of the EvoMapX framework. The process begins with the execution of a population-based optimization algorithm, where operator applications, fitness changes, and lineage data are logged. These logs are processed to generate three interpretable structures: OAM, PEG, CDS. These components provide insights into operator behavior, convergence dynamics, and algorithm performance over time.

4.3. Convergence driver score (CDS)

To quantitatively assess the relative impact of each operator or algorithm on convergence, EvoMapX introduces the Convergence Driver Score (CDS). For a given operator $\mathcal{O}_k$ at iteration t , the CDS is defined as:

$$CDS_{\mathcal{O}_k}^{(t)} = \frac{1}{n_{\mathcal{O}_k}^{(t)}} \sum_{j=1}^{n_{\mathcal{O}_k}^{(t)}} \Delta f_{\mathcal{O}_k,j}^{(t)} \quad (12)$$

where $n_{\mathcal{O}_k}^{(t)}$ is the number of times operator $\mathcal{O}_k$ was applied at iteration t , and $\Delta f_{\mathcal{O}_k,j}^{(t)}$ is the fitness gain of the j -th application.

To aggregate this across iterations for a global perspective:

$$CDS_{\mathcal{O}_k}^{\text{total}} = \sum_{t=1}^T CDS_{\mathcal{O}_k}^{(t)} \quad (13)$$

CDS values can also be normalized within each algorithm to highlight relative importance:

$$\widetilde{CDS}_{\mathcal{O}_k}^{(t)} = \frac{CDS_{\mathcal{O}_k}^{(t)}}{\sum_{\mathcal{O}_m \in \mathcal{A}} CDS_{\mathcal{O}_m}^{(t)}} \quad (14)$$

This normalization enables operator comparisons within an algorithm and is especially useful for generating rule-based textual explanations. In our heuristics, an operator is considered *dominant* in a generation if its normalized CDS exceeds 40% of the total. This threshold is empirically defined and used only for narrative purposes; it does not affect algorithm execution.

4.3.1. Interpretation of CDS values and thresholding

Experimentally, a high CDS for an operator $\mathcal{O}_k$ at iteration t means that $\mathcal{O}_k$ made a significant contribution to the total fitness improvement during that generation. In contrast, a low CDS value implies that the operator had minimal impact on convergence.

We use the normalized CDS metric to improve interpretability $\widetilde{CDS}_{\mathcal{O}_k}^{(t)}$ and apply a threshold rule to determine operator dominance. Specifically, we consider an operator *dominant* in iteration t if:

$$\widetilde{CDS}_{\mathcal{O}_k}^{(t)} > 0.4 \quad (15)$$

This 40% threshold was selected heuristically based on empirical analysis. It provides a balance between sensitivity and specificity. For example, lower thresholds (e.g., 20%) may over-report dominance of weak operators, while higher thresholds (e.g., 60%) may ignore impactful but moderately used operators. This rule enables interpretable insights such as “Mutation was the dominant contributor to fitness improvement in 7 out of the first 10 generations.”

A detailed numerical example demonstrating how CDS, OAM, and PEG values are computed and interpreted has been added in Section 4.5.

4.4. The EvoMapX algorithm

Algorithm 1 covers the algorithmic details of the EvoMapX. The process starts by accepting a collection of optimization algorithms $\mathcal{A}$ (such as GA, PSO, CS, and DE), one targeted objective function, $f(\mathbf{x})$, a specific population size, N , and a maximum number of iterations, T . In the initialization phase of Algorithm 1, EvoMapX goes through each algorithm $A \in \mathcal{A}$, to generate an initial population $\mathcal{P}_A(0)$ and to set up logging infrastructure tailored for explainability. This includes data structures for the OAM, PEG, and additional performance and traceability metrics.

The main loop follows a two-level structure, where each algorithm is executed independently over T iterations. In each iteration, EvoMapX first assesses the fitness of all individuals in the population. Then, it applies the respective algorithm-specific operators (e.g., crossover, mutation, velocity updates, Lévy flights) to produce new candidate solutions. Importantly, each operator’s influence on the offspring is recorded and attributed in the OAM, which serves as a temporal heatmap of operator usage and impact. The population is then updated, and the PEG is extended with directed links that trace lineage from parent to offspring, annotated with operator labels.

After complete execution of all algorithms, EvoMapX combines all operator data into comprehensive OAMs and constructs the complete PEG to visualize the structural evolution of each population. It further computes the CDS for all operators based on their contributions to convergence over time.

Finally, EvoMapX produces a complete set of visualizations such as convergence curves, operator impact plots, and graphical visualizations

with interpretable text explanations alongside. All artifacts are contributed to the same repository in a structured PDF report for easy reference and analysis as well as reproducibility. The end result will then be an explanation rich in content, entities, and layers describing how each algorithm behaved and evolved in the course of the optimization process, furnishing the necessary transparency into the exact mechanics behind population-based search.

Algorithm 1 EvoMapX: Explainable metaheuristic optimization framework.

Require: Set of algorithms $\mathcal{A} = \{\text{GA}, \text{PSO}, \text{CS}, \text{DE}, \dots\}$; objective function $f(x)$; population size N ; maximum iterations T

Ensure: Convergence plots, Operator Attribution Matrix (OAM), Population Evolution Graph (PEG), CDS metrics, PDF report

- 1: **Initialize** objective function $f(x)$ and algorithm parameters
- 2: **for all** algorithm $A \in \mathcal{A}$ **do**
- 3: Initialize population $\mathcal{P}_A(0)$ of size N
- 4: Initialize logging structures: OAM, PEG, convergence logs, operator labels
- 5: **end for**
- 6: **for all** algorithm $A \in \mathcal{A}$ **do**
- 7: **for** iteration $t = 1$ to T **do**
- 8: Evaluate fitness of individuals in $\mathcal{P}_A(t)$
- 9: Apply algorithm-specific operators to generate offspring
- 10: Record which operator(s) influenced each individual
- 11: Update population: $\mathcal{P}_A(t+1) \leftarrow$ selected offspring
- 12: Log convergence data and operator usage (OAM)
- 13: Update PEG with parent-offspring links and operator labels
- 14: **end for**
- 15: **end for**
- 16: Construct OAM for all algorithms
- 17: Generate PEG with operator annotations
- 18: Compute CDS based on operator contributions
- 19: Generate visual outputs: convergence plots, OAM heatmaps, PEG diagrams
- 20: Embed textual explanations and trend analysis in figures
- 21: Compile complete PDF report with all visualizations and diagnostics
- 22: **return** Final EvoMapX report and logs

4.5. Illustrative example of OAM, PEG, and CDS

To improve clarity and illustrate the explainability outputs of EvoMapX, we present a simplified toy example using a 1D minimization problem: $f(x) = x^2$. We simulate a short run of a GA with 3 individuals at Generation 0: $x_1 = -2$, $x_2 = 1$, and $x_3 = 3$.

In this example, two genetic operators were applied during the next generation. First, **Crossover** was applied to x_1 and x_2 to produce the offspring $x_4 = -0.5$. Then, **Mutation** was applied to x_3 to produce the offspring $x_5 = 2.7$.

OAM tracks the fitness gain Δf associated with each operator $\mathcal{O}_k$ as:

$$\Delta f_{\text{crossover}}^{(1)} = f(x_1) - f(x_4) = 4 - 0.25 = 3.75$$

$$\Delta f_{\text{mutation}}^{(1)} = f(x_3) - f(x_5) = 9 - 7.29 = 1.71$$

These values are recorded under Generation 1 in the OAM, attributing credit to the respective operators.

PEG visualizes the parent-offspring lineage, with directed edges representing which operator produced which individual:

- $x_1 \rightarrow x_4$ and $x_2 \rightarrow x_4$ (via crossover)
- $x_3 \rightarrow x_5$ (via mutation)

Each edge is labeled by the responsible operator, and nodes retain their fitness values.

CDS computes the average fitness gain per operator per generation. In this case:

$$\text{CDS}_{\text{crossover}}^{(1)} = \frac{3.75}{1} = 3.75, \quad \text{CDS}_{\text{mutation}}^{(1)} = \frac{1.71}{1} = 1.71$$

These values quantify how much each operator contributed to convergence during this generation.

Fig. 3 provides a complete visualization of this example. It shows the OAM matrix, PEG lineage graph, and CDS bar chart. This toy example demonstrates how EvoMapX enables operator-level transparency in evolutionary runs.

5. Experimental results

5.1. Parameter settings

5.1.1. Optimization operator table in EvoMapX

The internal dynamics of metaheuristic optimization algorithms should be clearer and more interpretable. EvoMapX makes this possible by tracking the contributions of different optimization operators, for example, mutation, crossover, or change in velocity, over the course of optimization. This claim is further strengthened by Table 2, which formally defines the primary operators of four popular metaheuristic optimization algorithms (GA, PSO, CS, and DE). Each algorithm is listed alongside its primary operators, with concise descriptions of their roles and behavioral characteristics. For example, PSO includes components such as cognitive, social and inertia terms, while DE includes a range of mutation strategies such as rand/1 and current-to-best/1. The identification of the primary operators in each tested algorithm is essential for constructing the OAM, which records operator usage over iterations, and the PEG, which visually maps how operators influence population dynamics.

Furthermore, Table 2 enhances the explainability aspect of EvoMapX by bridging the gap between algorithmic actions and human understanding. It allows users (researchers, practitioners, or students) to interpret visualizations and logs in the context of familiar operator

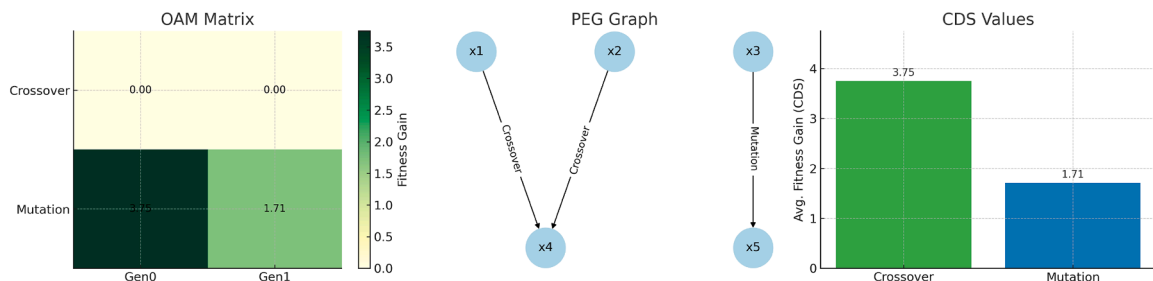


Fig. 3. Toy example illustrating how EvoMapX tracks operator impact. **Left:** The OAM records fitness gains for each operator across generations. **Middle:** The PEG encodes parent-offspring relationships and operator lineage. **Right:** CDS quantifies the average contribution of each operator to convergence at a given generation.

Table 2
Optimization operators and tracked metrics in EvoMapX.

Algorithm	Main Operator	Definition	Tracked Metrics (Equation Ref.)
GA	Crossover	Combines genetic material using two-point crossover.	OAM (Eqs. 9–10), PEG (Eq. 11), CDS (Eqs. 12–14)
	Mutation	Non-uniform mutation with adaptive rate ($0.1 \rightarrow 0.01$).	OAM (Eqs. 9–10), PEG (Eq. 11), CDS (Eqs. 12–14)
	Cognitive Component	Moves toward the particle's personal best position.	OAM (Eqs. 9–10), PEG (Eq. 11), CDS (Eqs. 12–14)
PSO	Social Component	Moves toward the swarm's global best position.	OAM (Eqs. 9–10), PEG (Eq. 11), CDS (Eqs. 12–14)
	Inertia Component	Maintains particle's current velocity.	OAM (Eqs. 9–10), PEG (Eq. 11), CDS (Eqs. 12–14)
	Position Update	Combines all components to compute velocity and position.	OAM (Eqs. 9–10), PEG (Eq. 11), CDS (Eqs. 12–14)
DE	Mutation	Applies rand/1, best/1, current-to-best/1, rand/2 with $F \in [0.2, 1.0]$.	OAM (Eqs. 9–10), PEG (Eq. 11), CDS (Eqs. 12–14)
	Crossover	Binomial (90 %) or exponential (10 %) with $CR \in [0.5, 0.9]$.	OAM (Eqs. 9–10), PEG (Eq. 11), CDS (Eqs. 12–14)
	Lévy Flight	Lévy-distributed exploration steps with adaptive scale.	OAM (Eqs. 9–10), PEG (Eq. 11), CDS (Eqs. 12–14)
CS	Successful Replacement	Keeps offspring if better than parent.	OAM (Eqs. 9–10), PEG (Eq. 11), CDS (Eqs. 12–14)
	Unsuccessful Attempt	Rejects offspring if worse.	PEG only
	Abandoned Nest	Discards solutions randomly ($p_a: 0.5 \rightarrow 0.1$).	OAM (Eqs. 9–10), PEG (Eq. 11), CDS (Eqs. 12–14)
	Random Init	Initializes new solutions biased toward best.	OAM (Eqs. 9–10), PEG (Eq. 11), CDS (Eqs. 12–14)

definitions. When EvoMapX generates convergence plots, operator contribution charts, or PDF summaries, this table serves as a glossary that helps decode the meaning of operator names and actions. For example, users can understand that a “Lévy Flight” in CS introduces exploration through long jumps or that “Non-uniform Mutation” in GA drives exploitation with an adaptive rate.

Finally, Table 2 supports the generation of comprehensive reports acting as a foundational element in the EvoMapX documentation layer. It can be directly inserted into the final PDF report from which any operator-related analysis can be referred. In this way, the framework's commitment to transparency and interpretability is furthered; optimization behavior can be seen not just to happen but also to be explained in a meaningful way.

5.1.2. Parameter settings of all algorithms

Table 3 outlines all parameter settings applied in four optimization algorithms (GA, DE, PSO, and CS) within the EvoMapX. The table starts with general framework settings such as the dimension of the problem, the size of the population, and the boundaries of the search space and provides the basic parameters valid for all algorithms before delving into the specifications of the algorithm. In addition, parameters and adaptive ranges for GA, DE, PSO, and CS are presented to clarify the interdependence of each algorithm with its settings. The explainability measures of EvoMapX (OAM to track the contributions of the operators, PEG to visualize evolutionary relationships, and CES to assess the impact) have also been documented with restart strategies and adaptivity methods. Table 3 provides an excellent reference that serves to provide a satisfying balance between completeness and readability for any user intending to work with the EvoMapX framework. It is important to note that no hyperparameter tuning was performed for the algorithms. Instead, parameter settings recommended in the literature were adopted to ensure fair and consistent comparisons. The references to each algorithm appear in the first column of Table 3. Moreover, EvoMapX does not modify or adapt algorithmic hyperparameters such as population size, mutation rates, or learning coefficients during execution. If an algorithm inherently includes adaptive components (e.g., mutation factor F in DE), EvoMapX passively logs their use and observed effects but does not influence or optimize them. Therefore, EvoMapX should not be viewed as a hyperheuristic or self-adaptive control system (Yates et al., 2025).

EvoMapX applies a Kamada-Kawai layout with caching for PEGs to manage complexity in large-scale visualizations. This ensures that the graph layout is interpretable over iterations. In the cases of dense graphs, users can opt to prune either by operator type or fitness threshold to examine transitions that matter. OAM data is stored inside sparse structures and thus can be aggregated by either iteration or operator to reduce memory and improve scalability.

5.2. The CEC 2021 benchmark suite

Table 4 shows a summary of the CEC 2021 test suite for single-objective optimization (Das et al., 2021): unimodal functions (F1–F2), multimodal functions (F3–F7), and composition functions (F8–F10). Each of these functions incorporates shifts and rotations to prevent algorithm bias and ensure thorough testing. The CEC 2021 test suite provides a robust and comprehensive framework for assessing the performance and capabilities of EvoMapX, ensuring that evaluations are thorough, standardized, and reflective of real-world optimization challenges.

5.3. If-then rules in EvoMapX framework

EvoMapX generates structured textual insights with respect to the performance of algorithms, effectiveness of operator, convergence behavior, and population dynamics through the implementation of a set of domain-specific *if-then* rules that govern the generation of natural language explanations.

The if-then rules proposed in this section to generate textual explanations were derived through iterative experiments through refinements of patterns observed in the CDS and OAM data across multiple test runs. Each rule links a quantitative condition (e.g., dominance of one operator over others, stagnation periods, abrupt fitness drops) with a corresponding linguistic template. The threshold values (e.g., CDS dominance $> 40\%$) were experimentally selected based on a dual approach: multiple trials and manual adjustment. This dual approach aims to avoid false positives. The if-then rules were validated by applying them to 20+ cases across GA, PSO, DE, and CS on CEC 2021 functions. This process confirmed that generated if-then rules matched known operator behavior and aligned with PEG observations.

These if-then rules are categorized as follows:

5.3.1. Performance comparison rules

- If an algorithm achieves the lowest final fitness value, **then** it is identified as the best-performing algorithm.
- If an algorithm achieves 90 % of its total improvement in fewer iterations than others, **then** it is described as having the fastest convergence.
- If an algorithm exhibits the highest improvement ratio in the first half of the iterations, **then** it is characterized by a strong initial descent.
- If an algorithm shows the highest improvement ratio in the second half of iterations, **then** it is interpreted to have superior refinement capabilities.

5.3.2. Operator contribution rules

- If an operator has the highest mean fitness improvement across iterations, **then** it is labeled as the most effective operator.

Table 3
Key parameter settings in the EvoMapX framework.

Algorithm	Parameter Description
General	Problem Dimension (DIM): 10 Population Size (POP_SIZE): 20 Max Generations (MAX_GEN): 100 Early Stopping Tolerance (ESC_TOLERANCE): 15 generations of stagnation Search Space Bounds (BOUNDS): $[-100, 100]$ Initialization Method: Latin Hypercube Sampling (LHS) Restart Strategy: Triggered based on stagnation via early stopping condition
DE (Cheng et al., 2024)	Mutation Factor (F): 0.5, adaptively ranges from $[0.2, 1.0]$ Crossover Rate (CR): 0.9, adaptively ranges from $[0.5, 0.9]$
PSO (Sekyere et al., 2024)	Inertia Weight: Linearly decreases from 0.9 to 0.4 Cognitive Coefficient (c_1): Decreases from 2.5 to 0.5 Social Coefficient (c_2): Increases from 0.5 to 2.5 Velocity Clamping: 20 % of total search space range
CS (Wei & Niu, 2022)	Abandonment Probability (p_a): Decreases from 0.5 to 0.1 Elite Preservation: Top 10 % of nests preserved Levy Flight Parameter (λ): $\lambda = 1.5$ Step Size: Adaptive step size decreases over generations
GA (Katoch et al., 2021)	Selection Method: Tournament selection ($k = 3$) Crossover Operator: Two-point crossover Mutation Rate: Adaptive: decreases from 0.1 to 0.01 Mutation Strength: Adaptive: decreases from 0.5 to 0.1 Elitism: Top 10 % of population preserved
OAM	Tracks operator-level contributions per algorithm per iteration
PEG	Graph structure capturing evolutionary transitions and relationships
CDS	Metric for evaluating operator impact on convergence trends

Table 4
CEC 2021 test functions with optimal values and boundaries.

ID	Function Name	Search Boundaries	Optimal Value	Key Properties
F1	Rotated Bent Cigar Function	$[-100, 100]^D$	$f(x^*) = 100$	Unimodal, non-separable, highly ill-conditioned with conditioning number of 10^6
F2	Rotated Discus Function	$[-100, 100]^D$	$f(x^*) = 200$	Unimodal, non-separable, with one highly sensitive dimension
F3	Shifted and Rotated Rosenbrock's Function	$[-100, 100]^D$	$f(x^*) = 300$	Multimodal, non-separable, narrow curved valley
F4	Shifted and Rotated Ackley's Function	$[-100, 100]^D$	$f(x^*) = 400$	Multimodal, non-separable, many local optima, nearly flat outer region
F5	Shifted and Rotated Rastrigin's Function	$[-100, 100]^D$	$f(x^*) = 500$	Highly multimodal with 10^D local optima, regular structure
F6	Shifted and Rotated Expanded Griewank's plus Rosenbrock's Function	$[-100, 100]^D$	$f(x^*) = 600$	Hybrid function, multimodal, complex variable interaction
F7	Shifted and Rotated Schwefel's Function	$[-100, 100]^D$	$f(x^*) = 700$	Multimodal, global optimum near boundary, deceptive local optima
F8	Shifted and Rotated Composition Function 1	$[-100, 100]^D$	$f(x^*) = 800$	Composition of multiple functions with different properties
F9	Shifted and Rotated Composition Function 2	$[-100, 100]^D$	$f(x^*) = 900$	Complex composition with asymmetric landscape, irregular local optima
F10	Shifted and Rotated Composition Function 3	$[-100, 100]^D$	$f(x^*) = 1000$	Most challenging composition, numerous local optima, narrow basins

- If comparing across algorithms, **then** operators are ranked by their average fitness contributions over time.
- If certain operators consistently produce higher fitness improvements within a specific algorithm, **then** these are highlighted as the primary drivers of the search behavior of that algorithm.

5.3.3. Convergence pattern rules

- If the improvement rate in any iteration exceeds the 90th percentile, **then** that iteration is classified as a phase of rapid improvement.
- If the improvement rate falls below a predefined threshold (e.g., 10^{-4}), **then** the iteration is identified as a stagnation point.
- If multiple consecutive iterations show subthreshold improvement, **then** they are grouped into a stagnation phase.

5.3.4. Algorithm-specific behavior rules

GA:

- If the fitness gain due to crossover exceeds that of mutation, **then** recombination is deemed more effective than random variation.
- If early-stage improvement is more than three times greater than the improvement in the late stage, **then** the algorithm is said to exhibit rapid early convergence followed by diminishing returns.

PSO:

- If the cognitive component contributes more than both the social and inertia components, **then** the swarm is more dependent on individual memory.

- If the social component is dominant, **then** it indicates effective information sharing among particles.
- If the average connectivity in the evolution graph is below 2, **then** the swarm is interpreted as following linear trajectories toward convergence.

CS:

- If Lévy flight contributions exceed those from nest abandonment, **then** local exploration is considered more effective than global resets.
- If the success rate for the acceptance of new solutions exceeds 30 %, **then** the search strategy is considered effective for the problem landscape.

DE:

- If mutation contributes more than crossover, **then** differential vector generation is considered particularly effective.
- If the convergence stability coefficient is less than 1.0, **then** the algorithm is said to achieve smooth and stable progress with a good balance between exploration and exploitation.

5.3.5. Population evolution graph (PEG) rules

- If the graph density exceeds 0.05, **then** the population structure is described as having high connectivity and robust information exchange.
- If the average number of parents per individual exceeds 1.5, **then** the population is considered to exhibit complex ancestral relationships.
- If certain individuals have significantly higher degree values, **then** they are identified as the most influential contributors to the evolution of the population.

5.3.6. Convergence driver score (CDS) rules

- If an operator has the highest normalized CDS, **then** it is identified as the primary convergence driver.
- If the ratio between the CDS of the top operator and the second highest exceeds 2, **then** the search process is interpreted as being heavily dependent on a single dominant mechanism.
- If driver diversity is compared using entropy, **then** the algorithm with the highest entropy is recognized as using the most diverse set of convergence drivers.

5.3.7. Lineage impact rules

- If an operator's descendants span many generations (high influence span), **then** its contributions have long-term effects on convergence.
- If an operator's descendants are densely clustered in later generations (high influence density), **then** it has a strong delayed impact on optimization.
- If an operator has many descendants in the PEG, **then** it plays a key role in shaping future solutions.
- If an operator's descendants consistently improve fitness (positive delayed CDS), **then** it is a sustained convergence driver.
- If an operator's descendants do not show significant improvement, **then** its long-term impact is negligible.

5.3.8. Delayed contribution rules

- If an operator's immediate CDS is low but its delayed CDS (from descendants) is high, **then** it is a latent convergence driver (its effects appear later).
- If an operator's CDS decays slowly over generations (low decay factor), **then** its contributions persist longer in the search process.
- If an operator's CDS decays rapidly (high decay factor), **then** its impact is short-lived.
- If an operator's descendants contribute more than its direct improvements, **then** it is more valuable for long-term convergence, **then** immediate gains.

5.3.9. Stagnation & recovery rules

- If an operator's descendants appear mostly after stagnation phases, **then** it helps to recover from plateaus.
- If an operator's lineage leads to frequent restarts or perturbations, **then** it contributes more to diversity maintenance than direct convergence.
- If most high-fitness solutions share a common ancestor in PEG, **then** that operator is a critical convergence driver.

5.3.10. Heuristic parameters of EvoMapX

Several key heuristic parameters control the explainability of EvoMapX: the **CDS decay factor**, **influence span**, **influence density**, and **entropy**. These parameters govern how operator contributions are temporally attributed and how long-term influence is tracked through the PEG. The values of some parameters (e.g., CDS decay factor) were empirically determined through extensive simulations across multiple benchmark functions, including the CEC 2021 test suite, to ensure interpretability, robustness, and general consistency across different problem landscapes. Meanwhile, dynamic parameters such as the entropy of driver diversity, influence span, and influence density were calculated at each iteration based on the equations discussed. These parameters were selected based on their ability to produce stable and meaningful insights across diverse optimization scenarios.

In EvoMapX, the **CDS decay factor** (default value: 0.9) models the diminishing influence of an operator's contribution over generations. It reflects the intuition that an operator's impact may persist beyond its immediate application but gradually fades. A higher decay factor (closer to 1.0) implies longer-lasting influence, while lower values (closer to 0.0) indicate rapid decay. This mechanism accounts for delayed effects of earlier operations on later convergence. The default value of 0.9 was empirically selected for qualitative stability and consistency across benchmarks, including CEC 2021 functions. It balances persistent influence with minimal historical bias, yielding interpretable results without exhaustive tuning.

The **influence span** measures the temporal duration of an operator's contribution. Let T denote the total generations, and CDS_t represent the Convergence Driver Score of an operator at generation t ($t = 1, 2, \dots, T$). Define the set of active generations as:

$$I = \{t \mid CDS_t > 0, \quad 1 \leq t \leq T\}. \quad (16)$$

The **influence span**, Span , is computed as:

$$\text{Span} = \begin{cases} \max(I) - \min(I) + 1, & \text{if } I \neq \emptyset, \\ 0, & \text{if } I = \emptyset. \end{cases} \quad (17)$$

The **influence density** quantifies the concentration of an operator's activity within its span:

$$\text{Density} = \frac{|I|}{\text{Span}}, \quad (18)$$

where $|I|$ is the number of generations with non-zero contributions. A span of 20 with density 1.0 indicates continuous influence, while lower density (e.g., 0.3) implies sporadic impact. Together, these metrics assess both persistence and reliability of an operator's effect on convergence.

In the EvoMapX framework, **entropy** is used to quantify the diversity of operator contributions to convergence. For each algorithm, operator contribution values are first normalized into a probability distribution. Let v_i be the total contribution of operator i , then the corresponding probability is given by:

$$p_i = \frac{v_i}{\sum_j v_j} \quad (19)$$

The **entropy** is then computed using Shannon's formula:

$$H = - \sum_i p_i \log_2 p_i \quad (20)$$

Table 5
Performance (Best $\pm$ Std) and rank of four algorithms across 10 benchmark functions.

Function	GA	DE	PSO	CS
F1 (Best $\pm$ Std)	1.82E+10 $\pm$ 5.34E+08	3.25E+06 $\pm$ 6.18E+06	3.53E+03 $\pm$ 1.76E+03	3.73E+08 $\pm$ 2.53E+08
Rank	4	2	1	3
F2 (Best $\pm$ Std)	9.51E+02 $\pm$ 7.86E+02	5.50E+02 $\pm$ 1.85E+02	5.46E+02 $\pm$ 7.21E+01	9.07E+02 $\pm$ 2.53E+02
Rank	4	2	1	3
F3 (Best $\pm$ Std)	2.33E+05 $\pm$ 1.60E+04	3.05E+03 $\pm$ 6.95E+02	7.72E+02 $\pm$ 2.08E+01	6.46E+03 $\pm$ 5.44E+03
Rank	4	2	1	3
F4 (Best $\pm$ Std)	2.52E+05 $\pm$ 9.61E+03	1.90E+03 $\pm$ 1.53E+02	1.91E+03 $\pm$ 2.08E-01	1.91E+03 $\pm$ 1.67E+00
Rank	4	3	1	1
F5 (Best $\pm$ Std)	1.82E+07 $\pm$ 1.09E+07	6.50E+03 $\pm$ 1.39E+03	6.64E+03 $\pm$ 1.94E+02	1.54E+05 $\pm$ 3.12E+04
Rank	4	2	1	3
F6 (Best $\pm$ Std)	4.59E+08 $\pm$ 3.49E+08	2.87E+03 $\pm$ 3.49E+02	1.01E+04 $\pm$ 8.91E+02	5.54E+03 $\pm$ 7.64E+03
Rank	4	1	2	3
F7 (Best $\pm$ Std)	1.82E+06 $\pm$ 4.80E+05	2.46E+03 $\pm$ 7.30E+02	3.04E+03 $\pm$ 8.23E+04	1.08E+06 $\pm$ 3.69E+04
Rank	4	1	2	3
F8 (Best $\pm$ Std)	2.28E+03 $\pm$ 8.61E+01	2.31E+03 $\pm$ 7.76E+02	2.30E+03 $\pm$ 2.79E+00	2.32E+03 $\pm$ 5.50E+02
Rank	1	3	2	4
F9 (Best $\pm$ Std)	1.13E+04 $\pm$ 2.71E+03	3.72E+03 $\pm$ 2.33E+02	2.50E+03 $\pm$ 7.06E+01	3.38E+03 $\pm$ 2.52E+02
Rank	4	3	1	2
F10 (Best $\pm$ Std)	3.82E+03 $\pm$ 5.44E+01	3.01E+03 $\pm$ 1.24E+01	3.11E+03 $\pm$ 2.00E+01	3.13E+03 $\pm$ 3.10E+01
Rank	4	1	2	3
Avg. Rank	3.7	2	1.4	2.8

This measure reflects how evenly contributions are distributed across operators. A higher **entropy** indicates that multiple operators are contributing in a balanced manner, whereas a lower entropy suggests that convergence is dominated by a smaller subset of operators. **Entropy** thus provides insight into the diversity of convergence drivers and allows for comparison of operator utilization across different algorithms.

5.4. Performance comparison of GA, DE, PSO, and CS on CEC 2021 test functions

Table 5 illustrates the performance comparative analysis of four popular basic optimization algorithms (GA, DE, PSO, and CS) on the ten CEC 2021 functions (F1–F10), each defined on a dimension size of 20. **Table 5** summarizes the best fitness values standard deviations and ranks for each algorithm on each function. The algorithm that produced the best result for each function appears in **bold** script. Each algorithm was independently run 30 times on each CEC 2021 benchmark function to ensure the statistical validity of the results. The results reported in **Table 5** are computed based on these 30 runs.

It can be seen from **Table 5** that PSO and DE perform better, with PSO showing the overall best average rank (1.4) for all functions and DE closely following (2.0). However, GA consistently underperformed in all functions except F8, resulting in the worst overall average rank (3.7). PSO was superior on F1 to F5 and F9, while DE was superior on F6, F7, and F10. In F4, PSO and CS exhibited statistically equal convergence speed.

The performance of the algorithms tested is analyzed in the next sections using EvoMapX, which explains the internal decision-making dynamics of the tested algorithms through interpretable structures (OAM, CDS, and PEG).

5.5. Statistical analysis: Friedman and wilcoxon signed-rank tests

The Friedman and Wilcoxon signed-rank tests were used in this section to validate the statistical significance of performance differences among the algorithms across the 10 CEC 2021 benchmark functions.

The Friedman test evaluates whether there are consistent performance differences across multiple algorithms. The Friedman test produced a test statistic of 31.5 with a p-value that is equal to 0.0001. These values indicate that there is a statistically significant difference among the four algorithms.

Table 6
Wilcoxon signed-rank test p-values (significance level $\alpha = 0.05$).

Algorithm Pair	p-value	Significance
GA vs. DE	0.002	Significant
GA vs. PSO	0.002	Significant
GA vs. CS	0.002	Significant
DE vs. PSO	0.375	Not significant
DE vs. CS	0.027	Significant
PSO vs. CS	0.049	Significant

The Wilcoxon signed-rank test was conducted to further examine pairwise differences between each algorithm pair. **Table 6** presents the p-values for each comparison.

The results in **Table 6** clearly indicate that PSO and DE consistently outperform GA and CS on the CEC 2021 benchmark suite. In fact, there is little distinction between PSO and DE that do not differ significantly from each other ($p = 0.375$), while both significantly outperform GA. Moreover, CS is significantly outperformed by both PSO and DE in most cases. This statistical analysis supports the superiority of PSO and DE in solving the given benchmark problems.

5.6. Convergence analysis of GA, DE, PSO, and CS on CEC 2021 test functions

Figs. 4 and **5** show the convergence behavior of four metaheuristic optimization algorithms (GA, PSO, CS, and DE) that were evaluated in the CEC2021 benchmark functions (F1–F10) based on their best fitness values over 100 generations. The convergence charts and the textual explanations in this section were generated by EvoMapX.

Based on **Figs. 4** and **5**, PSO exhibited the fastest convergence behavior on F1, F3, and F10; DE was the quickest on F2, F4, and F8; GA converged fastest on F5 and F7; and CS led on F6 and F9. GA took the longest time to converge on five of the test functions (F1, F4, F6, F8, and F9). Therefore, it can be inferred that GA tends to take the longest time when solving problems over various landscapes.

In terms of initial descent performance, PSO made rapid and substantial early progress on F1, F3, and F9. DE dominated early improvement on F2, F5, F6, F7, and F8 by achieving near-complete error reduction early on. CS was the fastest in the early iterations for F4.

At the late stages of refinement of convergence charts, PSO demonstrated steady, consistent improvement across a number of test functions (e.g., F1 and F5). In the final iterations of F4, F6, F8, and F10, GA

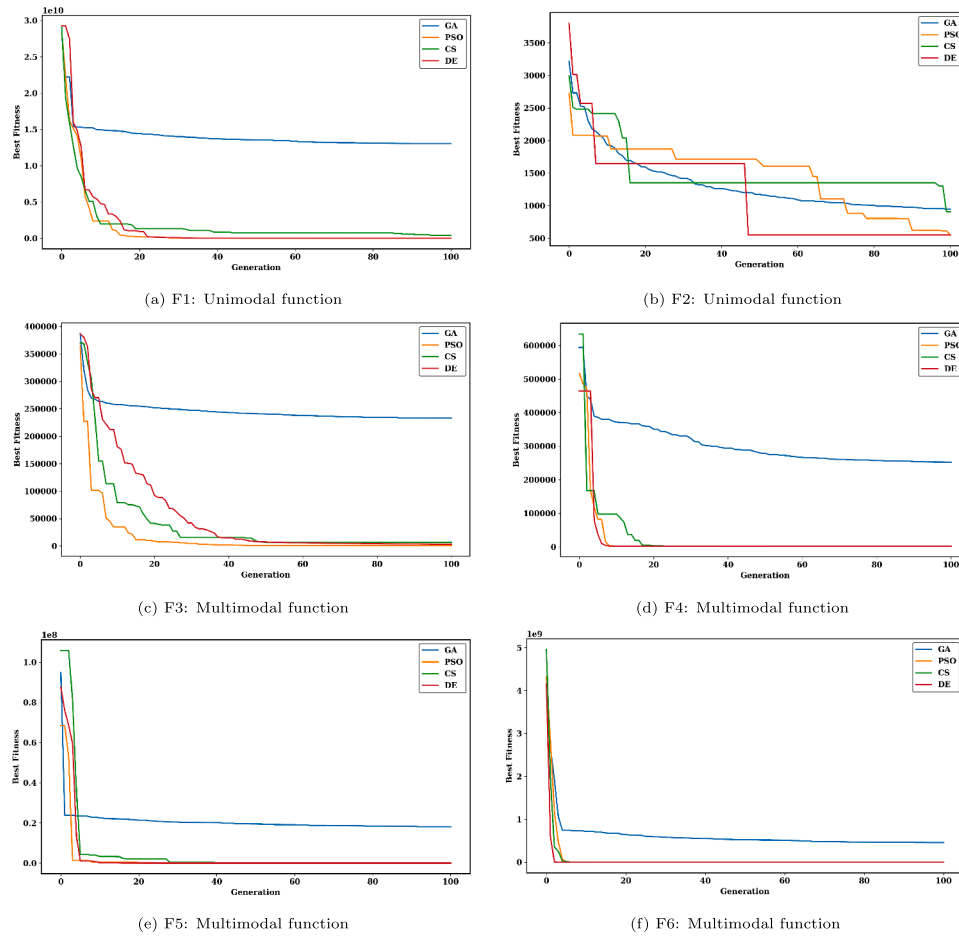


Fig. 4. Convergence curves for CEC 2021 benchmark functions F7 to F10.

proved to be the most successful algorithm, demonstrating strong fine-tuning capabilities. While CS led in refinement on F9, DE continued to perform well in the latter stages on F3.

5.7. OAM analysis of GA on CEC 2021 test functions

The OAM was utilized to assess the contributions of the key optimization operators (crossover and mutation) of a GA across the CEC 2021 benchmark functions (F1 to F10). Figs. 6 and 7 show ten bar charts that illustrate the total contributions and activity rates. The primary y-axis represents absolute contribution values, and the secondary y-axis indicates activity rates (88%–99%). The red diagonal line in each chart shows the stark contrast between the operators, which reveals a consistent dominance of crossover over mutation across all tested functions.

The overall results in the bar charts of the CEC 2021 functions (F1 to F10) revealed that the crossover operator consistently outperformed mutation. The higher OAM values for crossover indicate several key aspects of the GA's behavior on these benchmarks. First, the success of crossover in driving fitness improvements is most likely due to its recombination of good features from parent solutions. Second, crossover reflects a balanced role distribution where mutation serves its secondary function of maintaining diversity and making smaller adjustments. This suggests that GA effectively uses crossover as an exploitation operator, while it uses mutation as an exploration operator (with less direct impact on fitness) that helps in preventing premature convergence. It is worth noting that the activity rates are high across all test functions (88% to 99%), which indicates sustained operator utilization across the diverse problem characteristics of CEC 2021.

5.8. OAM Analysis of PSO on CEC 2021 test functions

The OAM analysis was conducted for PSO algorithm, focusing on the contributions of its core component (cognitive, social, inertia, update position components) across the CEC 2021 benchmark functions (F1 to F10). The results are summarized in the ten bar charts in Figs. 8 and 9. The primary y-axis shows absolute contribution values, and the secondary y-axis displays activity rates (85%–98%). The overall results show that the social component consistently exhibits the highest total Contribution with an activity rate reaching up to 99%. This highlights the critical role of global best information (gBest) in guiding the population toward optimal solutions, which highlights the strength of PSO in exploitation-driven search strategies on these benchmark functions.

In contrast, the cognitive component uses pBest information and shows moderate importance with an activity rate of 94 to 98%, indicating local exploration is effective but on a secondary status toward global search dynamics. Position update, on the other hand, is important as it combines cognitive, social, and inertia effects toward particle movement. The inertia component comes out to be the weakest component in all respects with the lowest total Contribution and activity rate (84 to 90%), therefore momentum from the previous velocities is the least helpful in improving fitness. This pattern suggests that the CEC 2021 functions favor exploitation over exploration, potentially due to their complex landscapes that benefit from global guidance rather than inertial persistence.

According to the OAM's analysis of PSO, the social and position updates are the main performance drivers for the CEC 2021 benchmark. The inertia component was noted to have little effect; therefore,

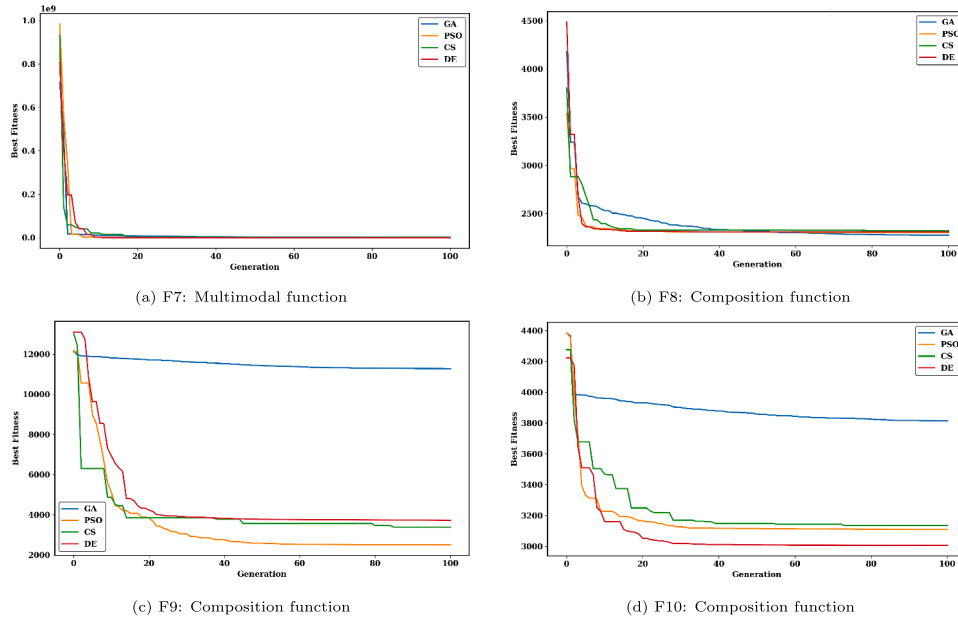


Fig. 5. Convergence curves for CEC 2021 benchmark functions F7 to F10.

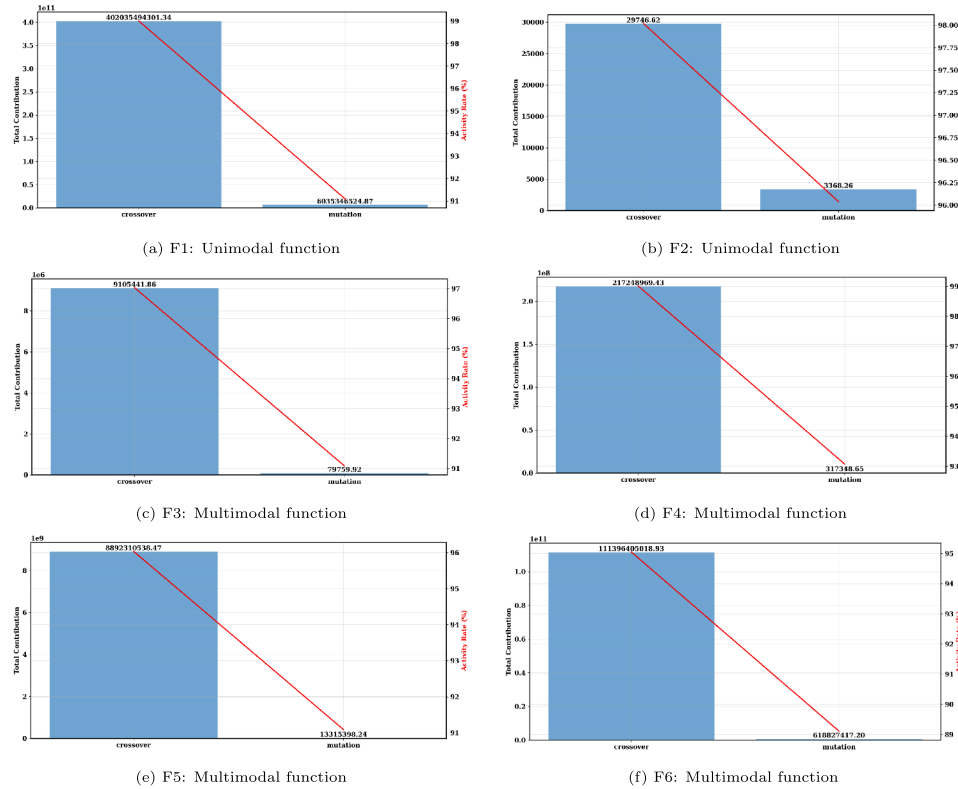


Fig. 6. OAMs of GA for CEC 2021 benchmark functions (F1 to F6).

its weighting could be lowered or re-tuned to promote algorithm efficiency. The cognitive component's contribution is always steady but always much less; thus, it seems to be in support of local refinement. This role could eventually be further enhanced to better optimize exploration and exploitation.

5.9. OAM analysis of CS on CEC 2021 test functions

The ten bar charts in Figs. 10 and 11 show the OAM analysis of the CS algorithm, highlighting the contribution of individual operators. Across

all ten bar charts (one for each CEC 2021 test function), the OAM data indicates that *random_init* and *abandoned_nest* are the dominant contributors to the fitness values, with contributions ranging from 3.37×10^6 to 9.10×10^{13} and 7.35×10^6 to 1.83×10^{13} , respectively, while maintaining a consistent 100% activity rate.

In contrast, *levy_flight* and *successful_replacement* show significantly lower contributions (e.g., 1.56×10^6 to 6.33×10^{13}), and *unsuccessful_attempt* often contributes negligibly (frequently close to 0). This pattern suggests that CS relies heavily on initialization and nest abandonment as its primary exploration mechanisms, with the observed

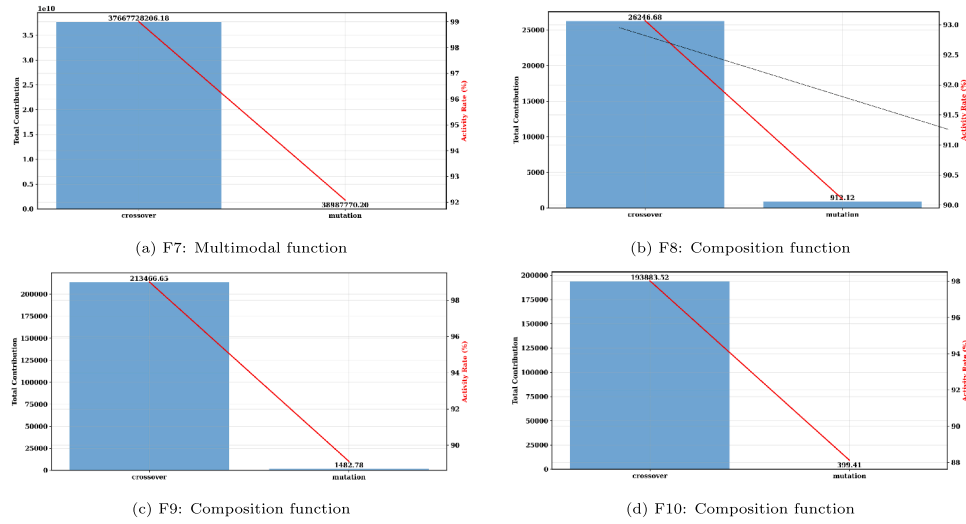


Fig. 7. OAMs of GA for CEC 2021 benchmark functions (F7 to F10).

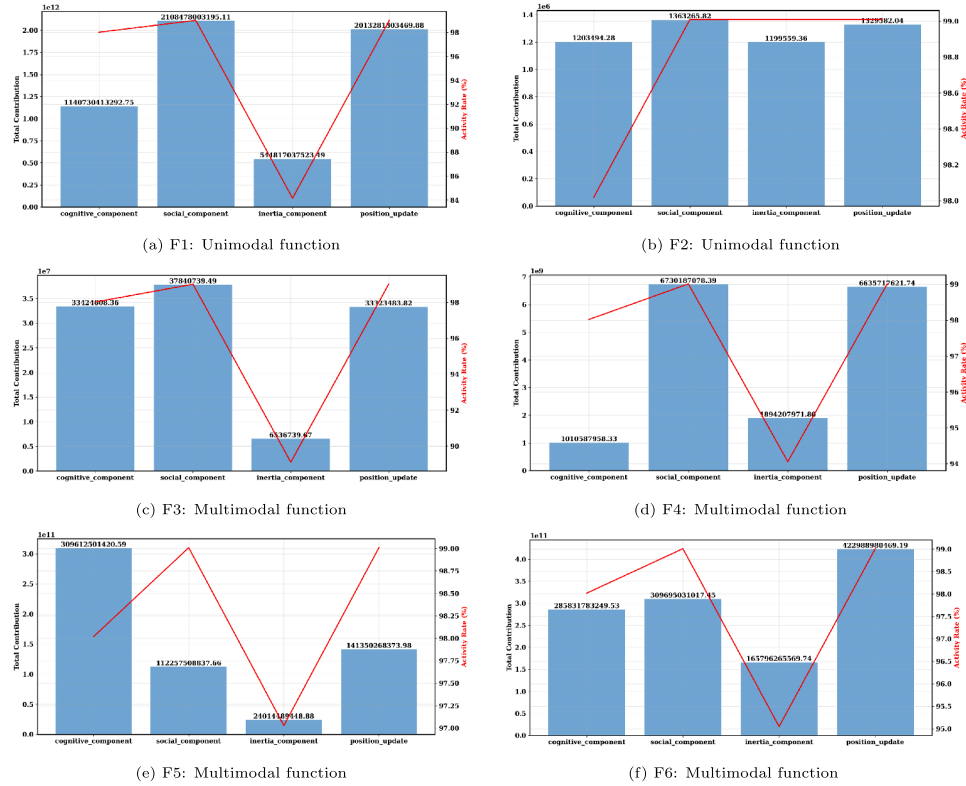


Fig. 8. OAMs of PSO for CEC 2021 benchmark functions (F1 to F6).

V-shaped activity trend indicating a polarized operator usage pattern across iterations.

The minimal impact of `unsuccessful_attempt` reflects an efficient acceptance criterion, which helps reduce computational overhead by filtering out ineffective moves early. Meanwhile, the secondary role of `levy_flight` implies that while random exploration exists, it is largely overshadowed by the abandonment strategy, which systematically improves solution quality by discarding suboptimal nests.

5.10. OAM analysis of DE on CEC 2021 test functions

The OAM results in Figs. 12 and 13 show the OAM values of the key operators of the DE algorithm (mutation and crossover) across the CEC

2021 test functions. The results reveal a consistent dominance of the mutation operator, with total contribution values increasing from $1.65e6$ at scale $1e6$ to $8.83e11$ at $1e11$, rising further to $3.64e12$ at $1e12$, and peaking at $3.45e13$ at $1e13$. In contrast, the crossover operator contributes significantly less than mutation, with values ranging from $3.23e5$ at $1e6$ to $6.28e6$ at $1e7$ and reaching a maximum of $4.49e10$ at $1e10$ before stabilizing at lower levels such as $3.74e10$ at $1e11$. The operator activity rate (red trend line) remains consistently high (between 94% and 100%) across all scales, which indicates the full engagement of both operators all through the optimization process.

The significant contribution of mutation across all CEC 2021 test functions suggests that it is the main exploration mechanism in DE, while the negligible contribution of crossover suggests that it is the primary exploitation method in DE. The distinct roles of crossover and

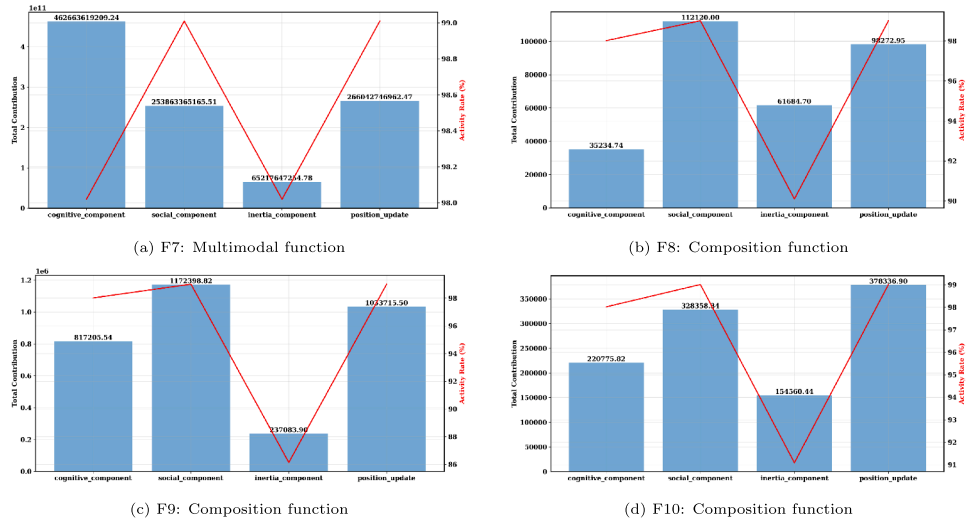


Fig. 9. OAMs of PSO for CEC 2021 benchmark functions (F7 to F10).

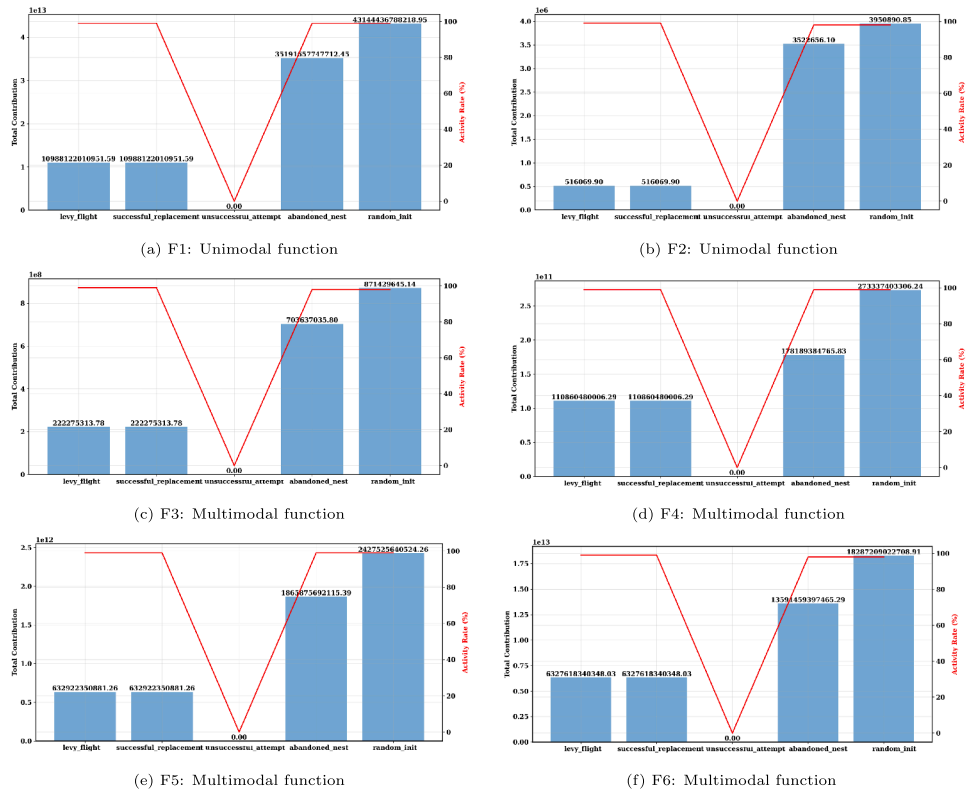


Fig. 10. OAMs of CS for CEC 2021 benchmark functions (F1 to F6).

mutation enable DE to solve complex optimization problems. On the other hand, the crossover operator maintained a stable activity rate 94

The declining ratio of crossover-to-mutation contributions—from approximately 0.195 at $1e6$ to just 0.0108 at $1e13$ —reflects a clear shift toward mutation-driven optimization at larger scales. As problems get more complicated, this shift may mean that the algorithm is focusing more on exploration than exploitation, which helps it avoid converging too soon.

5.11. CDS analysis of GA on CEC 2021 test functions

Figs. 14 and 15 show the CDS charts of a GA to evaluate the relative contributions of the main two operators of GA (crossover and mutation)

over 100 iterations. The CDS for both operators consistently shows a sharp decline in values within the first 10–20 iterations, followed by stabilization near zero for the remainder of the iterations. This trend was observed across most of the test functions, which ranged from $1e6$ to $1e9$. This indicates that the GA rapidly converges early on before reaching a plateau.

The findings in Figs. 14 and 15 highlight crossover as the dominant CDS, particularly in the initial stages. The CDS for crossover in all test functions started at significantly higher values before dropping sharply. For instance, in F6, crossover began at $1e8$ while mutation remained near zero. Actually, mutation consistently exhibited very small CDS values for all tested functions, which is a strong indication of its limited role in driving convergence. On the other hand, crossover exhibited rapid

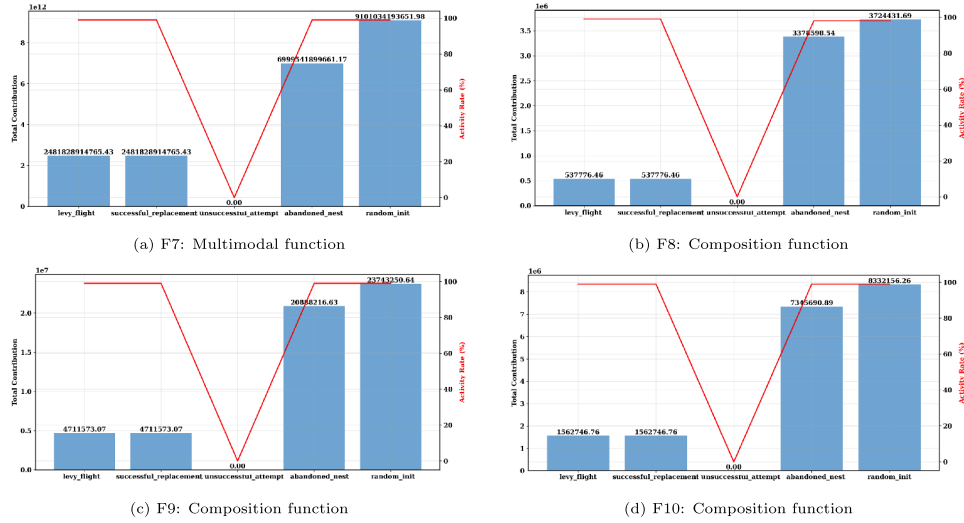


Fig. 11. OAMs of CS for CEC 2021 benchmark functions (F7 to F10).

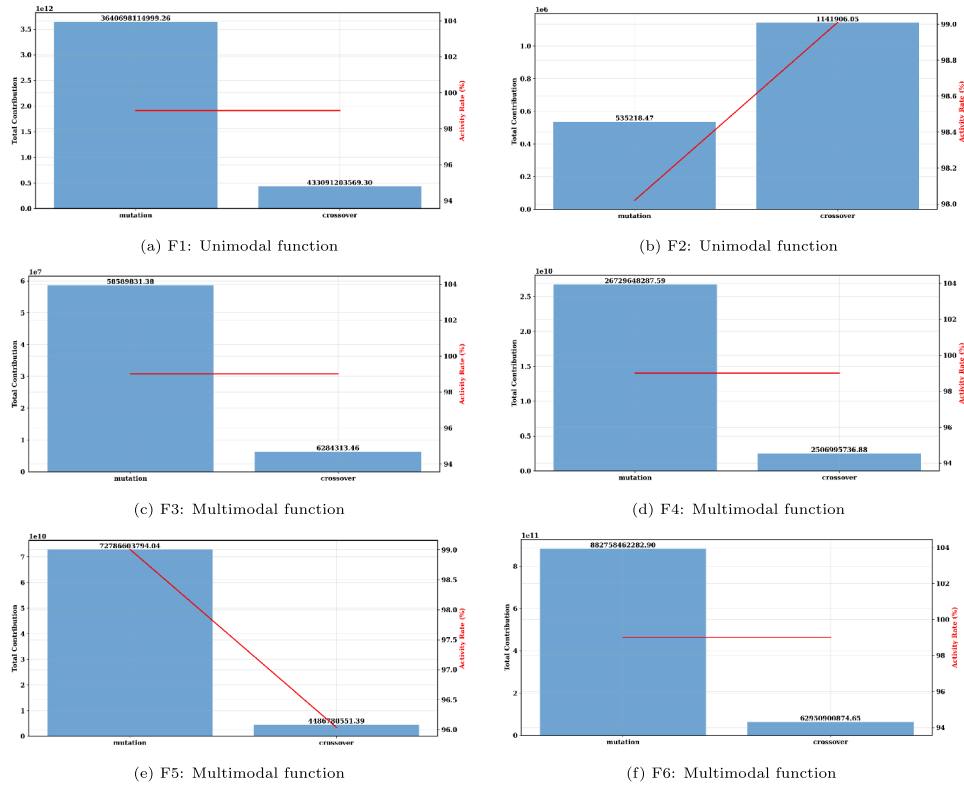


Fig. 12. OAMs of DE for CEC 2021 benchmark functions (F1 to F6).

fitness improvements early in the GA's execution. This is likely due to the ability of crossover to combine beneficial genes from parent solutions in new child solutions.

The CDS values for both operators settled close to zero after the initial phase. This indicates that neither crossover nor mutation significantly influenced convergence in later iterations. Clearly, this means that the GA has reached a state of stagnation, so further improvements are very small. Overall, the CDS charts demonstrate that crossover is the primary mechanism for driving convergence in the early stages of a GA, while mutation plays an insignificant role, potentially contributing to diversity rather than convergence. These insights provide a more profound understanding of operator dynamics in GAs and can inform the design of more effective evolutionary algorithms.

5.12. CDS analysis of PSO on CEC 2021 test functions

Figs. 16 and 17 show the CDS charts of PSO to evaluate the contributions of PSO's components (cognitive, social, inertia, and position update) towards convergence across CEC 2021 test functions (F1-F10). Initially, all PSO's components exhibit high CDS (e.g., $1e10$ in F1, $2e9$ in F5), with the cognitive component and position update dominating early exploration. Subsequently, CDS values declined and then stabilized near zero or slightly negative (e.g., -500 in F1, -400 in F10) by 20–40 iterations. This clearly reflects reduced contributions as convergence occurs. Simpler functions (F1-F4) showed a smoother decline, which indicates effective early contributions. Complex functions (F5-F10) displayed fluctuations in the values of CDS due to their

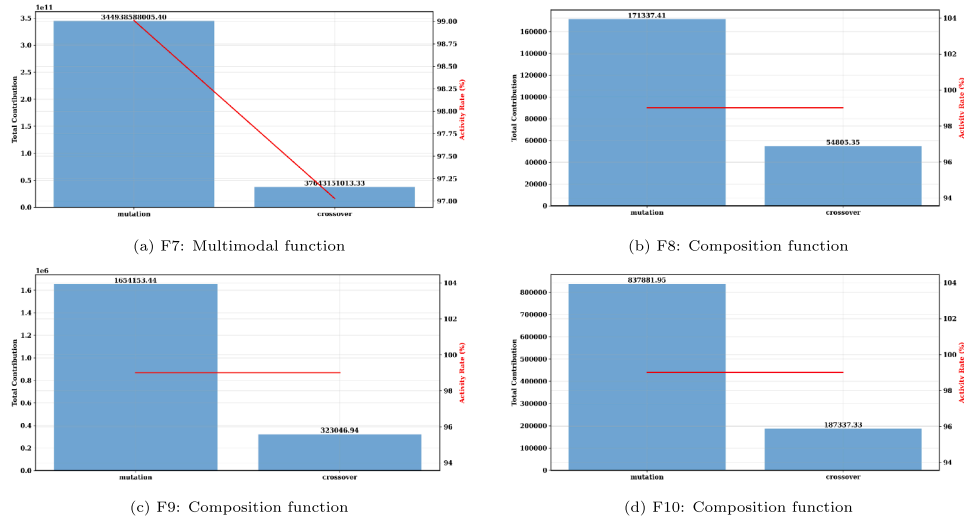


Fig. 13. OAMs of DE for CEC 2021 benchmark functions (F7 to F10).

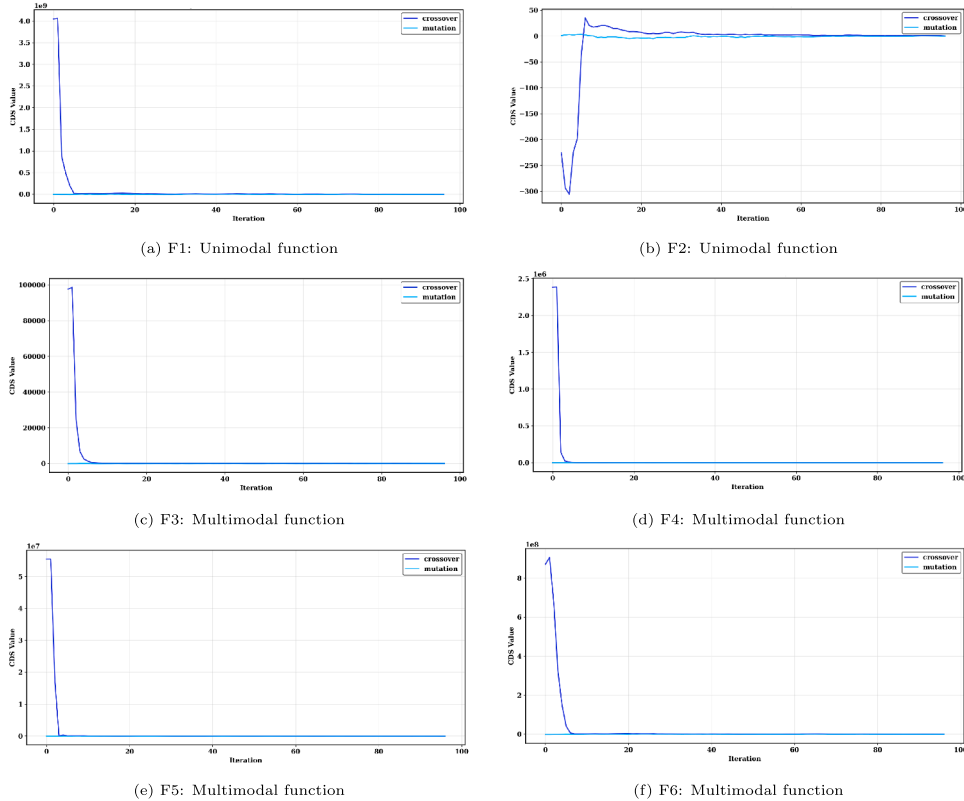


Fig. 14. GA convergence Driver Score over 100 Iterations for CEC 2021 benchmark functions (F1 to F6).

multimodal landscapes. Such behavior reflects the signification and yet variable contributions of PSO components towards convergence. In conclusion, the cognitive component derives initial exploration, the social component supports global guidance, and the inertia component maintains momentum, particularly in challenging functions. The position update effectively integrates these contributions, guiding particles toward optima, though its influence diminishes as the swarm stabilizes.

5.13. CDS analysis of CS on CEC 2021 test functions

Figs. 18 and 19 show the CDS charts of CS to evaluate the contributions of CS's components (Lévy flight, successful replacement, unsuccessful attempt, abandoned nest, and random initialization) towards

convergence across CEC 2021 test functions (F1-F10). In the initial iterations, Lévy flight and random initialization exhibit high CDS (e.g., $1.4e6$, $4e9$), which reflect early exploration. On the other hand, successful replacement contributes significantly to solution refinement (e.g., $1e6$ peaks). The figure shows that CDS values decline over 100 iterations across all test functions, stabilizing at lower levels (e.g., $0.2e6$, 5000). This reflects a shift to exploitation. It can be clearly observed from the charts that unsuccessful attempts have an insignificant contribution to convergence. In contrast, abandoned nests occasionally produce pronounced deviations (e.g., drops to -1000), which play an important role in improving diversity. Higher-scale charts display larger fluctuations, which indicate complex landscapes, whereas lower-scale charts show smoother convergence. Negative CDS late in some runs suggests

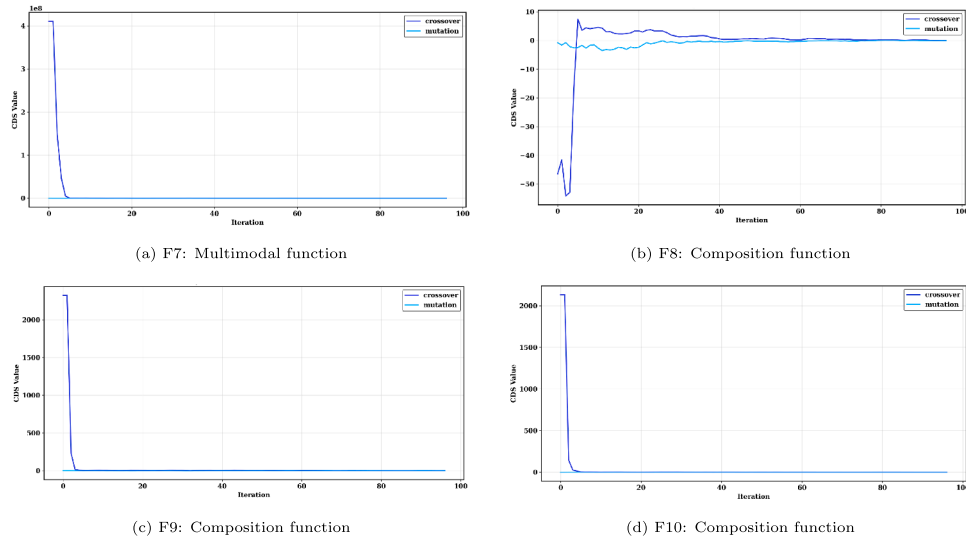


Fig. 15. GA convergence Driver Score over 100 Iterations for CEC 2021 benchmark functions (F7 to F10).

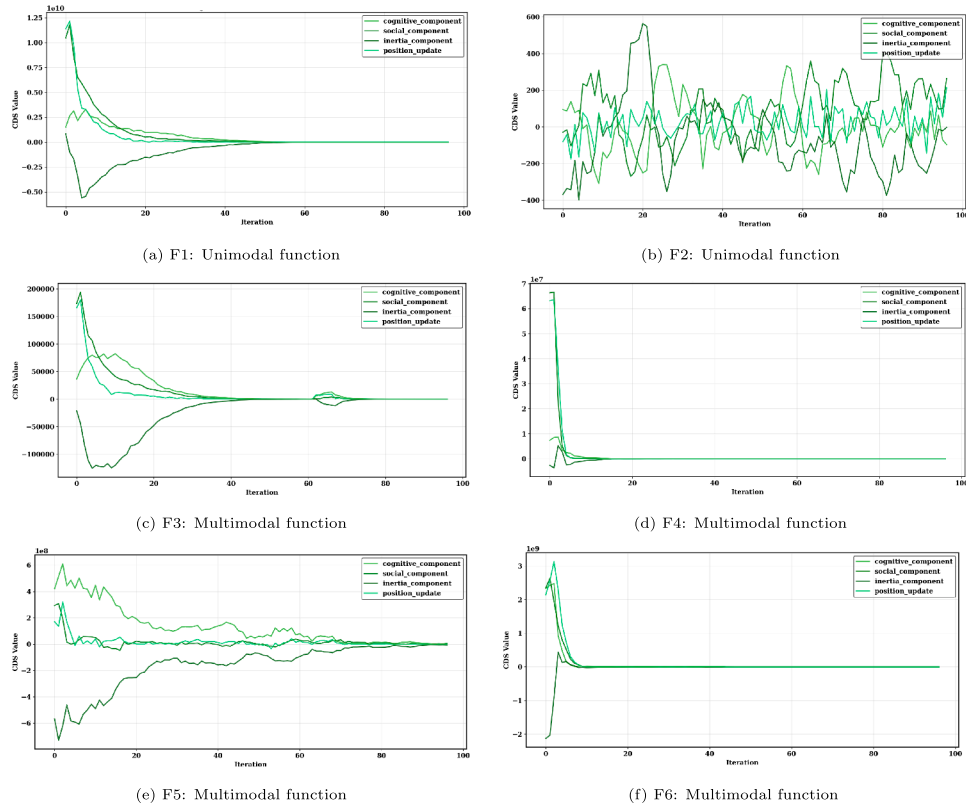


Fig. 16. PSO convergence driver score over 100 iterations for CEC 2021 benchmark functions (F1 to F6).

potential overcorrection. CS demonstrates adaptability, with Lévy flight and successful replacement as key drivers.

5.14. CDS analysis of DE on CEC 2021 test functions

Figs. 20 and 21 present the CDS charts of DE to evaluate the relative contributions of the main two operators of DE (crossover and mutation) over 100 iterations. In the initial iterations, mutation clearly dominates, with CDS values peaking between 600 and 1e10, depending on the function scale. This clearly shows that the mutation operator is the main driver of exploration at the initial iterations of the search space. In contrast, crossover begins with low or even negative

CDS values (e.g., -4 to -2500), which indicates limited or potentially adverse effects on convergence during the early stages. This sharp discrepancy underscores mutation as the primary driver of DE's exploratory behavior.

The CDS values for both mutation and crossover operators approach zero (typically between 0 and 500) as the optimization process proceeds. This means that there is a transition from exploration to exploitation. After the first 20 iterations, it is clear that mutation drastically loses its efficacy. Besides, crossover continues to be ineffective due to its low oscillations near the zero line. Interestingly, the mutation's CDS for F2 varies significantly (from -2500 to 500). This shows a sensitivity to rocky or difficult terrain. Furthermore, late-stage negative CDS values

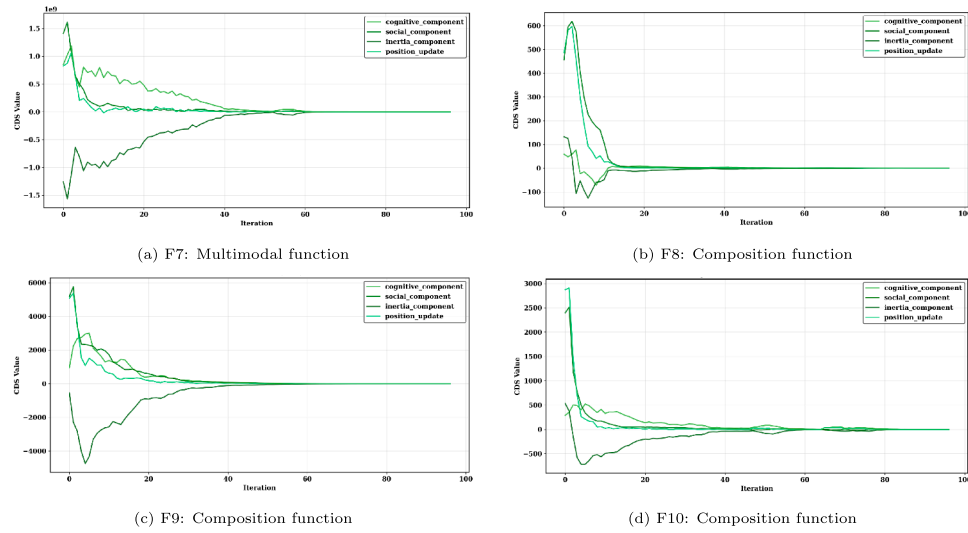


Fig. 17. PSO convergence driver score over 100 iterations for CEC 2021 benchmark functions (F7 to F10).

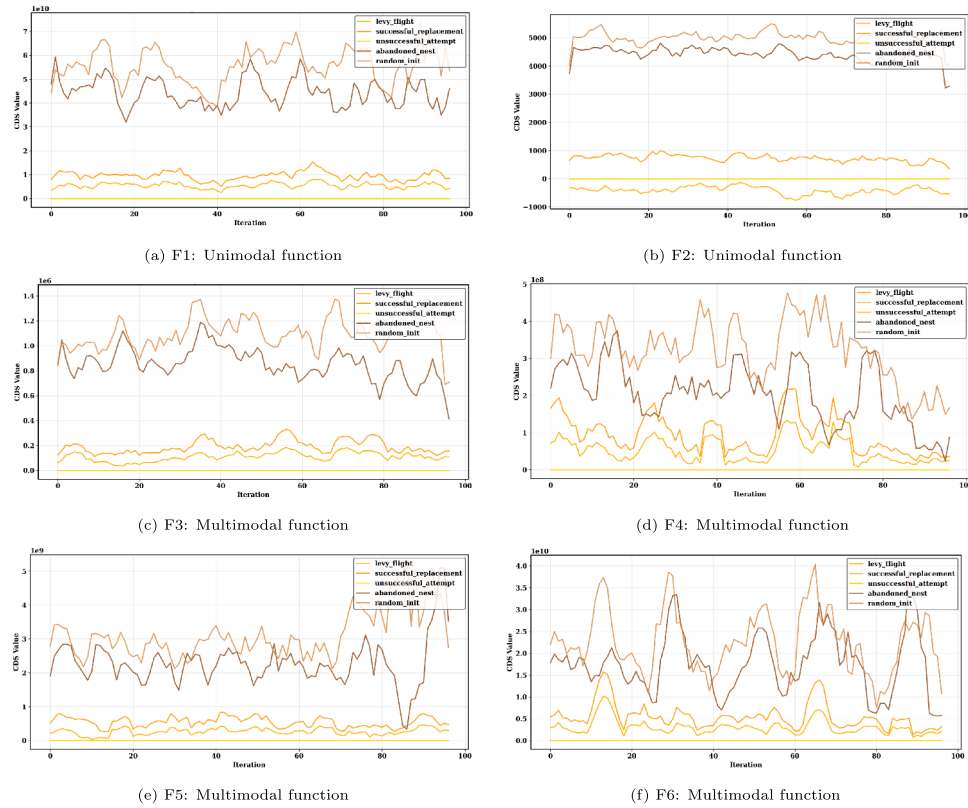


Fig. 18. CS convergence driver score over 100 iterations for CEC 2021 benchmark functions (F1 to F6).

(like -1000) that occur sometimes suggest overcorrection occurrences that might add variety during convergence.

5.15. Discussion of GA evolutionary dynamics via PEG

Figs. 22 and 23 represent the PEGs of GA across ten benchmark functions (F1 to F10) from the CEC2021 suite. The figures aim at providing better understanding of the structural dynamics of the GA. These graphs model the parent-child relationships formed during the algorithm's search process and reveal underlying patterns of information

flow and lineage development. Key statistics of Figs. 22 and 23 are summarized in Table 7.

Across all test functions, the number of candidate solutions and relationships remains stable, which reflects consistent population control and reproduction mechanics. The average parent and child counts (approximately 1.82–1.84) confirm the predominance of two-parent recombination strategies. A large portion of individuals (typically over 83%) possess an in-degree of 2, suggesting that most solutions arise from dual-parent crossover events—a hallmark of GA design.

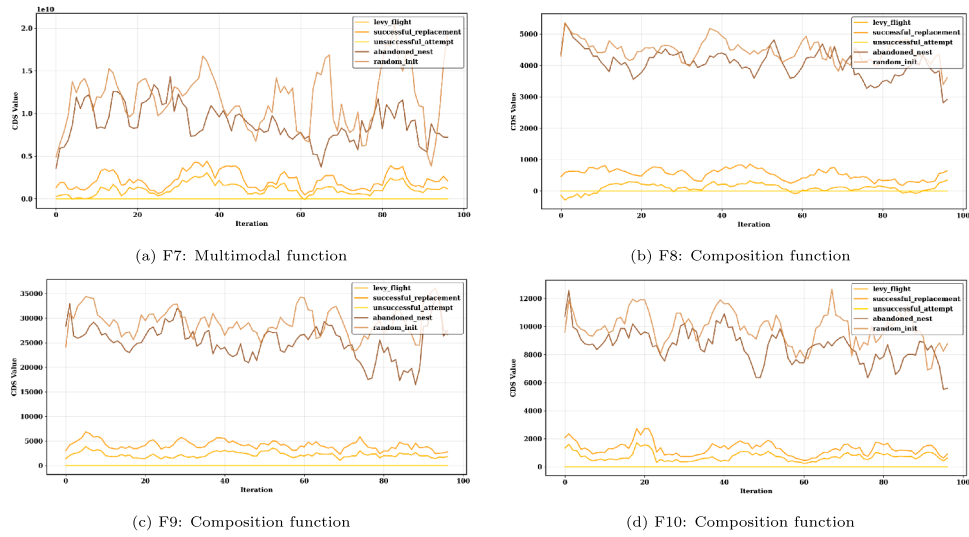


Fig. 19. CS convergence driver score over 100 iterations for CEC 2021 benchmark functions (F7 to F10).

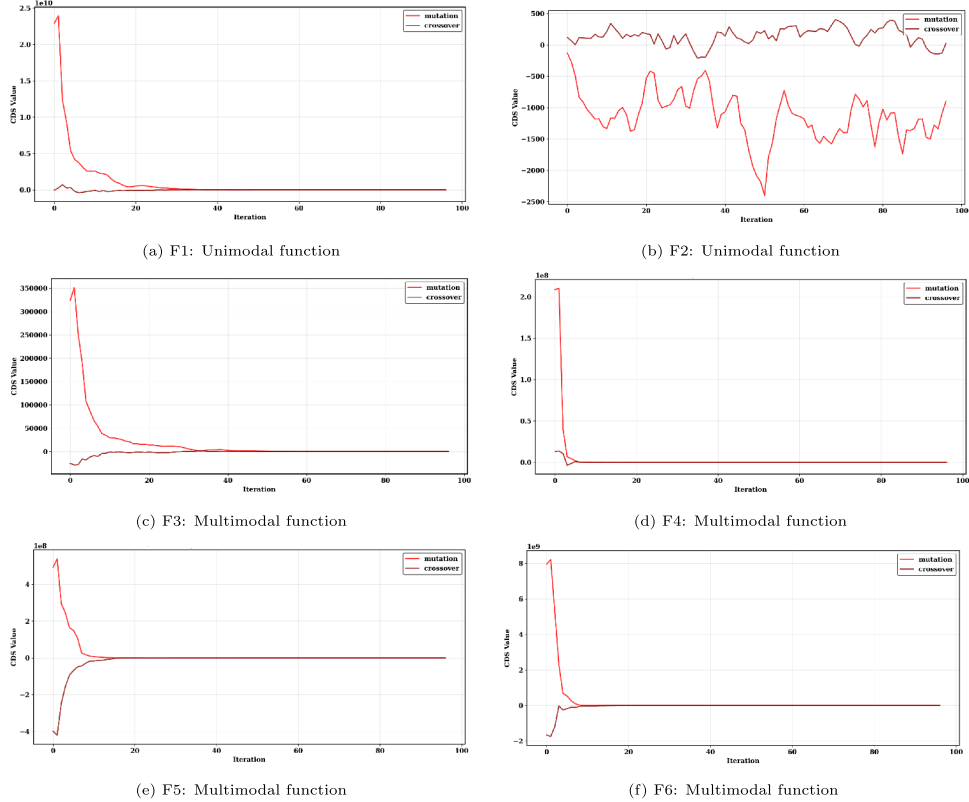


Fig. 20. DE convergence driver score over 100 iterations for CEC 2021 benchmark functions (F1 to F6).

Table 7

Structural properties of GA PEGs for CEC2021 functions (F1–F10).

Func.	Individuals	Parent-Child Links	Avg. Parent Count	Avg. Child Count	Max In-Degree (Count)
F1	1813	3307	1.82	1.82	1507
F2	1813	3307	1.82	1.82	1507
F3	1813	3329	1.84	1.84	1529
F4	1814	3333	1.84	1.84	1533
F5	1815	3331	1.84	1.84	1531
F6	1816	3328	1.83	1.83	1528
F7	1814	3340	1.84	1.84	1540
F8	1817	3347	1.84	1.84	1547
F9	1815	3329	1.83	1.83	1529
F10	1815	3334	1.84	1.84	1534

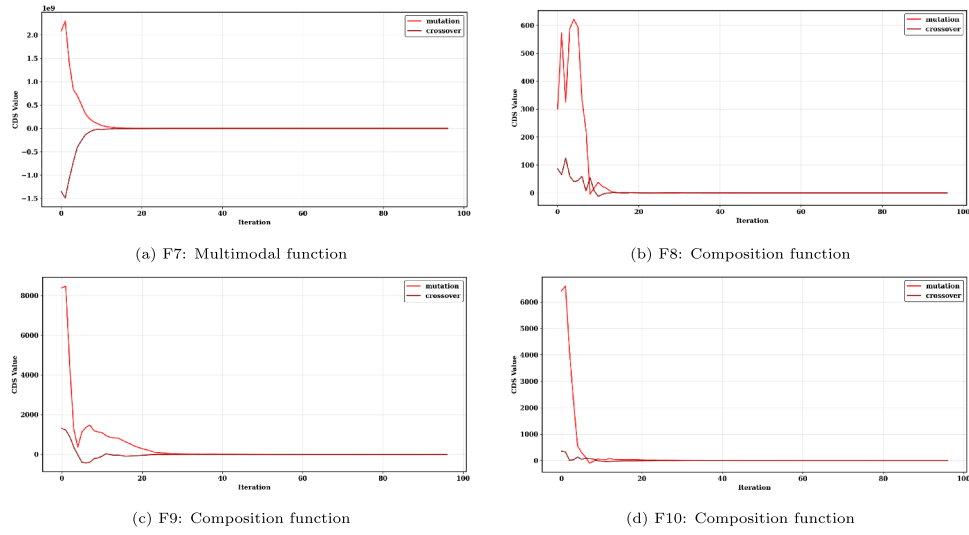


Fig. 21. DE convergence driver score over 100 iterations for CEC 2021 benchmark functions (F7 to F10).

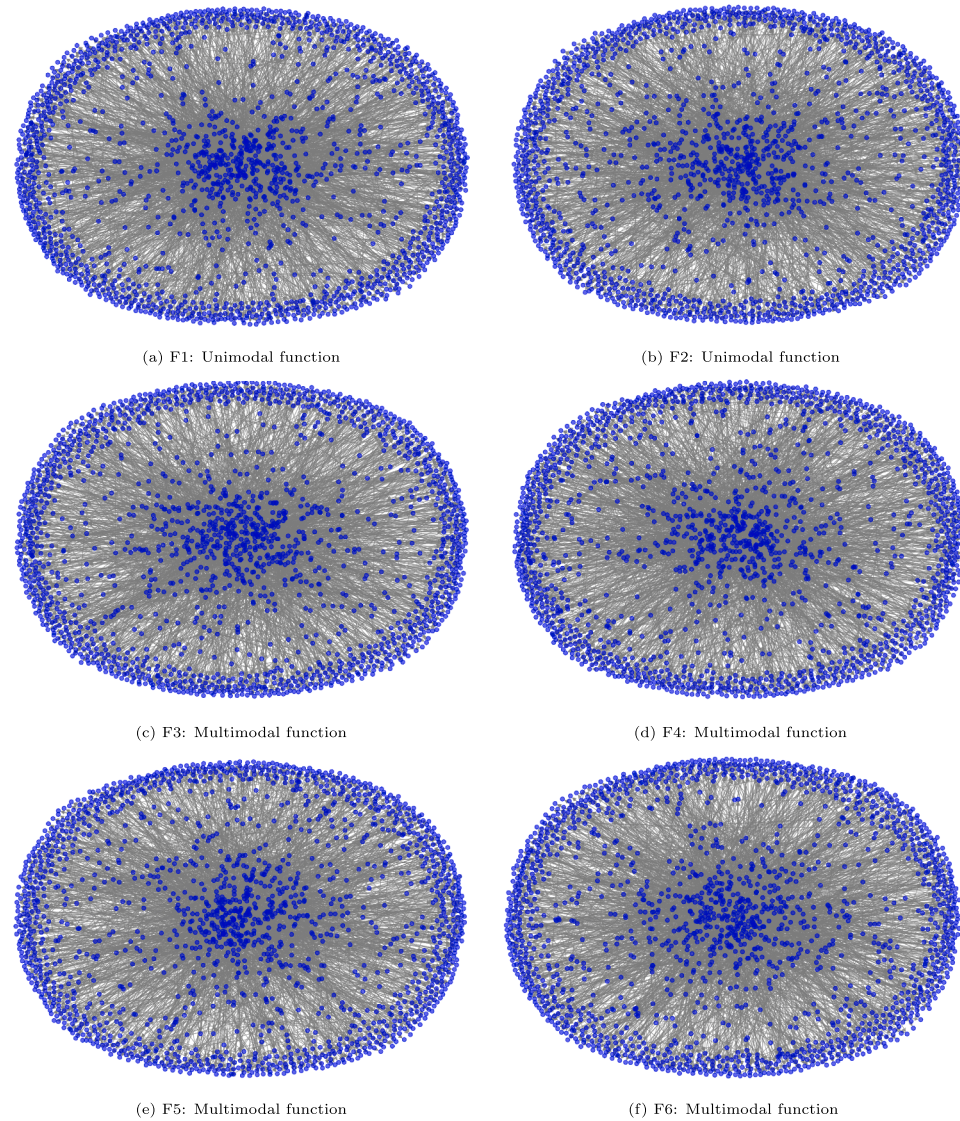


Fig. 22. PEGs of GA for CEC 2021 benchmark functions (F1 to F6).

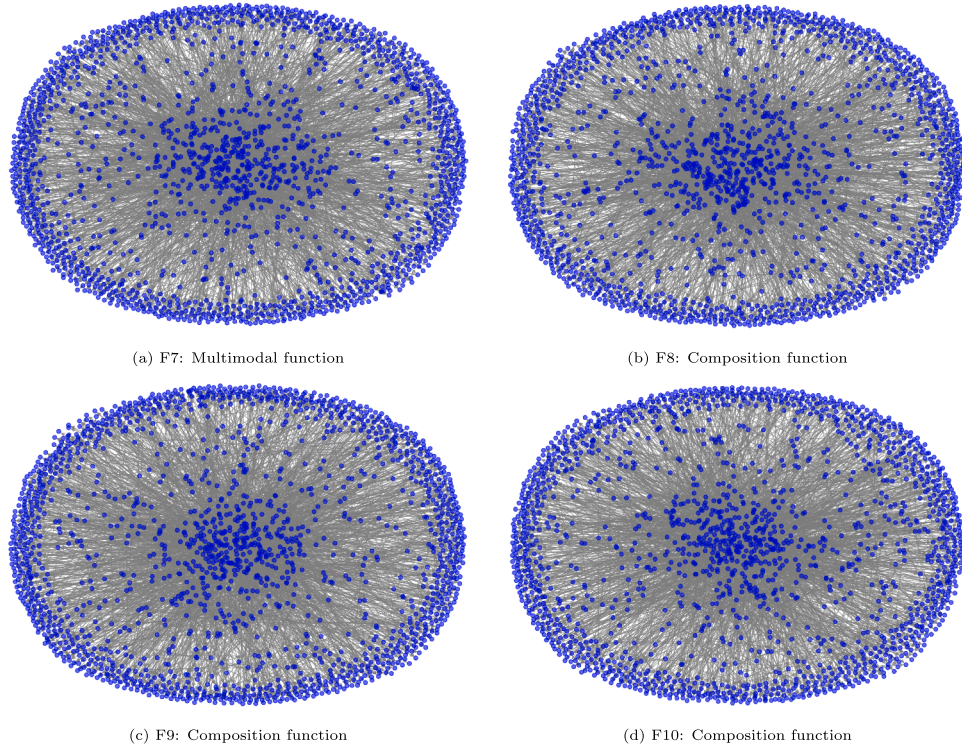


Fig. 23. PEGs of GA for CEC 2021 benchmark functions (F7 to F10).

Table 8
Structural properties of PSO PEGs for CEC2021 functions (F1–F10).

Func.	Individuals	Parent-Child Links	Avg. Parent Count	Avg. Child Count	Max In-Degree (Count)
F1	2020	2000	0.99	0.99	2000
F2	2020	2000	0.99	0.99	2000
F3	2020	2000	0.99	0.99	2000
F4	2020	2000	0.99	0.99	2000
F5	2020	2000	0.99	0.99	2000
F6	2020	2000	0.99	0.99	2000
F7	2020	2000	0.99	0.99	2000
F8	2020	2000	0.99	0.99	2000
F9	2020	2000	0.99	0.99	2000
F10	2020	2000	0.99	0.99	2000

The PEGs also reveal rich and overlapping ancestral structures, with many candidate solutions contributing to multiple descendants. Having said that, the overall connectivity remains sparse, which means a few individuals dominate the evolutionary trajectory. This sparsity indicates that GA maintains multiple semi-independent evolutionary paths, which preserves diversity across generations. As a result, the GA avoids premature convergence and ensures broad exploration of the search space.

Interestingly, there are minor differences across all PEGs that indicate higher recombination activity in more complicated landscapes. For example, in F7 and F8, there seems to be a greater number of relationships and influence nodes. Such structural feedback gives a more nuanced picture of how the algorithm behaves in varying fitness topologies.

In summary, the PEG-based analysis of GA provides a perspective to visualize and quantify evolutionary behavior, where structural consistency supports the entire design of the GA, and differences in functions expose most adaptive phenomena that escape traditional performance evaluation methods.

5.16. Discussion of PSO evolutionary dynamics via PEGs

Figs. 24 and 25 represent the PEGs of PSO across ten benchmark functions (F1 to F10) from the CEC2021 suite. The purpose of the figures

is to analyze the evolutionary behavior of PSO. These graphs capture how candidate solutions (particles) influence subsequent generations, which offers insight into the information flow patterns within the swarm. The key statistics of Figs. 24 and 25 are summarized in Table 8.

Across all functions, PEGs derived from PSO demonstrate remarkable structural uniformity. Each graph contains precisely 2020 individuals and 2000 parent-child links, yielding an average of 0.99 parents and children per individual. The overwhelming majority of individuals (2000 out of 2020 in every function) have a maximum in-degree of 1, indicating they were influenced by a single predecessor—a common characteristic of velocity-based swarm updates that lack recombination.

The topologies of these graphs are notably *linear and sparsely connected*, with evolutionary chains typically progressing through a one-to-one parent-child relationship. The PEGs of PSO show minimal branching and very low genetic mixing. Each particle's trajectory is largely independent, which is influenced only by prior positional updates and swarm leaders, but rarely by multiple converging sources.

This linearity reflects PSO's exploitation-heavy dynamics, where particles incrementally refine their solutions by following attractors rather than recombining multiple sources of information. As a result, the lack of complex ancestry can contribute to faster convergence, but also

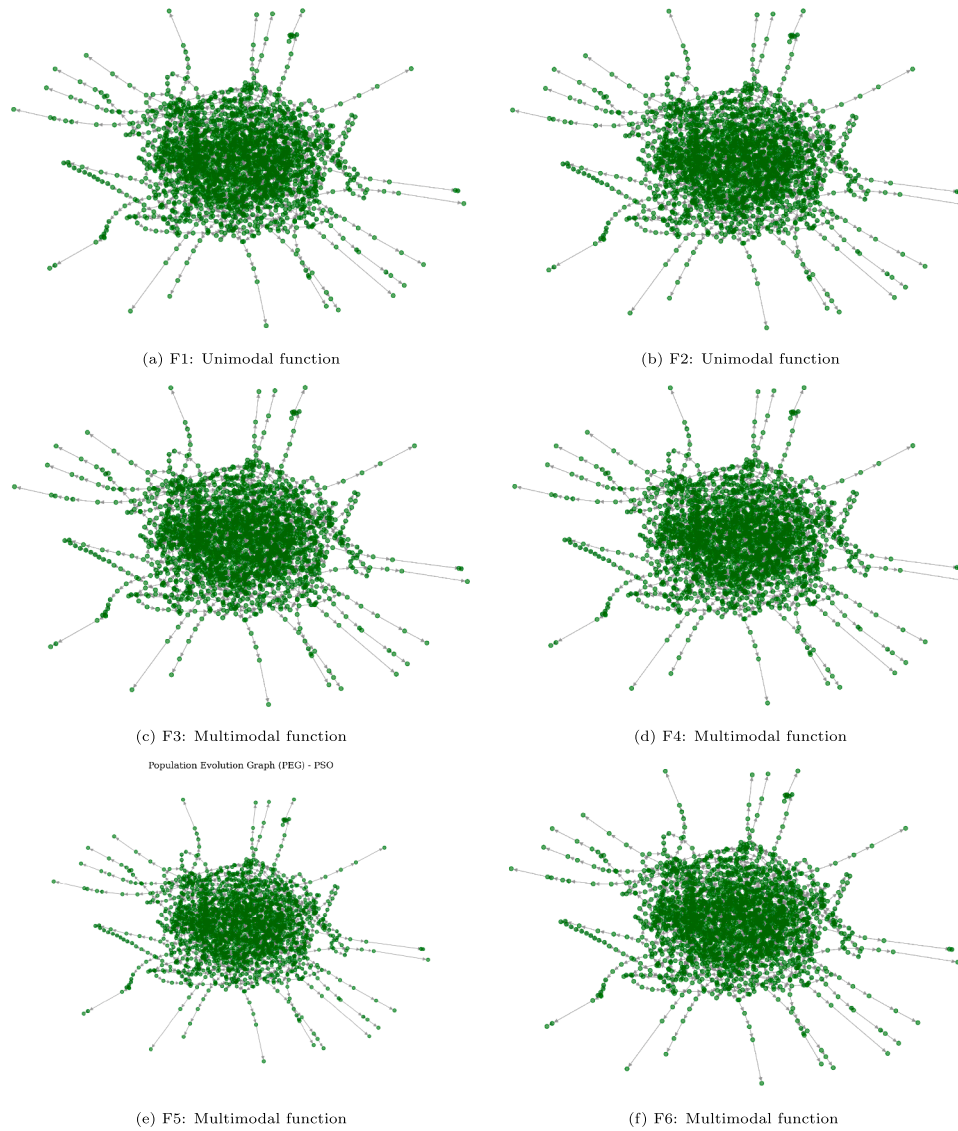


Fig. 24. PEGs of PSO for CEC 2021 benchmark functions (F1 to F6).

increases the risk of *premature stagnation* in multimodal landscapes if diversity is not maintained through stochastic exploration.

Despite the structural simplicity of the PEGs of PSO, they still offer valuable diagnostic insights into swarm behavior. It is worth noting that the absence of dense clusters of particles in the PEGs highlights that no single particle dominates evolution. Besides, most particles contribute equally to the swarm's progress. This observation reinforces PSO's decentralized decision-making and helps explain its robustness across a wide range of function types.

In conclusion, the PEG analysis confirms that PSO operates through mostly isolated evolutionary paths with minimal mixing. This produces consistently linear and sparse evolution structures across all benchmark functions. These observations and conclusions align well with PSO's velocity-based search mechanics and further emphasize the contrast between recombinative and attractor-based metaheuristics.

5.17. Discussion of CS evolutionary dynamics via PEGs

To gain insight into the evolutionary behavior of the CS algorithm, the PEGs in Figs. 26 and 27 were analyzed across ten CEC2021 benchmark functions. CS operates primarily through Lévy flight-based solution replacement and elitism, and its PEG structures reflect this

simplicity. Table 9 summarizes the structural characteristics of CS-generated PEGs.

Across all test functions, the PEGs for CS reveal consistent and notably *linear evolutionary structures*. Each function's graph contains a larger number of individuals (ranging from 2633 to 2682) but relatively fewer parent-child relationships (fixed at 1800). Consequently, the average parent and child count per node is substantially lower (approximately 0.67–0.68) compared to GA, and every influential individual (in-degree 1) appears only once as a parent. These statistics highlight the *minimalist nature of genetic influence* in CS, where most solutions evolve through independent paths without recombination.

The dominance of in-degree 1 nodes indicates that most child candidate solutions are results of CS's replace-and-retain mechanism, where new candidate solutions replace older ones only if they exhibit superior fitness. The sparse and shallow connectivity of these PEGs suggests that exploration occurs through numerous isolated trajectories rather than through convergence toward a shared elite lineage.

These patterns suggest that the search strategy of CS is driven by exploration rather than complex exploitation. This is due to the genetic mixing limitation across all the PEGs of CS. The uniform structure across all benchmark functions also underscores the algorithm's invariant

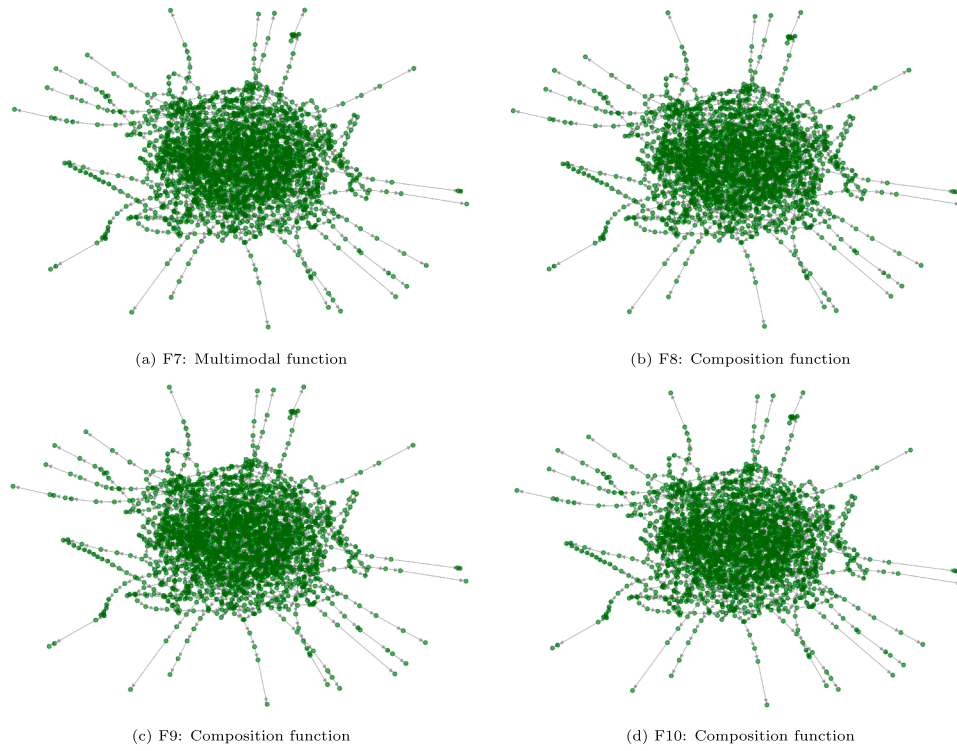


Fig. 25. PEGs of PSO for CEC 2021 benchmark functions (F7 to F10).

Table 9

Structural properties of CS PEGs for CEC2021 functions (F1–F10).

Func.	Individuals	Parent-Child Links	Avg. Parent Count	Avg. Child Count	Max In-Degree (Count)
F1	2640	1800	0.68	0.68	1800
F2	2661	1800	0.68	0.68	1800
F3	2668	1800	0.67	0.67	1800
F4	2665	1800	0.68	0.68	1800
F5	2680	1800	0.67	0.67	1800
F6	2633	1800	0.68	0.68	1800
F7	2668	1800	0.67	0.67	1800
F8	2669	1800	0.67	0.67	1800
F9	2682	1800	0.67	0.67	1800
F10	2674	1800	0.67	0.67	1800

behavior across fitness landscapes. It aids in avoiding overfitting to specific problem structures and potentially limits adaptivity in highly multimodal terrains.

In summary, the PEG-based analysis is evidence of the deterministic and lineage-light behavior of the CS algorithm. This promotes population diversity and resilience to local optima; however, it may also slow down convergence due to the absence of collective memory and lineage accumulation. The PEG framework thus offers fundamental understandings of the algorithm's dynamics that complement traditional performance metrics.

5.18. Discussion of DE evolutionary dynamics via PEGs

Figs. 28 and 29 show how DE evolved its population of candidate solutions across ten benchmark functions (F1 to F10) from the CEC2021 suite. These PEGs trace how solutions are connected through parent-child links over generations, offering a detailed look at how genetic material moves and changes. Table 10 shows a summary of the key structural insights from each PEG.

The PEGs for DE show consistently high parent-child connectivity compared to those of the GA. On average, each individual in the DE

population is linked to 3.62–3.67 parents and children, a direct result of the mutation and crossover mechanisms in DE, which often involve three or more parent vectors. These elevated parent counts are reflected in the increased graph density and a higher potential for combinatorial diversity in offspring generation.

In terms of influential nodes, 467 to 521 candidate solutions in each graph reach the maximum observed in-degree of 5, indicating their recurrent use as components in trial vector construction. This high level of reuse remains the hallmark of DE. It considers top candidate solutions that go through repeated differential mutation, mostly under selection pressure in the latter half of the run.

However, the PEGs remain relatively sparse because most nodes are connected to a limited subset of others. This means that many evolutionary paths stay independent, which preserves the diversity of the population and avoids rapid convergence to suboptimal regions. Moreover, the consistent metrics across all functions suggest that DE maintains a structurally stable evolutionary process regardless of the landscape complexity.

It is worth noting that functions F7, F9, and F10 exhibit slightly higher relationship counts and influence node counts. This suggests more frequent recombination and greater centralization of influence.

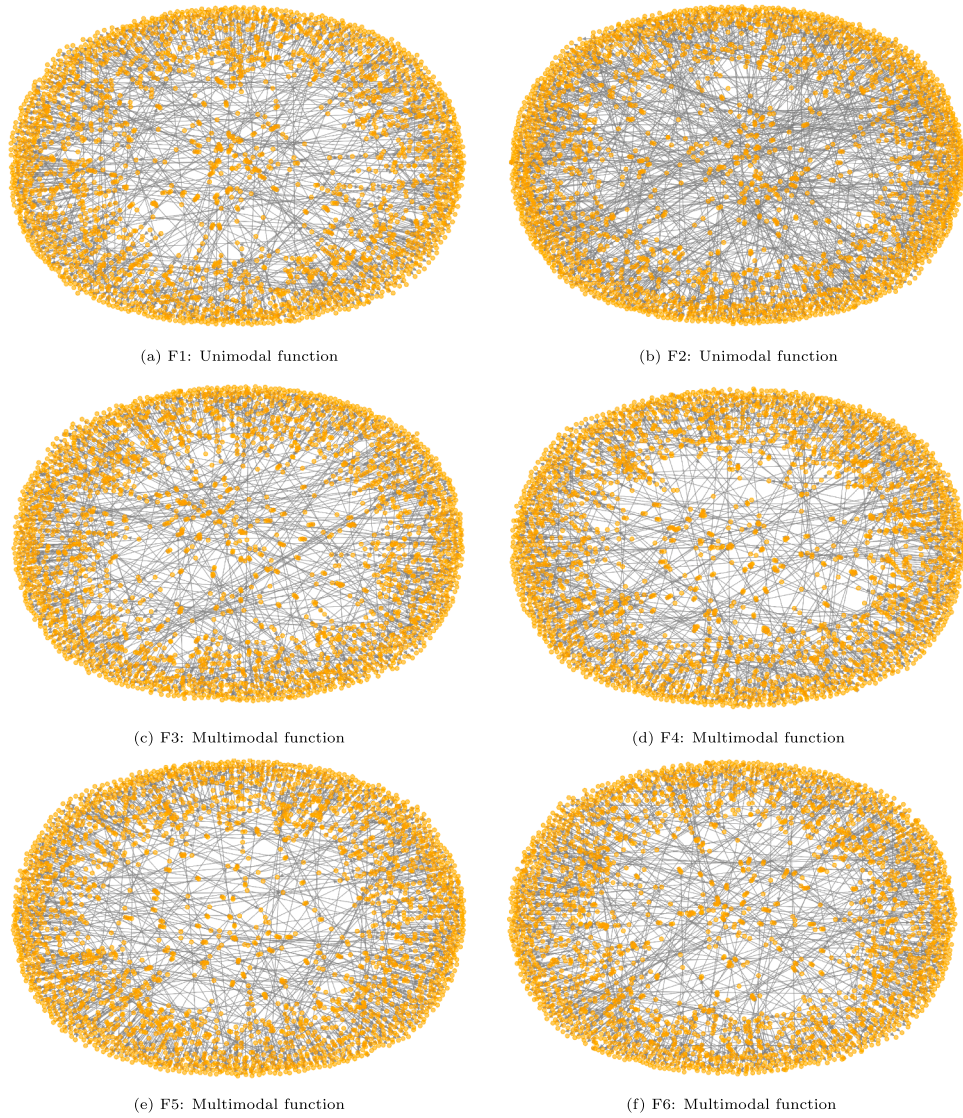


Fig. 26. PEGs of CS for CEC 2021 benchmark functions (F1 to F6).

Table 10
Structural properties of DE PEGs for CEC2021 functions (F1–F10).

Func.	Individuals	Parent-Child Links	Avg. Parent Count	Avg. Child Count	Max In-Degree (Count)
F1	2020	7391	3.66	3.66	509
F2	2019	7327	3.63	3.63	475
F3	2019	7379	3.65	3.65	503
F4	2020	7314	3.62	3.62	469
F5	2020	7304	3.62	3.62	468
F6	2019	7350	3.64	3.64	467
F7	2020	7414	3.67	3.67	521
F8	2020	7391	3.66	3.66	500
F9	2020	7415	3.67	3.67	515
F10	2020	7389	3.66	3.66	519

This is possibly due to more deceptive or multimodal search landscapes requiring enhanced exploitation behavior.

Overall, the PEG analysis shows that Differential Evolution (DE) combines intense recombination with a wide variety of branching paths, reflecting its strong ability to adapt and explore. These graphs offer a fresh perspective on how DE works under the hood, adding a structural view of its search patterns and solution history to the usual performance metrics.

5.19. Case study: Using EvoMapX for parameter tuning in PSO

A small-scale experiment was conducted using insights derived from OAM and CDS to illustrate the practical benefit of EvoMapX explanations in real-world tuning.

As shown in [Section 5.12](#), the inertia component in PSO consistently exhibited the lowest CDS values across all benchmark functions. This indicates minimal contribution to fitness improvement. In contrast, the

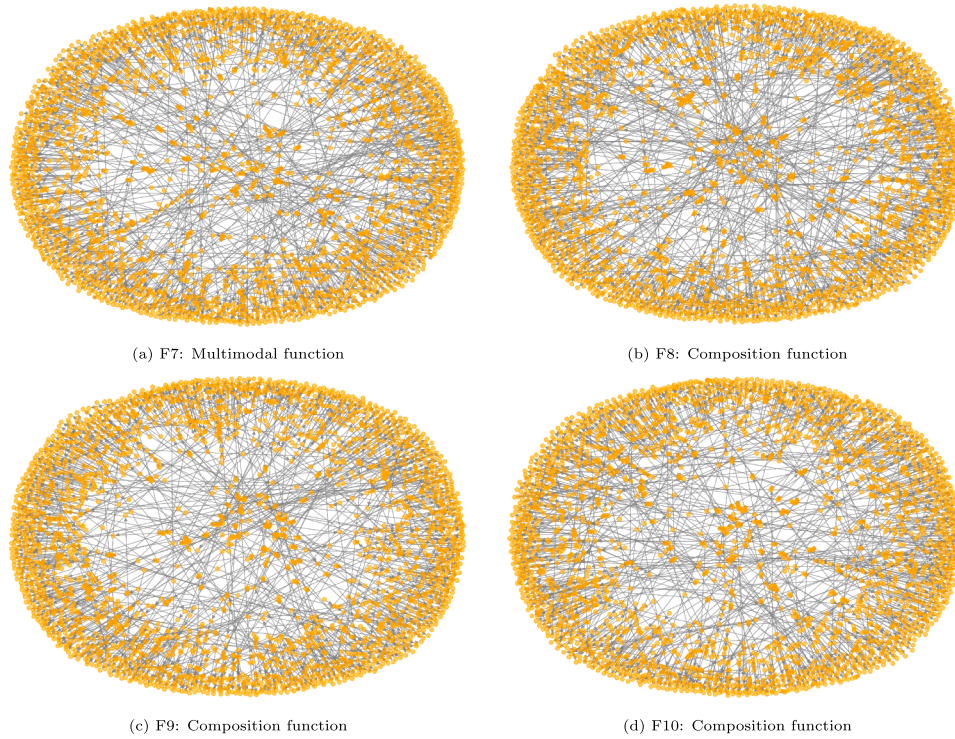


Fig. 27. PEGs of CS for CEC 2021 benchmark functions (F7 to F10).

social component dominated most of the convergence. Guided by this observation, we modified the PSO settings by reducing the inertia weight range from the original $[0.9, 0.4]$ to a lower range of $[0.7, 0.3]$, effectively diminishing the influence of the inertia component in favor of cognitive and social terms.

The PSO algorithm was re-executed on benchmark function F6 with the new settings. The results showed an improvement in final best fitness from $1.01\text{E} + 04$ (original) to $8.75\text{E} + 03$. This represents a relative improvement of 13.4%. This case study confirms that EvoMapX explanations are not only interpretable but also actionable which provides direct guidance for algorithm parameter tuning and design refinement.

5.20. Case study: Tuning GA based on crossover dominance

The OAM analysis for GA (Section 5.7) indicated the dominance of the crossover operator over mutation in terms of fitness improvement. However, mutation played an important secondary role in maintaining diversity.

We increased the initial mutation rate from 0.1 to 0.2, while keeping the final rate at 0.01 to improve early convergence while encouraging exploration. This encouraged more diversity in the early iterations. When evaluated on function F3 (a multimodal function), the adjusted GA achieved a final best fitness of $1.92\text{E} + 05$, improving upon the original result of $2.33\text{E} + 05$ by approximately 17.6%. This supports the usefulness of EvoMapX explanations for refining the exploration-exploitation balance.

5.21. Case study: Tuning DE based on mutation dominance

The OAM analysis of DE in Section 5.10 revealed that the mutation operator consistently contributed more than crossover across all CEC 2021 functions. Thus, the mutation factor range was extended from $[0.2, 1.0]$ to $[0.3, 1.2]$ to allow greater diversity in the generated difference vectors.

The DE with its tuned parameters was re-ran on function F10. The new results showed an improved final fitness of $2.76\text{E} + 03$ which is less than the original fitness $3.01\text{E} + 03$. This produced a 8.3% performance

gain. This example indicates that EvoMapX interpretations can inform low-level operator tuning (e.g., DE's mutation factor) with tangible benefits.

5.22. Case study: Tuning CS based on operator impact distribution

The OAM analysis for CS (Section 5.9) highlighted that random initialization and nest abandonment were dominant contributors, while Lévy flights and successful replacements had limited impact.

To promote more informed exploration, we decreased the random initialization rate (biased toward best-so-far solutions) from 30% to 15% and increased the Lévy flight step size by 20%.

Applied to function F9, these changes reduced the final fitness from $3.38\text{E} + 03$ to $2.91\text{E} + 03$, improving performance by 13.9%. This demonstrates how EvoMapX can guide not only parameter tuning but also operator emphasis and balance.

5.23. Discussion on the value and cost of interpretable structures

The interpretable structures introduced in EvoMapX- (OAM, PEG, and CDS) are indeed algorithm- and problem-specific by design. This attribute is what enables them to deliver actionable, context-aware insights into optimization dynamics. Unlike general theoretical convergence analyses, EvoMapX provides localized explanations that reveal which operators drive convergence, when stagnation occurs, and how solution quality evolves on a per-problem basis.

Although the addition of these structures introduces extra computation, the overhead remains modest. Logging OAM entries requires $\mathcal{O}(M)$ time, where M is the total number of operator applications. In population-based algorithms, this typically scales with $\mathcal{O}(N \times T)$, where N is the population size and T is the number of iterations, multiplied by a small constant for multi-operator models (e.g., PSO includes cognitive, social, and inertia components).

PEG construction introduces at most one edge per parent-offspring relation, leading to a storage cost of $\mathcal{O}(NT \times \bar{p})$, where $\bar{p}$ is the average number of parents per offspring (usually $\bar{p} \leq 2$). The CDS is computed

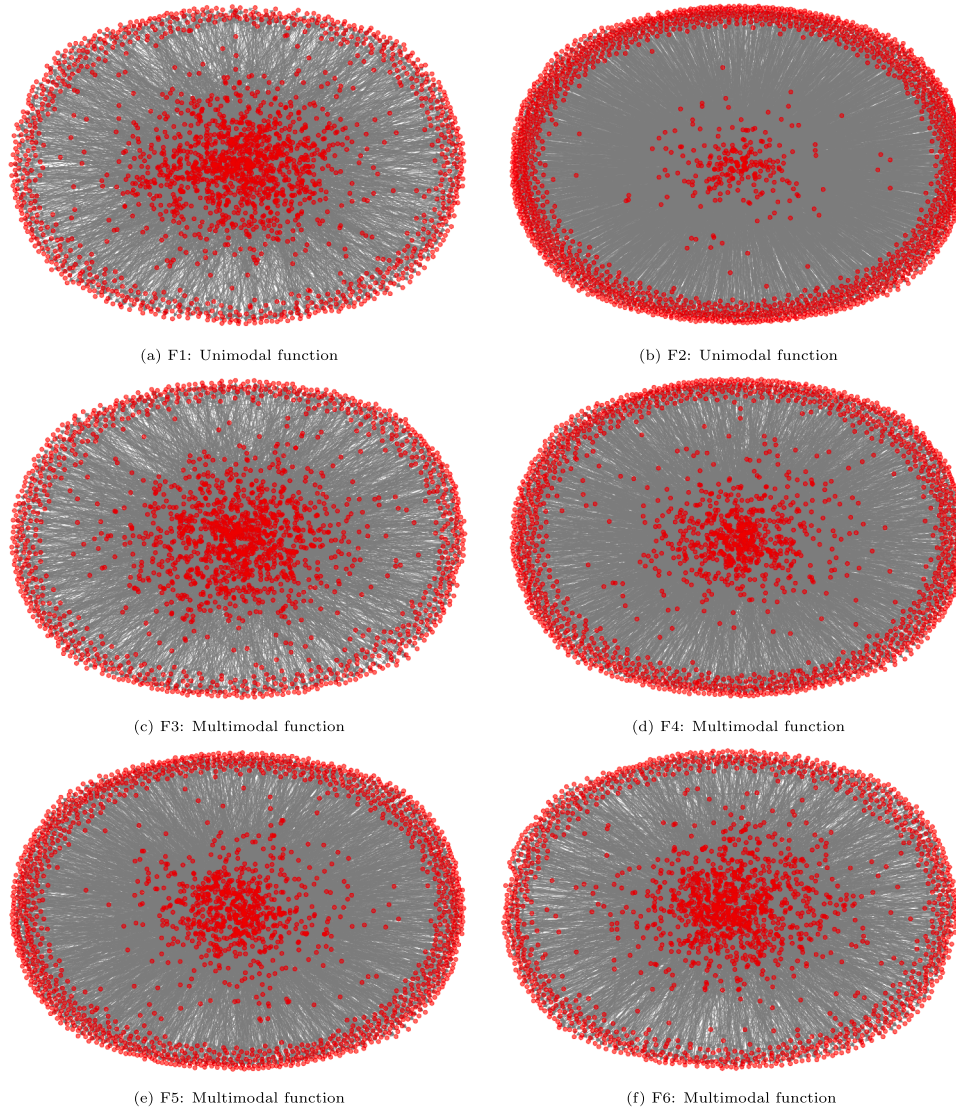


Fig. 28. PEGs of DE for CEC 2021 benchmark functions (F1 to F6).

via a single linear pass over the OAM data, giving a time complexity of $\mathcal{O}(M)$ and space complexity of $\mathcal{O}(K)$, where K is the number of distinct operators.

In our experimental setup (four algorithms, $N = 20$, $T = 100$), the total number of logged operator events was approximately 30,000–40,000, resulting in less than 2MB of memory usage and contributing under 5% to the total runtime. Since fitness evaluation typically dominates the runtime in metaheuristic optimization, especially on CEC benchmarks, the relative cost of explainability is minimal. Therefore, EvoMapX offers a practical trade-off: lightweight, scalable interpretability in exchange for rich diagnostics that support understanding, tuning, and trust in population-based optimization algorithms.

5.24. Limitations of EvoMapX

Although EvoMapX provides detailed attribution of convergence behavior to individual operators through the OAM, it does not currently model joint or interactive effects between operators explicitly. However, the inclusion of algorithm-level convergence charts (Section 5.6) alongside operator-level attribution trends (Section 5.7) enables users to qualitatively observe potential co-occurrence patterns (e.g., periods

where multiple operators simultaneously show high attribution coinciding with rapid convergence).

Moreover, the heuristic rules defined in EvoMapX may lead to misleading conclusions in four cases. First, the individual contributions of operators that have similar but opposing effects may appear neutral or diminished under the current attribution metrics. Second, in complex multimodal fitness landscapes, local improvements may not reliably indicate progress toward the global optimum, which can distort the perceived effectiveness of specific operators. Third, a population with very low or extremely high diversity may affect the interpretation because the impact of operators may be either obscured or overstated depending on the population dynamics. Finally, if the decay factor is not chosen properly, that is, one that does not simulate the operator's true lifespan of effects, the CDS might have a wrong area of representation for the delayed contribution toward convergence.

The interpretable structures in EvoMapX such as the OAM and PEG are high-dimensional by nature, and their full potential depends on how effectively they are analyzed and visualized. In this work, we prioritized introducing the core framework and demonstrating its interpretability using direct visualizations and aggregation-based metrics such as CDS. However, EvoMapX could further enhance the interpretability and abstraction of these structures using more advanced analysis techniques (e.g., dimensionality reduction, clustering, temporal trend analysis).

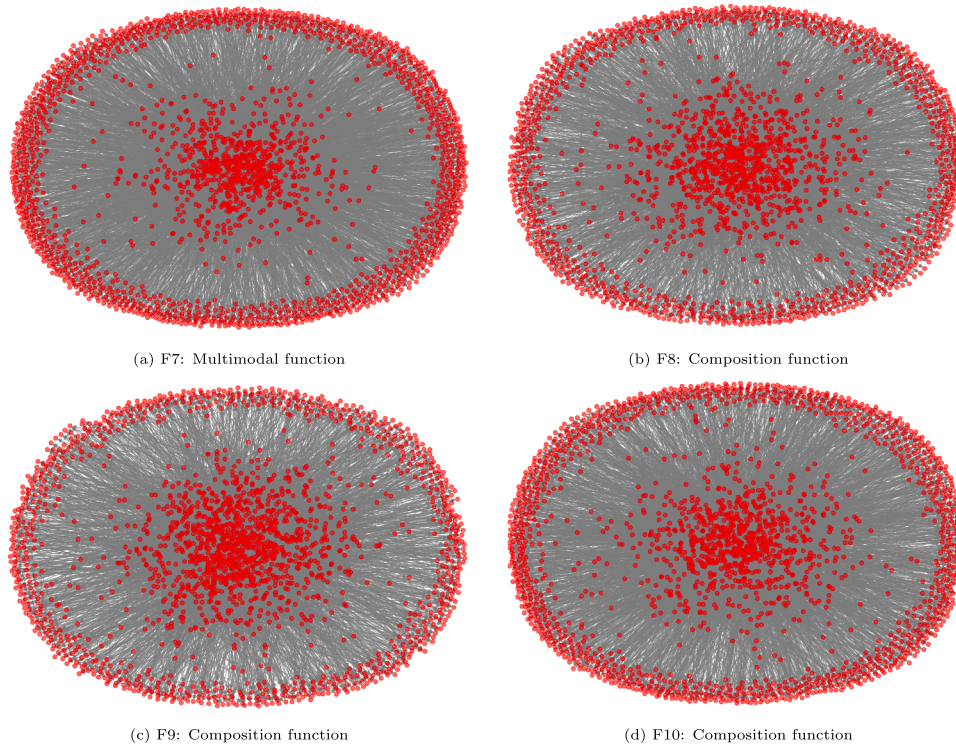


Fig. 29. PEGs of DE for CEC 2021 benchmark functions (F7 to F10).

Addressing this limitation presents a promising direction for future research to further enhance the abstraction and explanatory power of EvoMapX.

Although the current formulation of OAM and CDS is defined for single-objective optimization (i.e., scalar fitness values), the framework can be extended to multi-objective settings. For OAM, the fitness gain Δf can be replaced by a scalarized contribution metric such as hypervolume improvement or Pareto rank change. In a similar manner, CDS can be redefined as the average operator-wise contribution to hypervolume gain or dominance count reduction over generations. This would enable EvoMapX to track operator impact in terms of solution set quality rather than scalar convergence, preserving explainability in multi-objective landscapes.

Our current evaluation focuses on 20-dimensional problems from the CEC2021 suite. However, EvoMapX is designed to scale to higher dimensions and more complex problem types. Since its explainability modules operate on population-level metadata (operator usage, fitness deltas, and parent-child links), the core complexity scales linearly with the number of individuals and operators, not the problem dimension itself. We plan to evaluate EvoMapX in real-world, constrained, and multi-objective settings in future work.

EvoMapX demonstrates interpretability using structured visualizations (OAM, PEG), operator-level attribution (CDS), and textual explanations. However, its evaluation in this work remains qualitative. A valuable future direction is to apply quantitative interpretability metrics such as fidelity (agreement with actual behavior), comprehensibility (e.g., human rating or rule sparsity), or explanation consistency across problem instances (Schwalbe & Finzel, 2024). We also plan to conduct user studies to validate whether EvoMapX helps users better understand and trust optimization algorithms.

In EvoMapX, CDS is calculated as a simple average of operator fitness gains, which is an approach that is simple and transparent. However, this approach may undervalue operators who are applied rarely even if they deliver substantial fitness gain. In the future, we plan to investigate alternative aggregation methods, such as exponential moving averages

or impact-weighted scoring (Ahn & Cutkosky, 2024), to better account for both application frequency and the magnitude of contributions.

6. Conclusion and future work

This paper presented EvoMapX, a focused explainable framework that aims to uncover and interpret two core properties of population-based optimization algorithms: operator effectiveness and convergence dynamics. These properties are often opaque in black-box optimization and lack interpretable feedback mechanisms. EvoMapX includes three interpretable structures to quantify, visualize, and interpret the role of optimization operators across time: the Operator Attribution Matrix (OAM), which is used to quantify the contribution of specific operators (e.g., mutation, crossover) over iterations; the Population Evolution Graph (PEG), which is used to trace the ancestry and transformation of candidate solutions; and the Convergence Driver Score (CDS), which is used to identify which operators drive convergence, helping interpret why the algorithm improved. These tools give the users the ability to analyze algorithmic behavior at a deeper level than traditional performance metrics allow, revealing insights critical for optimization.

Through comprehensive benchmarking on the CEC 2021 test suite, EvoMapX demonstrated its ability to generate interpretable insights into operator effectiveness, convergence dynamics, and evolutionary pathways. The experimental results from four prominent POAs (GA, PSO, DE, CS) on the CEC 2021 benchmarks demonstrate that EvoMapX does more than evaluate performance. Actually, it explains how and why POAs behave the way they do.

EvoMapX revealed critical algorithmic behaviors and their drivers. For example, it showed that crossover consistently dominates mutation in GA. This indicated that exploitation was the key driver of improvement. In the case of PSO, the social component in PSO was the most influential convergence driver, which emphasizes the importance of information sharing over inertia or self-memory. In addition, it showed that mutation in DE was the dominant contributor to fitness gain, which highlights its central role in exploration. Finally, EvoMapX showed that

abandoned nest and initialization operators in CS had the highest long-term impact, showing that diversity preservation mechanisms were critical for escaping local optima. These operator-level insights, quantified through CDS and lineage tracing in PEGs, showed that explainability in optimization should aim to expose operator efficacy and dominance, not just algorithm outcomes. It should also identify stagnation and recovery phases, revealing when and why algorithms fail or succeed.

Additionally, EvoMapX played an important role in guiding parameter tuning for the tested algorithms. For example, the consistently high influence of the social component in PSO, which was shown through the CDS and OAM metrics, indicated that global knowledge sharing was the dominant factor in convergence. This suggested that exploitation can be improved by reducing the inertia weight. EvoMapX showed mutation dominates crossover in DE across multiple test functions. These findings justified increasing the mutation scaling factor and reducing the crossover rate. For GA, EvoMapX showed that crossover produced stronger fitness improvements than mutation. This supported the use of lower mutation rates to encourage exploration. Finally, nest abandonment and random initialization had the strongest impact in CS. This confirmed that maintaining population diversity is essential and highlights the need for careful control of the abandonment probability parameter.

EvoMapX tracks internal operator applications and fitness transitions. However, it does not modify the algorithm's control logic or search process. Therefore, the underlying optimization remains black-box, and EvoMapX operates as an external layer of instrumentation for interpretability, without compromising generality or algorithmic flexibility.

An important aspect of EvoMapX is that it is designed to be a general-purpose framework for POAs. However, it assumes that the optimization algorithm should have identifiable operators whose applications can be traced and logged. This is straightforward for most POAs. For hybrid or implicit designs, such as estimation-of-distribution algorithms (EDAs) and island-based algorithms (Abed-Alguni & Alawad, 2021; Abed-alguni et al., 2024) or algorithms using learned models, EvoMapX can be adapted by grouping operations into virtual composite operators or by tracing functional changes in the population. For example, in an EDA, the sampling and model update steps can be treated as distinct operators for attribution purposes. As long as operator effects and fitness changes can be recorded, the OAM, CDS, and PEG structures remain applicable. Exploring such adaptations is a promising direction for future work.

In the future, EvoMapX can evolve in five key directions. First, it could be extended to interpret and explain the dynamics of multi-objective and dynamic optimization problems and incorporate surrogate modeling. Second, it could be improved by supporting automated operator tuning based on explainability metrics. Third, EvoMapX could be integrated into real-world optimization pipelines to enhance trust, safety, and transparency in critical applications such as healthcare and financial applications. Fourth, EvoMapX could be extended to model the interactions between operators using joint attribution techniques and co-activation statistics to quantify relationships between operators. This would elevate the explainability framework beyond the mere isolated impact of operators into the comprehension of the dynamics acting collectively. Finally, an important future direction is to apply EvoMapX to a broader range of population-based optimization algorithms, including both recently proposed metaheuristics (Rajwar et al., 2023) and canonical evolutionary algorithms such as the $(\mu + \lambda)$ Evolution Strategy (ES) (Urbańczyk et al., 2024), to complement the current analysis. While CS was included in this study to represent a highly exploratory, diversity-focused search mechanism, we acknowledge that standard ES variants would offer a more widely accepted baseline and help further validate the generality and applicability of the EvoMapX framework across established optimization paradigms.

Human and animal rights

This article does not contain any studies with human participants or animals performed by any of the authors.

Data availability

The benchmark functions used in this study are from the IEEE Congress on Evolutionary Computation (CEC) 2021 Competition on Single Objective Bound Constrained Numerical Optimization (DOI: [10.13140/RG.2.2.36130.66245](https://doi.org/10.13140/RG.2.2.36130.66245)).

Data availability

Data will be made available on request.

CRedit authorship contribution statement

Bilal H. Abed-Alguni: Conceptualization, Methodology, Software, Writing – original draft, Writing – review & editing, Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

- Abed-alguni, B. H., & Al-Jarah, S. H. (2024). IBJA: An improved binary DJaya algorithm for feature selection. *Journal of Computational Science*, 75, 102201.
- Abed-Alguni, B. H., & Alawad, N. A. (2021). Distributed grey wolf optimizer for scheduling of workflow applications in cloud environments. *Applied Soft Computing*, 102, 107113.
- Abed-alguni, B. H., Alzaboun, B. M., & Alawad, N. A. (2024). BOC-PDO: An intrusion detection model using binary opposition cellular prairie dog optimization algorithm. *Cluster Computing*, 27(10), 14417–14449.
- Abed-alguni, B. H., & Paul, D. (2022). Island-based cuckoo search with elite opposition-based learning and multiple mutation methods for solving optimization problems. *Soft Computing*, 26(7), 3293–3312.
- Agrawal, P., Abutarboush, H. F., Ganesh, T., & Mohamed, A. W. (2021). Metaheuristic algorithms on feature selection: A survey of one decade of research (2009–2019). *IEEE Access*, 9, 26766–26791.
- Ahn, K., & Cutkosky, A. (2024). Adam with model exponential moving average is effective for nonconvex optimization. *Advances in Neural Information Processing Systems*, 37, 94909–94933.
- Al-qudah, N. E. A., Abed-alguni, B. H., & Barhoush, M. (2024). Bi-objective feature selection in high-dimensional datasets using improved binary chimp optimization algorithm. *International Journal of Machine Learning and Cybernetics*, 15(12), 6107–6148.
- Alawad, N. A., Abed-alguni, B. H., Al-Betar, M. A., & Jaradat, A. (2023). Binary improved white shark algorithm for intrusion detection systems. *Neural Computing and Applications*, 35(26), 19427–19451.
- Alawad, N. A., Abed-alguni, B. H., & Shakhathreh, A. M. (2025). Ebao: An intrusion detection framework for wireless sensor networks using an enhanced binary aquila optimizer. *Knowledge-Based Systems*, 312, 113156.
- Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bénéttot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., Chatila, R., Herrera, F. (2020). Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58, 82–115.
- Beeramoole, P. B., Haque, M. M., Pinz, A., & Paz, A. (2022). A multi-objective optimization framework for assisted estimation of discrete choice models. In *Australasian transport research forum (ATRF)*, 43rd, 2022, Adelaide, South Australia, Australia.
- Bischl, B., Binder, M., Lang, M., Pielok, T., Richter, J., Coors, S., Thomas, J., Ullmann, T., Becker, M., Boulesteix, A.-L., Deng, D., Lindauer, M. (2023). Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(2), e1484.
- Chakraborty, S., Saha, A. K., Ezugwu, A. E., Agushaka, J. O., Zitar, R. A., & Abualigah, L. (2023). Differential evolution and its applications in image processing problems: A comprehensive review. *Archives of Computational Methods in Engineering*, 30(2), 985–1040.
- Cheng, L., Zhou, J.-X., Hu, X., Mohamed, A. W., & Liu, Y. (2024). Adaptive differential evolution with fitness-based crossover rate for global numerical optimization. *Complex & Intelligent Systems*, 10(1), 551–576.
- Corrente, S., Greco, S., Matarazzo, B., & Słowiński, R. (2024). Explainable interactive evolutionary multiobjective optimization. *Omega*, 122, 102925.
- Črepinšek, M., Mernik, M., & Liu, S.-H. (2011). Analysis of exploration and exploitation in evolutionary algorithms by ancestry trees. *International Journal of Innovative Computing and Applications*, 3(1), 11–19.
- Das, S., Suganthan, P. N. et al. (2021). Problem Definitions and Evaluation Criteria for the CEC 2021 Special Session and Competition on Single Objective Bound Constrained Numerical Optimization. Technical Report Nanyang Technological University. <https://www.ntu.edu.sg/home/EPNSugan/Public%20files/cec2021/cec2021-technical-report-v1.pdf>.

- Daviran, M., Maghsoudi, A., & Ghezlbash, R. (2025). Optimized AI-MPM: Application of PSO for tuning the hyperparameters of SVM and RF algorithms. *Computers & Geosciences*, 195, 105785.
- Dubey, R. (2024). Explainable AI assisted evolutionary search of engineering designs. In *2024 IEEE 36th international conference on tools with artificial intelligence (ICTAI)* (pp. 447–453). IEEE.
- Fyvie, M., McCall, J. A. W., Christie, L. A., Brownlee, A. E. I., & Singh, M. (2025). Towards explainable metaheuristics: Feature extraction from trajectory mining. *Expert Systems*, 42(1), e13494.
- Gad, A. G. (2022). Particle swarm optimization algorithm and its applications: A systematic review. *Archives of Computational Methods in Engineering*, 29(5), 2531–2561.
- Guerrero-Luis, M., Valdez, F., & Castillo, O. (2021). A review on the cuckoo search algorithm. *Fuzzy Logic Hybrid Extensions of Neural and Optimization Algorithms: Theory and Applications*, 940 (pp. 113–124).
- Hein, D., Hentschel, A., Runkler, T., & Udluft, S. (2017). Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies. *Engineering Applications of Artificial Intelligence*, 65, 87–98.
- Hosseini, S. M., Shirvani, M. H., & Motameni, H. (2024). Multi-objective discrete cuckoo search algorithm for optimization of bag-of-tasks scheduling in fog computing environment. *Computers and Electrical Engineering*, 119, 109480.
- Huang, Y., Zhou, Y., Zhao, H., Fang, L., Riedel, T., & Beigl, M. (2024). ExTea: An evolutionary algorithm-based approach for enhancing explainability in time-series models. In *Joint European conference on machine learning and knowledge discovery in databases* (pp. 429–446). Springer.
- Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic algorithm: Past, present, and future. *Multimedia Tools and Applications*, 80, 8091–8126.
- Kora, P., & Yadlapalli, P. (2017). Crossover operators in genetic algorithms: A review. *International Journal of Computer Applications*, 162(10).
- Li, W., Chen, Y., Dong, Y., & Huang, Y. (2024). A solution potential-based adaptation reference vector evolutionary algorithm for many-objective optimization. *Swarm and Evolutionary Computation*, 84, 101451.
- Liefooghe, A., Verel, S., Lacroix, B., Zăvoianu, A.-C., & McCall, J. (2021). Landscape features and automated algorithm selection for multi-objective interpolated continuous optimisation problems. In *Proceedings of the genetic and evolutionary computation conference* (pp. 421–429).
- Lotif, M. (2014). Visualizing the population of meta-heuristics during the optimization process using self-organizing maps. In *2014 IEEE congress on evolutionary computation (CEC)* (pp. 313–319). IEEE.
- Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30, 4768–4777.
- Ma, L., Cheng, S., & Shi, Y. (2020). Enhancing learning efficiency of brain storm optimization via orthogonal learning design. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(11), 6723–6742.
- Ma, L., Huang, M., Yang, S., Wang, R., & Wang, X. (2021). An adaptive localized decision variable analysis approach to large-scale multiobjective and many-objective optimization. *IEEE Transactions on Cybernetics*, 52(7), 6684–6696.
- Manazir, A., & Raza, K. (2021). Comparative evaluation of genetic operators in cartesian genetic programming. In *International conference on intelligent systems design and applications* (pp. 765–774). Springer.
- Nikolikh, A., Cenikh, G., Trajanov, R., Kostovska, A., & Eftimov, T. (2025). Explainable landscape analysis. *Explainable AI for Evolutionary Computation*, (pp. 41–65).
- Ochoa, G., Malan, K. M., & Blum, C. (2021). Search trajectory networks: A tool for analysing and visualising the behaviour of metaheuristics. *Applied Soft Computing*, 109, 107492.
- Osaba, E., Carballedo, R., Diaz, F., Onieva, E., De La Iglesia, I., & Perallos, A. (2014). Crossover versus mutation: A comparative analysis of the evolutionary strategy of genetic algorithms applied to combinatorial optimization problems. *The Scientific World Journal*, 2014(1), 154676.
- Parkavi, R., Karthikeyan, P., & Abdullah, A. S. (2024). Enhancing personalized learning with explainable AI: A chaotic particle swarm optimization based decision support system. *Applied Soft Computing*, 156, 111451.
- Prager, R. P., & Trautmann, H. (2024). Pflacco: Feature-based landscape analysis of continuous and constrained optimization problems in python. *Evolutionary Computation*, 32(3), 211–216.
- Prity, F. S., Gazi, M. H., & Uddin, K. M. A. (2023). A review of task scheduling in cloud computing based on nature-inspired optimization algorithm. *Cluster Computing*, 26(5), 3037–3067.
- Quach, L.-D., Quoc Khang, N., Thai-Nghe, N., & Nguyen, C.-N. (2025). XAI-BO: An architecture using Grad-CAM technique to evaluate Bayesian optimization algorithms on deep learning models. *Journal of Information and Telecommunication*, 9 (pp. 1–22).
- Rajwar, K., Deep, K., & Das, S. (2023). An exhaustive review of the metaheuristic algorithms for search and optimization: Taxonomy, applications, and open challenges. *Artificial Intelligence Review*, 56(11), 13187–13257.
- Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). “Why should I trust you?” Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 1135–1144).
- Schwalbe, G., & Finzel, B. (2024). A comprehensive taxonomy for explainable artificial intelligence: A systematic survey of surveys on methods and concepts. *Data Mining and Knowledge Discovery*, 38(5), 3043–3101.
- Sekyere, Y. O. M., Effah, F. B., & Okyere, P. Y. (2024). An enhanced particle swarm optimization algorithm via adaptive dynamic inertia weight and acceleration coefficients. *Journal of Electronics and Electrical Engineering*, 3 (pp. 53–67).
- Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2020). Grad-CAM: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128, 336–359.
- van Stein, N., Vermetten, D., Kononova, A. V., & Bäck, T. (2024). Explainable benchmarking for iterative optimization heuristics. arXiv preprint arXiv:2401.17842.
- Urbaczycz, A., Kucaba, K., Wojtulewicz, M., Kisiel-Dorohinicki, M., Rutkowski, L., Duda, P., Kacprzyk, J., Chong, S. Y., Yao, X., & Byrski, A. (2024). ($\mu + \lambda$) Evolution strategy with socio-cognitive mutation. *Journal of Automation, Mobile Robotics and Intelligent Systems*, 18 (pp. 1–11).
- Wang, Y.-C., & Chen, T. (2024). Adapted techniques of explainable artificial intelligence for explaining genetic algorithms on the example of job scheduling. *Expert Systems with Applications*, 237, 121369.
- Wei, J., & Niu, H. (2022). A ranking-based adaptive cuckoo search algorithm for unconstrained optimization. *Expert Systems with Applications*, 204, 117428.
- Xue, X., Mei, Y., Zhao, B., & Zhang, M. (2025). Adaptive similarity feature construction for ontology matching via multi-layer hybrid genetic programming. *IEEE Transactions on Evolutionary Computation*.
- Yates, W. B., Keedwell, E. C., & Kheiri, A. (2025). Explainable optimisation through online and offline hyper-heuristics. *ACM Transactions on Evolutionary Learning*, 5(2), 1–29.
- Zeng, R., Zeng, C., Wang, X., Li, B., & Chu, X. (2022). Incentive mechanisms in federated learning and a game-theoretical approach. *IEEE Network*, 36(6), 229–235.
- Zhou, R., Bacardit, J., Brownlee, A. E. I., Cagnoni, S., Fyvie, M., Iacca, G., McCall, J., van Stein, N., Walker, D. J., & Hu, T. (2024). Evolutionary computation and explainable AI: A roadmap to understandable intelligent systems. *IEEE Transactions on Evolutionary Computation*.
- Zou, F., Chen, D., Liu, H., Cao, S., Ji, X., & Zhang, Y. (2022). A survey of fitness landscape analysis for optimization. *Neurocomputing*, 503, 129–139.