Performance Assessment of Evolutionary Algorithms in Dynamic Optimization Benchmark Functions

Josué Serrano Ramos[1*], Ana Evelyn Vázquez Pérez[1*], Aldo Escamilla Resendiz[1*] and Daniel Molina Pérez[2*]

[1*]Department of Artificial Intelligence, Escuela Superior de Cómputo, Instituto Politécnico Nacional, Av. Juan de Dios Bátiz s/n, Col. Nueva Industrial Vallejo, Ciudad de México, 07738, CDMX, México.
[2]Department of Computation, CINVESTAV-IPN, Av. Juan de Dios Bátiz s/n, Col. Nueva Industrial Vallejo, Ciudad de México, 07738, CDMX, México.

*Corresponding author(s). E-mail(s): josuserram@gmail.com; evelyn.vazquez070898@gmail.com; resendizae318@gmail.com; danielmolinaperez90@gmail.com;

Abstract

This paper assesses the performance of four evolutionary algorithms on five dynamic benchmark functions: the Firefly Algorithm, Ant Colony Optimization for Continuous Domains (ACOR), a Multiploid Genetic Algorithm (MGA), and a multi-population approach to Particle Swarm Optimization (PSO). The algorithms were ranked based on solution quality and adaptability, and the results showed that MGA achieved superior performance, often recovering from changes in a single iteration. The findings confirm the significant advantage of implicit memory for tracking moving optima in dynamic environments.

Keywords: Evolutionary Algorithms, Dynamic Optimization Problems, Firefly Algorithm, Ant Colony Optimization for Continuous Domains (ACOR), Multiploid Genetic Algorithm (MGA), Particle Swarm Optimization (PSO).

1 Introduction

Evolutionary computation (EC) encompasses a class of optimization techniques inspired by natural evolution [1]. Evolutionary algorithms (EAs) are stochastic heuristic search algorithms used to optimize complex functions [2], inspired by the principles of natural selection and genetic inheritance. These algorithms evolve populations of candidate solutions through selection, mutation, and recombination, guided by fitness evaluations. Selection favors those individuals in the algorithm's population that represent solutions of higher quality, while crossover and mutation generate offspring from those individuals to advance the search [3].

Due to their adaptability, parallelism, and robustness, EAs are considered highly effective for solving complex, multimodal, and high-dimensional optimization problems [4]. With evolutionary algorithms, it has been possible to successfully tackle problems that could not be handled by conventional optimization techniques [5]. Today, EAs such as Genetic Algorithms, Evolution Strategies, and Differential Evolution have been successfully applied in diverse fields such as engineering design, logistics, bioinformatics, and machine learning [6].

Problems in the realistic realm often involve environments that change over time, and their objective function and constraints do not remain fixed; these are known as Dynamic Optimization Problems (DOPs) [7]. In dynamic optimization, notions of optimality are more complex, and it is necessary not only to measure the final result but also to evaluate the search process itself [8]. Dynamic environments are typically generated by translating the base function along a number of linear, circular, or random trajectories [9].

Addressing DOPs requires an optimization algorithm to not only locate an optimal solution of a given problem but also to track the changing optimal solution over time as the problem evolves [6]. Real-world examples of DOPs include financial portfolio management [2], modeling of ship trajectories [10], and production scheduling under varying demand [6].

EAs are effective for solving DOPs since they are able to maintain sufficient diversity for continuous adaptations to changes in the landscape [9]. However, for an EA to be successful in a dynamic environment, it must accomplish two goals: detecting a change in the search space, and efficiently responding to it [2]. Since EAs draw their inspiration from the principles of natural evolution, which is itself a stochastic and dynamic process, they are naturally suited for solving DOPs [5].

However, the main problem with standard EAs is that they eventually converge to an optimum and thereby lose the diversity necessary for efficiently exploring the landscape [5]. For that reason, many researchers have developed approaches within EAs to enhance their performance for DOPs, such as diversity schemes, memory schemes, multi-population schemes, prediction and anticipation schemes, and adaptive schemes [11].

Evaluating the effectiveness of EAs in dynamic environments is inherently more complex than in static scenarios. In a static context, it is enough to report the best solution achieved by the algorithm at the end of its execution. In contrast, in DOPs, reporting only these values is insufficient, since other aspects must be considered, such as how well the methods are able to detect problem changes and move to new promising areas of the search space as they appear, or how effectively they can track

existing good solutions as they shift through the search space [10]. Metrics include the offline error measure [6], the best-error-before-change measure, the optimization accuracy measure, distance-based measures, behavior-based performance measures [7], convergence speed after changes, among others [6].

This article provides a comprehensive review of EAs tailored to DOPs, analyzing their design and adaptation strategies as well as their tracking performance in environments with moving optima. The algorithms reviewed are Firefly, Ant Colony Optimization for Continuous Domains (ACOR), Multiploid Genetic Algorithm (MGA), and cooperative Particle Swarm Optimization (PSO).

This paper also examines the benchmark setup itself, analyzing the dynamic functions Moving Peaks, Ackley, Rastrigin, and Schwefel, each modified to introduce environmental volatility at specified intervals. The experimental results reveal that the Multiploid Genetic Algorithm (MGA) consistently outperformed all other methods, demonstrating superior solution quality and remarkable adaptability, often recovering in a single iteration after environmental changes. In contrast, the cooperative PSO with a multi-population approach showed the weakest performance due to its non-adaptive diversity structure. Finally, the FA and ACOR algorithms produced intermediate results, exhibiting premature convergence and exploration–exploitation imbalances in complex landscapes.

2 Evolutionary Algorithms in Dynamic Environments

2.1 Algorithm 1: Firefly Algorithm

This algorithm was first proposed by Cambridge scholar Xinshe Yang in 2008 [12]. The Firefly Algorithm (FA) is a random search algorithm based on the swarm intelligence metaheuristic, inspired by the bioluminescent communication and mating behavior of fireflies [13]. In the algorithm, the goal is for fireflies, which represent potential solutions, to move toward the best possible answer. Candidate fireflies have a unique brightness that corresponds to the quality of their solution and are attracted toward their brighter peers. The brightest fireflies move randomly through the search space because they cannot be attracted to any other fireflies, and the absolute brightness of the algorithm is expressed by the objective function value [12].

The FA has attracted attention due to its effectiveness in dealing with global optimization problems and for having a simple model, few parameters to adjust, and strong searchability [12]. However, in dynamic optimization problems, the standard FA struggles with premature convergence and the inability to adapt to shifting optima, since it can easily fall into local optima or suffer from low accuracy when solving higher-dimensional problems [12]. The proposed algorithm 1 addresses these limitations by implementing a re-injection of diversity when the search stagnates or environmental changes are detected.

At initialization, the algorithm tracks the best fitness value in a history buffer (Steps 1–4). When the changeFlag parameter is true, this indicates a change in the environment, and consequently 30% of the population is randomly reinitialized (Steps

6–7). The same reinjection occurs when the search stagnates for more than 10 iterations without improvement (Steps 9–11). This strategy allows the fireflies to escape local optima and resume exploration, an approach inspired by techniques proposed by Yang [14], Farahani [12], and Li [12]. Finally, the population is fully re-evaluated and its light intensity is recomputed, moving the fireflies toward brighter individuals based on distance, attractiveness, and a random perturbation factor (Steps 13–23).

Algorithm 1 Firefly Algorithm with 30% Exploratory Population

Require: initial population P, fitness values F, iteration index t, parameters, evaluation function $Eval$, environment parameters, changeFlag

Ensure: updated population P', updated fitness F', bestSolution, bestFitness

if $t = 1$ **or** no history **then**
    initialize bestFitnessHistory as empty
    Find current best solution and fitness
    Append bestFitness to bestFitnessHistory
end if
if changeFlag is true **then**
    Reinject 30% of the population with random solutions
    Re-evaluate fitness of entire population
else if $t > 10$ **and** no significant improvement in last 10 iterations **then** ▷ Stagnation detected
    Reinject 30% of the population with random solutions
    Re-evaluate fitness of entire population
end if
Compute light intensity for each firefly based on fitness
for each firefly i **do**
    for each firefly $j \neq i$ **do**
        if firefly j is brighter **then**
            Compute distance r between i and j
            Compute attractiveness $\beta = \beta_0 \cdot \exp(-\gamma r^2)$
            Compute random perturbation step
            Move firefly i toward j with attractiveness and random step
            Enforce lower and upper bounds
        end if
    end for
end for
Re-evaluate fitness of updated population
Update bestSolution and bestFitness
return P', F', bestSolution, bestFitness

2.2 Algorithm 2: Ant Colony Optimization for Continous Domains (ACOR)

The Ant Colony Optimization (ACO) algorithm is well-known for its ability to solve discrete optimization problems [15]. ACO uses a pheromone model to move ants toward promising areas within the search space, whose main components are solution construction and pheromone updating. However, Socha and Dorigo proposed an ACO variant, called ACOR, for continuous search spaces [16]. In ACOR, the central idea is to maintain an archive of the best solutions found so far, which serves as the pheromone model. New candidate solutions are then generated by sampling Gaussian probability distributions centered on the archived solutions, where the variance is proportional to the spread of the archive [17].

To balance exploration and exploitation, this mechanism uses solution quality or a ranking to prioritize sampling, while the search width and selection process maintain diversity, and the archive update guides the colony [17]. Other variants include IACOR and LIACOR, which have shown improved performance in solving real-world engineering optimization problems using random-walk selection or local search [15].

In Algorithm 2, the kernel width of the Gaussian distributions is temporarily increased by 15% to intensify exploration when an environmental change is detected, allowing the colony to search beyond the previous local neighborhood. First, the initial solution archive is constructed if no prior colony exists, replacing invalid fitness values with sentinel values and retaining the top k solutions after sorting (Steps 1–3). When the parameter changeFlag is true, an environmental change is indicated and all archived solutions are re-evaluated and re-sorted, while the kernel width is temporarily

expanded to encourage exploration (Steps 4–6). New candidates are then generated by Gaussian sampling around archive members (Steps 9–15), after which the population and fitness values are updated (Step 17-19), and the archive is refreshed (Step 20).

Algorithm 2 ACOR with 15% Proximity-Based Exploration

Require: population P, fitness F, iteration t, params, eval $Eval$, changeFlag

Ensure: updated P', F', bestSolution, bestFitness

if $t = 1$ **then**

    Build archive (colony) from P and F; sort and keep top k

    Precompute Gaussian selection weights; set kernel width w

end if

if changeFlag **then**

    Re-evaluate archive; resort; temporarily increase w

end if

Compute pairwise distances in archive

for each of $\lfloor 0.15\,k \rfloor$ replacements **do** ▷ 15% exploration

    Find closest pair; if distance $<$ τ, replace one with random solution (re-evaluate)

end for

for each ant in P **do**

    Select archive solution by Gaussian weights

    For each dimension: $\sigma \leftarrow w\times$ mean-distance; sample $x' \sim \mathcal{N}(\mu, \sigma)$; clip to bounds

    Evaluate x' and store in new population

end for

$P' \leftarrow$ new solutions; $F' \leftarrow$ their fitness

Update archive with P'; sort and keep best k

bestSolution $\leftarrow$ best archive member; bestFitness $\leftarrow$ its fitness

return P', F', bestSolution, bestFitness

2.3 Algorithm 3: Multiploid Genetic Algorithm (MGA)

MGAs employ a multiploid representation, which means that each solution candidate has multiple genotypes but only one phenotype [18]. The goal is to create an implicit memory scheme that transfers what has been learned so far to future generations. This design mimics biological polyploidy, which provides an implicit memory mechanism that can be reactivated when environmental changes occur. The MGA utilizes statistical inference methods, specifically the Bayesian Optimization Algorithm (BOA), to determine which genetic material is expressed and to exploit interactions between variables [19].

In this regard, our approach allows each individual to maintain multiple alleles per variable, from which a dominant allele is probabilistically selected for phenotype expression. Inspired by Oksuz's thesis [19], in this algorithm when environmental changes are detected, controlled noise is introduced into allele dominance probabilities, reactivating hidden diversity and promoting rapid adaptation. In Algorithm 3, dominance levels are perturbed with noise and re-normalized to reintroduce the above-mentioned diversity (Step 6). The expressed population is then sampled according to its dominance probabilities (Step 8), and new offspring are generated through crossover and mutation (Step 9). Finally, the algorithm records the best solution and fitness before returning the updated population (Steps 10–14).

Algorithm 3 Multiploid Genetic Algorithm with Dynamic Dominance

Require: population P, fitness F, iteration t, parameters, eval function $Eval$, changeFlag

Ensure: updated P', F', bestSolution, bestFitness

if $t = 1$ **then**
    Initialize multiploid array [$popSize \times numVariables \times numAlleles$]
    Initialize dominance levels uniformly across alleles
end if
if changeFlag **then** ▷ Environmental change
    Perturb dominance levels with noise and normalize
end if
Express population by sampling alleles using dominance levels
Generate offspring via crossover and mutation
Evaluate offspring fitness with $Eval$
Selection: combine parents and offspring, keep top $popSize$
Update multiploid pool with survivors' expressed values
bestSolution ← best individual; bestFitness ← its fitness
return P', F', bestSolution, bestFitness

2.4 Algorithm 4: Particle Swarm Optimization (PSO)

PSO is a stochastic, population-based search algorithm inspired by the collective behavior of flocks of birds and schools of fish, in which each particle represents a candidate solution and updates its position in the search space based on both its personal

best experience and that of its neighbors [20]. Particles explore the search space by constantly adjusting their positions and velocities, optimizing their solutions by interacting with neighboring particles and the global optimum [21]. To balance between exploration and exploitation, three components govern the movement of particles: inertia weight, and the cognitive and social learning factors [22].

In DOPs, however, the challenge is significant since the problems change over time, and PSO presents difficulties in two aspects: outdated memory when the environment shifts, and diversity loss due to convergence [14]. To address these issues, multiple enhancements have been studied, including diversity schemes, memory schemes, multipopulation schemes, and adaptive schemes [6].

In Algorithm 4, we employed a multi-population PSO framework that extends the standard swarm approach by dividing the population into multiple cooperative subpopulations, each with a distinct role. Subpopulations are smaller groups whose members work closely with each other to achieve better optimization performance through information sharing and exchange [21]. Many researchers have considered multipopulations as a means of enhancing the diversity of EAs to address DOPs, maintaining multiple swarms on different peaks [14]. The idea is that one subpopulation, or search group, chases new optima while the trackers preserve continuity after environmental shifts [23].

The cooperative PSO algorithm begins by checking for environmental changes (Step 1-2) and setting optimization mode and subpopulation parameters, including the ratio of tracking to searching groups (Steps 4–8). At initialization, the swarm is divided into the search and tracking subpopulations (Step 10) [22], and each group is

optimized independently using PSO with doubly linked neighborhoods and adaptive inertia weights (Steps 12–13). After independent updates, the algorithm performs experience sharing, injecting elite solutions from the tracking subpopulations into the searching ones to accelerate adaptation (Step 14).

Algorithm 4 Cooperative PSO with Tracking and Searching Subpopulations

Require: Pop, Fit, It, AlgoParams, EvalFun, Change

Ensure: Pop, Fit, BestSol, BestFit

1: **if** Change **then**
2: PrintEnvironmentChange()
3: **end if**
4: Max ← AlgoParams.maximize
5: NumSubPop ← AlgoParams.numSubPop
6: TrackRatio ← AlgoParams.trackRatio
7: NumTrack ← round(NumSubPop × TrackRatio)
8: NumSearch ← NumSubPop − NumTrack
9: **if** It = 0 **then**
10: $P_{search}, P_{track} \leftarrow InitSubpopulations(NumSearch, NumTrack)$
11: **end if**
12: OptimizeSubpops(P_{search}, DefaultPSO)
13: OptimizeSubpops(P_{track}, DefaultPSO)
14: ShareExperience(P_{track}, P_{search})
15: Pop, Fit, LocalBests ← MergeResults(P_{search})
16: BestSol, BestFit ← GlobalBest(LocalBests)
17: **if** Max **then**
18: Fit, BestFit ← InvertSigns(Fit, BestFit)
19: **end if**
20: **return** Pop, Fit, BestSol, BestFit

Note: DefaultPSO is a PSO strategy with doubly-linked neighborhoods and adaptive inertia weight w_m.

3 Methodology

3.1 Benchmark Functions

To assess the performance of the algorithms in DOPs, this article employs five benchmark functions widely recognized in evolutionary computation studies. These functions have diverse properties, making them truly useful for testing the reliability, efficiency, and robustness of the EAs presented [24]. The functions range from simple shifting landscapes to highly multimodal and rotating environments. Their dynamic variants were constructed by introducing changes in the location, shape, or orientation of optima at fixed intervals of 50 iterations.

- **Benchmark 1 – Moving Peaks (low dynamics):** A set of randomly generated peaks with controllable height and width. This function type belongs to the landscape generator class [25], where search landscapes are generated by randomly distributed Gaussians. In our setup, a single change in the vector space structure is introduced midway through the run, representing mild environmental dynamics. Since Gaussians are completely uncorrelated, the function is inherently more difficult than traditional benchmarks.
- **Benchmark 2 – Moving Peaks (periodic shifts):** In this variant, the peaks' centers move periodically with a step size, causing the global optimum to relocate after 50 iterations. This version of the function is explicitly advocated by Dieterich,

since it offers the additional advantage of markedly increased complexity and a broad tunability in search space characteristics [25].

- **Benchmark 3 – Dynamic Ackley Function:** A non-convex and multimodal function with many local optima, where the global optimum is located in a very small basin [26]. In the dynamic version, the global optimum shifts randomly while the entire search landscape rotates every 50 iterations.
- **Benchmark 4 – Dynamic Rastrigin Function:** In the classical Rastrigin function, the global optimum is at the origin (0,0). The search space for this function is typically restricted to $[-5.12, 5.12]$ and is known to have fewer minima compared to other functions such as Ackley [25]. In the dynamic version, the optimum is displaced randomly within the search space every 50 iterations.
- **Benchmark 5 – Rotating Schwefel 2.26 (Disjoint Islands):** The Schwefel 2.26 function is continuous, differentiable, separable, scalable, and multimodal [24]. Its multimodality produces isolated islands, making the benchmark challenging. Schwefel's function is known to present difficulties because it is less symmetric and has the global minimum at the edge of the search space [25]. Moreover, there is no overall guiding slope towards the global minimum as in Ackley's function, or a less extreme slope as in Rastrigin's. Consequently, this is the most challenging of the benchmark functions and the one that most closely resembles the difficulty of real-world problems.

3.2 Algorithm Parameters

Each algorithm was configured with parameters adapted for dynamic environments, balancing exploration and exploitation to improve responsiveness. The settings were based on prior studies in dynamic evolutionary computation [12–15, 17–24], and adjusted empirically for our benchmarks.

- **Firefly Algorithm (FA):** [12, 13] $B_0 = 1$; $\alpha = 0.2$; $\gamma = 1$; $\alpha_{scale} = 0.2$; reinjection of 30% population after stagnation (>10 iterations) or change detection.
- **Ant Colony Optimization for Continuous Domains (ACOR):** [15, 17] Archive size $k = 45$; selectivity $q = 0.3$; Gaussian width = 0.75; 15% of close solutions replaced randomly each iteration.
- **Multiploid Genetic Algorithm (MGA):** [18, 19] Crossover rate = 0.95; mutation rate = 0.1; alleles per gene = 4; dominance perturbed after change to reactivate diversity.
- **Multi-Population Cooperative PSO:** [20–22] Number of subpopulations = 4; tracking ratio = 0.5 (two subpopulations track environmental changes while two search for new optima); experience sharing = elites from tracking injected into search subpopulations.

3.3 Experiment Description

All algorithms were implemented in `MATLAB` following the pseudocode presented in Algorithms 1, 2, 3, and 4. The experimental protocol was defined as follows:

- **Population size:** 100 individuals for each algorithm.

- **Iterations:** 500 per run.
- **Runs:** 20 independent runs per algorithm-benchmark combination.
- **Evaluation metrics:** Offline error, best-so-far fitness, convergence behavior after change, and variance across runs.
- **Stopping criterion:** None; all runs executed for the fixed iteration budget.

Table 1 Experimental setup parameters used for assesed algorithms

Parameter	**Value**
Population size per algorithm	100
Number of iterations	500
Independent runs per setup	20
Stopping criterion	Fixed iterations (no early stopping)[1]
Environment change frequency	Every 50 iterations (Benchmarks 2–5); one mid-run change (Benchmark 1)
Evaluation metrics	Offline error, best-so-far fitness, convergence speed, variance

[1]All algorithms were executed until the maximum number of iterations; no adaptive termination condition was used.

4 Results

4.1 Evaluation Metrics

The performance of the algorithms was assessed using DOP's performance indicators related to fitness-errror and efficiency [7]. Quality was measured through *Average Best*

Pre-Change, *Average Average Pre-Change*, *Average Worst Pre-Change*, and *Average Deviation Standard Pre-Change*, which captures robustness of solutions before each environmental shift [11]. These metrics reflect the algorithm's capacity to remain close to the optimum even in the face of disturbances.

Adaptability was captured by the *Average Recovery Iterations*, which is the average number of iterations required for the algorithm to regain a competitive solution after a change [2]. These complementary metrics jointly assess both the stability and the reactivity of the algorithms.

4.2 Algorithm Performance

4.2.1 Firefly Algorithm (FA)

Table 2 shows the results for FA with 30% reinjection of the population after stagnation or environmental changes. Results show that FA adapted reasonably well in simple landscapes (Benchmarks 1–2), achieving recovery in 2 to 6 iterations. For instance, in Benchmark 1, the algorithm reached an average best pre-change value of 4.326799, with the worst solution being 4.936076. The standard deviation was low (0.303534), and recovery required 6 iterations, indicating stable though not immediate adaptation.

In Benchmark 2, results were very close to zero across all metrics (best: 0.000020, worst: 0.008346), and the algorithm recovered in just 2 iterations, reflecting its capacity to handle smooth, unimodal landscapes. However, performance deteriorated sharply in multimodal scenarios, such as Benchmark 3, where the best solution was unstable. Additionally, in Benchmark 4, the algorithm achieved a best of 0.009526 and a worst of 1.574578, with a moderate deviation (0.533295), recovering in 3 iterations. Finally,

in Benchmark 5, the best result was -307.245057 and the worst -124.780520, with very high dispersion ($\sigma = 56.984993$). Here recovery required 9 iterations, reflecting the difficulty of escaping local traps in fragmented landscapes.

These outcomes can be explained by the fact that the FA convergence curve is essentially linear and does not improve with additional iterations, which indicates that FA cannot escape local optima [12]. Even though our reinjection strategy mitigated this to some extent, it was insufficient in landscapes with many deceptive local optima.

4.2.2 Ant Colony Optimization for Continuous Domains (ACOR)

In Benchmark 1, ACOR achieved a best value of 4.322080, with low dispersion ($\sigma = 0.315635$) and recovery in 4 iterations, indicating a stable search in smooth unimodal landscapes. In Benchmark 2, solutions were very close to the optimum (best: 0.000003, worst: 0.004291), recovering in only 2 iterations. These results can be attributed to Gaussian sampling around elite solutions, which favors exploitation of promising regions [15].

However, performance declined in more complex environments. In Benchmark 3, for example, results ranged from a best of 0.615696 to a worst of 16.017607, with high variability ($\sigma = 6.289126$), and recovery required 5 iterations. This suggests difficulties in maintaining robustness in highly multimodal landscapes, where many local optima compete for pheromone reinforcement [15]. Similarly, in Benchmark 5, ACOR achieved a best of -372.732798 and a worst of -196.890480, with very high dispersion ($\sigma = 56.532546$), and recovery took 12 iterations, the slowest among benchmarks.

This behavior can be explained by the algorithm's reliance on archive-based solution generation. Since it maintains elite solutions and generates new candidates using Gaussian probability distributions centered on them, it promotes exploitation but limits long-distance exploration. This is why additional mechanisms such as Lévy flights or local search are often needed [15].

Our version, with a 15% reinjection of proximate solutions, was included to counteract archive stagnation. While this improved recovery in moderately dynamic cases (Benchmarks 1 and 2), it was insufficient in problems with deceptive local optima (Benchmarks 3 and 5).

4.2.3 Multiploid Genetic Algorithm (MGA)

This algorithm was the best performer in dynamic environments, particularly due to its rapid adaptability and solution stability [19]. In Benchmark 1, the algorithm achieved near-optimal behavior, reaching a minimum average value of 0.000000, with a general average of 0.035916 and a worst-case of 0.358191. The low standard deviation (0.113236) reflects stability across executions. The algorithm recovered its performance in just one iteration, demonstrating a high level of adaptability.

In Benchmark 2, the best value was 0.000000, with an average of 0.000002, and a worst-case of only 0.000012. The standard deviation was again minimal (0.000005), highlighting the algorithm's precision. The system recovered its performance in a single iteration, which can be explained by MGA's multiploid structure, where multiple alleles remain latent until reactivated after an environmental change [18].

In Benchmark 3, the algorithm attained a best value of 0.118543, a very close average of 0.120936, and a worst value of 0.126520. In Benchmark 4, the best, average, and worst outcomes were 0.049361, 0.062799, and 0.075962, respectively. The standard deviation remained low (0.011708), and recovery occurred in one iteration. Finally, in Benchmark 5, the values were negative, with a best case of -365.078117, a worst of -241.255015, and an average of -336.985790. The standard deviation was significantly higher (42.507788), and recovery took three iterations, indicating a slower adaptation.

This outcome suggests that while MGA is highly effective in moderately dynamic environments, its performance can become more volatile in highly complex or high-dimensional landscapes. As noted by Leonard, in rapidly shifting environments, optimization algorithms often require adaptive parameter control or hybrid mechanisms to maintain stability [20].

4.2.4 Particle Swarm Optimization (PSO)

The multi-population approach demonstrated mixed performance across the dynamic benchmarks. For instance, in Benchmark 1, the algorithm reached a best value of 7.3931, with a mean of 7.9375 and a worst-case of 8.4146, requiring 3 iterations for recovery after an environmental change. While relatively stable, the solution quality was far from optimal, given that PSO is challenged by diversity loss due to population convergence, limiting its performance in highly dynamic conditions [21].

In Benchmark 2, the best solution reached was 0.00007, with a standard deviation of 0.00581. In Benchmark 3, the best solution was 3.7319, but the mean escalated to 12.5649 and the worst case peaked at 21.0446, requiring 7 iterations for adaptation.

As Yazdani [22] points out, benchmarks with higher complexity or volatility reduce the potential of the multi-population framework.

Benchmark 4 proved to be more consistent, with the best value being 0.676358 and the average 2.349877. The worst result was 5.216153, and variability (1.387639) was moderate. Recovery was completed in 4 iterations, maintaining an acceptable level of adaptability. Finally, Benchmark 5 showed negative values, with the best case at -325.050923, the average at -231.300190, and the worst at -152.200474. In this case, recovery was slower than in the other benchmarks, taking an average of 7 iterations.

4.3 General Performance Assesment

The experimental evaluation shows that MGA is consistently the strongest algorithm in both quality and adaptability, followed by FA and ACOR, which produced reasonable but less consistent results. Finally, multi-population PSO exhibited the weakest performance, characterized by lower solution quality and slower adaptation.

The top performer, MGA, achieved exceptional performance, often recovering from environmental changes in a single iteration and finding solutions extremely close to the global optimum. This can be explained by the algorithm's use of a multiploid chromosome structure, which serves as a powerful implicit memory mechanism [18]. This structure encompass a memory scheme that transfers what has been learned so far to future generations. As noted by Nguyen [11], memory schemes offer a clear advantage in solving DOPs, since they allow for almost instantaneous adaptation. When a change occurs, the MGA can alter allele dominance probabilities, reactivating hidden diversity and promoting rapid adaptation [19].

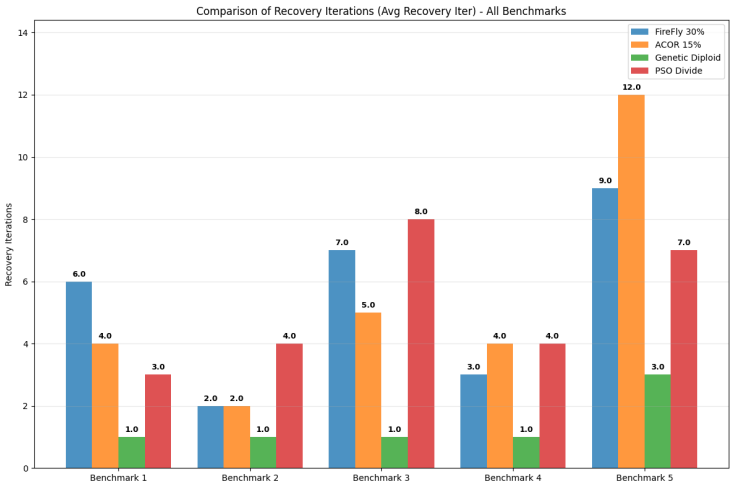

Fig. 1 Comparison of Recovery Iterations

The recovery curve for MGA on Benchmark 1 shows that its fitness drops and then immediately returns to its previous optimal level in a single iteration, demonstrating its ability to rapidly adapt whenever the environment changes. MGA does not need to re-explore the search space for the new optima but instead adjusts the dominance probabilities of its existing alleles, reactivating diversity. This contrasts with the other algorithms, which respond to change by initiating a new search process.

In contrast, PSO with population division demonstrated high variability and slower responsiveness, as the algorithm is highly dependent on the coordinated interaction of its components. The core challenge for multi-population algorithms is managing subpopulations effectively [22]. Moreover, Li et al. point out that traditional multi-population mechanisms often struggle to determine a suitable number of subpopulations [21]. Therefore, the number of subpopulations should be chosen carefully, taking into account the specific characteristics of the problem.

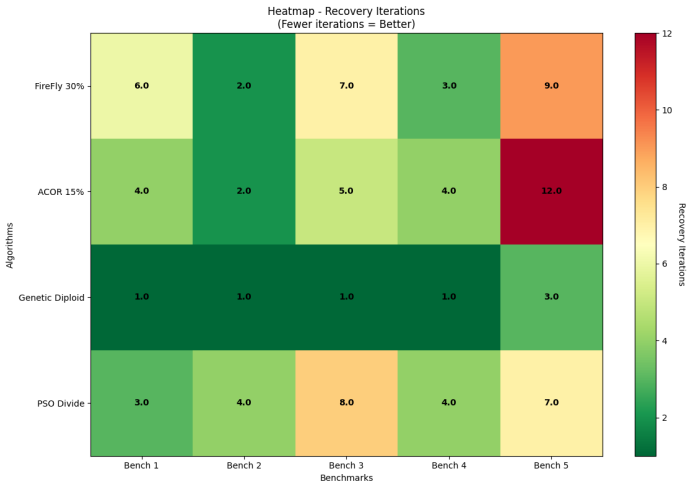

Fig. 2 Heatmap for Algorithm Adaptability Analysis

In the heatmap, which provides a dynamic overview of which algorithms perform well and which ones struggle, a visual of the Average Recovery Iterations is presented, a direct measure of an algorithm's ability to regain a competitive solution after a change. Aside from the top performer, Firefly and ACOR are represented by rows with a mix of yellow and light orange, indicating moderate but inconsistent recovery times. Our FA algorithm, although implementing a 30% population reinjection strategy to introduce diversity after a change, was not always sufficient for rapid recovery. On the other hand, ACOR's performance depends on its archive of the best solutions found so far, which makes its adaptability highly dependent on the nature of the environmental change. If a new optimum is near an archived solution, recovery will be fast. Conversely, if the optimum shifts to an unexplored region, the archive becomes a liability, slowing the search.

5 Discussions

This study conducted a comparative performance assessment of four EAs: the Firefly Algorithm (FA), Ant Colony Optimization for Continuous Domains (ACOR), a Multiploid Genetic Algorithm (MGA), and a multi-population approach to Particle Swarm Optimization (PSO). The primary goal was to evaluate their ability to find high-quality solutions and adapt efficiently to environmental changes. The results revealed a clear and consistent performance hierarchy, with MGA as the top performer and PSO as the weakest.

The performance of MGA is a consequence of its foundational design, which employs a multiploid chromosome structure that constitutes an implicit memory scheme. The redundancy in the alleles allows the algorithm to transfer useful information to subsequent generations. As a result, near-instantaneous recovery was observed. On the other hand, PSO struggled significantly due to the difficulty of determining a suitable number of subpopulations and managing diversity. The authors suggest that future studies explore dynamic adaptability of subpopulations to better address real-world problems.

The FA and ACOR showed strong performance but lacked consistency. For FA, the 30% population reinjection was insufficient to overcome the swarm's converged momentum. A dynamic step factor would likely be a more suitable mechanism to avoid falling into local optima. Similarly, ACOR's performance was tied to its archive of best solutions with its pheromone model, which showed poor performance when optima shifted to distant, unexplored areas of the search space, such as in the Schwefel benchmark. The 15% proximate solution replacement was insufficient as a diversity

mechanism, and other tools to facilitate the required global exploratory leaps are suggested for future works.

Another recommendation concerns the benchmark functions used in this study. Dieterich and Hartke argue that many commonly used benchmarks are not particularly challenging and were therefore modified for this analysis. However, there is an opportunity to create more challenging functions for a deeper and more demanding evaluation. Moreover, the overwhelming success of MGA may be explained by its ability to effectively exploit the structured, symmetric, and separable nature of these functions, while the failure of cooperative PSO may have been exacerbated by these relatively simple landscapes.

Finally, the clear strengths of MGA in memory management and the known exploratory power of multi-population methods suggest a promising direction for hybridization. Future studies could explore whether a combination of these algorithms—for instance, integrating the implicit memory of MGAs with the adaptive multi-population framework—would leverage the strengths of both approaches to better explore unforeseen optima, achieving a superior balance of exploitation and exploration.

References

[1] Yang, S., Nguyen, T.T., Li, C.: Evolutionary dynamic optimization: Test and evaluation environments. In: Yang, S., Yao, X. (eds.) Evolutionary Computation for Dynamic Optimization Problems. Studies in Computational Intelligence, vol. 490, pp. 3–37. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/

978-3-642-38416-5_1

[2] Morrison, R.W.: Designing Evolutionary Algorithms for Dynamic Environments. Natural Computing Series. Springer, ??? (2004). https://doi.org/10.1007/978-3-540-39863-2

[3] Rohlfshagen, P., Lehre, P.K., Yao, X.: Theoretical advances in evolutionary dynamic optimization. In: Yang, S., Yao, X. (eds.) Evolutionary Computation for Dynamic Optimization Problems. Studies in Computational Intelligence, vol. 490, pp. 221–240. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38416-5_9

[4] Yazdani, D., Cheng, R., Yazdani, D., Branke, J., Jin, Y., Yao, X.: A survey of evolutionary continuous dynamic optimization over two decades – part a. IEEE Transactions on Evolutionary Computation (2021) https://doi.org/10.1109/TEVC.2021.3060014

[5] Branke, J.: Evolutionary Optimization in Dynamic Environments. Genetic Algorithms and Evolutionary Computation. Springer, ??? (2001). https://doi.org/10.1007/978-1-4613-0131-2

[6] Nguyen, T.T., Yang, S., Branke, J., Yao, X.: Evolutionary dynamic optimization: Methodologies. In: Yang, S., Yao, X. (eds.) Evolutionary Computation for Dynamic Optimization Problems. Studies in Computational Intelligence, vol. 490, pp. 39–64. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38416-5_2

[7] Yazdani, D., Cheng, R., Yazdani, D., Branke, J., Jin, Y., Yao, X.: A survey of evolutionary continuous dynamic optimization over two decades – part b. IEEE Transactions on Evolutionary Computation (2021) https://doi.org/10.1109/TEVC.2021.3060012

[8] Rohlfshagen, P., Yao, X.: Evolutionary dynamic optimization: Challenges and perspectives. In: Yang, S., Yao, X. (eds.) Evolutionary Computation for Dynamic Optimization Problems. Studies in Computational Intelligence, vol. 490, pp. 65–84. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38416-5_3

[9] Bäck, T.: On the behavior of evolutionary algorithms in dynamic environments. In: Proceedings of the IEEE World Congress on Computational Intelligence, Evolutionary Computation, pp. 446–451 (1998). https://doi.org/10.1109/ICEC.1998.700106

[10] Cruz, C., González, J.R., Pelta, D.A.: Optimization in dynamic environments: A survey on problems, methods and measures. Soft Computing **15**(7), 1427–1448 (2011) https://doi.org/10.1007/s00500-010-0681-0

[11] Nguyen, T.T., Yao, X.: Evolutionary optimization on continuous dynamic constrained problems – an analysis. In: Yang, S., Yao, X. (eds.) Evolutionary Computation for Dynamic Optimization Problems. Studies in Computational Intelligence, vol. 490, pp. 193–217. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38416-5_8

[12] Li, Y., Zhao, Y., Shang, Y., Liu, J.: An improved firefly algorithm with dynamic self-adaptive adjustment. PLoS ONE **16**(10), 0255951 (2021) https://doi.org/10.1371/journal.pone.0255951

[13] Wang, W., Wang, H., Zhou, X., Zhao, J., Lv, L., Sun, H.: Dynamic step factor based firefly algorithm for optimization problems. In: 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), pp. 128–134 (2017)

[14] Yang, S., Li, C.: A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. IEEE Transactions on Evolutionary Computation (2010) https://doi.org/10.1109/TEVC.2010.2046667 . Manuscript received April 30, 2009; revised September 9, 2009, December 4, 2009, and February 3, 2010. Date of publication August 26, 2010; date of current version November 30, 2010.

[15] Omran, M.G.H., Al-Sharhan, S.: Improved continuous ant colony optimization algorithms for real-world engineering optimization problems. Engineering Applications of Artificial Intelligence **85**, 818–829 (2019) https://doi.org/10.1016/j.engappai.2019.08.009

[16] Socha, K., Dorigo, M.: Ant colony optimization for continuous domains. European Journal of Operational Research **185**(3), 1155–1173 (2008) https://doi.org/10.1016/j.ejor.2006.06.046

[17] Abdelbar, A.M., Salama, K.M.: Parameter self-adaptation in an ant colony algorithm for continuous optimization. IEEE Access (2019) https://doi.org/10.1109/ACCESS.2019.2896104 . Received December 1, 2018, accepted January 11, 2019, date of publication January 30, 2019, date of current version February 20, 2019.

[18] Gazioglu, E., Etaner-Uyar, A.S.: Experimental analysis of a statistical multiploid genetic algorithm for dynamic environments. Engineering Science and Technology, an International Journal **35**, 101173 (2022) https://doi.org/10.1016/j.jestch.2022.101173

[19] Öksüz: Multiploid genetic algorithms for multi-objective turbine blade aerodynamic optimization. Ph.d. thesis, Middle East Technical University (December 2007). Supervisor: Prof. Dr. İ. Sinan Akmandor

[20] Leonard, N., Yang, S.: Cooperative particle swarm optimization in dynamic environments. Swarm Intelligence Symposium (SIS), 172–179 (2013)

[21] Li, F., Yue, Q., Liu, Y., Ouyang, H., Gu, F.: A fast density peak clustering based particle swarm optimizer for dynamic optimization. Expert Systems With Applications **236**, 121254 (2024) https://doi.org/10.1016/j.eswa.2023.121254

[22] Yazdani, D., Yazdani, D., Blanco-Davis, E., Nguyen, T.T.: A survey of multi-population optimization algorithms for tracking the moving optimum in dynamic environments. Journal of Membrane Computing **7**, 85–107 (2025) https://doi.org/10.1007/s41965-024-00163-y

[23] Li, C., Nguyen, T.T., Yang, M., Mavrovouniotis, M., Yang, S.: An adaptive multi-population framework for locating and tracking multiple optima. IEEE Transactions on Evolutionary Computation (2016) https://doi.org/10.1109/TEVC.2015.2504383 . Early access

[24] Jamil, M., Yang, X.-S.: A literature survey of benchmark functions for global optimization problems. Int. Journal of Mathematical Modelling and Numerical Optimisation **4**(2), 150–194 (2013) https://doi.org/10.1504/IJMMNO.2013.055204

[25] Dieterich, J.M., Hartke, B.: Empirical review of standard benchmark functions using evolutionary global optimization. 1207.4318v1.pdf (Preprint/Working Paper) (2012). Bernd Hartke is the corresponding author [1]; affiliations listed are Institut für Physikalische Chemie, Christian-Albrechts-Universität, Kiel, Germany [1].

[26] Plevris, V., Solorzano, G.: A collection of 30 multidimensional functions for global optimization benchmarking. Data **7**, 46 (2022) https://doi.org/10.3390/data7040046

Table 2 Results Table

Algorithm	Benchmark	Avg Best	Avg Avg	Avg Worst	Avg Std	Recovery Iter	Time (s)
FA	1	4.3268	4.6520	4.9361	0.3035	6	21.70
	2	0.00002	0.00163	0.00835	0.00282	2	142.55
	3	1.9493	10.7963	19.5597	7.1031	7	44.18
	4	0.3954	4.0308	9.4337	4.0339	6	36.02
	5	64.8217	327.5693	649.3185	221.8431	9	38.52
ACOR	1	4.3221	4.6368	4.9361	0.3156	4	21.43
	2	0.000003	0.00081	0.00429	0.00148	2	229.78
	3	0.6157	7.7536	16.0176	6.2891	5	45.06
	4	0.2994	3.4147	8.4310	3.6366	7	39.26
	5	54.3209	312.4286	689.0379	228.9923	12	40.85
MGA	1	4.3220	4.6349	4.9360	0.3170	1	22.01
	2	0.000001	0.00025	0.00128	0.00036	1	167.23
	3	0.0271	3.2127	7.1144	3.0191	2	41.02
	4	0.0008	2.7063	7.1650	2.7926	2	34.97
	5	5.9324	196.0138	562.4792	178.2431	3	36.84
PSO	1	7.3931	7.9275	8.4146	0.5083	3	21.11
	2	0.00007	0.00453	0.01967	0.00581	3	195.34
	3	3.7319	12.5649	21.0446	7.5622	7	42.93
	4	0.7035	5.1404	10.0745	4.2101	6	36.77
	5	78.9186	347.5257	648.9114	229.4877	8	38.26

Values correspond to averages over 20 independent runs with population size of 100 and 500 iterations. Recovery iteration indicates the average number of generations needed after a change to reach competitive performance.