This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

The Firefighter Algorithm for Optimization Problems

M.Z. Naser[1,2], Ahmad Z. Naser[3]
[1]School of Civil & Environmental Engineering and Earth Sciences (SCEEES), Clemson University, USA
[2]Artificial Intelligence Research Institute for Science and Engineering (AIRISE), Clemson University, USA
E-mail: mznaser@clemson.edu, Website: www.mznaser.com
[3]Department of Mechanical Engineering, University of Manitoba, Winnipeg R3T 5V6, Canada
Email: naser@umanitoba.ca

Abstract

This paper presents the Firefighter Optimization (FFO) algorithm as a new metaheuristic for optimization problems that stems inspiration from the collaborative strategies often deployed by firefighters in firefighting activities. Such strategies include adaptive response to changing conditions, coordination among multiple firefighters (i.e., agents) to converge on a common goal, balancing exploration and exploitation by maintaining diversity within the search space and adapting its parameters to navigate complex landscapes. To evaluate the performance of FFO, extensive experiments were conducted, wherein the FFO was examined against 13 commonly used optimization algorithms, namely, the Ant Colony Optimization (ACO), Bat Algorithm (BA), Biogeography-Based Optimization (BBO), Flower Pollination Algorithm (FPA), Genetic Algorithm (GA), Grey Wolf Optimizer (GWO), Harmony Search (HS), Particle Swarm Optimization (PSO), Simulated Annealing (SA), Tabu Search (TS), and Whale Optimization Algorithm (WOA), and across 24 benchmark functions, as well as 10 standard functions and 4 real engineering problems from the CEC 2020 suite. The results demonstrate that FFO achieves comparative performance and, in some scenarios, outperforms commonly adopted optimization algorithms in terms of the obtained fitness, time taken for exaction, and research space covered per unit of time. More specifically, FFO ranked first in the Distance per Unit Time metric and maintained a top 5 performance in higher dimensions (i.e., 20D and 50D).

Keywords: Optimization; Benchmarking; Metaheuristics.

1.0 Introduction

Metaheuristics play a large role in the domain of optimization. These algorithms have been renowned for their efficacy in tackling complex and multidimensional problems that can typically be beyond the reach of traditional methods [1]. Metaheuristics are distinguished by their flexibility and robustness, making them particularly suitable for problems where the solution landscape is rugged or poorly defined, such as those expected across diverse disciplines, ranging from engineering and logistics to economics and data science [2]. For example, metaheuristics' versatility enables their engineering application to optimize design parameters for complex systems. In logistics, metaheuristics can help solve scheduling and routing problems. Similarly, metaheuristics can prove beneficial in finance systems to optimize investment portfolios, to name a few.

Metaheuristics can be defined as high-level strategies that coordinate simpler investigative methodologies to explore and exploit the search space efficiently [3]. These strategies are characterized by their reliance on processes that promote some form of balance between

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

exploration of the search space to avoid entrapment in local optima and exploitation mechanisms that refine promising areas to converge toward global optima [4]. Thus, metaheuristics can be thought of as generic and adaptable to a broad spectrum of problems with minimal modification. This can be advantageous to problem-specific algorithms.

Metaheuristics are broadly classified into two categories based on their approach: single-solution based and population-based. Single-solution metaheuristics iteratively improve a single candidate solution. These algorithms capitalize on the collective intelligence of the population to explore and exploit the search space more broadly. Some such metaheuristics include Simulated Annealing (SA) and Tabu Search (TS) [5]. This group of algorithms employs mechanisms to escape local optima, like probabilistic acceptance of worse solutions in SA or using memory structures in TS to avoid revisiting previously explored areas of the search space. The efficiency of single-solution metaheuristics is tied to their ability to fine-tune a solution through mechanisms like adaptive neighborhood searches and intensification strategies that home in on promising regions of the search space [5].

On the other hand, population-based metaheuristics evolve a group (i.e., population) of solutions to leverage interactions within this group to explore and exploit the search space collectively [6]. Some examples under this group include Genetic Algorithms (GA) and Particle Swarm Optimization (PSO). Such algorithms can maintain diversity within the population and avoid premature convergence to suboptimal solutions. For instance, GA employs operators such as crossover and mutation to introduce variability and ensure robust exploration, while PSO mimics social behaviors of swarms [7]. These two examples can balance exploration and exploitation through the dynamic adjustment of particle velocities based on individual and collective experiences. This population-based approach enables these algorithms to effectively navigate complex, multimodal optimization landscapes by simultaneously exploring multiple regions of the search space, increasing the likelihood of finding a global optimum. It goes without saying that hybrid approaches that combine elements from both single-solution and population-based metaheuristics have emerged as a means to leverage the strengths of both strategies [8]. These hybrid methods often incorporate mechanisms like adaptive parameter control, multi-stage search processes, and cooperative co-evolution to enhance performance on a wide range of optimization problems [9].

Similarly, metaheuristics can also be classified based on their source of inspiration. Some metaheuristics are nature-inspired algorithms, wherein they are inspired by natural phenomena, biological processes, or behaviors observed in animals/plants. These nature-inspired algorithms often mimic survival mechanisms, evolutionary processes, or social behaviors. For example, GA, PSO, and Ant Colony Optimization (ACO) are representative of such classification. Then, some metaheuristics draw inspiration from human behaviors, the physical world, and societal structures. For instance, both TS and SA incorporate actions and behaviors seen in humans (i.e., memory) and physical processes (i.e., annealing in metallurgy) [10].

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

Despite their versatility and ease of use, metaheuristics can suffer from challenges [11]. One such challenge is drawing a balance between exploration and exploitation capabilities. These capabilities can be crucial for avoiding premature convergence and ensuring the global optimum is reached. Moreover, the stochastic nature of these algorithms often requires multiple runs to achieve consistent results, which can be computationally expensive [12]. Fortunately, the field of metaheuristics continues to grow in response to addressing increasingly complex problems [13,14]. One notable trend is the hybridization of metaheuristic algorithms, where two or more distinct strategies are combined to exploit their complementary strengths. For instance, hybrid algorithms might combine one algorithm's explorative power with another's intensive exploitation capabilities. Such hybridization can potentially yield solutions that are both diverse and precise.

Further, recent studies have proposed novel metaheuristic algorithms such as the Improved Crowding Particle Algorithm (I-CPA) and the Tree Seed Algorithm (TSA) [15]. For instance, I-CPA has been applied to optimize engineering design problems with multi-objectives and complex constraints, improving the balance between exploration and exploitation [16]. Similarly, TSA, inspired by tree seed dispersal mechanisms, has been successfully used in solving scheduling and resource allocation problems [17]. These advancements highlight the ongoing evolution and diversification of metaheuristic algorithms and showcase their adaptability to various domains. In addition to these recent advancements, other metaheuristics such as the Harris Hawks Optimization (HHO) and the Arithmetic Optimization Algorithm (AOA) have gained attention due to their robust performance across a variety of domains [18,19]. HHO, inspired by the cooperative hunting strategy of Harris hawks, has been applied to challenging problems, including engineering, medical, data mining and clustering [20]. AOA, based on arithmetic operations, has also shown success in solving complex mathematical and engineering optimization problems. These emerging algorithms further underline the potential of metaheuristics in addressing diverse optimization challenges, contributing to a growing repository of tools designed to tackle specific industrial and research needs [19].

Still, as computational challenges grow and the need for efficient optimization strategies intensifies, the role of metaheuristics becomes increasingly important and warranted. The ability of metaheuristics to adapt and provide feasible and efficient solutions. The above motivates this work wherein we propose a novel algorithm, Firefighter Optimization (FFO), for optimization problems that could be applied in various areas (e.g., a general purpose algorithm). FFO is motivated by the strategies and tactics employed by firefighters, such as the dynamic distribution of resources, adaptive responses to evolving conditions, and coordinated efforts among multiple firefighters (agents) to achieve a unified objective. FFO also balances exploration and exploitation by maintaining diversity within the search space and adjusting its parameters to navigate complex optimization challenges. A number of comparative experiments, supplemented with various metrics, were carried out to validate the effectiveness and competitiveness of FFO. More specifically, the performance of FFO and the other selected algorithms was examined across over

600 tests on a wide range of benchmark and test functions. Our experimental results demonstrate the effectiveness and competitiveness of FFO compared to state-of-the-art algorithms.

The rest of the paper is organized as follows: Section 2 presents a description of FFO and explains each component in detail. Sections 3 and 4 describe the aforementioned algorithms and 24 commonly used benchmarking test functions for various complexity levels. Finally, Sections 4 and 5 conclude the paper with a presentation of our comparative results and conclusions/findings learned from our analysis.

2.0 Description of the Firefighter Optimization (FFO) algorithm

This section describes the FFO in more detail (see flowchart in Fig. 1). We start with a general description and then dive into a more detailed analysis of FFO's functions.

2.1 General description

The Firefighter optimization (FFO) algorithm is inspired by the strategies and tactics used by firefighters to combat fires in real-world scenarios. In the face of a fire, firefighters must strategically allocate resources, decide when to focus on extinguishing the flames, and when to protect specific areas or perform rescue operations. More specifically, this optimization technique draws from various aspects of firefighting, including the dynamic allocation of resources, adaptive response to changing conditions, and coordination among multiple firefighters (i.e., agents) to achieve a common goal. Further, the FFO algorithm is designed to balance exploration and exploitation by maintaining diversity within the search space and adapting its parameters to navigate complex optimization landscapes. See Table 1 for a comparison of this algorithm against other commonly used in optimization.

Initialization: The algorithm starts by randomly initializing a population of agents within the specified bounds of the search space. Each agent represents a potential solution to the optimization problem.

Evaluation: Each agent's position is evaluated using the objective function, which measures the fitness or quality of the solution. The best agent, i.e., the one with the lowest fitness value for minimization problems, is identified as the global best solution.

Adaptive Local Search: The FFO algorithm employs an adaptive local search mechanism to refine the positions of the agents. This process involves generating perturbations around the current position of an agent and evaluating the new positions. The perturbations are adjusted adaptively based on the agent's performance and the iteration count. This local search helps agents escape local optima and explore the solution space more effectively.

Crossover and Mutation: To enhance diversity and promote exploration, the FFO algorithm incorporates crossover and mutation operators. The crossover operator allows agents to exchange information, creating new solutions by combining parts of two parent agents. The mutation

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

operator introduces random changes to an agent's position, providing an additional mechanism to explore new areas of the search space.

Perturbation Mechanism: When agents fail to improve over a certain number of iterations, a perturbation mechanism is triggered. This mechanism mimics the adaptive response of firefighters to changing conditions. Agents undergo a larger perturbation, guided by the global best agent, to move towards potentially better regions of the search space. The intensity of the perturbation increases with the number of unsuccessful iterations, allowing the algorithm to escape stagnation and continue searching for optimal solutions.

Adaptive Step Size: The step size used in the local search and perturbation mechanisms is adaptively adjusted based on the algorithm's progress. If the algorithm detects stagnation, the step size is increased to encourage exploration. Conversely, if the algorithm is converging towards a solution, the step size is reduced to fine-tune the search and improve solution accuracy.

Cooling Schedule: The FFO algorithm incorporates a cooling schedule inspired by simulated annealing. The temperature parameter, which controls the acceptance probability of worse solutions during the local search, is gradually reduced over iterations. This allows the algorithm to initially explore more freely and then gradually focus on exploitation as it approaches convergence.

Termination Criteria: The algorithm runs until one or more termination criteria are met. These criteria can include reaching a maximum number of iterations, exceeding a predefined number of iterations without improvement, or achieving a target fitness value. The termination criteria ensure that the algorithm does not run indefinitely and provides a solution within a reasonable time frame.

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

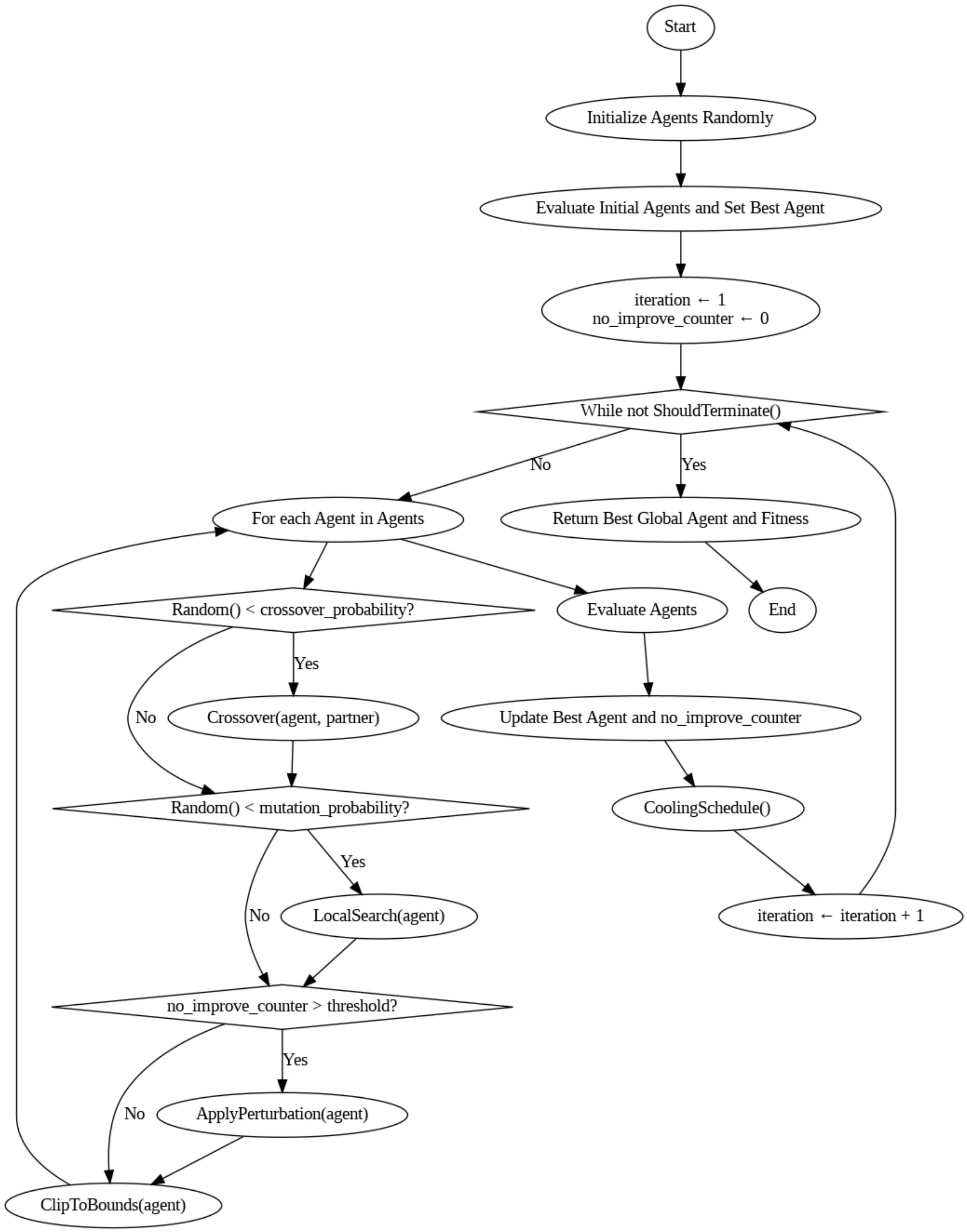


Fig. 1 Flowchart of FFO

2.2 Detailed description

A more detailed description of FFO's functions is provided herein.

Initialization (`__init__`)

The initialization function (`__init__`) of the `FirefighterOptimization` class sets up the algorithm's parameters, agents, and initial conditions for optimization. These parameters include the objective function, dimensions of the problem, number of agents, maximum iterations, no-improvement limit, and bounds for the search space. Additional parameters stemming from existing algorithms for crossover, mutation, simulated annealing, and perturbation control are also set up. The agents (i.e., solutions) are initialized randomly within the specified bounds, and the best global agent and fitness are identified at the start.

Initialization Process:

The process can be broken down into the following steps:

- <u>Parameter Setup</u>
- <u>Agents Initialization:</u> Agents are randomly distributed within the bounds:
- <u>Best Agent Identification:</u> The initial best global agent and its fitness are determined:
- <u>Other Parameters:</u> Additional parameters like step size, mutation rates, and counters are initialized:

Agent Evaluation (`evaluate_agents`)

The `evaluate_agents` function assesses the fitness of each agent within the initiated population. This function updates the best global fitness and agent if/when a better solution is found. This function calculates the fitness of each agent based on the objective function and updates the global best agent if an improved solution is identified. This evaluation process also guides the algorithm's search process toward better solutions. Mathematically, the evaluation involves computing the objective function for each agent and identifying the agent with the minimum fitness value.

- <u>Objective Function, $f(x)$:</u> The function to be minimized and is evaluated for each agent x_i.
- <u>Fitness Calculation:</u> For each agent x_i, the fitness is $f(x_i)$.
- <u>Best Fitness Update:</u> The global best fitness and agent are updated if a new minimum is found.
- <u>Evaluation Process</u>
- The evaluation process can be broken down into the following steps:
    - <u>Fitness Calculation</u>
    - <u>Best Agent Identification</u>
    - <u>Return Fitness Values</u>

Agent Update (`update_agents`)

The `update_agents` function is responsible for evolving the population of firefighters (i.e., agents) to maintain diversity through crossover, mutation, and perturbation operations. This function involves modifying agent positions in the solution space to explore new regions. As inspired by genetic algorithms, this function applies crossover to combine traits from different agents,

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

mutation[1] to introduce random changes, and perturbations to escape local optima. Mathematically, the update process includes the following key components:

- Crossover Operation
- Mutation Operation
- Perturbation Operation
- Update Process*:* The update process involves the following steps:
    - *Evaluate Agents*: The fitness of agents is then evaluated to update the global best fitness achieved:
    - *Crossover and Mutation*: Each agent undergoes crossover and mutation based on set probabilities:
    - *Perturbation*: If no improvement is observed for an extended period, this function applies perturbations:
    - *Boundary Check*: Agents are clipped within the bounds:
    - *Trajectory and Perturbation History*: Updates to agents are recorded for trajectory analysis

Local Search (local_search)

The local_search function refines an agent's position by exploring its neighborhood. This function helps agents escape local optima and find better solutions. This function involves making small adjustments to an agent's position to find a better solution in its vicinity. The process is guided by a temperature parameter, allowing the acceptance of worse solutions early on to escape local optima. As the temperature decreases, the search becomes more focused on local refinement – in a similar process to controlling fires. Mathematically, local search applies perturbations to an agent's position and evaluates the new positions. The acceptance of new positions is probabilistic, influenced by a temperature parameter.

- Perturbation
- Temperature
- Acceptance Probability
- Local Search Process: The process involves the following steps:
    - *Temperature Calculation:* The temperature is calculated based on the iteration:
    - *Local Best Initialization*: The current agent is considered the local best:
    - *Perturbation and Evaluation:* Small adjustments are made to the agent's position, and new positions are evaluated:
    - *Acceptance Check:* New positions are accepted based on fitness improvement or probabilistically:
    - *Return Local Best:* The refined local best position is then returned:

Perturbation Application (apply_perturbation)

Perturbation application is a strategy to escape local optima by making larger adjustments to agents' positions, which can be particularly useful when the algorithm stagnates. The apply_perturbation function introduces significant changes to agents' positions based on the global best agent when no improvement is observed. In mathematical notation, perturbation involves adjusting an agent's position towards the global best agent, scaled by an intensity factor, such that:

- Direction Vector:
- Perturbation:

[1] Further information on these operations will be provided in a subsequent section.

Perturbation Process

The process involves the following steps:

Direction Calculation: The direction vector from the agent to the global best is calculated:

Perturbation Calculation: A perturbation is applied based on the direction vector and intensity:

New Position Calculation: The agent's new position is calculated:

Crossover (`crossover`)

The `crossover` function combines parts of two agents to create new agents as a means of introducing diversity into the initialized population. This genetic algorithm-inspired method helps explore new solutions by recombining existing ones. Key components in this function include:

Crossover Point

New Agents

Crossover Process: The process involves the following steps:

Crossover Point Selection: A random crossover point is selected:

Agent Combination: New agents are created by combining segments of the parent agents:

Return New Agents: The new agents are returned:

Cooling Schedule (`cooling_schedule`)

The `cooling_schedule` function adjusts the step size based on the algorithm's progress, similar to the *cooling* option in simulated annealing. The cooling schedule involves gradually reducing the step size as the algorithm progresses (in a similar manner to controlling the fire toward the later stages of firefighting). This process allows for a finer search of the solution space over time, balancing exploration and exploitation. Mathematically, the cooling schedule involves updating the step size based on the number of iterations and the no-improvement counter such that:

Step Size Update:

The step size is reduced based on a cooling factor.

Cooling Process:

The process involves the following steps:

Step Size Adjustment: The step size is adjusted based on the no-improvement counter:

Execution Loop (`run`)

The `run` function controls the main execution loop of the algorithm, where agents are updated, evaluated, and the cooling schedule is applied until a termination condition is met. It also tracks the best solution and its fitness across iterations. The execution loop is the core of the optimization process. It iteratively updates agents, evaluates their fitness, applies the cooling schedule, and checks termination conditions. This loop continues until the optimization criteria are met, ensuring that the algorithm converges to an optimal solution. Mathematically, the execution loop involves iterating over the update and evaluation processes while tracking the best solution. Key components include:

Execution Process

The process involves the following steps:

Initialization: The fitness history and trajectory are initialized

Main Loop: The main loop iterates until the termination condition is met

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

Return Best Solution: The best global agent and fitness are returned

Termination Check (should_terminate**)**

The should_terminate function determines whether the algorithm should stop based on several conditions: *maximum iterations reached*, *no improvement for a set number of iterations*, or *achieving a target fitness level*. Thus, this function ensures that the algorithm terminates when further exploration is unlikely to yield better results.

<u>Iteration Check:</u> The algorithm stops if the maximum number of iterations is reached (i.e., $k \geq$ max_iter).
<u>No Improvement Check:</u> The algorithm stops if no improvement is observed for a set number of iterations.
<u>Target Fitness Check:</u> The algorithm stops if the target fitness level is achieved.
<u>Termination Process</u>
The process involves the following steps:

Condition Check: The termination condition is evaluated based on the iteration, no-improvement counter, and best global fitness
Return Condition: The termination condition is returned

Execution Time (get_execution_time**)**

The get_execution_time function calculates the total runtime of the algorithm as a means to present a measure for evaluating the time efficiency of the algorithm. This execution time is calculated as the difference between the end time and the start time.

Trajectory Tracking (get_trajectory**)**

The get_trajectory function records the sequence of solutions explored by the algorithm, which can be further analyzed to examine the search behavior and pathway through the solution space. The trajectory involves recording the positions of agents over time.

Total Distance Traveled (get_total_distance**)**

The get_total_distance function computes the cumulative distance traveled by the algorithm in the solution space. Such a distance can be an indicator of the algorithm's exploratory behavior and efficiency. The total distance traveled involves summing the Euclidean distances between consecutive positions of agents.

```
class FirefighterOptimization:
    def __init__(self, objective_func, dimension, num_agents=100, max_iter=500, no_improve_limit=30, bounds=(-
5.12, 5.12), step_size=1.0, crossover_probability=0.5, mutation_probability=0.1, initial_temp=100.0,
cooling_rate=0.95, verbose=False, use_additional_conditions=False, target_fitness=1e-5):
        self.objective_func = objective_func
        self.dimension = dimension
        self.num_agents = num_agents
        self.max_iter = max_iter
        self.no_improve_limit = no_improve_limit
        self.bounds = bounds
        self.agents = np.random.uniform(bounds[0], bounds[1], (num_agents, dimension))
        self.best_global_agent = np.copy(self.agents[np.argmin([self.objective_func(agent) for agent in self.agents])])
```

```
        self.best_global_fitness = self.objective_func(self.best_global_agent)
        self.step_size = step_size
        self.crossover_probability = crossover_probability
        self.mutation_probability = mutation_probability
        self.initial_temp = initial_temp
        self.cooling_rate = cooling_rate
        self.mutation_rates = np.full(self.num_agents, 0.1)
        self.no_improve_counter = 0
        self.iteration = 1
        self.fitness_history = []
        self.perturbation_history = []
        self.verbose = verbose
        self.use_additional_conditions = use_additional_conditions
        self.target_fitness = target_fitness
        self.trajectory = []
        self.start_time = None
        self.end_time = None

    def evaluate_agents(self):
        fitness = np.array([self.objective_func(agent) for agent in self.agents])
        best_index = np.argmin(fitness)
        if fitness[best_index] < self.best_global_fitness:
            self.best_global_fitness = fitness[best_index]
            self.best_global_agent = np.copy(self.agents[best_index])
            self.no_improve_counter = 0
        else:
            self.no_improve_counter += 1
        return fitness

    def update_agents(self):
        self.evaluate_agents()
        for i in range(self.num_agents):
            if np.random.rand() < self.crossover_probability:
                partner_index = np.random.randint(self.num_agents)
                self.agents[i], self.agents[partner_index] = self.crossover(self.agents[i], self.agents[partner_index])
            if np.random.rand() < self.mutation_probability:
                self.agents[i] = self.local_search(self.agents[i], i)
            if self.no_improve_counter > 50:
                self.agents[i] = self.apply_perturbation(self.agents[i], 0.1 + 0.02 * (self.no_improve_counter - 50))
            self.agents[i] = np.clip(self.agents[i], self.bounds[0], self.bounds[1])
            self.trajectory.append(np.copy(self.agents[i]))
            self.perturbation_history.append(self.agents[i])
```

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

```
    def local_search(self, agent, index):
        temp = self.initial_temp * (self.cooling_rate ** self.iteration)
        best_local = agent
        best_local_fitness = self.objective_func(agent)
        for _ in range(10 + 5 * (self.no_improve_counter // 100)):
            perturbation = np.random.normal(0, self.step_size * self.mutation_rates[index], self.dimension)
            candidate = best_local + perturbation
            candidate_fitness = self.objective_func(candidate)
            if candidate_fitness < best_local_fitness or np.random.rand() < np.exp((best_local_fitness -
candidate_fitness) / temp):
                best_local = candidate
                best_local_fitness = candidate_fitness
        return best_local

    def apply_perturbation(self, agent, intensity):
        direction = self.best_global_agent - agent
        perturbation = np.random.normal(0, intensity, self.dimension) * direction
        return agent + perturbation

    def crossover(self, agent1, agent2):
        crossover_point = np.random.randint(1, self.dimension)
        new_agent1 = np.concatenate((agent1[:crossover_point], agent2[crossover_point:]))
        new_agent2 = np.concatenate((agent2[:crossover_point], agent1[crossover_point:]))
        return new_agent1, new_agent2

    def cooling_schedule(self):
        if self.no_improve_counter > 50:
            self.step_size *= 0.98
        else:
            self.step_size *= 0.99

    def run(self):
        self.fitness_history = []
        self.trajectory = []
        self.iteration = 1
        self.start_time = time.time()
        while not self.should_terminate():
            self.update_agents()
            self.cooling_schedule()
            self.fitness_history.append(self.best_global_fitness)
            if self.verbose:
```

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

```
                print(f"Iteration {self.iteration}, Best Fitness {self.best_global_fitness}, No Improve Counter {self.no_improve_counter}")
            self.iteration += 1
        self.end_time = time.time()
        return self.best_global_agent, self.best_global_fitness, self.fitness_history

    def should_terminate(self):
        if self.use_additional_conditions:
            termination_condition = (
                self.iteration >= self.max_iter or
                self.no_improve_counter > self.no_improve_limit or
                self.best_global_fitness < self.target_fitness
            )
        else:
            termination_condition = self.iteration >= self.max_iter
        if termination_condition and self.verbose:
            print(f"Terminating: Iteration={self.iteration}, No Improve Counter={self.no_improve_counter}, Best Fitness={self.best_global_fitness}")
        return termination_condition

    def get_execution_time(self):
        return self.end_time - self.start_time

    def get_trajectory(self):
        return self.trajectory

    def get_total_distance(self):
        distance = 0
        for i in range(1, len(self.trajectory)):
            distance += np.linalg.norm(self.trajectory[i] - self.trajectory[i - 1])
        return distance
```

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

Table 1 Qualitative comparison between FFO and commonly used optimization algorithms

Feature/Aspect	FFO	ACO	BA	BBO	FPA	GA	GWO	HS	PSO	SA	TS	WOA
Inspiration	Firefighting strategies	Ant foraging behavior	Echolocation of bats	Biogeography concepts	Flower pollination	Natural evolution	Grey wolves' hunting	Musical harmony	Swarming behavior of birds/fish	Annealing process in metallurgy	Memory-based search	Whale bubble-net hunting
Exploration vs. Exploitation	Balanced through adaptive mechanisms	Strong exploration initially, then exploitation	Balanced	Balanced	Balanced	Exploration initially, then exploitation	Balanced	Balanced	Balanced	Exploration initially, then exploitation	Exploitation-focused	Balanced
Memory Usage	Utilizes historical data for perturbation	Pheromone trails as indirect memory	Historical positions of bats	Habitat suitability index	Best solutions pollinate	Population-based, uses historical data	Pack leader memory	Harmony memory	Historical best positions	Simulated states	Tabu list	Whale positions memory
Main Operators	Local search, crossover, mutation, perturbation	Pheromone update, path selection	Echolocation, frequency tuning	Migration, mutation	Global and local pollination	Selection, crossover, mutation	Encircling prey, attacking, searching	Pitch adjustment, random selection	Velocity and position update	Temperature-based state changes	Tabu list and neighborhood search	Encircling prey, bubble-net hunting
Adaptivity	Adaptive step size and perturbation intensity	Pheromone evaporation rate	Frequency adjustment, loudness and pulse rate	Migration rates, mutation rates	Switching probability	Mutation and crossover probabilities	Adaptation in search phases	Adjustments in harmony memory consideration rate	Inertia weight, cognitive and social coefficients	Temperature and cooling schedule	Adaptive tabu list size	Adaptive hunting mechanism
Convergence Speed	Generally fast due to adaptive mechanisms	Moderate	Fast	Moderate	Fast	Fast	Fast	Moderate	Fast	Moderate	Slow to moderate	Fast
Computational Complexity	Moderate to high, depends on parameter settings	Moderate	Moderate	Moderate	Moderate	Moderate	Moderate	Moderate	Low to moderate	Low	Moderate to high	Moderate
Parameter Sensitivity	Moderately sensitive; requires tuning	Highly sensitive to pheromone parameters	Moderately sensitive	Moderately sensitive	Moderately sensitive	Highly sensitive	Moderately sensitive	Moderately sensitive	Moderately sensitive	Highly sensitive	Moderately sensitive	Moderately sensitive
Scalability	Good for high-dimensional problems	Moderate	Good	Good	Good	Good	Good	Good	Good	Moderate	Good	Good
Flexibility	High, can incorporate various strategies	Moderate	Moderate	Moderate	Moderate	High	Moderate	Moderate	High	High	Moderate	Moderate

- Inspiration: The natural or artificial process that inspired the algorithm's development.
- Exploration vs. Exploitation: The algorithm's balance between searching new areas (exploration) and refining known good areas (exploitation).
- Memory Usage: How the algorithm uses past information to guide future searches.
- Main Operators: The primary mechanisms or processes the algorithm uses to find solutions.
- Adaptivity: The algorithm's ability to adjust its parameters dynamically during the optimization process.
- Convergence Speed: How quickly the algorithm typically finds a solution.
- Solution Quality: The effectiveness of the algorithm in finding high-quality solutions.
- Computational Complexity: The computational resources required by the algorithm, often related to time and memory usage.
- Parameter Sensitivity: The degree to which the algorithm's performance is affected by its parameter settings.
- Scalability: The algorithm's capability to handle problems of increasing size or complexity.
- Flexibility: The algorithm's adaptability to different types of optimization problems.
- Diversity Maintenance: How the algorithm ensures a diverse set of solutions to avoid premature convergence.
- Typical Applications: Common fields or problems where the algorithm is frequently applied.

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

3.0 Description of benchmarking algorithms, experiments, and functions

This section describes the experimental examination used to benchmark FFO. For a start, FFO was examined against 13 other commonly used optimization algorithms, namely, the Ant Colony Optimization (ACO), Bat Algorithm (BA), Biogeography-Based Optimization (BBO), Cuckoo Search (CS), Firefly Algorithm (FA), Flower Pollination Algorithm (FPA), Genetic Algorithm (GA), Grey Wolf Optimizer (GWO), Harmony Search (HS), Particle Swarm Optimization (PSO), Simulated Annealing (SA), Tabu Search (TS), and Whale Optimization Algorithm (WOA). All of these algorithms were used in their default settings[2] , and a brief description of each is presented herein for completion. We invite interested readers to review the original publications for these algorithms to learn more about their settings and applications.

3.1 Ant Colony Optimization (ACO)

The Ant Colony Optimization (ACO) was formulated by Dorigo and colleagues [21] based on the pheromone-laying behavior observed in certain ant species. This method uses ants as artificial agents to simulate the decision-making process of real ants in selecting paths. As ants traverse paths, they deposit pheromones that guide subsequent ants toward promising solutions. The ACO algorithm is a probabilistic approach to problem-solving where the search space is represented as a graph, and paths through this graph are evaluated based on the intensity of pheromone deposits. The algorithm's efficiency hinges on several parameters: alpha (influence of pheromone on path selection, set at 1.0), beta (influence of heuristic information on path choice, set at 2.0), and rho (rate of pheromone evaporation, set at 0.5). This algorithms has been successfully applied to network routing, scheduling, and other optimization problems that involve finding optimal paths through graphs [22].

3.2 Bat Algorithm (BA)

Developed by Yang et al. in 2012 [23], the Bat Algorithm (BA) was designed to mimic the echolocation behavior of bats. This algorithm models bats that emit sound waves to navigate and locate prey, translating this biological mechanism into a search and optimization strategy. In BA, each simulated bat adjusts its flight based on velocity, loudness, and echolocation frequency, which dynamically changes from exploration to exploitation phases depending on the proximity to optimal solutions. Key parameters of BA include alpha (initial loudness, set at 0.5), gamma (rate of loudness decrease and emission rate increase, set at 0.5), and frequency range (f_{min} at 0, f_{max} at 2.0). BA is adept at tackling complex problems characterized by continuous and multimodal search spaces and has shown effectiveness in engineering design and dynamic optimization tasks [24].

[2] The default setting for FFO include:

```
FirefighterOptimization:
    def __init__(self, objective_func, dimension, num_agents=100, max_iter=500, no_improve_limit=30, bounds=(-5.12, 5.12), step_size=1.0, crossover_probability=0.5, mutation_probability=0.1, initial_temp=100.0, cooling_rate=0.95, verbose=False, use_additional_conditions=False, target_fitness=1e-5):)
```

3.3 Biogeography-Based Optimization (BBO)

Introduced by Simon in 2008 [25], Biogeography-Based Optimization (BBO) leverages migration concepts from biogeography to solve optimization problems. BBO operates on the premise that species migrate between habitats, affecting their survival and reproduction rates. The algorithm features migration operators that simulate gene flow by exchanging solution features, akin to species migration in nature. Key components include the habitat suitability index, which evaluates the desirability of solutions, and migration rates that determine the exchange intensity between solutions. BBO also uses mutation to enhance genetic diversity and avoid premature convergence on suboptimal solutions. The BBO algorithm has proven effective in network design, power systems optimization, and other applications where geographic considerations are crucial [26].

3.4 Cuckoo Search (CS)

Cuckoo Search (CS) was developed by Yang and Deb [27]. This algorithm is inspired by the obligate brood parasitism of some cuckoo species by laying their eggs in the nests of other host birds. If a host bird discovers the eggs are not its own, it will either throw them away or abandon its nest. The algorithm uses this idea to lay a new solution (egg) into a randomly chosen nest, and the best nests with high-quality eggs will be carried over to the next generations. CS is known for its simplicity and flexibility. It has been effectively applied in solving problems like structural design, scheduling, and routing problems where the search space is discrete, and the global optimum is hidden among many local optima [28].

3.5 Firefly Algorithm (FA)

The flashing behavior of fireflies inspires the Firefly Algorithm (FA). Such a flashing behavior acts as a signal system to attract other fireflies. FA, developed by Yang in 2008 [29], uses these biologically inspired techniques to handle optimization problems and functions. Fireflies in the algorithm search the space by moving towards brighter and more attractive fireflies. The attractiveness is proportional to the brightness, and both decrease as their distance increases. The landscape of the objective function determines the brightness of a firefly. A key advantage of FA is its ability to deal with multimodal optimization problems, as it naturally divides the population into subgroups that converge to different optima. Some of the key settings in the FA algorithm include alpha (a randomness factor that affects the movement of a firefly and helps fireflies explore the search space beyond the immediate neighboring fireflies, selected at 0.5), beta (controls how strongly other fireflies are drawn towards it, selected at 1.0), and gamma (influences how the attractiveness of a firefly decreases with distance, selected at 1.0). This feature makes it particularly useful for complex functions with multiple local optima. FA has been applied to problems like economic dispatch, clustering, and image processing [30].

3.6 Flower Pollination Algorithm (FPA)

Devised by Yang et al. in 2012 [31], the Flower Pollination Algorithm (FPA) algorithm emulates the natural pollination processes of flowers. It aims to optimize solutions by alternating between self-pollination and cross-pollination mechanisms that are naturally facilitated by natural vectors like insects, wind, or water. This approach maintains solution diversity and promotes effective convergence. In FPA, solutions are represented as flowers whose attractiveness—determined by fitness—guides the pollination process. The algorithm uses local pollination for minor adjustments within the immediate search area and global pollination, employing Levy flights for broader searches to escape local optima. FPA's dual strategy has been effectively applied to engineering design and economic load dispatch challenges [32].

3.7 Genetic Algorithm (GA)

Holland [33] formulated the Genetic Algorithm (GA) as a computational analog to natural selection, embodying the principle of survival of the fittest. GA begins with a population of randomly generated individuals, evolving over generations to optimize solutions. Selection is fitness-based, favoring solutions that perform better under a defined fitness function. GA incorporates mutation and crossover as genetic operators to introduce variability and new traits into offspring. Typical parameters include a population size of 100, a mutation rate of 0.1, and a crossover rate of 0.1. This algorithm is widely utilized across fields such as optimization, automatic programming, and machine learning, where it helps solve complex problems efficiently [34].

3.8 Grey Wolf Optimizer (GWO)

The Grey Wolf Optimizer (GWO), introduced by Mirjalili et al. in 2014 [35], is a nature-inspired metaheuristic algorithm inspired by grey wolves' social structure and hunting behavior. Grey wolves exhibit a distinct hierarchical system consisting of alpha, beta, delta, and omega wolves, with each tier playing a specific role within the pack. In GWO, this hierarchy is mirrored in the solution process: the alpha wolf represents the optimal solution, followed by beta and delta as the second and third best solutions, respectively, while omega wolves embody the remaining candidate solutions. The algorithm leverages this structure to simulate the wolves' hunting strategy, which is segmented into three phases: tracking, encircling, and attacking prey, each reflecting a critical phase of the optimization process. GWO is adept at navigating complex, multidimensional landscapes, making it valuable in fields such as mechanical engineering design and renewable energy optimization, where the search spaces often exhibit high nonlinearity and multimodality.

3.9 Harmony Search (HS)

Developed by Geem et al. in 2001 [36], Harmony Search (HS) is an optimization algorithm inspired by the improvisational process of musicians tuning their instruments to achieve aesthetic harmony. This algorithm iteratively adjusts solution vectors in a similar fashion to musicians' adjust pitches to optimize a given function. HS employs a stochastic approach rather than a gradient-based method, enhancing its efficacy in addressing non-differential and discrete problems. The algorithm’s performance is governed by two primary parameters: the harmony memory consideration rate (set at 0.9), which dictates the likelihood of selecting existing memory solutions for new harmonies, and the pitch adjustment rate (set at 0.3), which determines how much the chosen solutions are modified. HS has shown significant utility in solving complex engineering problems such as structural and water network design, where traditional methods may struggle due to the extensive search spaces involved.

3.10 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) was introduced by Kennedy and Eberhart in 1995 [37]. This algorithm simulates the social behaviors observed in flocks of birds or schools of fish. This metaheuristic optimizes problem solutions by iteratively enhancing a population of candidate solutions based on the personal and collective experiences of the particles. For example, the PSO starts with randomly initialized particles (solutions) and updates their positions within the search space by balancing personal best achievements and global knowledge shared across the swarm. The algorithm is known for its simplicity and adaptability, often requiring few parameter adjustments. It utilizes three key parameters: swarm size (typically 100 particles), cognitive coefficient (influence of the particle's own memory, set at 1.0), and social coefficient (influence of neighboring particles, also set at 1.0). These factors influence the dynamics of particle movements towards optimal solutions. PSO is particularly effective in continuous, high-dimensional environments and has been applied successfully across various domains, including electrical power systems, robotics, and bioinformatics [38].

3.11 Simulated Annealing (SA)

Simulated Annealing (SA) was developed by Kirkpatrick et al. in 1983 and Cerny in 1985 [39]. This is a probabilistic method designed to approximate the global optimum of a function. SA is inspired by the metallurgical process of annealing, where materials are heated and then gradually cooled to improve their structural properties. This process is mimicked by allowing a system to explore higher energy states (solutions) by heating, thereby overcoming local optima, followed by slow cooling to stabilize at a lower energy state (optimal solution). The algorithm makes random transitions to neighboring solutions, accepting improvements outright and worse solutions based on a decreasing probability over time. Key parameters include the initial temperature (set at 100) and cooling rate (set at 0.95). SA has been successfully applied across various domains, from economics to computational science [40].

3.12 Tabu Search (TS)

Tabu Search (TS) was introduced by Glover in the late 1980s [41] as a metaheuristic that extends beyond local search methods. TS extends such methods by adopting a memory structure, the tabu list, to avoid cycling back to previously encountered suboptimal solutions. This list temporarily bans certain moves, helping the algorithm to escape local optima and explore less favorable solutions that might lead to a globally optimal solution. TS is adaptable, allowing periodic resetting of the tabu status to balance exploration and exploitation. It is particularly effective in complex scheduling, logistical planning, and assignment problems, where its ability to navigate challenging solution spaces is applications [42].

3.13 Whale Optimization Algorithm (WOA)

The Whale Optimization Algorithm (WOA) was developed by Mirjalili and Lewis in 2016 [43]. The WOA draws inspiration from the bubble-net feeding behavior of humpback whales. This algorithm simulates the whales' strategies of encircling prey and using a spiral path to close in, which are mirrored in the shrinking encircling mechanism and spiral updating position phases of the algorithm. These methods allow the WOA to balance exploration and exploitation dynamically, making it adept at handling complex, non-separable, and nonlinear optimization problems. WOA's effectiveness is demonstrated in its applications across mechanical design and industrial engineering, where it optimizes a variety of challenging problem landscapes [44].

4.0 Description of utilized benchmarking functions

This section describes 24 benchmark functions commonly used in benchmarking analysis.

4.1 Ackley Function

The Ackley function has a two-dimensional form with a relatively uniform plane [45]. This function also has several dozen local minimums and one global extreme of significantly smaller value than most of the local minimums. This function allows very efficient testing of optimization algorithms as regards stopping at local extremes. The Ackley function is designed to test the ability of optimization algorithms to escape local minima and converge towards a global minimum in a complex landscape. The global minimum is at x=0 where $f(x)$=0. This function has the following form:

$$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \qquad \text{Eq. 1}$$

4.2 Alpine Function

The Alpine function is a multimodal and non-smooth function and hence can provide significant challenges in terms of local minima and ruggedness tests [46]. This function examines the ability of optimization algorithms to handle non-differentiable points with abrupt changes. The global minimum for this function occurs at x=0 where $f(x)$=0. This function can be useful for testing optimization algorithms in real-world problems involving non-smooth dynamics, such as mechanical systems with friction or other resistive forces. This function has the following form:

$$f(x) = \sum_{i=1}^{n} |x_i \sin(x_i) + 0.1x_i| \qquad \text{Eq. 2}$$

4.3 Booth's Function

This function presents a simple test case for algorithm testing with a convex with a single global minimum at (x,y) = (1,3) where $f(x,y)$ = 0. This function has the following form:

$$f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2 \qquad \text{Eq. 3}$$

4.4 Cross-in-Tray Function

This function is known for its challenging landscape, characterized by a high degree of multimodality [47]. The function contains several deep holes, indicative of global minima, which are located symmetrically in the function's domain, and may present a significant challenge in the convergence process of algorithms to navigate complex landscapes and avoid local minima in favor of locating and confirming global minima. The global minima occur at approximately (x,y)=(1.34941,−1.34941),(−1.34941,1.34941),(1.34941,1.34941),(−1.34941,−1.34941) with $f(x,y)$≈−2.06261. This function has the following form:

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

$$f(x,y) = -0.0001\left(\left|\sin(x)\sin(y)\exp\left(\left|100-\frac{\sqrt{x^2+y^2}}{\pi}\right|\right)\right|+1\right)^{0.1} \quad \text{Eq. 4}$$

4.5 Drop-Wave Function

This function features a rippled wave surface that turn challenging to optimize due to its frequent local minima and a pronounced global minimum [48]. This is a multimodal function with a global minimum at (x,y)=(0,0) where $f(x,y)$=−1. It tests an algorithm's capability to navigate through frequent oscillations to find the lowest point and is particularly relevant for simulations and optimizations in fields involving vibrational analysis and wave propagation (i.e., acoustics and materials science, etc.). This function has the following form:

$$f(x,y) = -\frac{1+\cos\left(12\sqrt{x^2+y^2}\right)}{0.5(x^2+y^2)+2} \quad \text{Eq. 5}$$

4.6 Easom Function

The Easom function is a highly unimodal benchmark function stemming from its narrow global peak that is surrounded by a flat landscape [49]. This function has a constant plane over the vast majority of the domain with one global minimum, at (x,y)=(π,π) where $f(x,y)$=−1, that is difficult to locate due to the flatness of the surrounding area. Oftentimes, this function is used in testing the precision and convergence characteristics of optimization algorithms, and their ability to hone in on and precisely converge to a sharply defined minima. This function has the following form:

$$f(x,y) = -\cos(x)\cos(y)\exp\left(-\left((x-\pi)^2+(y-\pi)^2\right)\right) \quad \text{Eq. 6}$$

4.7 Eggholder Function

The Eggholder function has a highly irregular and complex surface characterized by an uneven plane with several local minimums (of values like its only global minimum) [50,51]. The global minimum is found at (x,y)=(512,404.2319) where $f(x,y)$≈−959.6407. This function is frequently used in benchmarking sophisticated global optimization algorithms, especially those intended to solve rugged and unpredictable landscape-based problems. This function has the following form:

$$f(x,y) = -(y+47)\sin\left(\sqrt{\left|\left(\frac{y+x}{2+47}\right)\right|}\right) - x\sin\left(\sqrt{|x-(y+47)|}\right) \quad \text{Eq.7}$$

4.8 Expanded Schaffer's F6 Function

This is an expansion of the original Schaffer's function that hopes to test for algorithm effectiveness over a broader area with a more complex landscape. More specifically, the function is highly multimodal and oscillatory and hence presents a significant challenge in identifying the global minimum amidst numerous local minima. The function has a global minimum at (x,y)=(0,0) where $f(x,y)$=0. This function can be used in testing spatial algorithms that may be applied in fields like geographic information systems and molecular dynamics where spatial relationships and dynamics are crucial.

$$f(x) = 0.5 + \frac{\sin^2\left(\sqrt{x^2+y^2}\right)-0.5}{[1+0.001(x^2+y^2)]^2} \quad \text{Eq. 8}$$

4.9 Expanded Zakharov Function

This is an extension of the Zakharov function [46] and provides a more challenging scenario for testing optimization algorithms by combining linear, quadratic, and quartic terms. This function has a single global minimum at x=0 where $f(x)$=0. This function is used to evaluate the performance of large-scale optimization algorithms in areas such as financial modeling and energy systems, where complex interactions between variables are common.

$$f(x) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} 50ix_i\right)^2 + \left(\sum_{i=1}^{n} 50ix_i\right)^4 \quad \text{Eq. 9}$$

4.10 Goldstein-Price Function

This function was created by Goldstein and Price [52] to provide a multimodal complex landscape with sharp peaks and valleys. The same function has several local minima and a global minimum found at $(x,y) = (0,-1)$ with $f(x,y) = 3$. This function has the following form:

$$f(x,y) = [1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)] \times [30 + 2(sx - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)] \quad \text{Eq. 10}$$

4.11 Griewank Function

The Griewank function has large, flat areas interrupted by periodic narrow, deep valleys [53]. This function is highly multimodal and oscillatory and can challenge the algorithm's ability to find the global minimum (at x=0) amidst frequent changes in gradient. It is particularly used to test the efficiency of algorithms in handling complex oscillations and multimodal functions with application within acoustic waveguides design and structural engineering with regard to vibrations. This function has the following form:

$$f(x) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(x_i/\sqrt{i}\right) \quad \text{Eq. 11}$$

4.12 Himmelblau's Function

Developed by Himmelblau [54], this function is characterized by multiple global minima, which makes it interesting for testing the robustness of optimization algorithms to locate and distinguish between multiple optima within a complex landscape. This function has four identical global minima located at (x,y)=(3,2),(−2.805118,3.131312),(−3.779310,−3.283186),(3.584428,−1.848126) where $f(x,y)$=0. This function has the following form:

$$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 \quad \text{Eq. 12}$$

4.13 Holder Table Function

The Holder Table function features several deep and narrow global minima and is designed to challenge optimization algorithms in finding and recognizing global solutions in a multimodal space [55]. The function’s global minima are symmetrically located around the origin, with four known global minima where $f(x,y)$=−19.2085. This function can serve as a good evaluation metric for multimodal optimization capabilities in logistics and routing problems where multiple equivalent optimal routes need to be evaluated. This function has the following form:

$$f(x,y) = -\left|\sin(x)\cos(y)\exp\left(\left|1-\frac{\sqrt{x^2+y^2}}{\pi}\right|\right)\right| \quad \text{Eq. 13}$$

4.14 Levy Function N.13

This is a variant to the Levi function. This variant is designed to test algorithms against steep gradients and local optima. More specifically, this function has steep ridges and a complex global structure with a global minimum at (x,y)=(1,1) where $f(x,y)$=0. The Levi can be useful in examining algorithms that need to handle sudden changes in gradient effectively. This function has the following form:

$$f(x,y) = \sin^2(3\pi x) + (x-1)^2(1+\sin^2(3\pi y)) + (y-1)^2\,(1+\sin^2(2\pi y) \quad \text{Eq. 14}$$

4.15 Matyas Function

This function was created by Matyas [56] as convex function that could serve as an elementary test case for basic functionality and efficiency of optimization algorithms in a controlled setting. The Matyas function has a global minimum at (x,y) = (0,0) where $f(x,y)$=0. This function has the following form:

$$f(x,y) = \ 0.26(x^2+y^2) - 0.48xy \quad \text{Eq. 15}$$

4.16 Michalewicz Function

This function was created by Michalewicz [57] to be especially difficult for evolutionary algorithms to solve. This function is highly multimodal, with sharp peaks and valleys that are sensitive to the variable m, which controls the steepness of the valleys and ridges. The function's global minimum becomes more difficult to locate as m increases (with a typical value of m=10). This function can be used in the testing and development of genetic and evolutionary algorithms, particularly effective for applications requiring high precision in aerodynamics and biomechanical engineering. This function has the following form:

$$f(x) = -\sum_{i=1}^{n}\sin(x_i)\sin^{2m}\left(\frac{ix_i^2}{\pi}\right) \quad \text{Eq. 16}$$

4.17 Rastrigin Function

This function is named after Rastrigin [58] and presents an example of a highly non-linear multimodal function with frequent local minima. The global minimum is at x=0 where $f(x)$=0. The function is particularly designed to test the algorithm's capability to escape local minima and is widely used in the testing and development of algorithms in evolutionary computation and real-world scenarios where noise and local minima are prevalent, such as in electronic circuit design. This function has the following form:

$$f(x) = 10n + \sum_{i=1}^{n}[x_i^2 - 10\cos(2\pi x_i)] \quad \text{Eq. 17}$$

4.18 Rosenbrock Function

This function was designed by Rosenbrock in 1960 as a non-linear and non-convex function to test the performance of optimization algorithms over rugged terrain with a narrow, curved valley leading to a global minimum [59]. This function has a global minimum inside a long, narrow, parabolic shaped flat valley. In general, finding the valley is straightforward but converging to the global minimum is difficult. A common multidimensional generalization of this function has the following form:

$$f(x) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \quad \text{Eq. 18}$$

4.19 Schaffer Function N. 2

This function is a variant belonging to the family of functions introduced by Schaffer, which are used to evaluate the performance of optimization algorithms in handling oscillating landscapes with narrow valleys [60]. The function is non-convex and multimodal, with a global minimum at (x,y)=(0,0) where $f(x,y)$=0. This variant can be sensitive to initial conditions due to the presence of sharp peaks and deep valleys. This function has the following form:

$$f(x,y) = 0.5 + \frac{\sin^2(x^2-y^2)-0.5}{[1+0.001(x^2+y^2)]^2} \quad \text{Eq. 19}$$

4.20 Schwefel Function

The Schwefel function is a classic optimization test problem introduced by Schwefel [61]. This function is initially created for evolutionary algorithms and has complex and non-linear large number of local minima (with a global minimum located near the bounds of the search space at x=(420.9687,…,420.9687) where $f(x)$≈0). This function can be helpful in in fields such as aerospace for optimizing the shapes and trajectories of dynamic flying bodies. This function has the following form:

$$f(x) = 418.9829n - x_i \sin\left(\sqrt{|x_i|}\right) \quad \text{Eq. 20}$$

4.21 Sphere Function

The Sphere function is one of the classical and simplest benchmark functions often used to test preliminary optimization algorithms in terms of convergence and accuracy. The Sphere function is continuous, convex, and unimodal with a global minimum at $x^* = 0$ where $f(x^*) = 0$. This function has the following form:

$$f(x) = \sum_{i=1}^{n} x_i^2 \quad \text{Eq. 21}$$

4.22 Styblinski-Tang Function

The Styblinski-Tang function has steep valleys and multiple local minima and hence is recognized for its utility in testing optimization algorithms [62]. This function is multimodal, with each variable contributing quadratically and quartically to the output. This function can be suitable for evaluating the efficiency of algorithms in high-dimensional spaces and their capability to scale with increasing dimensionality (which makes it appropriate for practical engineering problems

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

involving material design and circuit optimization with multiple variables required to be simultaneously optimized). The global minimum for this function is at x_i=−2.903534x (for all i) where $f(x)$=−39.16599n (where n is the number of dimensions). This function has the following form:

$$f(x) = \frac{1}{2}\sum_{i=1}^{n}(x_i^4 - 16x_i^2 + x_i 5) \qquad \text{Eq. 22}$$

4.23 Three-hump Camel Function

This function is named for its shape, resembling three camel humps and is designed as a simple and effective benchmark for testing optimization algorithms [63]. This function can be used to evaluate an algorithm's capability to escape local minima and find the global minimum. The Three-hump Camel function has a global minimum at (x,y)=(0,0) where $f(x,y)$=0. This function has the following form:

$$f(x,y) = 2x^2 - 1.05x^4 + \frac{x^6}{6} + xy + y^2 \qquad \text{Eq. 23}$$

4.24 Whitley's Function

Whitley's function is known for its complex landscape with a high number of local minima [51]. This function is tests algorithms for their ability to distinguish subtle gradient changes and avoid premature convergence. This function can be used for complex, real-world problems such as landscape exploration and molecular configuration. This function has the following form:

$$f(x) = \sum_{i=1}^{n}\sum_{j=1}^{n}(\frac{100\left(x_i^2-x_j\right)^2+\left(1-x_j\right)^2}{4000} - \cos\left(200\left(x_i^2 - x_j\right)^2 + (1 - x_j)\text{^}2\right) + 1) \qquad \text{Eq. 24}$$

Table 2 includes the name of each function, its typical domain range, primary characteristics, known challenges, and the dimensionality for which the function is typically evaluated. Figure 1 compares these functions visually.

Table 2 Holistic comparison of the examined functions.

Function Name	Domain Range	Characteristics	Challenges	Typical Dimensionality	Minima/ Minimum
Ackley	(-5, 5)	Nearly flat outer region, large hole	Global minimum difficult to find due to flatness	Multiple	0
Alpine	(-10, 10)	Multimodal, peaks and valleys	Peaks make locating global minima challenging	Multiple	0
Beale's	(-4.5, 4.5)	Multimodal	Several local minima	2	0
Booth's	(-10, 10)	Smooth, few local minima	Simplicity, limited challenge	2	0
Cross-in-Tray	(-10, 10)	Highly multimodal	Multiple global minima spread out	2	−2.0626
Drop-Wave	(-5.12, 5.12)	Central peak surrounded by ring of minima	Central peak difficult to stabilize on	Multiple	-1

Easom	(-100, 100)	Narrow peak, unimodal	Extremely narrow attraction region	2	-1
Eggholder	(-512, 512)	Large search space, numerous local minima	Complex landscape, many minima	Multiple	-959.640
Expanded Schaffer's F6	(-10, 10)	Numerous local minima, complex landscape	Maintaining algorithm stability	2	0
Expanded Zakharov	(-10, 10)	Combines parabolic and linear terms	Optimization of combined effects	Multiple	0
Goldstein-Price	(-2, 2)	Complex topology, multimodal	Local minima near boundaries	2	3
Griewank	(-600, 600)	Many widespread minima	Large search space, many local minima	Multiple	0
Himmelblau's	(-5, 5)	Multiple global minima	Identifying correct global minimum	2	0
Holder Table	(-10, 10)	Symmetric, multiple global minima	Symmetry can confuse algorithms	Multiple	-19.2085
Levy N.13	(-10, 10)	Steep valleys, complex structure	Pronounced ridges and steep drops	2	0
Matyas	(-10, 10)	Smooth, unimodal	Limited complexity	2	0
Michalewicz	(0, π)	Designed for evolutionary algorithms	Sharp, narrow valleys	Multiple	Multiple
Rastrigin	(-5.12, 5.12)	Highly multimodal, oscillating	Numerous local minima, large search space	Multiple	0
Rosenbrock	(-2.048, 2.048)	Non-convex, narrow curved valley	Finding the global minimum in the narrow valley	Multiple	0
Salomon's	(-100, 100)	Strong global structure, multimodal	Multiple deceptive local minima	Multiple	0
Schaffer N. 2	(-100, 100)	Sharp ridges, flat areas	Balancing exploration and exploitation	Multiple	0
Schwefel	(-500, 500)	Sinusoidal, maxima and minima	Identifying global minimum amid deceptive maxima	Multiple	0
Sphere	(-5.12, 5.12)	Smooth, convex, unimodal	Simplicity may not challenge advanced algorithms	Multiple	0
Styblinski-Tang	(-5, 5)	Steep valleys, multimodal	Harsh penalties for incorrect solutions	Multiple	-39.165*n*
Three-hump Camel	(-5, 5)	Three humps, unimodal	Misleading local minima	2	0
Whitley's	(-10, 10)	Highly complex and multimodal	Complexity and size of search space	Multiple	0

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

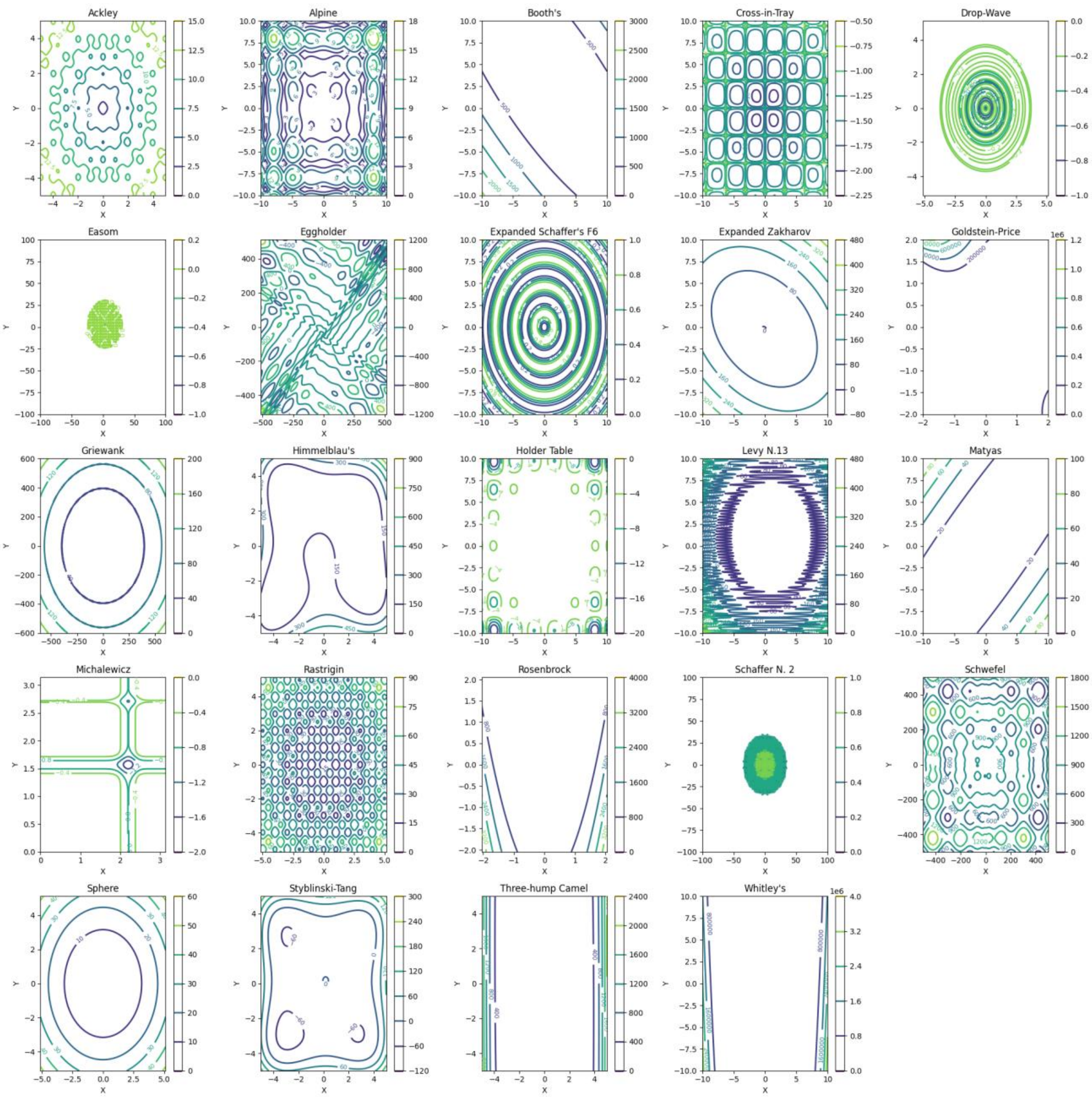


Fig. 1 Visual representation of the utilized test and benchmark functions

5.0 Discussion, experiments, results, and analysis

Two sets of experiments were conducted to evaluate the performance of the FFO algorithm. In the first, we examine the performance of the FFO algorithm and the other listed algorithms against the aforenoted benchmark functions in 2D settings. Then, we examine the algorithmic performance against the scalable functions at higher dimensions (20D and 50D). In all cases, the experiments entitled a comparison of algorithmic performance in terms of best fitness and fitness history

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

achieved, time taken to execute the analysis per algorithm, trajectory analysis, and a combined combination of metrics. All of these items are discussed herein in detail.

The best fitness is defined as the most favorable value of the objective function obtained in an analysis by an algorithm across all of its iterations or agents. This metric is traced progressively, with the algorithm updating this metric whenever an improved solution is found. Then, the exaction time is defined as the total time from the start of its execution until it terminates. Trajectory refers to the sequence of points that document the position of agents (or the best agent) in the search space. This distance is calculated by summing up the Euclidean distances between consecutive points in the trajectory. Further, four additional categories were used to evaluate the selected algorithms. These include documenting the performance of algorithms in terms of identifying the functions that took the longest (and shortest) time to solve and those that achieved the most (and least) accuracy.

To facilitate a leaner comparison, the execution time and distance explored metrics were combined into a new metric named the Distance per Unit Time metric. This metric directly measures the average distance covered per unit of time, which is directly interpretable as:

$$Distance\ per\ Unit\ Time\ metric\ = \frac{Total\ distance}{Excution\ time}$$

This measures how much distance is covered on average per second. Therefore, higher values indicate more efficient exploration of the search space.

The conducted analysis was ran and evaluated in a Python 3.10.5 environment using an Intel(R) Core(TM) i7-9700F CPU @ 3.00GHz and an installed RAM of 32.0GB. To ensure fairness, the control parameters of FFO and the 13 metaheuristics employed in the performance evaluation simulation were presented earlier and are found in our simulation script. In all cases, all algorithms ran for 100, 1000, and 3000 iterations and with 10, 50, and 100 agents. As mentioned above, the first leg of the analysis focused on all benchmarking functions at 2D, and the second leg focused on scalable functions (12 out of the original 24 functions) at higher dimensions (20D and 50D).

2D setting

Table 3 and Fig. 2 list the overall obtained results from the analysis carried out on all algorithms and functions in 2D setting. This table presents a comparative performance analysis of various optimization algorithms based on metrics such as best fitness, execution time, and distance metrics. Ant Colony Optimization, for instance, shows a mean best fitness of 1.22E+05 with high variability (standard deviation of 9.48E+05) and ranges from -563 to 7.35 in its best fitness performance. In terms of spatial efficiency, it maintains a low average distance metric (mean of 0.092). On the other hand, the Firefly algorithm demonstrates a broader range in execution time, peaking at 1803 seconds, and exhibits a significantly wider spread in the distance metric, reaching

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

up to 91,600. The FFO with additional conditions[3] "OFF" significantly improves best fitness and execution efficiency and large exploration capabilities to a maximum of 1.20E+08. It is quite clear that the FFO ranked well in all metrics and achieved the top ranking in terms of the Distance per Unit Time metric.

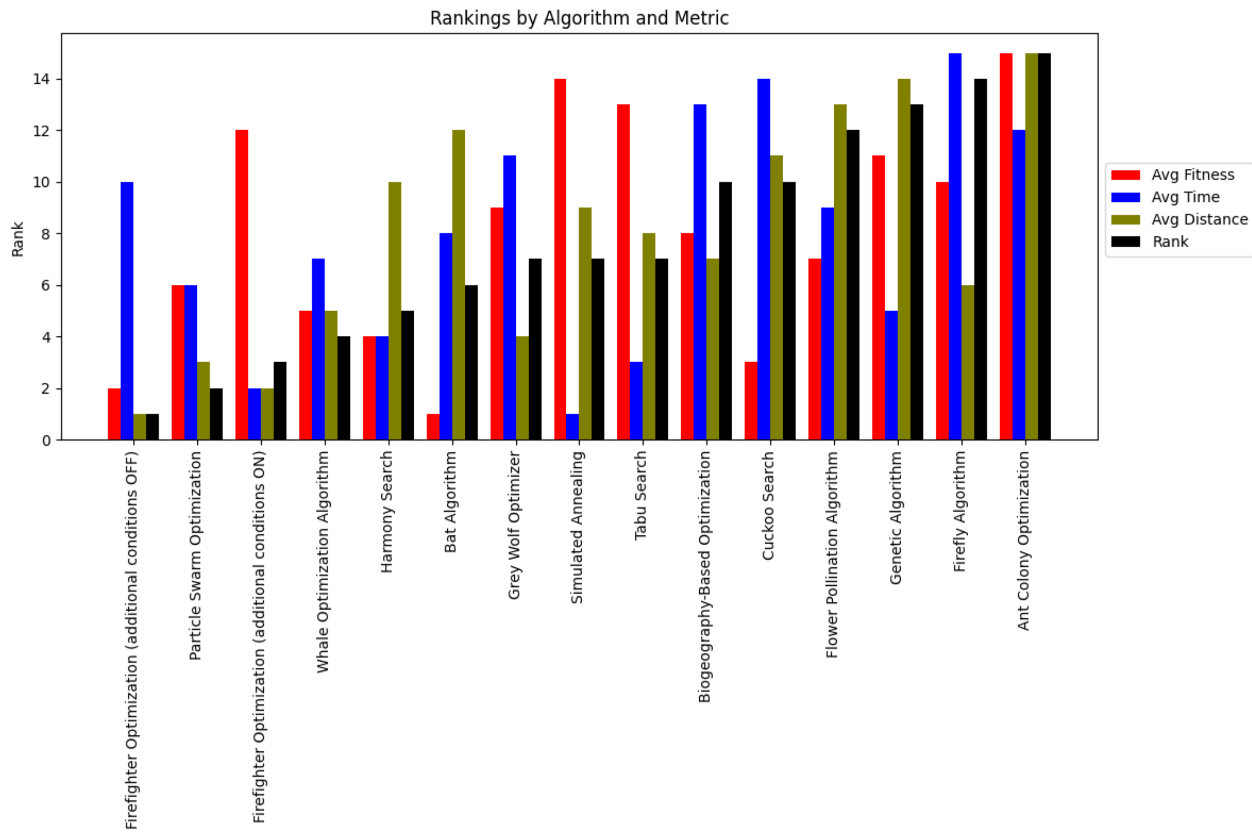


Fig. 2 Ranking of algorithms in 2D settings

[3] The FFO (with additional conditions OFF) runs the FFO in a similar fashion to the other algorithms (i.e., to the same number of iterations without any additional stopping criteria). The counterpart version (with additional conditions ON) runs the FFO with all stopping criteria as described above. A true comparison between all algorithms should rely on the OFF version and hence is maintained herein.

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

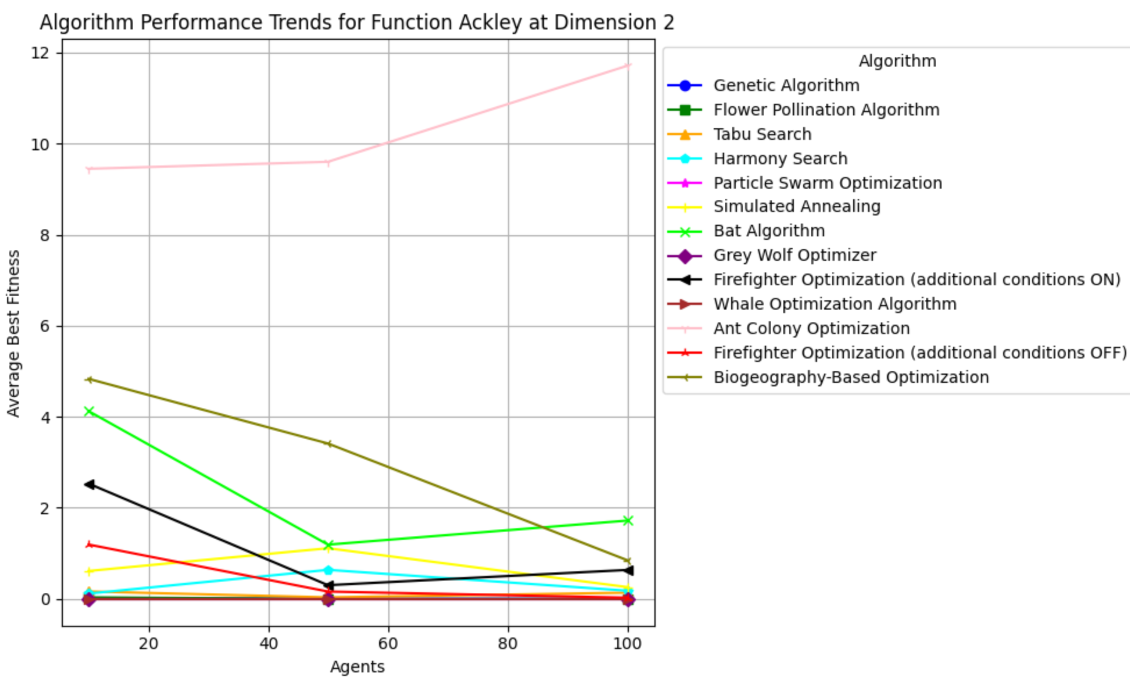
Table 3 Overall results for 2D setting

Algorithm	Best Fitness				Execution Time (s)				Distance metric				Distance per Unit Time
	mean	std	min	max	mean	std	min	max	mean	std	min	max	
Ant Colony Optimization	1.22E+05	9.48E+05	-5.63E+02	7.35E+00	7.35E+00	9.39E+00	9.00E-02	3.51E+01	9.26E-02	1.36E+00	0.00E+00	2.00E+01	1.26E-02
Bat Algorithm	-1.83E+00	4.58E+02	-9.77E+02	2.63E+00	2.63E+00	3.67E+00	1.88E-02	1.99E+01	3.15E+01	1.58E+02	0.00E+00	1.14E+03	1.20E+01
Biogeography-Based Optimization	-9.72E+00	1.93E+02	-8.81E+02	7.34E+00	7.34E+00	9.76E+00	7.13E-02	4.42E+01	2.64E+03	1.17E+04	0.00E+00	1.41E+05	3.60E+02
Cuckoo Search	-4.46E+01	1.92E+02	-9.60E+02	5.99E+01	5.99E+01	1.18E+02	5.19E-02	7.51E+02	8.48E+01	3.83E+02	0.00E+00	3.74E+03	1.42E+00
FFO (additional conditions OFF)	-4.03E+01	1.84E+02	-9.60E+02	4.61E+00	4.61E+00	7.82E+00	2.50E-02	5.23E+01	2.64E+06	1.21E+07	1.47E+03	1.20E+08	5.74E+05
FFO (additional conditions ON)	-2.04E+01	1.20E+02	-8.18E+02	9.11E-02	9.11E-02	1.25E-01	0.00E+00	7.27E-01	1.02E+05	3.86E+05	0.00E+00	3.36E+06	1.12E+06
Firefly Algorithm	-2.25E+01	1.71E+02	-8.75E+02	1.73E+02	1.73E+02	3.09E+02	1.31E-01	1.80E+03	7.31E+02	6.83E+03	0.00E+00	9.16E+04	4.23E+00
Flower Pollination Algorithm	-3.85E+01	1.75E+02	-9.41E+02	2.86E+00	2.86E+00	3.93E+00	2.33E-02	2.07E+01	6.87E+01	3.08E+02	0.00E+00	2.39E+03	2.40E+01
Genetic Algorithm	-3.02E+01	1.65E+02	-9.56E+02	1.92E+00	1.92E+00	2.48E+00	2.40E-02	1.17E+01	5.62E+00	1.44E+01	3.62E-03	1.14E+02	2.93E+00
Grey Wolf Optimizer	-1.51E+01	1.78E+02	-9.60E+02	5.38E+00	5.38E+00	7.49E+00	4.14E-02	4.01E+01	3.59E+03	1.62E+04	8.90E-02	1.44E+05	6.67E+02
Harmony Search	-3.95E+01	1.78E+02	-9.60E+02	7.52E-01	7.52E-01	1.19E+00	6.00E-03	7.33E+00	1.18E+02	4.52E+02	0.00E+00	3.42E+03	1.57E+02
Particle Swarm Optimization	-3.76E+01	1.74E+02	-9.60E+02	2.39E+00	2.39E+00	3.13E+00	2.46E-02	1.43E+01	1.07E+05	9.96E+05	3.74E+01	1.44E+07	4.45E+04
Simulated Annealing	-3.80E+00	1.34E+02	-7.18E+02	3.65E-02	3.65E-02	3.41E-02	9.97E-04	1.25E-01	1.04E+02	9.39E+01	2.30E+00	7.66E+02	2.86E+03
Tabu Search	1.02E+01	1.81E+02	-7.87E+02	8.23E-02	8.23E-02	7.41E-02	3.99E-03	2.63E-01	5.77E+02	5.14E+02	2.89E+01	1.51E+03	7.01E+03
Whale Optimization Algorithm	-4.14E+01	1.78E+02	-9.58E+02	2.45E+00	2.45E+00	3.22E+00	2.47E-02	1.47E+01	3.73E+03	1.30E+04	2.94E-02	9.68E+04	1.52E+03

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

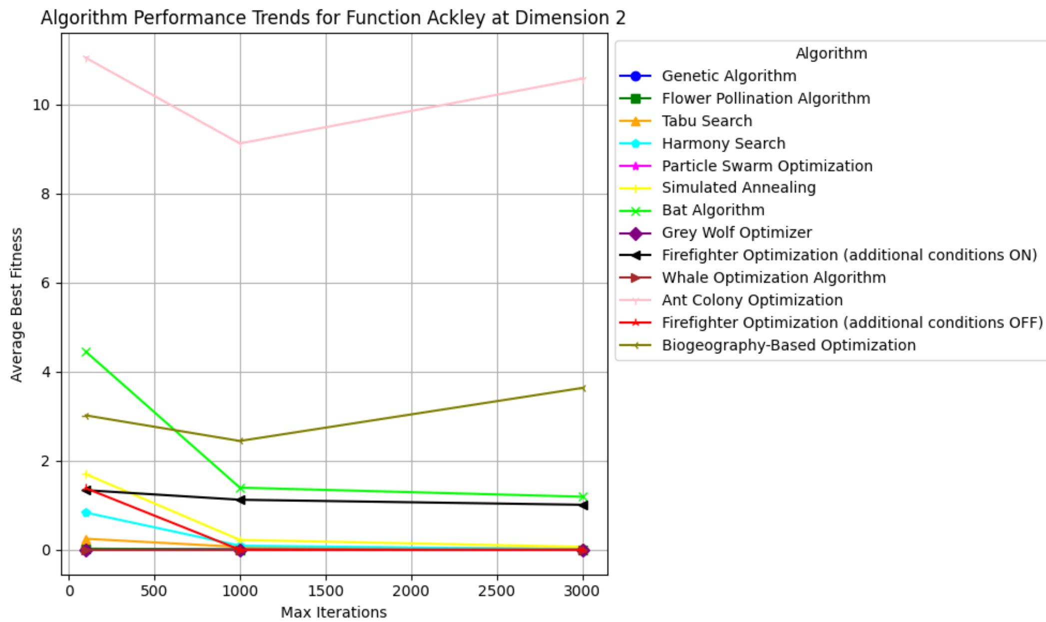
Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

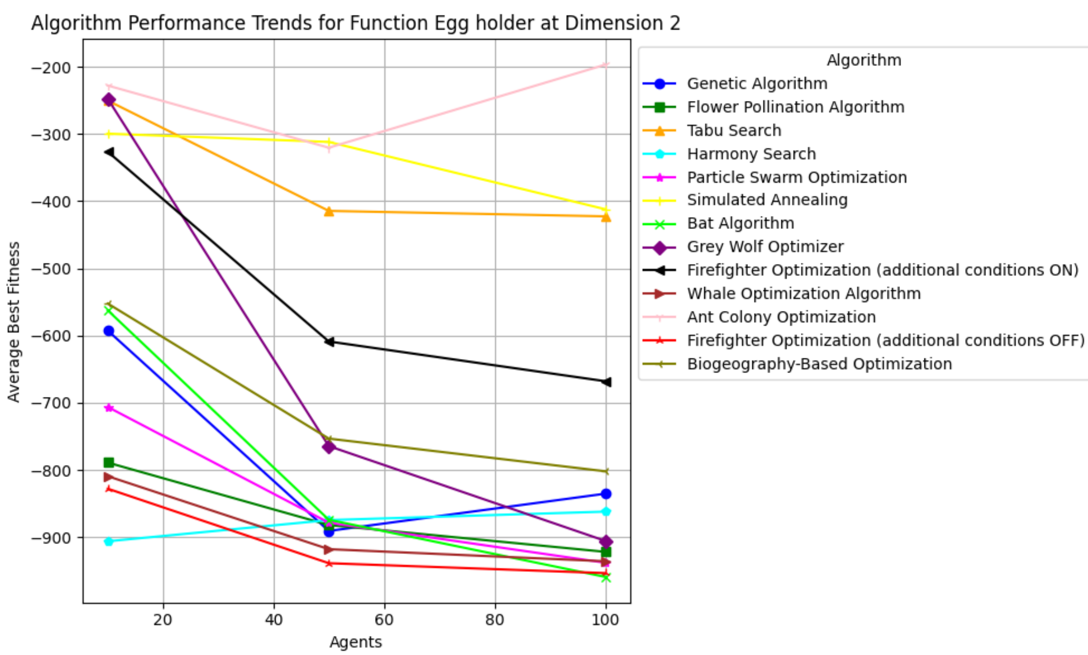
Now, we gain further insights into the performance of all algorithms across different settings through the illustration presented in Fig. 3. This figure shows a sample of trends collected on the selected functions to show how the algorithms perform. It is quite clear that the FFO performs similarly to other algorithms. For example, the first sub-figure highlights performance variations of several optimization algorithms on the Ackley function at 2D, with a specific focus on the number of agents employed. The FFO with additional conditions "OFF" consistently maintains lower fitness values across all agent configurations compared to most algorithms, suggesting higher efficiency. This is in contrast to algorithms like Bat algorithm and Ant Colony Optimization, which show a marked increase in average best fitness. The FFO with additional conditions "OFF" outperforms the latter at higher agent counts, which indicates better scalability with increasing agents. Similar observations can also be made in terms of the algorithmic performance with the number of iterations – especially in the case of the Egg Holder function, which shows superior performance for the FFO algorithm.



This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.





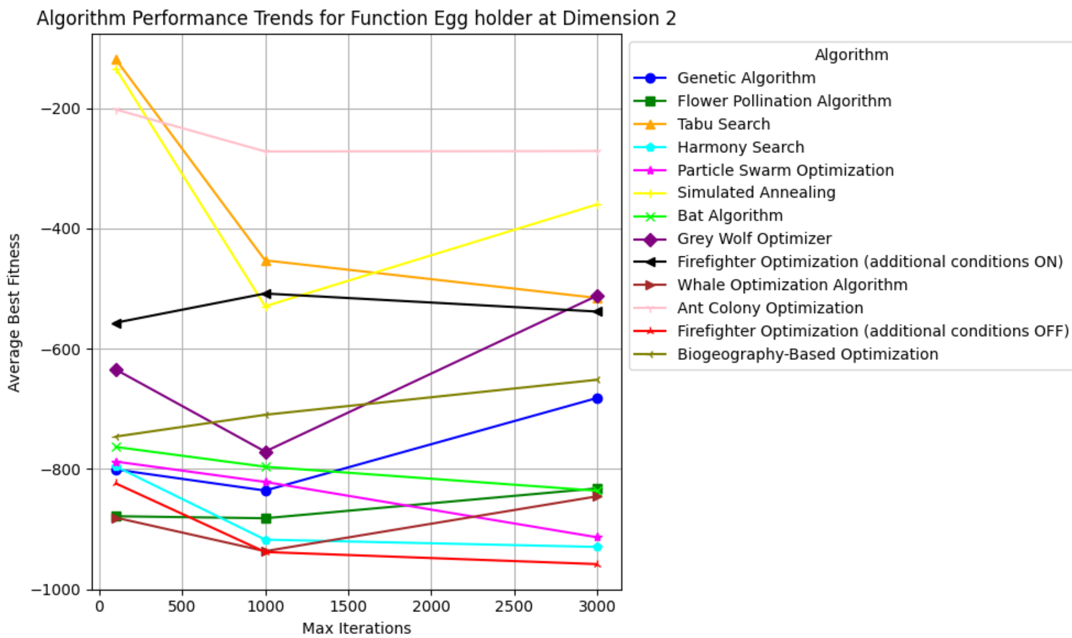


Fig. 3 Sample of trends across different experimental settings

Here, we analyze and rank the performance of optimization algorithms across various settings, specifically focusing on the dimensions, maximum iterations, and the number of agents involved in terms of identifying those that took the longest (and shortest) time to solve, and that achieved the most (and least) accuracy. This process was conducted at each unique setting. First, we rank the algorithms by returning the top three entries of the specified metric and setting. Then, we aggregate these frequencies both locally (for each setting) and globally (across all settings) to provide a comprehensive view of which algorithms consistently perform well or underperform across varied configurations.

We report that the Firefly algorithm appears predominantly in the longest to solve category with a total of 27 out of 27 occurrences. This suggests that the Firefly algorithm, despite its potential advantages in exploring complex landscapes, tends to have longer execution times compared to other algorithms in the study. This could be due to its inherent characteristics, such as the attractiveness parameter and light intensity, which might cause slower convergence, especially in scenarios involving complicated objective functions. The FFO (additional conditions ON) algorithm consistently appears as the fastest solver, also with 27 instances. This algorithm was followed by the Tabu Search. The success in this category indicates the algorithmic potential for applications requiring quick solutions where computational resources can be limited.

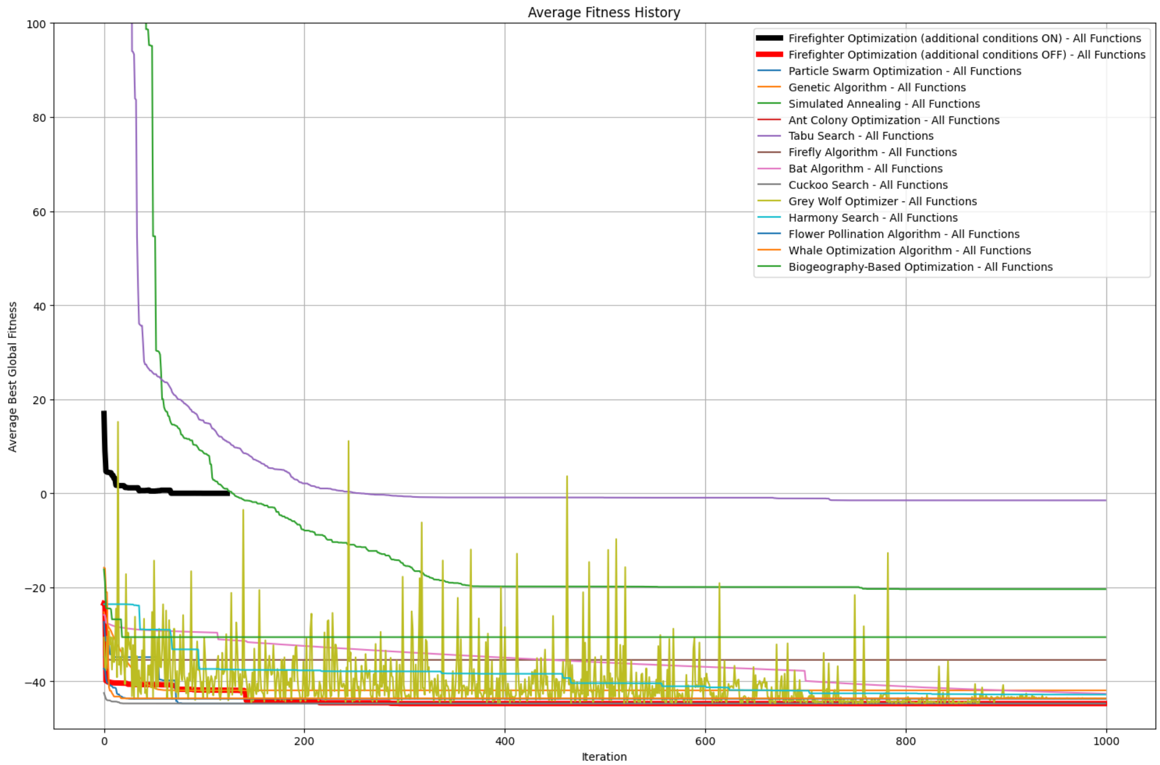
In the most accurate category, the Cuckoo Search leads with 9 occurrences, followed by the FFO (additional conditions OFF) with 5 occurnaces and various other algorithms. The notable performance of Cuckoo Search could be attributed to its unique search capabilities, leveraging Lévy flights for global search combined with a probabilistic switch to local search. On the other hand, the Ant Colony Optimization is predominantly featured in the least accurate category with 25 instances. This might indicate ACO's challenges in maintaining high accuracy across the settings tested, potentially due to its reliance on pheromone trails, which might lead to premature convergence or difficulty in escaping local optima in certain types of problems.

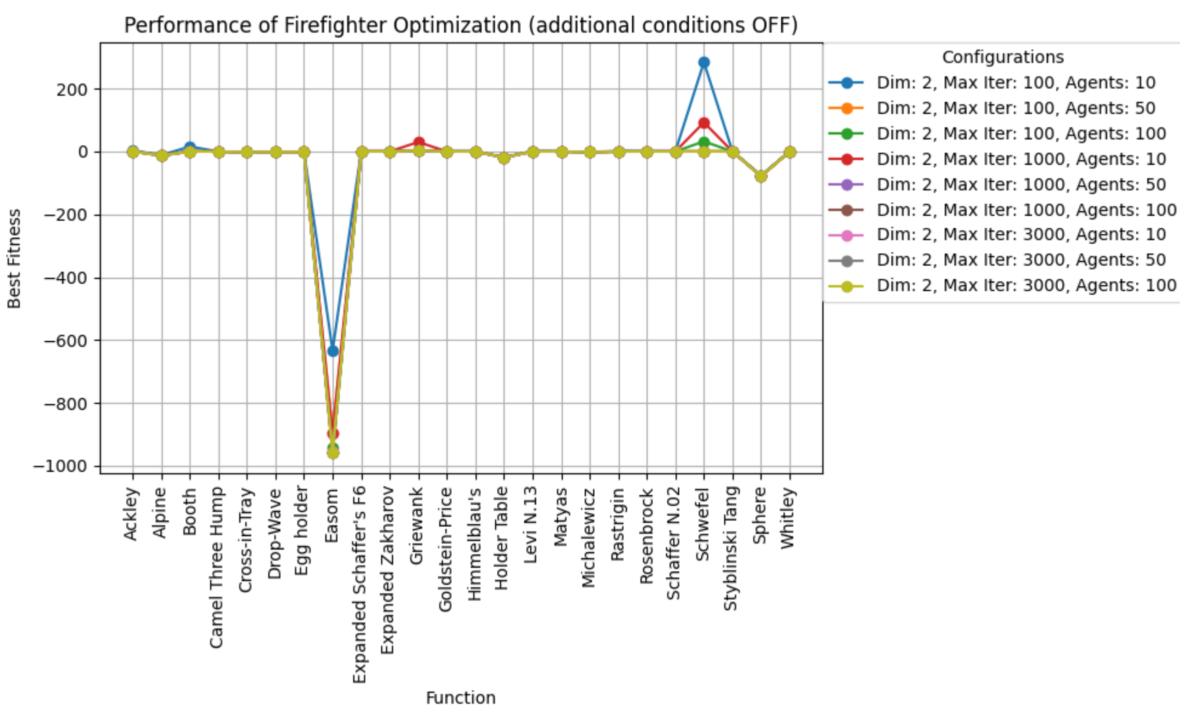
Other sets of comparisons between selected algorithms can be seen in Fig. 4. For example, Fig. 4a compares the average history fitness across all functions and for all algorithms. This figure shows that the FFO (additional conditions OFF) demonstrates a rapid convergence initially compared to others like the Bat and Grey Wolf Optimizer, which exhibit more gradual improvements. Figure 4b compares the best fitness achieved in FFO, Tabu Search, the Bat, and the Grey Wolf Optimizer algorithms. In this figure, FFO (additional conditions OFF) excels in deeply multi-modal landscapes like the Ackley and Griewank functions. This suggests that FFO (additional conditions OFF) is particularly adept at managing and escaping local optima in complex search spaces. The graph also shows minimal variance in performance across different configurations (agent counts and iteration limits), indicating the robustness and effectiveness of this algorithm.

Figure 4c shows the performance of all algorithms in tackling continuous and non-continuous functions to provide a clear comparison of how each algorithm handles different types of function landscapes. Here, the FFO (additional conditions OFF) shows comparable performance in both categories, underscoring its versatility. More specifically, in continuous functions, it ranks among the top performers, closely competing with algorithms like Particle Swarm Optimization and Genetic Algorithm. The FFO (additional conditions OFF) also shows similar performance in non-continuous functions. These figures further show the comparative performance and consistency of the newly proposed FFO algorithm against those well established methods.

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

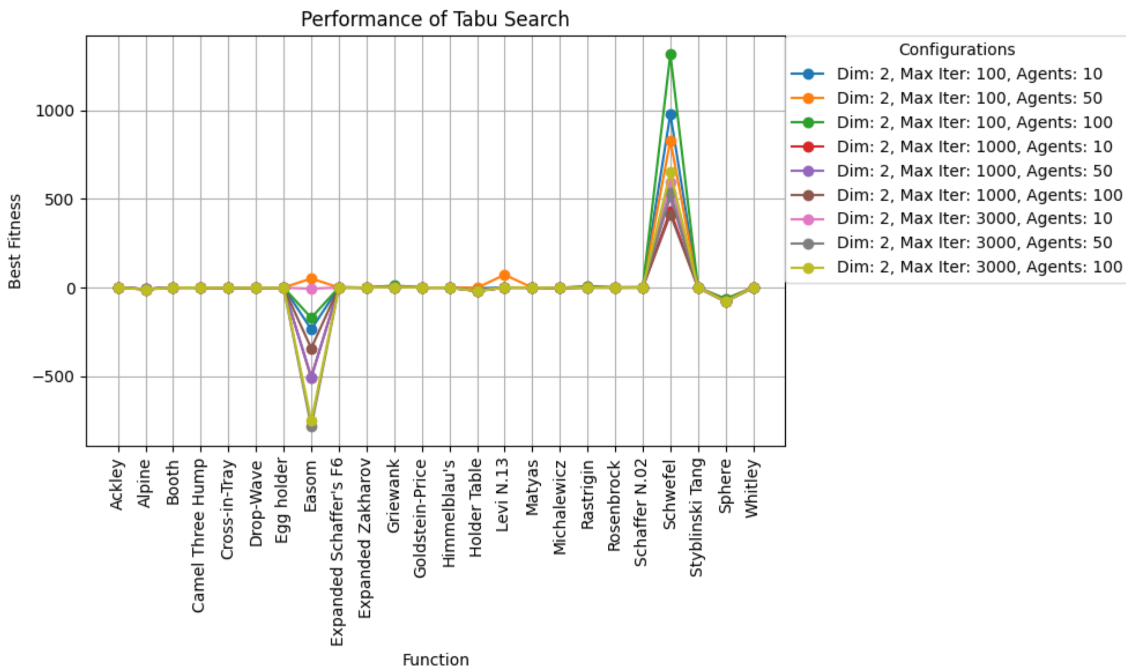
Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

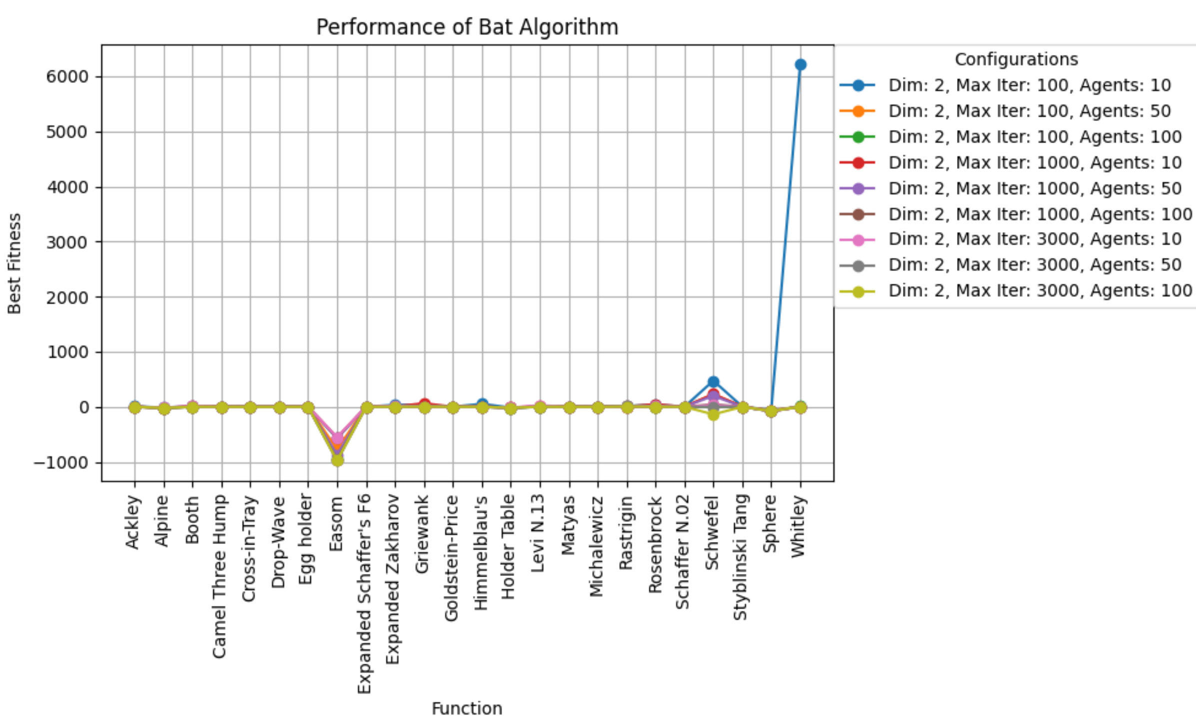




This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

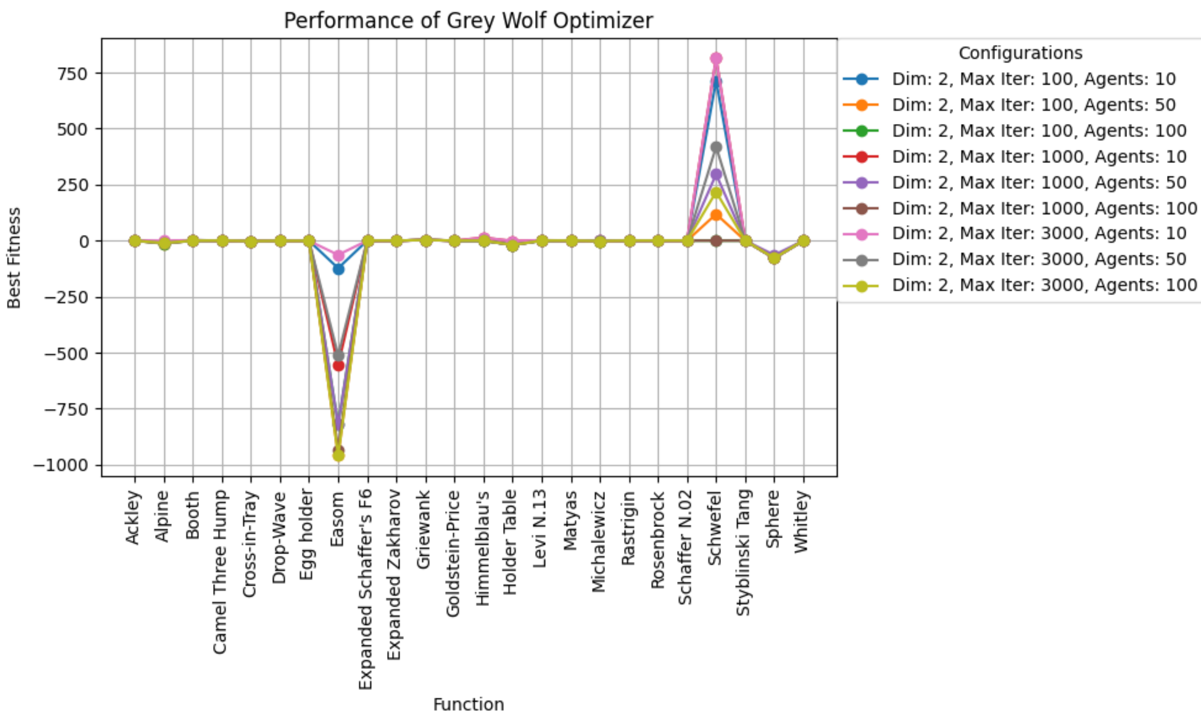
Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.





This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.



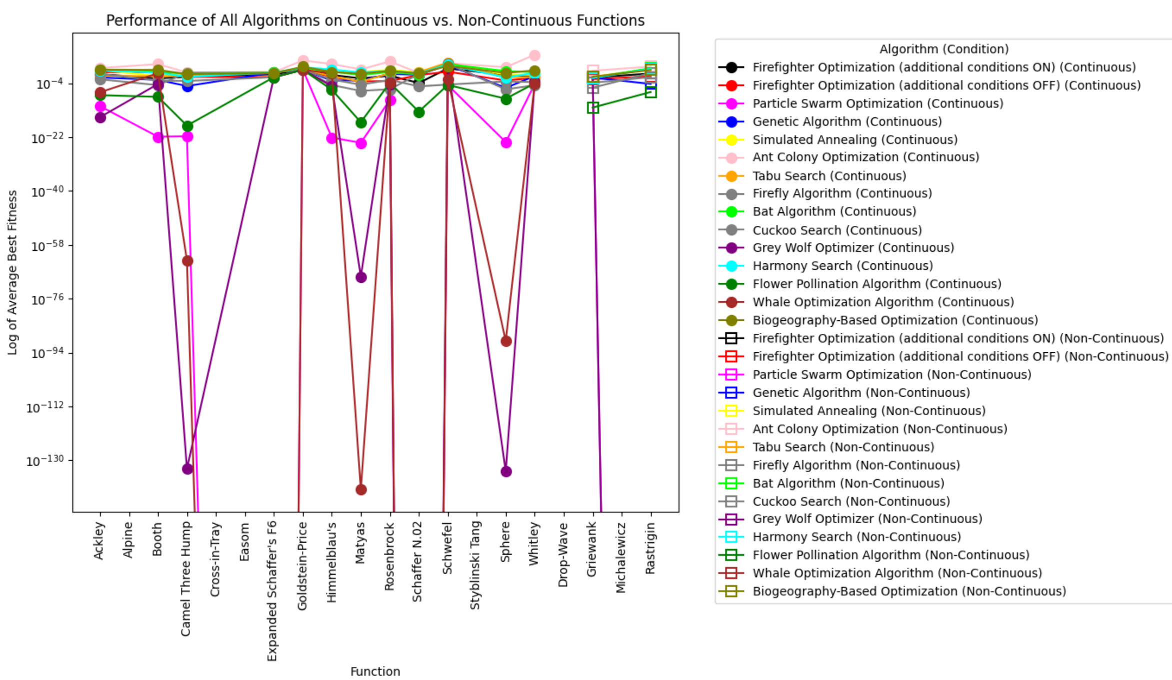


Fig. 4 Further cross examination between the FFO and other notable algorithms

To complement the above analysis, the Friedman and Wilcoxon non-parametric statistical tests for ranking the above algorithm were carried out [64,65]. These statistical methods can evaluate and compare the performance of optimization algorithms across multiple functions. The Friedman test

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

ranks algorithms based on their mean performance across all tested functions, where a lower rank indicates superior performance (i.e., achieving better, usually lower, fitness values). Each function is treated equally, and the ranks are averaged to provide an overall ranking for each algorithm. The test then checks if the observed rankings are statistically significant.

On the other hand, the Wilcoxon test focuses on pairwise comparisons between algorithms. This test calculates the signed rank of the differences in performance between each pair to assess whether one algorithm consistently outperforms the other. While both tests aim to highlight the best-performing algorithms, they approach the evaluation from slightly different perspectives—Friedman emphasizes overall ranking across all scenarios, while Wilcoxon emphasizes consistency in pairwise dominance. Figure 5 shows the outcome of these tests. As one can see, the FFO ranks 4 and 5 in these tests, respectively. PSO, GWO, and FPO rank in the top three spots.

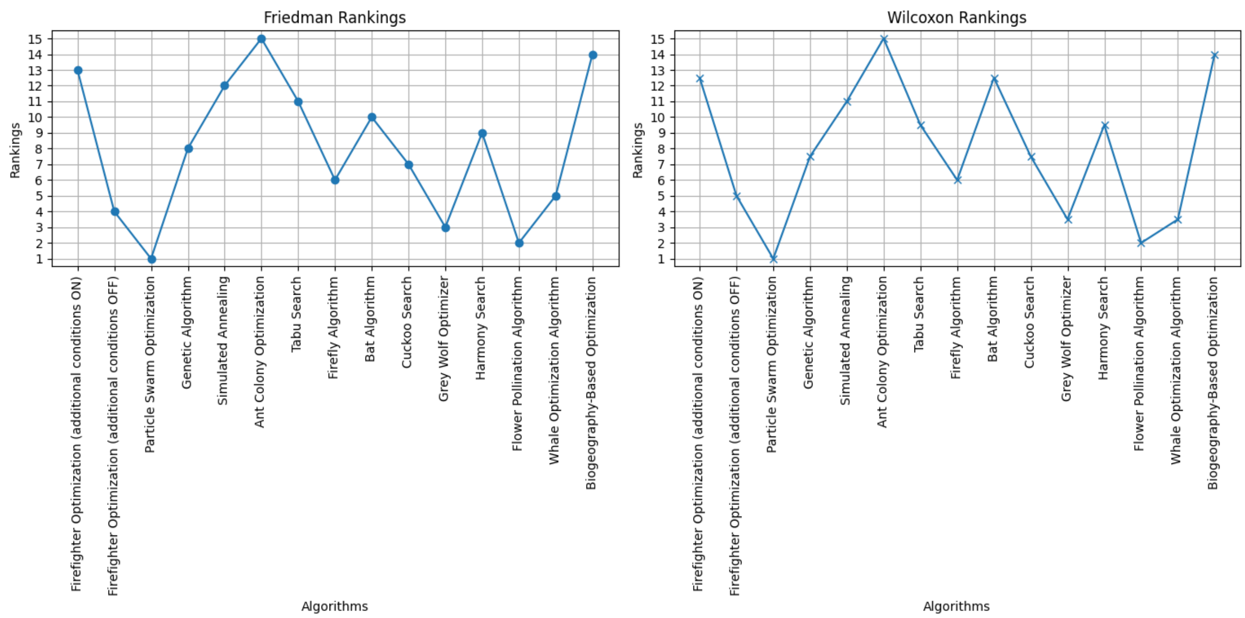


Fig. 5 Friedman and Wilcoxon non-parametric statistical tests for 2D

20D and 50D setting

Similar to the previous analysis and discussion, Table 4 and Fig. 6 list the overall obtained results from the analysis carried out on all algorithms and functions in higher dimensions of 20D and 50D. Thus, only scalable functions were used in this examination. These functions include Ackley, Eggholder, Easom, Expanded Shaffer's F6, Expanded Zakharov, Griewank, Gldstien-Price, Rosenbrock, Schaffer N.02, Schwefel, Sphere, and Whitely.

In this comparative analysis, the FFO (additional conditions OFF) demonstrates a creditable mean best fitness of 2.88E+03, which, while not the lowest in our dataset, offers a viable trade-off between fitness achievement and computational resources when compared to algorithms like the Simulated Annealing, which has a lower mean best fitness of 5.17E-02 but at a significantly

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

reduced complexity. In terms of execution time, the FFO (additional conditions OFF) records a mean of 9.35E+02 seconds, positioning it as a middle-range performer. It is notably faster than high-accuracy contenders such as the Firefly algorithm and the Biogeography-Based Optimization, which clocks in at mean times of 2.93E+02 seconds and 1.06E+01 seconds, respectively, reflecting a more efficient performance considering the relatively lower fitness figures.

On a more positive note, the FFO (additional conditions OFF) has signifncat exploration capability, as measured by the Total Distance metric, where it posts a mean of 1.58E+07. This is significantly higher than that of the Particle Swarm Optimization and Grey Wolf Optimizer, which stand at 2.98E+05 and 3.10E+04, respectively. Moreover, the Distance per Unit Time for the FFO (with conditions OFF) is large and stands at 2.95E+06, shadowing those of Cuckoo Search and Harmony Search, which report 3.41E+01 and 2.85E+03, respectively. This metric highlights the FFO's efficiency in covering large distances in the search space per unit of time, reinforcing its utility in expansive and complex problem spaces where speed and breadth of exploration are paramount. It is quite clear that the FFO ranked well in all metrics (see Fig. 6).

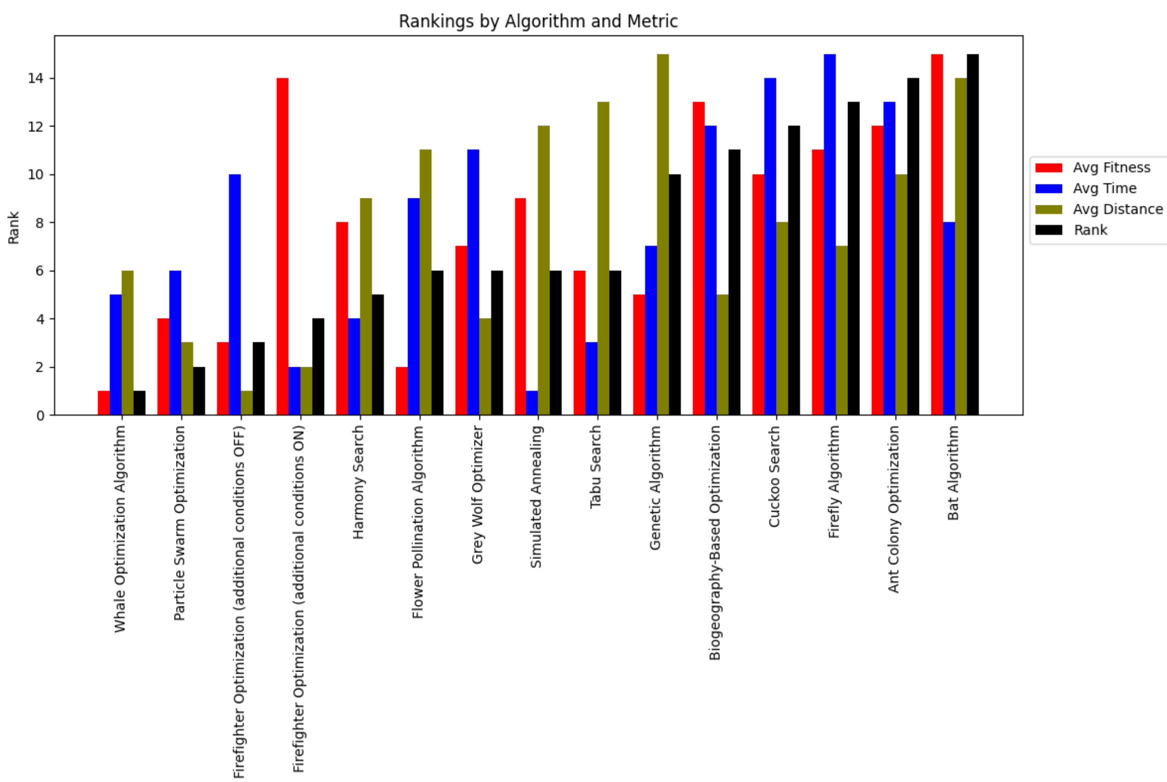


Fig. 6 Ranking of algorithms in 20D and 50D settings

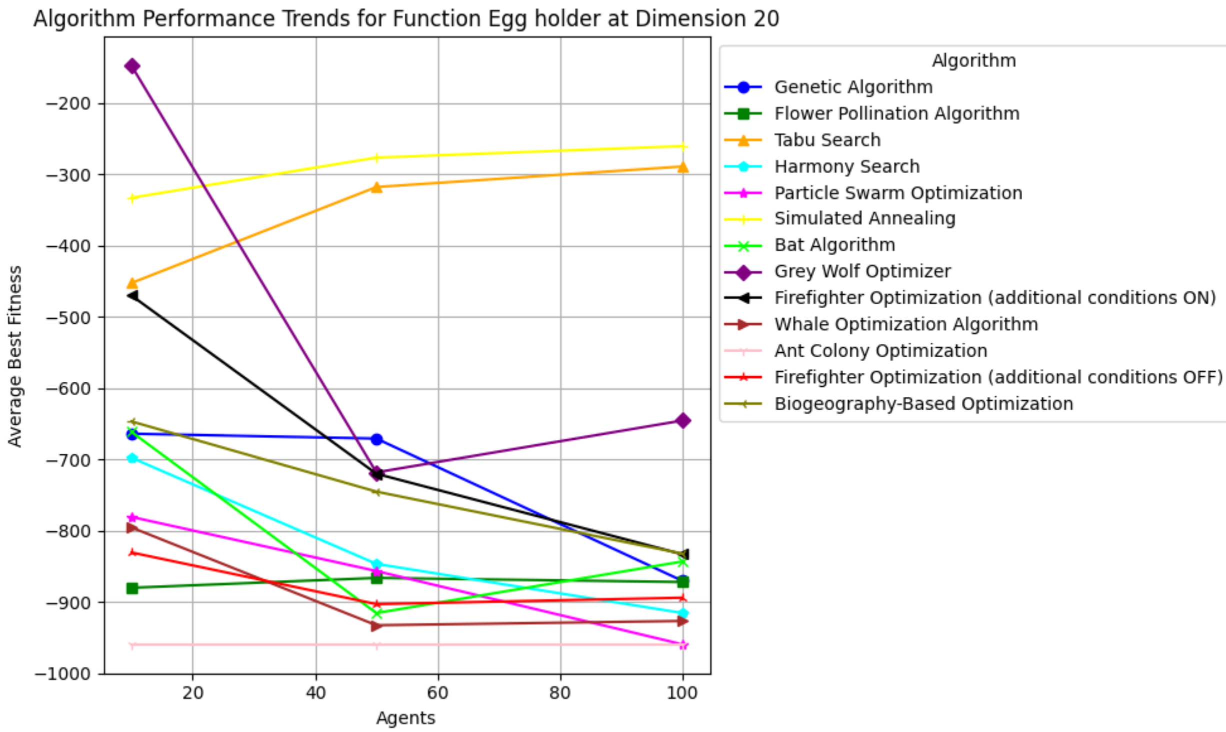
This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

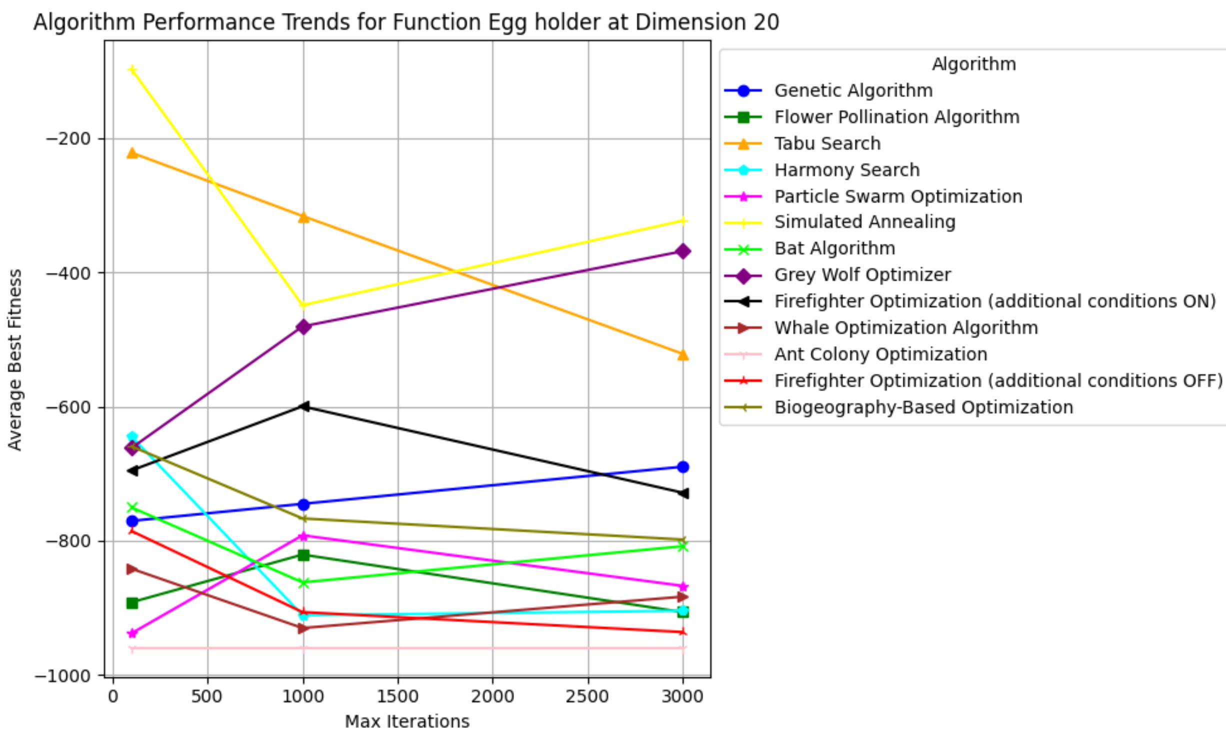
Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

Table 4 Overall results for 20D and 50D settings

Algorithm	Best Fitness				Execution Time (s)				Distance metric				Distance per Unit Time
	mean	std	min	max	mean	std	min	max	mean	std	min	max	
Ant Colony Optimization	7.61E+01	1.41E+02	9.04E-02	6.78E+02	9.42E+05	1.02E+07	-9.60E+02	1.48E+08	1.59E+03	4.57E+03	0.00E+00	3.20E+04	2.09E+01
Bat Algorithm	4.07E+00	8.43E+00	1.80E-02	7.49E+01	1.58E+04	7.26E+04	-9.77E+02	5.13E+05	9.75E+01	7.05E+02	0.00E+00	9.91E+03	2.39E+01
Biogeography-Based Optimization	1.06E+01	1.74E+01	7.30E-02	1.37E+02	1.22E+04	5.86E+04	-9.45E+02	4.44E+05	2.69E+04	9.81E+04	0.00E+00	1.24E+06	2.55E+03
Cuckoo Search	1.20E+02	3.66E+02	5.37E-02	3.63E+03	4.22E+03	2.62E+04	-9.60E+02	2.99E+05	4.10E+03	1.23E+04	0.00E+00	9.31E+04	3.41E+01
FFO (additional conditions OFF)	5.35E+00	1.00E+01	2.62E-02	8.16E+01	2.88E+03	2.19E+04	-9.60E+02	2.70E+05	1.58E+07	6.46E+07	1.50E+03	6.49E+08	2.95E+06
FFO (additional conditions ON)	1.81E-01	2.90E-01	0.00E+00	2.67E+00	9.01E+03	4.40E+04	-9.02E+02	3.41E+05	5.80E+05	1.95E+06	0.00E+00	1.78E+07	3.21E+06
Firefly Algorithm	2.93E+02	7.74E+02	1.36E-01	7.35E+03	1.03E+04	5.17E+04	-9.48E+02	5.16E+05	2.06E+03	9.80E+03	0.00E+00	1.02E+05	7.03E+00
Flower Pollination Algorithm	4.43E+00	8.81E+00	2.39E-02	7.74E+01	4.27E+02	2.82E+03	-9.60E+02	4.03E+04	8.12E+02	2.66E+03	3.89E-02	2.16E+04	1.83E+02
Genetic Algorithm	3.50E+00	5.87E+00	2.41E-02	4.73E+01	4.01E+02	2.17E+03	-9.60E+02	1.86E+04	9.72E+01	2.11E+02	0.00E+00	1.95E+03	2.78E+01
Grey Wolf Optimizer	8.40E+00	1.72E+01	4.20E-02	1.52E+02	7.23E+02	3.46E+03	-9.60E+02	2.09E+04	3.10E+04	1.05E+05	6.21E-02	8.51E+05	3.68E+03
Harmony Search	1.83E+00	4.06E+00	6.99E-03	3.69E+01	4.15E+03	2.89E+04	-9.60E+02	3.48E+05	5.22E+03	2.10E+04	0.00E+00	2.12E+05	1.57E+02
Particle Swarm Optimization	3.19E+00	5.31E+00	2.50E-02	4.31E+01	4.80E+02	2.67E+03	-9.60E+02	3.35E+04	2.98E+05	9.34E+05	3.24E+01	1.05E+07	4.45E+04
Simulated Annealing	5.17E-02	7.16E-02	9.97E-04	5.57E-01	4.58E+03	3.05E+04	-8.21E+02	3.73E+05	5.18E+02	7.21E+02	1.78E+00	4.96E+03	2.86E+03
Tabu Search	1.40E+00	3.07E+00	3.91E-03	2.14E+01	7.04E+02	3.04E+03	-7.53E+02	2.18E+04	5.04E+02	5.25E+02	2.17E+01	2.04E+03	7.01E+03
Whale Optimization Algorithm	3.19E+00	5.25E+00	2.54E-02	4.18E+01	-5.35E+01	4.24E+02	-9.60E+02	6.14E+03	2.10E+04	6.20E+04	9.85E-02	5.91E+05	1.52E+03

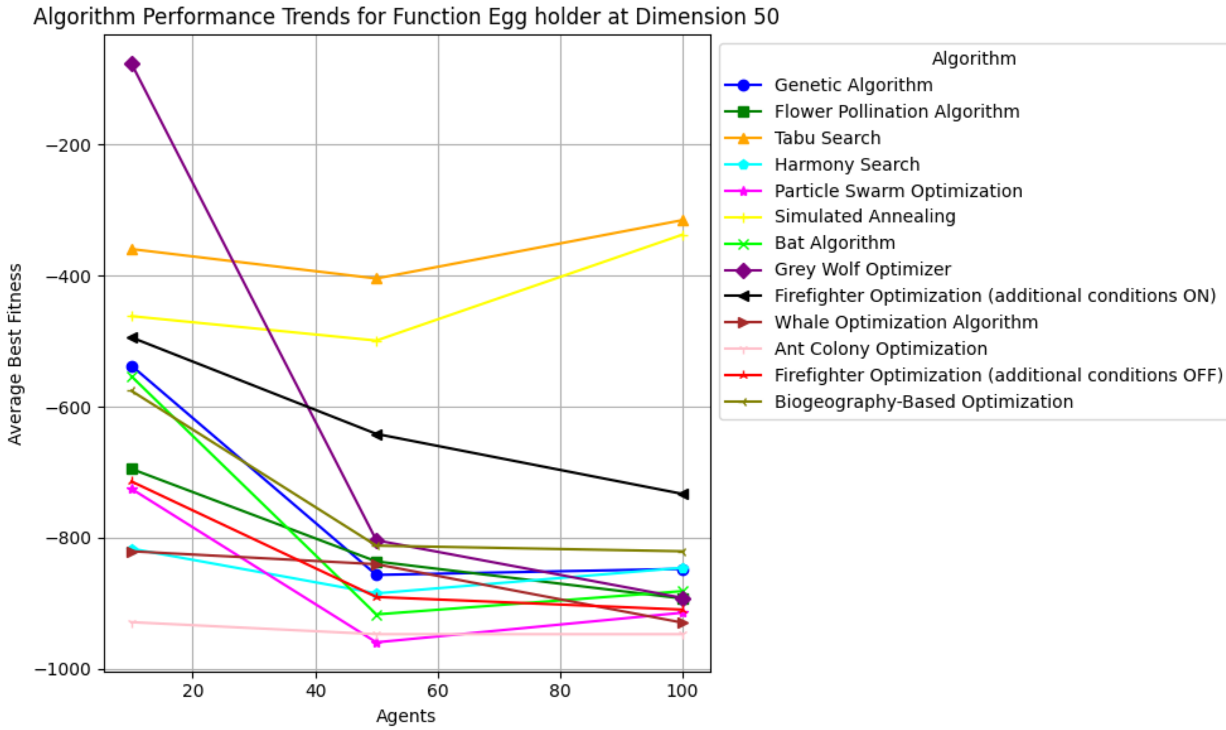
Now, we gain further insights into the performance of all algorithms across the settings of 20D and 50D, as seen in Fig. 7. It can be seen that the FFO performs consistently similarly to other algorithms at different settings of agents and iterations.





This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.



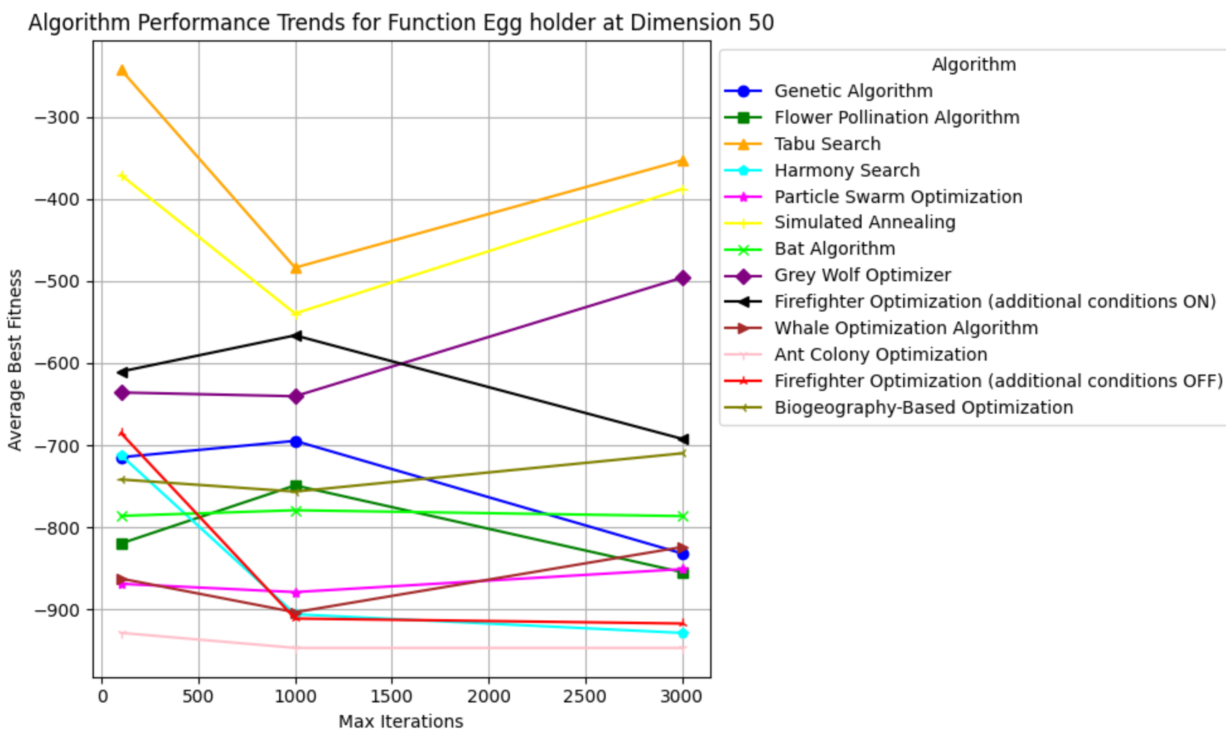


Fig. 7 Sample of trends across different experimental settings

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

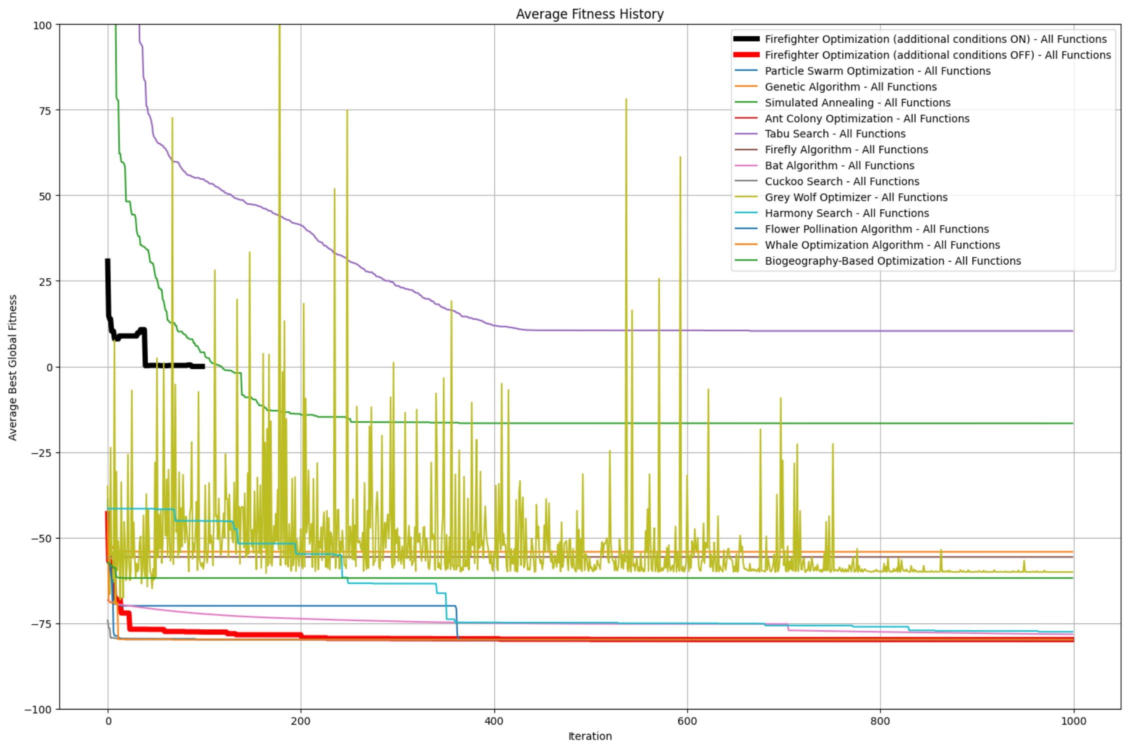
We revisit the ranking performance of the optimization algorithms across the longest (and shortest) time to solve and that achieved the most (and least) accuracy. Similar to the case of 2D, the Firefly algorithm leads this category with an appearance frequency of 75, indicating it often requires the most time to solve the selected benchmark functions. On the other hand, the Ant Colony Optimization appears far less frequently with a count of 6, suggesting a quicker resolution time but possibly at the expense of other performance metrics like accuracy or search depth. Then, the Simulated Annealing dominates the fastest to solve category and hence demonstrates its ability to swiftly find solutions. In terms of accuracy, Cuckoo Search stands out with 22 appearances, followed by the FFO (additional conditions OFF) and Particle Swarm Optimization, each scoring 13 occurrences. The Ant Colony Optimization leads as the least accurate with 29 appearances, followed by the Bat Algorithm at 19, suggesting these algorithms might prioritize exploration or speed over precision.

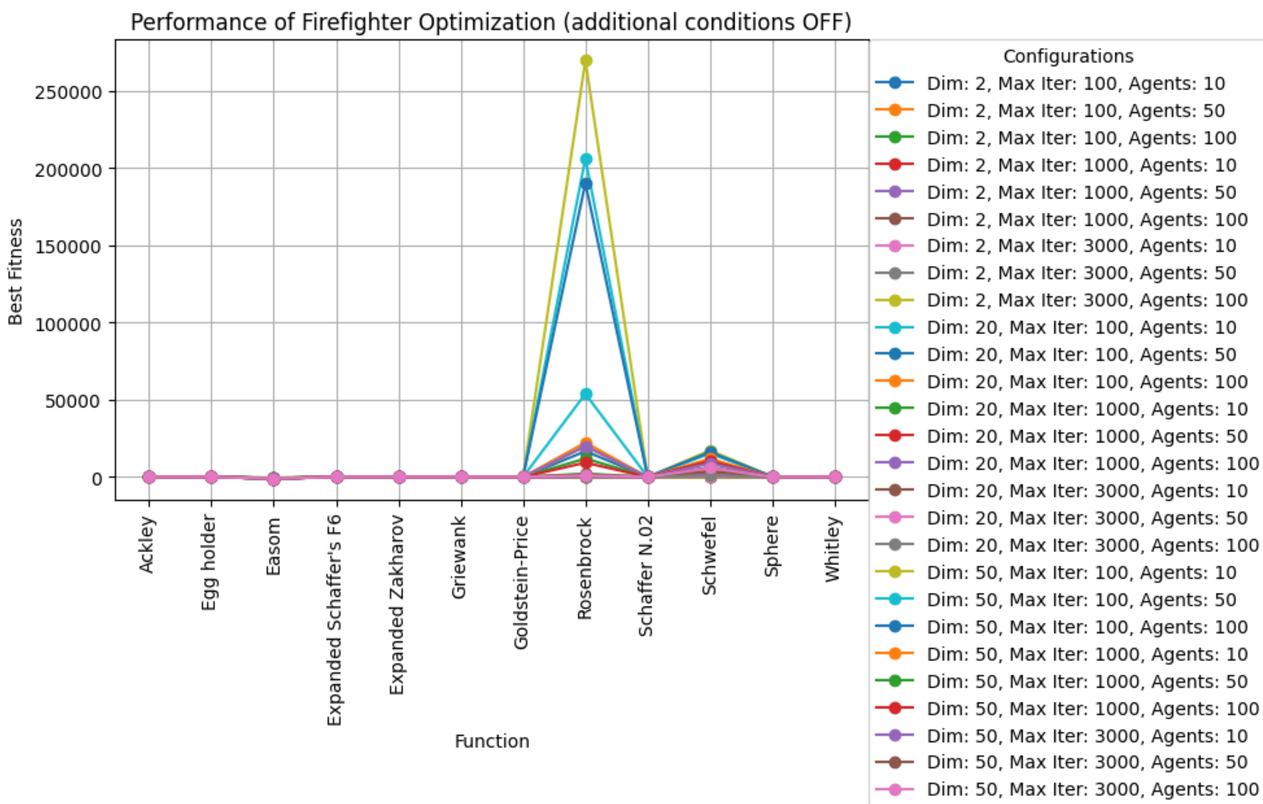
Figure 8 paints a series of comparisons between selected algorithms. Figure 8a compares the average history fitness across all functions and algorithms. This graph shows the average fitness history across iterations for various optimization algorithms. The FFO (additional conditions OFF) starts with a rapid convergence compared to other algorithms, which is particularly noticeable against the backdrop of more gradual improvements shown by the Genetic Algorithm and Simulated Annealing. The FFO (additional conditions OFF) demonstrates a stabilization around 200 iterations, where its fitness value flatlines. This early convergence suggests that, on average, this algorithm is efficient in quickly finding a promising area of the search space.

While Fig. 8b compares the best fitness achieved in FFO, Tabu Search, the Bat, and the Grey Wolf Optimizer algorithms. In Fig. 8b, the performance of FFO (additional conditions OFF) is plotted across various scalable functions under different settings, showing stable performance for all functions, except the Rosenbrock, and especially under conditions of higher iterations and larger agent numbers. These spikes indicate that FFO (additional conditions OFF) can excel in complex, multimodal landscapes but show variable performance dependent on the function's characteristics and the search space complexity. Comparing this performance to other algorithms, such as Tabu Search, Bat, and Grey Wolf Optimizer, across the same benchmark functions shows generally stable performance as well, with some exceptions where fitness spikes, similar to the behavior seen in FFO. Figure 8c shows the performance of all algorithms in tackling continuous and non-continuous scalable benchmarking functions. The FFO (additional conditions OFF) shows competitive performance in all function types. These figures reinforce the performance of the newly proposed FFO algorithm against notable algorithms.

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

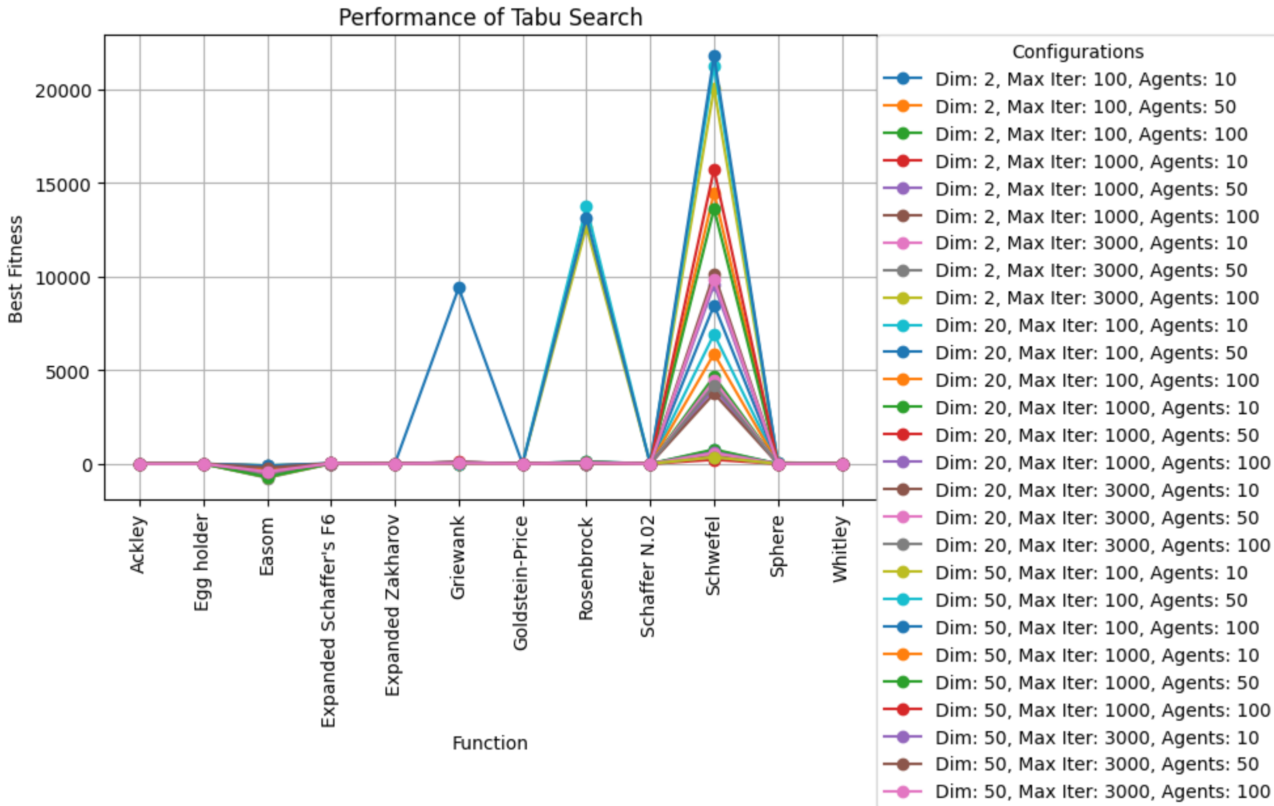
Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.





This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

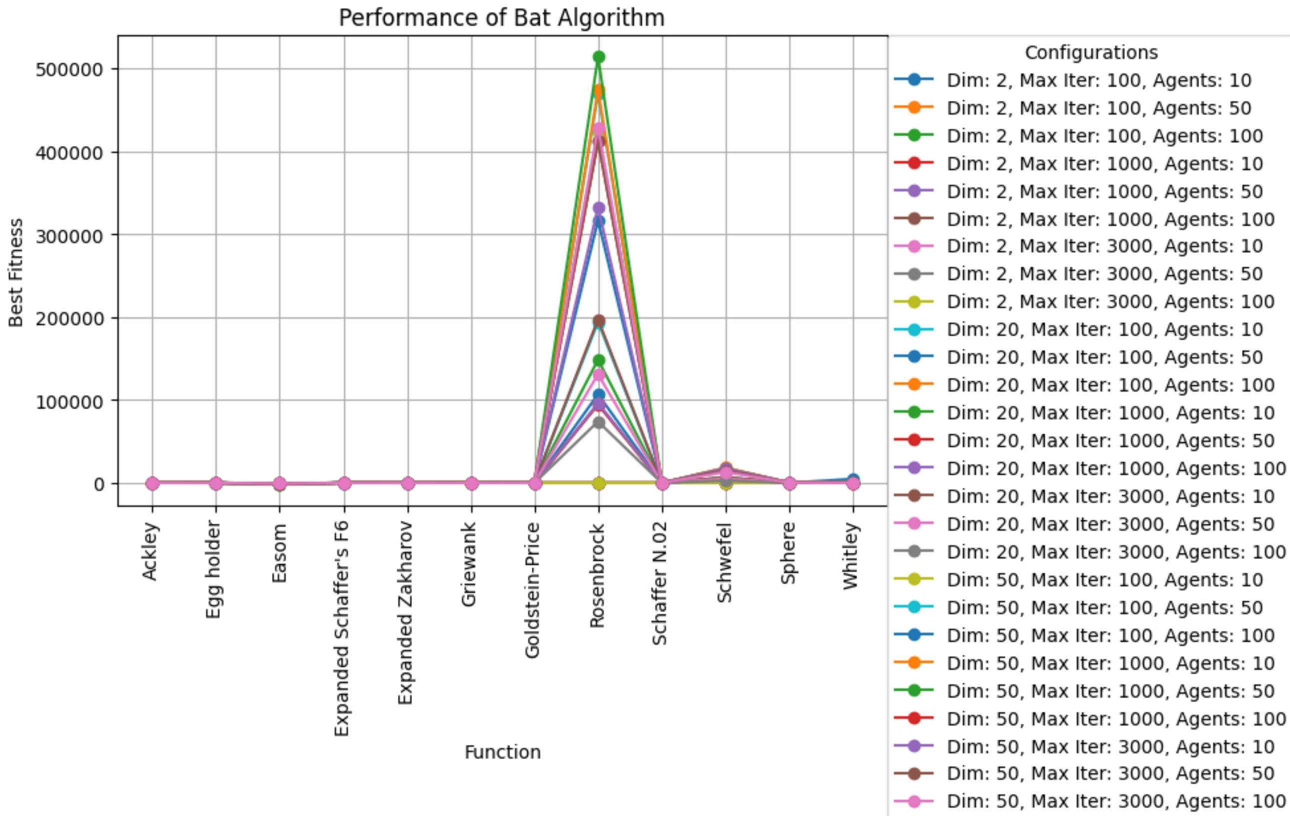
Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

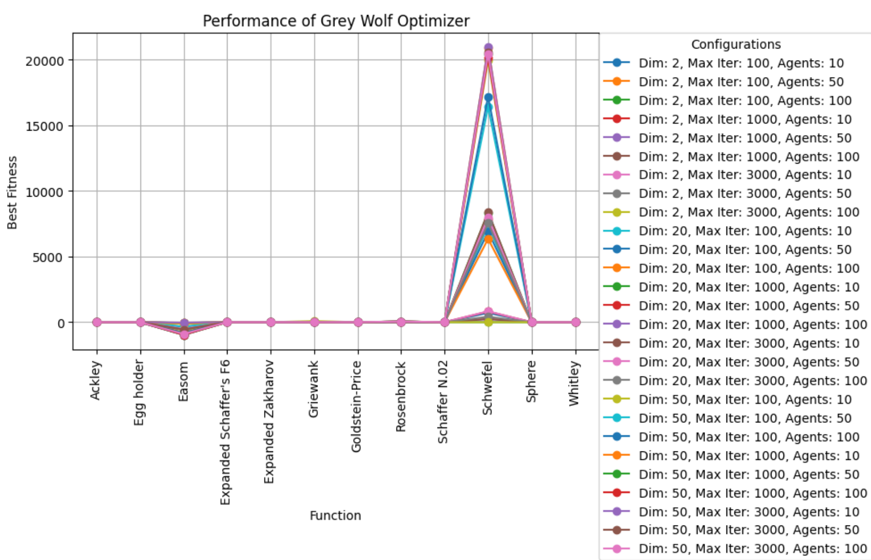


1009

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

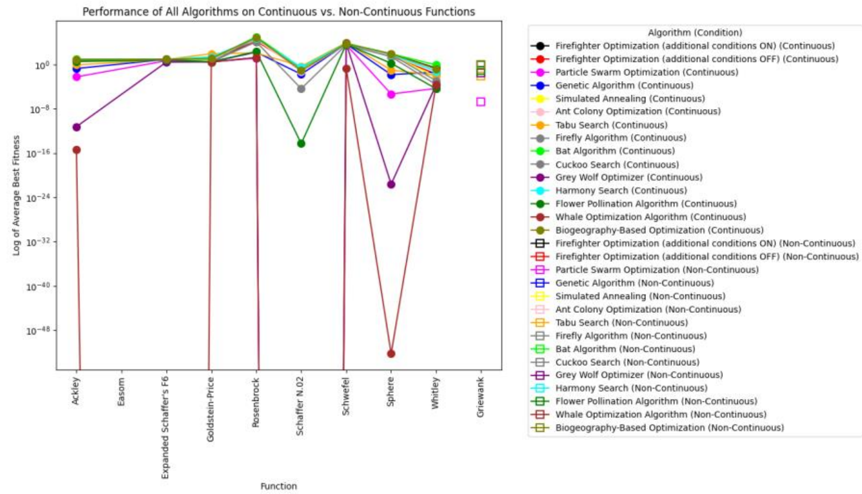
Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.





This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.



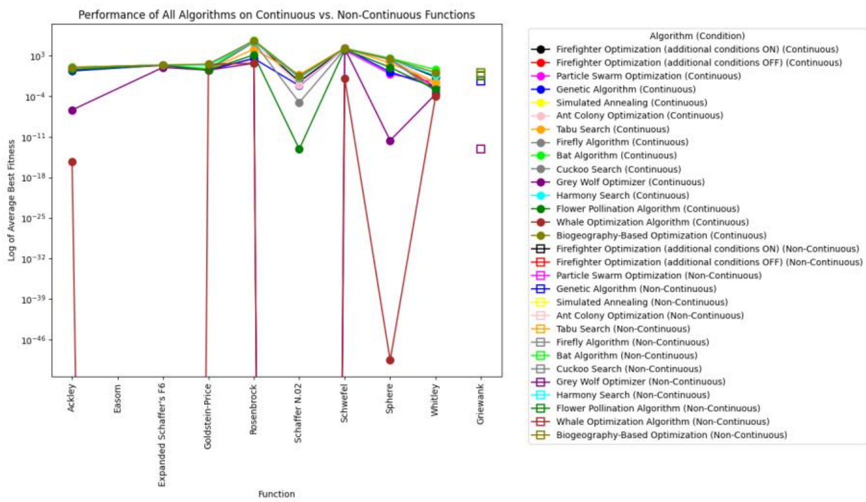


Fig. 8 Further cross examination between the FFO and other notable algorithms

The Friedman and Wilcoxon non-parametric statistical tests for ranking the above algorithm were again carried out for this leg of the investigation. Figure 9 shows that the FFO ranks 5 and 6.5 in 2D and 6 and 9.5 for 50D for these tests, respectively. PSO, GWO, FPA, and WOA rank in the top spots. As seen above, FFO always ranks better without constraints (i.e., conditions = off).

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

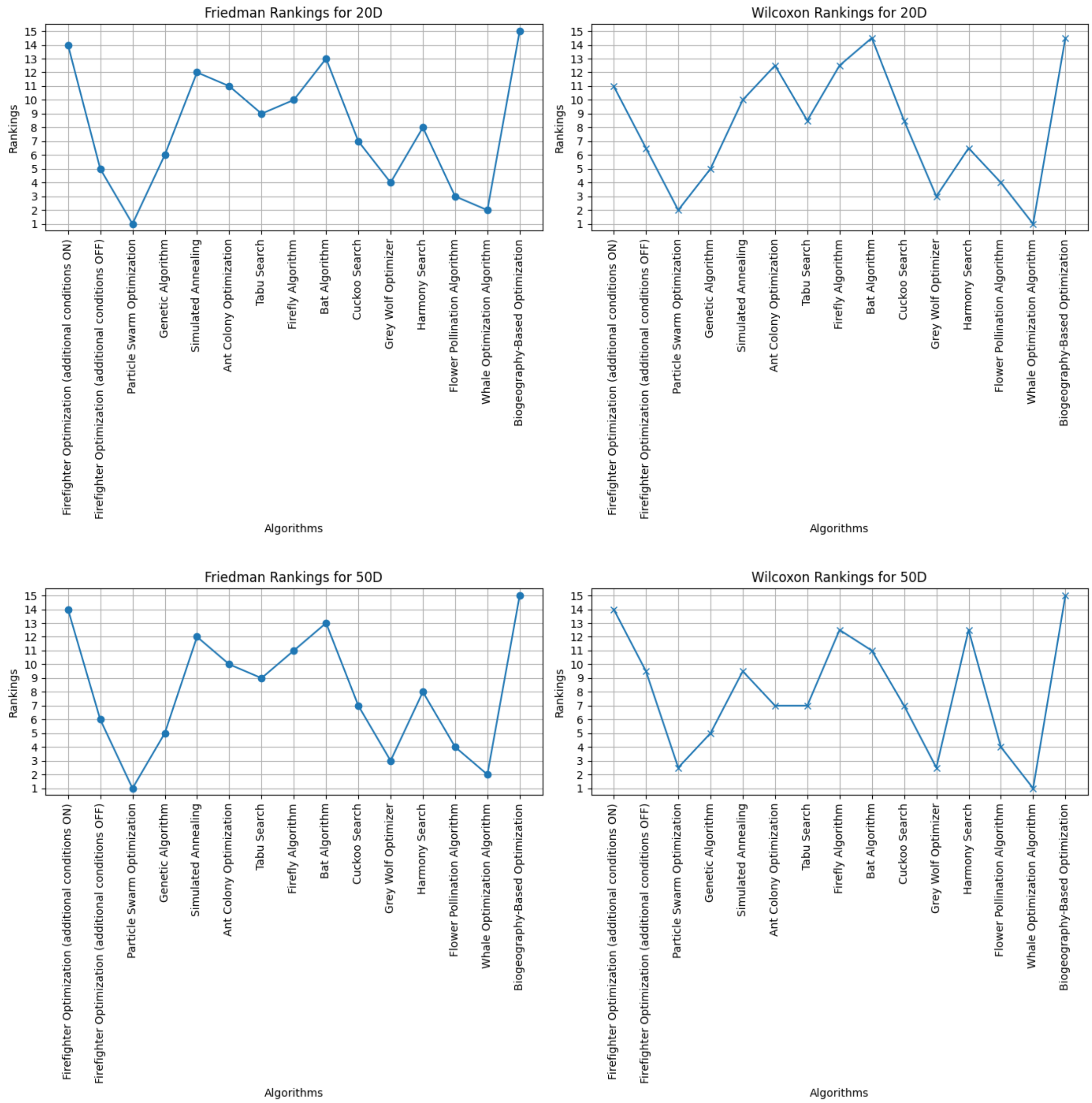


Fig. 9 Friedman and Wilcoxon non-parametric statistical tests carried out for higher dimensions (note: Top: 20D, Bottom: 50D)

Additional testing on functions from the CEC benchmarks (at 2D, 20D and 50D settings)

The IEEE Congress on Evolutionary Computation (IEEE CEC) is an annual conference that focuses on the latest developments and research in evolutionary computation. As part of this conference, benchmark functions are commonly used to evaluate the performance of optimization algorithms. These benchmark functions, often called CEC functions, are specifically designed to present various challenges to optimization methods, such as multimodality, non-separability, and

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

ruggedness. The CEC benchmark suites are updated periodically, and we examine some of the functions that appear in the 2020 edition. Each function typically represents a specific optimization problem with known difficulty, allowing for a comprehensive evaluation of an algorithm's strengths and weaknesses across various problem landscapes. Table 5 lists the CEC 2020 functions; additional details can be found in the CEC report [66].

Table 5 CEC 2020 functions.

No.	Functions	$F_i{*}=F_i(x{*})$	*Dimensions tested*
1	Shifted and Rotated Bent Cigar Function (also, CEC 2017 F1)	100	2, 20, 50
2	Shifted and Rotated Schwefel's Function (also, CEC 2014 F11)	1100	2, 20, 50
3	Shifted and Rotated Lunacek bi-Rastrigin Function (also, CEC 2017 F7)	700	2, 20, 50
4	Expanded Rosenbrock's plus Griewangk's Function (also, CEC 2017 F19)	1900	2, 20, 50
5	Hybrid Function 1 (N=3) (also, CEC 2014 F17)	1700	20, 50
6	Hybrid Function 2 (N=4) (also, CEC 2017 F16)	1600	20, 50
7	Hybrid Function 3 (N=5) (also, CEC 2014 F21)	2100	20, 50
8	Composition Function 1 (N=3) (also, CEC 2017 F22)	2200	2, 20, 50
9	Composition Function 2 (N=4) (also, CEC 2017 F24)	2400	2, 20, 50
10	Composition Function 3 (N=5) (also, CEC 2017 F25)	2500	2, 20, 50

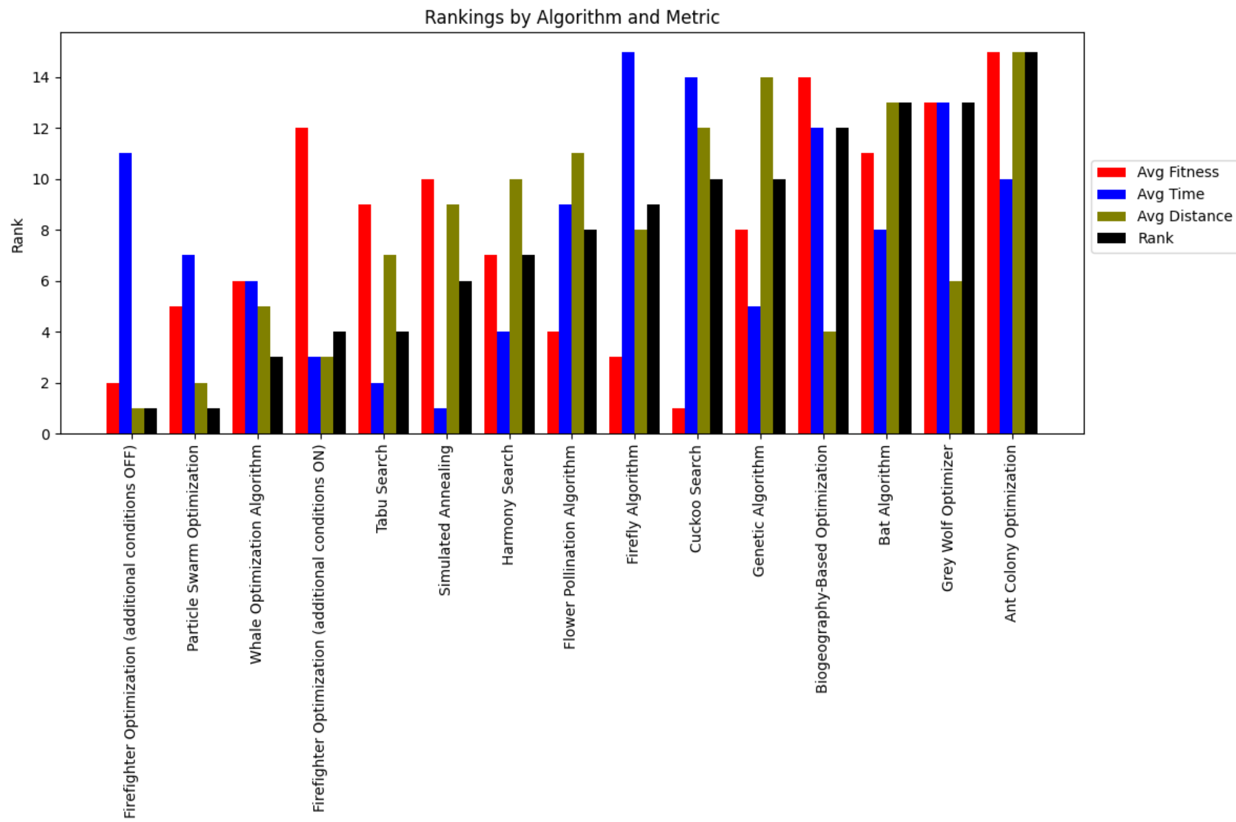
The outcome of this analysis is listed in Table 6. This table shows that both FFO and PSW ranked first in terms of overall ranking among all metrics. In addition, Fig. 10 shows that the best version of FFO ranked between 3 and 6 for the Friedman and Wilcoxon non-parametric statistical tests, respectively. Overall, PSO, GA, and CS achieved the best rankings in this experiment.

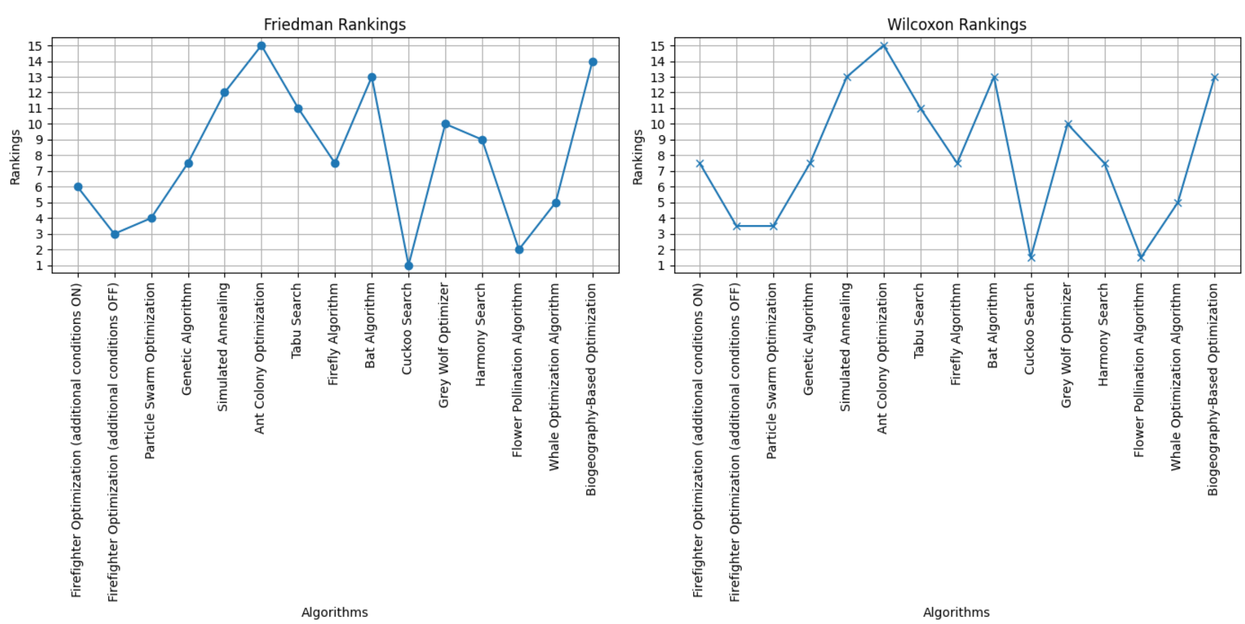
This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

Table 6 Overall results for 2D, 20D, and 50D settings

Algorithm	Best Fitness				Execution Time (s)				Distance metric				Distance per Unit Time
	mean	std	min	max	mean	std	min	max	mean	std	min	max	
2D													
Ant Colony Optimization	1.30E+08	3.57E+08	1.43E+03	1.20E+09	1.15E+01	1.58E+01	1.27E-01	6.76E+01	7.90E+00	3.60E+01	0.00E+00	2.00E+02	6.86E-01
Bat Algorithm	4.82E+06	3.55E+07	8.03E+01	2.82E+08	9.10E+00	1.45E+01	2.70E-02	6.32E+01	1.64E+01	4.79E+01	0.00E+00	2.68E+02	1.80E+00
Biogeography-Based Optimization	6.95E+07	4.58E+08	4.34E+02	3.56E+09	1.79E+01	2.67E+01	9.80E-02	1.12E+02	5.47E+03	1.13E+04	0.00E+00	7.82E+04	3.05E+02
Cuckoo Search	1.37E+03	9.97E+02	-3.23E+01	2.91E+03	3.36E+02	6.73E+02	1.27E-01	3.07E+03	9.79E+01	9.71E+01	1.11E+00	4.88E+02	2.91E-01
FFO (additional conditions OFF)	9.30E+04	7.24E+05	-3.24E+01	5.75E+06	1.33E+01	2.30E+01	6.80E-02	1.01E+02	4.13E+06	7.28E+06	8.82E+04	4.05E+07	3.10E+05
FFO (additional conditions ON)	5.78E+05	4.49E+06	2.99E+01	3.56E+07	4.71E-01	6.22E-01	1.90E-02	4.02E+00	3.28E+05	3.09E+05	3.33E+04	1.74E+06	6.97E+05
Firefly Algorithm	2.49E+07	1.89E+08	-2.72E+01	1.50E+09	7.20E+02	1.42E+03	2.66E-01	6.58E+03	2.99E+02	4.82E+02	0.00E+00	2.29E+03	4.16E-01
Flower Pollination Algorithm	1.49E+03	1.03E+03	2.40E+01	2.92E+03	9.40E+00	1.47E+01	3.20E-02	6.43E+01	7.75E+01	6.90E+01	5.10E+00	2.78E+02	8.25E+00
Genetic Algorithm	1.89E+03	1.04E+03	2.13E+02	4.96E+03	5.04E+00	8.00E+00	3.10E-02	3.69E+01	1.44E+01	1.83E+01	1.37E-01	1.03E+02	2.86E+00
Grey Wolf Optimizer	2.37E+03	1.54E+03	2.13E+02	5.63E+03	1.87E+01	2.96E+01	6.70E-02	1.28E+02	6.63E+03	8.23E+03	9.07E+01	3.49E+04	3.55E+02
Harmony Search	1.56E+05	8.66E+05	1.60E+02	6.20E+06	4.31E+00	6.99E+00	1.60E-02	3.03E+01	8.13E+01	9.96E+01	0.00E+00	6.73E+02	1.88E+01
Particle Swarm Optimization	1.72E+03	1.12E+03	-3.24E+01	5.62E+03	5.63E+00	8.73E+00	4.20E-02	4.06E+01	1.66E+05	5.65E+05	1.82E+03	3.82E+06	2.95E+04
Simulated Annealing	5.52E+08	2.81E+09	1.51E+02	1.59E+10	9.67E-02	1.11E-01	1.03E-03	3.79E-01	9.49E+01	6.39E+01	2.27E+01	2.69E+02	9.81E+02
Tabu Search	6.58E+08	3.17E+09	1.11E+02	1.87E+10	2.21E-01	2.35E-01	1.10E-02	7.94E-01	5.77E+02	5.14E+02	2.89E+01	1.51E+03	7.01E+03
Whale Optimization Algorithm	2.51E+04	1.79E+05	2.24E+02	1.42E+06	5.65E+00	8.40E+00	3.70E-02	3.59E+01	3.73E+03	1.30E+04	2.94E-02	9.68E+04	1.52E+03
20D													
Ant Colony Optimization	2.65E+09	8.24E+09	4.62E+02	3.57E+10	5.11E+01	6.37E+01	3.71E-01	2.23E+02	1.85E+03	8.06E+02	0.00E+00	3.45E+03	3.62E+01
Bat Algorithm	6.61E+09	1.89E+10	-2.62E+03	8.31E+10	1.03E+01	1.60E+01	1.64E-02	7.78E+01	5.03E+01	2.72E+02	0.00E+00	2.14E+03	4.87E+00
Biogeography-Based Optimization	5.12E+09	1.64E+10	2.55E+03	9.25E+10	1.87E+01	2.66E+01	4.89E-02	1.14E+02	4.00E+04	5.07E+04	0.00E+00	2.41E+05	2.14E+03
Cuckoo Search	2.19E+08	1.22E+09	-3.52E+03	8.52E+09	3.86E+02	7.26E+02	6.76E-02	3.29E+03	3.46E+03	1.79E+03	6.28E+02	8.95E+03	8.96E+00
FFO (additional conditions OFF)	1.09E+09	5.71E+09	-5.30E+03	4.41E+10	1.12E+01	1.67E+01	2.69E-02	7.25E+01	9.78E+06	1.10E+07	3.31E+05	4.18E+07	8.73E+05
FFO (additional conditions ON)	4.14E+09	1.36E+10	1.61E+03	8.26E+10	4.58E-01	4.52E-01	8.18E-03	2.13E+00	1.08E+06	8.76E+05	1.14E+05	4.05E+06	2.35E+06
Firefly Algorithm	7.10E+09	2.00E+10	7.12E+02	9.67E+10	7.83E+02	1.46E+03	1.46E-01	6.43E+03	2.26E+03	4.21E+03	0.00E+00	2.21E+04	2.89E+00
Flower Pollination Algorithm	1.13E+08	9.81E+08	-3.44E+03	9.29E+09	1.03E+01	1.55E+01	1.65E-02	7.08E+01	1.36E+03	5.65E+02	4.54E+02	2.80E+03	1.32E+02
Genetic Algorithm	1.24E+09	7.11E+09	-1.72E+03	5.63E+10	6.42E+00	9.13E+00	2.55E-02	4.11E+01	4.70E+02	4.02E+02	5.16E+01	1.62E+03	7.32E+01
Grey Wolf Optimizer	7.37E+08	3.09E+09	-9.46E+02	1.93E+10	2.06E+01	3.06E+01	3.25E-02	1.32E+02	4.25E+04	3.87E+04	3.35E+03	1.30E+05	2.06E+03
Harmony Search	1.21E+09	5.96E+09	-4.65E+03	4.49E+10	5.16E+00	7.54E+00	1.69E-02	3.30E+01	2.85E+03	2.56E+03	0.00E+00	1.50E+04	5.53E+02
Particle Swarm Optimization	2.97E+08	1.81E+09	-1.68E+03	1.54E+10	6.06E+00	8.65E+00	9.51E-03	3.73E+01	1.21E+05	1.79E+05	1.01E+04	1.24E+06	1.99E+04
Simulated Annealing	4.98E+09	2.48E+10	-1.65E+03	1.71E+11	1.05E-01	1.14E-01	0.00E+00	4.56E-01	8.90E+02	7.58E+02	9.54E+00	2.63E+03	8.44E+03
Tabu Search	3.90E+09	1.64E+10	-1.89E+03	1.05E+11	2.25E+00	2.34E+00	2.06E-02	7.89E+00	5.84E+02	5.05E+02	5.60E+01	1.77E+03	2.60E+02
Whale Optimization Algorithm	3.11E+09	9.79E+09	1.58E+03	4.70E+10	5.91E+00	8.41E+00	1.57E-02	3.61E+01	1.84E+04	1.57E+04	6.51E+02	5.97E+04	3.12E+03
50D													
Ant Colony Optimization	1.85E+10	5.18E+10	7.57E+03	2.09E+11	1.18E+02	1.45E+02	1.45E+00	5.02E+02	3.26E+03	1.34E+03	5.48E+02	6.52E+03	2.77E+01
Bat Algorithm	3.69E+10	8.22E+10	-1.85E+03	3.27E+11	1.05E+01	1.55E+01	3.36E-02	8.09E+01	6.49E+01	2.63E+02	0.00E+00	1.84E+03	6.21E+00
Biogeography-Based Optimization	2.92E+10	6.74E+10	7.83E+03	2.67E+11	2.08E+01	2.93E+01	8.32E-02	1.50E+02	6.94E+04	8.48E+04	0.00E+00	3.52E+05	3.33E+03
Cuckoo Search	3.77E+09	1.99E+10	-6.37E+02	1.32E+11	3.94E+02	7.39E+02	9.99E-02	4.15E+03	8.57E+03	4.31E+03	2.15E+03	2.48E+04	2.17E+01
FFO (additional conditions OFF)	9.99E+09	3.85E+10	-1.74E+03	2.37E+11	1.19E+01	1.77E+01	4.70E-02	9.02E+01	1.66E+07	1.96E+07	5.24E+05	8.41E+07	1.39E+06
FFO (additional conditions ON)	2.45E+10	6.20E+10	8.85E+03	2.31E+11	5.04E-01	4.91E-01	1.42E-02	2.27E+00	1.71E+06	1.32E+06	1.81E+05	5.88E+06	3.39E+06
Firefly Algorithm	3.71E+10	7.91E+10	8.31E+03	2.75E+11	8.20E+02	1.53E+03	2.12E-01	8.64E+03	2.75E+03	6.27E+03	0.00E+00	4.05E+04	3.35E+00
Flower Pollination Algorithm	9.95E+08	6.53E+09	-1.77E+03	5.87E+10	1.08E+01	1.61E+01	3.27E-02	8.78E+01	3.30E+03	1.33E+03	3.62E+02	6.91E+03	3.05E+02
Genetic Algorithm	1.03E+10	4.35E+10	-3.59E+03	3.07E+11	7.94E+00	1.09E+01	4.99E-02	5.24E+01	1.08E+03	9.81E+02	1.00E+02	3.82E+03	1.36E+02
Grey Wolf Optimizer	3.80E+09	1.38E+10	2.07E+03	8.35E+10	2.15E+01	3.22E+01	7.34E-02	1.71E+02	6.80E+04	6.53E+04	5.60E+03	2.49E+05	3.15E+03
Harmony Search	9.83E+09	3.29E+10	-7.79E+03	2.09E+11	5.66E+00	7.96E+00	4.68E-02	4.25E+01	5.24E+03	4.16E+03	0.00E+00	2.22E+04	9.26E+02
Particle Swarm Optimization	2.04E+09	1.02E+10	-2.89E+03	7.03E+10	6.36E+00	9.28E+00	3.11E-02	4.93E+01	2.78E+05	6.00E+05	1.68E+04	5.13E+06	4.37E+04
Simulated Annealing	2.35E+10	7.58E+10	1.72E+03	3.82E+11	1.09E-01	1.24E-01	0.00E+00	5.94E-01	2.07E+03	1.84E+03	4.15E+01	6.29E+03	1.91E+04
Tabu Search	3.13E+10	8.78E+10	-3.73E+03	3.94E+11	5.59E+00	6.11E+00	6.74E-02	2.30E+01	8.88E+02	9.18E+02	6.63E+01	2.73E+03	1.59E+02
Whale Optimization Algorithm	1.34E+10	3.27E+10	8.56E+03	1.30E+11	6.11E+00	8.77E+00	1.22E-02	4.60E+01	2.97E+04	2.56E+04	8.29E+02	8.30E+04	4.85E+03

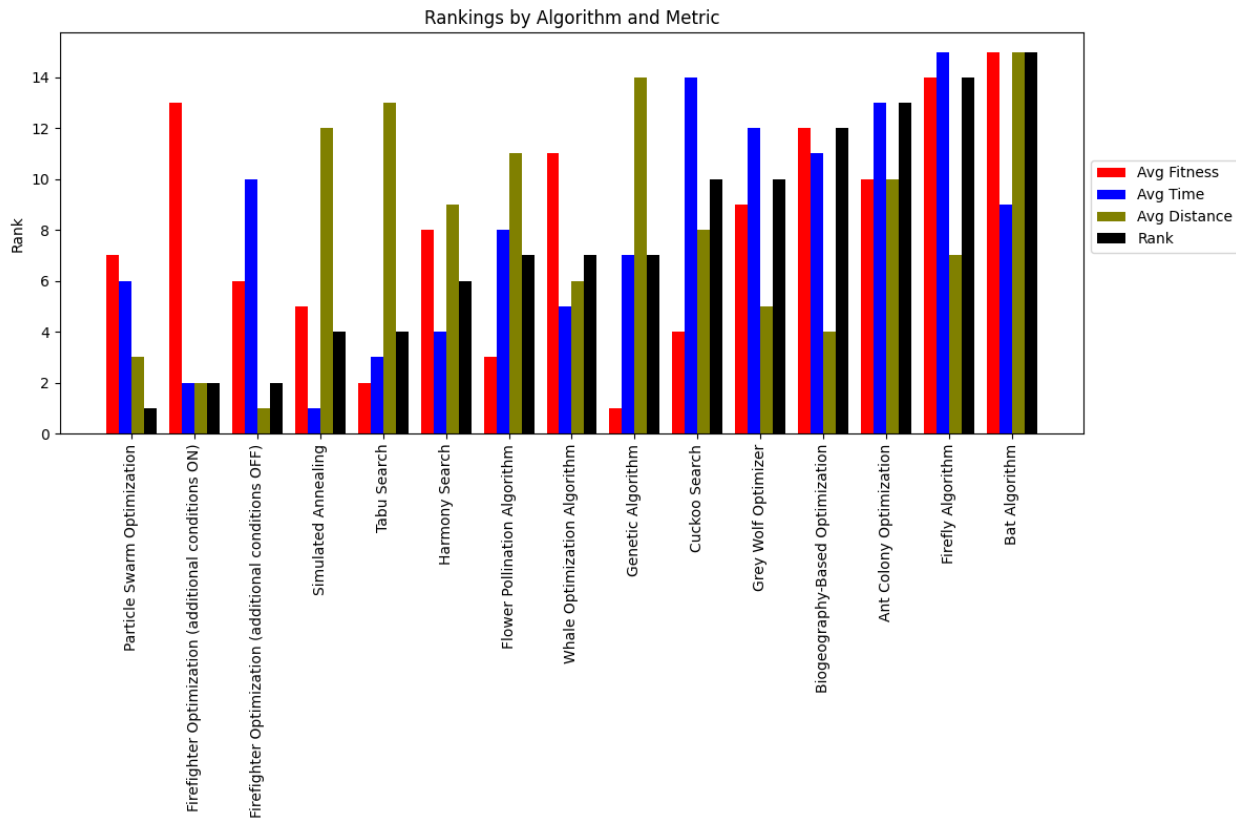

Rankings by Algorithm and Metric
Rank
Avg Fitness
Avg Time
Avg Distance
Rank
Firefighter Optimization (additional conditions OFF)
Particle Swarm Optimization
Whale Optimization Algorithm
Firefighter Optimization (additional conditions ON)
Tabu Search
Simulated Annealing
Harmony Search
Flower Pollination Algorithm
Firefly Algorithm
Cuckoo Search
Genetic Algorithm
Biogeography-Based Optimization
Bat Algorithm
Grey Wolf Optimizer
Ant Colony Optimization

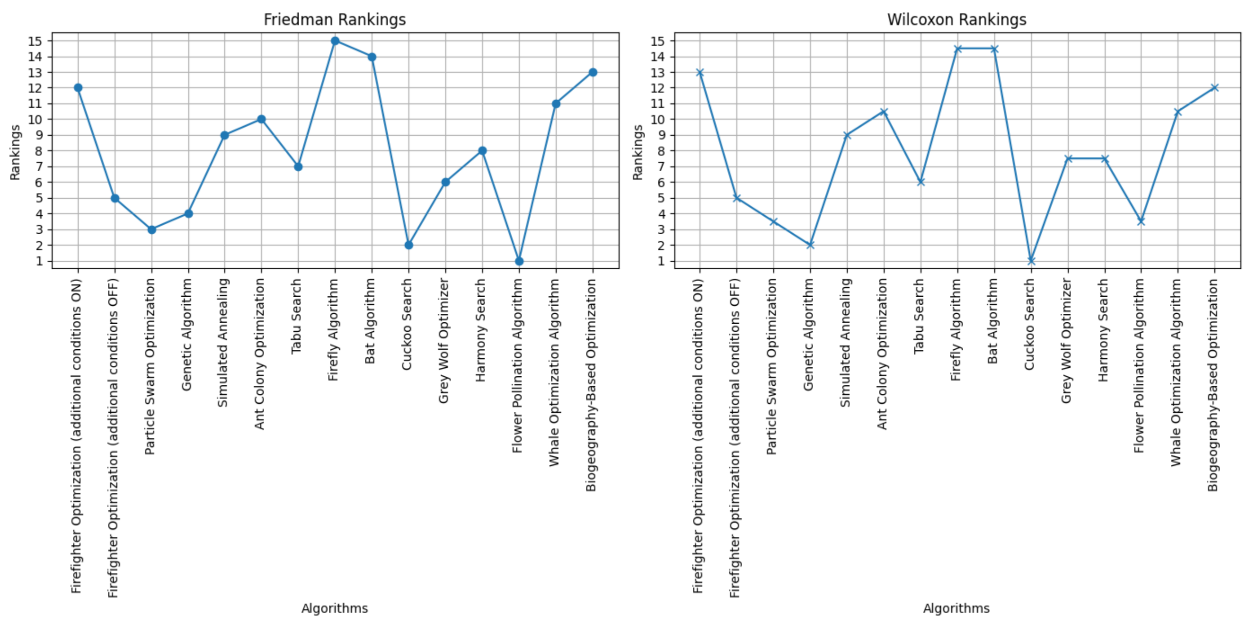

Friedman Rankings
Rankings
Algorithms
Wilcoxon Rankings
Rankings
Algorithms
Firefighter Optimization (additional conditions ON)
Firefighter Optimization (additional conditions OFF)
Particle Swarm Optimization
Genetic Algorithm
Simulated Annealing
Ant Colony Optimization
Tabu Search
Firefly Algorithm
Bat Algorithm
Cuckoo Search
Grey Wolf Optimizer
Harmony Search
Flower Pollination Algorithm
Whale Optimization Algorithm
Biogeography-Based Optimization

(a) At 2D

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

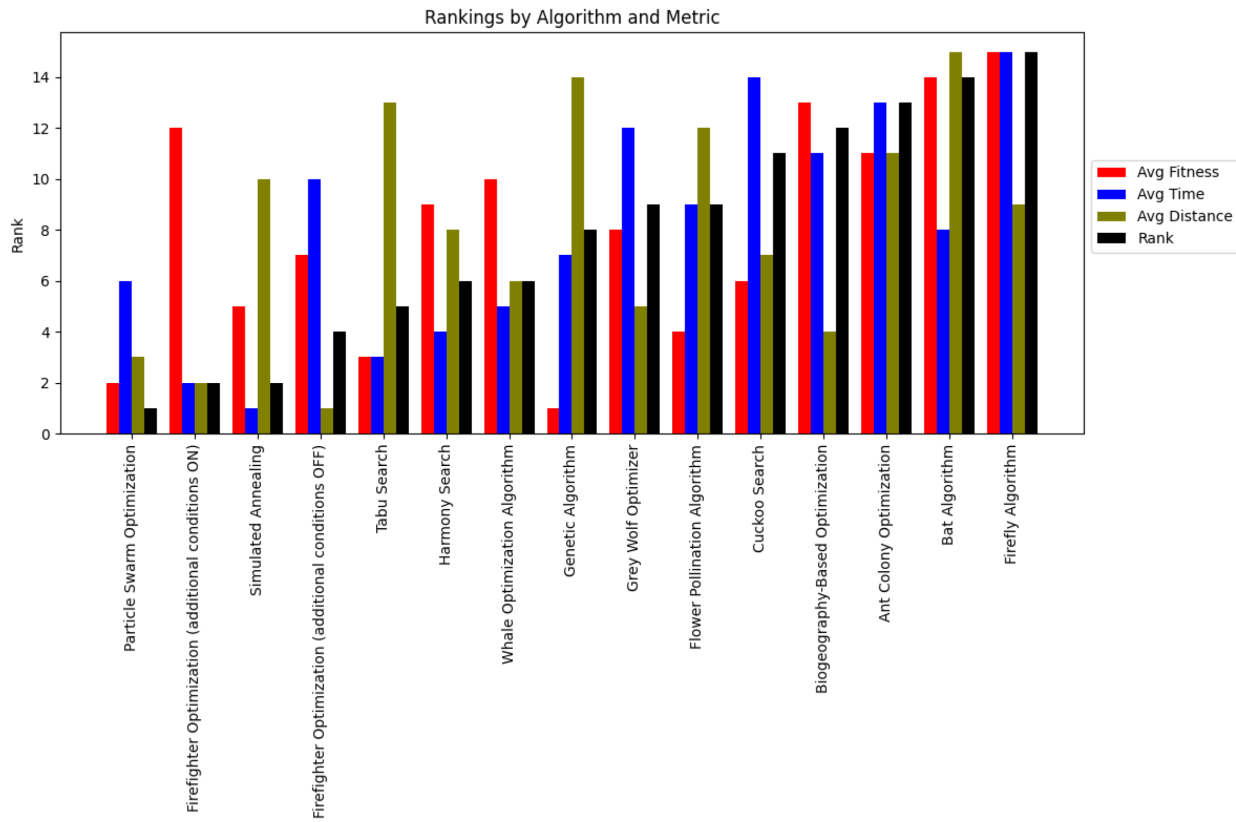


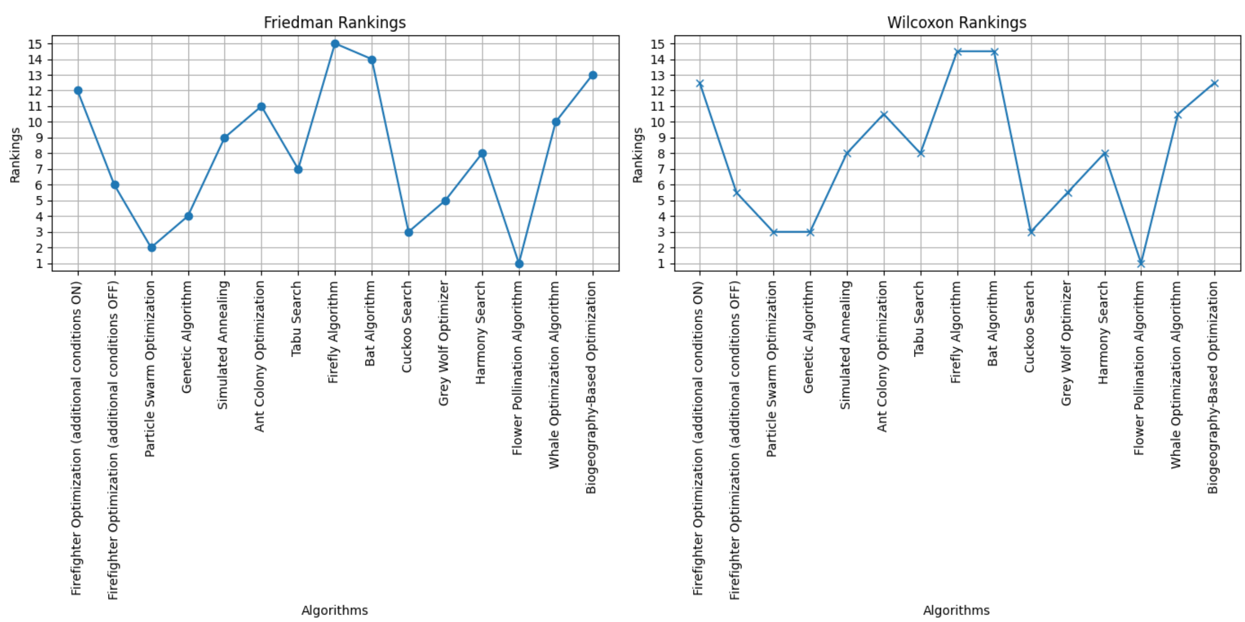


(b) At 20D

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.





(c) At 50D

Fig. 10 Results on CEC 2020 functions at 2D, 20D, and 50D

Real engineering problems

The CEC benchmark suit also contains several constraints and real engineering problems commonly used in the benchmarking analysis. As such, five additional engineering problems were examined herein. Table 6 lists these problems and their characteristics. It should be noted that the mathematical derivation for these problems is not included herein for brevity, yet it can be easily accessible from the original source of CEC 2020, as well as in [67]. The Friedman and Wilcoxon non-parametric statistical tests were carried out for each problem individually. Each problem was run with the following settings: Agents [25, 50, 100] and iterations [25, 100, 100]. Table 7 and Fig. 11 show the outcome of this analysis as well as a comparative history run for the three-bar truss design problem. Overall, it can be seen that the proposed algorithm has a decent performance, ranking within the top 5 spots in each problem (and first on the distance covered), with its best performance recorded on the three-bar truss design problem. This analysis also shows the need to improve further and tune FFO to allow it to enhance its performance on real engineering problems.

Table 6 CEC 2020 engineering problems.

No.	Functions	*No. of constraints*	*Objective/Remark*
10	Process flow sheeting problem	3	Minimize the flow sheeting process
17	Tension/Compression spring design	4	Optimize the weight of a spring
18	Pressure vessel design	4	Optimize the welding cost, material, and forming
19	Three-bar truss design problem	3	Minimize the weight of the bar structures

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

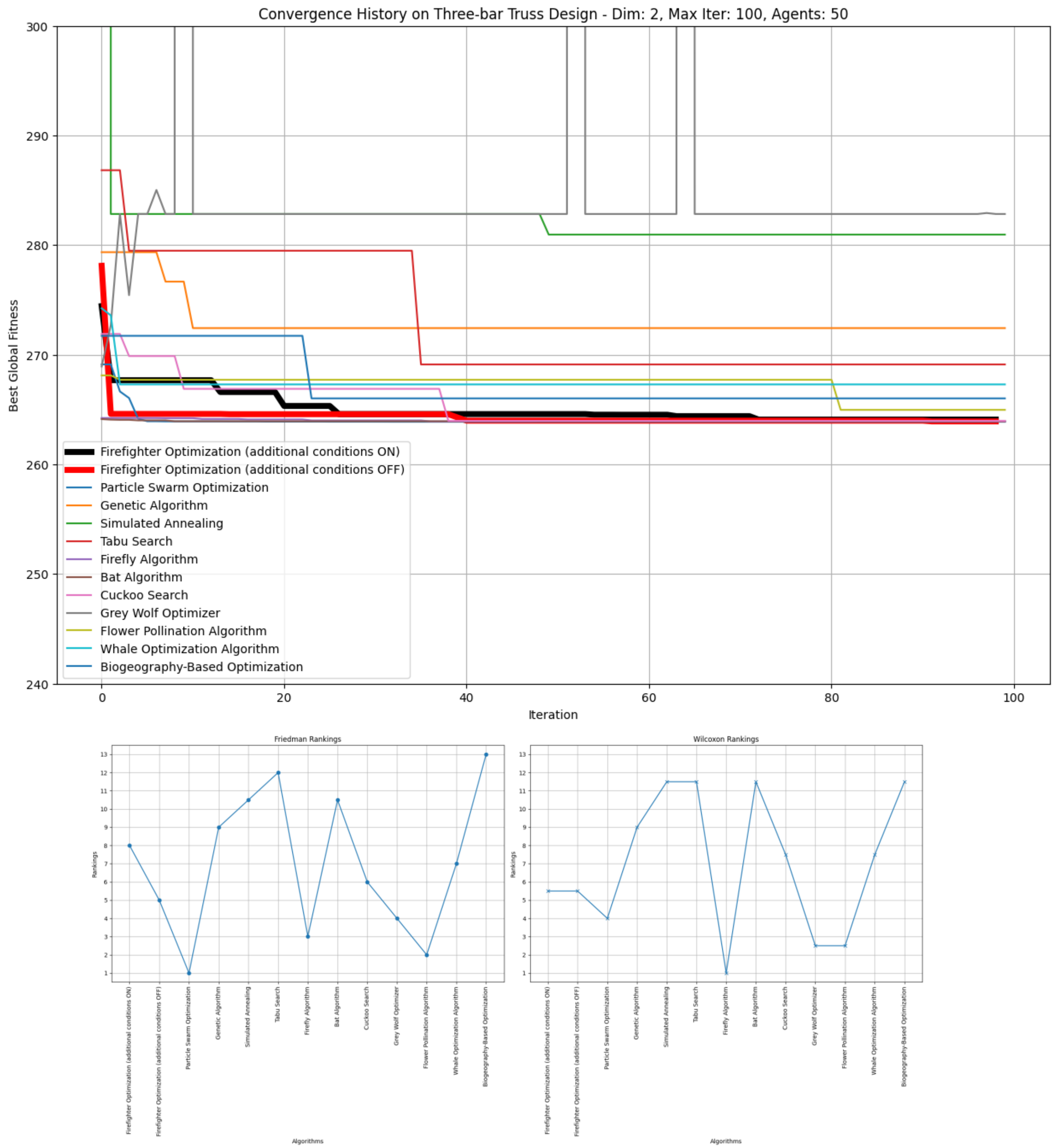
Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.

Table 7 Results on engineering problems (Ranks are based on fitness)

Tension/Compression spring design												
Algorithm	Max Iterations: 25, Agents: 25				Max Iterations: 100, Agents: 50				Max Iterations: 500, Agents: 100			
	Rank	Best Fitness_mean	Execution Time_mean	Total Distance_mean	Rank	Best Fitness_mean	Execution Time_mean	Total Distance_mean	Rank	Best Fitness_mean	Execution Time_mean	Total Distance_mean
BA	11	0.0351	6.0900	0.1800	11	0.0714	0.5600	0.0501	13	966229.5285	0.0900	0.0000
BBO	12	0.1346	12.4000	304.4317	9	0.0232	1.1500	296.3512	12	900872.9867	0.2000	0.0000
CS	5	0.0130	283.8200	0.9638	7	0.0161	13.4600	16.7597	7	0.0347	1.1400	3.8306
FFO (conditions OFF)	2	0.0127	6.5400	104755.4358	6	0.0157	0.6100	24247.5107	8	0.0348	0.0800	2769.1185
FFO (conditions ON)	8	0.0152	0.6800	22753.4404	8	0.0167	0.2900	9874.0739	6	0.0184	0.0800	2896.5719
FA	4	0.0127	592.9600	15.4207	4	0.0140	27.9600	15.5375	4	0.0150	2.0400	29.8529
FPA	3	0.0127	6.5900	5.8156	2	0.0127	0.5800	6.8138	3	0.0140	0.0800	5.9666
GA	7	0.0145	3.2400	10.7102	5	0.0157	0.3100	3.3469	10	55481.1184	0.0500	4.3979
GWO	1	0.0127	12.5700	644.1331	3	0.0131	1.1400	150.7236	2	0.0138	0.2100	61.2991
PSW	9	0.0156	3.7200	2870.0385	1	0.0127	0.3600	1362.2240	1	0.0135	0.0500	653.0028
SA	6	0.0133	0.0300	11.0837	13	20016.2994	0.0100	3.7422	11	443221.9830	0.0000	2.5749
TS	13	2763.4846	0.1000	179.8630	12	7800.8091	0.0200	29.3575	9	0.2461	0.0100	9.0047
WOA	10	0.0169	3.9800	643.2659	10	0.0270	0.3600	181.7931	5	0.0161	0.0500	47.2800
Pressure vessel design												
Algorithm	Max Iterations: 25, Agents: 25				Max Iterations: 100, Agents: 50				Max Iterations: 500, Agents: 100			
	Rank	Best Fitness_mean	Execution Time_mean	Total Distance_mean	Rank	Best Fitness_mean	Execution Time_mean	Total Distance_mean	Rank	Best Fitness_mean	Execution Time_mean	Total Distance_mean
BA	10	27672.8778	0.0700	0.0141	11	30169.8352	0.5100	0.1832	8	8101.1895	5.6300	2.4383
BBO	8	20177.3138	0.1400	651.7029	10	19536.3625	1.0900	0.0000	13	13435.2796	11.5400	16406.9804
CS	3	8588.6665	0.8700	297.5321	1	6255.4730	12.0500	380.9517	2	6055.6658	250.7500	461.4131
FFO (conditions OFF)	6	16238.2724	0.0900	79738.3119	7	13338.4993	0.5500	528538.3793	3	6058.6276	6.0300	3550676.4451
FFO (conditions ON)	7	19499.2357	0.0900	71257.3334	9	14521.7170	0.2600	266136.4509	10	10273.0019	0.8200	796115.0139
FA	11	57102.8448	1.8100	0.0000	8	13490.6739	25.3300	62.4217	12	12401.2549	541.9300	123.3595
FPA	5	15354.0109	0.0700	39.0251	2	6384.4138	0.5900	229.8373	4	6082.2291	5.8500	118.4741
GA	9	21774.6840	0.0400	14.9825	6	11447.9394	0.2900	37.1125	5	6195.9443	2.9900	21.8192
GWO	2	7770.6504	0.1300	437.6659	4	7325.6852	1.0400	1456.1620	1	6049.9952	11.2400	11543.6784
PSW	1	6093.6822	0.0500	10362.2512	3	6767.6893	0.3300	22045.4224	6	7330.6288	3.5100	35351.4385
SA	13	853026517754487000.0000	0.0000	20.7782	13	277802.5589	0.0100	85.2185	11	11584.2557	0.0300	167.9289
TS	12	197045.3629	0.0100	13.2349	12	132269.8417	0.0300	58.0393	7	7336.3953	0.1800	226.9320
WOA	4	12478.3154	0.0400	212.7318	5	10313.6550	0.3400	972.9236	9	9937.7694	3.6700	10364.2569
Three-bar truss design problem												
Algorithm	Max Iterations: 25, Agents: 25				Max Iterations: 100, Agents: 50				Max Iterations: 500, Agents: 100			
	Rank	Best Fitness_mean	Execution Time_mean	Total Distance_mean	Rank	Best Fitness_mean	Execution Time_mean	Total Distance_mean	Rank	Best Fitness_mean	Execution Time_mean	Total Distance_mean
BA	11	275.5093	0.4100	0.0113	3	263.9097	0.9600	0.0012	4	263.8950	10.1600	0.0435
BBO	2	264.1611	0.8100	0.0000	10	265.4840	1.8100	0.0000	8	264.8880	19.1400	84.4678
CS	6	268.3153	4.9800	0.6687	5	264.5199	24.0400	0.4967	6	264.1509	439.9200	0.4726
FFO (conditions OFF)	3	264.3765	0.2700	284.0285	4	264.0866	0.8900	1761.7764	2	263.8919	10.4300	8258.9518
FFO (conditions ON)	5	266.2357	0.2600	314.5034	6	264.6600	0.4200	926.2980	5	264.1359	0.8500	1930.4116
FA	1	264.0344	9.1500	0.2640	2	263.9003	53.0000	0.2236	3	263.8926	952.0100	0.0785
FPA	8	269.5027	0.5800	0.7339	7	264.6662	0.9300	0.0294	7	264.2540	10.3400	0.2196
GA	7	268.8947	0.1200	0.0072	8	264.6835	0.4600	0.0142	9	265.1644	4.8600	0.3835
GWO	12	282.8427	0.7400	0.0157	12	282.8427	1.8100	2.8466	13	282.8402	18.4800	3.3070
PSW	4	265.8990	0.1600	75.3812	1	263.8915	0.5400	172.3326	1	263.8915	5.9400	402.4285
SA	12	282.8427	0.0000	3.2932	12	282.8427	0.0100	3.3107	12	272.4788	0.0500	4.3317
TS	10	272.5299	0.0100	4.2519	12	282.8427	0.0200	20.3575	10	267.1915	0.1100	91.2797
WOA	9	270.1763	0.4000	1.0332	9	264.7512	0.6200	3.1695	11	267.9062	5.9900	2.2217
Process flow sheeting problem												
Algorithm	Max Iterations: 25, Agents: 25				Max Iterations: 100, Agents: 50				Max Iterations: 500, Agents: 100			
	Rank	Best Fitness_mean	Execution Time_mean	Total Distance_mean	Rank	Best Fitness_mean	Execution Time_mean	Total Distance_mean	Rank	Best Fitness_mean	Execution Time_mean	Total Distance_mean
BA	12	69813.0780	0.3500	0.0000	12	2711.2231	0.7600	0.2014	5	1.2507	6.8500	0.5123
BBO	13	158420.4126	1.3600	8.6180	13	10149.9007	1.4100	22.3039	13	14285.1378	15.5600	0.0000
CS	6	16.4045	5.7400	0.0989	5	1.3677	16.5000	0.8389	6	1.2510	325.9700	0.4688
FFO (conditions OFF)	8	667.6444	0.3500	744.3846	7	1.4775	0.7100	4498.6072	2	1.2500	8.1900	21269.7306
FFO (conditions ON)	9	1542.7608	0.2700	701.7935	2	1.2500	0.5400	3816.3316	10	2.3309	1.0200	7224.1551
FA	3	1.2526	8.8700	0.1592	3	1.2501	37.6700	0.2927	3	1.2500	716.7600	1.1939
FPA	4	8.7331	0.2300	1.4028	6	1.4384	0.6400	0.9795	9	1.2690	8.2700	0.6424
GA	7	56.1340	0.0900	1.3388	8	8.7747	0.3700	0.3217	7	1.2573	4.7700	0.6688
GWO	2	1.2509	0.8400	0.4131	4	1.2504	1.2700	1.2805	4	1.2500	14.3800	4.8077
PSW	1	1.2500	0.1300	114.8741	1	1.2500	0.3900	177.3422	1	1.2500	4.0700	378.2226
SA	11	8200.0204	0.0000	0.3118	10	26.0589	0.0100	3.1196	11	248.9490	0.0400	3.4190
TS	10	5914.7086	0.0100	1.9175	9	18.0536	0.0200	13.5297	8	1.2597	0.1500	48.5607
WOA	5	14.0063	0.1200	6.2861	11	1880.2030	0.5200	6.9295	12	1245.9236	4.6700	25.7932

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

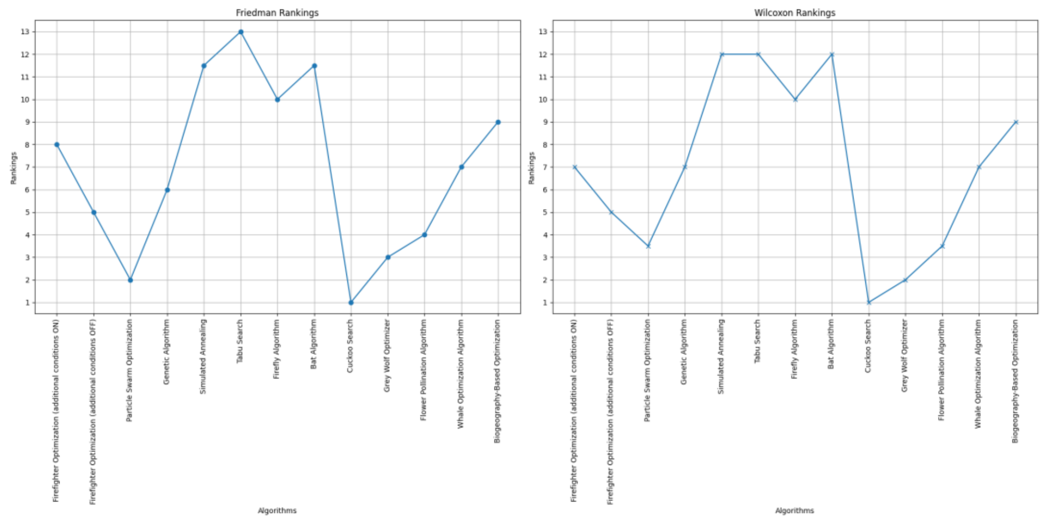
Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.



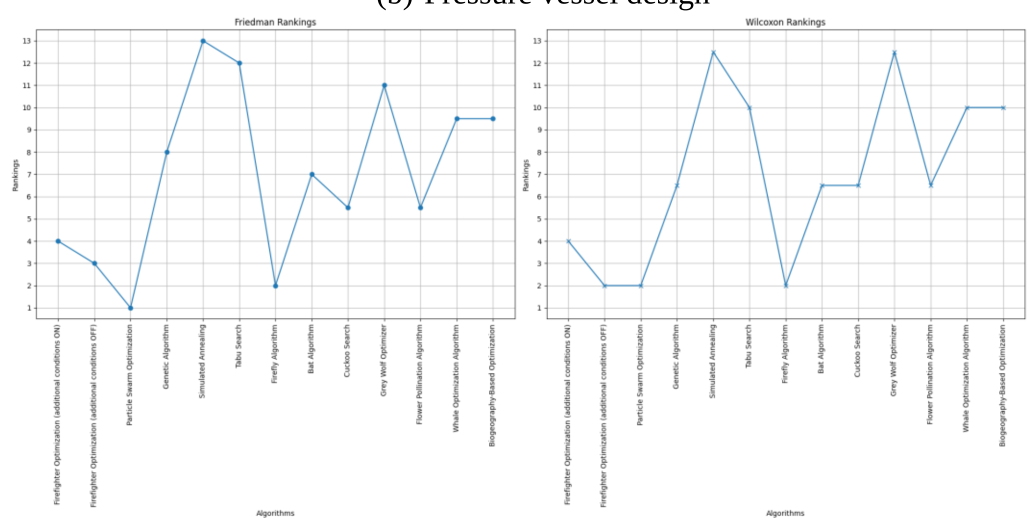
(a) Tension/Compression spring design

This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

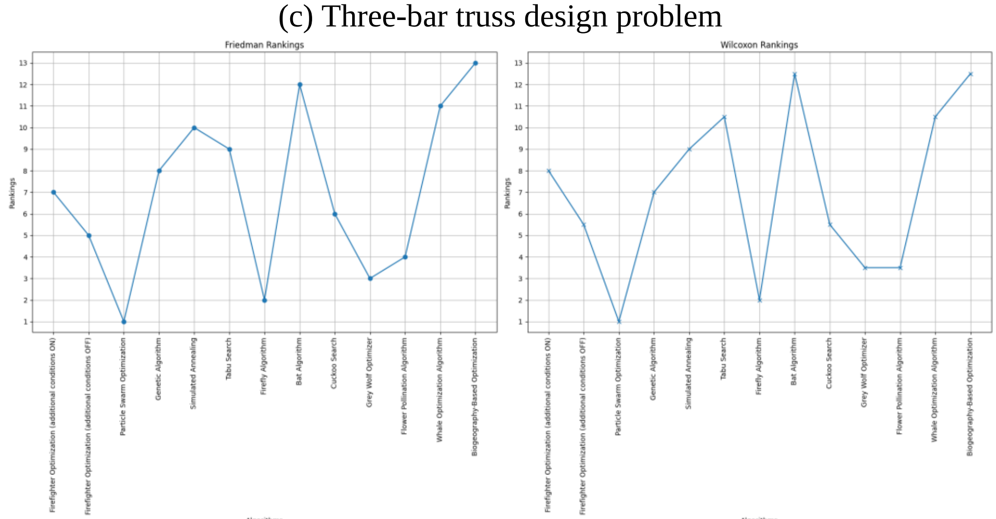
Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.



(b) Pressure vessel design



(c) Three-bar truss design problem



(d) Process flow sheeting problem

Fig. 11 Rankings on real engineering problems

Future research needs

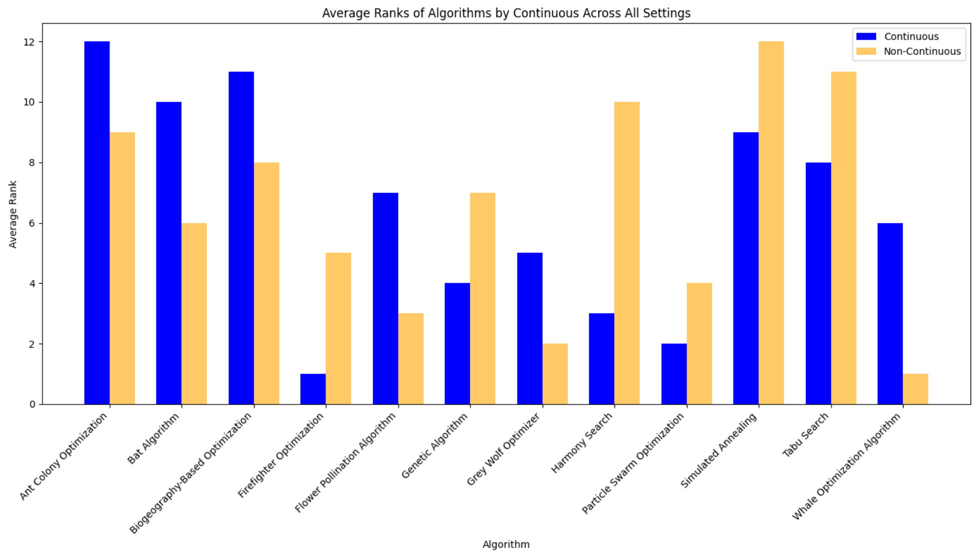
In order to further examine the performance of the proposed algorithm, we present a new set of results (see Fig. 12). In this analysis, we focus on function attributes like continuity, separability, multimodality, differentiability, and performance in continuous function optimization are demonstrated. The plots assess performance based on the average of metrics and visually display how each algorithm performs across these attributes under the specific settings.

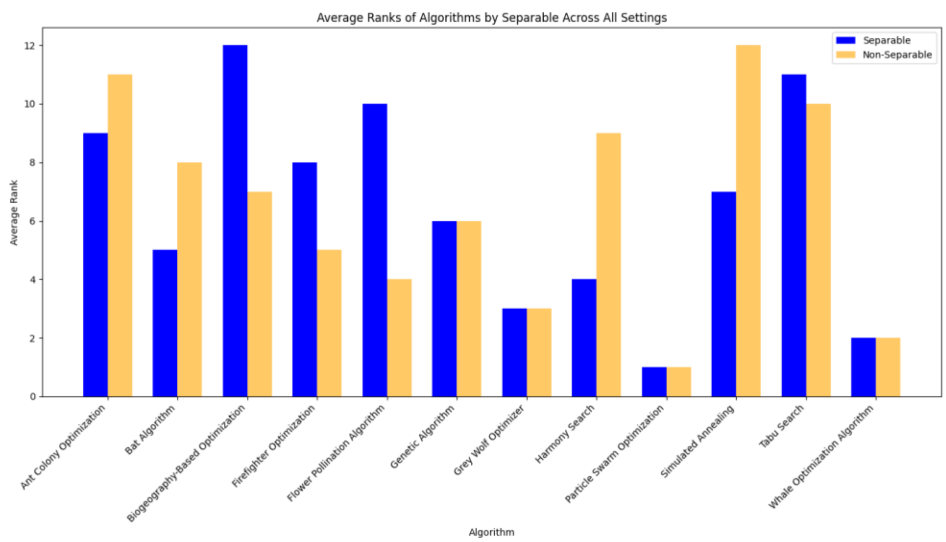
This figure shows results produced by filtering for specific settings (Iterations=1000, Agents=100) and merging these results with predefined function attributes. These plots reveal distinct performance characteristics that generally show more modest peaks across the attributes, suggesting that some algorithms perform better in handling specific functions (i.e., Separable functions). For example, the FFO algorithm ranks first and fifth in continuous functions, first and fourth in differentiable and non-differentiable, second and sixth on multimodal and non-multimodal functions, fifth in scalable and non-scalable functions, and eighth and fifth on separable and non-separable functions.

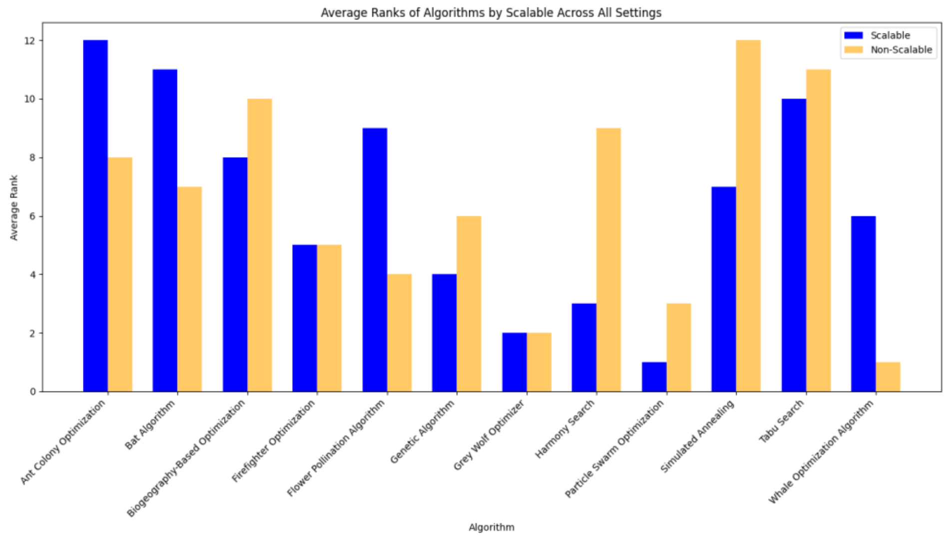
These insights, together with those gained from the testing on real engineering problems, can also provide technical improvement areas for FFO (as well as the selected algorithms). For example, FFO could focus on enhancing its performance in non-separable and multimodal landscapes, perhaps by integrating more adaptive search strategies or improving its handling of function differentiability through better derivative estimation or step size adjustment mechanisms. Additionally, efforts to boost scalability could be crucial, especially for handling higher-dimensional optimization problems more effectively. In addition, there is a need to further examine the proposed algorithms against some of the recently developed algorithms (namely, SASS, COLSHADE, sCMAGES). At the moment, the performance of FFO can indeed be improved to match it with the aim of further enhancing it against the aforementioned leading algorithms. In addition, the authors hope that future editions from FFO include multi- and many-objective optimization capabilities.

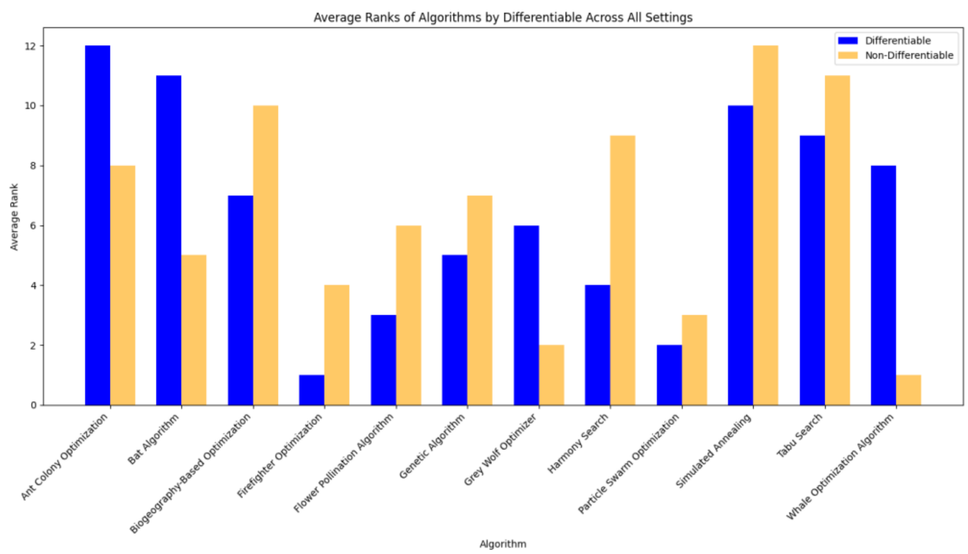
This is a preprint draft. The published article can be found at: https://doi.org/10.1007/s00521-025-11074-z.

Please cite this paper as:
Naser M.Z., Naser, A. (2025). The firefighter algorithm for optimization problems. *Neural Computing and Applications*. https://doi.org/10.1007/s00521-025-11074-z.






Average Ranks of Algorithms by Scalable Across All Settings
Scalable
Non-Scalable
Average Rank
Algorithm
Ant Colony Optimization
Bat Algorithm
Biogeography-Based Optimization
Firefighter Optimization
Flower Pollination Algorithm
Genetic Algorithm
Grey Wolf Optimizer
Harmony Search
Particle Swarm Optimization
Simulated Annealing
Tabu Search
Whale Optimization Algorithm


Average Ranks of Algorithms by Differentiable Across All Settings
Differentiable
Non-Differentiable
Average Rank
Algorithm
Ant Colony Optimization
Bat Algorithm
Biogeography-Based Optimization
Firefighter Optimization
Flower Pollination Algorithm
Genetic Algorithm
Grey Wolf Optimizer
Harmony Search
Particle Swarm Optimization
Simulated Annealing
Tabu Search
Whale Optimization Algorithm

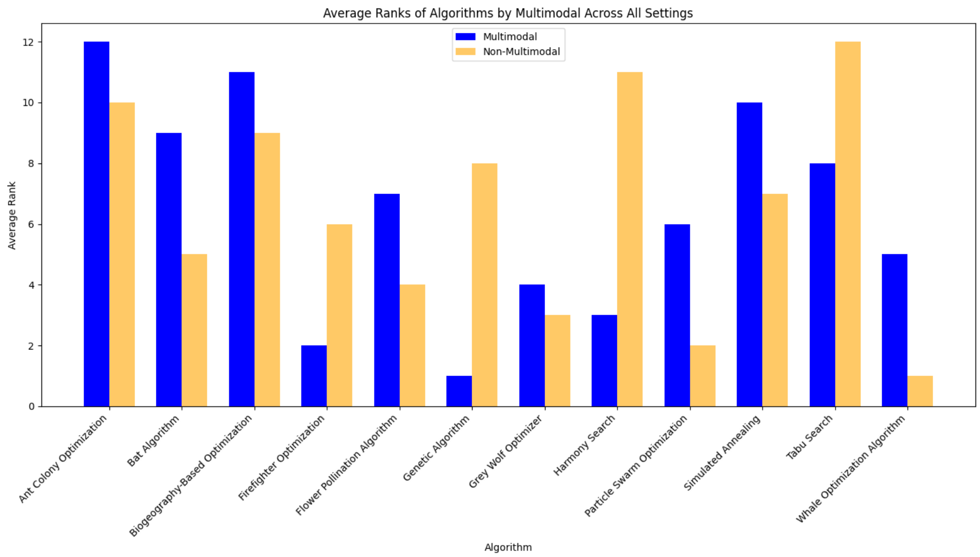


Fig. 11 Comparison of algorithms for function categories

6.0 Conclusions

The Firefighter optimization (FFO) algorithm offers a metaheuristic approach for optimization. This algorithm was examined against 13 commonly used optimization algorithms and 38 functions (including benchmark functions, CEC 2020 standard functions, and real engineering problems). Our results demonstrate that in 2D analysis, FFO ranks in the top 3 slots in terms of the best fitness and space covered and ranks first in the Distance per Unit Time metric. Similarly, the performance of the FFO algorithm also ranks third in higher dimensions (20D and 50D) and maintains a top 5 performance across various metrics. Our analysis also indicates a few possible means to improve the performance of FFO, primarily on scalable/non-scalable functions and separable/non-separable functions, as well as on performing on real optimization problems.

Data Availability

Some or all data, models, or code that support the findings of this study are available from the corresponding author upon reasonable request.

Firefighter optimization (FFO) can be accessed from [**to be added**].

Conflict of Interest

The authors declare no conflict of interest.

References

[1] S.S. Rao, Engineering optimization: Theory and practice, 2019. https://doi.org/10.1002/9781119454816.

[2] J. Silberholz, B. Golden, Comparison of Metaheuristics, in: 2010. https://doi.org/10.1007/978-1-4419-1665-5_21.

[3] A. Alorf, A survey of recently developed metaheuristics and their comparative analysis, Eng. Appl. Artif. Intell. (2023). https://doi.org/10.1016/j.engappai.2022.105622.

[4] R.M. Lewis, V. Torczon, M.W. Trosset, Direct search methods: Then and now, J. Comput. Appl. Math. 124 (2000) 191–207. https://doi.org/10.1016/S0377-0427(00)00423-4.

[5] T. Guilmeau, E. Chouzenoux, V. Elvira, Simulated Annealing: A Review and a New Scheme, in: IEEE Work. Stat. Signal Process. Proc., 2021. https://doi.org/10.1109/SSP49050.2021.9513782.

[6] X.C. Pardo, P. González, J.R. Banga, R. Doallo, Population based metaheuristics in Spark: Towards a general framework using PSO as a case study, Swarm Evol. Comput. (2024). https://doi.org/10.1016/j.swevo.2024.101483.

[7] D. Caicedo, L. Lara-Valencia, Y. Valencia, Machine Learning Techniques and Population-Based Metaheuristics for Damage Detection and Localization Through Frequency and Modal-Based Structural Health Monitoring: A Review, Arch. Comput. Methods Eng. (2022). https://doi.org/10.1007/s11831-021-09692-6.

[8] A. Bavar, A. Bavar, F. Gholian-Jouybari, M. Hajiaghaei-Keshteli, C. Mejía-Argueta, Developing new heuristics and hybrid meta-heuristics to address the bi-objective home health care problem, Cent. Eur. J. Oper. Res. (2023). https://doi.org/10.1007/s10100-023-00862-4.

[9] X. Guo, J. Hu, H. Yu, M. Wang, B. Yang, A new population initialization of metaheuristic algorithms based on hybrid fuzzy rough set for high-dimensional gene data feature selection, Comput. Biol. Med. (2023). https://doi.org/10.1016/j.compbiomed.2023.107538.

[10] A. Gogna, A. Tayal, Metaheuristics: Review and application, J. Exp. Theor. Artif. Intell. (2013). https://doi.org/10.1080/0952813X.2013.782347.

[11] K. Rajwar, K. Deep, S. Das, An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges, Artif. Intell. Rev. (2023). https://doi.org/10.1007/s10462-023-10470-y.

[12] R. Sala, R. Müller, Benchmarking for Metaheuristic Black-Box Optimization: Perspectives and Open Challenges. (arXiv:2007.00541v1 [cs.NE]), ArXiv Comput. Sci. (2020).

[13] K.-L. Du, M.N.S. Swamy, Search and Optimization by Metaheuristics, 2016. https://doi.org/10.1007/978-3-319-41192-7.

[14] A. Beşkirli, İ. Dağ, M.S. Kıran, A tree seed algorithm with multi-strategy for parameter estimation of solar photovoltaic models, Appl. Soft Comput. 167 (2024) 112220. https://doi.org/10.1016/J.ASOC.2024.112220.

[15] A. Beşkirli, İ. Dağ, I-CPA: An Improved Carnivorous Plant Algorithm for Solar Photovoltaic Parameter Identification Problem, Biomimetics. (2023). https://doi.org/10.3390/biomimetics8080569.

[16] Q. Li, X. Zeng, W. Wei, Multi-objective particle swarm optimization algorithm using Cauchy mutation and improved crowding distance, Int. J. Intell. Comput. Cybern. (2023). https://doi.org/10.1108/IJICC-04-2022-0118.

[17] F.S. Gharehchopogh, Advances in Tree Seed Algorithm: A Comprehensive Survey, Arch. Comput. Methods Eng. (2022). https://doi.org/10.1007/s11831-021-09698-0.

[18] C. Qu, W. He, X. Peng, X. Peng, Harris Hawks optimization with information exchange, Appl. Math. Model. (2020). https://doi.org/10.1016/j.apm.2020.03.024.

[19] L. Abualigah, A. Diabat, S. Mirjalili, M. Abd Elaziz, A.H. Gandomi, The Arithmetic Optimization Algorithm, Comput. Methods Appl. Mech. Eng. (2021). https://doi.org/10.1016/j.cma.2020.113609.

[20] M. Shehab, I. Mashal, Z. Momani, M.K.Y. Shambour, A. AL-Badareen, S. Al-Dabet, N. Bataina, A.R. Alsoud, L. Abualigah, Harris Hawks Optimization Algorithm: Variants and Applications, Arch. Comput. Methods Eng. (2022). https://doi.org/10.1007/s11831-022-09780-1.

[21] M. Dorigo, M. Birattari, T. Stützle, Ant colony optimization artificial ants as a computational intelligence technique, IEEE Comput. Intell. Mag. (2006). https://doi.org/10.1109/CI-M.2006.248054.

[22] J.E. Bell, P.R. McMullen, Ant colony optimization techniques for the vehicle routing problem, Adv. Eng. Informatics. (2004). https://doi.org/10.1016/j.aei.2004.07.001.

[23] X.S. Yang, A.H. Gandomi, Bat algorithm: A novel approach for global engineering optimization, Eng. Comput. (Swansea, Wales). (2012). https://doi.org/10.1108/02644401211235834.

[24] P.W. Tsai, J.S. Pan, B.Y. Liao, M.J. Tsai, V. Istanda, Bat algorithm inspired algorithm for solving numerical optimization problems, in: Appl. Mech. Mater., 2012. https://doi.org/10.4028/www.scientific.net/AMM.148-149.134.

[25] D. Simon, Biogeography-based optimization, IEEE Trans. Evol. Comput. (2008). https://doi.org/10.1109/TEVC.2008.919004.

[26] R.A. Gupta, R. Kumar, A.K. Bansal, BBO-based small autonomous hybrid power system optimization incorporating wind speed and solar radiation forecasting, Renew. Sustain. Energy Rev. (2015). https://doi.org/10.1016/j.rser.2014.09.017.

[27] X.S. Yang, S. Deb, Cuckoo search via Lévy flights, in: 2009 World Congr. Nat. Biol. Inspired Comput. NABIC 2009 - Proc., 2009. https://doi.org/10.1109/NABIC.2009.5393690.

[28] A.H. Gandomi, X.S. Yang, A.H. Alavi, Cuckoo search algorithm: A metaheuristic approach to solve structural optimization problems, Eng. Comput. (2013). https://doi.org/10.1007/s00366-011-0241-y.

[29] X.-S. Yang, A. Slowik, Firefly Algorithm, Swarm Intell. Algorithms. (2020) 163–174. https://doi.org/10.1201/9780429422614-13.

[30] H. Xie, L. Zhang, C.P. Lim, Y. Yu, C. Liu, H. Liu, J. Walters, Improving K-means clustering with enhanced Firefly Algorithms, Appl. Soft Comput. J. (2019). https://doi.org/10.1016/j.asoc.2019.105763.

[31] X.S. Yang, M. Karamanoglu, X. He, Flower pollination algorithm: A novel approach for multiobjective optimization, Eng. Optim. (2014). https://doi.org/10.1080/0305215X.2013.832237.

[32] S. Lalljith, I. Fleming, U. Pillay, K. Naicker, Z.J. Naidoo, A.K. Saha, Applications of Flower Pollination Algorithm in Electrical Power Systems: A Review, IEEE Access. (2022). https://doi.org/10.1109/ACCESS.2021.3138518.

[33] J.H. Holland, Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence, 1975.

[34] S. Ansari, K.A. Alnajjar, M. Saad, S. Abdallah, A.A. El-Moursy, Automatic Digital Modulation Recognition Based on Genetic-Algorithm-Optimized Machine Learning Models, IEEE Access. (2022). https://doi.org/10.1109/ACCESS.2022.3171909.

[35] S. Mirjalili, S.M. Mirjalili, A. Lewis, Grey Wolf Optimizer, Adv. Eng. Softw. (2014). https://doi.org/10.1016/j.advengsoft.2013.12.007.

[36] Z.W. Geem, J.H. Kim, G. V. Loganathan, A New Heuristic Optimization Algorithm: Harmony Search, Simulation. (2001). https://doi.org/10.1177/003754970107600201.

[37] J. Kennedy, R. Eberhart, Particle swarm optimization, Proc. ICNN'95 - Int. Conf. Neural Networks. 4 (n.d.) 1942–1948. https://doi.org/10.1109/ICNN.1995.488968.

[38] F. Marini, B. Walczak, Particle swarm optimization (PSO). A tutorial, Chemom. Intell. Lab. Syst. (2015). https://doi.org/10.1016/j.chemolab.2015.08.020.

[39] P. Siarry, Simulated annealing, in: Metaheuristics, 2016. https://doi.org/10.1007/978-3-319-45403-0_2.

[40] W.L. Goffe, G.D. Ferrier, J. Rogers, Global optimization of statistical functions with simulated annealing, J. Econom. (1994). https://doi.org/10.1016/0304-4076(94)90038-8.

[41] F. Glover, Tabu Search—Part I, ORSA J. Comput. (1989). https://doi.org/10.1287/ijoc.1.3.190.

[42] A. Alfieri, C. Castiglione, E. Pastore, A multi-objective tabu search algorithm for product portfolio selection: A case study in the automotive industry, Comput. Ind. Eng. (2020). https://doi.org/10.1016/j.cie.2020.106382.

[43] S. Mirjalili, A. Lewis, The Whale Optimization Algorithm, Adv. Eng. Softw. (2016). https://doi.org/10.1016/j.advengsoft.2016.01.008.

[44] N. Rana, M.S.A. Latiff, S.M. Abdulhamid, H. Chiroma, Whale optimization algorithm: a systematic review of contemporary applications, modifications and developments, Neural Comput. Appl. (2020). https://doi.org/10.1007/s00521-020-04849-z.

[45] C. Wei, Y. Li, Y. Yu, Solution of ackley function based on particle swarm optimization algorithm, in: Proc. 2020 IEEE Int. Conf. Adv. Electr. Eng. Comput. Appl. AEECA 2020, 2020. https://doi.org/10.1109/AEECA49918.2020.9213634.

[46] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, A novel population initialization method for accelerating evolutionary algorithms, Comput. Math. with Appl. (2007). https://doi.org/10.1016/j.camwa.2006.07.013.

[47] P.R.D. Marinho, R.B. Silva, M. Bourguignon, G.M. Cordeiro, S. Nadarajah, AdequacyModel: An R package for probability distributions and general purpose optimization, PLoS One. (2019). https://doi.org/10.1371/journal.pone.0221487.

[48] E.J. Solteiro Pires, J.A. Tenreiro MacHado, P.B. De Moura Oliveira, J. Boaventura Cunha, L. Mendes, Particle swarm optimization with fractional-order velocity, Nonlinear Dyn. (2010). https://doi.org/10.1007/s11071-009-9649-y.

[49] M. Molga, C. Smutnicki, Test functions for optimization needs, Test Funct. Optim. Needs. (2005).

[50] J.M. Czerniak, H. Zarzycki, Artificial Acari Optimization as a new strategy for global optimization of multimodal functions, J. Comput. Sci. (2017). https://doi.org/10.1016/j.jocs.2017.05.028.

[51] D. Whitley, S. Rana, J. Dzubera, K.E. Mathias, Evaluating evolutionary algorithms, Artif. Intell. (1996). https://doi.org/10.1016/0004-3702(95)00124-7.

[52] A.A. Goldstein, J.F. Price, On descent from local minima, Math. Comput. (1971). https://doi.org/10.1090/s0025-5718-1971-0312365-x.

[53] A.O. Griewank, Generalized descent for global optimization, J. Optim. Theory Appl. (1981). https://doi.org/10.1007/BF00933356.

[54] D.M. Himmelblau, Applied Nonlinear Programming, McGraw-Hill, 1972.

[55] S.K. Mishra, Global Optimization by Differential Evolution and Particle Swarm Methods: Evaluation on Some Benchmark Functions, SSRN Electron. J. (2011). https://doi.org/10.2139/ssrn.933827.

[56] J. Matyas, Random Optimization, Autom. i Telemekh. (1965).

[57] Z. Michalewicz, A Survey of Constraint Handling Techniques in Evolutionary Computation Methods, in: Evol. Program. IV, 2020. https://doi.org/10.7551/mitpress/2887.003.0018.

[58] A. Omeradzic, H.G. Beyer, Convergence Properties of the (μ/μI, λ)-ES on the Rastrigin Function, in: FOGA 2023 - Proc. 17th ACM/SIGEVO Conf. Found. Genet. Algorithms, 2023. https://doi.org/10.1145/3594805.3607126.

[59] Y.-W.W. Shang, Y.-H.H. Qiu, A Note on the Extended Rosenbrock Function, Evol. Comput. 14 (2006) 119–126. https://doi.org/10.1162/106365606776022733.

[60] P.C. Chou, J.L. Chen, Enforced mutation to enhancing the capability of particle swarm optimization algorithms, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), 2011. https://doi.org/10.1007/978-3-642-21515-5_4.

[61] A. Tripathy, H.-P. Schwefel, Numerical Optimization of Computer Models, J. Oper. Res. Soc. (1982). https://doi.org/10.2307/2581158.

[62] M.A. Styblinski, T.S. Tang, Experiments in nonconvex optimization: Stochastic approximation with function smoothing and simulated annealing, Neural Networks. (1990). https://doi.org/10.1016/0893-6080(90)90029-K.

[63] T. Malik, E.H. Winer, An analytical curve based approach for multi-modal optimization, in: 9th AIAA/ISSMO Symp. Multidiscip. Anal. Optim., 2002. https://doi.org/10.2514/6.2002-5520.

[64] J. Carrasco, S. García, M.M. Rueda, S. Das, F. Herrera, Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: Practical guidelines and a critical review, Swarm Evol. Comput. (2020). https://doi.org/10.1016/j.swevo.2020.100665.

[65] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm Evol. Comput. (2011). https://doi.org/10.1016/j.swevo.2011.02.002.

[66] C.T. Yue, Problem Defnitions and Evaluation Criteria for the CEC 2020 Special Session and Competition on Single Objective Bound Constrained Numerical Optimization, 2020.

[67] CEC2020-Algorithms/test_functions/cec2020_constrained.py at main · strzecha/CEC2020-Algorithms · GitHub, (n.d.). https://github.com/strzecha/CEC2020-Algorithms/blob/main/test_functions/cec2020_constrained.py#L390 (accessed August 23, 2024).