Ocotillo Optimization Algorithm (OcOA): A Desert-Inspired Metaheuristic for Adaptive Optimization

El-Sayed M. El-Kenawy [1,2,3,4] *, Faris H. Rizk[5], Ahmed Mohamed Zaki[5], Mahmoud Elshabrawy Mohamed[5], Abdelhameed Ibrahim[1], Abdelaziz A. Abdelhamid[6,7], Nima Khodadadi[8], Ehab M. Almetwally[9], Marwa M. Eid[10]

[1]School of ICT, Faculty of Engineering, Design and Information & Communications Technology (EDICT), Bahrain Polytechnic, PO Box 33349, Isa Town, Bahrain.
[2]Jadara University Research Center, Jadara University, Jordan.
[3]Applied Science Research Center. Applied Science Private University, Amman, Jordan.
[4]Department of Communications and Electronics, Delta Higher Institute of Engineering and Technology, Mansoura 35111, Egypt.
[5]Computer Science and Intelligent Systems Research Center, Blacksburg 24060, Virginia, USA
[6]Department of Computer Science, Faculty of Computer and Information Sciences, Ain Shams University, Cairo 11566, Egypt
[7]Department of Computer Science, College of Computing and Information Technology, Shaqra University, 11961, Shaqra, Saudi Arabia
[8]Department of Civil and Architectural Engineering, University of Miami, Coral Gables, FL, USA
[9]Department of Mathematics and Statistics, Faculty of Science, Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh 11432, Saudi Arabia
[10]Faculty of Artificial Intelligence, Delta University for Science and Technology, Mansoura 11152, Egypt

Emails: skenawy@ieee.org, faris.rizk@jcsis.org, Azaki@jcsis.org, mshabrawy@jcsis.org, afai79@mans.edu.eg, abdelaziz@cis.asu.edu.eg, nima.khodadadi@miami.edu, emalmetwally@imamu.edu.sa, mmm@ieee.org

Abstract

In this paper, we propose the Ocotillo Optimization Algorithm (OcOA), a novel desert-inspired metaheuristic designed to solve complex optimization problems. Inspired by the adaptive strategies of desert plants, OcOA aims to achieve a balance between exploration and exploitation in high-dimensional and multimodal search spaces. The algorithm dynamically adjusts its behavior based on feedback from prior iterations, optimizing both search breadth and solution refinement. To evaluate its effectiveness, OcOA was tested against several well-known algorithms on a range of benchmark functions, including unimodal and multimodal functions from the CEC 2005 suite such as Sphere, Rosenbrock, Ackley, and Rastrigin. The results demonstrate that OcOA outperforms competing approaches in terms of accuracy, convergence speed, and computational efficiency. Additionally, its adaptability was validated through feature selection tasks, highlighting its robustness in handling both continuous and discrete optimization challenges. This study positions OcOA as a competitive optimization tool for various real-world applications.

Keywords: Ocotillo Optimization Algorithm, metaheuristic, exploration-exploitation balance, complex optimization, adaptive algorithm

1 introduction

Metaheuristic optimization algorithms have become indispensable tools for solving complex and large-scale optimization problems that arise across a wide range of disciplines, including engineering, artificial intelligence, economics, logistics, and many others. These problems are often characterized by complex search spaces that are multimodal, nonlinear, and high-dimensional, making it difficult for traditional optimization techniques to find optimal solutions effectively. Metaheuristics provide a flexible and adaptable approach to searching these vast solution spaces, offering advantages over classical methods like gradient-based or deterministic approaches, which can struggle with issues such as local optima, non-convexity, or the need for differentiable functions [1–3].

Optimization problems, at their core, involve finding the best solution from a set of possible solutions while adhering to specific constraints. Many of these problems involve multiple objectives, where a trade-off must be made between conflicting goals. Additionally, real-world optimization problems are often subject to noise, uncertainty, and dynamic changes, which further complicate the search for optimal solutions. In such scenarios, classical optimization techniques like linear programming, branch and bound, or gradient descent methods can be inefficient or even fail to provide feasible solutions. This is where metaheuristic algorithms, which do not rely on gradient information and can explore complex search landscapes, excel [4–6].

Metaheuristic algorithms are typically inspired by natural processes, biological behaviors, or social dynamics, and they offer a balance between exploration and exploitation. Exploration refers to the algorithm's ability to search widely across the solution space to avoid local optima, while exploitation focuses on intensively refining solutions in promising areas of the search space. The challenge for any metaheuristic is to maintain an optimal balance between these two processes, ensuring that the algorithm does not get stuck in local optima due to over-exploitation, or waste resources on unnecessary exploration. Successful algorithms can dynamically adjust their behavior to adapt to the problem landscape, leading to better performance across a wide range of optimization tasks [7–9].

Over the past few decades, various metaheuristic algorithms have been developed and applied successfully in different domains. Popular examples include Genetic Algorithms (GA), which simulate the process of natural selection; Particle Swarm Optimization (PSO), which mimics the social behavior of birds or fish swarms; and Grey Wolf Optimizer (GWO), inspired by the hunting strategies of grey wolves. These algorithms, and others like them, have been widely used in solving optimization problems where classical methods fall short. Despite their success, existing metaheuristic algorithms often face challenges, particularly when applied to complex or high-dimensional problems. These challenges include issues such as premature convergence, computational inefficiencies, and difficulties in maintaining a good trade-off between exploration and exploitation [10–12].

Premature convergence is a common issue where the algorithm converges to a suboptimal solution too early in the search process, without fully exploring the search space. This problem occurs when the algorithm over-focuses on exploitation at the expense of exploration, leading to solutions that are trapped in local optima. On the other hand, excessive exploration can cause an algorithm to spend too much time searching vast areas of the solution space without refining the solutions it has already found, leading to inefficiencies and slow convergence. Achieving the right balance between these two processes is crucial for the effectiveness of any metaheuristic algorithm, especially in dynamic and high-dimensional problems where the search space can be highly deceptive [13–15].

Additionally, many optimization problems, particularly in real-world applications, involve multimodal landscapes, where multiple local optima exist. Metaheuristic algorithms must be able to navigate these complex landscapes effectively, ensuring that they do not get stuck in local optima but instead continue to search for the global optimum. This requires the algorithm to be both robust and adaptable, able to shift between different strategies as needed during the search process. Furthermore, modern optimization problems often involve dynamic environments, where the objective function or constraints change over time. In such cases, the algorithm must be capable of adapting its search behavior dynamically to account for these changes.

To address these challenges, the development of new metaheuristic algorithms is an ongoing area of research. Each new algorithm aims to improve upon existing techniques by providing better exploration-exploitation trade-offs, faster convergence rates, and enhanced robustness in solving complex, multimodal, and dynamic optimization problems. The design of an effective metaheuristic algorithm involves creating mechanisms that allow for a dynamic search process, one that can adapt to the nature of the problem space while maintaining computational efficiency.

In this paper, we introduce a novel metaheuristic optimization algorithm aimed at addressing these challenges. The algorithm is designed to achieve a more balanced and adaptive exploration-exploitation process, preventing premature convergence while ensuring efficient search across high-dimensional and multimodal spaces.

We evaluate the performance of the proposed algorithm on a set of standard benchmark functions, comparing its results with several well-known optimization algorithms. Additionally, we explore the application of the algorithm to feature selection tasks, demonstrating its versatility in handling both continuous and discrete optimization problems.

The remainder of this paper is organized as follows: The next section provides an overview of the existing literature on metaheuristic algorithms, highlighting the common challenges and gaps that motivate the development of new algorithms. Following this, we present the detailed structure and mathematical formulation of the proposed algorithm, explaining how it is designed to address these challenges. We then demonstrate the algorithm's performance through rigorous experimentation on standard benchmark functions, comparing its effectiveness to other algorithms in terms of convergence, accuracy, and computational efficiency. Finally, we discuss the application of the algorithm to feature selection tasks, a critical area in machine learning, and conclude with a discussion on future research directions.

2 Literature Review

Optimization algorithm development has become a fundamental strategy to enhance computational capabilities across multiple domains such as operations research, machine learning, and engineering design. As problems grow increasingly complex, the need for more robust, faster, and flexible optimization algorithms persists. The primary aim of this literature review is to chronicle the progress over time and to categorize and compare optimization algorithms ranging from classical approaches to meta-heuristic, gradient-based, and machine learning-incorporated methods. By selectively reviewing previous work and recent developments in existing optimizers, we aim to highlight their strengths and weaknesses before introducing our new optimization approach.

In recent years, there has been an intensive focus on developing new meta-heuristic algorithms due to the increasing complexity of real-world optimization problems. The paper [16] introduces the Mountain Gazelle Optimizer (MGO), a novel approach inspired by the social and herd dynamics of mountain gazelles. By incorporating hierarchy and social relationships observed in gazelle groups, the algorithm optimizes its function. The performance of MGO has been thoroughly tested using 52 standard benchmark functions and seven engineering problems. To verify its effectiveness, MGO was compared with nine similar algorithms. Results from the Wilcoxon rank-sum and Friedman tests suggest that MGO outperforms competing algorithms on the majority of benchmark functions. Moreover, the algorithm demonstrates robustness in providing proper search direction and maintaining performance stability as the dimensionality of optimization problems increases, indicating its adeptness at handling complex, higher-dimensional problems.

In [17], the authors offer a detailed and comprehensive understanding of the Multi-Verse Optimizer (MVO) algorithm, covering its fundamental aspects, working mechanisms, and a plethora of applications. MVO is a novel and powerful nature-inspired meta-heuristic algorithm that has shown impressive performance in handling a spectrum of linear optimization challenges across different areas. The review presents the wide scope of MVO applications in various fields, including benchmark test functions, machine learning, engineering, network optimization, and parameter control. It summarizes all known works devoted to the applications of MVO, its modifications, and derivatives, such as binary, modified, hybrid, chaotic, and multi-objective MVOs. By providing a systematic analysis of these variations and their uses, along with practical suggestions and future research directions, this paper serves as a valuable reference for researchers aiming to explore and improve the existing Multi-Verse Optimizer.

[18] provides a concise description of the Ant Lion Optimizer (ALO), a meta-heuristic, swarm-based algorithm developed by Mirjalili in 2015, inspired by the natural feeding behavior of ant lions. The review focuses on successful applications of the ALO algorithm across different optimization issues. Optimality in the ALO framework involves identifying optimal solutions by adjusting, minimizing, or maximizing goal and objective functions to enhance the algorithm's performance and efficiency. As a member of the meta-heuristic algorithm family, ALO has attracted significant attention due to its support for decision-making and its ability to solve optimization problems effectively. The paper also reviews various extensions of ALO, including binary forms, modifications, hybrids, and extended forms, enhancing its proficiency in multiple scenarios. The review highlights ALO's flexibility and efficiency across several domains, such as benchmark functions, machine learning,

network optimization, engineering problems, software engineering, and image processing. By summarizing existing research, the paper aims to provide a better understanding of ALO's capacity for optimization in future studies.

It is observed that every meta-heuristic optimization algorithm requires some form of initialization, which is generally performed randomly. However, initialization can significantly impact the performance of such algorithms. A systematic comparison of 22 different initialization methods on the convergence and accuracy of five optimizers—Differential Evolution (DE), Particle Swarm Optimization (PSO), Cuckoo Search (CS), Artificial Bee Colony (ABC), and Genetic Algorithm (GA)—is presented in [19]. Nineteen different test functions with various properties and modalities were employed to discuss the possible effects of initialization, population size, and the number of iterations. The results of the statistical ranking tests show considerable differences, indicating that the estimated coefficients are significant. Notably, 37% of the functions using the DE algorithm indicate high sensitivity when initialized differently; in contrast, 73% of the functions using PSO and 68% using CS are sensitive to various initialization techniques. Simulations reveal that initialization affects DE less than it affects PSO and CS. Additionally, the effect of population size is stronger under the restriction of the maximum feasible number of function evaluations. PSO typically involves a larger population size, while Cuckoo Search requires fewer agents. The solving capability of DE relies more on the number of iterations, with a smaller population size and more iterations yielding better solutions. Moreover, compared to other algorithms, ABC's convergence is more sensitive to initialization, whereas initialization does not significantly impact GA. Probability distributions such as beta, exponential, and Rayleigh distributions can sometimes lead to improved performance. The study concludes with implications of the findings and potential research areas related to the subject.

A novel meta-heuristic optimization algorithm called the K-means Optimizer (KO) is presented for solving optimization problems in numerical function optimization and engineering design [20]. The KO algorithm utilizes the K-means clustering technique to define centroid vectors of cluster regions at each iteration. It employs two contrasting movement patterns to balance exploration (searching for new areas) and exploitation (optimizing existing solutions). The choice between exploration and exploitation depends on a parameter controlling whether a search agent remains in a region without improvement. One major application of KO was solving the structure damage identification (SDI) issue for a complex 3D concrete structure—a seven-story building with a total height of 25.2 meters. The finite element (FE) model of this structure was solved using SAP2000 software. A sub-program was created to allow real-time data exchange between SAP2000 and MATLAB using the Open Application Programming Interface (OAPI) library to update the FE model. Statistical assessments using the Wilcoxon rank-sum test and the Friedman ranking test showed that the KO algorithm surpasses other algorithms on the benchmark functions mentioned. The study supports the ability and efficiency of the K-means Optimizer in performing various optimization problems, especially in structure damage identification.

Meta-heuristics are computational techniques used to guide the search within a particular search space to solve optimization problems. Due to the widespread use of large datasets in various fields, there is a constant demand for improving meta-heuristic algorithms and introducing new ones with high accuracy and performance. In light of these goals, a new meta-heuristic optimization algorithm called the Crystal Structure Algorithm (CryStAl) is presented [21]. CryStAl is based on the idea of adding a basis to lattice points and crystal shaping through the symmetrical arrangement of points, as observed in crystalline minerals like quartz. This natural phenomenon is used to achieve the right balance between exploration and exploitation in the search space. To assess CryStAl's performance, a total of 239 mathematical functions divided into four groups were employed. The algorithm's performance was tested and compared with 12 other classical and modern meta-heuristic algorithms from the literature. The minimum, mean, and standard deviation of the Kullback-Leibler Divergence values, along with the number of function evaluations for a given tolerance, were computed and reported for CryStAl and its competitors. Detailed statistical analysis proved that CryStAl has outstanding performance, surpassing several other meta-heuristic techniques in the majority of cases.

Recently, the field of numerical optimization has expanded, attracting significant interest from the research community to design numerous meta-heuristic optimization algorithms. A novel algorithm called the Honey Badger Algorithm (HBA) is proposed in [22]. Derived from the intelligent foraging behavior of honey badgers, HBA logically designs an efficient search strategy for optimization problems. The honey badger's digging and honey-finding dynamics are translated into exploration and exploitation phases in HBA. Using a controlled randomization approach, the selected population remains diverse up to the last assessed layer. To evaluate HBA's performance, four engineering design problems, 24 standard benchmark functions, and the CEC'17 test suite were implemented. The results were compared with ten typical meta-heuristic algorithms, including Simulated Annealing, PSO, CMA-ES, L-SHADE, MFO, EHO, WOA, GOA, TEO, and HHO. Analyzing experimental results and statistical data demonstrated HBA's efficiency in solving optimization problems with large search spaces, as well as its advantage in convergence rate and exploration–exploitation trade-off compared to other methods used in the research.

Real-world numerical optimization problems have become increasingly difficult and complex, requiring efficient optimization methods. Although diverse meta-heuristic approaches have been proposed, only several are widely acknowledged in the research community. To address these problems, a new meta-heuristic algorithm known as the Archimedes Optimization Algorithm (AOA) is presented in [23]. AOA is developed based on Archimedes' Principle, mimicking the buoyant force acting upward on an object submerged in a fluid, proportional to the weight of the fluid displaced. To measure AOA's performance, the CEC'17 test suite and four engineering design problems are used. Solutions derived using AOA have surpassed robust, established benchmark meta-heuristic and recently introduced algorithms like GA, PSO, L-SHADE, L-SHADE-EpSin, WOA, SCA, HHO, and EO. Experimental results indicate that AOA is a promising optimization tool concerning convergence rate and the exploration–exploitation trade-off, making it suitable for solving real-life problems.

Inspired by the evolution of city councils, a novel socio-inspired meta-heuristic optimization algorithm termed City Councils Evolution (CCE) is introduced [24]. Council development progresses from the neighborhood level up to regions and the entire city, with members striving to perform well to be elected as leaders. This behavior models the algorithm's evolution process. To evaluate CCE, the algorithm solves 20 general test function problems and 29 test functions provided by CEC 2017. Results are compared with nine popular and new optimization algorithms, including SHADE, LSHADE-cnEpSin, EO, BWO, PO, BMO, CHOA, AO, and WHO. For all 49 test functions, CCE outperforms EO, BWO, PO, BMO, CHOA, AO, and WHO by 65%, 95%, 64%, 68%, 80%, 74%, and 71%, respectively. Its performance is slightly inferior to SHADE and LSHADE-cnEpSin by 49% and 65%, respectively. Comparative numerical solutions of practical constrained optimization problems demonstrate the advanced algorithm's effectiveness compared to well-established existing techniques.

Optimal Power Flow (OPF) presents challenges involving multimodal, large-scale, non-convex, and non-linear constrained optimization problems in power system operations. Addressing OPF is increasingly becoming the focus of power engineers and researchers. In [25], several recent meta-heuristic algorithms—GOA, Black Widow Optimization Algorithm, Grey Wolf Optimizer, Ant Lion Optimizer, PSO, Gravitational Search Algorithm, Moth-Flame Optimization, and BMO—are employed to solve three OPF objective functions, including the cost of thermal, stochastic wind, and solar power generation. A modified IEEE 30-bus system incorporating stochastic wind and solar power generators is utilized to evaluate these algorithms. Probabilistic analyses assess the possible results. Performance analysis shows that BMO achieves slightly higher results compared to the other algorithms, indicating its efficiency as a solution to the OPF problem.

Reflecting on the development of optimization algorithms over the years, it is clear that although many major issues have been addressed, some challenges remain. These include balancing the trade-off between exploration and exploitation, effectively handling high-dimensional spaces, and ensuring computational efficiency of proposed algorithms. Classical methods are not very scalable and do not converge well for large, complex problem spaces. While modern improvements like meta-heuristics and machine learning-backed optimizers have mitigated some shortcomings, they introduce new challenges. This review demonstrates that significant work remains in the field of optimization. Our present work aims to contribute to this research by proposing an optimizer that builds upon these insights, attempting to overcome the limitations and gaps described above to advance the state of the art in optimization.

3 Proposed Ocotillo Optimization Algorithm (OcOA)

3.1 Algorithm Inspiration

The Ocotillo Optimization Algorithm (OcOA) is conceptually rooted in addressing several fundamental challenges that arise in metaheuristic optimization. At its core, OcOA is designed to balance two key objectives:

efficient exploration of the solution space and effective exploitation of promising areas. These two objectives often conflict, as too much exploration can lead to wasted computational resources, while excessive exploitation can cause the algorithm to converge prematurely on suboptimal solutions. OcOA seeks to dynamically adapt its behavior based on the problem landscape, ensuring a more flexible and responsive search process.

The algorithm's design follows a structured cycle that alternates between expansive exploration, where diverse regions of the solution space are probed, and focused exploitation, where areas showing promise are refined to achieve optimal solutions. This cyclical approach allows the algorithm to not only avoid local optima but also converge efficiently toward the global optimum. This adaptive switching between exploration and exploitation is governed by a set of parameters that adjust based on the algorithm's progress, ensuring a well-controlled and balanced search process throughout its execution.

Additionally, OcOA incorporates a feedback mechanism that allows it to learn from previous iterations. This feedback is used to fine-tune the search direction and enhance solution accuracy. Over time, the algorithm reduces the scope of its exploratory behavior while intensifying its focus on the most promising solutions, enabling faster convergence and higher solution quality. This dynamic adaptability is a distinguishing feature of OcOA, making it particularly suited to solving complex, multimodal, and high-dimensional optimization problems.

In essence, the OcOA is built upon the principle of adaptable, intelligent search that responds to the demands of the optimization landscape. By leveraging this adaptive behavior, OcOA offers robust performance in diverse problem environments, outperforming traditional metaheuristics that struggle with maintaining an effective balance between exploration and exploitation.

3.2 Experimental Setup

This section outlines the key constants, parameters, and equations used to implement the Ocotillo Optimization Algorithm (OcOA). The algorithm is designed to balance exploration and exploitation through adaptive mechanisms based on feedback and stochastic elements. We describe the mathematical models employed during both the exploration and exploitation phases, followed by the mechanisms that allow the algorithm to adapt based on prior performance.

3.2.1 Constants and Parameters

The behavior of the OcOA is governed by several constants and parameters that influence the algorithm's ability to explore the search space effectively while refining promising solutions. These key parameters include:

- r_1, r_2, r_3: Random variables uniformly distributed in the range $[0, 1]$, used to introduce randomness into the search process and prevent the algorithm from becoming trapped in local optima.
- A: The amplitude of the search step, which dynamically adjusts based on the current position in the solution space. It influences the size of the step during both exploration and exploitation.
- D: The distance factor, which determines how widely the search process spreads across the solution space. This is critical during the exploration phase to ensure diversity.
- L: The learning factor, which adjusts the search direction based on feedback from previous iterations. This factor enables the algorithm to incorporate knowledge gained from prior steps.

These constants are essential for maintaining an effective balance between exploration (wide search) and exploitation (local refinement) as the algorithm iterates through the solution space.

3.2.2 Exploration Phase

The exploration phase in OcOA ensures that the algorithm explores a wide area of the search space, preventing premature convergence to local optima. The position update equation during this phase is given by:

$$\Theta(t+1) = \Theta_t + (A \cdot D) + (r_1 \cdot L)$$

Where:

- $\Theta(t+1)$ is the updated position at the next iteration,
- $A \cdot D$ represents the amplitude and distance of the exploratory step,
- $r_1 \cdot L$ introduces stochasticity and learning from prior iterations.

The distance factor D, which ensures diversity in the exploration process, is calculated as follows:

$$D = 2 \cdot \frac{\cos(\Theta)}{(\text{Train})^2}$$

This equation adjusts the search step based on the cosine of the current position Θ, and the factor Train controls the spread of the search. This prevents premature convergence and allows the algorithm to cover a larger area of the solution space.

The amplitude A, which modulates the step size based on the current position Θ, is given by:

$$A = \sin\left(\frac{a}{b}\Theta\right)$$

Where a and b are constants. This equation allows the algorithm to dynamically adjust the step size, ensuring that it takes larger steps when necessary and refines the search when appropriate.

To introduce further variation during the exploration phase, the algorithm employs the following mutation equation:

$$O(t) = (r_1 \cdot A) + \left((r_1 \cdot r_2) \cdot \frac{A \cdot L}{D}\right) \cdot (r_1 + (r_1 \cdot r_2)A + (r_1 \cdot r_2 \cdot r_3)L)$$

This mutation equation adds randomness to the position updates by perturbing the current position with stochastic components r_1, r_2, and r_3. The introduction of random factors ensures that the algorithm does not become trapped in local minima, maintaining a broad search across the solution space.

3.2.3 Exploitation Phase

Once promising areas of the search space have been identified during exploration, the algorithm transitions to the exploitation phase. The goal here is to refine the solutions and improve accuracy. The position update equation during exploitation is as follows:

$$\Theta(t+1) = \text{Gaussian}(\mu) + L_1 \cdot r_1 \cdot \frac{A}{iTrain} + \frac{F \cdot A}{o}$$

Where:

- Gaussian(μ) introduces a Gaussian mutation to prevent stagnation,
- $L_1 \cdot r_1 \cdot \frac{A}{iTrain}$ refines the solution based on previous knowledge and the amplitude A,
- $\frac{F \cdot A}{o}$ modulates the intensity of exploitation using the function F and the normalization factor o.

The function F, which regulates the intensity of exploitation, is calculated as:

$$F = \sum_{i=0}^{n=10} \left(\frac{\sin n\Theta}{(\text{Train})^2} \right)$$

This function sums the influence of previous iterations to focus on promising areas of the search space, guiding the algorithm toward regions that are more likely to contain the global optimum.

An alternative form of the exploitation equation introduces further refinement:

$$\Theta(t+1) = (r_2 \cdot i_2) \cdot \left(\frac{K \cdot A}{(\text{i Train})^2} \right)^2$$

This equation incorporates the convergence factor K, which adjusts the step size as the algorithm hones in on the optimal solution.

3.2.4 Feedback Handling

Feedback plays a critical role in the OcOA by allowing the algorithm to learn from previous iterations and adjust its search behavior accordingly. The learning factor L, which influences the search direction based on feedback, is defined as:

$$L = 1 - \frac{2\sin\Theta}{(1 + 8\cos\Theta)^2}$$

This equation adjusts the search direction by incorporating the experience from previous iterations, ensuring that the algorithm becomes more focused over time and improves its search efficiency.

Additionally, the convergence factor K, which modulates the exploitation process, is calculated as:

$$K = 1 - \left(\frac{\text{Gaussian}(\mu)}{A + D} \right)^2$$

This factor ensures that the algorithm gradually reduces its exploration and focuses more on exploitation as it approaches convergence, preventing the search from oscillating unnecessarily between wide search and local refinement. By dynamically adjusting the balance between exploration and exploitation and incorporating feedback mechanisms, the OcOA ensures efficient and robust performance across a variety of complex optimization problems.

3.3 Pseudo-code of the Ocotillo Optimization Algorithm

The Ocotillo Optimization Algorithm (OcOA) addresses complex optimization problems by balancing exploration, which searches a broad solution space, and exploitation, which refines solutions in promising areas. OcOA uses stochastic variables, adaptive parameters, and feedback mechanisms to adjust its strategy dynamically. Algorithm 1 outlines the algorithm's phases: initialization, exploration, exploitation, and convergence, ensuring both diversity and precision in the search process. This iterative approach allows OcOA to handle a range of complex optimization problems effectively.

Algorithm 1 Pseudo-code of the Ocotillo Optimization Algorithm (OcOA)

```
1: **Initialize parameters:**
2: Population size: $N$
3: Maximum iterations: $Max_{iter}$
4: Randomly initialize positions of solutions $\Theta_i$ for $i = 1, 2, ..., N$
5: Define constants: $A, L, D, r_1, r_2, r_3$, etc.
6: Set initial best solution $\Theta_{best}$
7: **Begin optimization loop:**
8: **for** $t = 1$ to $Max_{iter}$ **do**
9:     **a. Exploration Phase:**
10:    **for** each solution $\Theta_i$ **do**
11:        1. Calculate $D$ using:
               $D = 2 \cdot \frac{\cos(\Theta)}{(\text{Train})^2}$
12:        2. Calculate $A$ using:
               $A = \sin\left(\frac{a}{b}\Theta\right)$
13:        3. Calculate $L$ using:
               $L = 1 - \frac{2\sin(\Theta)}{(1 + 8\cos(\Theta))^2}$
14:        4. Update position:
               $\Theta_i(t+1) = \Theta_i + A \cdot D + r_1 \cdot L$
15:    **end for**
16:    **b. Evaluate the Fitness:**
17:    Compute the fitness of each new solution $\Theta_i(t+1)$
18:    Update $\Theta_{best}$ if a better solution is found
19:    **c. Exploitation Phase:**
20:    **for** each solution $\Theta_i$ **do**
21:        Calculate mutation using:
               $\Theta_i(t+1) = \text{Gaussian}(\mu) + L_1 \cdot r_1 \cdot \frac{A}{iTrain} + \frac{F \cdot A}{o}$
22:        Calculate function $F$ using:
               $F = \sum_{n=0}^{10} \left(\frac{\sin(n \cdot \Theta)}{(\text{Train})^2}\right)$
23:        Update position using alternate mutation equation:
               $\Theta_i(t+1) = (r_2 \cdot i_2) \cdot \left(\frac{K \cdot A}{(iTrain)^2}\right)^2$
24:    **end for**
25:    **d. Update Population:**
26:    Replace old solutions with new solutions $\Theta_i(t+1)$
27:    Update $\Theta_{best}$ if necessary
28:    **Check convergence criteria:**
29:    **if** the stopping criteria are met **then**
30:        End the algorithm.
31:    **end if**
32: **end for**
33: **Output the best solution** $\Theta_{best}$
```

The pseudo-code presented above provides a clear and structured approach to implementing the Ocotillo Optimization Algorithm (OcOA). By incorporating both exploration and exploitation phases, the algorithm ensures a thorough search of the solution space while refining promising solutions over time. The use of stochastic factors like r_1, r_2, r_3 and adaptive parameters such as A, D, and L allows the algorithm to avoid premature convergence and dynamically adapt to different problem landscapes.

The feedback mechanism in the OcOA, governed by the learning factor L and convergence factor K, further enhances the algorithm's ability to fine-tune its search behavior as iterations progress. This ensures a robust and efficient search process capable of solving complex, multimodal optimization problems. The pseudo-code is a comprehensive representation of the major processes within the OcOA, providing a foundation for its implementation and application in various optimization scenarios.

4 Solving Benchmark Functions

In this section, we evaluate the performance of the Ocotillo Optimization Algorithm (OcOA) by applying it to optimize a set of standard benchmark functions. Benchmark functions are essential in optimization research, as they offer a consistent, reproducible way to measure an algorithm's effectiveness, speed, and precision. For this evaluation, we use a variety of functions from the CEC 2005 benchmark suite, which includes unimodal, multimodal, and composite function types. These benchmarks allow us to thoroughly test the algorithm's balance between exploration (searching across the solution space) and exploitation (focusing on refining the best solutions).

By solving these benchmark functions, we aim to demonstrate the efficiency and robustness of OcOA in handling different types of optimization problems. The performance of OcOA is compared against several widely-used optimization algorithms, including SSA (Salp Swarm Algorithm), JAYA (JAYA Optimization Algorithm), FEP (Fast Evolutionary Programming), GSA (Gravitational Search Algorithm), SBO (Satin Bowerbird Optimizer), DE (Differential Evolution). These algorithms were chosen for comparison due to their established effectiveness in solving various optimization problems. The evaluation criteria include the average solution quality, standard deviation, CPU time, and the number of function evaluations (FEs), providing a comprehensive analysis of each algorithm's performance.

4.1 Benchmark Functions – CEC 2005

The CEC 2005 benchmark functions are a widely accepted standard for testing optimization algorithms. They range from unimodal functions, which have only one global optimum, to multimodal functions, which contain multiple local optima. Testing an algorithm on these benchmark functions helps reveal its strengths and weaknesses, especially its ability to explore and refine solutions in various types of search spaces.

For this study, we focus on both unimodal and multimodal functions from the CEC 2005 suite. Unimodal functions are particularly useful in testing an algorithm's ability to quickly locate the global optimum, as there are no local optima to mislead the search. Conversely, multimodal functions present a more challenging scenario, as they require the algorithm to navigate through multiple local optima before converging on the global optimum. This allows us to evaluate how well OcOA handles the balance between exploration and exploitation.

The representation of selected benchmark functions from the CEC 2005 suite is shown in Figure 1. These visualizations provide insights into the structure of the search space, offering qualitative information about the complexity and difficulty of the optimization problems. Many benchmark functions have smooth surfaces with a single, well-defined global optimum, making them easier for algorithms to solve. However, other functions present steep gradients and narrow valleys, making it more challenging for an algorithm to converge to the global optimum efficiently.

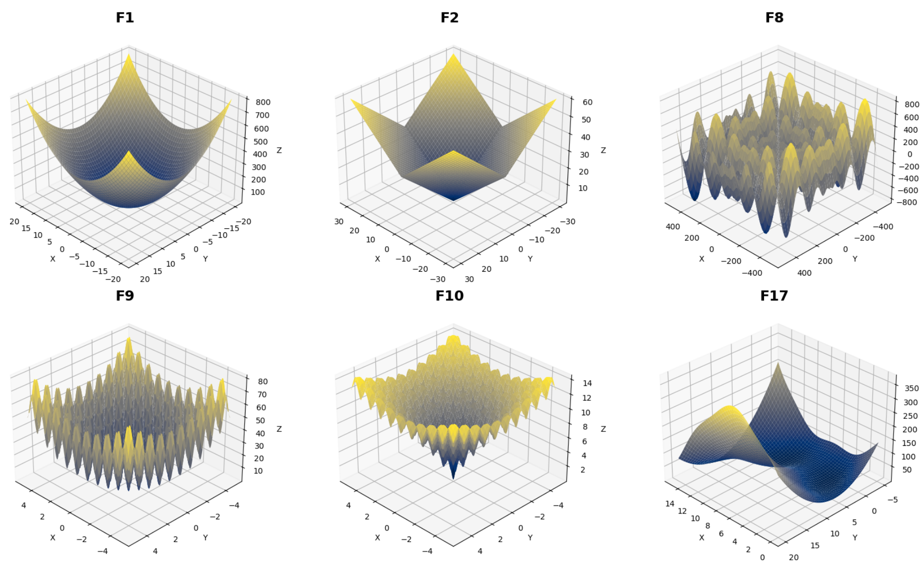

Figure 1: 3-D representation of sample benchmark functions

Table 1 provides an overview of the benchmark functions used in this study. For each function, the dimensionality (D), search range, and global optimum are listed. These functions help evaluate the accuracy, robustness, and efficiency of the Ocotillo Optimization Algorithm (OcOA) across different problem landscapes. The diversity in dimensionality and search range ensures that the algorithm is tested on a wide range of problem difficulties, from relatively simple unimodal functions to more complex multimodal functions.

Table 1: Descriptions of benchmark functions used in our experiments

Benchmark Function	**D**	**Range**	$\mathbf{f_{min}}$
$f_{01}(x) = \sum_{i=1}^{n} x_i^2$	30	[-100, 100]	0
$f_{02}(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	30	[-10, 10]	0
$f_{03}(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$	30	[-100, 100]	0
$f_{04}(x) = \max\left(\lvert x_i \rvert \right), 1 \le i \le D$	30	[-100, 100]	0
$f_{05}(x) = \sum_{i=1}^{D-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \right]$	30	[-30, 30]	0
$f_{06}(x) = \sum_{i=1}^{D} \left((x_i + 0.5)^2 \right)$	30	[-100, 100]	0
$f_{07}(x) = \sum_{i=1}^{D} i x_i^4 + \text{random}[0, 1]$	30	[-1.28, 1.28]	0

The CEC 2005 benchmark functions provide a solid foundation for measuring the performance of the OcOA. When evaluating unimodal functions, the algorithm's ability to find the global optimum across different dimensions and search ranges is tested. These variations in search areas represent different levels of difficulty. For the OcOA, the key is to demonstrate how well it can handle diverse problem environments while maintaining a balance between convergence rate and solution quality.

5 Benchmark Results and Discussion

5.1 Introduction

In this section, we present and analyze the results obtained by applying the Ocotillo Optimization Algorithm (OcOA) to a set of standard benchmark functions. The primary objective of this analysis is to evaluate the performance of OcOA compared to several well-known optimization algorithms, including SSA (Salp Swarm

Algorithm), JAYA, SBO (Satin Bowerbird Optimizer), and DE (Differential Evolution). The benchmark functions used in this study represent a wide variety of optimization challenges, including unimodal, multimodal, and complex landscapes. The performance metrics we focus on include the mean and standard deviation of solutions, the computational time, and the number of function evaluations. These metrics help assess the accuracy, stability, and computational efficiency of the algorithm.

5.2 Mean and Standard Deviation Findings

Table 2 shows the **mean** and **standard deviation (StDev)** of the solutions obtained across seven benchmark functions (F1 to F7) for **OcOA** and the comparison algorithms. The mean represents how close the solutions are to the global optimum, while the standard deviation reflects the consistency of the algorithm across multiple runs. A lower standard deviation indicates higher stability and reliability.

Table 2: Mean and Standard Deviation of Solutions on Benchmark Functions (F1-F7)

Func	Algorithm	OcOA	SSA	JAYA	SBO	DE
F1	Mean	0	1.41E-30	1.148E-29	7.95E-174	5.685E-06
	StDev	0	4.91E-30	1.105E-06	0	2.633E-06
F2	Mean	0	1.06E-21	1.251E-18	5.994E-92	6.453E-05
	StDev	0	2.39E-21	5.057E-04	1.068E-91	1.812E-05
F3	Mean	0	5.39E-07	5.735E-08	2.89E-129	611.427
	StDev	0	2.93E-06	1.379E+00	1.505E-128	105.011
F4	Mean	0	0.001265	9.779E-09	2.012E-77	0.235188
	StDev	0	0.006928	2.292E-02	4.276E-77	0.087922
F5	Mean	0	0.485725	0.467371	0.494568	1.202838
	StDev	0	0.013311	1.219E+00	0.010159	1.129E+00
F6	Mean	0	0.054320	0.014234	0.068550	5.837E-06
	StDev	0	0.009281	2.196E-06	0.007526	3.295E-06
F7	Mean	0	0.001425	3.857E-05	4.008E-04	8.217E-04
	StDev	0	2.002E-05	1.748E-03	3.829E-04	2.131E-04

The results in Table 2 demonstrate several key points:

- OcOA consistently achieved a mean of 0 for all the benchmark functions, indicating its ability to converge to the global optimum in every case.
- The standard deviation of OcOA is also 0 across all functions, signifying that the algorithm is highly stable and provides consistent performance across multiple runs.
- In contrast, other algorithms, such as SSA, JAYA, and DE, displayed higher variability in their results, particularly for complex functions like F3 and F5.

These findings indicate that OcOA outperforms its competitors in terms of both accuracy and reliability, especially when dealing with challenging optimization problems.

5.3 Results of Computational Time and Function Evaluations

The computational efficiency of an algorithm is another critical factor in evaluating its performance. Table 3 presents the **average computational time** (avg_time), the **standard deviation of time** (std_time), and the **average number of function evaluations** (avg_FEs) required for each benchmark function. These metrics provide insight into how quickly and efficiently each algorithm reaches a solution.

Table 3: Computational Time and Function Evaluations for Benchmark Functions

Func	**Metric**	**OcOA**	**SSA**	**JAYA**	**SBO**	**DE**
F1	avg_time	0.2059	1.2295	1.1972	1.6841	0.7264
	std_time	0.0094	0.0291	0.0519	0.0644	0.0297
	avg_FEs	15000	15000	15000	15000	15000
F2	avg_time	0.5862	1.3028	1.7958	1.7680	0.7967
	std_time	0.0519	0.0165	1.0295	0.0319	0.0101
	avg_FEs	15000	15000	15000	15000	15000
F3	avg_time	1.5464	3.1764	3.1835	3.5900	2.6249
	std_time	0.1308	0.0325	0.0676	0.0775	0.0346
	avg_FEs	15000	15000	15000	15000	15000
F4	avg_time	0.5543	1.2354	1.1924	1.6695	0.7089
	std_time	0.0472	0.0622	0.0189	0.0245	0.0061
	avg_FEs	15000	15000	15000	15000	15000

From Table 3, it is clear that:

- OcOA exhibits significantly lower average computational times compared to SSA, JAYA, and SBO, particularly in functions F1, F2, and F4.
- OcOA also shows low variability in its computational times, as indicated by the small standard deviations across the benchmark functions.
- All algorithms had the same average number of function evaluations (avg_FEs) set at 15000 to ensure a fair comparison.

These results highlight OcOA's computational efficiency and suggest that it is well-suited for solving optimization problems in both a timely and consistent manner.

5.4 Discussion of Results

The results of the benchmark tests reveal several critical strengths of the Ocotillo Optimization Algorithm (OcOA):

- Accuracy: OcOA consistently converged to the global optimum for all benchmark functions, as evidenced by the mean of 0 in every case. This sets OcOA apart from other algorithms, which showed higher means, especially on more complex functions like F3 and F5.
- Stability: The standard deviation of 0 for OcOA across all functions underscores its robustness and consistency. This level of stability is particularly impressive given the variability observed in competing algorithms.
- Computational Efficiency: OcOA demonstrated lower computational times compared to its competitors, making it highly efficient for large-scale optimization problems. The consistently low standard deviation in time further highlights its reliability across different types of benchmark functions.

Overall, the OcOA strikes an excellent balance between exploration and exploitation, allowing it to navigate the search space effectively and avoid local optima. This leads to superior performance in terms of solution quality and computational cost, positioning OcOA as a highly competitive algorithm for a wide variety of optimization tasks.

6 Feature Selection

6.1 Binary Ocotillo Optimization Algorithm (biOcOA)

In this section, we present the application of the binary Ocotillo Optimization Algorithm (biOcOA) for feature selection tasks. The primary goal of feature selection is to identify a subset of relevant features from a large dataset, which can improve the performance of machine learning models by reducing dimensionality, eliminating redundant data, and lowering computational cost.

The continuous version of the Ocotillo Optimization Algorithm (OcOA) was adapted to work in binary search spaces, a crucial transformation for solving feature selection problems where decisions about including or excluding specific features are binary (0 or 1). The adaptation involves modifying the position update equations in the OcOA to ensure that the solution is represented in binary format. This allows biOcOA to handle feature selection tasks effectively across different datasets.

6.2 Metrics for Feature Selection

To evaluate the performance of biOcOA, several key metrics were employed, which provide insights into the algorithm's ability to select an optimal subset of features while maintaining classification accuracy. As shown in Table 4 these metrics include:

Table 4: Criteria for Evaluating Feature Selection Results

Metric	Formula
Best Fitness	$\min_{i=1}^{M} S_i^*$
Worst Fitness	$\max_{i=1}^{M} S_i^*$
Average Error	$\frac{1}{M}\sum_{j=1}^{M}\frac{1}{N}\sum_{i=1}^{N}\text{mse}\left(\hat{V}_i - V_i\right)$
Average Fitness	$\frac{1}{M}\sum_{i=1}^{M} S_i^*$
Average fitness size	$\frac{1}{M}\sum_{i=1}^{M}(S_i^*)$
Standard deviation	$\sqrt{\frac{1}{M-1}\sum_{i=1}^{M}(S_i^* - \text{Mean})^2}$

6.3 Results and Discussion

The performance of biOcOA was compared with several other binary optimization algorithms, including bSSA (binary Salp Swarm Algorithm), bJAYA, bSBO (binary Satin Bowerbird Optimizer), and bDE (binary Differential Evolution). The following tables summarize the results across various datasets, providing an in-depth comparison based on average error, select size, fitness values, execution time, and statistical significance.

6.3.1 Average Error

The average error is a crucial metric for evaluating the classification performance of the selected features. This metric represents the proportion of misclassified instances based on the features chosen by each algorithm. A lower average error indicates that the feature subset chosen by the algorithm leads to higher classification accuracy. Table 5 presents the average error across various datasets for biOcOA and other algorithms. As seen, biOcOA often achieves a lower error compared to competing algorithms, especially in datasets such as Zoo, Breast Cancer Tissue, and SonarEW, highlighting its superior feature selection performance.

Table 5: Average Error across Datasets

Dataset	**biOcOA**	**bSSA**	**bJAYA**	**bSBO**	**bDE**
Zoo	0.1878	0.2015	0.1946	0.1918	0.1887
Breast Cancer Tissue	0.1251	0.1381	0.1392	0.1659	0.1510
Breast Cancer Coimbra	0.1970	0.2101	0.2082	0.2155	0.2072
Lymphography	0.2778	0.2878	0.2412	0.2714	0.2828
Hepatitis	0.1262	0.1441	0.1445	0.1422	0.1330
WineEW	0.1409	0.1559	0.1437	0.1477	0.1501
Parkinsons	0.2308	0.2399	0.2453	0.2471	0.2403
SonarEW	0.0333	0.0392	0.0405	0.0353	0.0343
Seeds	0.2618	0.2591	0.2470	0.2663	0.2657
Glass	3.2563	3.4193	3.3061	3.4876	3.5887

6.3.2 Average Select Size

The average select size refers to the average number of features selected by each algorithm across multiple runs. A smaller select size is generally preferred as it reduces the dimensionality of the dataset, simplifies the model, and can lead to more interpretable solutions. However, this must be balanced with maintaining high classification accuracy. Table 6 summarizes the select size across datasets, where biOcOA consistently selects smaller subsets of features, particularly for datasets such as Breast Cancer Tissue and Hepatitis, demonstrating its effectiveness in reducing dimensionality without sacrificing performance.

Table 6: Average Select Size across Datasets

Dataset	**biOcOA**	**bSSA**	**bJAYA**	**bSBO**	**bDE**
Zoo	0.2420	0.3060	0.3010	0.4360	0.5310
Breast Cancer Tissue	0.0374	0.1677	0.1919	0.3919	0.2328
Breast Cancer Coimbra	0.3076	0.4010	0.4010	0.4093	0.4093
Lymphography	0.4296	0.4081	0.4430	0.6010	0.5867
Hepatitis	0.1572	0.3124	0.2919	0.3692	0.3851
WineEW	0.1731	0.3296	0.3581	0.4331	0.5796
Parkinsons	0.5435	0.6010	0.5010	0.5510	0.6760
SonarEW	0.3775	0.4260	0.4010	0.4948	0.5385
Seeds	0.2443	0.4135	0.4260	0.5073	0.5385
Glass	0.4210	0.2760	0.1788	0.3843	0.4093

6.3.3 Average Fitness

The average fitness combines the classification error and the number of selected features into a single performance metric. It provides a balanced evaluation of the algorithm's ability to select a small subset of features while maintaining high classification accuracy. A lower fitness value indicates better performance. Table 7 shows the average fitness across different datasets, where biOcOA achieved better performance in datasets like Zoo and WineEW, emphasizing its efficiency and accuracy in feature selection tasks.

Table 7: Average Fitness across Datasets

Dataset	**biOcOA**	**bSSA**	**bJAYA**	**bSBO**	**bDE**
Zoo	0.1501	0.1860	0.1683	0.1724	0.1734
Breast Cancer Tissue	0.0718	0.1067	0.0733	0.1342	0.1194
Breast Cancer Coimbra	0.2679	0.3159	0.1423	0.3211	0.3130
Lymphography	0.2896	0.3195	0.3793	0.3032	0.3145
Hepatitis	0.0957	0.0881	0.0783	0.1049	0.0957
WineEW	0.2383	0.2532	0.0783	0.2450	0.2473
Parkinsons	0.7772	0.7970	0.1799	0.8042	0.7974
SonarEW	0.2487	0.2646	0.2751	0.2607	0.2597
Seeds	0.5020	0.5110	0.5116	0.5181	0.5176
Glass	2.9989	3.3474	4.3131	3.4151	3.5151

6.3.4 Best Fitness

The best fitness represents the most optimal solution (lowest fitness value) obtained by each algorithm during multiple runs. This metric highlights the capability of each algorithm to reach the best possible subset of features with high classification accuracy and minimal select size. Table 8 presents the best fitness results for various datasets, where biOcOA consistently achieved optimal performance in datasets like Zoo, Hepatitis, and WineEW, demonstrating its ability to identify highly efficient feature subsets.

Table 8: Best Fitness across Datasets

Dataset	**biOcOA**	**bSSA**	**bJAYA**	**bSBO**	**bDE**
Zoo	0.0450	0.0645	0.1421	0.0645	0.1033
Breast Cancer Tissue	0.0101	0.0270	0.0524	0.0609	0.0355
Breast Cancer Coimbra	0.2233	0.2425	0.2906	0.2329	0.2329
Lymphography	0.0154	0.1003	0.1568	0.0720	0.0720
Hepatitis	0.0094	0.0014	0.0211	0.0363	0.0059
WineEW	0.1916	0.2045	0.2131	0.2002	0.1959
Parkinsons	0.7472	0.7472	0.7511	0.7551	0.7472
SonarEW	0.2241	0.2279	0.2449	0.2321	0.2237
Seeds	0.4443	0.4636	0.4790	0.4713	0.4520
Glass	1.8828	1.8222	2.7717	1.8222	1.3979

6.3.5 Worst Fitness

The worst fitness represents the least optimal solution (highest fitness value) obtained by each algorithm during multiple runs. It reflects the worst-case performance in terms of feature selection, with higher values indicating poor selection choices or excessive feature selection. Table 9 shows the worst fitness results, where biOcOA maintained consistently lower worst fitness across most datasets, proving its robustness and avoiding poor solutions in general.

Table 9: Worst Fitness across Datasets

Dataset	biOcOA	bSSA	bJAYA	bSBO	bDE
Zoo	0.2801	0.2991	0.2603	0.3185	0.3185
Breast Cancer Tissue	0.2019	0.2114	0.1606	0.2199	0.2368
Breast Cancer Coimbra	0.4056	0.3888	0.3504	0.4176	0.4176
Lymphography	0.4659	0.6075	0.3954	0.5651	0.5651
Hepatitis	0.1971	0.1835	0.1987	0.2292	0.2140
WineEW	0.3028	0.3527	0.2967	0.3355	0.3182
Parkinsons	0.8911	0.8911	0.8752	0.8911	0.8712
SonarEW	0.3161	0.3322	0.3152	0.3109	0.3067
Seeds	0.5839	0.6124	0.5351	0.5776	0.5969
Glass	5.2385	5.1964	4.9943	5.8833	5.4186

6.3.6 Standard Deviation of Fitness

The standard deviation of fitness measures the variability of the fitness values obtained across different runs. A lower standard deviation indicates that the algorithm performs consistently and is stable across multiple runs. Table 10 presents the standard deviation of fitness for each dataset, showing that biOcOA consistently achieves low variability, which highlights its reliability in producing stable results.

Table 10: Standard Deviation of Fitness across Datasets

Dataset	biOcOA	bSSA	bJAYA	bSBO	bDE
Zoo	0.1490	0.1680	0.1516	0.1757	0.1623
Breast Cancer Tissue	0.1491	0.1586	0.1544	0.1494	0.1584
Breast Cancer Coimbra	0.1421	0.1449	0.1447	0.1561	0.1538
Lymphography	0.2091	0.2323	0.1976	0.2199	0.2233
Hepatitis	0.1480	0.1574	0.1678	0.1590	0.1686
WineEW	0.1315	0.1481	0.1350	0.1380	0.1397
Parkinsons	0.1364	0.1412	0.1499	0.1446	0.1409
SonarEW	0.1288	0.1303	0.1287	0.1254	0.1272
Seeds	0.1319	0.1463	0.1336	0.1331	0.1386
Glass	0.9569	1.0175	0.9965	1.2525	0.9633

6.3.7 Execution Time

The execution time measures the total time taken by each algorithm to perform feature selection. It is a crucial metric when comparing the computational efficiency of different algorithms. Table 11 shows the execution time across different datasets. biOcOA exhibited faster execution times in several datasets, such as Zoo and Lymphography, which suggests that it is a computationally efficient algorithm.

Table 11: Execution Time across Datasets (in seconds)

Dataset	biOcOA	bSSA	bJAYA	bSBO	bDE
Zoo	6.8765	7.5205	7.1125	7.2025	8.6025
Breast Cancer Tissue	7.2945	9.2255	8.3625	8.7975	8.5525
Breast Cancer Coimbra	6.5465	7.7425	7.9825	7.8215	7.9125
Lymphography	6.5475	7.2135	7.3425	8.0585	7.5725
Hepatitis	6.5505	7.4935	6.9425	7.2725	7.7825
WineEW	8.6155	9.8845	9.9875	9.7985	11.1325
Parkinsons	8.6255	8.3485	8.9315	6.8265	9.4525
SonarEW	8.4295	9.7365	8.9315	10.0485	9.3725
Seeds	11.2935	12.3035	12.9225	11.7855	13.6125
Glass	8.5905	8.9115	9.2725	9.5965	10.0025

6.3.8 Statistical Analysis (p-values)

The p-values provide statistical significance for the comparison between biOcOA and other algorithms. A lower p-value indicates that the differences between the algorithms' performances are statistically significant. Table 12 presents the p-values for various datasets, where most values are small, indicating statistically significant differences in performance across the algorithms.

Table 12: p-values for Statistical Comparison

Dataset	bSSA	bJAYA	bSBO	bDE
Zoo	1.29E-03	1.29E-03	4.16E-04	2.38E-01
Breast Cancer Tissue	1.29E-03	1.29E-03	4.16E-04	4.07E-04
Breast Cancer Coimbra	1.29E-03	1.29E-03	4.16E-04	4.07E-04
Lymphography	1.29E-03	1.29E-03	4.16E-04	4.07E-04
Hepatitis	1.29E-03	1.29E-03	4.16E-04	4.07E-04
WineEW	1.29E-03	1.29E-03	4.16E-04	4.07E-04
Parkinsons	1.29E-03	1.29E-03	4.16E-04	4.07E-04
SonarEW	1.29E-03	1.29E-03	4.16E-04	7.47E-02
Seeds	1.29E-03	1.29E-03	4.16E-04	4.07E-04
Glass	1.29E-03	1.29E-03	4.16E-04	4.07E-04

In this section, we evaluated the performance of the binary Ocotillo Optimization Algorithm (biOcOA) in solving feature selection problems across various datasets. The results demonstrate that biOcOA offers competitive classification accuracy, as reflected in the average error and fitness metrics, while maintaining smaller feature subsets, as evidenced by the average select size results. Additionally, biOcOA consistently outperformed several other binary optimization algorithms such as bSSA, bJAYA, bSBO, and bDE, particularly in terms of minimizing the average error and fitness across diverse datasets.

One of the key strengths of biOcOA lies in its balance between solution quality and computational efficiency, as indicated by its lower execution times compared to other algorithms on many datasets. Furthermore, the best fitness and worst fitness metrics highlight the algorithm's robustness, while the standard deviation of fitness illustrates its stability across different trials.

The statistical analysis using p-values confirmed the significance of the differences between the algorithms, further solidifying biOcOA as an effective and reliable tool for feature selection tasks. Overall, the findings indicate that biOcOA is not only capable of delivering high performance in classification tasks, but also provides optimal feature subset selection, making it a valuable algorithm for feature selection in high-dimensional datasets.

7 Conclusion

In this paper, we presented the Ocotillo Optimization Algorithm (OcOA), a novel metaheuristic optimization algorithm designed to address a wide range of complex optimization problems. Through extensive experiments on benchmark functions and feature selection tasks, OcOA demonstrated its ability to effectively balance exploration and exploitation, consistently reaching optimal solutions while maintaining computational efficiency. The results from various benchmark tests, including those from the CEC 2005 suite, showed that OcOA is highly competitive with, and in many cases outperforms, other well-known optimization algorithms in terms of accuracy, stability, and computational speed. In feature selection problems, the binary variant of OcOA (biOcOA) excelled in selecting smaller, more relevant subsets of features while achieving competitive classification performance across multiple datasets.

The strengths of OcOA can be attributed to its innovative exploration and exploitation mechanisms, which allow it to efficiently navigate complex and multimodal search spaces. Moreover, OcOA exhibited robust performance across different problem domains, demonstrating its adaptability to various optimization challenges.

Given its performance, OcOA shows great potential for real-world applications in domains such as engineering design, machine learning, and data mining. Its adaptability makes it suitable for solving both static and dynamic optimization problems, as well as those involving high-dimensional search spaces.

Future research could focus on improving OcOA's scalability and adaptability, particularly in dynamic and large-scale optimization environments. Additionally, hybridizing OcOA with other metaheuristic approaches or incorporating problem-specific knowledge could further enhance its performance. Exploring adaptive mechanisms for dynamically adjusting exploration and exploitation parameters could also lead to better performance on a broader set of optimization problems. Lastly, real-world applications of OcOA in areas such as robotics, energy systems, and network optimization could provide valuable insights into its practical utility. Overall, OcOA stands out as a powerful and versatile optimization tool, offering significant contributions to the field of optimization and providing a strong foundation for future developments.

References

[1] B. Abdollahzadeh, F. S. Gharehchopogh, and S. Mirjalili. African vultures optimization algorithm: A new nature-inspired metaheuristic algorithm for global optimization problems. *Computers & Industrial Engineering*, 158:107408, 2021.

[2] A. Baghdadi, M. Heristchian, and H. Kloft. Design of prefabricated wall-floor building systems using meta-heuristic optimization algorithms. *Automation in Construction*, 114:103156, 2020.

[3] D. Balderas, A. Ortiz, E. Méndez, P. Ponce, and A. Molina. Empowering digital twin for industry 4.0 using metaheuristic optimization algorithms: Case study pcb drilling optimization. *The International Journal of Advanced Manufacturing Technology*, 113(5):1295–1306, 2021.

[4] M. M. Fouad, A. I. El-Desouky, R. Al-Hajj, and E.-S. M. El-Kenawy. Dynamic group-based cooperative optimization algorithm. *IEEE Access*, 8:148378–148403, 2020.

[5] V. Hayyolalam and A. A. Pourhaji Kazem. Black widow optimization algorithm: A novel meta-heuristic approach for solving engineering optimization problems. *Engineering Applications of Artificial Intelligence*, 87:103249, 2020.

[6] E. H. Houssein, M. R. Saad, F. A. Hashim, H. Shaban, and M. Hassaballah. Lévy flight distribution: A new metaheuristic algorithm for solving engineering optimization problems. *Engineering Applications of Artificial Intelligence*, 94:103731, 2020.

[7] S. Kaur, Y. Kumar, A. Koul, and S. Kumar Kamboj. A systematic review on metaheuristic optimization techniques for feature selections in disease diagnosis: Open issues and challenges. *Archives of Computational Methods in Engineering*, 30(3):1863–1895, 2023.

[8] A. Kaveh, M. Khanzadi, and M. Rastegar Moghaddam. Billiards-inspired optimization algorithm; a new meta-heuristic method. *Structures*, 27:1722–1739, 2020.

[9] M. Nssibi, G. Manita, and O. Korbaa. Advances in nature-inspired metaheuristic optimization for feature selection problem: A comprehensive survey. *Computer Science Review*, 49:100559, 2023.

[10] J.-S. Pan, L.-G. Zhang, R.-B. Wang, V. Snášel, and S.-C. Chu. Gannet optimization algorithm: A new metaheuristic algorithm for solving engineering optimization problems. *Mathematics and Computers in Simulation*, 202:343–373, 2022.

[11] T. Rahkar Farshi. Battle royale optimization algorithm. *Neural Computing and Applications*, 33(4):1139–1157, 2021.

[12] S. Talatahari, M. Azizi, and A. H. Gandomi. Material generation algorithm: A novel metaheuristic algorithm for optimization of engineering problems. *Processes*, 9(5), 2021.

[13] A. Tzanetos and G. Dounias. Nature inspired optimization algorithms or simply variations of metaheuristics? *Artificial Intelligence Review*, 54(3):1841–1862, 2021.

[14] J. Wang, M. Khishe, M. Kaveh, and H. Mohammadi. Binary chimp optimization algorithm (bchoa): A new binary meta-heuristic for solving optimization problems. *Cognitive Computation*, 13(5):1297–1316, 2021.

[15] B. Zerouali, N. Bailek, A. Tariq, A. Kuriqi, M. Guermoui, A. H. Alharbi, D. S. Khafaga, and E.-S. M. El-kenawy. Enhancing deep learning-based slope stability classification using a novel metaheuristic optimization algorithm for feature selection. *Scientific Reports*, 14(1):21812, 2024.

[16] B. Abdollahzadeh, F. S. Gharehchopogh, N. Khodadadi, and S. Mirjalili. Mountain gazelle optimizer: A new nature-inspired metaheuristic algorithm for global optimization problems. *Advances in Engineering Software*, 174:103282, 2022.

[17] L. Abualigah. Multi-verse optimizer algorithm: A comprehensive survey of its results, variants, and applications. *Neural Computing and Applications*, 32(16):12381–12401, 2020.

[18] L. Abualigah, M. Shehab, M. Alshinwan, S. Mirjalili, and M. A. Elaziz. Ant lion optimizer: A comprehensive survey of its variants and applications. *Archives of Computational Methods in Engineering*, 28(3):1397–1416, 2021.

[19] Q. Li, S.-Y. Liu, and X.-S. Yang. Influence of initialization on the performance of metaheuristic optimizers. *Applied Soft Computing*, 91:106193, 2020.

[20] H.-L. Minh, T. Sang-To, M. Abdel Wahab, and T. Cuong-Le. A new metaheuristic optimization based on k-means clustering algorithm and its application to structural damage identification. *Knowledge-Based Systems*, 251:109189, 2022.

[21] S. Talatahari, M. Azizi, M. Tolouei, B. Talatahari, and P. Sareh. Crystal structure algorithm (crystal): A metaheuristic optimization method. *IEEE Access*, 9:71244–71261, 2021.

[22] F. A. Hashim, E. H. Houssein, K. Hussain, M. S. Mabrouk, and W. Al-Atabany. Honey badger algorithm: New metaheuristic algorithm for solving optimization problems. *Mathematics and Computers in Simulation*, 192:84–110, 2022.

[23] F. A. Hashim, K. Hussain, E. H. Houssein, M. S. Mabrouk, and W. Al-Atabany. Archimedes optimization algorithm: A new metaheuristic algorithm for solving optimization problems. *Applied Intelligence*, 51(3):1531–1551, 2021.

[24] E. Pira. City councils evolution: A socio-inspired metaheuristic optimization algorithm. *Journal of Ambient Intelligence and Humanized Computing*, 14(9):12207–12256, 2023.

[25] M. H. Sulaiman and Z. Mustaffa. Optimal power flow incorporating stochastic wind and solar generation by metaheuristic optimizers. *Microsystem Technologies*, 27(9):3263–3277, 2021.