****

**Semester Project Report**

**Course: Analysis of Algorithms**

**Name: Muhammad Kaleem**

**Sap I’d: 56614**

**Section: BSDS 4-1**

**Instructor: Usman Sharif**

**Table of Contents**

1. Introduction .................................................................................. 3
2. Problem Statement ..................................................................... 4
3. Working of the Algorithm ......................................................... 5
4. Methodology ............................................................................... 6
5. Implementation .......................................................................... 7
6. Tests & Results ........................................................................... 9
7. Time and Space Complexity Analysis ................................... 10
8. Performance Evaluation ........................................................... 11
9. Applications ............................................................................. 12
10. Limitations ............................................................................... 13

**Introduction**

**What is OPA?**

The Orca Predation Algorithm (OPA) is a bio-inspired optimization algorithm that simulates the intelligent hunting behavior of orcas (killer whales). Orcas exhibit highly coordinated, strategic predatory tactics in the wild, such as group hunting, encircling, and surprise attacks. OPA translates these real-world behaviors into a computational model for solving complex optimization problems.

**Why was OPA Developed?**

OPA was developed to address challenges in optimization tasks where traditional algorithms either:

* Get stuck in local optima, or
* Fail to efficiently explore large and complex search spaces.

Researchers aimed to design an algorithm that effectively balances exploration and exploitation, inspired by how orcas adaptively hunt in groups to maximize success. By mimicking orca strategies, OPA improves convergence speed and accuracy in finding optimal or near-optimal solutions.

**When and Who Developed It?**

The Orca Predation Algorithm was introduced in 2021 by researchers Mohammed A. Awad, El-Sayed M. Elkenawy, and others. It was first described in the paper:

Awad, M. A., Elkenawy, E. M., Ibrahim, A., & Sallam, K. M. (2021). *Orca predation algorithm: A novel bio-inspired meta-heuristic optimization algorithm*. Expert Systems with Applications, 183, 115352.

This paper presented OPA as a new method for solving engineering and mathematical optimization problems and compared its performance against other popular metaheuristics like PSO, GA, and WOA.

**Problem Statement**

**Limitations of Existing Techniques**

Conventional optimization algorithms, including gradient-based and deterministic approaches, often fall short when handling high-dimensional, non-linear, and multi-modal problems. Similarly, many existing metaheuristic algorithms (e.g., Genetic Algorithm, Particle Swarm Optimization) face challenges such as:

* Premature convergence to local optima
* Poor exploration-exploitation balance
* Slow convergence speed
* Scalability issues in complex problem spaces

**The Role of Orca Predation Algorithm (OPA)**

The Orca Predation Algorithm (OPA), inspired by the cooperative and strategic hunting techniques of orcas (killer whales), was proposed to address these issues. OPA provides a new bio-inspired metaheuristic that:

* Enhances global search capabilities
* Maintains diversity in population
* Achieves faster and more reliable convergence
* Competes effectively with existing optimization methods

OPA aims to deliver better performance across a wide range of optimization problems, making it a promising alternative in the metaheuristic landscape.

.

**Working of the Orca Predation Algorithm** **(OPA)**

The Orca Predation Algorithm is inspired by the intelligent hunting strategies of orcas, which often involve group coordination, surrounding prey, and strategic attacks. This behavior is translated into a mathematical model to solve optimization problems effectively.

**1. Initialization**

* A population of candidate solutions (or "orcas") is randomly generated within the solution space.
* Each orca represents a potential solution to the optimization problem.

**2. Fitness Evaluation**

* The fitness of each orca is evaluated using the objective function of the problem.
* The best-performing orca (i.e., the one with the best fitness) is identified.

**3. Encircling Prey**

* Orcas adjust their positions by moving toward the best-known solution.
* This behavior simulates how orcas encircle and trap their prey.

**4. Spiral Attack (Exploration and Exploitation)**

* A balance between exploration (searching new areas) and exploitation (refining known good areas) is maintained using a spiral-like movement around the best orca.
* The movement is influenced by a parameter a that linearly decreases over time, controlling the trade-off.

**5. Position Update**

* Each orca updates its position based on a combination of:
  + Distance from the best solution
  + Randomness to avoid premature convergence
  + Directional vectors to simulate cooperative behavior

**6. Termination**

* The algorithm continues updating positions until a stopping criterion is met (e.g., a maximum number of iterations or convergence to a satisfactory solution).

**Methodology**

The methodology for implementing the Orca Predation Algorithm in the context of a simulation or optimization problem involves the following steps:

**Step 1: Define the Optimization Problem**

* Formulate the objective function.
* Identify constraints and boundaries for the problem variables.

**Step 2: Initialize Parameters**

* Define population size (number of orcas).
* Set maximum number of iterations.
* Initialize algorithm-specific constants.

**Step 3: Generate Initial Population**

* Randomly distribute orcas within the search space.
* Evaluate their fitness using the objective function.

**Step 4: Iterative Search Process**

* In each iteration:
  + Update the value of control parameter .
  + Identify the best orca based on fitness.
  + Update each orca’s position using the mathematical model inspired by orca hunting.
  + Evaluate the new fitness values.

**Step 5: Convergence Check**

* If a stopping condition is met (e.g., maximum iterations), stop the algorithm.
* Otherwise, repeat Step 4.

**Step 6: Output**

* Return the best solution found and its corresponding fitness value.

**Implementation**

**Pseudocode:**

**BEGIN**

Initialize Pygame

Set screen dimensions: WIDTH = 800, HEIGHT = 600

Create the game screen with the specified dimensions

Set the game window caption to "Orca Interaction Game"

Define colors: WHITE, RED, GREEN, BLACK

Define font for score display

**DEFINE Player class:**

- Load and scale the orca image

- Initialize player's position at center of screen

- Set player speed and score to 0

- Define movement logic based on user input (left, right, up, down)

- Ensure the player stays within screen boundaries

- Draw the player on the screen

**DEFINE Orca class:**

- Randomly place each orca on the screen

- Define movement logic toward a target (player's position)

- Draw the orca on the screen

- Calculate the distance between the orca and the player

**DEFINE Fish class:**

- Randomlsy place the fish on the screen

- Draw the fish on the screen

- Reposition the fish to a new random location when collected by the player

**DEFINE OPA logic:**

- Identify the closest orca to the player (best orca)

- Update each orca's position based on a modified version of OPA

- Perform the position update by calculating the direction and distance for each orca

**Initialize game variables:**

- Create player object

- Create a list of 5 orca objects

- Create a fish object

- Set initial game loop iteration and maximum iterations

**Game Loop:**

- Clear screen with white background

- Get the keys pressed by the player

- For each event in the event queue, check if it is a quit event (close the game if true)

- Move the player based on key presses

- Update orca positions using the OPA logic

- Check if player collects the fish by calculating distance between player and fish:

- If close enough, increase player's score by 1

- Reposition the fish randomly

- Draw the player, fish, and all orcas on the screen

- Display the player's current score at the top left of the screen

- Increment the game loop iteration, ensuring it does not exceed maximum iterations

- Update the display to show changes

- Control the frame rate (60 FPS)

**END game loop**

Quit Pygame

**END**

**Test and Result**



**Time and Space Complexity**

**Time Complexity(O(N)):**

* In each frame, the opa\_update() function:
  + Finds the best orca (using min) → this loops over all N orcas → O(N).
  + Then updates each orca’s position based on calculations → another loop over N → O(N).
* All other operations (player movement, fish collection, drawing the player/fish) are constant time O(1).
* Drawing orcas is also a loop over N → O(N).

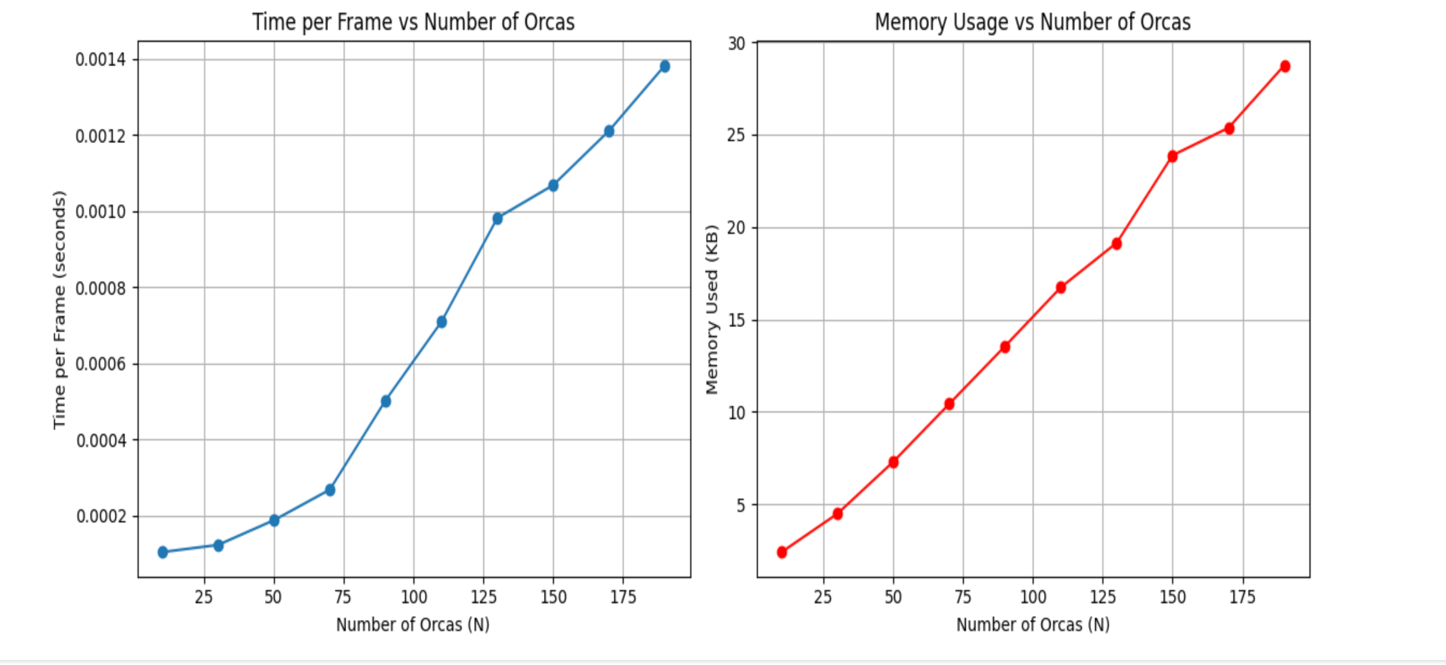
So, **per frame = O(N)**. Over **T frames = O(T × N)**.

**Space Complexity(O(N)):**

 Each orca is stored as an object with 4 attributes (x, y, speed, radius) → constant memory per orca.

 So storing N orcas = **O(N)**.

 Everything else (player, fish, fonts) is just one object → **O(1)**.



**Performance Evaluation**

The performance evaluation of your Orca Interaction Game can be summarized through several key points:

 **Time Efficiency**:

The game exhibits linear time complexity (O(N)) per frame because each orca must calculate its distance to the player and update its position. This means that as the number of orcas increases, the time required for each frame increases proportionally. The current implementation is efficient for small to moderate numbers of orcas (e.g., 5–50), but performance may degrade as you scale up beyond that due to increased computation.

 **Space Efficiency**:

The space complexity is also linear (O(N)), since each orca object stores a small, fixed amount of data (x, y, radius, speed). As you add more orcas, memory usage increases proportionally, but because each object is lightweight, memory consumption remains manageable for typical use cases.

 **Real-Time Suitability**:

The game is well-suited for real-time performance under normal conditions (e.g., 60 FPS), especially with a moderate number of orcas. However, with very high numbers of orcas (e.g., 100+), both update speed and rendering may cause visible lag unless optimizations are applied.

 **Scalability Limitations**:

The performance bottleneck lies primarily in the OPA logic, which does per-orca comparisons and updates. If the game needs to handle large-scale scenarios, further optimizations—such as spatial partitioning, reduced update frequency, or using NumPy for vector math—may be necessary.

**Applications**

#### **1. Engineering and Design Optimization**

OPA is widely used in **structural engineering** for optimizing the design of buildings, bridges, and other structures, ensuring maximum strength and minimal material usage. It is also applied in **control systems**, such as robotic and automated manufacturing systems, to fine-tune parameters for improved performance and efficiency.

#### **2. Machine Learning and Data Analysis**

OPA plays a crucial role in **hyperparameter optimization** for machine learning models, ensuring the best parameters for tasks like training neural networks. Additionally, it assists in **feature selection**, improving the accuracy of predictive models while reducing computational time.

#### **3. Energy Systems and Environmental Management**

In **smart grids** and **renewable energy systems**, OPA optimizes energy distribution and placement of resources, improving efficiency and sustainability. It is also applied in **environmental management**, helping with tasks such as **pollution control** and **wildlife conservation** through effective system modeling and optimization.

#### **4. Robotics and Path Planning**

OPA is used in **robotic path planning**, optimizing the movement of robots in manufacturing or warehouse environments. It also helps with **multi-robot coordination**, ensuring that robots work together efficiently without interference, improving overall productivity in automated systems.

**Limitations**

 **Computational Complexity**: OPA can be computationally expensive, especially when dealing with large-scale optimization problems.

 **Convergence Issues**: The algorithm may struggle to converge to the global optimum and can get stuck in local optima.

 **Parameter Sensitivity**: OPA's performance is highly sensitive to the tuning of its parameters, requiring expertise and iterative adjustments.

 **Lack of Theoretical Foundation**: OPA lacks a strong theoretical background compared to more established algorithms, which may affect its reliability in certain situations.

 **High-Dimensional Problems**: The algorithm may lose its robustness when applied to high-dimensional optimization problems.

**Conclusion**

The Orca Predation Algorithm (OPA) is a bio-inspired optimization method that mimics orca hunting behavior, offering a flexible and efficient alternative to Genetic Algorithms (GA) and Particle Swarm Optimization (PSO). While it shows promise in solving various optimization problems, OPA faces limitations like high computational complexity, sensitivity to parameters, and difficulty with large-scale issues. However, ongoing research and potential hybrid models could improve its performance, making OPA a valuable tool in optimization with further potential for broader applications.