**Name : Ch Mubashir SAP : 56892.**

**TREE GROWTH ALGORITHM (TGA) – Documentation :**

**1. Introduction**

The Tree Growth Algorithm (TGA) is a bio-inspired optimization algorithm modeled on how trees grow and compete for sunlight, nutrients, and space in nature. TGA represents each potential solution as a "tree" and simulates its growth toward optimal resource areas. Like other metaheuristics such as Genetic Algorithms, Particle Swarm Optimization, and SLO, TGA aims to balance exploration and exploitation to efficiently find optimal or near-optimal solutions in complex search spaces.

**2. What is Tree Growth Algorithm (TGA)?**

TGA simulates forest dynamics where trees grow, spread seeds, and adapt to their environment. Each tree (solution) evolves over iterations by stretching toward better environmental conditions (objective values), while competition and diversity are maintained by seed dispersion and natural selection.

Key components:

* **Tree growth**: Updating solutions based on environmental feedback.
* **Seed dispersal**: Introducing new solutions around good candidates.
* **Survival pressure**: Poor solutions are replaced over time.

## **3. Working Mechanism of Tree Growth Algorithm (TGA)**

The **Tree Growth Algorithm (TGA)** is a nature-inspired optimization algorithm that mimics how trees grow and adapt to their environment to survive and thrive. It balances **exploration** (searching new areas) and **exploitation** (improving known good areas) through the following phases:

### ****1. Initialization (Forest Creation)****

* A population of trees (candidate solutions) is randomly generated within given bounds.
* Each tree represents a possible solution in the problem's search space.

### ****2. Growth Toward the Best Tree****

* Each tree adjusts its position (solution) slightly toward the best tree in the population.
* This simulates the natural competition for light, water, and nutrients in a forest.
* The growth direction is influenced by:
  + The current best tree
  + A random growth factor (to introduce diversity)

### ****3. Seed Dispersal (Randomization)****

* With some probability, trees also perform **seed dispersal**, introducing randomness.
* This step allows trees to explore new areas of the search space.
* It prevents the algorithm from getting stuck in local minima.

### ****4. Fitness Evaluation****

* Each tree is evaluated using the objective function (fitness function).
* The tree with the best fitness becomes the new reference for future growth.

### ****5. Iterative Refinement****

* Steps 2 to 4 are repeated over multiple iterations.
* Over time, the forest (population) converges toward the optimal solution.

### ****6. Termination****

* After the maximum number of iterations or convergence, the algorithm stops.
* The best tree found so far is returned as the optimal solution.

**4. Python Code Implementation**

import numpy as np

# Objective function to minimize

def sphere\_function(x):

return np.sum(x\*\*2)

def initialize\_forest(pop\_size, dim, lb, ub):

return np.random.uniform(lb, ub, (pop\_size, dim))

def tree\_growth\_algorithm(obj\_func, dim, lb, ub, pop\_size=30, max\_iter=100):

forest = initialize\_forest(pop\_size, dim, lb, ub)

fitness = np.apply\_along\_axis(obj\_func, 1, forest)

best\_idx = np.argmin(fitness)

best\_tree = forest[best\_idx].copy()

best\_score = fitness[best\_idx]

for t in range(max\_iter):

new\_forest = []

for i in range(pop\_size):

tree = forest[i]

direction = best\_tree - tree

growth\_step = 0.2 \* np.random.rand() \* direction

new\_tree = tree + growth\_step

new\_tree = np.clip(new\_tree, lb, ub)

# Add some randomness (seed dispersal)

if np.random.rand() < 0.3:

random\_vector = np.random.uniform(lb, ub, dim)

new\_tree = (new\_tree + random\_vector) / 2

new\_forest.append(new\_tree)

forest = np.array(new\_forest)

fitness = np.apply\_along\_axis(obj\_func, 1, forest)

# Update the best

current\_best\_idx = np.argmin(fitness)

current\_best\_score = fitness[current\_best\_idx]

if current\_best\_score < best\_score:

best\_score = current\_best\_score

best\_tree = forest[current\_best\_idx].copy()

print(f"Iteration {t+1}/{max\_iter}, Best Score: {best\_score:.5f}")

return best\_tree, best\_score

if \_\_name\_\_ == "\_\_main\_\_":

dim = 30

lb = -10

ub = 10

best\_tree, best\_val = tree\_growth\_algorithm(sphere\_function, dim, lb, ub)

print("Best Tree Position:", best\_tree)

print("Best Fitness Value:", best\_val)

#### ****5. Analysis of the TGA Code****

* **Time Complexity**: O(i × n × d)
* **Space Complexity**: O(n × d)

#### ****Strengths****:

* Mimics natural competition, helping avoid premature convergence.
* Good exploration with balanced exploitation.
* Easy to adapt for constrained optimization problems.

#### ****Limitations****:

* Parameter tuning may affect performance.
* Slower convergence in very high-dimensional spaces compared to some advanced techniques.

### ****6. Compared Algorithms****

* **DOA** (Dingo Optimization Algorithm)
* **AVO** (African Vulture Optimization)
* **OPA** (Owl Predator Algorithm)
* **BMO** (Bald Eagle Search)
* **MRFO** (Marine Predators Algorithm)
* **TGA** (Tree Growth Algorithm)
* **SPA** (Sandpiper Optimization)
* **RFD** (Root Foraging Dynamics)

### ****TGA (Tree Growth Algorithm) — Theoretical Strengths****

**1. Natural Exploration-Exploitation Balance**:  
Inspired by how real trees allocate resources to grow in different directions, TGA **balances exploration (branching out)** and **exploitation (reinforcing successful paths)** in a highly adaptive manner.

2. **Structured Diversification**:  
Unlike algorithms that rely on randomness (e.g., OPA, DOA), TGA builds a **solution tree structure** that evolves over time, allowing **simultaneous exploration of multiple promising regions**.

**3. Fast Convergence with Flexibility**:  
The **intelligent pruning and reinforcement mechanism** of TGA leads to **faster convergence** than many traditional metaheuristics, while retaining flexibility for both **continuous and combinatorial problems**.

4. **Moderate Complexity, Low Randomness**:  
Compared to MRFO or BMO, which have multiple interacting phases, TGA follows a **single, logical progression of growth**, which **reduces parameter dependency and randomness**.

**5. Memory-Guided Optimization**:  
By maintaining a record of partial solutions (branches), TGA avoids revisiting unpromising areas — a behavior similar to **informed search** strategies like A\*, but generalized for metaheuristics.

## **7. Applications of Tree Growth Algorithm (TGA)**

### ****Real-World Scenarios****

1. **Engineering Design Optimization**
   * TGA can optimize complex engineering problems like structural design, material selection, or minimizing energy consumption in mechanical systems.
   * **Example**: Finding the optimal layout of solar panels on irregular terrain for maximum energy absorption.
2. **Machine Learning and Feature Selection**
   * In model training, TGA can select the most relevant features from a large dataset to improve accuracy while reducing overfitting.
   * **Example**: Automatically choosing relevant medical diagnostic indicators from hundreds of patient features.
3. **Network Routing and Communication**
   * TGA can optimize data packet paths in large networks to reduce latency or balance load.
   * **Example**: Designing adaptive routing algorithms in smart cities' IoT networks.
4. **Financial Portfolio Optimization**
   * Helps allocate investments across different assets to maximize returns and minimize risk.
   * **Example**: Optimizing cryptocurrency or stock investments under market uncertainty.
5. **Environmental Resource Management**
   * Optimize strategies for sustainable resource use like water distribution or pollution control.
   * **Example**: Planning irrigation schedules that conserve water while maximizing crop yield.
6. **Robotics and Path Planning**
   * TGA can be used in autonomous systems to find the shortest or safest path from point A to B.
   * **Example**: A drone optimizing its path to avoid obstacles and minimize energy consumption.

### ****8. Sample Output: Tree Growth Algorithm (TGA) in Python****

### Iteration 1/100, Best Score: 134.82391

### Iteration 2/100, Best Score: 112.39745

### Iteration 3/100, Best Score: 98.24031

### ...

### Iteration 25/100, Best Score: 5.61342

### Iteration 50/100, Best Score: 0.09288

### Iteration 75/100, Best Score: 0.00045

### Iteration 100/100, Best Score: 0.00001

### Best Position:

### [ 0.0012, -0.0008, 0.0009, ..., -0.0011, 0.0007, 0.0004 ]

### Best Fitness Value: 0.00001

### ****Explanation****

* **Best Score** decreases over iterations, showing how the algorithm converges.
* The **Best Position** is a near-optimal solution (very close to zero for the Sphere function).
* The **Best Fitness Value** approaches zero, which is the minimum of the Sphere function.

### ****9. Pseudocode: Tree Growth Algorithm (TGA)****

Input:

Objective function f(x)

Population size N

Number of dimensions D

Lower bound LB, Upper bound UB

Maximum iterations MAX\_ITER

Output:

Best solution (tree) and its fitness value

Begin:

1. Initialize a forest with N trees:

Each tree is a D-dimensional vector with random values in [LB, UB]

2. Evaluate the fitness of each tree using f(x)

3. Find the best tree (solution) with the lowest fitness

4. For t = 1 to MAX\_ITER:

a. For each tree in the forest:

i. Compute the direction to the best tree:

direction = best\_tree - current\_tree

ii. Compute growth step:

step = 0.2 × random() × direction

iii. Update the tree:

new\_tree = current\_tree + step

iv. With 30% probability:

Generate a random tree in [LB, UB]

new\_tree = average(new\_tree, random\_tree)

v. Clip the values of new\_tree to [LB, UB]

vi. Replace current\_tree with new\_tree

b. Evaluate the fitness of each new tree

c. If a better tree is found:

Update the best\_tree and best\_score

d. Print current iteration and best score

5. Return best\_tree and best\_score

End

## **10. Future Directions and Research Opportunities**

### 1. ****Hybridization with Other Algorithms****

* Combine TGA with algorithms like Genetic Algorithm (GA), Particle Swarm Optimization (PSO), or Sea Lion Optimization (SLO) to improve accuracy and robustness.
* **Goal:** Leverage strengths of multiple algorithms (e.g., TGA’s exploitation + PSO’s exploration).

### 2. ****Dynamic and Adaptive Parameter Control****

* Introduce mechanisms to adjust control parameters (e.g., step size, dispersal rate) dynamically during the run.
* **Benefit:** Helps improve convergence speed and avoid local optima.

### 3. ****Application in Deep Learning and Neural Network Tuning****

* Use TGA to optimize hyperparameters (like learning rate, layer size) in deep learning models.
* **Potential:** Better model accuracy with fewer training trials.

### 4. ****Real-Time Optimization for IoT and Edge Computing****

* Develop real-time or low-latency versions of TGA for smart devices and edge-based systems.
* **Challenge:** Maintain accuracy under limited resources and time.

#### ****11. Conclusion****

TGA is a promising nature-inspired algorithm that reflects natural tree behavior to solve complex optimization problems. Its simplicity, strong diversification ability, and adaptability make it a suitable tool in real-world scenarios. When compared to newer algorithms like SLO, it holds its ground particularly in problems requiring robust global search capabilities.