**Ch Mubashir\_56892.**

**Analysis of Algorithm :**

**TREE GROWTH ALGORITHM (TGA) – REPORT**

**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.

This mechanism helps avoid local optima and ensures a broad search of the solution space.

**3. 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)

**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

**4. Analysis of the TGA Code**

* **Time Complexity**: O(i × n × d)

Where:

i = number of iterations (MAX\_ITER)

n = number of trees (POP\_SIZE)

d = dimensionality of the problem (DIM)

* **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.

**5. Comparison with Other Algorithms**

TGA provides a strong balance between global and local search through its dual mechanisms (growth and seed dispersal). Compared with SLO, TGA:

* Has more diversity due to environmental-inspired randomness.
* May be more robust in complex, multimodal landscapes.
* But might require slightly more iterations to converge tightly.

**6. Applications of Tree Growth Algorithm (TGA)**

**Real-World Scenarios :**

* Engineering Design Optimization
* Feature Selection in Machine Learning
* Network Routing and Load Balancing
* Financial Portfolio Optimization
* Environmental Resource Management
* Robotics Path Planning

**7. 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.

Github Repository Link :

<https://github.com/choudharymubashir11/AOA-Semester-Project/tree/main>