PSO

Particle Swarm Optimization for Terrain Analysis

Using Python and Matplotlib

Introduction

- Particle Swarm Optimization (PSO) is a computational method used for optimizing a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality.
- The primary objective of this project is to use PSO to identify optimal center points within a specified terrain. These centers need to fulfill certain criteria based on the terrain's properties and data points.
- This technique can be applied in various fields such as environmental monitoring, resource allocation, and strategic planning where optimal placement of points (like machine systems, sensors, etc.) is crucial.

Function Defined:

Convert Degrees, Minutes, Seconds (DMS) to Decimal Degrees:

Check if a Point is Within Distance Using Haversine Formula:

Calculate Haversine Distance Between Two Points:

Check if a Point is Within Elevation Range:

Calculate Fitness of a Particle(coverage):

PSO Algorithm Execution

Initialize Particles:

- Particles are randomly initialized within the defined geographic bounds.
- Each particle is checked to ensure it falls within the specified elevation range. If not, it is re-initialized.

Initialize Personal and Global Bests:

- Each particle's initial position is considered its personal best (`p_best`).
- The particle with the best fitness among all particles is considered the global best (`g_best`).

Particle History:

- A history of particle positions (`particle_history`) is maintained for visualization.

Iterations(Working)

Velocity and Position Update:

- Each particle's velocity is updated based on its inertia, its distance from its personal best, and its distance from the global best.
- Each particle's position is updated based on its new velocity.
- Boundary conditions are applied to ensure particles remain within the defined geographic bounds.

Elevation Constraint Check:

- Each particle is checked to ensure it remains within the specified elevation range. If not, it is re-initialized within the bounds.

Fitness Evaluation:

- The fitness of each particle is evaluated based on how many target points it covers.
- If a particle's current fitness is better than its personal best fitness, the personal best is updated.
- If a particle's current fitness is better than the global best fitness, the global best is updated.

Track Particle History:

- The current positions of all particles are stored in the particle history for visualization purposes.

Early Termination Check:

- If the global best particle covers all target points, the algorithm terminates early.

Final Output

Return Results:

- The best particle (`g_best`), its fitness (`g_best_fitness`), the set of covered points (`g_best_covered`), the particle history (`particle_history`), the list of centers (`centers_list`), and the list of points covered (`points_covered_list`) are returned.

Data Preparation

Converting Degrees, Minutes, Seconds (DMS) to Decimal Degrees (DD)

For eg:

dd = degrees + minutes / 60 + seconds / 3600

Haversine Distance Calculation

Calculating the distance between two geographic coordinates

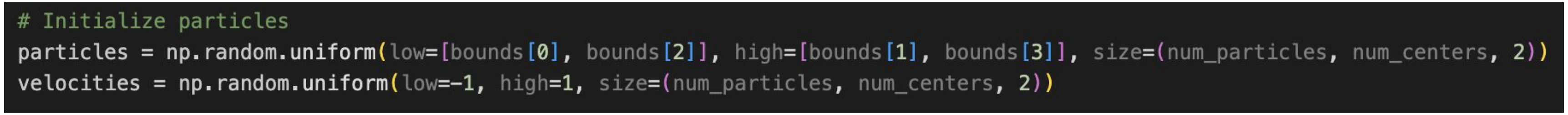
The `haversine` function calculates the great-circle distance between two points on the Earth's surface given their latitude and longitude in decimal degrees. It uses the haversine formula, which accounts for the spherical shape of the Earth.

Particle Swarm Optimization (PSO) Algorithm

Initialization of particles and velocities:

The particles and velocities arrays are initialized for a swarm optimization algorithm. Each particle has `num_centers` centers represented by 2D coordinates (x, y). The particles array has dimensions `(num_particles, num_centers, 2)`, where each particle's centers are bounded by specified coordinate ranges, allowing optimization in a 2D space.

We can say that each particle has a dimension of num_center*2(i.e. Num_center = 3 -> 3*2 = 6)



```
# Initialize particles
particles = np.random.uniform(low=[bounds[0], bounds[2]], high=[bounds[1], bounds[3]], size=(num_particles, num_centers, 2))
velocities = np.random.uniform(low=-1, high=1, size=(num_particles, num_centers, 2))
```

Formula Used to calculate the velocity

$$v_i^{t+1} = \underbrace{v_i^t}_{\text{Inertia}} + \underbrace{c_1 r_1 (pbest_i^t - p_i^t)}_{\text{Personal influence}} + \underbrace{c_2 r_2 (gbest^t - p_i^t)}_{\text{Social influence}}$$

The main parameters used to model the PSO are:

- $S(n) = \{s_1, s_2, \ldots, s_n\}$: a swarm of n particles
- s_i: an individual in the swarm with a position p_i and velocity v_i, $i \in [|1, n|]$
- p_i: the position of a particle s_i
- v_i: the velocity of a particle p_i
- $pbest_i$: the best solution of a particle
- $gbest$: the best solution of the swarm (Global)
- f: fitness function
- c_1, c_2: acceleration constants (cognitive and social parameters)
- r_1, r_2: random numbers between 0 and 1
- t: the iteration number

```
w = 0.5  # Inertia weight
c1 = 1.5  # Cognitive (particle) weight
c2 = 1.5 # Social (swarm) weight
```

Ref - https://www.baeldung.com/cs/pso

No. of Particles - 30
No. of Iterations - 100

Fitness Function

Calculating the fitness of particles based on coverage:

The fitness function evaluates particle performance by counting covered points within a specified distance and acceptable elevation range (0-2000 meters). It ensures solutions respect terrain constraints, optimizing for practical applicability.

The fitness function evaluates each particle based on its ability to cover data points within a specified distance and elevation range.
For each data point, the function checks if any center within a particle covers the point.
The function reads the elevation data and only counts points covered within the 0-2000 meters elevation range.

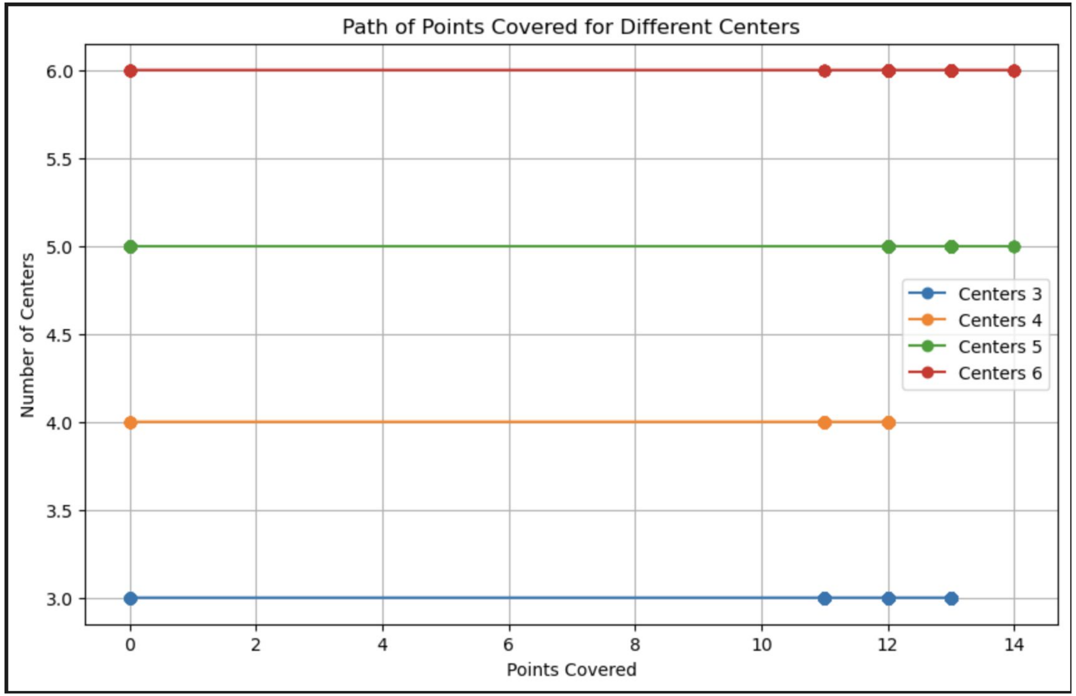
Terrain Data and Animation

Loading terrain data using `rasterio`

Creating an animation using `matplotlib.animation.FuncAnimation`

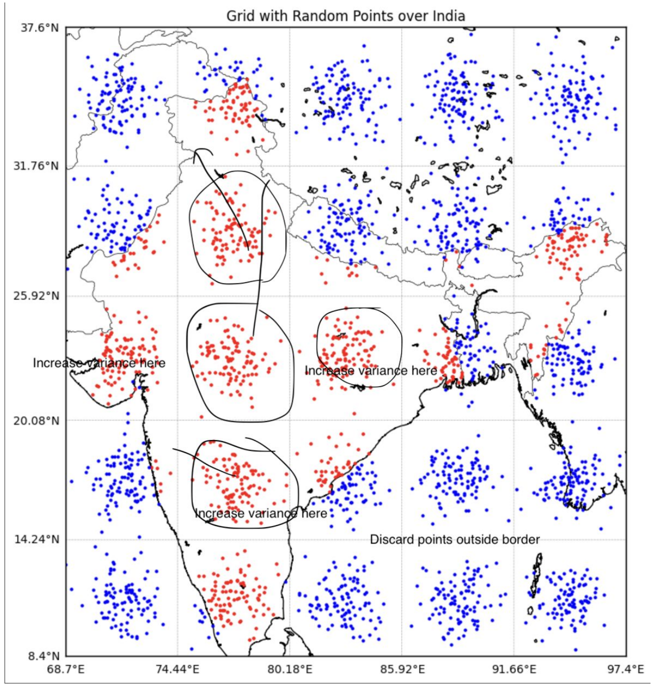
Results and Analysis

Center vs no. of points

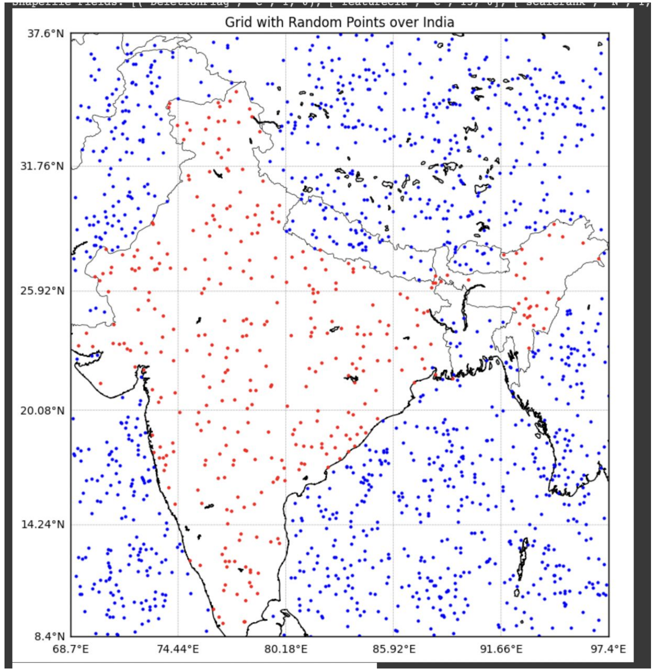


DataSet Generated

Before Correction:



After Correction:



Animations generated for Centers:

center 3 - https://github.com/ParaDhim/PSO/blob/main/pso_optimization.mp4
center 4 - https://github.com/ParaDhim/PSO/blob/main/pso_optimization4.mp4
center 5 - https://github.com/ParaDhim/PSO/blob/main/pso_optimization5.mp4
center 6 - https://github.com/ParaDhim/PSO/blob/main/pso_optimization6.mp4