# ग्रीष्मकालीन औद्योगिक परियोजना प्रशिक्षण प्रतिवेदन

# Summer Industrial Project Training Report

on

## Machine Point Placement Using Particle Swarm Optimization in Constructive Simulation

**जनू -जुलाई, 2024**

**Jun - July, 2024**

**(दो महने/Two Months)**


**द्वारा प्रस्तुत** / Submitted By -
**नाम** / Name : Paras Dhiman
**पंजीकृत क्रमांक** / Roll No. : 2021482
**पाठ्यक्रम** / Course : B. tech (CSB- Compuer Science and Bioscience)
**विश्वविद्यालय** / University: IIIT-D (Indraprastha Institute Of Information Technology Delhi)

**परीक्षक के अधीन** / Under the supervision of
नाम / Name: Mr. Sourabh Jaiswal
पद / Designation: वैज्ञानिक 'ई' / Scientist 'E'
कार्यालय / Office: आईएसएसए / ISSA

**पद्धत अध्ययन एवं विश्लेषण संस्थान**
Institute for Systems Studies and Analyses

**रक्षा अनुसंधान एवं विकास संगठन**
Defence R&D Organisation

**रक्षा मंत्रालय** / Ministry of Defence
**मेटकाफ भवन, दिल्ली**-110054
Metcalfe House, Delhi-110054

---

## अभ्यर्थी द्वारा घोषणा
## DECLARATION BY THE CANDIDATE

I hereby declare that the work which is being presented by me in this project/study entitled "**Machine Point Placement Using Particle Swarm Optimization in Constructive Simulation**" is an authentic record of my own work carried out during the period from 1st June 2018 to 20th July 2018 under the supervision of Mr. Sourabh Jaiswal, Scientist 'E', Institute for Systems Studies and Analyses, Defence R&D Organisation, Ministry of Defence, Metcalfe House, Delhi 110054.

---

**द्वारा** / By:
**श्री** / Mr.
Paras Dhiman
**पंजीकृत क्रमांक** / Roll No.: 2021482
**पाठ्यक्रम** / Course: B.tech (CSB- Compuer Science and Bioscience)
**विश्वविद्यालय** / University: IIIT-D (Indraprastha Institute Of Information Technology Delhi)

**तारीख** / Date: June 2024 / July 2024

# अभिवादन
# ACKNOWLEDGEMENT

I am grateful to the Director, ISSA and Head of HRD for providing me the opportunity to carry out my project at this esteemed organization. I wish to express my deep gratitude to Mr. Sourabh Jaiswal, Scientist 'E', ISSA, DRDO for providing guidance and support throughout the project work. I would also like to thank all other scientists, especially the technical staff and my fellow trainees. They were always there when needed and provided all the help and facilities required for the completion of my project.

# सिस्टम अध्ययन और विश्लेषण संस्थान के बारे में
# Institute for System Studies and Analyses (ISSA)

Institute for System Studies and Analyses (ISSA) is a premier institution involved in systems analysis of Defence Systems. It provides analysis support to the top echelons of the three services, SA to RM, and DRDO HQs for scientific decision making. It also provides systems analysis support to sister labs and other institutions under the Ministry of Defence. ISSA is primarily devoted to systems analysis and specializes in modelling and simulation in a wide range of applications.

ISSA adopts state-of-the-art info-technologies such as Computer Networking, Software Engineering, Distributed Database, Distributed Simulation, Web Technologies, Situational Awareness, and Soft-Computing Techniques in the development of complex simulation products.

# रक्षा अनुसंधान एवं विकास संगठन के बारे में
# About the Defence Research and Development Organization

DRDO was formed in 1958 from the amalgamation of the then already functioning Technical Development Establishment (TDEs) of the Indian Army and the Directorate of Technical Development & Production (DTDP) with the Defence Science Organization (DSO). Today, DRDO has more than 50 labs engaged in developing Defence Technologies covering various disciplines like aeronautics, armaments, electronics, combat vehicles, engineering systems, instrumentation, missiles, advanced computing and simulation, special materials, naval systems, life sciences, training, information systems, and agriculture. DRDO is backed by over 5000 scientists and about 25,000 other scientific, technical, and supporting personnel.

## Vision

Make India prosperous by establishing a world-class science and technology base and provide our Defence Services with a decisive edge by equipping them with internationally competitive systems and solutions.

## Mission

- Design, develop, and lead to production state-of-the-art sensors, weapon systems, platforms, and allied equipment for our Defence Services.
- Provide technological solutions to the Defence Services to optimise combat effectiveness and to promote the well-being of the troops.
- Develop infrastructure and committed quality manpower and build a strong technology base.

## Core Competence

The Department of Defence Research and Development (R&D) is working for the indigenous development of weapons, sensors, and platforms required by the three wings of the Armed Forces. To fulfill this mandate, the Department of Defence Research and Development (R&D) is closely working with academic institutions, Research and Development (R&D) Centres, and production agencies of Science and Technology (S&T) Ministries/Departments in the Public & Civil Sector, including Defence Public Sector Undertakings & Ordnance Factories.

# अंतर्वस्तु

## CONTENTS

# 1. Introduction

## 1.1 Motivation

In various industrial and research applications, the optimal placement of machines or sensors is crucial for maximizing efficiency and effectiveness. This challenge is especially significant in contexts such as resource monitoring, environmental sensing, and network coverage. The goal is to place a limited number of machines or sensors in such a way that they cover a specified area as effectively as possible.

Traditional methods of optimization can be limited by their ability to handle complex and high-dimensional search spaces. In this context, Particle Swarm Optimization (PSO) emerges as a powerful technique due to its flexibility and efficiency in solving optimization problems, including those related to machine placement.

The motivation behind using PSO for optimizing machine point placement is to leverage its ability to explore and exploit the search space efficiently. By employing PSO, we can address the challenges associated with machine placement, such as coverage optimization, elevation constraints, and boundary conditions.

## 1.2 Background

Particle Swarm Optimization (PSO) is a metaheuristic optimization technique inspired by the social behavior of birds flocking or fish schooling. It was introduced by Kennedy and Eberhart in 1995 and has since been applied to a wide range of optimization problems.

The core idea behind PSO is to simulate the behavior of a swarm of particles moving through a search space to find optimal solutions. Each particle represents a potential solution and adjusts its position based on its own experience and that of its neighbors. This collaborative approach helps the swarm converge towards the optimal solution.

PSO is particularly well-suited for complex optimization problems where traditional methods might struggle. Its simplicity, ease of implementation, and ability to handle nonlinear and high-dimensional problems make it a valuable tool for machine placement optimization.

# 2. PSO

## 2.1 Description of PSO

PSO is a population-based optimization algorithm that simulates the social behavior of swarms to find optimal solutions. Each particle in the swarm represents a potential solution to the problem and moves through the search space influenced by its own best-known position and the best-known positions of its neighbors.

Key components of PSO include:

- **Particles**: Represent potential solutions and have a position and velocity in the search space.
- **Personal Best (p_best)**: The best position a particle has encountered.
- **Global Best (g_best)**: The best position encountered by any particle in the swarm.
- **Velocity Update**: Particles update their velocities based on their personal best and the global best positions.
- **Position Update**: Particles update their positions based on their updated velocities.

## 2.2 Algorithm Steps

The PSO algorithm involves the following steps:

1. **Initialization**: Randomly initialize particles' positions and velocities within the search space.
2. **Evaluation**: Evaluate the fitness of each particle based on a fitness function.
3. **Update Personal Best**: Update the personal best position of each particle if the current position is better.
4. **Update Global Best**: Update the global best position if any particle's personal best is better.
5. **Velocity and Position Update**: Update velocities and positions of particles based on the personal and global best positions.
6. **Termination**: Repeat the process until a stopping criterion is met (e.g., a maximum number of iterations or convergence).

## 2.3 Applications of PSO

PSO has been successfully applied to various optimization problems, including:

- **Function Optimization**: Finding the maximum or minimum of a function.

- **Scheduling**: Optimizing schedules for tasks or resources.
- **Network Design**: Designing efficient network topologies.
- **Machine Learning**: Tuning hyperparameters of machine learning models.
- **Resource Allocation**: Optimizing the allocation of resources in various applications.

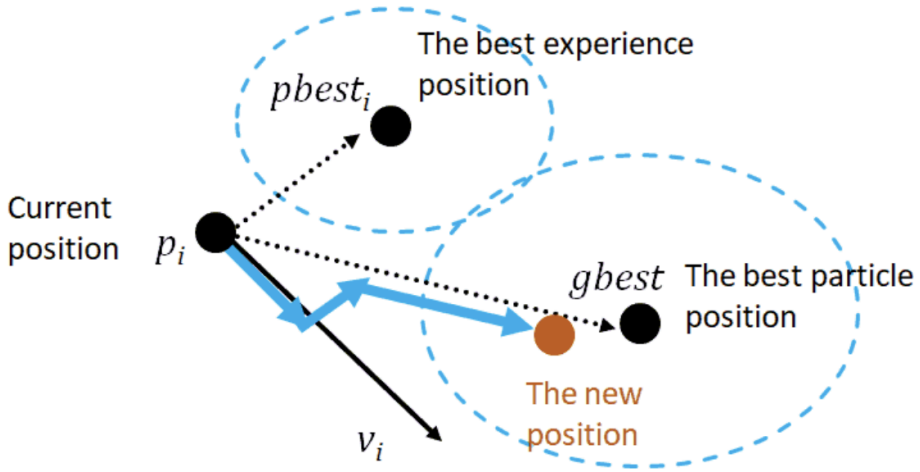
In the context of machine point placement, PSO can effectively handle the complexities of coverage optimization and elevation constraints.

### 2.4 Parameters Of PSO

**The main parameters used to model the PSO are:**

- $S(n) = \{s_1, s_2, \ldots, s_n\}$ : a swarm of nn particles
- $s_i$ : an individual in the swarm with a position pipi and velocity vivi, i∈[|1,n|]i∈[|1,n|]
- $p_i$ : the position of a particle sisi
- $v_i$ : the velocity of a particle pipi
- $pbest_i$ : the best solution of a particle
- gbest : the best solution of the swarm (Global)
- ff: 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

$$\mathbf{v_i^{t+1} = v_i^t + c_1 r_1 (pbest_i^t - p_i^t) + c_2 r_2 (gbest^t - p_i^t)}$$
$$\mathbf{p_i^{t+1} = p_i + v_i^{t+1}}$$



# 3. Data Set Generation

The process of generating the dataset necessary for our project. This dataset is crucial for subsequent modeling and simulation tasks. The following steps outline the methodology:

## 3.1 Defining the Boundaries of India

To focus our area of interest, we defined the latitude and longitude boundaries that encompass India. This helps in constraining our dataset to the relevant region. The defined boundaries are:

- Latitude: 8.4° to 37.6° N
- Longitude: 68.7° to 97.4° E

## 3.2 Loading the Shapefile

We used a shapefile containing global country boundaries to extract the specific boundary for India. This shapefile is essential for identifying whether generated points lie within India's borders. The shapefile used named as:

`ne_10m_admin_0_countries.shp`

## 3.3 Identifying the Country Name Field

The shapefile contains various fields, one of which stores the names of the countries. We identified this field to filter and extract the shape of India. By printing the fields, we identified the correct field that contains the country names, which is necessary for extracting India's boundary.

## 3.4 Extracting the Shape of India

Using the identified country name field, we extracted the polygon that represents India's boundary from the shapefile. This polygon is used to determine whether points generated in the dataset lie within India.

## 3.5 Point-In-Polygon Test

We created a function to check if a given point lies within the boundaries of India using the extracted shape. This function uses the Shapely library to perform a point-in-polygon test, ensuring accurate determination of point locations.

### 3.6 Defining the Grid

To systematically generate points, we divided the area of interest into a grid. Each grid cell is defined by its latitude and longitude ranges, creating a structured approach for point generation. The grid size is calculated as:

- Latitude grid size: (max_lat - min_lat) / 5
- Longitude grid size: (max_lon - min_lon) / 5

### 3.7 Creating the Map

Using the Basemap library, we created a map centered around India. This map serves as the basis for plotting the generated points. The map includes boundaries, coastlines, and other geographical features to provide context for the generated data.

### 3.8 Generating Random Points

Within each grid cell, we generated random points using a Gaussian distribution to ensure a varied distribution of points. We differentiated points inside India from those outside and categorized them accordingly. The number of points generated per grid cell is adjusted based on whether the cell is inside or outside India.

### 3.9 Plotting the Points

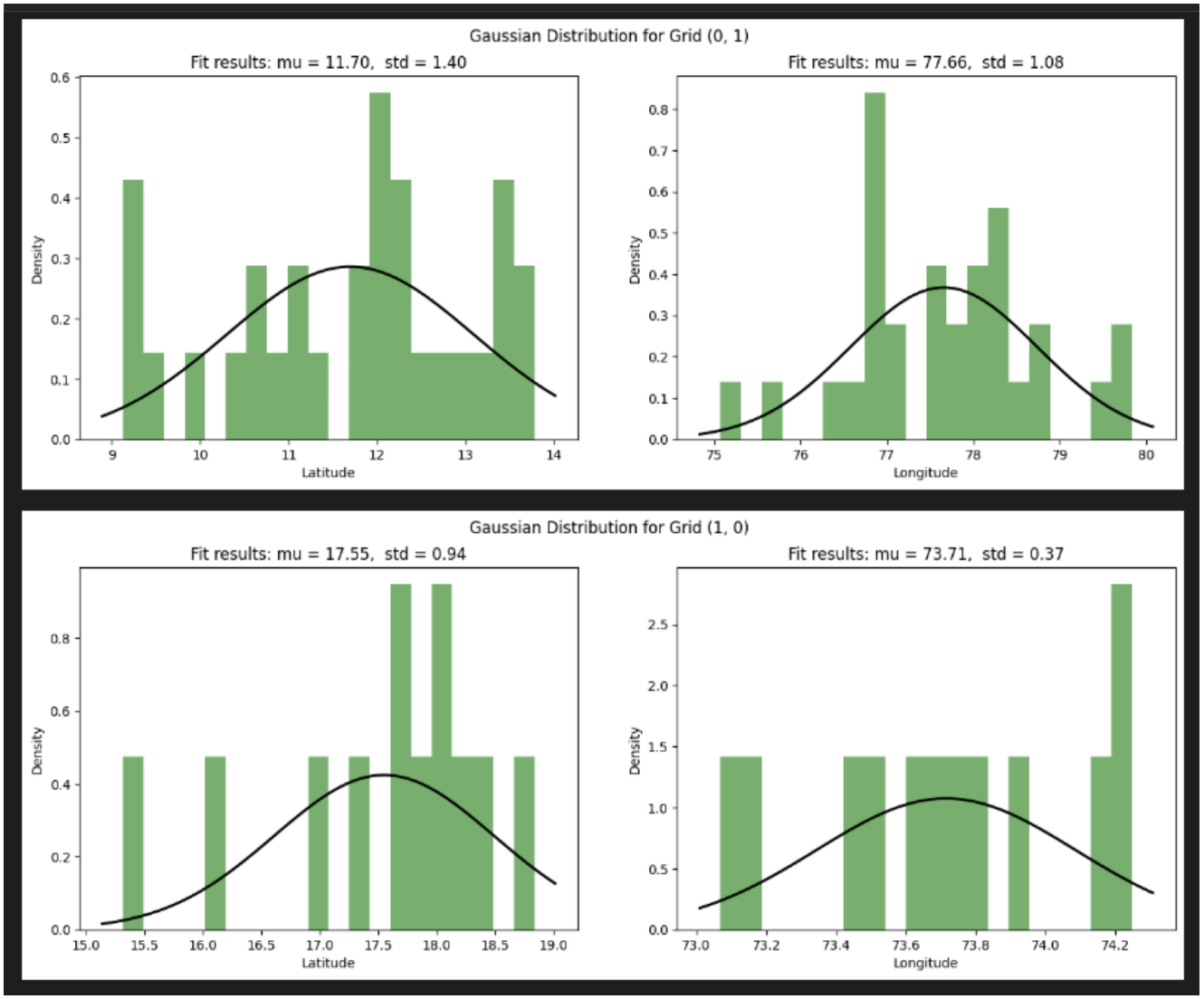
We plotted the generated points on the map, using different colors to distinguish between points inside and outside India. Points inside India are marked in red, while points outside are marked in blue. This visual representation helps in verifying the accuracy of the point generation process.

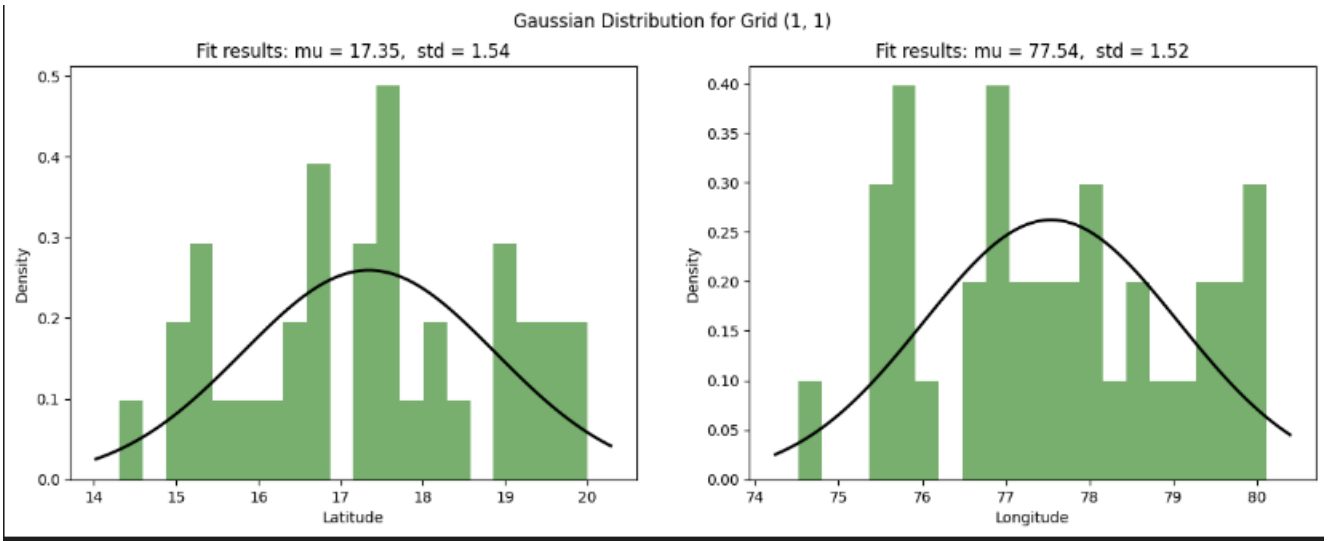
### 3.10 Plotting Distribution for Each Grid

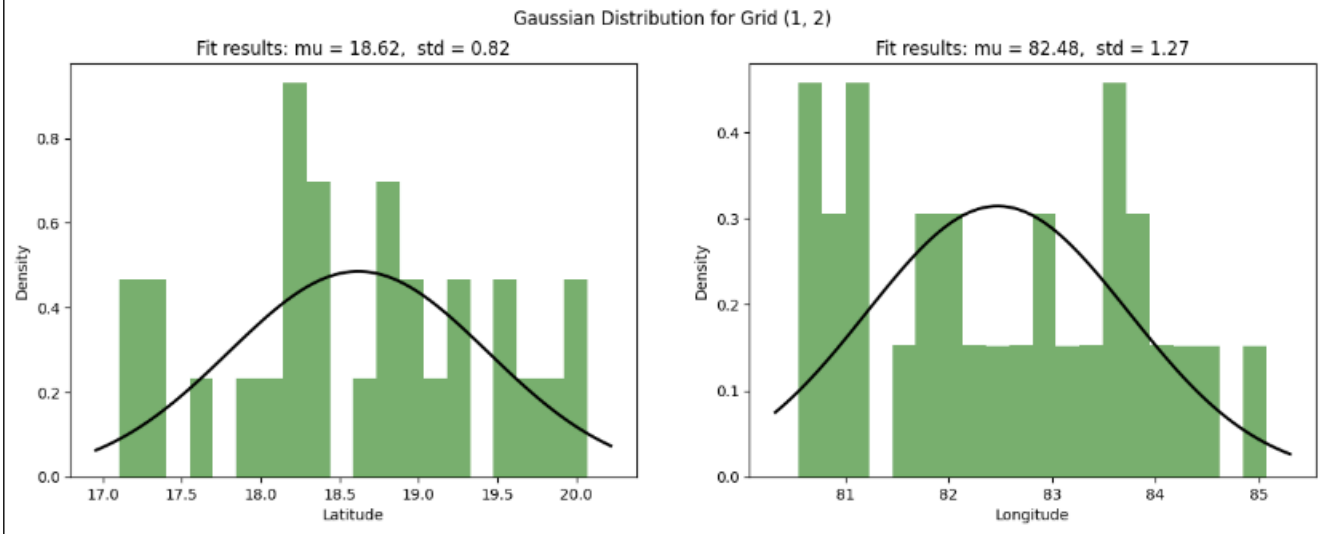
For each grid cell covering India, we plotted the distribution of points to visualize their placement. This detailed plotting ensures that we can analyze the distribution and density of

points within each grid cell, providing insights into the dataset's structure.

Some of them are :




Gaussian Distribution for Grid (1, 1)
Fit results: mu = 17.35, std = 1.54
Density
Latitude
Fit results: mu = 77.54, std = 1.52
Density
Longitude


Gaussian Distribution for Grid (1, 2)
Fit results: mu = 18.62, std = 0.82
Density
Latitude
Fit results: mu = 82.48, std = 1.27
Density
Longitude

# 4. Python

## 4.1 Python Libraries Used

The implementation of PSO for machine point placement relies on several Python libraries:

- **numpy**: Provides support for numerical operations and array manipulations.
- **matplotlib**: Used for plotting graphs and creating visualizations, including animations.
- **rasterio**: Handles raster data, such as terrain data, for geographical analysis.
- **pandas**: Manages data manipulation and reading from CSV files.
- **geopandas**: Extends `pandas` to work with geospatial data, such as shapefiles.
- **shapely**: Facilitates geometric operations and spatial analysis.
- **scipy**: Offers spatial data structures and algorithms, although not used in this specific implementation.

## 4.2 Code Explanation

The Python code for PSO involves several key functions:

1. **dms_to_dd(degrees, minutes, seconds, direction)**
    - Converts DMS coordinates to decimal degrees for geographical analysis.
    - Handles both positive and negative directions.
2. **haversine(coord1, coord2)**
    - Calculates the distance between two geographical coordinates using the Haversine formula.
    - Useful for determining the distance between points on the Earth's surface.
3. **covers(point, target, distance)**
    - Checks if a target point is within a specified distance from a given point.
    - Uses the Haversine distance for this calculation.
4. **is_within_elevation_range(center, terrain_data, bounds)**
    - Ensures that a center point's elevation is within a valid range based on terrain data.
    - Handles boundary conditions and elevation constraints.
5. **fitness(particle, data, distance, terrain_data, bounds)**
    - Evaluates the fitness of a particle based on coverage of target points.

- Returns the total coverage and set of covered points.

6. **pso(data, bounds, distance, num_particles, num_iterations, num_centers, terrain_data, w=0.5, c1=1.5, c2=1.5, threshold=1.0)**
   - Implements the PSO algorithm to find optimal machine point placements.
   - Handles particle initialization, velocity and position updates, fitness evaluation, and convergence criteria.

# 5. Modelling and Simulation

## 5.1 Data Preparation

The PSO algorithm requires several inputs:

- **Target Points**: Extracted from a CSV file containing latitude and longitude coordinates.
- **Geographical Boundaries**: Obtained from a shapefile of administrative boundaries.
- **Terrain Data**: Loaded from a TIFF file representing elevation data.

## 5.2 PSO Execution

The simulation involves:

1. **Initialization**: Particles are randomly placed within the geographical bounds. Initial velocities are set, and particles are adjusted to meet elevation constraints.
2. **Optimization Process**: The algorithm iterates to update particle positions and velocities, evaluates fitness, and updates personal and global bests.
3. **Stopping Criteria**: The process continues until either all target points are covered or a stopping criterion based on fitness variance is met.

## 5.3 Visualization

The results are visualized using `matplotlib`:

- **Terrain Data Visualization**: Displays elevation data with a color map.
- **Particle Movement Animation**: Shows the movement of particles and their convergence towards optimal locations.
- **Coverage Visualization**: Highlights the final placement of machine points and their coverage.

# 6. Objective

The primary objective of this study is to develop an optimized approach for the placement of machine points, aimed at maximizing coverage of target points within a defined area. The optimization process is guided by three main considerations:

## 6.1 Coverage Optimization

**Coverage optimization** refers to the strategic placement of machine points to ensure that they cover all target points effectively. This involves several key aspects:

1. **Definition of Coverage Area**:
    - **Coverage Radius**: Each machine point has a specified radius within which it can cover target points. This radius is defined based on operational requirements and constraints.
    - **Target Points**: These are specific locations that need to be covered by the machine points. The goal is to ensure that every target point falls within the coverage radius of at least one machine point.
2. **Coverage Calculation**:
    - **Distance Measurement**: The distance between each machine point and target point is calculated using geographical distance metrics, such as the Haversine formula. This formula accounts for the curvature of the Earth and provides accurate distance measurements.
    - **Coverage Check**: For each machine point, a check is performed to determine whether it covers a target point based on the defined coverage radius. The coverage is assessed to ensure that all target points are included within the machine points' coverage areas.
3. **Optimization Strategy**:
    - **Objective Function**: The objective function for the optimization problem is designed to maximize the number of covered target points. This function is used to evaluate the performance of different machine point placements.
    - **Fitness Evaluation**: The fitness of each solution (placement of machine points) is evaluated based on how effectively it covers the target points. Solutions that cover a higher number of target points are considered better.

### 6.2 Elevation Constraints

**Elevation constraints** involve ensuring that machine points are placed in locations where the elevation is within acceptable limits. This consideration is crucial for several reasons:

1. **Elevation Data**:
    - **Terrain Analysis**: Terrain elevation data is obtained from sources such as Digital Elevation Models (DEMs) or raster files. This data provides information about the elevation at various geographic locations.
    - **Elevation Range**: A specific range of elevation values is defined based on operational constraints and environmental considerations. Machine points must be placed in areas where the elevation falls within this range.
2. **Elevation Constraints Application**:
    - **Validity Check**: For each potential machine point location, a check is performed to ensure that the elevation is within the acceptable range. This helps avoid placing machine points in areas with extreme elevations, which may be impractical or infeasible.
    - **Constraint Enforcement**: During the optimization process, the elevation constraints are enforced to ensure that machine points are only placed in valid locations. This may involve adjusting the positions of machine points that fall outside the acceptable elevation range.
3. **Impact on Coverage**:
    - **Trade-offs**: Balancing elevation constraints with coverage optimization can involve trade-offs. Machine points may need to be placed in suboptimal locations to meet elevation requirements. The optimization algorithm must consider these trade-offs to achieve the best overall solution.

### 6.3 Boundary Conditions

**Boundary conditions** ensure that machine points are placed within specified geographical boundaries. This is essential for several reasons:

1. **Geographical Boundaries**:
    - **Boundary Definition**: The geographical boundaries are defined based on the area of interest, which may be determined by

administrative boundaries, operational zones, or other geographic constraints.
    - **Shapefiles**: The boundaries are often represented using shapefiles or other geospatial data formats that define the limits of the study area.
2. **Boundary Constraints Application**:
    - **Boundary Checking**: During the optimization process, machine points are checked to ensure that they fall within the defined boundaries. Points that fall outside the boundaries are adjusted or discarded.
    - **Constraint Enforcement**: The optimization algorithm must respect these boundaries, ensuring that all machine points are placed within the valid area. This is crucial for practical implementation and compliance with geographical constraints.
3. **Impact on Optimization**:
    - **Feasibility**: Enforcing boundary conditions ensures that the final machine point placements are feasible and applicable within the real-world geographical constraints.
    - **Optimization Adjustment**: The boundary constraints may affect the optimization process, requiring adjustments to the algorithm or solution strategies to accommodate these constraints.

### 6.4 Integration of Objectives

The integration of coverage optimization, elevation constraints, and boundary conditions involves a holistic approach to machine point placement. The optimization process aims to balance these objectives to achieve the best overall solution:

1. **Multi-Objective Optimization**:
    - **Combined Objectives**: The optimization problem is formulated to consider coverage, elevation, and boundary constraints simultaneously. This ensures that the solution is optimal in terms of coverage while adhering to practical constraints.
    - **Algorithm Adaptation**: The PSO algorithm is adapted to handle multiple objectives by incorporating constraints into the fitness function and optimization process.
2. **Trade-off Analysis**:
    - **Balancing Act**: Achieving the best coverage while respecting

elevation and boundary constraints often involves trade-offs. The optimization process must balance these factors to find a solution that meets all requirements effectively.

3. **Evaluation and Validation**:
    - **Performance Metrics**: The performance of the optimization solution is evaluated based on coverage metrics, constraint satisfaction, and practical feasibility. Validation ensures that the final solution meets all objectives and constraints.

# 7. Results

## 7.1 Optimization Performance

The PSO algorithm successfully identified optimal machine point placements, with coverage exceeding the specified distance for all target points. The final placements were visualized through animations showing the convergence of particles towards the optimal solution.

## 7.2 Visualization and Animation

- **Initial Particles**: Early iterations showed particles exploring the search space.
- **Convergence**: As iterations progressed, particles converged towards optimal locations.
- **Final Results**: The final animation highlighted the best machine point placements and their coverage areas.

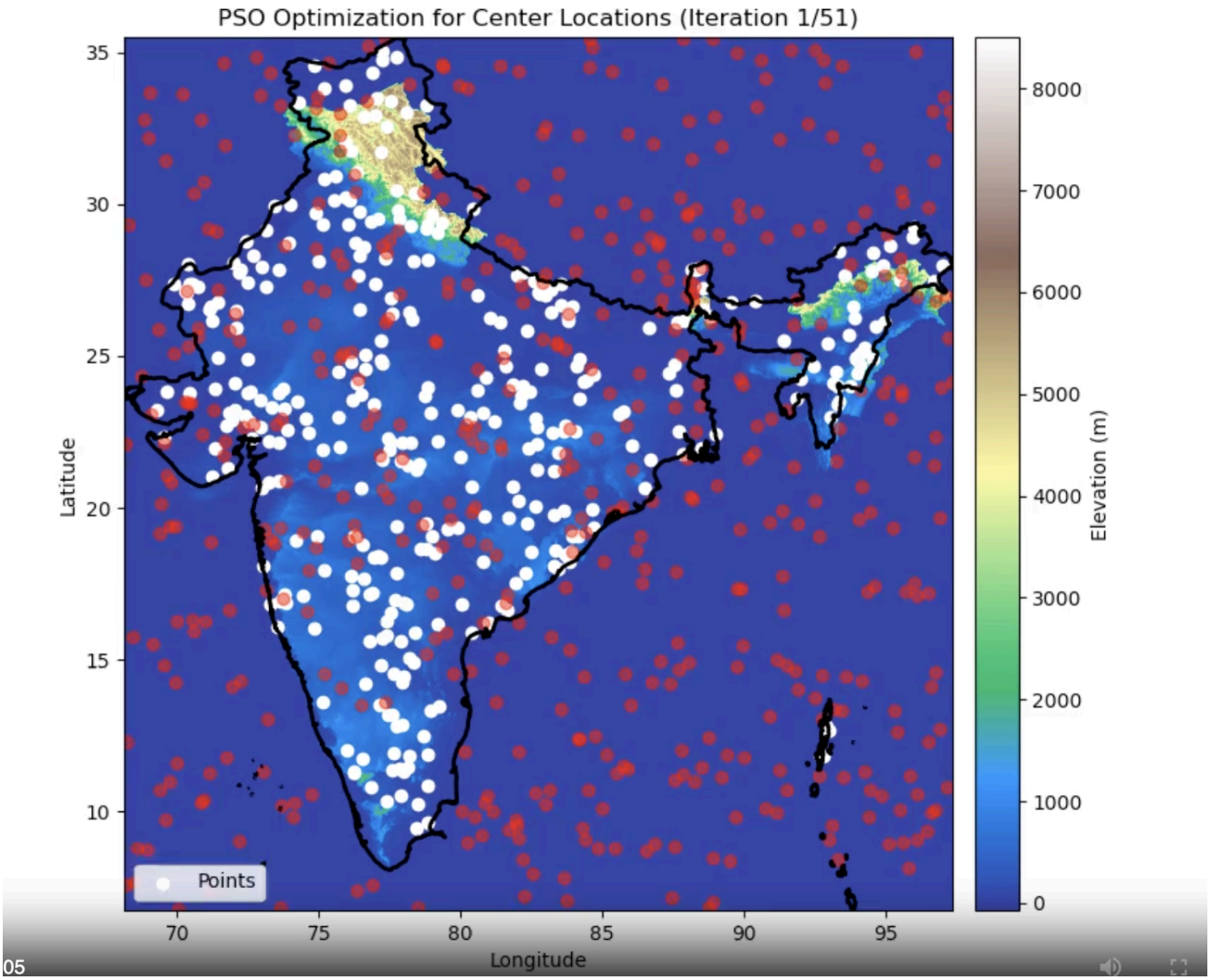
Tested the code for the number of centers ranging from 0 to 10 and checked which configuration covers the maximum possible points.

The animation can be viewed here:

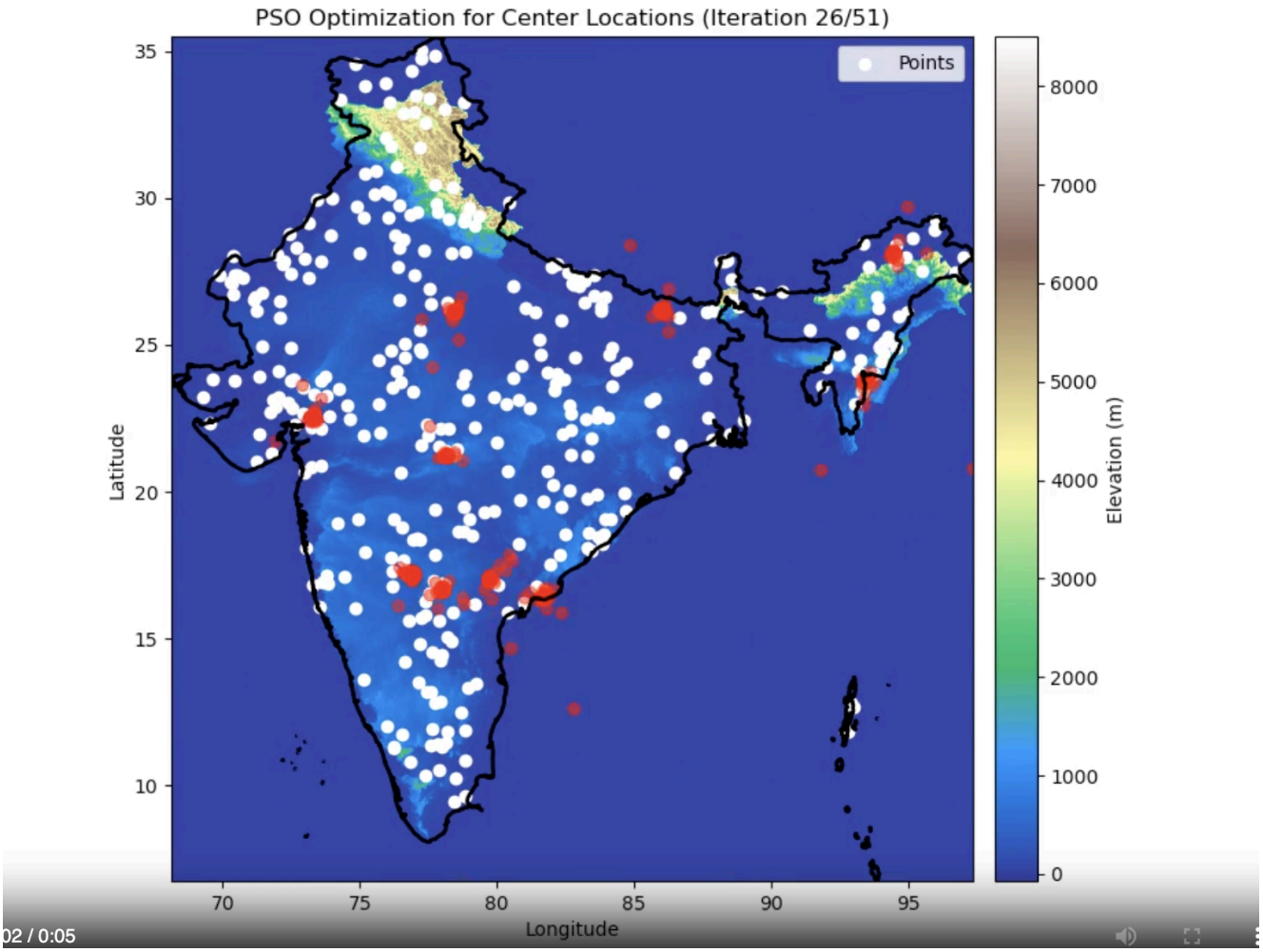
https://github.com/ParaDhim/PSO/blob/main/pso_optimization_india_R.mp4

Snippet of the running code:

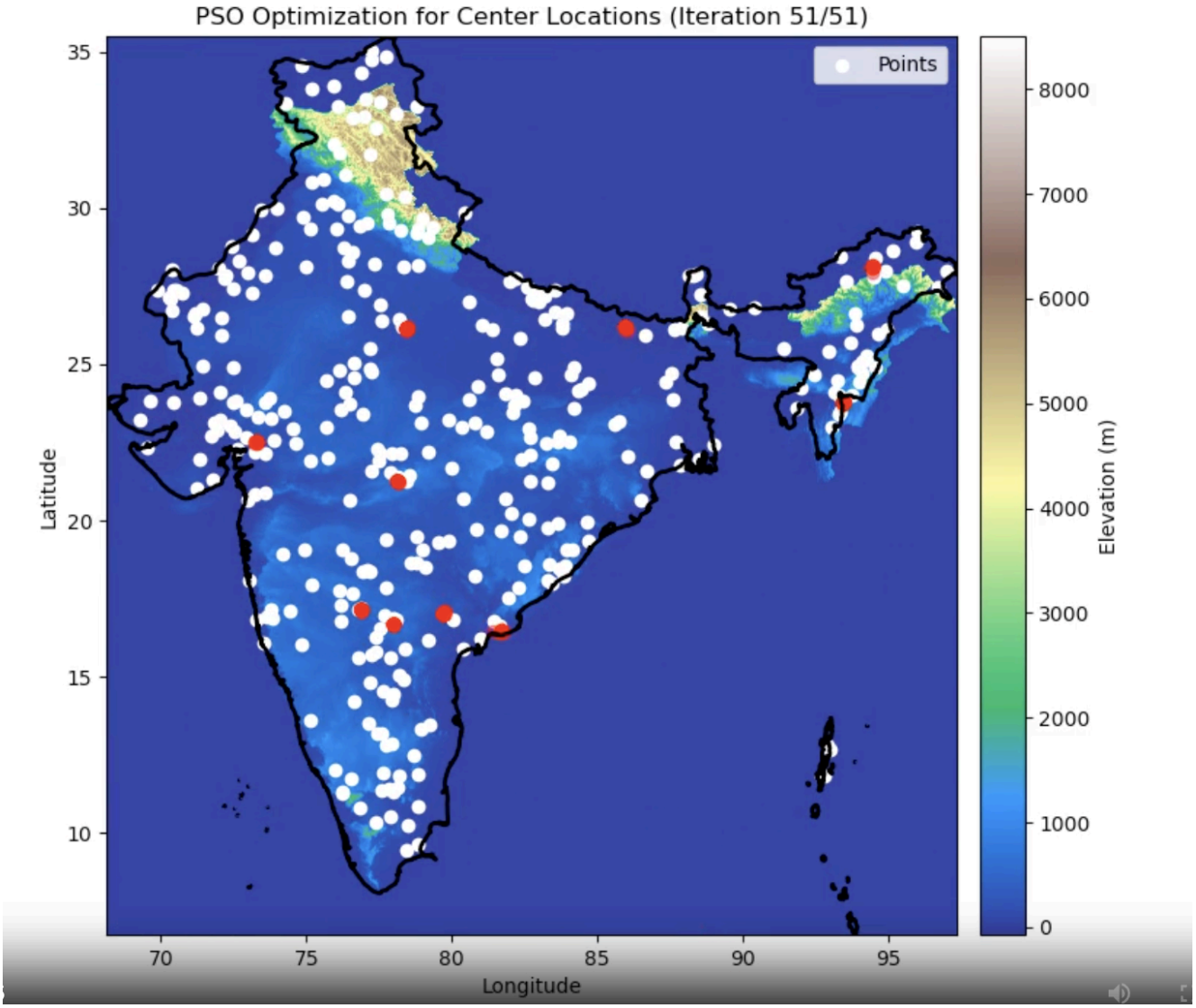
Start:


PSO Optimization for Center Locations (Iteration 1/51)
Latitude
Longitude
Elevation (m)
Points
35
30
25
20
15
10
70
75
80
85
90
95
8000
7000
6000
5000
4000
3000
2000
1000
0
05

Middle Way :


PSO Optimization for Center Locations (Iteration 26/51)
Points
Latitude
Longitude
Elevation (m)
35
30
25
20
15
10
70
75
80
85
90
95
8000
7000
6000
5000
4000
3000
2000
1000
0
02 / 0:05

End :




### 7.3 Fitness Evaluation

The fitness values of particles improved over iterations, demonstrating the effectiveness of the PSO algorithm in optimizing machine placements.

# 8. Conclusion

The Particle Swarm Optimization (PSO) algorithm has demonstrated its effectiveness in optimizing machine point placement to achieve maximum coverage of target points. Through iterative adjustments to particle positions and velocities, the algorithm successfully identified configurations where machine points were strategically placed to ensure comprehensive coverage of all target points within a specified distance. This capability highlights PSO's strength in exploring and exploiting complex search spaces, ultimately providing a solution that maximized the reach and efficiency of the machine points.

In addition to optimizing coverage, the PSO algorithm adeptly handled elevation constraints and geographical boundary conditions. By ensuring that machine points were placed within valid elevation ranges, the algorithm maintained the practical viability of the solution. Furthermore, the adherence to geographical boundaries ensured that all placements were relevant to the specified area, thereby aligning the results with real-world spatial constraints. This dual focus on coverage and constraint adherence underscores the algorithm's capacity to deliver practical and effective solutions for real-world applications.

The visualizations created throughout the study provided valuable insights into both the optimization process and the final results. These visual tools illustrated the movement of particles and the convergence towards an optimal placement of machine points, offering a clear representation of how the PSO algorithm achieved its objectives. Overall, the successful application of PSO in this study not only demonstrates its effectiveness in solving complex optimization problems but also sets the stage for future research and applications in similar domains.

# 9. Future Scope

## 9.1 Algorithm Improvements

Future work can focus on:

- **Enhanced Fitness Functions**: Incorporating additional factors such as cost and operational constraints.
- **Hybrid Approaches**: Combining PSO with other optimization techniques for improved performance.
- **Real-Time Adaptation**: Adapting the algorithm for dynamic environments where target points and constraints change over time.

## 9.2 Application Areas

PSO can be extended to other applications, such as:

- **Network Design**: Optimizing network infrastructure for communication and coverage.
- **Resource Management**: Efficient allocation of resources in various industries.
- **Environmental Monitoring**: Optimizing sensor placement for environmental data collection.

# 10. References


**Papers and Articles:**

1. Bisht, S., Taneja, S. B., Jindal, V., & Bedi, P. (Year). **APSO Based Automated Planning in Constructive Simulation.** *DRDO-Institute for Systems Studies & Analyses*, Delhi, India; *Keshav Mahavidyalaya, University of Delhi*, Delhi, India; *Department of Computer Science, University of Delhi*

**Tools and Libraries:**

- numpy: [https://numpy.org/]
- matplotlib: [https://matplotlib.org/]
- rasterio: [https://pypi.org/project/rasterio/]
- pandas: [https://pandas.pydata.org/]
- geopandas: [https://geopandas.org/en/stable/getting_started/introduction.html]
- shapely: [https://pypi.org/project/shapely/]
- scipy: [https://scipy.org/]

**Datasets:**

- Points Data: [https://figshare.com/articles/dataset/India_s_elevation_profile_in_a_GeoTIFF_file/12479306]
- Terrain Data: [Link or citation to TIFF file]
- Boundary Data : [https://www.naturalearthdata.com/downloads/10m-cultural-vectors/]

**Links:**

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

# 11. Appendix

## 11.1 Code Listings

- Particle Swarm Optimization Implementation:

```
def is_within_elevation_range(center, terrain_data, bounds):
    lat_index = int((center[0] - bounds[0]) / terrain_data.res[0])
    lon_index = int((center[1] - bounds[2]) / terrain_data.res[1])

    # Check if indices are within bounds
     if lat_index < 0 or lat_index >= terrain_data.shape[0] or lon_index < 0 or lon_index >=
terrain_data.shape[1]:
        return False

    elevation = terrain_data.read(1)[lat_index, lon_index]
    return 0 <= elevation <= 2000

# Fitness function
def fitness(particle, data, distance, terrain_data, bounds):
    total_coverage = 0
    covered_points = set()
    for point in data:
        for center in particle:
            if covers(center, point, distance):
                total_coverage += 1
                covered_points.add(tuple(point))
                break
    return total_coverage, covered_points
```

```
def pso(data, bounds, distance, num_particles, num_iterations, num_centers, terrain_data, w=0.5,
c1=1.5, c2=1.5, threshold=1.0):
    # Precompute bounds and elevation check parameters
    bounds_low = np.array([bounds[0], bounds[2]])
    bounds_high = np.array([bounds[1], bounds[3]])
    print("part init")
    def initialize_particles():
            particles = np.random.uniform(low=bounds_low, high=bounds_high, size=(num_particles,
num_centers, 2))
        velocities = np.random.uniform(low=-1, high=1, size=(num_particles, num_centers, 2))
        for i in range(num_particles):
            for j in range(num_centers):
                while not is_within_elevation_range(particles[i, j], terrain_data, bounds):
                    particles[i, j] = np.random.uniform(low=bounds_low, high=bounds_high, size=2)
        return particles, velocities

    # Initialize particles
    particles, velocities = initialize_particles()

    # Initialize personal bests and global best
    p_best = particles.copy()
    p_best_fitness = np.zeros(num_particles)
    for i in range(num_particles):
        p_best_fitness[i], _ = fitness(p_best[i], data, distance, terrain_data, bounds)
```

```
    g_best_index = np.argmax(p_best_fitness)
    g_best = p_best[g_best_index].copy()
    g_best_fitness = p_best_fitness[g_best_index]
    g_best_covered = set()
    print("checked")
    # Initialize particle history and fitness history
    particle_history = [particles.copy()]
    fitness_history = []
    centers_list = []
    points_covered_list = []

    # PSO iterations
    for iteration in range(num_iterations):
        all_covered = False
        current_fitness_values = []

        for i in range(num_particles):
            # Update velocity
            velocities[i] = (w * velocities[i]
                             + c1 * np.random.rand() * (p_best[i] - particles[i])
                             + c2 * np.random.rand() * (g_best - particles[i]))
            # Update particle position
            particles[i] += velocities[i]
            # Boundary conditions
            particles[i] = np.clip(particles[i], bounds_low, bounds_high)

            # Ensure particles are within the elevation range
            for j in range(num_centers):
                if not is_within_elevation_range(particles[i, j], terrain_data, bounds):
                    particles[i, j] = np.random.uniform(low=bounds_low, high=bounds_high, size=2)

            # Update personal best
             current_fitness, current_covered = fitness(particles[i], data, distance, terrain_data,
bounds)
            current_fitness_values.append(current_fitness)
            if current_fitness > p_best_fitness[i]:
                p_best[i] = particles[i].copy()
                p_best_fitness[i] = current_fitness

                # Update global best
                if current_fitness > g_best_fitness:
                    g_best = particles[i].copy()
                    g_best_fitness = current_fitness
                    g_best_covered = current_covered.copy()

                    # Check if all points are covered
                    if len(g_best_covered) == len(data):
                        all_covered = True

        # Store particle positions for animation
        particle_history.append(particles.copy())
        fitness_history.append(current_fitness_values)

        # Terminate early if all points are covered
        if all_covered:
            break

        # Early stopping criterion based on variance of fitness values for the last 10 iterations
        if iteration >= 9 and iteration % 10 == 9:
```

```
                last_10_fitness_values = [fitness_history[-j][i] for j in range(1, 11) for i in
range(num_particles)]
            fitness_variance = np.var(last_10_fitness_values)
            print(fitness_variance)
            if fitness_variance < threshold:
                    print(f"Early stopping at iteration {iteration} due to low variance in fitness
values.")
                break

    # Return the best particles, best fitness, and particle history
          return    g_best,    g_best_fitness,    g_best_covered,    particle_history,    centers_list,
points_covered_list
```

- **Visualization Code**: Code for generating plots and animations.

```
import numpy as np
import matplotlib.pyplot as plt
import rasterio
from matplotlib.animation import FuncAnimation
import pickle

# Define the file path to load the parameters
param_file_path = 'pso_params_India.pkl'

# Load the parametric information
with open(param_file_path, 'rb') as f:
    params = pickle.load(f)

particle_history = params['particle_history']
bounds = params['bounds']
data = params['data']
best_points = params['best_points']
distance = params['distance']

# Load the terrain data
terrain_file = "/Users/parasdhiman/Desktop/DRDO/india_clipped.tif"
terrain_data = rasterio.open(terrain_file)

# Create animation
fig, ax = plt.subplots(figsize=(10, 8))

# Show the terrain data with custom elevation-based colormap
terrain_image    =    ax.imshow(terrain_data.read(1),    extent=[bounds[2],    bounds[3],    bounds[0],
bounds[1]], cmap='terrain', origin='upper')

# Add color bar for elevation
cbar = plt.colorbar(terrain_image, ax=ax, orientation='vertical', pad=0.02)
cbar.set_label('Elevation (m)')

def update(frame):
    ax.clear()
        ax.imshow(terrain_data.read(1),    extent=[bounds[2],    bounds[3],    bounds[0],    bounds[1]],
cmap='terrain', origin='upper')
    ax.scatter(data[:, 1], data[:, 0], c='white', label='Points')
```

```
    particles = particle_history[frame]
    for particle in particles:
        ax.scatter(particle[:, 1], particle[:, 0], c='blue', alpha=0.5)

    if frame == len(particle_history) - 1:
        for center in best_points:
            ax.scatter(center[1], center[0], c='red', marker='x', s=100, label='Center')
            circle = plt.Circle((center[1], center[0]), distance / 111, color='blue', fill=False)
            ax.add_artist(circle)

    ax.set_xlim(bounds[2], bounds[3])
    ax.set_ylim(bounds[0], bounds[1])
    ax.set_xlabel('Longitude')
    ax.set_ylabel('Latitude')
          ax.set_title(f'PSO   Optimization   for   Center   Locations   (Iteration   {frame   +
1}/{len(particle_history)})')

    cbar.set_label('Elevation (m)')

    ax.legend()

# Animate the PSO process
ani = FuncAnimation(fig, update, frames=len(particle_history), interval=200)
ani.save('pso_optimization_india.mp4', writer='ffmpeg', fps=10)
```

- **Data Generation**(Code for Generating the Dataset):

```
import shapefile
from shapely.geometry import Point, shape
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import random

# Define the boundaries of India
min_lat, max_lat = 8.4, 37.6
min_lon, max_lon = 68.7, 97.4

# Load the shapefile for India's boundaries
sf = shapefile.Reader("/Users/parasdhiman/Desktop/DRDO/ne_10m_admin_0_countries
2/ne_10m_admin_0_countries.shp")

# Print fields to find the correct field for country names
print("Shapefile Fields:", sf.fields)

# Identify the field containing country names
country_name_field = None
for field in sf.fields:
    if 'name' in field[0].lower():
        country_name_field = field[0]
        break

if country_name_field is None:
    raise ValueError("Field containing country names not found in the shapefile.")

# Extract the shape of India
```

```
india_shape = None
for shape_rec in sf.shapeRecords():
    if 'India' in shape_rec.record[country_name_field]:
        india_shape = shape_rec.shape
        break

if india_shape is None:
    raise ValueError("India's shape not found in the shapefile.")

def point_in_india(lat, lon):
    """Check if a given point is inside India using the shapefile."""
    point = Point(lon, lat)
    india_polygon = shape(india_shape.__geo_interface__)
    return india_polygon.contains(point)


# Define the size of each grid cell
grid_size_lat = (max_lat - min_lat) / 5
grid_size_lon = (max_lon - min_lon) / 5

# Create a map centered around India
fig, ax = plt.subplots(figsize=(10, 10))
m = Basemap(projection='merc', llcrnrlat=min_lat, urcrnrlat=max_lat,
            llcrnrlon=min_lon, urcrnrlon=max_lon, resolution='i', ax=ax)

# Draw map boundaries and coastlines
m.drawcoastlines()
m.drawcountries()
m.drawmapboundary()

# Calculate the number of grids
lat_grids = int((max_lat - min_lat) / grid_size_lat)
lon_grids = int((max_lon - min_lon) / grid_size_lon)

# Draw the grid
parallels = [min_lat + i * grid_size_lat for i in range(lat_grids + 1)]
meridians = [min_lon + j * grid_size_lon for j in range(lon_grids + 1)]

m.drawparallels(parallels, labels=[1,0,0,0], color='grey', dashes=[1, 1], linewidth=0.5)
m.drawmeridians(meridians, labels=[0,0,0,1], color='grey', dashes=[1, 1], linewidth=0.5)

# Generate random points within each grid
random_points_inside = []
random_points_outside = []

# Store grid cells that intersect with India
intersecting_grids = []

for i in range(lat_grids):
    for j in range(lon_grids):
        grid_min_lat = min_lat + i * grid_size_lat
        grid_max_lat = grid_min_lat + grid_size_lat
        grid_min_lon = min_lon + j * grid_size_lon
        grid_max_lon = grid_min_lon + grid_size_lon

        # Generate points using Gaussian distribution with increased variance
        lat_mean = (grid_max_lat + grid_min_lat) / 2
        lon_mean = (grid_max_lon + grid_min_lon) / 2
        lat_std = (grid_max_lat - grid_min_lat) / 3  # Increased variance
```

```
        lon_std = (grid_max_lon - grid_min_lon) / 3  # Increased variance

        # Check if the grid cell is entirely within India
        points_to_generate = 50 if point_in_india(lat_mean, lon_mean) else 100
        grid_points_inside = []
        grid_points_outside = []

        for _ in range(points_to_generate):
            rand_lat = random.gauss(lat_mean, lat_std)
            rand_lon = random.gauss(lon_mean, lon_std)

            # Ensure the points are within the grid boundaries
            if grid_min_lat <= rand_lat <= grid_max_lat and grid_min_lon <= rand_lon <=
grid_max_lon:
                if point_in_india(rand_lat, rand_lon):
                    random_points_inside.append((rand_lat, rand_lon))
                    grid_points_inside.append((rand_lat, rand_lon))
                else:
                    random_points_outside.append((rand_lat, rand_lon))
                    grid_points_outside.append((rand_lat, rand_lon))

        if grid_points_inside or grid_points_outside:
            intersecting_grids.append((i, j, grid_points_inside, grid_points_outside))

# Plot random points inside India on the map (in red)
for point in random_points_inside:
    x, y = m(point[1], point[0])
    m.plot(x, y, 'ro', markersize=2)  # Red color

# Plot random points outside India on the map (in blue)
for point in random_points_outside:
    x, y = m(point[1], point[0])
    m.plot(x, y, 'bo', markersize=2)  # Blue color

plt.title('Grid with Random Points over India')
plt.show()

# Plot distribution for each grid that covers India
fig, axes = plt.subplots(len(intersecting_grids), 1, figsize=(10, len(intersecting_grids) * 5))

for idx, (i, j, grid_points_inside, grid_points_outside) in enumerate(intersecting_grids):
    ax = axes[idx] if len(intersecting_grids) > 1 else axes
    grid_min_lat = min_lat + i * grid_size_lat
    grid_max_lat = grid_min_lat + grid_size_lat
    grid_min_lon = min_lon + j * grid_size_lon
    grid_max_lon = grid_min_lon + grid_size_lon

    m = Basemap(projection='merc', llcrnrlat=grid_min_lat, urcrnrlat=grid_max_lat,
                llcrnrlon=grid_min_lon, urcrnrlon=grid_max_lon, resolution='i', ax=ax)

    m.drawcoastlines()
    m.drawcountries()
    m.drawmapboundary()

    for point in grid_points_inside:
        x, y = m(point[1], point[0])
        m.plot(x, y, 'ro', markersize=2)  # Red color

    for point in grid_points_outside:
```

```
        x, y = m(point[1], point[0])
        m.plot(x, y, 'bo', markersize=2)  # Blue color

    ax.set_title(f'Grid ({i}, {j}) with Random Points')

plt.tight_layout()
plt.show()
```

## 11.2 Data Files

- **CSV Data File**: Contains target points for optimization.
- **Shapefile**: Contains geographical boundaries for the study area.
- **TIFF File**: Contains terrain elevation data.

## 11.2 Additional Visualizations

- **Coverage Maps:** Detailed maps showing coverage results.
- **Particle Movement Animations:** Additional animations of particle movements during the optimization process.

## 11.3 Glossary

- **Particle Swarm Optimization (PSO):** An optimization algorithm inspired by social behavior of swarms.
- **Haversine Formula:** A formula used to calculate the distance between two points on the surface of a sphere.

```
def haversine(coord1, coord2):
    lat1, lon1 = np.radians(coord1)
    lat2, lon2 = np.radians(coord2)

    dlat = lat2 - lat1
    dlon = lon2 - lon1

    a = np.sin(dlat / 2) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon / 2) ** 2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))

    # Earth radius in kilometers
    r = 6371.0
    return r * c
```