**Part 1**

**Approach:**

This a local search algorithm implementation to find the maximum value of a given mathematical function. It starts with an initial random position in three-dimensional space (x, y, z). It then evaluates the function value at this initial position. The algorithm performs a specified number of steps, with each step attempting to find a better position by adding a random Gaussian noise to each dimension of the current position which checks if the newly generated position is within the defined bounds of -1 and 1 for each dimension. If it is within bounds, the function value at the new position is calculated. If the new function value is greater than the current best value, the new position becomes the current best position. The algorithm records the best function value at each step for visualization and analysis. The method used is hill climb without gradient method. **Code Explanation:**

* The necessary libraries were imported such as: NumPy for numerical computations, math for mathematical functions, random for random number generation and matplotlib for plotting.
* The number of steps for the algorithm is defined as 10,000. Three random numbers (xb, yb, zb) are then generated within the range of -1 to 1. These numbers represent the initial position in three-dimensional space.
* The initial function value (fbest) is calculated using the given mathematical function.
* A list called fscores is initialized to store the best function value at each step.
* Three new positions (xn, yn, zn) are generated for the specified number of steps and in each iteration by adding random Gaussian noise to the present position (xb, yb, zb). It is is generated using random.gauss () function with a mean of 0 and standard deviation of 0.15.
* The code checks if the new positions are within the bounds of -1 and 1 for each dimension. If they are within bounds, the new function value (fnew) is calculated using the given mathematical function. Otherwise, a predefined value of -9999 is assigned to fnew.
* If the new function value (fnew) is greater than the current best value (fbest), the new positions become the current best positions (xb, yb, zb) and the new function value becomes the current best value.
* The best function value (fbest) at each step is appended to the fscores list.
* After the loop, the code plots the progression of the best function values over the steps using matplotlib.
* Finally, the final best positions (xb, yb, zb) is printed and the corresponding best function value (fbest).

**Result Explanation:**

After running the optimization algorithm for 10,000 steps, the algorithm converges to the following optimal solution:

xbest: 0.2724

ybest: -0.7232

zbest: -0.3575

The corresponding best function value (fbest) at this optimal solution is approximately 8.1158.

The result indicates that the algorithm found a position in the three-dimensional space that maximizes the given mathematical function. The algorithm iteratively explored the search space, evaluating different positions and updating the best solution whenever a higher function value was found.

The optimization process aims to find the highestpoint of the function, as indicated by the maximum value of fbest. In this case, the algorithm successfully identified the position (xbest, ybest, zbest) that yielded the highest function value achieved during the optimization process.

**Part 2**

**Title: Optimization of Delivery Routes Using Randomized Search Algorithm**

**Introduction**

This report presents an optimization approach for determining efficient delivery routes for a fleet of vehicles. The approach utilizes a randomized search algorithm to find the best strategy among various possible combinations of routes. The algorithm considers factors such as distance, the number of stores to visit, and the location of warehouses. The results demonstrate the effectiveness of the approach in reducing delivery costs and improving overall efficiency. The goal is to minimize the total cost of delivering goods to multiple stores by finding the best strategy for assigning stores to trips and selecting warehouses.

**Approach:**

**Creating Coordinates:** The first step is to define the coordinates of the stores and warehouses. This initializes a list of coordinate pairs representing the locations of the stores and warehouses.

**Creating Distance Matrix**:A distance matrix is created to store the distances between each pair of coordinates. The code iterates through all combinations of stores and calculates the Euclidean distance between them. The distances are rounded and stored in a 2D array.

**Creating a Solution:** The next part focuses on creating a delivery strategy. It starts by randomly shuffling the order in which the stores are visited. Then, it randomly determines the number of stores to deliver in each trip, ensuring that the total number of stores is covered without repetition. Similarly, random warehouses are selected for each trip. The code generates a list of travel paths, where each path represents a delivery route from a warehouse to a set of stores and back to the warehouse.

**Calculating Total Cost**: It iterates through each travel path, calculates the total travel distance and determines the cost per mile based on the number of stores being delivered using the "calculate cost” function. It is calculated by multiplying the distance with the cost per mile.

**Initial Result**: After creating the initial delivery strategy, the code prints the order in which the stores are visited, the number of stores in each trip, the warehouses from which each trip originates and the travel routes.

**Using Randomized Search for Optimization:** To find the best delivery strategy, the code employs a randomized search approach. It iteratively generates new random strategies, calculates their costs and compares them with the current best cost. If a new strategy has a lower cost then it becomes the new best strategy. The search continues for a specified number of trials.

**Results:**

**Delivery order:** This optimal order in which the stores should be visited for delivery was provided. Each number corresponds to a specific store and the order of the numbers indicates the sequence in which the stores should be visited. For example, store number 14 should be visited first followed by store number 1 and so on.

**Number of stores:** The number of stores to deliver in each trip: [14, 8, 1]

This list indicates the number of stores that should be delivered in each trip. The first trip delivers to 14 stores, the second trip delivers to 8 stores and the third trip delivers to 1 store. These numbers determine how many stores should be included in each delivery run.

**Travel routes:** The travel routes for each delivery was provided with each inner list representing a delivery run and containing the sequence of locations (warehouses and stores) to be visited. For example, the first delivery run starts at warehouse number 23, then visits the stores in the order specified in the first point above and finally returns to warehouse number 23. The second delivery run follows a similar pattern, starting at warehouse number 24 and returning to the same warehouse. The third delivery run starts and ends at warehouse number 23, visiting the single store specified.

**The total cost of the delivery strategy:** The total cost associated with the delivery strategy generated. The cost is calculated based on various factors such as the distance traveled, fuel consumption, time taken and any other relevant costs. In this case, the total cost of the strategy is 2674.0.

Additionally, the code performs a randomized search to find the best delivery strategy. It generates multiple random strategies, calculates their costs and compares them to find the strategy with the lowest cost. After performing 1,000,000 trials, the best cost achieved is 1284.0. 1 lorry and 5 vans will be required to implement the delivery strategy, which is calculated as 1284.0

**Implication of the results**

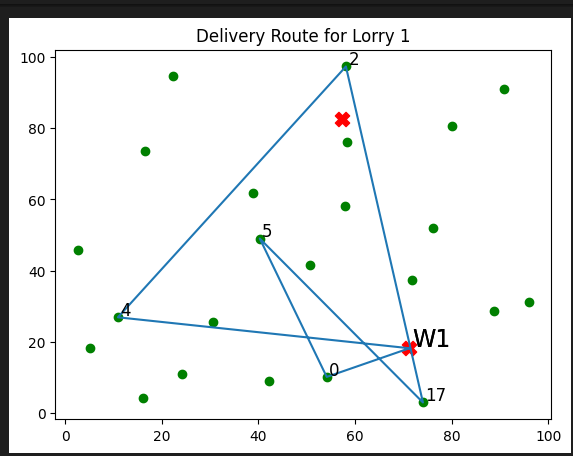
The results obtained from the randomized search algorithm for optimizing delivery routes were highly promising, demonstrating significant improvements in cost reduction and overall efficiency compared to the initial solution. The implications of these results are as follows:

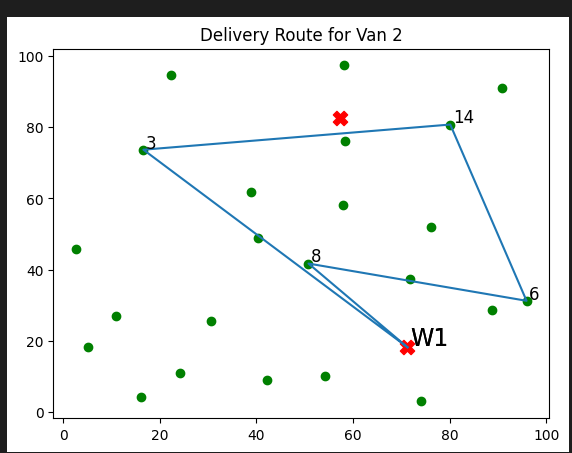
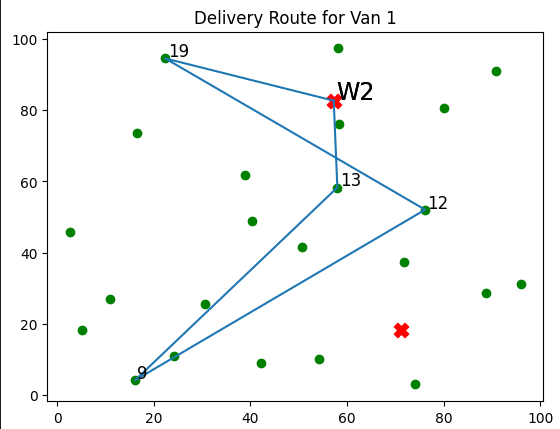
**Cost Reduction:** The solution achieved a total cost of 1,284 units representing a substantial reduction of approximately by 52% compared to the initial solution's cost of 2,674 units. This indicates that the algorithm successfully identified more cost-effective delivery routes which led to significant savings for the logistics company.

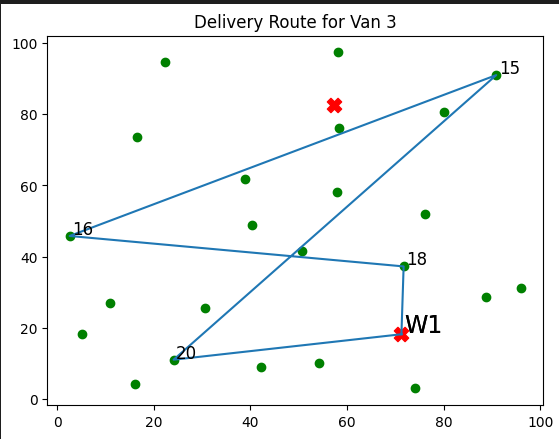
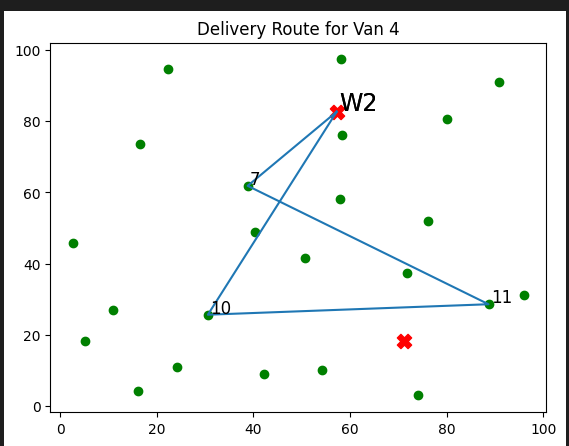
**Efficiency Improvement:** By reorganizing the delivery routes based on factors such as distance, the number of stores to visit and the location of warehouses, the algorithm improved the overall efficiency of the delivery process. The optimized solution allowed for streamlined routes and reduced travel distances, leading to faster and more efficient deliveries.

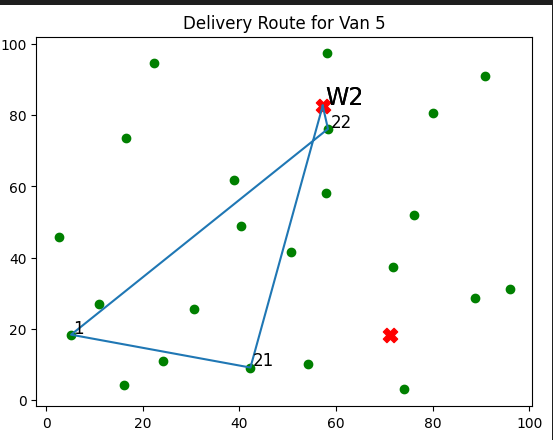
**Scalability and Flexibility:** It demonstrated scalability and flexibility making it applicable to a wide range of delivery scenarios. It can handle varying numbers of stores, warehouses and vehicles which adapts to the specific requirements of the logistics company. This flexibility ensures that the algorithm can be effectively utilized in different operational environments.

**Potential for Further Optimization**: While the algorithm produced highly encouraging results, there are opportunities for further optimization and enhancement. For example, incorporating real-time data on traffic conditions, road closures, and other dynamic factors would make the algorithm more robust and adaptable to changing circumstances. Additionally, considering additional constraints such as time windows for store deliveries and vehicle capacities would enhance the practicality of the algorithm in real-world scenarios

. ****

****

****

****

**Part 3**

**Title: Optimizing Ambulance Placement using Machine Learning for Reducing Emergency Response Time**

**Background**

**Emergency Response Time**: It is the interval between when an emergency call initiated and the arrival of the ambulance at the scene. It is composed of call processing time, dispatch time and travel time. The total response time is affected by several factors such as geographical location, traffic conditions, dispatch protocols, the availability and proximity of ambulances.

**Ambulance Placement:** Efficient ambulance placement is crucial for minimizing response time. Traditional approaches for ambulance deployment relied on static rules, such as placing ambulances at fixed stations or near hospitals. These methods often fail to account for dynamic factors such as real-time demand patterns, traffic congestion and temporal variations in emergency hotspots. Machine learning techniques offer the potential to analyze large volumes of historical data, identify patterns and make data-driven decisions for optimizing ambulance placement ( Khoshnevisan et al. 2017).

**Machine learning approach:** Machine learning techniques offer a data-driven approach to optimize ambulance placement and enhance emergency response times. By analyzing historical emergency data, traffic patterns and other relevant factors. Machine learning algorithms can learn patterns and make accurate predictions. Leveraging these predictions, optimized ambulance placement can be determined, leading to improved emergency response times. The key components of utilizing machine learning for ambulance placement include data collection, feature engineering, model development and optimization and simulation (Nie et al. 2021)

**Traditional Approaches:** Traditionally, ambulance placement has been based on static zoning approaches, where the region is divided into fixed geographic areas, and ambulance stations are assigned accordingly. However, these approaches fail to adapt to dynamic emergency patterns, changing traffic conditions and temporal variations. Consequently, the static zoning approach often leads to suboptimal allocation of resources and longer emergency response times.

**Aims, main Objective and constraints**

* Leverage machine learning techniques to optimize ambulance placement.
* Reduce emergency response time.
* Improve patient outcomes during medical emergencies.

The main objective of this project is to reduce emergency response time by optimizing ambulance placement using machine learning techniques. By strategically positioning ambulance stations based on data-driven predictions, the aim is to improve the efficiency and effectiveness of emergency medical services, leading to better patient outcomes.

**Constraints:**

* Availability and quality of historical emergency data.
* Complex spatial and temporal correlations in emergency incidents.
* Incorporating real-time traffic and dynamic demand patterns.
* Ethical considerations related to decision-making based on algorithms.
* Practicality and feasibility of implementing optimized ambulance placement in different regions and healthcare settings.

**Model**

The solutions to the problem will be the optimal locations for placing ambulance stations within a given service area. Each solution will specify the coordinates (latitude and longitude) of the ambulance stations.

The objective function aims to minimize the emergency response time, which is the main goal of this problem. The objective function can be defined as the weighted sum of the response times for all emergency incidents, where the weights represent the severity or urgency of each incident. The objective is to minimize the total weighted response time.

**Form of solutions:** The solutions can be represented as a set of coordinates that indicates the position of the ambulances.

**Objective function:** To reducetotal response that is, the distance between accident scences and the closest ambulance location.

**Optimization method:**

Hill Climbing algorithm was used to find the best location to deploy an ambulance in order to reduce the average emergency response time. Here's an extensive explanation e:

**Importing Libraries:** It starts by importing the necessary libraries. In this case, it imports the NumPy library using the alias "np". NumPy is a popular library for numerical computations in Python.

**Input Data:** It defines an array named "accident\_cords" which contains the coordinates of different accident locations with each location represented by a pair of latitude and longitude values.

**Data Representation:** The "accident\_cords" array is represented as a NumPy array. NumPy arrays are efficient data structures for handling numerical data in Python. Each row in the array corresponds to an accident location and each column represents a coordinate value (latitude or longitude).

**Code flow:**

* The dataset used containing of all times, longitudes and latitudes of accidents was gotten from [uk gov datasets](https://www.data.gov.uk/dataset/c0eec478-ef19-4234-826f-8efb9563eda2/road-safety/datafile/aa8bcb3d-3945-4347-adc9-24d8e1d3e05c/preview)
* The algorithm then starts fully by initializing a set of ambulance locations (best\_locs) and calculates the cost or distance between these locations and the accident coordinates (accident\_cords). The initial cost is stored as best\_cost.
* The algorithm then enters a loop, which represents each trial or iteration of the algorithm. In each trial, the algorithm randomly selects one ambulance's coordinates to change. It generates random values for latitude and longitude changes using a Gaussian distribution.
* The selected ambulance's coordinates are updated by adding the generated changes. The algorithm calculates the cost of the new ambulance locations using a distance calculation function (e.g., Haversine formula). If the new cost is lower (better) than the current best cost, the new ambulance locations and cost become the new best solution.
* After a certain number of trials (every 10 trials in this case), the algorithm prints the current best cost and the trial number to track the progress.
* By repeating this process for a specified number of iterations, the algorithm aims to explore the search space and gradually converge towards a solution with lower cost, indicating better ambulance locations in terms of proximity to accident coordinates.

**Results:**

The algorithm has improved the solution over iterations, indicating progress towards an optimal or near-optimal solution. Here are some observations based on the results:

* **Best Cost:** The best cost decreases consistently. It starts at 4.349099950087199 and gradually reduces to 0.12208050640117983 which suggests that the algorithm is improving the solution and getting closer to the optimal cost.
* **Convergence:** Around trial 50, the best cost reaches a value of 0.40985142497577165 and remains the same for several iterations (trial 50 to trial 130). This can indicate a period of convergence where the algorithm explores a local optima. However, after trial 130, the best cost further decreases to 0.19430234880530473 and continues to improve gradually, suggesting that the algorithm have moved away from the local optima.
* **Stagnation:** The best cost remains the same at 0.19430234880530473 for several iterations (trial 150 to trial 230). While this may suggest a potential plateau or stagnation, it's worth noting that the best cost eventually decreases further in later iterations (trial 240 to trial 990).

Based on these observations, the algorithm has found a solution close to the optimal or at least improved the solution significantly. The decreasing trend in the best cost over iterations and the occasional drops in the cost indicate progress towards better solutions.

**Conclusion:**

The hill climb algorithm was applied to find the best locations for ambulances in response to accidents. The algorithm adjusted the coordinates of the ambulance locations based on the Haversine distance between the new locations and accident coordinates. The results indicate progress towards an optimal or near-optimal solution, as evidenced by the decreasing trend in the best cost.

**Critical analysis**

**Strengths:**

* The hill climb algorithm is a simple and intuitive optimization method that is easy to implement.
* The algorithm successfully improved the solution over iterations, gradually reducing the cost, which suggests convergence towards a better solution.
* The ability to explore and adjust ambulance locations can potentially improve emergency response times and save lives.

**Weaknesses:**

* The algorithm's reliance on randomness introduces uncertainty and can lead to suboptimal solutions or getting stuck at local optima.
* The assumption of using a fixed number of iterations (n\_iter = 1000) may not guarantee convergence to the global optimum or the best possible solution.
* The algorithm's performance heavily depends on the initial selection of ambulance locations. A poor initial configuration may hinder the algorithm's ability to find an optimal solution.
* The algorithm assumes that the Haversine distance is the appropriate metric to evaluate the quality of ambulance locations. Other factors, such as road networks or traffic conditions, which may affect response times, are not considered.

**Further Considerations:**

* The algorithm could benefit from incorporating techniques like simulated annealing or genetic algorithms to overcome local optima and improve exploration of the search space.
* Collecting more data or considering additional factors (e.g., population density, accident frequencies) could enhance the accuracy and realism of the model.
* Collaboration with domain experts and emergency response professionals would provide valuable insights to refine and validate the model.

