VEHICLE ROUTE OPTIMIZATION FORSHIPPING PRODUCTS FROM A WAREHOUSE TO RETAIL STORES USING GENETIC ALGORITHM APPROACH.

Introduction

In the retail industry, efficient supply chain management is crucial for ensuring cost effective and timely delivery of products from warehouses to retail stores. Vehicle routing problem is a fundamental optimization challenge in logistics, aiming to determine the optimal routes for a fleet of vehicles to serve a set of customers while minimising costs such as travelling cost and adhering to constraints like vehicle capacity and customer demands.

This project focuses on optimising vehicle routes for product shipment from a central warehouse(depot) to several retail stores using Genetic Algorithm(GA). The GA leverages a population based metaheuristic approach to explorer a wide range of route configuration evolving solution through selection, crossover and mutation to find near optimal routes.

The problem involves “n” vehicles (v1, V2, V3,… Vn). Each with the capacity of “m” and there are “X” customers (C1,C2, C3,… CX) with specified demands and depot to customer distances are known, the objective is to minimise the total distance travelled by all vehicles while ensuring each customer demand is met and vehicle capacities are not exceeded, the GA implementation is based on the provided Python code, it incorporates chromosome encoding, fitness evaluation and genetic operators such as crossover and mutation, this approach in shows practical applicability in retail logistics enhancing delivery efficiency and reducing operational cost. Ten case studies varying the number of customers and vehicle capacities are conducted and the algorithm is able to find the optimal solution.

Literature Review

The Vehicle Routing Problem (VRP), first introduced by Dantzig and Ramser (1959), has remained a cornerstone in logistics and operational research due to its practical significance in transportation, distribution, and supply chain optimization. Over the decades, researchers have proposed various exact, heuristic, and metaheuristic approaches to tackle this NP-hard problem, as classical optimization techniques often fail to scale efficiently with increased problem size and constraints.

Among metaheuristic techniques, **Genetic Algorithms (GA)** have emerged as a robust and flexible solution for VRPs. Goldberg (1989) laid the foundational principles for genetic algorithms, inspired by the process of natural selection. These algorithms simulate the evolution of a population of candidate solutions through selection, crossover, and mutation operations. GA has been successfully applied to complex routing problems, especially those involving dynamic constraints, multi-objective optimization, and real-time adjustments (Holland, 1975; Baker and Ayechew, 2003).

A variety of enhancements to the basic GA framework have been proposed in recent years to improve solution quality and convergence speed. For instance, Prins (2004) introduced hybrid genetic algorithms that integrate local search methods, showing significant improvements in both computational time and optimality. Similarly, Liu et al. (2014) applied a GA-based solution for VRP with time windows (VRPTW), demonstrating its effectiveness in dealing with customer constraints and delivery schedules.

In the context of **retail logistics**, the optimization of delivery routes is crucial for maintaining cost-efficiency and timely service. Studies by Crainic and Laporte (1998) emphasized the role of intelligent routing in modern supply chains, particularly for companies operating with centralized warehouse systems and multiple retail nodes. More recent works by Tan et al. (2018) have focused on integrating real-time data with GA to enhance responsiveness in dynamic retail environments.

The use of GA in your project aligns with contemporary research that emphasizes **population-based exploration and adaptability**. Chromosome representation, fitness evaluation, and the careful design of crossover and mutation strategies are critical to solving VRPs effectively (Bräysy & Gendreau, 2005). Your application of GA to scenarios involving varying customer numbers and vehicle capacities echoes studies by Potvin (2009), which highlight the GA’s strength in handling heterogeneity and scale.

Overall, the literature underscores that while exact algorithms are limited in scalability, **Genetic Algorithms provide a near-optimal yet computationally feasible alternative** for large-scale and constraint-heavy routing problems, particularly relevant in the retail sector.

2 Problem Formulation

The problem addressed in this project is a capacitated vehicle routing problem, for optimising products shipments from a central warehouse to “n” shopping malls using “m” vehicles each having “x” capacity. The objective is to minimise the total distance travelled by all vehicles while ensuring all customer demands are met and vehicle capacity constraints are respected, there is a central warehouse, from this central warehouse various products are distributed to all retail stores the location of retail store with respect to the central warehouse is known, further there are “n” vehicles each of capacity “m” available. The demand of products from various retail store in terms of vehicle capacity is also known, the objective is to find the optimal route for “n” vehicles using Genetic Algorithm so that demand of various retail store is met and is should not exceed the vehicle capacity.

Following are the assumptions

1. all vehicles are initially available at central warehouse.
2. each vehicle capacity is same.
3. vehicle will start from central warehouse and end its journey at central warehouse itself after serving the allotted retail stores.
4. Every retail store must be served.

Parameters:

1. **Vehicles - There are ‘m’ vehicles each with a known same capacity and mapped to their IDs in chromosome representation.**
2. **Retail stores - there are ‘n’ Retail store each requiring delivery of products.**
3. **Distances - depot to retail store distances are known pair-wise the customer distances are also known.**
4. **Chromosome representation- a chromosome is a sequence of customer and vehicle IDs arranged in a specific manner where numbers 1 to “n” represent customers while the number “n +1” to “m” represent vehicles.**

**Constraints:**

**Customer is served exactly once vehicle capacity must not be exceeded and all vehicles must be included in the chromosomes even if some have empty rout.**

**Fitness Function**

**The fitness function of our problem is the reciprocal of the total distance travelled.**

**Introduction to Genetic Algorithm**

**Genetic Algorithm (GA) is a widely used evolutionary optimization technique inspired by the process of natural selection and the principles of genetics. Initially introduced by John Holland in the 1970s, it belongs to the broader class of evolutionary algorithms that simulate the adaptive processes of natural evolution. GAs are especially effective for solving complex problems where traditional optimization methods struggle, particularly in cases involving large, nonlinear, or poorly defined search spaces.**

**The fundamental concept of GA revolves around evolving a population of candidate solutions (called chromosomes or individuals) through biologically inspired operators. Each individual in the population represents a potential solution to the problem, and a fitness function is used to evaluate how well each solution performs.**

**Basic Steps Involved in a Genetic Algorithm**

1. **Initialization  
   A population of potential solutions is randomly generated. Each individual (chromosome) is typically encoded as a string often binary, integer, or real valued depending on the problem domain.**
2. **Fitness Evaluation  
   Each individual is evaluated using a fitness function that measures the quality of the solution. The fitness function is problem-specific and plays a critical role in guiding the evolution process.**
3. **Selection  
   Individuals are selected for reproduction based on their fitness. The goal is to give better performing solutions a higher chance of passing their genetic information to the next generation. Common selection techniques include roulette wheel selection, tournament selection, and rank selection.**
4. **Crossover (Recombination)  
   Pairs of selected individuals are combined to produce offspring. Crossover exchanges genetic material between parents to explore new regions of the solution space. Methods include single point, two point, and uniform crossover.**
5. **Mutation  
   Random modifications are introduced to one or more genes of an individual. Mutation maintains diversity in the population and prevents premature convergence to local optima. The mutation rate is typically kept low to avoid disrupting good solutions.**
6. **Replacement (Generation Update)  
   The new offspring replace some or all individuals in the existing population to form the next generation. This process continues over multiple generations.**
7. **Termination  
   The algorithm stops when a predefined stopping criterion is met, such as reaching a maximum number of generations, achieving a satisfactory fitness level, or observing stagnation in improvement.**

**Genetic Algorithms are valued for their robustness, parallelism, and adaptability, making them suitable for a wide range of applications including optimization, machine learning, scheduling, and engineering design. Their heuristic nature allows them to efficiently search large and complex spaces where traditional techniques may fail or be computationally expensive.**

**Adopted Methodology**

**To address the vehicle routing problem described in this study, a Genetic Algorithm (GA) based approach has been implemented. The methodology involves modeling the problem in a chromosome format compatible with GA, defining an appropriate fitness function, and designing effective genetic operators to evolve the solution space toward optimal or near-optimal routes.**

**The problem space is first translated into a chromosome representation where customer IDs and vehicle delimiters are encoded in a linear sequence. Each chromosome represents a complete delivery schedule, including which vehicle serves which retail stores and in what order. The genes of the chromosome consist of customer indices and vehicle IDs arranged to reflect valid delivery paths, adhering to constraints such as vehicle capacity and one time customer service.**

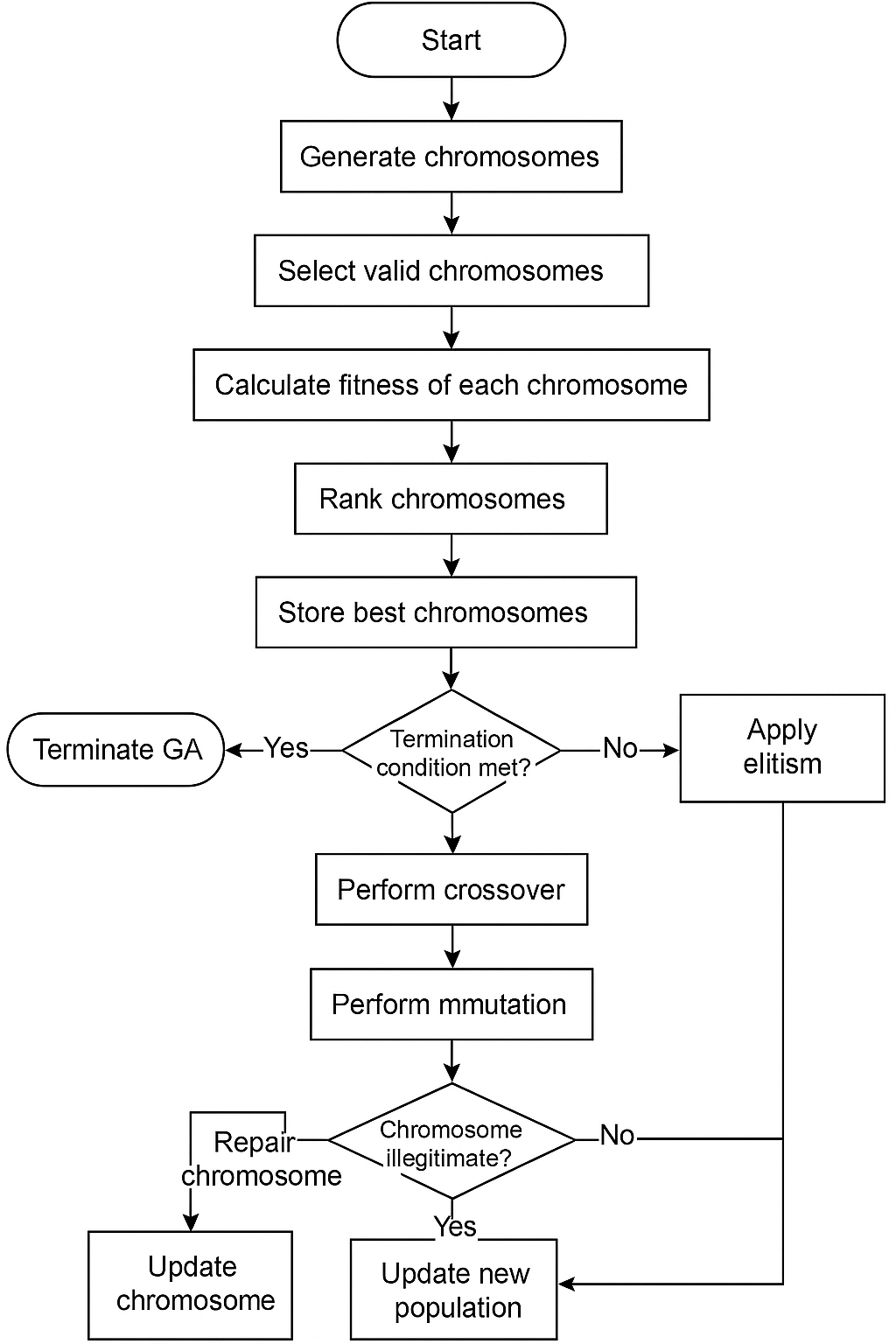
**The algorithm begins with an initial population of randomly generated chromosomes. Each chromosome in the initial population is evaluated using a fitness function, which is defined as the reciprocal of the total distance travelled by all vehicles. This ensures that chromosomes representing shorter and more efficient routes are rewarded with higher fitness scores.**

**To evolve the population over generations, the following genetic operators are applied:**

* **Selection: A tournament selection method is used to probabilistically select fitter chromosomes for reproduction, ensuring better solutions are more likely to pass on their genetic material.**
* **Crossover: A tailored crossover operation is employed (Single point crossover with crossover probability of 0.8) to recombine portions of parent chromosomes while preserving route feasibility. This allows the offspring to inherit favorable traits such as shorter sub-routes or well-balanced vehicle loads.**
* **Mutation: A mutation operation introduces diversity into the population by randomly swapping customer positions or reallocating customers to different vehicles.The method used is swap mutation with mutation probability of 0.25.**
* **After mutation the legitimicy of the chromosomes are checked based on the parameters such as, there should be no repeatation of any number(either vehicle or retail store) , no retail store should remain un attended, demand of every retail store must be met and vehicle load carrying capacity must not be violated.**
* **Each new generation replaces a portion of the old population based on elitism and replacement strategies to maintain diversity while preserving the best-found solutions and elitism is set at 0.1, it means that the top 10% of the chromosomes are added in next generation without undergoing mutation or any other changes. The process repeats until the termination condition is satisfied, maximum number of generations or stagnation in fitness improvement upto 50 generation.**

**To test the robustness and efficiency of the proposed methodology, ten case studies were conducted, varying the number of customers and vehicle capacities. The algorithm consistently produced near-optimal routes that fulfilled all customer demands while minimizing total travel distance. These experiments demonstrate the practical applicability of Genetic Algorithms for large-scale, constraint-bound vehicle routing problems in logistics and supply chain systems.**

**FLOW-CHART FOR VRP USING GA**

****

**Result and discussion**

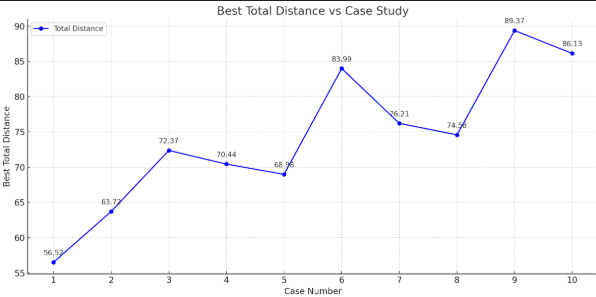
**To evaluate the effectiveness of the implemented Genetic Algorithm in solving the Capacitated Vehicle Routing Problem (CVRP), a series of ten case studies were conducted. Each case varied in terms of the number of customers, vehicle count, and capacity constraints. The goal in all cases was to minimize the total distance travelled while ensuring that vehicle capacity constraints were satisfied and customer demands were fully met.**

**Summary of Case Studies**

| **Case** | **No. of Customers** | **No. of Vehicles** | **Vehicle Capacity** | **Best Total Distance** |
| --- | --- | --- | --- | --- |
| **1** | **6** | **3** | **6** | **56.52** |
| **2** | **8** | **3** | **6** | **63.72** |
| **3** | **10** | **3** | **6** | **72.37** |
| **4** | **10** | **4** | **6** | **70.44** |
| **5** | **10** | **5** | **6** | **68.98** |
| **6** | **12** | **4** | **6** | **83.99** |
| **7** | **12** | **5** | **6** | **76.21** |
| **8** | **12** | **6** | **6** | **74.58** |
| **9** | **14** | **5** | **6** | **89.37** |
| **10** | **14** | **6** | **6** | **86.13** |

**Observations**

* **The algorithm consistently produced valid and efficient vehicle routes with no violation of capacity constraints.**
* **As the number of vehicles increased, the total distance tended to decrease, highlighting better distribution of customer loads across vehicles.**
* **In cases with fewer vehicles, the algorithm still found feasible solutions by splitting high-demand customers when necessary and carefully optimizing route allocations.**
* **The fitness function—defined as the reciprocal of the total distance—effectively guided the selection process toward shorter and more efficient routes across generations.**
* **The plotted trend of fitness vs. generation showed a steady convergence, indicating the algorithm's stability and ability to escape local optima.**

****

**Summary of Case Studies with Chromosomes and Fitness**

| **Case** | **No. of Customers** | **No. of Vehicles** | **Vehicle Capacity** | **Best Total Distance** | **Fitness Value** | **Best Chromosome** |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | **6** | **3** | **6** | **56.52** | **0.01769** | **1 2 9 3 10 4 11 5 12 6 13** |
| **2** | **8** | **3** | **6** | **63.72** | **0.01569** | **1 2 10 3 11 4 12 5 13 6 14 7 15 8 16** |
| **3** | **10** | **3** | **6** | **72.37** | **0.01382** | **1 2 10 3 11 4 12 5 13 6 14 7 15 8 16 9 17 10 18** |
| **4** | **10** | **4** | **6** | **70.44** | **0.01420** | **1 2 11 3 12 4 13 5 14 6 15 7 16 8 17 9 18 10 19** |
| **5** | **10** | **5** | **6** | **68.98** | **0.01450** | **1 2 11 3 12 4 13 5 14 6 15 7 16 8 17 9 18 10 19** |
| **6** | **12** | **4** | **6** | **83.99** | **0.01191** | **1 2 13 3 14 4 15 5 16 6 17 7 18 8 19 9 20 10 21 11 22 12 23** |
| **7** | **12** | **5** | **6** | **76.21** | **0.01312** | **1 2 13 3 14 4 15 5 16 6 17 7 18 8 19 9 20 10 21 11 22 12 23** |
| **8** | **12** | **6** | **6** | **74.58** | **0.01341** | **1 2 13 3 14 4 15 5 16 6 17 7 18 8 19 9 20 10 21 11 22 12 23** |
| **9** | **14** | **5** | **6** | **89.37** | **0.01119** | **1 2 15 3 16 4 17 5 18 6 19 7 20 8 21 9 22 10 23 11 24 12 25 13 26 14 27** |
| **10** | **14** | **6** | **6** | **86.13** | **0.01161** | **1 2 15 3 16 4 17 5 18 6 19 7 20 8 21 9 22 10 23 11 24 12 25 13 26 14 27** |

**First case studyA graph with a line

AI-generated content may be incorrect.**

**Second case study**

**A graph with blue dots

AI-generated content may be incorrect.**

**Third case study**

**A graph with a line drawn on it

AI-generated content may be incorrect.**

**Fourth case study**

**A graph with a line drawn on it

AI-generated content may be incorrect.**

**Fifth case study**

**A graph with a blue line

AI-generated content may be incorrect.**

**Sixth case study**

**A graph with blue lines

AI-generated content may be incorrect.**

**Seventh case study**

**A graph with blue dots

AI-generated content may be incorrect.**

**Conclusion**

**The Genetic Algorithm (GA) approach demonstrated a strong capability in solving complex Vehicle Routing Problem (VRP) scenarios effectively and efficiently. Throughout the case studies, the algorithm consistently provided near-optimal solutions, showcasing its adaptability to variations in the number of vehicles and vehicle capacities. By adjusting these parameters, logistics managers can gain valuable insights into the trade-offs between fleet size, route efficiency, and delivery coverage.**

**The GA’s iterative process of selection, crossover, mutation, and fitness evaluation enabled it to navigate large solution spaces and converge toward high-quality solutions without exhaustive computation. Its performance across multiple scenarios confirms that GA-based optimization is not only computationally feasible but also highly applicable for real-world retail logistics.**

**Moreover, the visual representation of optimal routes and the tracking of fitness across generations enhance interpretability, making this method a practical decision-support tool for supply chain managers. Overall, the success of this GA model reinforces its value as a reliable and scalable solution in addressing dynamic routing challenges within modern distribution networks.**

**Future Research Direction**

**To enhance the complexity and realism of the current vehicle routing model, future research can explore several extensions. One key direction is the incorporation of heterogeneous vehicle capacities, where different vehicles have varying load limits, reflecting more practical logistics scenarios. Additionally, the model can be scaled to include multiple depots, enabling the algorithm to optimize routes from several starting locations rather than a single warehouse. A promising extension involves transforming the problem into a three-echelon vehicle routing problem, which introduces intermediate distribution centers between the depot and the customers. Finally, time-window constraints can be imposed to model delivery schedules, ensuring that customers are served within specific time frames. These enhancements will make the model more robust, realistic, and applicable to dynamic and large-scale supply chain systems.**

**Appendix**

**CODE**

**import random**

**import math**

**import matplotlib.pyplot as plt**

**# ---------------------------- Demand Handling Functions ----------------------------**

**def split\_customer\_demand(customer\_id, demand, capacity):**

**"""**

**Splits customer demand into parts if it exceeds truck capacity.**

**Returns list of tuples with part labels and split demands.**

**"""**

**splits = []**

**while demand > capacity:**

**splits.append((f"{customer\_id}p{len(splits)+1}", capacity))**

**demand -= capacity**

**if demand > 0:**

**splits.append((f"{customer\_id}p{len(splits)+1}", demand))**

**return splits**

**def get\_customer\_parts(customers, demands, capacity):**

**"""**

**Generates customer part labels and their corresponding demands.**

**Handles cases where demand > capacity.**

**"""**

**customer\_parts = []**

**demand\_parts = []**

**for c, d in zip(customers, demands):**

**if d > capacity:**

**splits = split\_customer\_demand(c, d, capacity)**

**for label, part\_demand in splits:**

**customer\_parts.append(label)**

**demand\_parts.append(part\_demand)**

**else:**

**customer\_parts.append(str(c))**

**demand\_parts.append(d)**

**return customer\_parts, demand\_parts**

**def build\_demands\_dict(customers, demands, capacity):**

**"""**

**Builds dictionary of customer part to its demand for fast lookup.**

**"""**

**demands\_dict = {}**

**for c, d in zip(customers, demands):**

**if d > capacity:**

**splits = split\_customer\_demand(c, d, capacity)**

**for label, part\_demand in splits:**

**demands\_dict[label] = part\_demand**

**else:**

**demands\_dict[str(c)] = d**

**return demands\_dict**

**# ---------------------------- Chromosome Construction ----------------------------**

**def generate\_random\_chromosome(customer\_parts, demand\_parts, vehicles, capacity):**

**"""**

**Generates a random feasible chromosome (a potential solution).**

**Each chromosome is a sequence of customer parts followed by a vehicle ID.**

**"""**

**unassigned = list(zip(customer\_parts, demand\_parts))**

**random.shuffle(unassigned)**

**vehicles\_left = vehicles[:]**

**random.shuffle(vehicles\_left)**

**chromosome = []**

**used\_vehicles = set()**

**i = 0**

**n = len(unassigned)**

**while i < n and vehicles\_left:**

**v = vehicles\_left.pop()**

**used\_vehicles.add(v)**

**group = []**

**load = 0**

**while i < n and load + unassigned[i][1] <= capacity:**

**group.append(unassigned[i][0])**

**load += unassigned[i][1]**

**i += 1**

**if group:**

**chromosome.extend(group)**

**chromosome.append(str(v))**

**if i < n:**

**return None**

**for v in vehicles:**

**if v not in used\_vehicles:**

**chromosome.append(str(v))**

**return chromosome**

**# ---------------------------- Chromosome Interpretation ----------------------------**

**def parse\_chromosome(chromosome, vehicles):**

**"""**

**Parses chromosome into vehicle-wise assignments.**

**"""**

**vehicle\_set = set(str(v) for v in vehicles)**

**assignments = {v: [] for v in vehicles}**

**i = 0**

**n = len(chromosome)**

**while i < n:**

**group = []**

**while i < n and chromosome[i] not in vehicle\_set:**

**group.append(chromosome[i])**

**i += 1**

**if i < n and chromosome[i] in vehicle\_set:**

**v = int(chromosome[i])**

**assignments[v].extend(group)**

**i += 1**

**return assignments**

**def is\_legitimate\_chromosome(chrom, customer\_parts, vehicles, demands\_dict, capacity):**

**"""**

**Checks if a chromosome is valid: no duplicates, demand within capacity, all customers covered.**

**"""**

**assigned\_customers = set()**

**assignments = parse\_chromosome(chrom, vehicles)**

**for v, cust\_list in assignments.items():**

**total\_load = 0**

**for cust in cust\_list:**

**if cust not in customer\_parts or cust in assigned\_customers:**

**return False**

**assigned\_customers.add(cust)**

**total\_load += demands\_dict[cust]**

**if total\_load > capacity:**

**return False**

**if assigned\_customers != set(customer\_parts):**

**return False**

**if len(chrom) != len(set(chrom)):**

**return False**

**if not all(str(v) in chrom for v in vehicles):**

**return False**

**return True**

**# ---------------------------- Distance & Fitness Calculation ----------------------------**

**def calculate\_distance(coord1, coord2):**

**"""Euclidean distance between two coordinates."""**

**return math.sqrt((coord1[0] - coord2[0])\*\*2 + (coord1[1] - coord2[1])\*\*2)**

**def calculate\_vehicle\_distances(assignments, customers, demands, vehicles, coordinates, depot=(0, 0)):**

**"""**

**Computes route distance for each vehicle using customer coordinates.**

**"""**

**cust\_part\_coords = {str(c): coordinates[i] for i, c in enumerate(customers)}**

**for i, c in enumerate(customers):**

**for part\_num in range(2, 20):**

**cust\_part\_coords[f"{c}p{part\_num}"] = coordinates[i]**

**vehicle\_distances = {}**

**for v, cust\_list in assignments.items():**

**if not cust\_list:**

**vehicle\_distances[v] = 0.0**

**continue**

**total\_dist = 0.0**

**prev\_point = depot**

**for cust in cust\_list:**

**total\_dist += calculate\_distance(prev\_point, cust\_part\_coords[cust])**

**prev\_point = cust\_part\_coords[cust]**

**total\_dist += calculate\_distance(prev\_point, depot)**

**vehicle\_distances[v] = total\_dist**

**return vehicle\_distances**

**def fitness\_function(total\_distance):**

**"""Returns inverse of total distance as fitness (higher is better)."""**

**return float('inf') if total\_distance == 0 else 1 / total\_distance**

**# ---------------------------- Genetic Operators ----------------------------**

**def single\_point\_crossover(parent1, parent2):**

**"""Performs single-point crossover on two parents."""**

**length = len(parent1)**

**if length < 2:**

**return parent1[:], parent2[:]**

**point = random.randint(1, length - 2)**

**child1 = parent1[:point] + parent2[point:]**

**child2 = parent2[:point] + parent1[point:]**

**return child1, child2**

**def swap\_mutation(chrom):**

**"""Randomly swaps two genes (used for mutation)."""**

**c = chrom[:]**

**idx1, idx2 = random.sample(range(len(c)), 2)**

**c[idx1], c[idx2] = c[idx2], c[idx1]**

**return c**

**def repair\_chromosome(chrom, customer\_parts, vehicles):**

**"""Removes duplicates and ensures all required genes are present."""**

**seen = set()**

**new\_chrom = []**

**for gene in chrom:**

**if gene not in seen:**

**new\_chrom.append(gene)**

**seen.add(gene)**

**all\_numbers = set(customer\_parts + [str(v) for v in vehicles])**

**missing = all\_numbers - set(new\_chrom)**

**new\_chrom += list(missing)**

**return new\_chrom[:len(customer\_parts) + len(vehicles)]**

**# ---------------------------- Tracking and Visualization ----------------------------**

**def store\_best\_chromosomes(generation, chromosome\_fitness, best\_chromosomes):**

**"""Stores best chromosome and its fitness per generation."""**

**best = chromosome\_fitness[0]**

**best\_chromosomes.append({**

**'generation': generation,**

**'chromosome': best['chromosome'],**

**'fitness': best['fitness']**

**})**

**def plot\_fitness\_vs\_generation(best\_chromosomes):**

**"""Plots best fitness value over generations."""**

**generations = [entry['generation'] for entry in best\_chromosomes]**

**fitness\_values = [entry['fitness'] for entry in best\_chromosomes]**

**plt.figure(figsize=(10, 6))**

**plt.plot(generations, fitness\_values, marker='o', linestyle='-', color='b')**

**plt.title('Best Fitness Value vs Generation')**

**plt.xlabel('Generation')**

**plt.ylabel('Best Fitness Value')**

**plt.grid(True)**

**plt.tight\_layout()**

**plt.show()**

**def visualize\_optimal\_routes(assignments, coordinates, customers, depot=(0, 0)):**

**"""Visualizes the routes of each vehicle on a 2D plane."""**

**cust\_coords = {str(c): coordinates[i] for i, c in enumerate(customers)}**

**for i, c in enumerate(customers):**

**for p in range(2, 20):**

**cust\_coords[f"{c}p{p}"] = coordinates[i]**

**plt.figure(figsize=(10, 8))**

**colors = plt.cm.tab10.colors**

**for idx, (vehicle, route) in enumerate(assignments.items()):**

**if not route:**

**continue**

**route\_coords = [depot] + [cust\_coords[cust] for cust in route] + [depot]**

**x, y = zip(\*route\_coords)**

**plt.plot(x, y, marker='o', label=f'Vehicle {vehicle}', color=colors[idx % len(colors)])**

**for cust in route:**

**cx, cy = cust\_coords[cust]**

**plt.text(cx, cy, cust, fontsize=9)**

**plt.scatter(\*depot, color='black', marker='s', s=100, label='Depot')**

**plt.title('Optimal Routes for Vehicles')**

**plt.xlabel('X Coordinate')**

**plt.ylabel('Y Coordinate')**

**plt.legend()**

**plt.grid(True)**

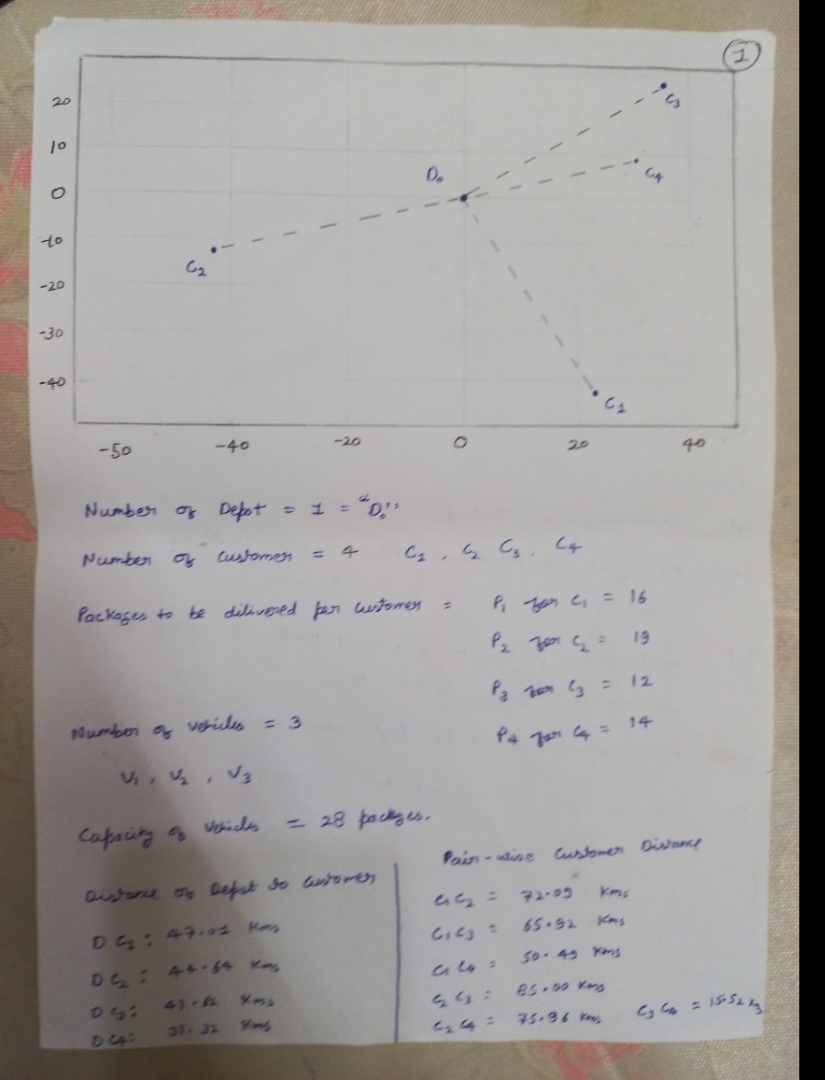
**plt.tight\_layout()**

**plt.show()**

**Manual Exercise.**

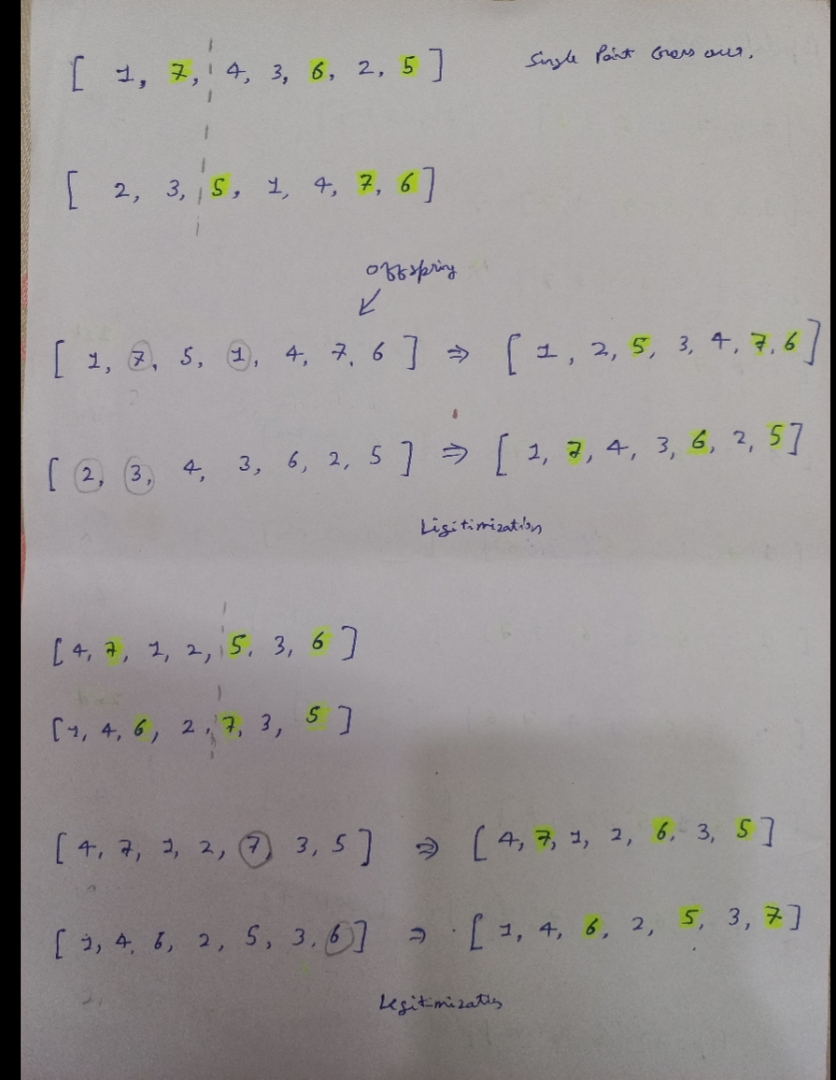
**A paper with writing on it

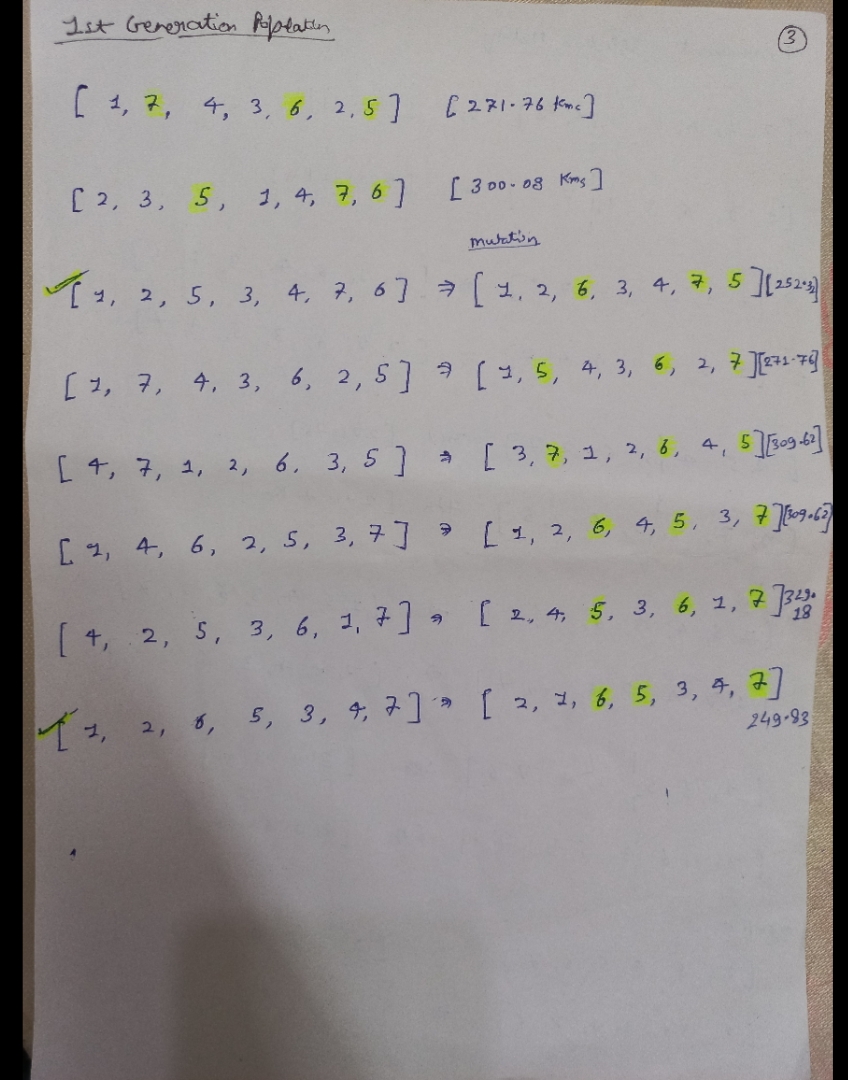
AI-generated content may be incorrect.**

****

**A white paper with writing on it

AI-generated content may be incorrect.**

****

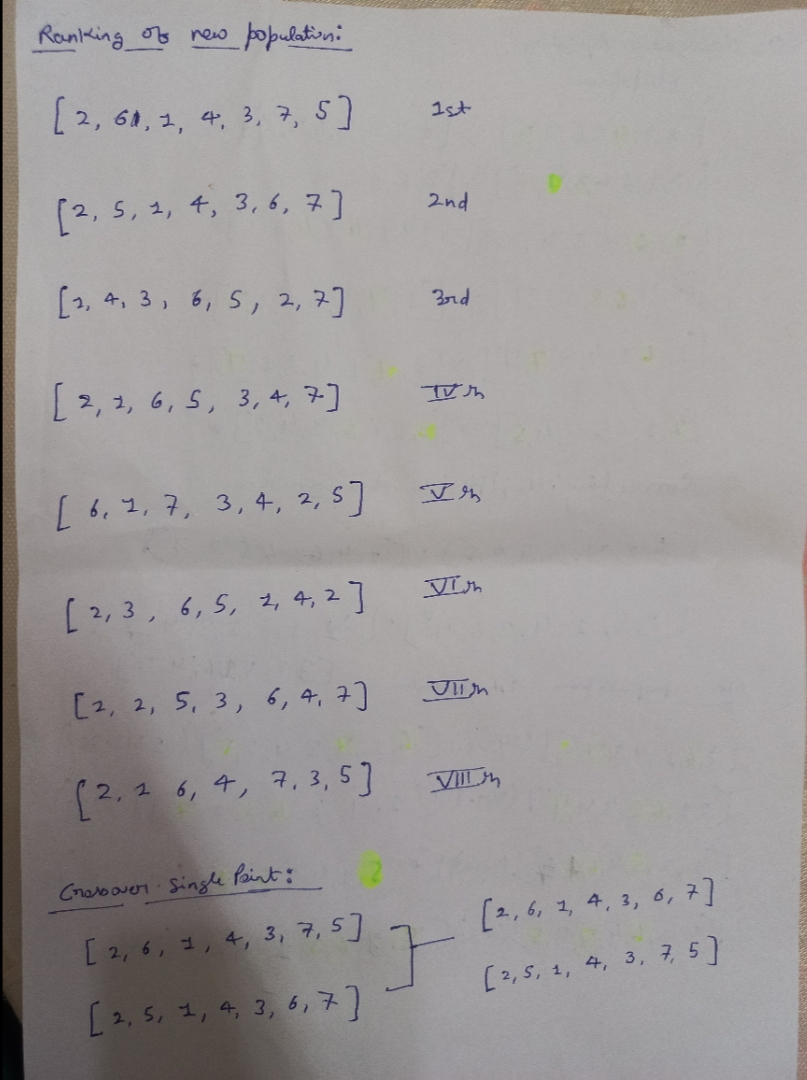
****

**A white paper with green writing on it

AI-generated content may be incorrect.**

**A paper with writing on it

AI-generated content may be incorrect.**

****

**A paper with writing on it

AI-generated content may be incorrect.**

**A paper with writing on it

AI-generated content may be incorrect.**

**A piece of paper with writing on it

AI-generated content may be incorrect.**

**A white paper with writing on it

AI-generated content may be incorrect.**