Project 3: CVRP GA Solver with different Initialization Strategies

William Sykes

October 30, 2024

1 Introduction

The capacitated vehicle routing problem (CVRP) proposed by [2] is the Traveling Salesman Problem but with an imposed condition that specified deliveries, **q** be made at every point **P** except for the terminal point. Another version of the problem is further described in[1] where it consists of designing a set of routes for a fleet of identical vehicles with an overall minimum route cost that services all demands such that,

1. All vehicles should begin and terminate at the central depot
2. like the TSP, problem each customer should be visited exactly once, by exactly one route
3. The total demand of each route does not exceed the capacity of the vehicle
4. The traveling distance of each vehicle cannot exceed the max traveling distance of each vehicle
5. Split deliveries are not allowed

This is an NP-hard problem with large-scale real-life applications as finding the optimal solution of an instance of the problem is not only very hard but requires a very long computational time. Thus, many metaheuristic methods have been developed in attempts to find optimal or near-optimal solutions with less computational time. This includes metaheuristics we have discussed like colony optimization, particle swarm optimization, and of course genetic and hybrid genetic algorithms. Given the research I have already conducted on genetic algorithms and the subsequent algorithm I developed, I am going to focus on a genetic and hybrid genetic approach to this problem.

2 Algorithm Introductions

2.1 General CVRP GA Framework

We start by defining functions for reading CVRP data, calculating distances, and evaluating the fitness of a solution. The **readcvrp** function parses a problem instance from the provided .txt file, extracting essential data such as node coordinates, customer demands, and vehicle capacity. The **calculate eeuclidean distance** and **calculate route distance** functions are utility methods used to measure the distance between nodes and the total length of a route, respectively. The most pivotal part of the GA I found, is the **fitness function**, which in a GA for the CVRP like the TSP assesses the quality of a solution based on the total distance of all routes; but this was also an opportunity to ensure adherence to vehicle capacity constraints and the minimum number of trucks required. The function incorporates penalties for solutions that exceed capacity limits or fail to meet the truck requirement, ensuring that the algorithm favors feasible and efficient solutions as generations progress.

2.2 Initial Population Generation

A genetic algorithm's performance relies heavily on the diversity and quality of the initial population, or initial search space of the problem. I took special focus in this area of the GA as it yielded widely

different results for different methods used[1]. Several initial population strategies were implemented. First, in **generate solution** we simply randomize the nodes while keeping to capacity constraints. An attempt at the Nearest Neighbor heuristic (**nearest neighbor route**) provided an obviously deterministic solution when applied to each problem. Deviating slightly due to the stochasticity introduced in later evolutionary operators. A stochastic variant of this heuristic was developed in the (**nearest random route**), where we sort the closest customer nodes and then randomly choose between 3. Lastly, we generate an initial population with a version of the **sweep algorithm** [1]. Calculating the angle between the depot and a customer node. This angle is calculated in radians relative to the positive x-axis, facilitating the sorting of nodes based on their angular position around the depot. The sweep algorithm sorts customers based on their angular position relative to the depot and then groups them into routes, ensuring that each route does not exceed the vehicle capacity. Checks for capacity constraints and minimum trucks had to be constant in population and offspring generation. Tailoring the fitness function to punish unfeasible routes had a great effect on the best solutions found with each strategies, however an attempt at further randomizing the nearest neighbor heuristic where the **random route** is applied to a random starting node constantly yielded lists of lists that were unfeasible.

2.3 Evolutionary Operators

We employ several genetic operators to evolve the population towards better solutions, starting with selection. The **roulette wheel selection** [1] function selects parent solutions for breeding based on their fitness, favoring higher-quality solutions while still allowing diversity. The crossover function combines routes from two parents to produce offspring, incorporating a **repair offspring** mechanism to ensure that the resulting solutions adhere to capacity constraints. Mutation is introduced via the **swap mutation** function, which randomly swaps nodes between routes to explore new solutions while maintaining feasibility. This process was tricky, as due to the representation of our solutions (a list of lists) our previously implemented Ordered Crossover (0X) did not translate exactly though it works similarly to what was implemented; a crossover operation between two parent solutions to produce offspring. The crossover selects one route from each parent randomly to form the basis of each offspring. It then fills the rest of the offspring routes by merging routes from both parents, avoiding duplication. However, due to constraints now present in the CVRP, after merging it attempts to repair any potential capacity issues in the offspring. This is done by splitting the route (not the delivery, customer demands are not split amongst trucks) if the route exceeds capacity. Spliting an overloaded route into two separate routes modifies the offspring in-place by adding a new route that takes some of the nodes from the overloaded route, ensuring both resulting routes do not exceed the vehicle's capacity. A similar (**is swap feasible**) function was implemented to return true if a swap adheres to constraints and false if not.

2.4 Evolution Process

The **create next generation** function orchestrates the evolution process, generating offspring from the current population using crossover and mutation, and selecting the best solutions to form the next generation. This process is repeated for 100 generations per run, allowing the population to evolve towards increasingly efficient and feasible solutions. To test each initilization method, the **run genetic algorithm** method must be adjusted for each respective initialization. Parameters are adjusted in main.

3 Results: Genetic Algorithm: Sweep Initialization vs Shuffled Initialization vs Nearest Random Neighbor Initialization

Each experiment is the Best, Worst, and average of 20 runs, each run with a population size of 50, 100 generations, and mutation rate of .8.

	Sweep Algorithm	Random Shuffle	Nearest Random Neighbor
Average Runtime (seconds)	19.45	27.93	15.73
Average	398.56	522.92	476.86
Worst	427.78	611.08	572.86
Best	390.63	444.60	428.10

Table 1: Results for E-n22-k4, 22 Dimensions, 6000 Capacity, Min number of trucks 4, Optimal value 375

	Sweep Algorithm	Random Shuffle	Nearest Random Neighbor
Average Runtime (seconds)	27.41	24.63	21.65
Average	1077.01	1070.69	980.26
Worst	1133	1240.53	1057.72
Best	1035.81	981.78	951.53

Table 2: Results for E-n33-k4, 33 Dimensions, 8000 Capacity, Min number of trucks 4, Optimal value 835

	Sweep Algorithm	Random Shuffle	Nearest Random Neighbor
Average Runtime (seconds)	38.63	38.04	32.29
Average	753.79	1056.56	747.28
Worst	806.76	1152.40	789.66
Best	719.49	951.15	651.68

Table 3: Results for E-n51-kf, 51 Dimensions, 160 Capacity, Min number of trucks 5, Optimal value 521

4 Analysis

4.1 E-n22-k4 (22 Dimensions, 6000 Capacity)

The Sweep Algorithm outperforms the other two methods in terms of average, worst, and best fitness scores, indicating it generates more efficient routes. Notably, its average runtime is significantly lower than Random Shuffle but slightly higher than Nearest Random Neighbor. Despite the Random Shuffle method's longer runtimes, it doesn't yield better solutions, suggesting inefficiency. The Nearest Random Neighbor, while faster, still falls short of the Sweep Algorithm's solution quality.

4.2 E-n33-k4 (33 Dimensions, 8000 Capacity)

Here, the Nearest Random Neighbor method shows a clear advantage in generating higher-quality solutions, as evidenced by its lowest average and best fitness scores. This indicates a better ability to find closer approximations to the optimal solution. Here, the Nearest Random Neighbor method shows a clear advantage in generating higher-quality solutions, as evidenced by its lowest average and best fitness scores. This indicates a better ability to find closer approximations to the optimal solution.

4.3 E-n51-kf (51 Dimensions, 160 Capacity)

The Nearest Random Neighbor method excels in this scenario with the best performance across all metrics, achieving the lowest average, worst, and best fitness scores and demonstrating superior efficiency with the shortest average runtime. The Sweep Algorithm presents a notable improvement over Random Shuffle in terms of solution quality but is comparable in runtime. This further underscores the Nearest Random Neighbor's methodological advantage in handling larger dimension problems efficiently.

4.4 Analysis: Conclusion

The Nearest Random Neighbor initialization method consistently provides high-quality solutions with efficient runtimes, particularly as problem complexity increases. Its performance in the E-n51-kf problem set is especially exciting. I believe it is because there is a balance between exploration and exploitation with this method. It introduces a level of stochasticity by choosing among the nearest neighbors rather than always picking the absolute nearest, which can help in avoiding local minima early in the search process. The randomness in neighbor selection can prevent the algorithm from converging prematurely to suboptimal solutions, a common issue with algorithms that are too deterministic, which I believe The Sweep algorithm runs into. Additionally, while the Nearest Random Neighbor introduces randomness, it still leverages the proximity of nodes to construct initial solutions, which can be more efficient than purely random approaches. This efficiency in route construction leads to better initial solutions, which are crucial for the success of subsequent evolutionary operations in the genetic algorithm, and heavily contributes to the consistently faster run times of the method. The approach's ability to flexibly choose among several close neighbors might also inadvertently help in managing capacity constraints more effectively, even in its initial runs. By not always choosing the closest neighbor, it might avoid clustering high-demand nodes together, thus better distributing demand across routes. The Sweep Algorithm proves to be an effective initialization strategy, particularly for smaller problems (E-n22-k4), showcasing its capability to generate competitive solutions with reasonable computational efficiency, compared to the other strategies. Random Shuffle appears to be the least effective among the three methods, often resulting in longer runtimes and lower-quality solutions, this could possibly change with a different PRNG or paired with The Sweep Algorithm. So, this suggests that a more structured approach to initial population generation, as seen in the other two methods, contributes significantly to the GA's overall effectiveness.

References

[1] H. Awada, R. Elshaer, A. AbdElmo'ez, and G. Nawar. An effective genetic algorithm for capacitated vehicle routing problem. In *Proceedings of the International Conference on Industrial Engineering and Operations Management*, pages Bandung, Indonesia, March 6–8, Zagazig, Egypt; King Khaled University, KSA, 2018. IEOM Society International.

[2] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[3] Y. Marinakis and M. Marinaki. A hybrid genetic - particle swarm optimization algorithm for the vehicle routing problem. *Expert Systems with Applications*, 37(2):1446–1455, Mar 2010.

[4] V. Praveen, D. Keerthika, G. Sivapriya, A. Sarankumar, and B. Bhasker. Vehicle routing optimization problem: A study on capacitated vehicle routing problem. *Materials Today: Proceedings*, 64(Part 1):670–674, 2022.

[5] C.-H. Wang and J.-Z. Lu. A hybrid genetic algorithm that optimizes capacitated vehicle routing problems. *Expert Systems with Applications*, 36(2):2921–2936, 2009.

[6] J. Zhu. Solving capacitated vehicle routing problem by an improved genetic algorithm with fuzzy c-means clustering. *Scientific Programming*, 2022:8, 2022.