CSE 624/424
Heuristic Optimization

Project
(Vehicle Routing Problem)

Muharrem Ozan Yeşiller
171044033

Problem Definition

In this project, within the scope of the CSE424 coded “Optimization” course, a solution was implemented for the vehicle routing problem with genetic algorithm and variable neighborhood search.

Vehicle Routing Problem is a combinatorial optimization problem that aims to determine the most suitable routes for vehicles that will serve customers from one or more warehouses. It is a more general version of the Traveling Salesman Problem, one of the best-known problems in the optimization literature.

Solution Format

The solution format is a list of a list. In the list, it contains the list of cities where the vehicles will stop.

$$\left[vehicle_1\left[city_x, city_y, city_z\right], vehicle_2\left[city_k\right], \dots\ vehicle_n\left[\square\right]\right]$$

Decision Variables

Decision variables are vertices in the graph data structure.

Constraints

Constraints are to visit all cities at once.

Objective Function

The graph data structure does not include the repository. There is a different data structure that contains the cost of each city from the warehouse.

The objective function is the sum of the cost of the route traveled by the vehicles and the cost between the first stop city and the warehouse.

Genetic Algorithm Design

This algorithm takes the capacity of the population as input. Genetic algorithm design consists of 3 different operators.

Crossingover Operation, this operation does its job with the PMX Crossover technique. While this technique is being implemented, the solution is first encoded into a bit string. Then, the pairs in the population are determined (first pair, next pair, etc.), a range is determined between the genes of these pairs, and the first parent inherits from the offspring, and the spacing of the first parent is inherited from the children to the second. Afterwards, children are filled with the remaining genes depending on their parents in a feasible way.

The implementation of the bitstring design has been chosen because it provides convenience in terms of programming languages. PMX, on the other hand, was preferred because it is a very powerful crossover operation.

Mutation Operation, the mutation will occur for each child with a 25% chance. In the mutation operation, the solution is again encoded into a bitstring, and two of these genes are randomly selected and swapped.

The reason for choosing this design is because the mutation provides us with diversification, so it is the random gene replacement design.

Natural Selection Operation, the natural selection is the survival of the best member from a combined list of all parents and children, as well as population capacity.

The reason for using this design is based on the continuity of good solutions for generations and provides intensification.

Variable Neighborhood Search Design

This algorithm includes 2 different sub-algorithms.

While initializing the Maximum Level information, half of the number of decision variables included in the solution was taken as reference.

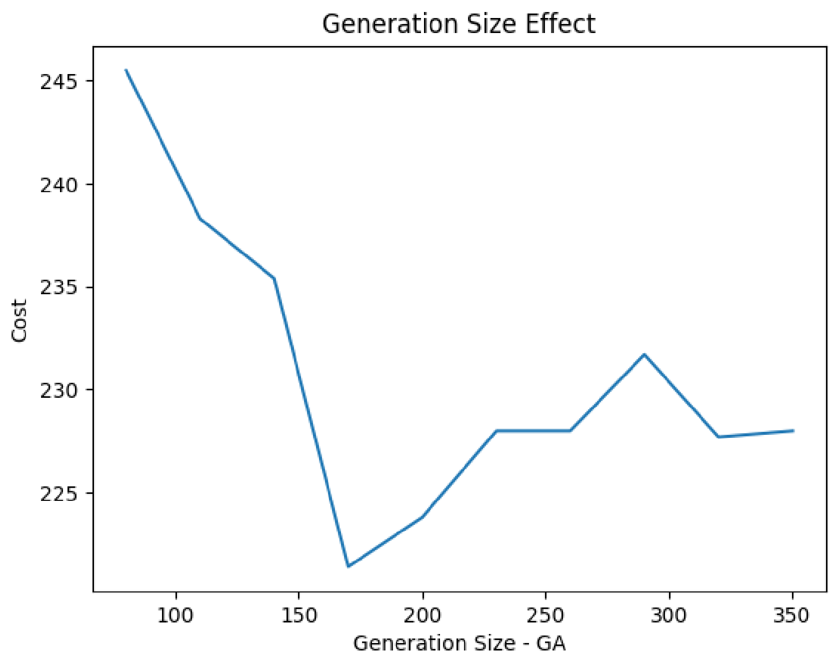
Stochastic Part, in this part, the operation called "shaking" takes place. This operation randomly swaps elements up to the current level with one another. The important point here is that the solution is encoded into a bitstring and swaps are made over this list. The freshly shaken solution is then dissolved.

The level increases with each iteration.

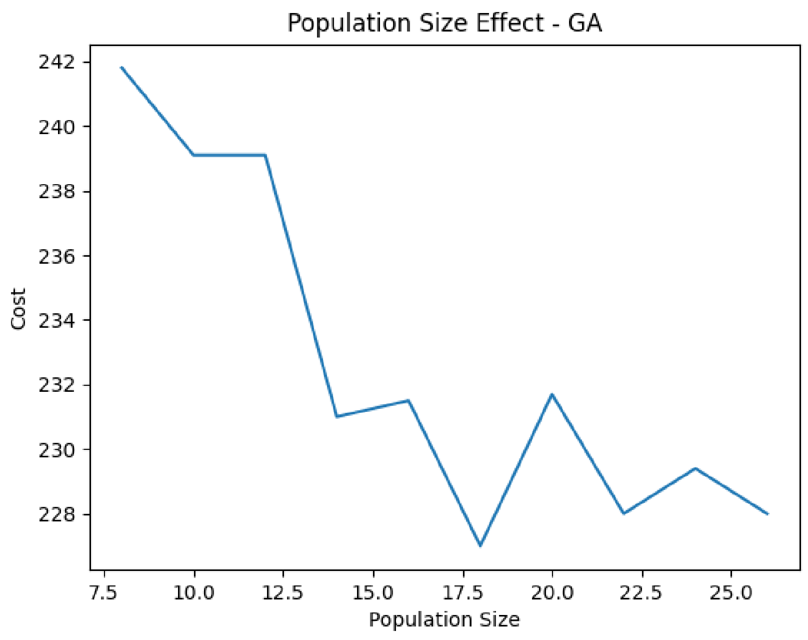
Deterministic Part, local search application is applied in this section. The solution is encrypted to the bitstring and neighbors are found because each element is swapped once with each element. It finds the best one among the neighbors and this process is repeated. If this process is at a point where it cannot be improved, the local optima points are reached and the current solution is solved and returned.

Compare Charts

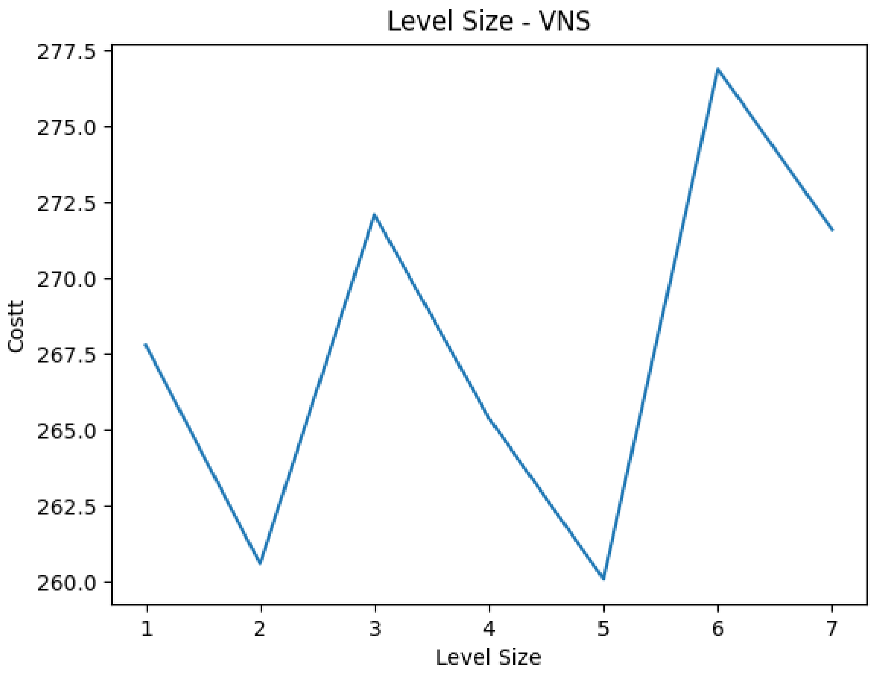
Generation Size - Genetic Algorithm

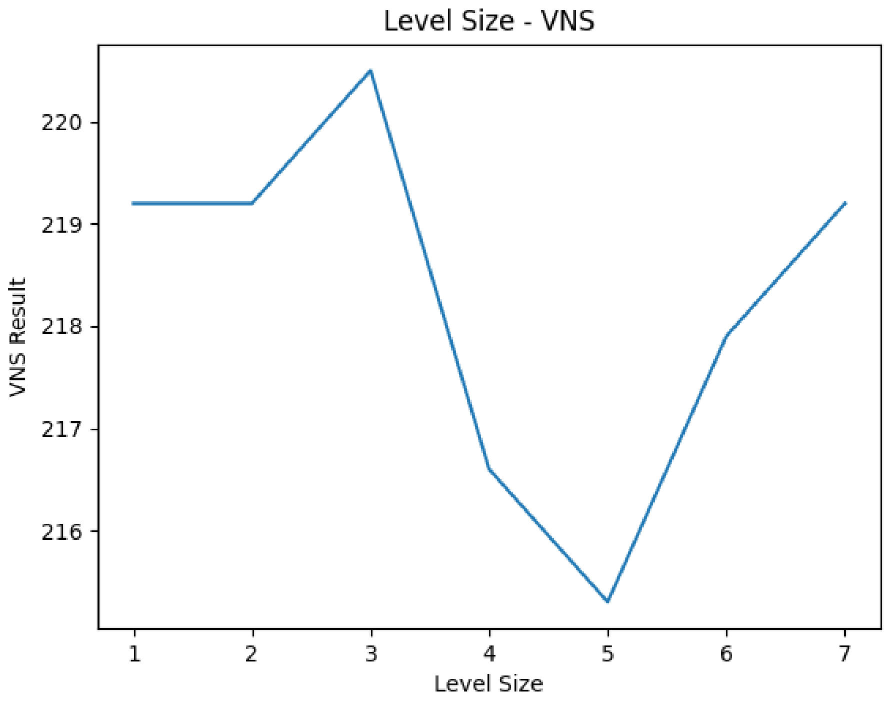


Population Size - Genetic Algorithm

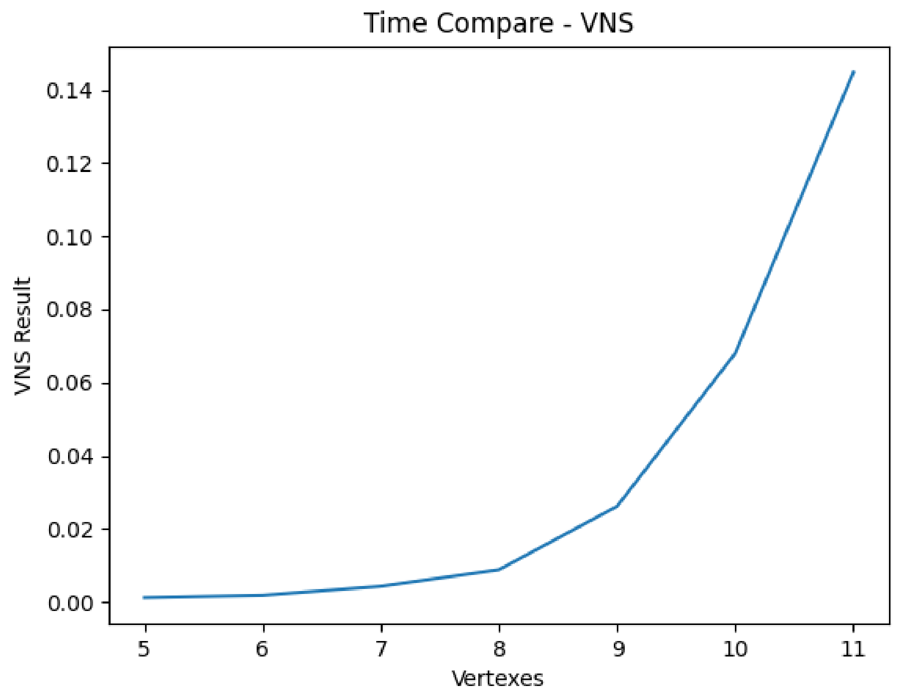


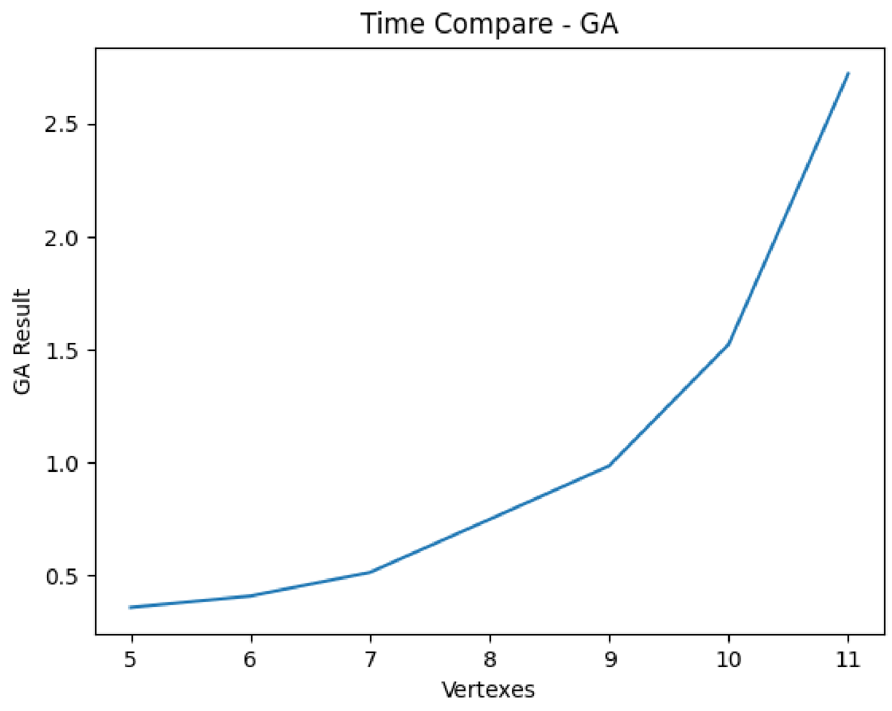
Level Size - Variable Neighborhood Search



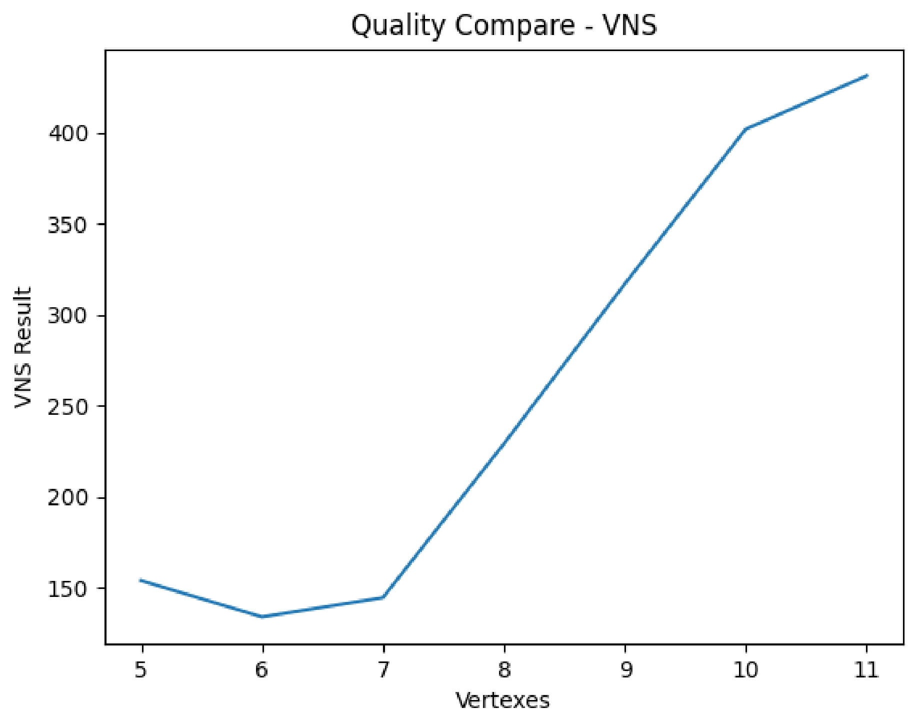


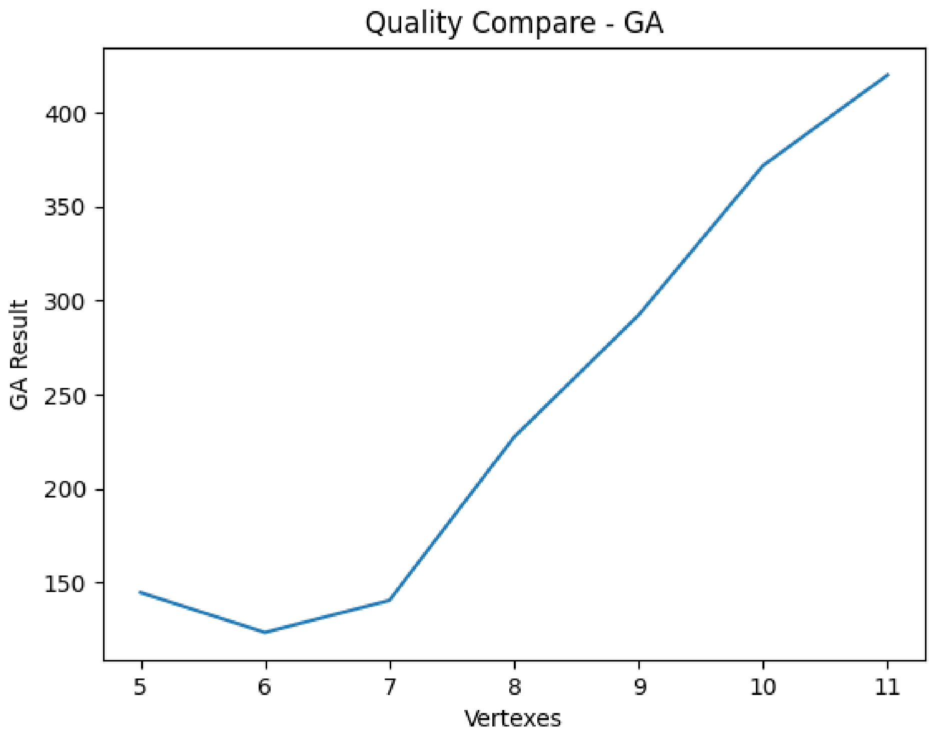
Time Comparison - VNS vs GA





Quality Comparison - VNS vs GA





Conclusions

- Increasing the number of generations for the genetic algorithm can optimize the solution, of course, there is a threshold value. The graph I found is for a graph with 10 vertices. I've had the best results in the generation size 150-200.

- Increasing the number of populations can also optimize the solution, but this may save time according to the design. If the algorithm that creates each solution of the population is slow, increasing this number will slow it down considerably. Finally, the population number must also have a certain threshold value. I took 18 as the best population size range for a 10 vertex graph.

- For a 10 vertex graph, I found the level number to be 5 as the optimum. When I did a few literature searches, half of the graf vertex count was an optimized result for level.

- In line with the information above, I set VNS and GA parameters and made almost the same results in terms of solution quality in the tests I performed on a few graphs, but VNS had superiority in terms of time. Since the population in GA is formed by random solutions, this may also provide a deficiency in terms of time and quality. Populating the population with the results of algorithms such as Local Search, Simulated Annealing, Greedy Algorithms, reducing the number of generations allows us to save time and quality.