Project 6: Genetic Algorithm with Wisdom of Artificial Crowds for NP Complete Problems

Nick York
CES Department
Speed School of Engineering
University of Louisville, USA
neyork01@louisville.edu

1. Introduction

Project 4 required us to write a Genetic Algorithm to solve the Traveling Salesman Problem ("TSP"). Project 5 required us to write a hybrid algorithm to solve the TSP using our Genetic Algorithm and a new metaheuristic – Wisdom of Artificial Crowds. For Project 6, our objective was to write a program ("the program") to apply our hybrid algorithm to a new NP-Complete problem.[1] I choose to solve the Capacitated Vehicle Routing Problem ("CVRP"), a variant of the Vehicle Routing Problem ("VRP").

The VRP is a combinatorial optimization problem. The problem was first defined by Dantzig and Ramser as a generalization of the TSP [1]. The TSP itself requires finding the minimum cost route for a salesman to visit a list of cities without visiting a city more than once, except for the starting city. The salesman must return to their starting city at the end. Each city is represented by Cartesian coordinates and numbered starting at 1. The cost to travel between two cities is the Euclidean distance between them, calculated as

$$cost = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (1)$$

Dantzig and Ramser generalized the TSP to dispatching gasoline trucks to make deliveries to gas stations. Each truck starts from and returns to a central terminal. The gas station locations (P) and demands (q) are known beforehand. Trucks are required to make q_i deliveries at every P_i location, excluding the terminal. Trucks share the same capacity, C. The constraint on C can be described in one of two ways. In the first instance where

$$C \geq \sum_i q_i \quad (2)$$

the problem resembles the TSP, as a single truck has the capacity to make all deliveries. However, in the second instance where

$$C \ll \sum_i q_i \quad (3)$$

[1] During a Microsoft Teams call on Nov 2nd with Dr. Yampolskiy he clarified we could choose any of the problems from the Wikipedia entry "List of NP-complete problems" he provided, except mentioned by name in the instructions as not an option. I asked Dr. Yampolskiy if NP-Hard problems (which appear on Wikipedia entry) were viable, and he said for our purposes they were.

a single truck cannot make all deliveries. This requires dispatching two or more trucks. Each truck visits one or more gas stations until cumulative demand meets or exceeds C. Each gas station is visited exactly once. The objective is to visit all the gas stations while minimizing total distance (or time) traveled. The authors consider other formulations, including multiple demands for N different products, multiple truck capacities, and under- or over-delivery of a fixed percentage based on demand.

Dantzig and Ramser's paper inspired research into new variants of and solutions to the VRP [2]. Variants include the CVRP, the VRP with Time Windows (VRPTW), and Dynamic Vehicle Routing Problems ("DVRP"). The VRP is NP-Hard; exact algorithms can solve for up to 100 deliveries depending on the variant and other factors. Exact algorithms are insufficient for solving large-scale real-world applications of the VRP. Decades of research on the VRP has focused on approximate algorithms – heuristics, metaheuristics, hybrid algorithms – that approach the optimal solution for large problems.

Christofides and Eilon tested solving the CVRP using a branch-and-bound approach, a "savings" approach, and the 3-opt algorithm [3]. The authors found that branch-and-bound (an exact algorithm) was substantially less effective than solving for an equivalent TSP. The largest problem the authors solved using branch-and-bound included 13 customers. Computation time and storage requirements made the branch-and-bound approach intractable for larger problems. On the other hand, 3-opt outperformed the "savings" approach in all but one of ten problems for minimum distance, with the largest problem solved including 100 customers.

Another approach is to use metaheuristics to solve the VRP. Nazif and Lee tested solving the CVRP using their genetic algorithm and several other commonly used metaheuristics [4]. The authors found that their genetic algorithm was competitive both in quality of solutions and time to find a solution. A genetic algorithm ("GA") is a metaheuristic based on the theory of evolution. The theory of evolution states the fittest individuals survive to maturity to reproduce and pass their genes onto their offspring. Data is encoded as genes, collections of genes as chromosomes, and collections of chromosomes as populations.

A population changes as new chromosomes replace old ones over successive generations. Chromosomes are selected for a chance to reproduce each generation based on their fitness. Fitness is a problem-specific measure of utility. During reproduction, a crossover operator exchanges parts of two chromosomes to produce two child chromosomes. Crossover repeats until the child population is the same size as the parent population. Afterward, a mutation operator changes one or more genes for some number of child chromosomes based on a predefined mutation rate. The child population replaces the parent population at this point. The new population is evaluated to see whether it meets some measure known as a stopping criterion. A stopping criterion can be the number of generations run or a lack of improvement over successive generations. If the stopping criterion is not met, then the algorithm repeats until it is.

Besides GAs, swarm intelligence metaheuristics have been researched to solve the VRP [5]. Despite a large body of research spanning decades, I could not find published research applying the wisdom of artificial crowds ("WoAC") to the VRP. Though crowds and swarms are what we might call "collective intelligence" they remain distinct [6]. Specifically, WoAC utilizes the collective wisdom of an artificial crowd to arrive at a solution that is more optimal than the experts' solutions alone. A percentage of the most optimal solution ("elites") are aggregated to combine into a new solution.

WoAC relies on four properties to find an optimal solution. First, a crowd must be diverse conceptually (how they arrive at a solution) and cognitively (how "smart" they are). Second, members of a crowd must arrive at solutions independent of other members so that new solutions are considered, regardless of how a solution is found or how optimal it may be. Third and fourth are decentralization and aggregation, which mean no one individual directs the actions of the crowd and that solutions are aggregated in a way that preserves their independence (i.e., solutions are not negatively weighted relative to the most optimal solution).

2. Approach

I wrote a Python program that allows the user to specify which file or files to use based on their filename or filepath. Each file contains a CVRP instance (see "Data"). The user is presented with two options: 1) solve for a single file, or 2) solve for multiple files. Next, the program reads and parses the vehicle capacity, coordinates, demands, and depot in the CVRP file(s). If the user selected option 2, then the program parses and solves for the CVRP one file at a time. The ability to solve for multiple files automates the process of running files and compiling statistics.

I used most of the same code from project 5 to compile statistics – chromosomes, costs, and runtimes for each iteration of the GA and WoAC were aggregated to find averages. The GA was run for 10 iterations; the WoAC is run for 4 iterations, each using a different number of elites in increments of five, starting at 5 and going 20. Afterward, averages of cost and runtime; standard deviation of cost; and total runtime are found. The chromosome with the lowest minimum cost across all iterations of the GA, the WoAC, and both are found, too.

The number of Elites Used (how many of the most optimal chromosomes from each GA are aggregated) and % Difference (1 minus the cost ratio of best chromosome from WoAC over the best chromosome of the GA) are also reported. If the % Difference is 0%, then the chromosomes' costs are identical. A negative percentage indicates WoAC performed worse than the GA, and vice versa for a positive percentage. The one new statistic included is the number of vehicles (same as the number of routes) in the optimal chromosomes for GA and WoAC.

Genetic Algorithm

I used the same code for the GA as project 5, with minor changes. Like before, mutation rate is fixed at 1%. However, I chose 100 for the population size (see "Discussion"). 20 elites (one-fifth of population size) are added to the selection pool. Generations to run are set manually and vary with each problem to allow the GA time to solve larger, more complex problems (see "Results"). These parameters form the basis of the GA's performance.

Customers and the depot are encoded as genes. Genes have one new attribute – *demand* stores the customer's demand (for the depot this is 0). Collections of genes are encoded as chromosomes, except for the depot to simplify changes to the GA. Finally, collections of chromosomes are encoded as populations.

Each chromosome has three new attributes: *depot*, *capacity*, and *routes*. The first two simply store the value of the depot and capacity for the CVRP. The *routes* attribute stores a list of routes. To find the routes, find_routes() is called. Several variables are initialized inside the function: demand is set to 0, routes to an empty list, route to a list with *depot* as its single element, and length as the number of genes in the chromosome. Each gene in the chromosome is iterated over and its individual demand added to demand. If *capacity* minus demand is greater than or equal to 0, then the gene is appended to route. Otherwise, adding the current gene's demand would exceed capacity, and the route is appended to routes. In this case, demand is reset to the current gene's demand and route is reset to a list with *depot* as its single element. The current gene is appended to the new route and the process repeats until all genes have been added to a route. The final value of routes is returned and assigned to *routes*.

Due to the way routes are constructed, the find_cost() function is altered to sum the cost to travel between each gene (Equation 1) for each route in *routes*. The final value of cost represents the cost to travel all routes, including starting from and returning to the depot each time. The final cost is returned and assigned to *cost*. In this way, *fitness* (calculated as 1/*cost*) accurately reflects the how minimal the overall distance across routes is.

Population has two new attributes: *depot* and *capacity*. *depot* is used to store the depot gene and *capacity* is used to store the vehicle capacity. Both attributes are used to generate the initial population of chromosomes and child chromosomes during crossover. Population is initialized by randomly sampling genes without replacement to construct 100 chromosomes. These 100 chromosomes form the initial population. Once the initial population is constructed, the GA is ready to begin.

The GA applies selection, crossover, mutation, and evaluation till the stopping criterion is met. Selection ranks the fittest chromosomes (those with lowest cost) to select 20 elites from them to automatically add to the selection pool. The remaining chromosomes are selected based on probability, with the fittest chromosomes having the highest probability of selection. Afterward, crossover takes place. While the child chromosomes are fewer than the original population, two parent chromosomes are randomly selected from the selection pool to mate.

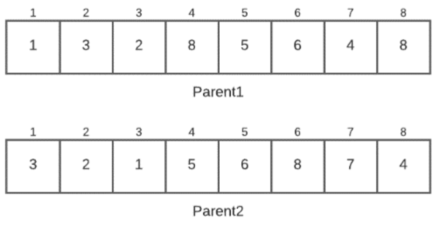

Figure 1. Two parent chromosomes selected to reproduce.

The crossover operator takes a gene sequence from one parent chromosome and combines it with genes from the second parent not in the sequence to begin creating two child chromosomes. The lightly shaded squares in Figure 2 represent the gene sequence from the first parent chromosome that are added to the first child chromosome. The genes before the sequence in the first parent are added to the second child.

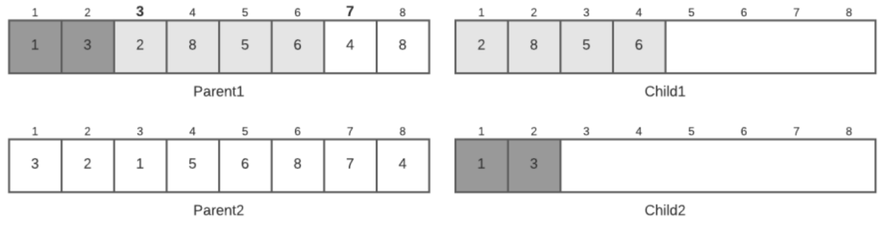

Figure 2. Child chromosomes after the first genes are added. The bold 3 and 7 signify the start and end of the gene sequence.

The genes in the second parent not in the first child are added next. The remaining genes in the second parent are added to the second child, followed by those after the sequence in parent one.

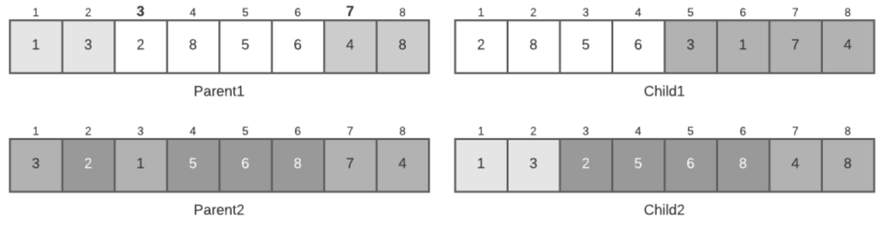

Figure 3. Child chromosomes after remaining genes are added.

After crossover is mutation; the mutation operator takes the child chromosomes and selects some or none to mutate based on a mutation rate. Reverse Sequence Mutation, described by Abdoun et al. [7], is used as the mutation operator. A random gene sequence in the original chromosome is taken, reversed, and inserted back into the chromosome to mutate it.

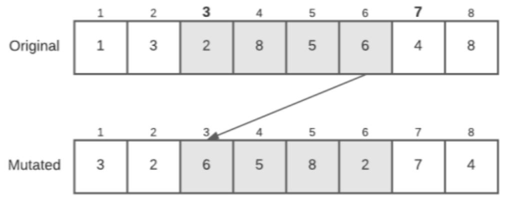

Figure 4. Result of Reverse Sequence Mutation. The bold 3 and 7 signify the start and end of the gene sequence to reverse.

Lastly the new population is evaluated. Since the stopping criterion is number of generations run, a while-loop is used instead of an evaluation function to determine whether GA is finished. The evaluation function is repurposed to select the most optimal chromosome and the best, worst, and average costs. If the current most optimal chromosome is less optimal than one from the child population, then its replaced. The final value of the most optimal chromosome is returned by the GA as the solution.

Wisdom of Artificial Crowds

In project 5, I wrote a modified version of Yampolskiy, Ashby, and Hassan's WoAC algorithm to solve the TSP [7]. Their aggregation method summed the edges found in the elites to create an agreement matrix. Then, the authors used a novel heuristic to find a new optimal solution using the agreement matrix to prioritize edges with the most agreement (the largest values). The CVRP has multiple routes per chromosome that need aggregated to combine into a new solution unlike the TSP. To aggregate multiple routes per chromosome, I modified create_agreement_matrix() to iterate over each route in a chromosome's *routes* attribute. This single change sufficed for aggregation.

The other WoAC function to modify was combine_solutions(), which takes the agreement matrix and cost matrix (matrix of cost to travel between genes) to combine the genes into a new solution. First, two new parameters are passed: depot and vehicle capacity. Since each route begins from the depot, the heuristic considers the index of the edge with the most agreement starting from the depot first. As the first of many indices, the index is used to initialize a list assigned to previous_max_indices. The depot and customer genes are used to initialize a list assigned to route. An empty list is initialized to hold each route as they're finished and assigned to routes. Finally, demand is initialized to the demand of the first customer.

Like before, the program calls a helper function find_maximum() to search for the index of the next edge with maximum agreement whose gene is not a part of the solution yet. This section of combine_solutions() differs as only the edge containing the gene last inserted is considered (i.e., a route [1, 4, 3] looks at edges starting from gene 3). Because of this, the function no longer requires comparing potential genes to add at the start or end, nor tracking where the gene was inserted. find_maximum() returns the index of the next gene to consider adding to the end of the route and its demand, assigning the gene to next_col (referring to the next column in the agreement matrix) and adding the gene's demand to the demand.

If next_col is not the depot (which indicates no other genes can be visited), then the new gene is appended to route. Otherwise, depot is appended to route, and route is appended to routes. Two variables, new_max_index and max_agreement, are initialized to None. Then, the agreement matrix is iterated over to find the index and agreement of the next edge starting from the depot and not yet in the solution with the most agreement. If the index is not in previous_max_indices, and if new_max_index is not None and agreement is less than max_agreement, then new_max_index is updated to the index and max_agreement is updated to the agreement.

Else, new_max_index is None; new_max_index is updated to index and max_agreement is updated to agreement. The final value of new_max_index is appended to previous_max_indices. The new_max_index is used to retrieve the next customer gene to add, reset demand to the new customer's demand and reset route to be the depot and new customer. Once all customer genes have been added, a new chromosome is created and routes are assigned to *routes*.

3.1 Data

I used a subset of CVRP files containing instances from the set E benchmarks. The set E benchmarks are problem instances used by Christofides and Eilon, including their own original instances and those of Dantzig and Ramser; Gaskell; and Gillet and Miller [3]. I downloaded a ZIP file of the benchmarks from the Capacitated Vehicle Routing Problem Library (Table 1) [9]. CVRP instance files have a .vrp file extension. Each filename specifies the benchmark set it belongs to, *n* number of nodes (*n*-1 customers), and *K* minimum number of vehicles (i.e., E-n13-k4.vrp). Additionally, the known optimal solution is available for each instance on their repository. The subset I choose represents less and more complex problems, with problem difficulty based on number of customers and *Q* (i.e., many customers and small *Q* requires more vehicles, which requires optimizing more routes).

Instance	Number of Customers (n-1)	Minimum Number of Vehicles (K)	Capacity (Q)	Tightness	Optimal
E-n22-k4	21	4	6000	94%	375
E-n51-k5	50	5	160	97%	521
E-n76-k7	75	7	220	89%	682
E-n101-k8	100	8	200	91%	815

Table 1. Parameters of benchmark instances and optimal minimum distance.

Uchoa et al. explain that Tightness is a ratio of the sum of vehicle capacity (*KQ*) and total demand [8]. The closer the ratio is to one, the more difficult it is to solve for the optimal solution.

The files use the TSPLIB format found in TSP files used in previous projects [10]. Each file includes the following keywords:

Keyword	Meaning
NAME	Identifies the data file.
COMMENT	Additional comment(s) on the data.
TYPE	Specifies the nature of the data.
DIMENSION	For a CVRP, it is the total number of nodes and depots.
EDGE_WEIGHT_TYPE	Specifies how the edge "weights" (or "lengths") are given if they are given explicitly.
CAPACITY	Specifies the truck capacity in a CVRP.
NODE_COORD_SECTION	List of node coordinates. An integer gives the number of a node. Two real numbers give the associated coordinates.
DEMAND_SECTION	List of demands of all nodes in a CVRP. The first integer specifies node number, the second its demand. The depot node(s) must also occur in this section. Any depot node's demand is 0.
DEPOT_SECTION	List of possible depot nodes in a CVRP. This list is terminated by -1.
EOF	Terminates the input data. This entry is optional.

Table 2. Keywords in set E benchmark files and their meanings.

The TYPE is CVRP for all files (if it were a TSP file, TYPE would be “TSP”). DIMENSION is the same as *n*, the number of nodes. EDGE_WEIGHT_TYPE is given as EUC_2D, which specifies edge weights are given by the 2D Euclidean distance between nodes (Equation 1). CAPACITY is simply an integer representing how much of a product a vehicle can hold. Nodes (both customers and depots) are given an integer to identify it and two real number coordinate values under NODE_COORD_SECTION. Each node is given a demand associated with their identifying integer under DEMAND_SECTION. The DEPOT_SECTION lists depots; the files chosen only list a single depot.

3.2 Results

The results and statistics for all four instances are written to .csv files under a folder named Results. Charts and plots can be displayed as a GUI or (by default) saved as a .png file under Results. I used Lucidchart to plot charts side-by-side. Figure 5 demonstrates how the optimal solution is constructed over generations. Figure 6 shows the optimal solution for the GA and WoAC for each instance. Figure 7 demonstrates how the optimal solution's cost changes over generations.

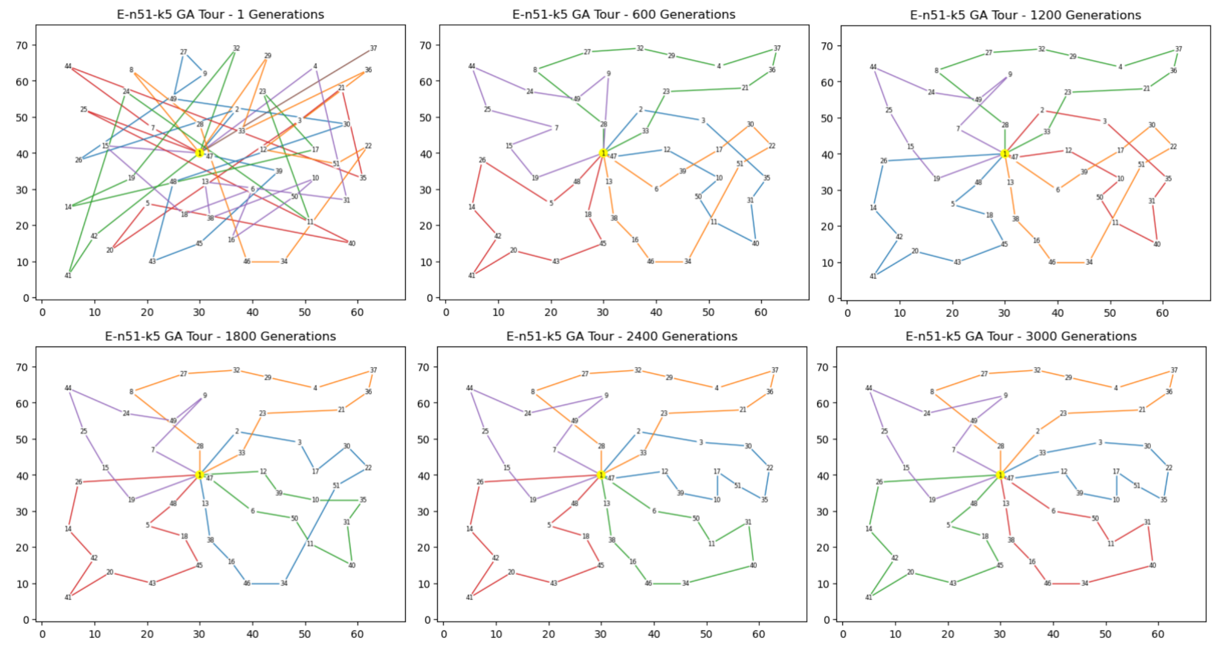

Figure 5. Demonstration of how optimal routes change over generations using the GA using benchmark E-n51-k5.

I used Excel to create tables from the .csv files to compare the optimality, runtime, and several other factors for GA and WoAC (Table 1). I also compared the known optimal solution for each instance with the experimental optimal for GA and WoAC (Table 2) and the effect of starting population on optimality and runtime for the GA (Table 3).

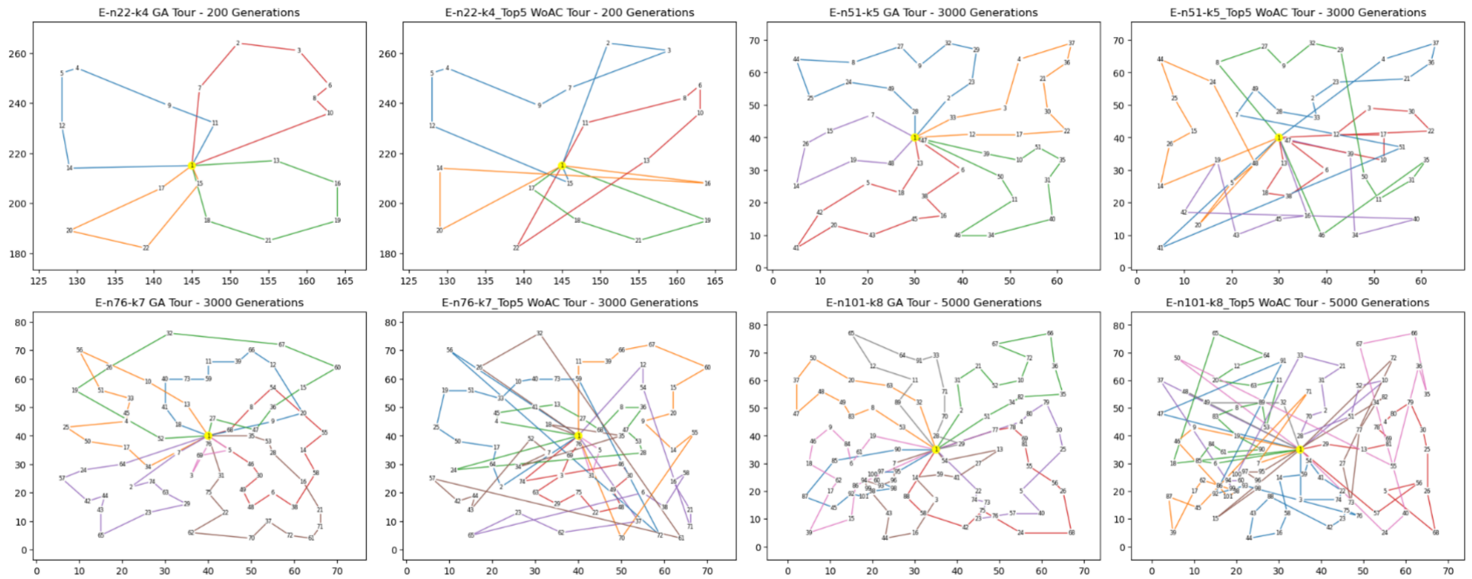

Figure 6. Most optimal solution found by GA and GA+WoAC for the benchmarks.

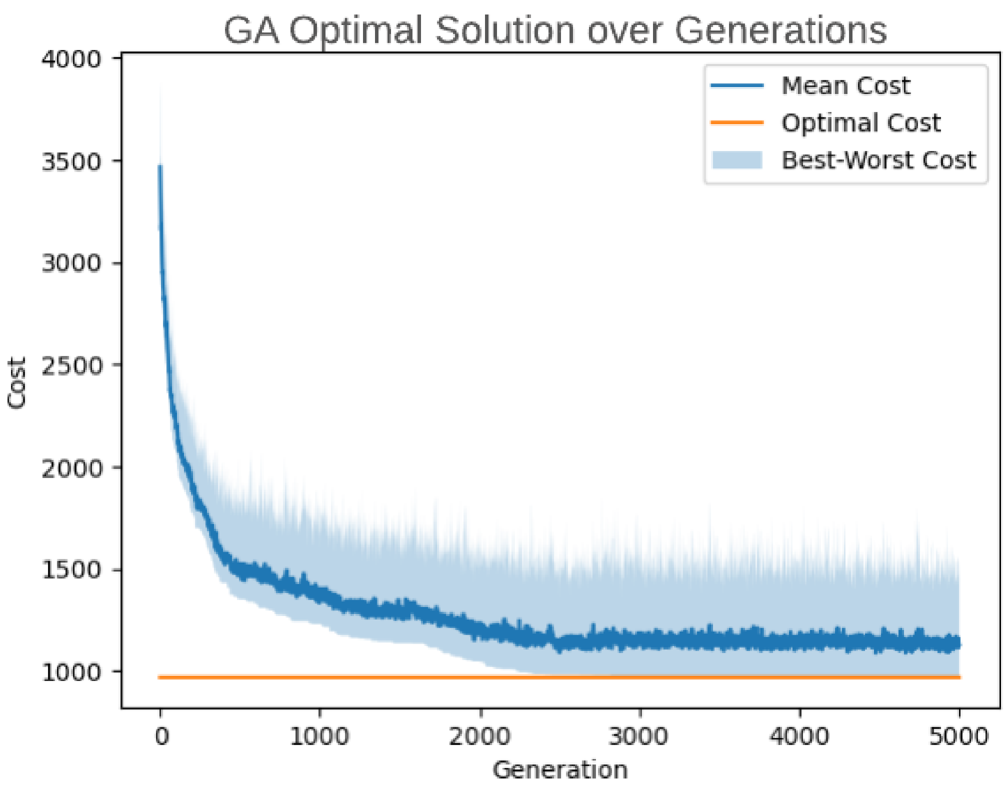

Figure 7. Instance E-n101-k8's cost changing over generations of the GA.

Algorithm	**GA**				**WoAC**			
Instance	**E-n22-k4**	**E-n51-k5**	**E-n76-k7**	**E-n101-k8**	**E-n22-k4**	**E-n51-k5**	**E-n76-k7**	**E-n101-k8**
Generations	200	3000	3000	5000	N/A	N/A	N/A	N/A
Best	**375.28**	**550.8**	**798.8**	**969.19**	495.76	837.05	1329.69	1713.75
Avg.	381.85	579.04	829.02	1075.04	495.76	837.05	1329.69	1713.75
Std Dev	3.7	18.03	18.78	61.59	0	0	0	0
Vehicles	4	5	7	8	4	5	7	8
Avg. Runtime GA (s)	9.34	37.5	61.33	121.39	0.00027	0.00001	0.00010	0.00014
Total Runtime GA (s)	93.43	374.97	613.31	1213.86	0.00027	0.00005	0.00010	0.00014
Elites Used	N/A	N/A	N/A	N/A	5	5	5	5
% Difference	N/A	N/A	N/A	N/A	-32%	-52%	-66%	-77%
Total Runtime GA+WoAC (s)	N/A	N/A	N/A	N/A	93.70	375.02	613.42	1214.00

Table 3. Experimental results for benchmark instances using the GA and the WoAC.

Instance	**Optimal**	**Minimum Number of Vehicles (K)**	**Best GA**	**% Difference GA**	**Vehicles GA**	**Best WoAC**	**% Difference WoAC**	**Vehicles WoAC**
E-n22-k4	375	4	375.28	0%	4	495.76	-32%	4
E-n51-k5	521	5	550.8	-6%	5	837.05	-61%	5
E-n76-k7	682	7	798.8	-17%	7	1329.69	-95%	7
E-n101-k8	815	8	969.19	-19%	8	1713.75	-110%	8

Table 4. Comparison of experimental results with the optimal solution to the benchmarks.

Population	**100**				**500**			
Generations	**10**	**50**	**100**	**1000**	**10**	**50**	**100**	**1000**
Best GA	**1186.75**	764.52	**598.9**	565.37	1216.1	**747.22**	604.33	**538.06**
Avg. GA	1305.18	858.48	658.72	617.06	**1260.98**	**783.74**	**623.7**	**587.5**
Avg. Runtime GA (s)	0.12	1.19	5.26	10.42	0.77	8.98	46.51	86.67
Best WoAC	1149.9	898.56	**793.47**	970.07	**966.31**	**852.56**	817.69	**778.9**
Avg. WoAC	1199.82	944.49	855.01	970.07	**1069.33**	**904.29**	**830.41**	**789.37**
Avg. Runtime WoAC (ms)	0.04	0.05	0.05	0.05	0.64	2.09	0.61	0.63
% Difference	3%	-18%	**-32%**	-72%	**21%**	**-14%**	-35%	**-45%**

Table 5. Table comparing the effect of population size on runtime and optimality based on E-n51-k5. Bold values indicate the best value for each value of generations for either value of population.

Discussion

Yampolskiy, Ashby, and Hassan’s WoAC metaheuristic I modified creates a valid solution without the need to alter it. However, it generally does much worse than the GA alone. One reason may be that CVRP requires optimizing two or more routes. For the TSP, the agreement matrix is used to build the route from the inside out, adding new edges to the start or end. The first edges tend to have the highest agreement, but the remaining edges with nodes that have yet to be added become less optimal, leading the metaheuristic to add less optimal edges over time. The same is true for the CVRP, except the process repeats itself over multiple routes, where locally optimal edges are selected, and the remainder become less optimal. Another possibility is that changing the metaheuristic to always add to the end of a route limited it to less optimal solutions than may have exists at the start. Future research should investigate new heuristics for combining solutions from the agreement matrix.

The number of customers appears to factor into the % Difference between Best GA and Best WoAC. Best WoAC did as good as Best GA for E-n22-k4 (0%) but performed worse on the remaining benchmark instances, all of which increases in size (Table 3). Likewise, % Difference GA (1 minus the cost ratio of Best GA and Optimal) and % Difference WoAC (1 minus the cost ratio of Best WoAC and Optimal) became progressively worse as number of customers increased (Table 4). Comparing the Tightness values in Table 1 to the % Difference GA in Table 4 suggests Tightness had little or no visible effect on the optimality of solutions, or that Tightness values themselves were too alike to reveal differences. While % Difference GA was 0% for E-n22-k4 with Tightness 94%, it was -17% for E-n76-k7 with Tightness 89%. However, the sample size is too small to draw meaningful conclusions. Future research should increase the number of problem instances tested on to yield more meaningful results.

In terms of speed, the Avg. Runtime GA is slower than Avg. Runtime WoAC (table 3). However, WoAC relies on GA to aggregate solutions. Because of this, combining GA and WoAC only increases runtime by milliseconds. Hypothetically, if fewer generations could be used for GA while getting the same or better solution by applying WoAC afterward, then the hybrid method may perform faster. As it is, I was unable to optimize the program to achieve a reduction in running time relative to GA alone. Without thoroughly testing the hypothesis, I did find that the % Difference for population sizes 100 and 500 were both better when GA ran for fewer generations on E-n51-k5 (Table 5). One possibility is that chromosomes converge the more generations GA runs, and that there are fewer local improvements to make when combining the solution.

Ideally, GA could be run for more generations to arrive at a better solution. While this does not correct the WoAC’s shortcomings, it would give more optimal solutions from which to aggregate and combine into new solutions. In reality, the number of generations does not guarantee a more optimal solution; Figure 3 shows how the minimum cost plateaus after so many generations, despite not reaching the optimal (969.19 vs 815). Increasing the solution space by increasing the population size may be necessary to see improvements from increasing number of generations run.

Two experimental constraints were the computer used to run the metaheuristic and a bottleneck in the GA itself. The computer used is a 4-year-old Asus ZenBook laptop, model UX330UAK, with an Intel Core i5-7200 CPU 2.50-2.71 GHz and 8 GB of RAM. Additionally, larger population sizes increase the GA running by minutes for larger problems but find more optimal solutions than smaller population sizes. To run the metaheuristic in a reasonable amount of time, I used a population size of 100 to generate results and statistics.

Table 5 shows how a population size of 500 has a greater runtime, but more optimal solutions, all else being equal. Population size 500 performed better than population size 100 for Best GA and Best WoAC (see bolded values in Table 5). One reason a larger population size may improve optimality is that the solution space (the initial population) is larger and new routes are possible. A newer laptop with greater processing power would have run the algorithms in less time, allowing me to test larger population sizes over many iterations in minutes rather than an hour at a time.

References

[1] G. B. Dantzig and J. H. Ramser, "The Truck Dispatching Problem," *Management Science,* vol. 6, no. 1, pp. 80-91, 1959.

[2] S. N. Kumar and R. Panneerselvam, "A Survey on the Vehicle Routing Problem and Its Variants," *Intelligent Information Management,* vol. 4, no. 3, pp. 66-74, 2012.

[3] N. Christofides and S. Eilon, "An Algorithm for the Vehicle-Dispatching Problem," *Operational Research Society,* vol. 20, no. 3, pp. 309-318, 1969.

[4] H. Nazif and L. S. Lee, "Optimised crossover genetic algorithm for capacitated vehicle routing problem," *Applied Mathematical Modeling,* vol. 36, no. 5, pp. 2110-2117, 2012.

[5] A. K. M. F. Ahmed and J. U. Sun, "Bilayer Local Search Enhanced Particle Swarm Optimization for the Capacitated Vehicle Routing Problem," *Algorithms,* vol. 11, no. 3, pp. 31-52, 2012.

[6] L. Rosenberg, D. Baltaxe and N. Pescetelli, "Crowds vs Swarms, a Comparison of Intelligence," in *Swarm/Human Blended Intelligence Workshop (SHBI),* Cleveland, 2016.

[7] R. V. Yampolskiy, L. Ashby and L. Hassan, "Wisdom of Artificial Crowds—A Metaheuristic Algorithm," *Journal of Intelligent Learning Systems and Applications,* vol. 4, no. 2, pp. 98-107, 2012.

[8] I. Xavier, "CVRPLIB - All Instances," 11 11 2021. [Online]. Available: http://vrp.atd-lab.inf.puc-rio.br/index.php/en/. [Accessed 12 11 2021].

[9] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, A. Subramanian and T. Vidal, "New Benchmark Instances for the Capacitated Vehicle Routing Problem," *European Journal of Operational Research,* vol. 257, no. 3, pp. 845-858, 2016.

[10] G. Reinelt, "TSPLIB-A Traveling Salesman Problem Library," *ORSA Journal on Computing,* vol. 3, no. 4, pp. 376-384, 1991.

[11] G. Laporte, "Fifty Years of Vehicle Routing," *Transportation Science,* vol. 43, no. 4, pp. 408-416, 2009.

[12] S. Gunawan, N. Susyanto and S. Bahar, "Vehicle Routing Problem with Pick-Up and Deliveries using Genetic Algorithm in Express Delivery Services," in *AIP Conference Proceedings*, Yogyakarta, 2019.