Solving the Vehicle Routing Problem with Multiple Multi-Capacity Vehicles

Michael S. Sanders, Jr.
Computer Systems Lab, 2008-2009
Thomas Jefferson High School for Science and Technology
Alexandria, Virginia

June 9, 2009

Abstract

The Vehicle Routing Problem (VRP) has existed as long as a distributor has needed to deliver items. As such, the VRP has been solved with many different methods, including agent architecture and ant colony modeling. However, these methods have generally been set up for an established organization that has a specific number of vehicles with only a few unique capacities. This project aims to create a program that will solve the VRP, but in a case where all the vehicles could have different capacities. This is the situation faced by some volunteer groups that do not have established vehicle fleets and rely on people volunteering vehicles when something needs to be distributed.

Keywords: Vehicle Routing Problem, heuristics, genetic algorithms

1 Introduction

The goal of this project is to create a program to quickly find the most efficient routing of a given number of vehicles with differing capacities to a variety of delivery points with a variety of demands of product. This is a majority applied research project. It is being created to assist a volunteer group in their distribution of goods.

The project is created linearly. The first step is the creation of a route finder, which includes the locating and processing of road data. The route finder is built upon through the addition of a heuristic algorithm to decrease the time required to find a route. The second major step is then to create a solution finder. The solution finder will implement another heuristic algorithm to determine a very good route. Neither the route finder or solution finder will be expected to routinely return the best results, given the scope of the problem. It is expected that this deficiency will be compensated for by human intuition, as the drivers of the vehicles can be expected to be familiar with the area road network.

2 Background

A great deal of research has been done into the VRP and its variants, such as the VRP with Time Windows (VRPTW) and Multi-Depot VRP (MDVRP). Projects have looked into using agent architecture and ant colony optimization to solve the problems. These projects have yielded such ideas as Clarke-Wright Algorithm to place delivery points into routes. Mennell, Shmygelska, and Thangiah did a project to evaluate using agents to solve the VRP. The program used agents to represent the vehicles and "auctioneers" that informed the vehicles of the current situation with regards to customers and deliveries. While the resulting program did not find the optimal solutions for any the sample problems, it was extremely adaptable for the MDVRP and VRP with multi-capacity vehicles.

3 Development

The project has two major components: the route finder and the solution finder.

3.1 Route Finder

The route finder was ultimately unsuccessful. However, it did result in insights as to challenges with desiging route finding software.

3.2 Route Finder Development

The route finder began as an extension of an assignment from a previous computer science course. Unfortuntely, due to data reliability issues, the route finder could not be programmed as intended. Data was obtained from the US Census Bureau's 2006 TigerLINE road database. An A* search was utilized for the attempted program. The program could partially function, but repeated searches would result in the program ceasing to function.

3.2.1 A* Search

The route finder implements an A* search for finding a route. After identifying the starting and ending points, the program then begins to iterate through paths. A path is an array containing a list of all the geographic coordinates that that path has passed through. The program then determines what streets intersect at the path's current location. It then assembles new paths by copying the current path and adding on each intersecting street. The program then determines how long each path is (how far the path has traveled from the starting point) and estimates how much farther the path has to go before it reaches the target. Once it has created all the new paths, the program sorts all the paths based on estimated total length.

The goal of the A* search is to reduce time used for searching by identifying the paths that are most likely to be the shortest routes. There are limitations, however. The A* search that was programmed estimated the remaining distance to the destination through the distance formula using latitude and longitude. The A* search, in this context, would work if all roads had the same speed limit. Obviously, this is very rarely the case. A higher speed limit may result in a road that is longer taking a shorter time to traverse than a shorter road, if the first road has a higher speed limit than the shorter road. This fact renders a geographic heuristic useless, and requires the development of a heuristic that can work around this problem.

3.3 Solution Finder

The solution finder is designed to return a better-than-average set of routes that efficiently utilize available vehicles and delivers the product to all de-

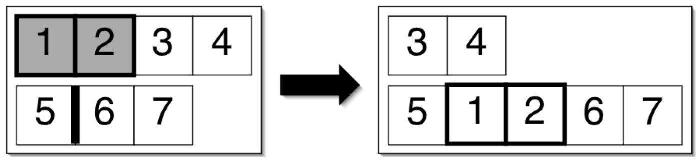


Figure 1: Displacement operation

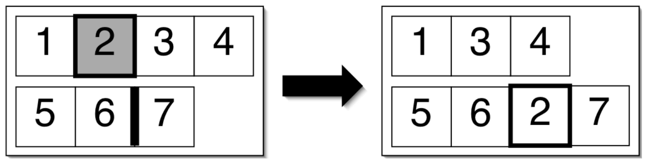


Figure 2: Insertion operation

livery points. It has two classes, the Route class and the Solution class. A Solution object has an array of Route objects. In terms of genetic algorithms, the Route objects are the "genes" of the Solution object. The solution finder will have a certain number of Solution objects that will be manipulated to find an acceptable result. A Route object has a vehicle assigned and contains an array listing all the deliveries on that route.

3.3.1 Genetic Operators

These are various operators that can be used to perform genetic mutations. Customers can be displaced from routes, deleted from one route and inserted in another, or swapped with other customers. Routes can also be randomized.

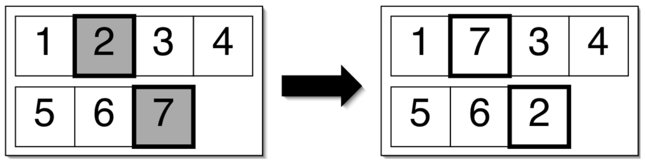


Figure 3: Swap operation

3.3.2 Initializing

The program initially creates a list of customers. From that list, it randomly selects a customer and tries to add it to the current route. The only constraint is that the addition of the customer must not cause the vehicle's capacity to be exceeded. If it is successful (i.e. the vehicle's capacity is greater than the sum of the assigned orders), then it proceeds to the next customer. If it is not, it then creates a new route and assigns the next vehicle. It then repeats this process until all customers are added. The solution is then copied nine times into new solutions.

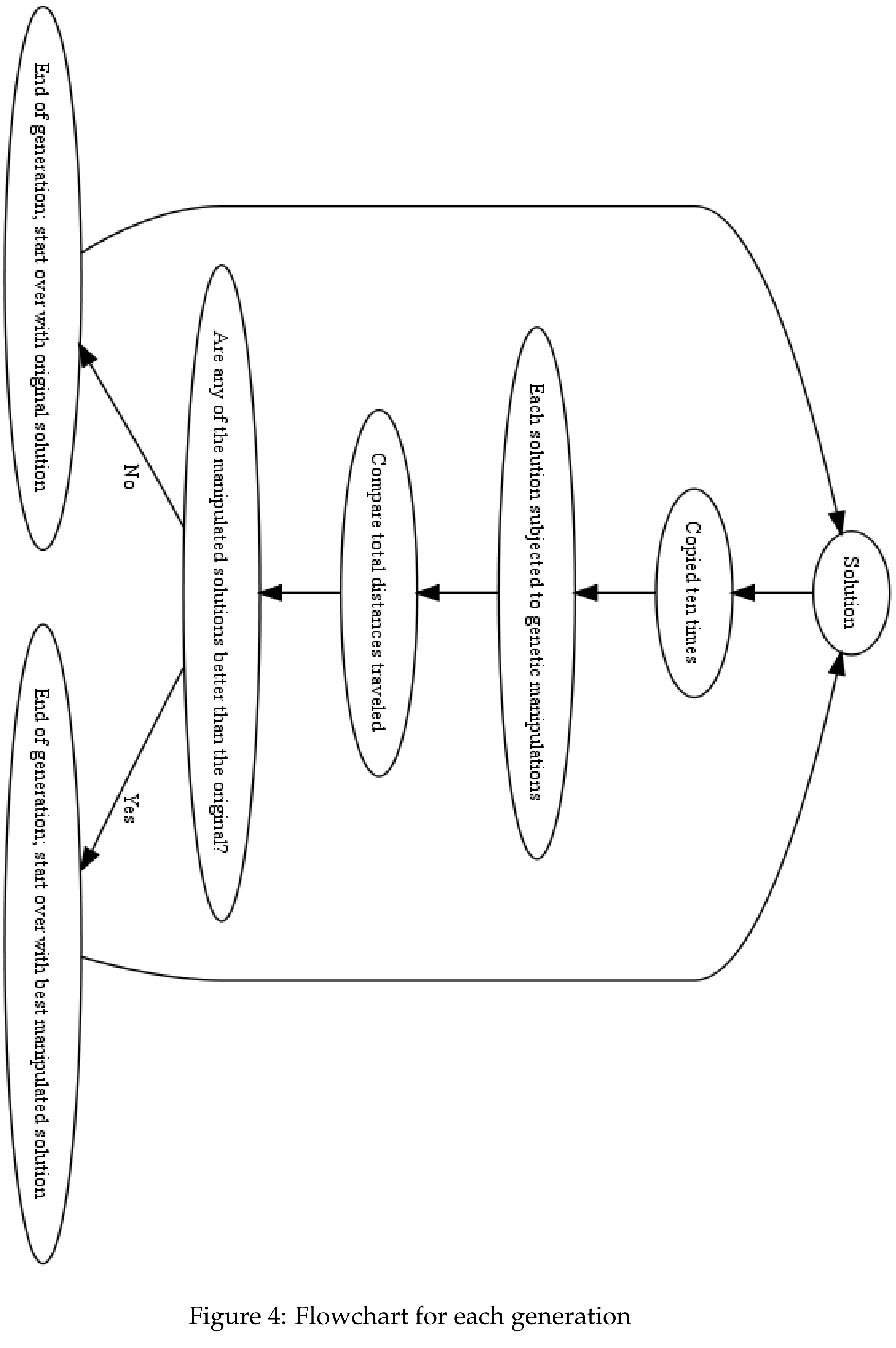
3.3.3 Manipulating

The program then begins iterating through generations. Each of the ten potential solutions is assigned a particular set of manipulations, as shown in table (1). The manipulations that had the highest success rate as determined by testing were assigned to two solutions each, and the other manipulations were assigned one solution each.

During each generation, the ten solutions that will be manipulated are copied from an original solution. The original solution is the best solution from the previous generation. The ten copied solutions are then run through their appropriate manipulations.

Each time a solution is run through the manipulations, the total distance traveled over all its routes is recalculated and stored. Once all solutions have been manipulated, the distances traveled of each of the solutions are compared. If none of the newly manipulated solutions are better (i.e. have a lesser sum of distances than) the original solution from which they were copied, then the generation terminates and the original solution remains the same. If, however, any of the newly manipulated solutions do have a better value (i.e. a lesser sum) than the original solution, then this new solution is copied as the original. The generation then terminates and proceeds to the next generation.

Upon reaching the specified number of generations, the program stops mutating solutions. The current original solution is outputted, as well as its improvement over the initial, randomly created solution. This improvement is expressed as a percentage of the distance traveled using the initial solution.



Solution
Copied ten times
Each solution subjected to genetic manipulations
Compare total distances traveled
Are any of the manipulated solutions better than the original?
No
Yes
End of generation; start over with original solution
End of generation; start over with best manipulated solution

Figure 4: Flowchart for each generation

3.4 Heuristics

This program attempts to replicate a person's job. Therefore, it has been given certain guidelines in terms of how to evaluate results intelligently. The guideline is called a heuristic, and is also known as a fitness function. For the solution finder to work, a function to determine how "fit" a solution is needs to be created. For this project, the fitness function f is

$$f = \frac{DT}{PD} \tag{1}$$

where DT is the sum of the distances traveled on each route in the solution, and PD is the total product delivered over the routes. The goal of this program is to minimize the result of f. Since, for a given situation the product delivered would be identical, the effect of this goal is to minimize distance traveled.

This function could be modified in serveral manners. Single variables, such as time spent traveling, could replace distance traveled. A more complex function that takes into account several variables, such as time, distance, and vehicle fuel economy, could be used that would weight each of the factors. The denominator could also be replaced by a quantitative value that needs to be maximized, such as number of stops per route.

4 Testing

Testing required two steps. First, each set of genetic manipulations were evaluated separately to determine each set's ability to find a better solution. Then, the top three sets of manipulations were assigned to two solutions each, and the remaining four sets assigned to one solution each. This set of ten solutions was evaluated many times using various combinations of the ten solutions. Each of these combinations is tested several times and the results averaged. The best combination of manipulations was selected for the final version of the project. See Appendix A for all tables and results.

5 Results

The goal of this project was to create a program to assist volunteer groups and other organizations that need to deliver items but do not maintain standarized fleets of vehicles. The object was not to find the best solution, but one that would be significantly better than what a human team could do, as well as remove the burden of such a laborous task from volunteer personnel. In this regards, it is successful. The core program works robustly, and simply needs to have output formatted.

From a research standpoint, this project was a success. A way to apply genetic algorithms to this version of the Vehicle Routing Problem was found. The program consistently offers improvements between 10% and 15% over the initial, random solution. Extensions of this project could look into improving several variables that regulate the likelyhood of a particular route being manipulated, or have the program start from a better position by more intelligently creating the first solution.

Appendix A. Tables

For tables (2) and (4), the numbers to the right of "Run #" indicate which set of manipulations was being tested in that run. They can be referenced in tables (3) and (5), respectively. For tables indicating test results, the numbers should be interpreted as percentages in decimal format. They were found by dividing the distance of the best solution found with the given manipulations by the distance of the inital, randomly created solution..

References

[1] B. Yu, Z. Yang, and B. Yao, "An improved ant colony optimization for vehicle routingnext term problem ", *European Journal of Operational Research*, pp. 171-176, 1 July 2009.

[2] D. Coltorti and A. E. Rizzoli, "Ant Colony Optimization for Real-World Vehicle Routing Problems", *SIGEVOlution*, pp. 2-9, Summer 2007.

Table 1: List of solutions assigned to each set of manipultions

Type of manipulation	Number of solution arrays
Swap	1
Randomize	1
Insertion	2
Swap+Insertion	2
Swap+Randomize	1
Insertion+Randomize	1
Swap+Insertion+Randomize	2

Table 2: Results of each manipulation applied singly (10,000 generations)

Run #	1	2	3	4	5	6	7
1	.921	.833	.930	.890	.933	.884	.901
2	.920	.875	.920	.915	.901	.894	.904
3	.905	.863	.938	.892	.937	.924	.913
4	.935	.862	.945	.850	.890	.944	.929
Avg.	.920	.858	.933	.887	.915	.912	.912

Table 3: Index table for table (2)

1	Swap
2	Insert
3	Random
4	Swap+Insert
5	Swap+Randomize
6	Insert+Randomize
7	All

Table 4: Results of combinations of manipulations (10,000 generations)

Run #	1	2	3	4	5	6
1	.872	.833	.857	.868	.868	.863
2	.885	.857	.861	.871	.912	.863
3	.886	.861	.873	.893	.923	.870
4	.900	.879	.879	.894	.930	.894
Avg.	.886	.858	.868	.882	.908	.873

[3] X. Gao and L. J. Schulman, “On A Capacitated Multivehicle Routing Problem”, *PODC '08*, pp. 175-184, 2008.

[4] M. Geiger, “Genetic Algorithms for multiple objective vehicle routing”, *MIC'2001*, pp. 349-352, 2001.

[5] B. A. Julstrom, “Greedy, Genetic, and Greedy Genetic Algorithms for the Quadritic Knapsack Problem”, *GECCO '05*, pp. 607-614, 2005.

[6] G. Lamont and M. Russell, ”A Genetic Algorithm for Unmanned Aerial Vehicle Routing”, *GEECO'05*, pp. 1523-1530, 2005

[7] H. W. Leong and M. Liu, “A Multi-Agent Algorithm for Vehicle Routing Problem with Time Window”, *SAC '06*, pp. 106-111, 2006.

[8] J. Tavares, P. Machado, F. Pereira, and E. Costa, “On the Influence of GVR in Vehicle Routing”, *SAC 2003*, pp. 753-758, 2003.

[9] S. R. Thangiah, O. Shmygelska, and W. Mennell, “An Agent Architecture for Vehicle Routing Problem”, *SAC 2001*, pp. 517-521, 2001.

Table 5: Index table for table (4)

1	1 Swap+Insert
2	2 Swap+Insert
3	2 Insert; 2 Swap+Insert
4	1 Insert; 1 Swap+Insert
5	All 10 Solutions
6	1 Insert; 2 Swap+Insert