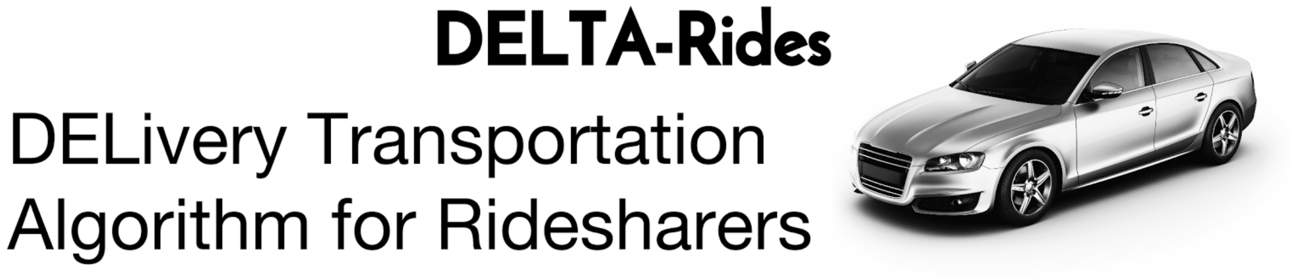
DELTA-Rides: DELivery Transportation Algorithm for Ridesharers



Imam Widodo and Ertugrul Oksuz

Supervisor: Mehmet Candas

Email: mehmet.candas@austin.utexas.edu

Harmony School of Political Science and Communication

Principal's email: iyilmaz@harmonytx.org

School phone: (512) 284-9880

2016

Dedication and Acknowledgements page

We are grateful to the following people and groups who have assisted and encouraged the development of our project:

Ahmet Taslama

Mehmet Candas

Aziz Koyuncu

Izzettin Aslan

Harmony School of Political Science

&

Our Loving Parents

Table of Contents

Abstract

Our project's purpose was to create an algorithm capable of ferrying attendees from their homes to an event location in the most efficient way possible through carpooling. Our original goal was to incentivize people to use carpooling by optimizing it, and we have further improved on our project by modifying several aspects and increasing its effectiveness.

After detailing our objective to create a simple application that would make it easy for attendees to enter in information and get back the best route, we started to research the problem and decided to create our heuristic based off of Greedy Heuristic logic. The difference between the classic Greedy Heuristic logic and our adaptation of it is that our version accounts for the heuristic's problem regarding getting stuck in local optimums through a series of switching the order in which passengers are picked up in each driver's list then swapping the passengers between different drivers. Finally, this is all repeated several times to guarantee that we have found a very efficient, if not perfect, routes possible.

Once tested, the algorithm we created surpassed our hypothesis and makes travelling and carpooling faster and more efficient by about 65% improvement on average for every case, which in some cases is an improvement of about 10% over our previous program. Our program also has several applications outside of carpooling (like delivery), and we hope to eventually make our program accessible for all to use so that we can save money and time during commutes.

Introduction

In our project, we wrote our own mathematical algorithm to find a more efficient solution to carpooling by making it easier and less time consuming. This problem that we focused on is actually a well known mathematical problem called the Traveling Salesman Problem. The Traveling Salesman Problem (TSP) is an NP hard math problem, meaning that there is almost infinitely many solutions to this problem based on how many combinations could come out of the inputted number of locations. For instance let's say there are 6 locations, the number of combinations that come out of these 6 locations would equal 6! (which is equal to 6x5x4x3x2x1=720). There are a total of 720 combinations that can come out of just 6 locations, so solving a problem involving 20 locations could take up to years even with today's fastest computers.

The motivation for this problem came to us after observing the many cars on the road and the rate at which gas prices are rising up. Because of the fact that most people aren't really willing to carpool, we wanted to create a system that would make carpooling more efficient and faster, and would therefore incentivize more people to carpool. Our objectives therefore were to create a simple web site that attendees would find easy to interact with and cluster each passenger to the closest available drivers so that the total distance that the driver must travel decreases to the lowest amount making it more efficient, faster and just less time consuming. Afterwards we use the haversine formula to find the distance between each and any given point on the surface of the globe. The way we do this is by finding the measure of the angle from the center of the Earth to two certain points on its surface and then multiplying the circumference of the globe by the measure of the degree between the two points (full definition in glossary).

Then comes the main part of this entire problem, the Traveling Salesman Problem. In this problem the Traveling Salesman wants to keep both the travel cost and the total distance as small as possible. This problem essentially has no real solution, there are only cheaper and more efficient solutions, which are what we are finding within our mathematical algorithm. The Traveling Salesman Problem was formulated originally in the 1800s by the Irish mathematician William Rowan Hamilton and an English mathematician named Thomas Kirkman. The original idea was the Hamiltonian cycle which was basically a recreational puzzle that was used to find a path that would touch up on each vertex exactly once.

Some other forms of approaching a cheap and efficient solution are such as the exact algorithms which this is the algorithm where all the points and combinations are being checked but like we already mentioned looking for the most efficient path with 6 passengers and 720 combinations will take a lot of time. Also this is just for 6 passengers whereas our algorithm deals with finding the most efficient route with at least 6 passengers. Then there is the nearest neighbor algorithm (NA) which goes to one point then to the closest unvisited next point from its current location. These methods are in today's modern world which unlike back in the 1900s these modern approaches to find the cheapest and shortest route within 100s of cities are closer to the most efficient route. Some algorithms, such as the brute force algorithm that analyzes every possible solution, would have given us better results, but would have taken too long too run and would have required a much more powerful computer. Through our research, we were able to narrow it down to the Greedy Randomized Adaptive Search Procedure (GRASP) and Genetic algorithms in order to optimize our initial path. The way that GRASP worked was basically as a modified version of the Greedy heuristic (defined in glossary) in that it still accepted the best possible answer but it was programmed to make sure that it did not get stuck in a local optimum. On the other hand, the genetic algorithm was set to imitate nature's process of natural selection by "breeding" (combining) different solution sets and "fitness" testing the result to see if the solution was more improved compared to the parents. After researching several search procedures, we decided that the Greedy Randomized Adaptive Search Procedure was one of the most efficient for our purposes and that the difference in results between the two would have been rather negligible as both had small advantages over the other.

Methodology

When we developed our algorithm, we aimed to accomplish three objectives: develop a web-based system to gather information from the drivers and passengers for car-pooling, cluster passengers to their drivers in a way that minimizes the total distance to travel, and find the shortest path for drivers to pick-up and deliver the passengers using our generated algorithm.

With these objectives, we developed our methodology to account for each of these requirements. First, we created a google forms document in order to gather information from our passengers and drivers such as their names, whether or not they would be able to volunteer, how many people they would be able to take if they can volunteer, and their address locations to be converted into latitude and longitude coordinates. This last part is important because currently our program calculates distances based off of a straight line to the destination using the haversine formula (full definition in glossary), which is a special distance formula that takes into account the curvature of Earth's surface. This approximate distance would be converted to the actual distance with Google Maps once the optimal route had been found (which also enables us to account for traffic and time).

Once we found the approximate distance between all of the attendees, we will then proceed to first cluster our passengers to our drivers in a way that will minimize the distance the drivers have to travel in order to pick up their passengers. To do this, we will first calculate the distance between each driver and passenger to find the distances between all of the drivers and the passengers. Next, we will then pair the passengers to the closest drivers around them until the driver capacity has been filled or all of the passengers have drivers. After the closest passenger to driver pairings have been made, a generated route will connect each driver to the passengers and bring them to the final destination. This route will act as an initial solution and will be further modified to become more efficient. Here, we have made sure that the program has the capacity to measure the distance each driver travels and add them up in order to find a final sum of a total distance that will be used to compare against other solutions.

Finally, we will move on to the most difficult part of the Traveling Salesman Problem and the heart of our program, the creation of a heuristic (defined in glossary) algorithm capable of finding an accurate answer to an np-hard problem and optimizing our previously created initial solution. As mentioned in our introduction, we chose to use the Greedy Heuristic because it would have taken less time than the other algorithms to run and it provides a very effective if not perfect solution.

In order to program our GRASP heuristic, we first created a pseudo code (defined in glossary) flowchart to dictate the characteristics of our code. We programmed our algorithm using the Java language because it is one of the most widely used coding languages and it is very effective at accomplishing the objectives we detailed earlier in a very efficient manner compared to other coding languages. After programming our code to find the best initial path for each driver in the pick up all of the passengers, we programmed our code to first pick two passengers randomly in a driver's list of passengers and swap the order in which they are picked up. Our code will then compare this new route and its resulting distance to the initial path's distance. If the code found that the route got shorter, it will keep the route and switch the order of two passengers again. If the code found that the route got longer, it reverted back to the previous shorter route and tried a new swap of two different passengers. This will keep looping on until no more better routes could be found. After this first loop, we decided to improve upon our program, but this time by searching for routes where drivers would switch their passengers. After our programmed switched two random passengers between drivers and compared the new distance, if it turned out that the distance indeed got shorter, our program went back into the first loop and ran the series of swaps again. These two loops will continue to cycle until our program has once again reached the maximum number of non-improving moves. Once we reached this point, we then wanted to make sure that our solution would not have been stuck in a *localized optimum* (a picture of an example can be found under Illustrations and Tables). We did this by programming our code to completely shuffle the passengers that the driver will pick up in order to start off with a completely new set of passengers for each driver. This is like having another person's perspective on the problem. With the closest passenger to path initial solution, we get one person's perspective on the problem, but by starting off with a completely random route as the initial solution, we can metaphorically get another person's perspective on the problem. The program will then proceed to run the loop of swaps and reverts until no more better solutions can be found just like before. At this point, the user may choose how many more shuffles they wish the program to perform in order to find the most efficient solution given the amount of time available to them and the power of their computer. Of course, a more powerful computer will be able to run more shuffles and swaps in a smaller amount of time than a less powerful one so we have left this option to the user so that they may decide what is best in their situation. After the amount of shuffles allowed by the user have been reached and the process of swaps and reverts have been done for each of these shuffles, the program will then proceed to compare the best results from each of these sets of shuffles (or perspectives) to try and find which final order that gives the globally most efficient distance.

The last step to our program is that once the globally best order has been found, the locations of the attendees will then be plotted out on a map and the routes for each driver to pick up their passengers will be displayed on the map to connect all of the drivers and passengers to the final destination. In addition, we enabled our program to create a google maps URL for each of the drivers and their passengers so that the final route for each driver will be mapped out in google maps and show the best route for the driver to take and pick up all of the passengers. At this point, we plan to further work on our program and enable it to send the results as an email to each of the attendees to let them know who will pick them up or who will be picking them up.

Results

To guarantee that our code was accurate, we created twenty different scenarios using fifty different locations (shown below) in our city and found the results our code would provide. On average, compared to a base case where everyone drove by themselves, we found that our code created overall savings of about 65% for each scenario, which translates to about 4 gallons/15 liters of gasoline and an average distance savings of 76 miles/122 kilometers for every scenario we created (below, we have attached a summary of every case and its results).

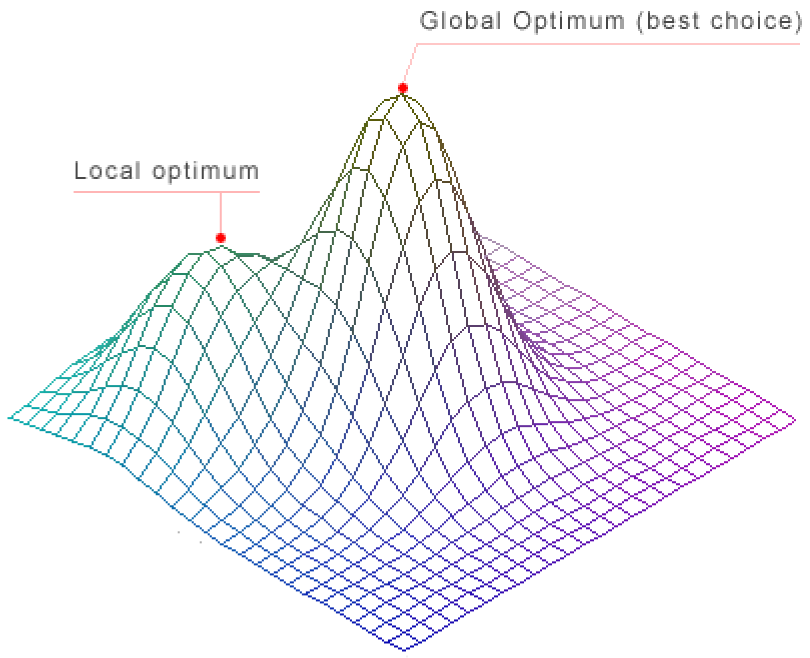
In addition to these quantitative savings, we also have qualitative savings. First of all, with less cars on the road due to the carpooling, there will be less gas used and therefore less gas emissions that can destroy the environment and the ozone layer (which slows global warming). Second, with less cars on the road, there will be less traffic congestion and stress, which I am sure every commuter will appreciate. Thirdly, with less cars on the road there will be less traffic accidents, which will also aid in reducing stress and the time it takes for commuters to reach their destinations. Finally, carpooling with others will give you more social interactions in a time when face to face social interactions are decreasing because of technology and cell phones. This alone will help improve your quality of life and the quality of life of those around you.

Summary of every case:

Case #	Base Case	ETRAM (previous yr travel distance)	DELTA (travel distance)	Improvement % (Base Case to: ETRAM / DELTA)	Saved Mileage	Saved Gas
1	25	13	13	48 %	12	.6 gal
2	65	26	20	60% / 69 %	39 / 45	~2 gal
3	81	26	26	68 %	55	2.75 gal
4	54	16	16	70 %	38	~2 gal
5	156	52	47	~67 % / 70%	104 / 109	~5 gal
6	117	34	28	~71 % / 76%	83 / 89	~4 gal
7	96	34	27	65 % / 72%	62 / 69	~3 gal
8	74	47	47	36 %	27	1.35 gal
9	128	54	54	58 %	74	3.7 gal
10	273	55	43	~80 % / 84%	218 / 230	~11 gal
11	205	72	70	~65 % / 66%	133 / 135	6.65 gal
12	59	34	34	~65 %	34	6.65 gal
13	257	80	76	~69 % / 70%	177 / 181	~9 gal
14	117	44	44	~62 %	73	3.65 gal
15	32	23	23	~28 %	9	0.45 gal
16	87	27	22	~69% / 74%	60 / 65	3 gal
17	185	45	39	~76% / 80%	140 / 146	7 gal
18	74	16	16	~78 %	58	~3 gal
19	72	35	32	~51% / 56%	37 / 40	~2 gal
20	152	62	58	~59 % / 60%	90 / 94	4.5 gal

Illustrations

Localized Optimum Example



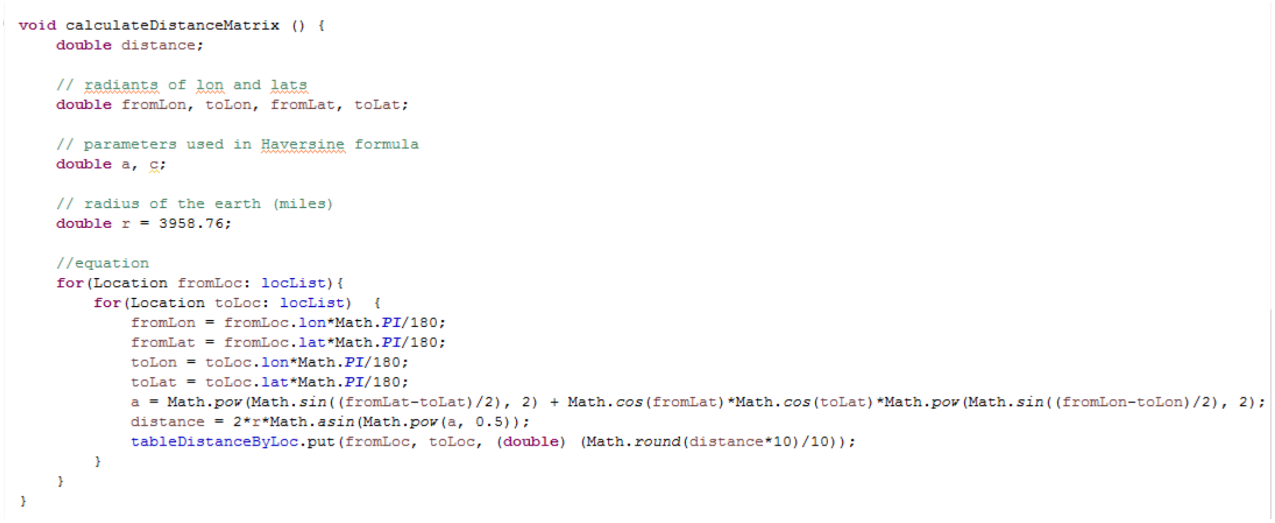
Snippets From Our Code

How our code reads an attendee:

Sample Input:
St1 John "12425 Fallen Tower Ln, Austin, TX 78753" 30.395087 -97.655854 No 0

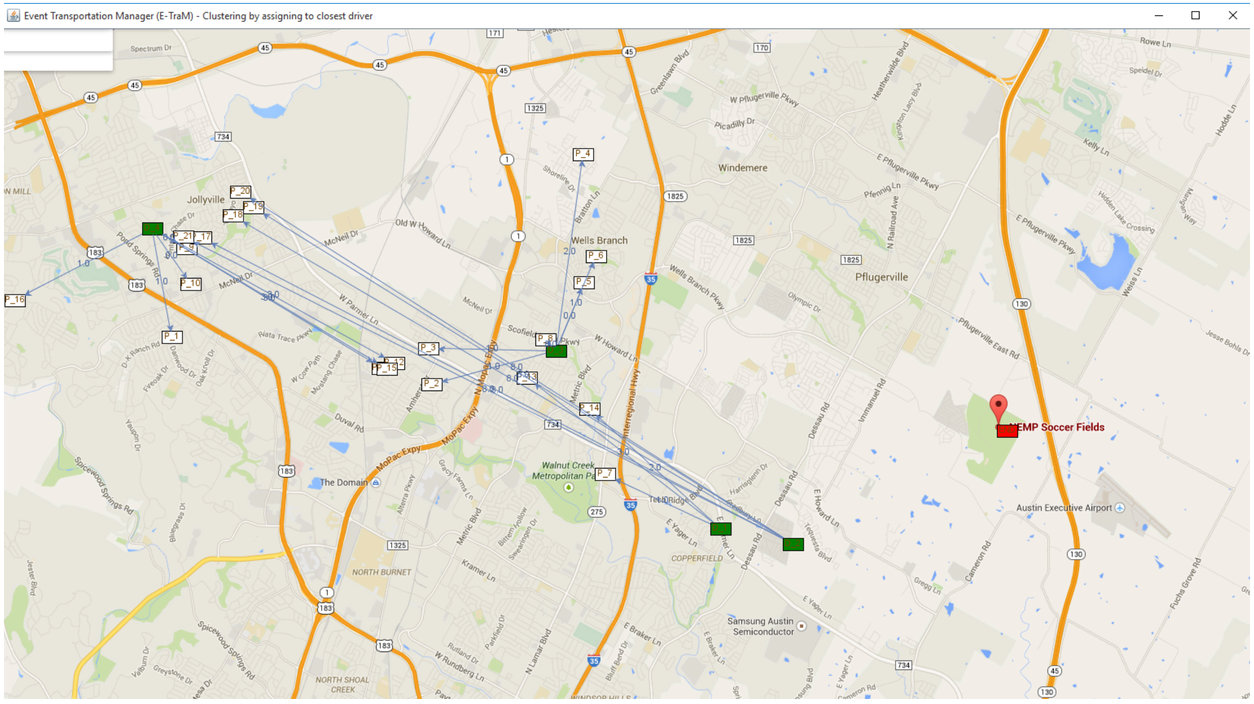
```
class Location {
        String locId; // Details the ID of the passenger when read in our code
        String locName; // Details the Name of the passenger
        String locType; // Details the type of location (driver, passenger, destination)
        double lat; // Details the Latitude
        double lon; // Details the Longitude
        int pickupCapacity; // Details the number of passengers able to pick up (passenger = 0)
}
```

How we programmed the Haversine Formula



```
void calculateDistanceMatrix () {
    double distance;

    // radiants of lon and lats
    double fromLon, toLon, fromLat, toLat;

    // parameters used in Haversine formula
    double a, c;

    // radius of the earth (miles)
    double r = 3958.76;

    //equation
    for(Location fromLoc: locList){
        for(Location toLoc: locList)  {
            fromLon = fromLoc.lon*Math.PI/180;
            fromLat = fromLoc.lat*Math.PI/180;
            toLon = toLoc.lon*Math.PI/180;
            toLat = toLoc.lat*Math.PI/180;
            a = Math.pow(Math.sin((fromLat-toLat)/2), 2) + Math.cos(fromLat)*Math.cos(toLat)*Math.pow(Math.sin((fromLon-toLon)/2), 2);
            distance = 2*r*Math.asin(Math.pow(a, 0.5));
            tableDistanceByLoc.put(fromLoc, toLoc, (double) (Math.round(distance*10)/10));
        }
    }
}
```

Clustering Example



Conclusion

After testing our project with our scenario cases, we found that our program was indeed capable of making carpooling and ride-sharing more efficient. With 60% improvement on average for every case, we have succeeded in creating a much more efficient way for commuters to save money during their daily drives while saving the environment and socializing at the same time. In addition, we had also succeeded in creating an algorithm capable of creating very good, if not perfect, solutions for the Traveling Salesman Problem.

Even though we had succeeded in fulfilling the tasks we had set out to accomplish, we plan to further develop our science fair project for next year. We first plan to incorporate Google Maps or a GPS system into our program so that we may find distances more accurately when calculating different routes. We also plan to modify the method by which our program will search for improvements in the paths by switching the passengers of different drivers so that we can see if this will generate and lead to more efficiency in the Greedy Randomised Adaptive Search Procedure. Furthermore, we plan to make our research available for the public to use and benefit from by developing a website and a phone application so that people looking to carpool may use our program to plan smarter and more efficient routes for themselves. With the rapidly increasing number of cars on the road, traffic gets very congested and the environment gets very polluted so by making carpooling easier to use, the main mission of our research is to inspire more people to carpool and help contribute to reducing the growing problem of traffic. We thank you for your kind attention in reading our research paper!

Abbreviations/Acronyms/Symbols/Glossary of important terms

Greedy Heuristic: Algorithm that follows the problem solving heuristic of making the locally optimal (best) choice at each stage with the hope of finding a global optimum

GRASP: Acronym for Greedy Randomized Adaptive Search Procedure

Globally best (distance/path): The shortest found path (best)

Haversine Formula: Calculates the distance between two points on the Earth's surface specified in longitude and latitude by finding the angle created by these two points and multiplying it by the Circumference of Earth

$$d = 2r\sin^{-1}\left(\sqrt{\sin^2\left(\frac{\phi_2-\phi_1}{2}\right)+\cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\psi_2-\psi_1}{2}\right)}\right)$$

Heuristic: Proceeding to a solution by trial and error and by defined rules

Java: A general-purpose computer programming language designed to produce programs that will run on any computer system

NP-hard problem: *n*on-deterministic *p*olynomial-time hard problem. It is at least as hard as any NP problem and if an algorithm for solving it can be found, it can be translated to solve other NP-Hard problems

Pseudocode: Informal high-level description of the operating principle of a computer program or other algorithm. Essentially looks like code itself but simplified

Bibliographic References

- Resende, M., & Ribeiro, C. (2003). Greedy Randomized Adaptive Search Procedures. *Handbook of Metaheuristics, 57*(0884-8289), 283-319. doi:10.1007/0-306-48056-5_8

- Abeel, T. (2008, December 16). Java Machine Learning Library (Java-ML). from http://java-ml.sourceforge.net/content/clustering-basics

- Malkevitch, J. (n.d.). Feature Column from the AMS. from http://www.ams.org/samplings/feature-column/fcarc-tsp

- Gidman, J. (2015, February 19). Algorithm Will Tell All UPS Trucks Where to Go. from http://www.supplychain247.com/article/algorithm_will_tell_all_ups_trucks_where_to_go

- Jacobson, L. (2013, April 11). Simulated Annealing for beginners. from http://www.theprojectspot.com/tutorial-post/simulated-annealing-algorithm-for-beginners/6