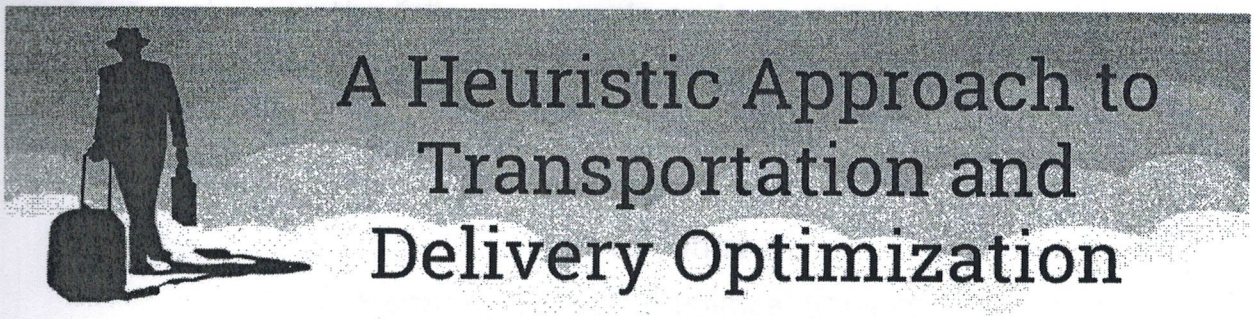


A Heuristic Approach to Transportation and Delivery Optimization

Fakhri W.

Supervisor: Dr. Mehmet Candas

Email: mehmet.candas@austin.utexas.edu

Harmony School of Political Science and Communication

Principal's email: wstengler@harmonytx.org

School phone: (512) 284-9880

2018

Dedication and Acknowledgements page

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

Dr. Mehmet Candas

Aziz Koyuncu

Harmony School of Political Science

&

The Loving Parents

Table of Contents

Dedication and Acknowledgements 2

Table of Contents 3

Abstract 4

Introduction 5-6

Methodology 7-10

Results 11-12
- Summary of every case

Illustrations 13-14
- Localized Optimum Example
- Snippets From The Code
- Clustering Example

Conclusion 15

Abbreviations/Acronyms/Symbols/Glossary of important terms 16

Bibliographic References 17

Abstract

The project's purpose was to create an algorithm capable of optimizing transportation/ delivery to an event location in the most efficient way possible. Through the implementation of a heuristic algorithm I also set out to devise a method to approximate an optimal solution to the Travelling Salesman Problem.

The motivation for my project was trying to figure out a more efficient way for first responders to get people to safety in emergency situations such as Hurricane Harvey and Hurricane Katrina. After detailing the objective to create an application that would return users the shortest route, I started to research the problem and decided to create the heuristic based off of modified GRASP logic. I chose this logic because it yields locally optimal solutions that approximate a global optimal solution in a reasonable time and it is cleaner than select others. My GRASP adaptation accounts for the Greedy heuristic's problem regarding getting stuck in local optimums, through a series of switching the order in which passengers are picked up and then swapping passengers between carriers.

Tested over 20 different scenarios, the algorithm surpassed hypotheses and made travelling much faster and efficient by about 65% improvement on average versus the base Greedy path, and which in some cases is an improvement of about 10% over another popular heuristic program, the 2-Opt model. With this immense savings in distance, time to calculate, and time to travel, more lives can be saved.

The algorithm is applicable for nearly every problem involving transportation optimization. In the future I plan to improve upon the current algorithm (priority, neural net) and compare it to other heuristics such as the genetic algorithm, and eventually I hope to make the program accessible for all to use so that everyone can save time, money, and lives.

Introduction

In the project, I wrote the mathematical algorithm to find a more efficient solution to emergency response by making it easier and less time consuming. This problem that I 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 based on the amount of input nodal points, the amount of possible best routes increases exponentially. The general equation to model the amount of possible unique routes is $(n-1)!/2$, so for instance let's say there are 6 locations, the number of combinations that come out of these 12 locations would equal $(11)!/2$ (which is equal to 19,958,400). There are a total of 19,958,400 combinations that can come out of just 12 locations, so solving a problem involving 20 locations could take up to years even with today's fastest computers.

The motivation for my project was trying to figure out a more efficient way for first responders to get people to safety in emergency situations such as Hurricane Harvey and Hurricane Katrina. The objectives therefore were to create an algorithm that would cluster each passenger to the closest available carriers so that the total distance that the carrier must travel decreases to the lowest amount making it more efficient, faster and just less time consuming. Afterwards I use the Haversine formula to find the distance between each and any given point on the surface of the globe. The way I 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 has essentially no real solution as the number of possible routes

increases exponentially with points, so there are only cheaper and more efficient solutions, which are what I are finding within the 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 I 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 the 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 the research, I were able to narrow it down to the Greedy Randomized Adaptive Search Procedure (GRASP) and Genetic algorithms in order to optimize the 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, I decided that the Greedy Randomized Adaptive Search Procedure was one of the most efficient for the 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 I developed the algorithm, I aimed to accomplish three objectives: develop a web-based system to gather information from the carriers and passengers, cluster passengers to their carriers in a way that minimizes the total distance to travel, and find the shortest path for carriers to pick-up and deliver the passengers using the generated algorithm.

With these objectives, I developed the methodology to account for each of these requirements. The algorithm takes addresses and finds their location in longitude and latitude. This last part is important because currently the 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 I found the approximate distance between all of the attendees, I will then proceed to first cluster the passengers to the carriers in a way that will minimize the distance the carriers have to travel in order to pick up their passengers. To do this, I will first calculate the distance between each carrier and passenger to find the distances between all of the carriers and the

passengers. Next, I will then pair the passengers to the closest carriers around them until the carrier capacity has been filled or all of the passengers have carriers. After the closest passenger-to-carrier pairings have been made, a generated route will connect each carrier 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, I have made sure that the program has the capacity to measure the distance each carrier 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, I will move on to the most difficult part of the Traveling Salesman Problem and the heart of the program, the creation of a heuristic (defined in glossary) algorithm capable of finding an accurate answer to an np-hard problem and optimizing the previously created initial solution. As mentioned in the introduction, I 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 the GRASP heuristic, I first created a pseudo code (defined in glossary) flowchart to dictate the characteristics of the code. I programmed the 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 I detailed earlier in a very efficient manner compared to other coding languages. After programming the code to find the best initial path for each carrier in the pick up all of the passengers, I programmed the code to first pick two passengers randomly in a carrier's list of passengers and swap the order in which they are picked up. The 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, I decided to improve upon the program, but this time by searching for routes where carriers would switch their passengers. After the programmed switched two random passengers between carriers and compared the new distance, if it turned out that the distance indeed got shorter, the program went back into the first loop and ran the series of swaps again. These two loops will continue to cycle until the program has once again reached the maximum number of non-improving moves. Once I reached this point, I then wanted to make sure that the solution would not have been stuck in a *localized optimum* (a picture of an example can be found under Illustrations and Tables). I did this by programming the code to completely shuffle the passengers that the carrier will pick up in order to start off with a completely new set of passengers for each carrier. This is like having another person's perspective on the problem. With the closest passenger to path initial solution, I get one person's perspective on the problem, but by starting off with a completely random route as the initial solution, I 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 cthese, 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 I 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 the 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 carrier to pick up their passengers will be displayed on the map to connect all of the carriers and passengers to the final destination. In addition, I enabled the program to create a google maps URL for each of the carriers and their passengers so that the final route for each carrier will be mapped out in google maps and show the best route for the carrier to take and pick up all of the passengers. At this point, I plan to further work on the 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 the code was accurate, I created twenty different scenarios using fifty different locations and found the results the code would provide. On average, compared to a base case where everyone drove by themselves, I found that the code created overall savings of about 65% average for each scenario over the Greedy algorithm and up to 10% improvement over the 2-Opt, translating to an average of 80 miles average savings from the scenarios.

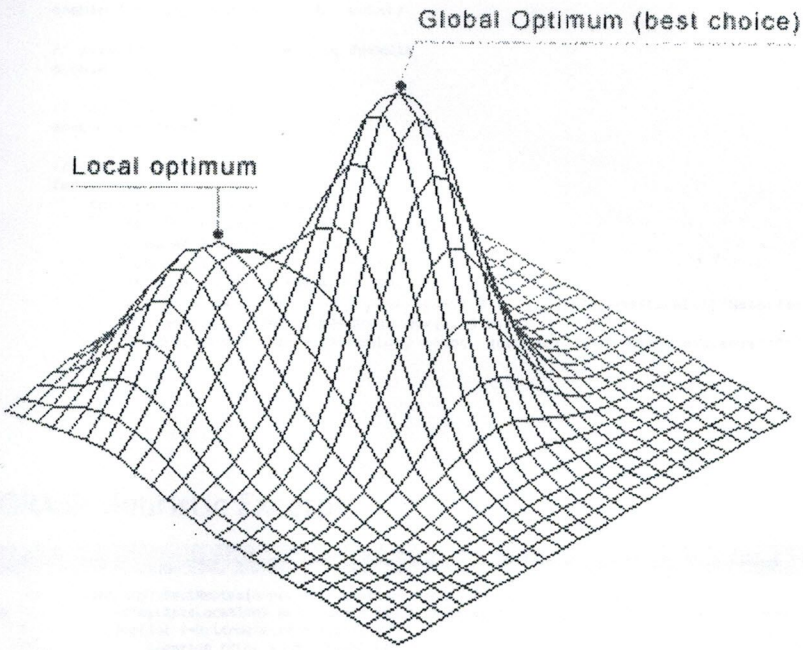
In addition to these quantitative savings, I also have qualitative savings. First of all, with a shorter distance and time to travel, more lives can be saved in emergency situations. Second, for general applications, there will be less congestion and stress with an optimized routing for every carrier. Finally, the algorithm is applicable for nearly every problem involving transportation optimization. In the future I plan to improve upon the current algorithm (priority, neural net) and compare it to other heuristics such as the genetic algorithm, and eventually I hope to make the program accessible for all to use so that everyone can save time, money, and lives.

Summary of every case:

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

Illustrations

Localized Optimum Example



Snippets From The Code

How the 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 the code
        String locName; // Details the Name of the passenger
        String locType; // Details the type of location (carrier, 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)
}
```

Haversine Formula Code

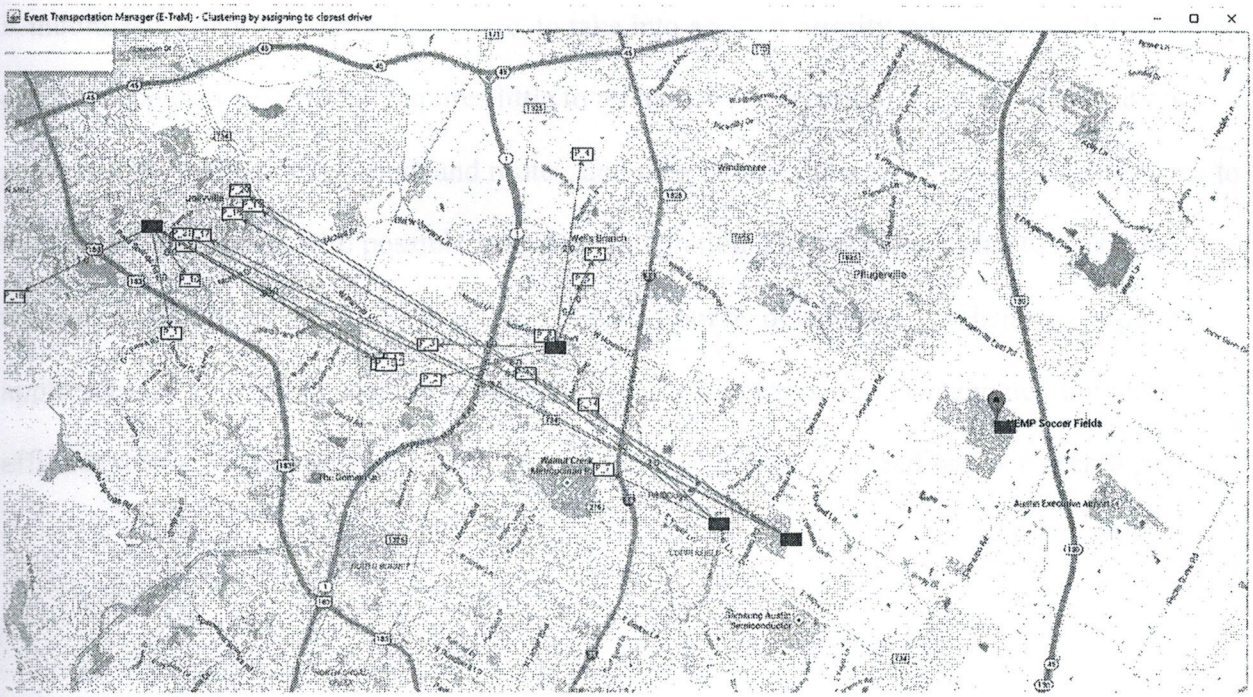
```
void calculateDistanceMatrix () {
    double distance;

    // radians 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));
        }
    }
}
```

GRASP Heuristic Excerpt

MainRunner.java | MainRunner.java | Locations.java | GRASPheuristic.java | Visualization.java

```
    void addToBestRoutes(ArrayList<Location> route) {
        ArrayList<Location> BestRoute = new ArrayList();
        for(int i=0;i<route.size();i++) {
            Location rtLoc = new Location();
            rtLoc = route.get(i);
            BestRoute.add(rtLoc);
        }
        driver.bestRoutes.add(BestRoute);
    }

    // Improve the path by doing random search (swapping and shuffling)
    void search() {
        try {

            // create a file "GeneticAlgorithm.out" to keep output
            String outFile = "./ouput/GreedyHeuristic_" + driver.locName +  ".out";
            File file = new File(outFile);
            // if file doesn't exists, then create it
            if (!file.exists()) {
                file.createNewFile();
            }
            FileWriter fw = new FileWriter(file.getAbsoluteFile());
            BufferedWriter bw = new BufferedWriter(fw);

            bw.write("Base distance: " +  Utils.getBaseTravelDistance(locs));

            int swapFrom, swapTo;
            boolean continueToRun=true;
            // initial travel distance for given travel order
            double travelDistance = Utils.calculateTotalDistance(locs, driver, driver.pickupList);
            //System.out.println(locs.printPath(locs.stList) + travelDistance);

            // initial optimal distance is equal to initial travel distance
            double bestDistance = travelDistance;
            double globalBestDistance = travelDistance;
            // add this route the list of best routes
            addToBestRoutes(driver.pickupList);

```

Clustering Example



Conclusion

After testing the project with the scenario cases, I found that the program was indeed capable of making routing through multiple nodes more efficient. With over 60% improvement on average for every case, I 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, the algorithm is capable of creating very good, if not perfect, solutions for the Traveling Salesman Problem.

Even though I had succeeded in fulfilling the tasks I had set out to accomplish, I plan to further develop the science fair project for next year. In the future, I plan to modify the method by which the program will search for improvements in the paths by switching the passengers of

different carriers so that I can see if this will generate and lead to more efficiency in the Greedy Randomized Adaptive Search Procedure, to take into account priority, and I also wish to develop a fully automated system that requires little to no human intervention; it gathers information, generates the optimal assignment and routes, and then returns the assignments and routes back to the carriers and nodes, with possible neural net integration. Furthermore, I plan to make the 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 the program to plan smarter and more efficient routes for themselves. I thank you for the kind attention in reading the 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: Cal-cu-lates the dis-tance between two points on the Earth's surface spe-cified in lon-gitude and latitude by finding the angle created by these two points and multiplying it by the Circumference of Earth

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

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.stheceforge.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