Generating Optimal Autonomous Vehicle Routes

Matriculation Number: s1879654

November 13, 2018

Abstract

Creating the best routes for self-driving cars acting like buses is difficult with so many constraints and variables to optimize. This report describes the problem in detail and how it can be solved using approximation algorithms, with an example using NYC taxi trip data and algorithms implemented in the Python programming language.

1 Introduction

As autonomous vehicles become closer to entering the mass market, cities must be able to utilize them effectively in order to capitalize on their potential cost savings. This paper will describe a methodology through which the most effective routes can be generated given historical location data from taxis already servicing a city. Let's begin by formally defining the problem we are trying to solve.

1.1 Problem Statement

Suppose we have k self-driving cars to deploy. Suppose we are also given pickup and dropoff locations (i.e. *two pairs* of latitude and longitude coordinates) representing taxi trips taken in the area over a set period of time (e.g. a year).

Let S represent the set of points corresponding to "stops" for these self-driving cars to drive between, picking up and dropping off passengers at each stop. The number of stops $|S|$ is some positive constant integer c_S. Let G be a completely connected, undirected graph with $V = S$. A valid route r is defined as any valid path you can find within graph G. Note that a path need not be cyclical (as the car can simply go along the same route backwards).

Furthermore, assume every traveller is willing to walk (from the pickup location) to a nearby stop at most a radius of some constant distance d metres away.

Now, our problem is as follows: how can we define routes for our k cars such that:

1. The stops maximize the number of travellers it can service (i.e. cover as many pickup/dropoff points within a circle of radius d with c_S stops), and

2. distribute the k self-driving cars such that at least one car will service every stop while minimizing the total distance that the k self-driving cars will drive (thus minimizing gas costs, the most significant variable cost for self-driving cars).

This problem is challenging because we are trying to optimize multiple variables at once, namely coverage and distance travelled by the self-driving cars. It turns out that several parts of this problem are in fact intractable; therefore, we will have to make several assumptions when devising a solution.

2 Related Work

Optimization of Bus Route Planning in Urban Commuter Networks is a paper by Steven I-Jy Chien, Branislav V. Dimitrijevic, and Lazar N. Spasovic about a similar topic. However, the biggest difference is that the paper attempts to optimize different variables (e.g. driver costs, trip times, size of network) and that they assume passengers can board a bus anywhere along its route, ignoring bus stops entirely.

The Maximum Coverage Problem, a thesis by Ymro Nils Hoogendoorn, covers various algorithms designed to create near-optimal solutions to the NP-hard problem. One particular algorithm, one involving the Lagrange Multiplier, is used in the solution.

Bus route planning in general is a well-studied topic as most city planners will have considered this problem. However, this paper focuses on different constraints and optimizes different costs due to the fact that we are dealing with *driverless* cars.

3 Solution

We attempt to solve this problem given a real dataset — taxi trips recorded by the NYC Taxi and Limousine Commission in 2013 — with the following assumptions:

3.1 Assumptions

1. The number of self-driving cars to deploy is $k = 4$.

2. The number of stops to place is $c_S = 20$.

3. The maximum distance the average person is willing to walk is $d = 319$ metres (taken from this article.

4. Our input location data specifies pickup or dropoff points, but we will treat them equally assuming there no large-scale patterns of movement from one area to another (i.e. everyone goes everywhere in NYC).

5. Cars have infinite capacity and car rides take a significantly better-than-taxiing time and cost, meaning as long as a bus stop is within a traveller's maximum willingness-to-walk radius, he/she will always choose to bus.

6. Stops will be placed *at* pickup/dropoff points, even though in real life this doesnt necessarily have to be the case (e.g. a stop could be somewhere between two pickup points).

All of the corresponding code to the solution can be found in the IPython notebook "STN-2018-Project". It has been annotated to match up with each section of the solution below. To ensure reasonably fast calculations, the notebook relies on a small 97-trip sample of the full 2013 taxi dataset. The full dataset is a massive 3.8GB (downloadable through the link at the beginning of the section), making it way too large for such heavy computation without the help of a cloud computing architecture.

3.1.1 Visualizing Taxi Data

First, to get a handle of the data, we must first visualize the pickup/dropoff points. Figure 1 below is what our small 96-trip sample looks like when plotted based on latitude and longitude, the extrinsic metrics.

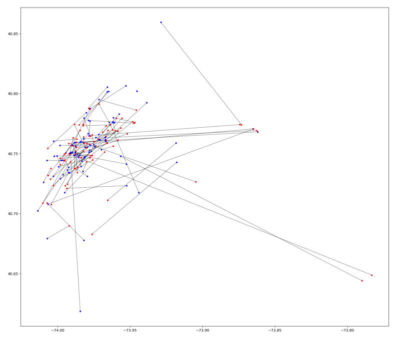

Figure 1: Pickup points (red nodes) and dropoff points (blue nodes), connected by an edge if they are part of the same taxi trip. Note how most nodes are concentrated in one area, meaning that we can treat pickup and dropoff points equally as travellers appear to go both ways anyway. For a much more detailed view, check the IPython notebook.

3.1.2 Coverage Created by a Potential Stop (Maximum Coverage Problem)

Now that we have a graph with n nodes — a node for every pickup/dropoff point — we can consider what subset of these n nodes of size c_s will maximize coverage of other pickup/dropoff points given coverage by any one stop is defined by a circle of radius d. This is also known as the Maximum Coverage Problem, which is unfortunately intractable (i.e. NP-hard). We can instead use a mathematical technique called Lagrangian Relaxation to get a decent approximation. An existing online Python implementation is used.

3.1.3 Generating Optimal Routes (Vehicle Routing Problem)

Lastly, now that we have chosen our stops S, we just need another algorithm to generate the best routes for our k self-driving cars in order to minimize total distance, pass by every stop at least once, and start/end at the same location (where the cars come to recharge at night). This problem is known as the Vehicle Routing Problem, a generalization of the Travelling Salesman Problem, i.e. with multiple salesmen. This problem is also intractable, so we use Google's OR-Tools library for the best approximation algorithm.

4 Results

Using the Lagrangian Relaxation method, the algorithm picked a set of stops that covered **61.34%** of all pickup/dropoff points in the dataset. This seems like a pretty positive result given that many points lay way beyond the concentrated area of most points and only 20 stops were used for over 194 pickup/dropoff points. This immediately improves to **71.13%** and **78.35%** coverage with 30 and 40 stops, respectively, which is even better coverage. The IPython notebook includes a graph of this increasing relationship and appears roughly linear.

Using Google's VRP algorithm, each car (out of a total of $k = 4$) created a route with an average distance of **8,172 meters**. These seem to be fairly optimal routes considering the average distance from the home base (where all cars started their routes) to just *one other stop* was 2,819 meters. Full statistics with relevant annotations can be also found in the IPython notebook.

All in all, this data is limited in that it was ran on an extremely small part of a much larger dataset. If these approximation algorithms could be computed using a robust cloud computing infrastructure, we may be able to get a better idea of how close to optimal these algorithms are, giving us evidence that these algorithms may have practical value.