Final Project Report for SDS 392: An Investigation of the Multiple Traveling Salesman Problem

Erik Cheng

December 9, 2019

1 Introduction

In this project, I implement and investigate various modifications of heuristic based solutions to the multiple traveling salesman (mTSP) problem applied to a theme of delivery services.

I first test greedy route construction in which, starting from the a designated origin, a route is recursively constructed by finding the next closest stop in the route which has not yet been visited, until all stops are visited. The greedy traversal is obviously a better traversal than an arbitrarily selected route (such as, in the case that we tested, the order of which we added stops to our program), but is obviously not an optimal solution.

The greedy approach, then, is then compared a heuristic based approach to the traveling salesman problem, opt2, where segments of a path are reversed to test if such a reversal will yield a more efficient path. I show that, as expected, this solution yields a more efficient path than the greedy solution.

The opt2 heuristic is then extended to a two salesman problem where, instead of reversing segments on a single route, segments of the two routes are exchanged such that in each exchange, 4 new candidate route configurations are generated. This is illustrated below, where letters correspond to stops, lines correspond to direct connections between stops, and dots represent an arbitrary number of intermediate stops.

- Original routes: $\cdots$ A-B$\cdots$C-D$\cdots$ / $\cdots$L-M$\cdots$N-O$\cdots$
- Pure swap: $\cdots$A-M$\cdots$N-D$\cdots$ / $\cdots$L-B$\cdots$C-O$\cdots$
- Swap, reverse 1: $\cdots$A-N$\cdots$M-D$\cdots$ / $\cdots$L-B$\cdots$C-O$\cdots$
- Swap, reverse 2: $\cdots$A-M$\cdots$N-D$\cdots$ / $\cdots$L-C$\cdots$B-O$\cdots$
- Swap, reverse both: $\cdots$A-N$\cdots$M-D$\cdots$ / $\cdots$L-C$\cdots$B-O$\cdots$

I first demonstrate expected behavior of this algorithm, minimizing total distance traveled based on the provided test case (Figure 1). Then, I incorporate "Prime" addresses, which are stops that cannot be exchanged between routes. The degree of restriction imposed on the algorithm obviously scales with the proportion of stops that are "Prime"; accordingly, we see that the distance reduction that the algorithm can achieve drops as this proportion increases, with no improvement achieved once the proportion reaches around 2/3 or so.

Finally, I explore strategies for modeling and optimizing this as a single truck delivering across multiple days. I model this by treating one of the two routes as a "current day" delivery route, and the other as a "next day" or "queue" route. With such an implementation, I can model the passage of time by clearing the "current day" delivery route, and proceeding to add the new stops of the day to both the "current day" route and the "queue" route according to a chosen strategy.

2 Methodology

There are 3 main classes in my graph implementation:

- `Address`: Represents a single address with a pair of `x` and `y` coordinates. Corresponds to nodes in a graph.
- `AddressList`: a collection of `Address` objects. Implemented as a `vector<Address>`, with the only allowed "jumps" being between neighbors in the vector. In other words, this represents a graph with edges between every pair of neighbors in the vector.
- `Route`: An `AddressList` that has a `Depot`, an `Address` with coordinates (0,0) as the first and last stops (first and last elements in the underlying vector.)

To evaluate efficacy of algorithms, I generally use total distance traveled (for either one or two routes) as a metric, reported as a percent reduction in distance relative to the "as-given" length. To test the algorithms, I have written test functions that randomly generate lists of addresses, and then compute the statistics regarding distance reduction for algorithm for which each test function is named. Generally, I limited the maximum x and y coordinates to be 20.

3 Results and Conclusions

3.1 Greedy construction

It is intuitively clear that greedy construction of a route should generally be better than just traversing the route as given. This is supported by running greedy routes on a sample of randomly populated routes.

For testing, I first confirmed that my output was the same as given example in the book (using the function `example_test`). To gather statistics, I then ran

20 optimizations for randomly generated routes of about 50 addresses each, using the function `greedy_test` This yielded an average reduction of 35.18 percentage points, with a standard deviation of 20.14 percentage points. It seems this strategy works relatively well, but is highly variable in its efficacy.

3.2 Opt2

As described in the text, the heuristic for the opt2 solution effectively takes every subsegment of the graph and reverses it, calculating to see if it results in a shorter total distance. This can be accomplished effectively through two loops, one cycling through every (non-end)point, and the other through all points preceding it in the list. If we assume the input list is already optimal, there will be two swaps per loop. Assuming reversing a list takes linear (n = number of elements in list) time, this means we have approximately $2n * x^2/2$ operations, where x is the number of addresses in the route. Furthermore, we know that n is on expectation $2/3$ of x (scaled expectation of a distribution of 0 to 1 with pdf 2x); in other words, a constant factor of x. Therefore, the overall runtime of this algorithm is x^3.

To test opt2, I again first referenced the test in the book (in the commented portion of `opt2test`). After this ran properly, I ran the same type of tests that were ran for the greedy construction (in `opt2_test`). Surprisingly, in this case, the average reduction was only 24.44 percentage points, with a standard deviation of 15.65. Of course, with the intervals of opt2 and greedy construction overlapping significantly, it is difficult to tell if there is truly a difference. However, it is still surprising that opt2 did not beat the greedy algorithm. With 100 addresses, however, they both yield decreases of around 32 points. With 200 addresses, they both yield decreases of 49 percentage points. This is a somewhat unexpected result.

Although the expected trend of increasing optimization with increasing graph size holds, the relationship between the greedy and opt2 routes does not seem to hold. Unfortunately, I have not been able to find an explanation for this from my limited knowledge of how these algorithms work.

		Addresses	
	50	100	200
Reduction (Greedy/opt2)	34/24	32/32	49/49

3.3 Testing multiple opt2

We can see that the program output is as expected from the illustration from the textbook (Figure 1).

```
initial route 1:
(0, 0) (0, 2) (2, 3) (3, 2) (2, 0) (0, 0)
length: 9.88635
initial route 2:
(0, 0) (1, 3) (1, 2) (2, 1) (3, 1) (0, 0)
```

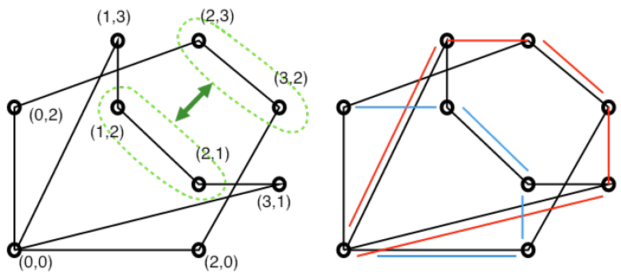

Figure 1: Test case from the textbook [1]

```
length: 9.73877

final delivery 1:
(0, 0) (0, 2) (1, 2) (2, 1) (2, 0) (0, 0)
length: 7.41421
final delivery 2:
(0, 0) (1, 3) (2, 3) (3, 2) (3, 1) (0, 0)
length: 9.73877
```

With verification that this works, it is appropriate to move onto the next set of questions, as all problems will involve the prime constraints.

3.4 "Prime" addresses

With prime addresses included, increased constraints are put on the multiple opt2 algorithm. Through testing, I found that the mopt2 algorithm tends to decrease in distance reduction in discrete-ish steps, until the proportion of prime addresses hits around 2/3, at which point there is no optimization achieved. Example output is shown below.

```
 nprimes, total addresses, length, difference
0,29,377.82,73.0625
1,29,391.62,59.2628
2,29,377.82,73.0625
3,29,377.82,73.0625
4,29,393.374,57.5083
5,29,388.142,62.741
6,29,391.62,59.2628
7,29,391.62,59.2628
8,29,393.374,57.5083
9,29,395.841,55.0415
10,29,405.988,44.895
11,29,405.988,44.895
12,29,395.841,55.0415
13,29,405.921,44.9616
14,29,411.327,39.5555
15,29,415.499,35.3841
16,29,450.883,0
17,29,443.786,7.09622
18,29,430.2,20.6831
19,29,450.883,0
20,29,450.883,0
21,29,450.883,0
22,29,450.883,0
23,29,450.883,0
24,29,450.883,0
25,29,450.883,0
26,29,450.883,0
27,29,450.883,0
28,29,450.883,0
```

3.5 Dynamicism

As noted in the text, we can treat the multiple traveling salesman setup of two separate paths as modeling simply what one truck does on multiple days. With two routes (as in the implementation), one can treat one route (`route1`) as being the current day and the other (`route2`) as being the next day. My general scheme for updating on each day, then, will be to take the new Prime orders and place them exclusively in `route1` due to their 1-day shipping constraint (I make the assumption that each day's update corresponds to orders placed the day before, therefore fulfilling the constraint). There will also be new non-Prime orders; the choice of placement of these will be the subject of exploration below. On each day's update, I will clear the previous values of `route1` (except, of course, on day 0) to model the fact that those addresses have been delivered to on the day prior to the update.

For the purposes of testing, I have assumed for a given new order that there is a 1/6 chance that order will be “Prime”. For my tests of these strategies, I will report the results of applying each strategy to 20 simulations, each of approximately 50 addresses and 50 days. The primary metric will be reduction in total distance over both routes, calculated as a percent reduction, where the average distances reduced will be averages of average distances over all days (i.e., $\frac{1}{num\_trials}\frac{1}{50}\sum\sum distance\_for\_today$).

Strategy/design choice 1 The first strategy I explored utilized the following scheme: on each day, some random number of new addresses is added to each of the two routes `route1` and `route2`. For the addresses selected for `route1`, they are selected to be prime with some fractional chance, resulting in total proportion of prime addresses randomly distributed around 1/6. After each update, I then run the multiple opt2 algorithm to approximate optimal paths. `route1` and the list of prime addresses, `primes` is then cleared to model all of those deliveries having been made. This process repeats for a desired number of days `days`. Such a strategy yields an average reduction in path length of 7.29 percentage points with a standard deviation of 0.96 percentage points.

This seems respectable, but has the slight issue that it is possible for some non-prime addresses to get stuck in the second route, and thus, never get delivered. To address this, I introduce the following two schemes.

Strategy/design choice 2 The second strategy will induce a sort of “churn” between the two routes; in contrast to the first strategy, which simply left the second route alone when updating on each day, this second strategy will take all addresses from `route2` and move them to `route1` when updating days. This, hopefully, will reduce the amount of time a given address will spend waiting in `route2`. For the newly added addresses on a given day, prime addresses must of course be added to `route1`. The new, nonprime addresses are then put into `route2`. As before, then, the multiple opt2 algorithm will be run. This strategy yields an average reduction of -10.39 percentage points with a standard deviation of 1.08; so on top of inducing “churn” between the two routes, it also seems to reduce the path length significantly.

Strategy/design choice 3 The final strategy I chose to evaluate is a minor modification of the above strategy 2, with the intent of addressing issues caused by prime addresses when using the multiple opt2 algorithm. In particular, imagine a `route1` in which every other address is prime, a situation which is of course perfectly valid from any perspective. However, due to the swapping mechanism of the multiple opt2 algorithm, and the fact that no segments containing prime addresses can be swapped, we have introduced a scenario in which for certain routes, very little can be done by the multiple opt2 algorithm to reduce the travel distance. Therefore, for my third strategy, I propose by using the update scheme from strategy 2, but to only apply opt2 to `route1`, therefore leaving `route2` untouched for one day but processing all of it on the subsequent day. This strategy has the benefit of allowing some optimization to made to prime delivery routing, whereas the multiple opt2 strategy is somewhat limited by prime. Simultaneously, it reduces the probability that any address will spend an unreasonably long time waiting for its delivery; in fact, it guarantees 2 day or less delivery for all addresses (quite a nice feature for the marketing team to share)! After testing, I find that this strategy yields an average distance reduction of 20.17 percentage points with a standard deviation of 1.49. We see that even with the addresses being added to `route2` being *completely random* and no optimization on `route2` done, the total distance between the two paths *decreases drastically.* Therefore, we may conclude that this is the best strategy.

Summary of strategies At first glance, it would seem that the algorithm suggested specifically to tackle the mTSP problem should perform the best on a mTSP problem here. Furthermore, one would initially think that the method of adding new addresses to each route on each new day would not play a significant role. However, we now see that the introduction of prime addresses plays a critical role in the optimization of these routes.

References

[1] Victor Eijkhout. *Introduction to Scientific Programming.* Online, 2019.