Amazon Delivery Truck Simulation

Hanna Butt [*] Ashton Cole [†] Kelechi Emeruwa [‡]

November 26, 2022

Abstract

Summary of whole paper. Note that this is not an introduction or context, but a summary.

1 Introduction

Introduce the context and the problem here. Also talk about project context (we're in COE 322 fall 2022, this is a final group project, bla bla bla).

Somewhere in the introduction there should be a paragraph or subsection precisely defining the Traveling Salesman and Multiple Traveling Salesmen Problems. Explain the general idea, assumptions, and limitations. For example, we use integer 2D coordinates on a flat plane, distances don't account for windy roads, bla bla bla, &c.

Imagine that a delivery company has a series of orders that it needs to fulfill. It has a truck that can stop at each address and make the delivery. The company naturally wants to save on time and fuel costs, so it tries to find the shortest path from its warehouse to cover all of the stops. This is the basic premise of the Traveling Salesman problem. One could simply test all $n!$ combinations of a list of n addresses for the best path, calculating distances and adding them up, but with 4 stops this becomes quite tedious, and after that nearly untenable. The prferable alternative is to use algorithms to find good and better paths, significantly cutting down on calculations.

This is the problem that our team chose to solve for our final project. We are undergraduate students in COE 322: Scientific Computation in the fall of 2022. KEEP TYPING HERE PLEASE

In Section 2 we discuss the algorithm used to solve the simple Traveling Salesman Problem, and expansions upon it to construct our final program. In Section 3, we display the outcomes of several test scenarios and discuss the results. Finally, in Section 4, we offer our final thoughts and reflect on ethical considerations.

[*] HFB352

[†] AVC687, ashtonc24@utexas.edu

[‡] KEE688, kelechi@utexas.edu

2 Methodology

To approach this problem, we wrote a library and scripts in C++. These were compiled with `g++` on our local machines and `icpc` on the Texas Advanced Computing Center's ISP supercomputer. These scripts are outlined below.

- `traveling_salesman.h`: a header file for our TravelingSalesman library, defining all of the objects and algorithms used in the project
- `traveling_salesman.cpp`: an implementation file
- `tester.cpp`: a script which tests the functionality of our TravelingSalesman library and generates TikZ code for figures displayed in this report; this script is compiled as `tester.exe`
- `routeGenerator.cpp`: a script which generates `.dat` files containing lists of orders to be processed by `delivery_truck_simulation.exe`; this script is compiled as `routeGenerator.exe`
- `delivery_truck_simulation.cpp`: a script which represents a hypothetical final product for use in industry, as described in Section 2.4; this script is compiled as `delivery_truck_simulation.exe`

We began our project by writing the header and implementation files, coupled with tests in our tester file. After we confirmed that all of our objects and algorithms functioned as expected, we designed a main program to best parallel a real world application of solving the Traveling Salesman Problem. The structures and algorithms that we developed are further detailed in this section.

In Section 2.1, we outline the structure of the classes that we used to represent and solve the problem. In Section 2.2, we describe the algorithms we used to solve the simple Traveling Salesman Problem. In Section 2.3, we describe the expansion of the problem to account for optimizing multiple delivery routes. Finally, in Section 2.4, we describe how we combined our algorithms into a final product for a hypothetical user.

2.1 Object-Oriented Structure

Our scripts took advantage of C++'s object-oriented capabilities to organize the problem. This section provides a brief overview; the contents of these classes are not described exhaustively.

Each delivery stop is represented by an `Address` object, which has two-dimensional integer Cartesian coordinates `i` and `j` representing the location of the address, an an integer `deliver_by` which describes the day by which the order is supposed to be delivered. The class can also calculate the distance to other `Address`es, using either the Cartesian distance $\sqrt{i^2 + j^2}$ or Manhattan distance $|i| + |j|$. In our implementation, we use the Cartesian distance, but it could easily be replaced with another formula.

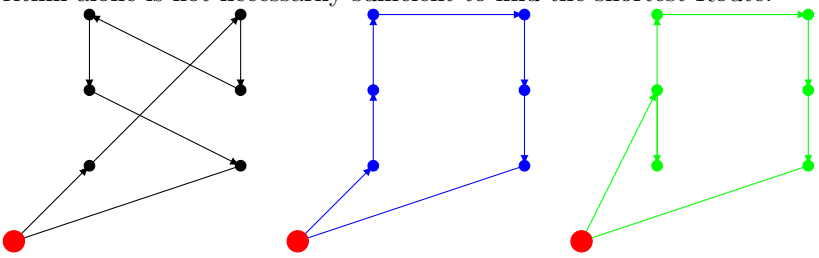
A list of `Addresses` is represented by an `AddressList` object, which holds the objects in a `std::vector<Address>` instance variable called `address_list`. This class can add, remove, and rearrange `Addresses`.

The `Route` class extends the `AddressList` class by including a `hub` instance of type `Address`. This represents the starting and ending point of the `Route`. This class contains several functions to solve variants of the Traveling Salesman problem.

2.2 Traveling Salesman Problem

Talk about greedy and opt2 algorithms.

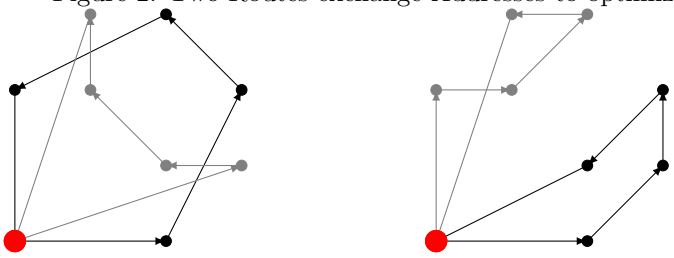
Figure 1: An unsorted Route is optimized through both the greedy algorithm (blue) and the opt2 algorithm (green). This demonstrates how the opt2 algorithm alone is not necessarily sufficient to find the shortest Route.



2.3 Multiple Traveling Salesmen Problem

Talk about how swap algorithm works. Figure 2 provides a simple example of how this algorithm improves the total distance. (Now prove it with data! What are the distances before and after?) ALSO: Talk about how the gray and black Addresses are symmetrical, but the trucks take different Routes. Is this okay, or does one or both of them need to go through greedy/opt2???

Figure 2: Two Routes exchange Addresses to optimize their distances.



2.4 Developing the Final Product

Our team

How is our final thing going to work? Basically we want it to work for businesses.

3 Results

Pretty pictures and tables go here. Describe each situation being displayed and talk about what they mean, e.g. is it the optimal solution? Good enough? Is there a tradeoff between time to execute and quality of results?

Hmm, maybe insert a table comparing number of nodes/trucks to program execution time. What rate does it increase at ($O(n), O(n^2), \&c.$)

3.1 Scenario 1

3.2 Scenario 2

3.3 Performance Data

go over execution time and stuff

4 Conclusion

Talk about what we learned, how this all applies to industry, ideas to scale the problem up, ethics, &c.

A Sample of Using Listings Package

Please remove this appendix before publishing! Here's some pretty C++ code from Listing 1.

Listing 1: Example Code

```
1 #include <iostream>
2 using sdt::cout;
3
4 int main() {
5     // Print out hello
6     cout << "Hello world!" << '\n';
7     return 0;
8 }
```