Traveling Salesman Problem: Optimizing Amazon's Delivery Truck Schedule

Hieu Nguyen (hcn357), Rushil Randhar (rnr825), Sharan Sethi (ss98799)

2023/12/05

1 Abstract

This project aims to address the complexities of optimizing Amazon's delivery truck scheduling, particularly focusing on the unique challenges posed by the Traveling Salesman Problem (TSP). The standard Traveling Salesman Problem involves finding the shortest possible route for a single salesman. The MTSP (Multiple Traveling Salesman Problem) must account for multiple trucks or routes, adding layers of complexity to the problem. This study leverages algorithmic techniques, particularly an extension of the opt2 heuristic, to multiple paths for efficient route management. The algorithms are designed to adapt to various operational constraints, such as specific delivery windows promised to customers and prioritized deliveries for Amazon Prime members. This research not only seeks to improve route efficiency but also aims to understand the potential impact of these optimizations on Amazon's labor practices and overall delivery operations.

2 Introduction

In the realm of logistics and delivery services, route optimization is a pivotal task, especially for companies like Amazon that handle a vast number of deliveries daily. Traditional approaches to route optimization are based on the Traveling Salesman Problem (TSP). However, the scenario becomes significantly more complex when dealing with multiple delivery trucks, each with its own set of delivery locations, known as the Multiple Traveling Salesman Problem (MTSP).

Our project aims to develop a comprehensive algorithmic approach that not only addresses the complexity of the MTSP but also considers the practical aspects of delivery operations, such as route flexibility, driver workload, and operational efficiency. We utilized techniques like the Greedy Algorithm and Opt 2 Heuristic to most optimally create a solution to the MTSP or at least reduce the load upon delivery workers.

In this project we hope to cover each algorithm and strategy used to formulate and justify our solutions. We will also explain the classes and syntax used within our solution and finally discuss the real world application of these solutions through an ethical lens.

3 Discussion

3.1 Address and Route Representation

The ‘Address‘ class serves as a foundational element in our delivery system. It represents each delivery location with specific geographic coordinates. This class is designed to handle additional information which can be crucial for delivery management, such as a flag indicating whether an address is an Amazon Prime destination, requiring prioritization due to tighter delivery schedules.

Additionally, the ‘AddressList‘ class acts as a container for multiple addresses. It also encapsulates the entire set of destinations for a delivery route. This class is equipped with functionalities for adding new addresses, calculating the total distance required to cover all the addresses in the list, and a method for identifying the closest address to a given point, which is particularly useful for route optimization.

The ‘Route‘ class, inheriting from ‘AddressList‘, is specifically made for optimizing delivery routes. It ensures the integrity of routes by maintaining fixed start and end points, usually the depot, and includes sophisticated methods for route optimization. These methods sort the addresses optimally to minimize travel distance and time.

3.2 Route Construction Strategies

Initially, routes are constructed using a greedy search algorithm, which orderly selects the nearest next delivery point from the current location. While this method is simple and efficient, it does not guarantee an optimal solution due to the inherent complexity and NP-hard nature of the TSP.

The Greedy algorithm[1], applied in the context of route optimization, operates on a straightforward principle by selecting the closest destination from the current location. This approach is particularly effective in scenarios where the goal is to minimize travel distance or time for tasks like delivery routing. The algorithm begins at a starting point (often a depot) and iteratively chooses the nearest unvisited address as the next stop, continuing this process until all destinations are covered. It doesn't always guarantee the most optimal solution, especially for larger, and more complex scenarios.

In terms of time complexity, the Greedy algorithm for route optimization typically exhibits a complexity of $O(n^2)$ for a problem with 'n' destinations. This is because, at each step of the route construction, the algorithm needs to evaluate the distances to all remaining unvisited destinations to find the nearest one. Since this process is repeated for each destination, the overall time complexity becomes quadratic in relation to the number of destinations. While this is manageable for smaller datasets, larger problems cause the performance to decline, making the Greedy algorithm less suitable for scenarios with a very high number of destinations. Despite this, its simplicity and ease of implementation make it a popular choice for initial route planning before applying more complex optimization methods.

The Opt2 (or 2-opt) algorithm[1] is a algorithmic method used in route optimization, particularly effective for solving problems like TSP. The purpose of Opt2 is to iteratively improve a given route by reversing segments of the route to reduce the total distance. At each step, the algorithm considers two edges (or links between desti-

nations) and checks if swapping these edges—effectively reversing the path between them—results in a shorter overall route. This process is repeated for different pairs of edges until no further improvements can be made, or a certain condition is met.

```
while(found_improvement){
    found_improvement = false;
    double current_length = route.length();
    for(int i = 1; i<address_list.size()-2; i++){
        for(int j = i+1; j<address_list.size()-1; j++){
            Route prev = route;
            prev.opt2_swap(i, j);
            current_length = prev.length();
            if(smallest_length > current_length){
                route = prev;
                smallest_length = current_length;
                found_improvement = true;
```

Looking the code snippet we can get the general idea of how the opt2 works by iterating through every possible segment and applying an opt2 swap by reversing the segment in the route. If we get a shorter route then keep it. The strength of the Opt2 algorithm lies in its ability to locally optimize a route. By focusing on reversing segments of a route, it can efficiently untangle routes and remove suboptimal paths like crossings, which are common in initially generated routes from simple algorithms.

In terms of computational complexity, the Opt2 algorithm typically exhibits a time complexity of $O(n^2)$ for a route with 'n' stops. This stems from the need to consider every pair of edges for reversal. However, various optimizations can be applied to the basic Opt2 approach to improve its efficiency, such as only considering edge pairs that are likely to result in an improvement, based on heuristics or distance criteria.

The improvement from Greedy to Opt2 is particularly evident in the quality of the final route. Opt2 tends to unravel and smooth out the route, leading to a more direct and shorter path overall. While it is more computationally intensive than the Greedy algorithm, due to its need to evaluate multiple pairs of edges, the Opt2 heuristic is more likely to approach the global optimum, especially in moderately sized problems.

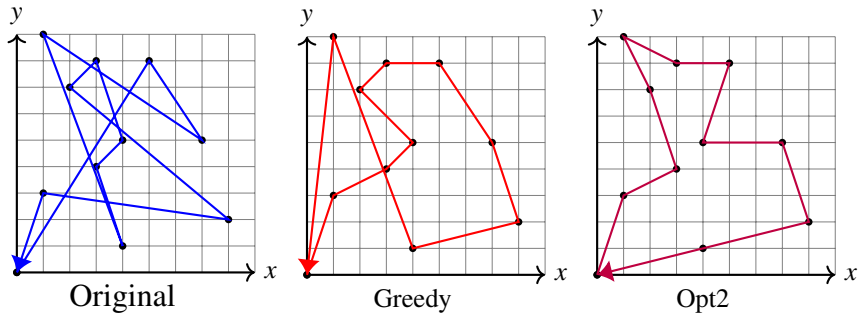

Figure 1: Comparison of Route Optimizations

3.3 Multiple Travelling Salesmen Problem

The ‘multi opt2‘ algorithm[1], as implemented in the below C++ code, is a route optimization technique used to find a more efficient path. It functions by systematically trying different combinations of segments from two routes and checking if these combinations yield a shorter total distance. It operates by performing a series of 2-opt swaps, a common method in solving routing problems like the Traveling Salesman Problem.

```
while(found_improvement){
    found_improvement = false;
    for(int i = 1; i<address_list.size()-2; i++){
        for(int j = i+1; j<address_list.size()-1; j++){
            for(int k = 1; k<other.address_list.size()-2; k++){
                for(int l = k+1; l<other.address_list.size()-1; l++){

                    if(try_multi_opt2_swap(route, other, i, j, k, l, false,
                    false) ||
                        try_multi_opt2_swap(route, other, i, j, k, l, true,
                        false) ||
                            try_multi_opt2_swap(route, other, i, j, k, l,
                            false, true) ||
                                try_multi_opt2_swap(route, other, i, j, k, l,
                                true, true)){
                                    found_improvement = true;
                                }
                }
            }
        }
    }
}
```

A 2-opt swap involves taking two edges in a route and swapping them to see if it results in a shorter path. The algorithm iterates through all possible pairs of edges in both routes and attempts four variations of swaps: both routes unchanged, the first route reversed, the second route reversed, and both routes reversed. This process is repeated until no further improvements can be found.

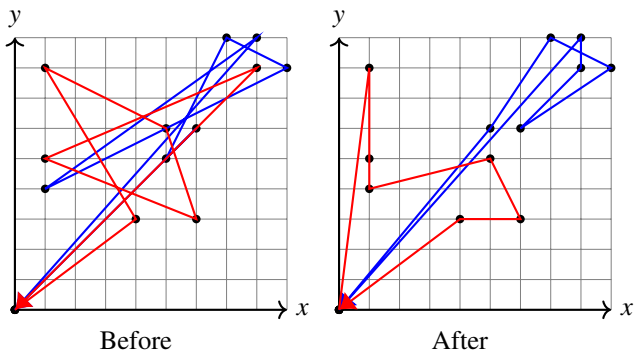

Figure 2: Optimization of two routes using multi opt2

In terms of time complexity, the ‘multi opt2‘ algorithm is quite intensive. It involves

four nested loops, each iterating over a subset of a route's addresses. If n represents the number of addresses in a route, the time complexity is $O(n^4)$, as each loop contributes $O(n)$ complexity. This makes the algorithm increasingly slower as the number of addresses grows, reflecting its non-polynomial time complexity (NP-hard).

4 Experiments

4.1 Introduction of Prime Addresses

In our delivery route optimization we had previously ignored Prime addresses. Within the context of Amazon's delivery system, the distinction between Prime and Non-Prime addresses plays a crucial role. Prime addresses, characterized by their urgent delivery requirements, often necessitate expedited handling and are typically non-exchangeable between routes once assigned. This contrasts with Non-Prime addresses, which represent the standard delivery workload without such stringent time constraints. The incorporation of this binary categorization introduces a unique constraint into the optimization process, notably when employing algorithms like multi-opt2, an advanced version of the 2-opt heuristic adapted for multiple routes. The non-exchangeability of Prime addresses substantially influences the flexibility and effectiveness of the route optimization process. In standard scenarios, optimization algorithms, including multi-opt2, strive to enhance efficiency by intelligently rearranging stops and balancing loads across different routes. However, the fixed nature of Prime addresses within a specific route significantly limits these optimization possibilities. This constraint often leads to an increase in the average distance of optimized routes compared to scenarios where addresses are freely exchangeable. The primary reason is that the algorithm is compelled to work around the immovable positions of Prime addresses, which can lead to suboptimal routing choices for the remaining stops.

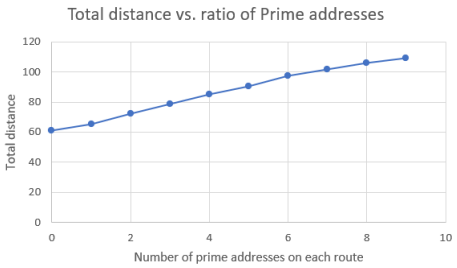

Figure 3: Total distance traveled as the number of prime addresses increase

In our simulation we tested 2 routes which each get 10 random addresses and changed the number of prime addresses within each route. Running the multi-opt2 algorithm 1000 times on the randomly generated routes at each prime ratio we observed as the number of prime addresses increases the average total distance traveled increases.

Adapting the multi-opt2 algorithm to this context requires careful consideration of these constraints. While the algorithm can still optimize the sequence of addresses within each route, it must do so by treating Prime addresses as fixed, unmovable points. This adaptation results in a scenario where the algorithm's capability to swap or reverse segments is considerably constrained by the presence of these prioritized stops. The outcome, as observed in our simulation, is an increase in the average travel distance for optimized routes, highlighting a significant trade-off. This trade-off underscores the complexity logistics and delivery services face in balancing operational efficiency with the imperative to meet time-sensitive delivery commitments.

4.2 Strategies to dynamic addition of Addresses

1. Push to End of List: A possible option of optimizing the delivery route is the simple Push to End solution. This is a very simple solution as it tacks on the most previously received address from the time the order was placed on the queue of addresses. There is no further process in the list.

It is very clear how this option can be proven as ineffective. This is because there is no true optimization happening; in turn the path that is curated for the delivery drivers is random and very inefficient. Furthermore, the inefficient route being curated by this option also increases fuel costs and wear and tear on the vehicle. Choosing this method is convenient when creating a system to inform the delivery drivers of which address to go to but never when creating an efficient and optimized path of delivery.

2. Re-Optimization Upon Each New Address: In scenarios in which optimization of the route is highly prioritized, certain addresses can be left behind in the delivery process.

Let's say there are 30 deliveries in City X, which is a highly populated city, and one delivery in a neighboring City Y, which is not as populated nor filled with delivery orders. It is expected that in the time that the driver completes their deliveries in City X there will be more orders within City X which will continuously optimize the route and put the driver to only deliver within City X. The one delivery that must be made in City Y will take days or even weeks on end to be completed because the delivery in City Y is not optimal for the previous path that was set.

While the optimized path might, at first, seem to be the most logical, it does not satisfy each order equally because there are some regions along a delivery path which have a higher number of orders. With this comes possible customer dissatisfaction and a potential decrease is the number of future orders from City Y. Because this option does not effectively balance both the equitable and optimal aspect of Amazon's delivery service, it is not the best choice.

3. Knapsack 0-1 Method: In an optimized delivery operation combining the 0-1 Knapsack method with route optimization, the process unfolds in two distinct but interconnected phases. First, using the 0-1 Knapsack approach, the most valuable mix of packages is selected for the delivery vehicle, strictly adhering to its weight capacity. This step involves an evaluation of each package's weight and its value, which could be based on factors like priority, profitability, or delivery urgency. The aim is to maximize the value of the load within the physical constraints of the vehicle.

Once the optimal load is determined, the focus shifts to route optimization. This stage involves planning the most efficient path for the delivery vehicle to take. It considers various factors such as the distance between delivery locations, traffic patterns, delivery time windows, and road conditions. Advanced algorithms can be used to calculate the quickest and most fuel-efficient route, ensuring that the vehicle not only carries the most valuable load but also travels in the most efficient manner.

The integration of load optimization with route planning thus ensures a high level of efficiency, balancing the load's weight and value with the practicalities of distances and delivery schedules.

5 Ethics

Our code offers insights into Amazon's logistics and delivery operations, indirectly shedding light on aspects of the company's labor policies, particularly those concerning its delivery drivers. Firstly, the focus on optimizing delivery routes suggests Amazon's commitment to efficient delivery management.

Within the code, there are provisions for managing lists of delivery addresses, indicating the importance of accurate and timely deliveries. Amazon's labor policies likely encompass guidelines for drivers to handle customer addresses professionally, ensuring that deliveries are carried out correctly. The inclusion of route optimization algorithms, such as the greedy algorithm and 2-opt swaps, demonstrates commitment to streamlining delivery processes. Efficient routing can help drivers complete deliveries within their allotted work hours, potentially affecting labor policies regarding work hours, rest breaks, and overtime. Labor policies related to driver compensation, such as hourly wages and bonuses, might depend on how efficiently drivers complete deliveries.

In conclusion, the code itself offers insights into the company's approach to logistics and delivery, which indirectly reflects its labor practices regarding delivery drivers.

6 Conclusion

Our comprehensive study underscores the complexity and challenges involved in Amazon's delivery truck scheduling. The opt2 heuristic, while significantly improved route efficiency, also highlights the limitations of current algorithmic approaches in handling the dynamic nature of delivery operations. The experiments conducted reveal the need for a balance between route optimization and operational systems, such as driver workload and delivery prioritization.

Furthermore, the study opens avenues for future research in advanced algorithmic solutions, such as machine learning-based and predictive modeling for route optimization. Additionally, our findings bring to light important considerations regarding delivery systems and the potential impacts of route optimization on driver workloads and overall operational efficiency. As delivery operations continue to evolve, it becomes increasingly crucial to develop solutions that are not only efficient but also sustainable and considerate of the human element in logistics.

References

[1] Victor Eijkhout. Introduction to scientific programming. pages 503–510, 2023.