Cost Optimization for Freight Distribution Networks with Consolidation Facilities

Business Analytics - Internship Report

Shipwell Advisor: Tyler Dorland
Supervisor: Oliver Fabert
2nd Reader: Rene Bekker

Author:
Austin Dickerson (2729612)

Cost Optimization for Freight Distribution Networks with Consolidation Facilities

Author:
Austin Dickerson (2729612)

Vrije Universiteit Amsterdam
Faculty of Science
Business Analytics
De Boelelaan 1081a
1081 HV Amsterdam

Host Organization:
Shipwell
515 Congress Avenue
Austin, TX, 78701
November 2024

Abstract

Supply chain logistics is a complex problem space, with increasing volumes of daily orders to transport and multi-echelon transportation requirements making traditional optimization algorithms struggle to find cost-optimal solutions to daily transportation needs. Shipwell needs a tool to optimize supply chains with Pool Points, where orders transfer to Pool Vehicles for discounted routes.

This study compares tools, modeling decisions, and optimization methodologies for solving a Hybrid Multi-Echelon Pickup and Delivery Problem with Time Windows and Pool Points, focusing on the data of Whole Foods Market. Google's OR Tools Library and the Gurobi Optimizer are tested using different problem modeling strategies and augmentations, focusing analysis on both the cost of solutions and the scalability limits of each model.

The best result was achieved by an Upper Confidence Bound selected Large Neighborhood Search Algorithm (UCB LNS), iteratively reconstructing partial solutions. Final results, where the UCB LNS Algorithm is bagged with one of its robust benchmarks project Whole Foods Market will save 1,540,000$ annually or 1.849% total on their produce procurement supply chain.

Preface

This paper is written for the Master Project in Business Analytics at the Vrije Universiteit Amsterdam. The main focus of this study is comparing different methods for modeling the Pool Distribution Optimization Problem (PDOP) to be solved primarily using off-the-shelf optimization software, while remaining scalable with over 100 orders in a single model.

This research was done as a collaboration between the Vrije Universiteit Amsterdam and the Data Science department at Shipwell, a company that specializes in handling the logistics needs of other companies through digital supply chain technologies, ranging from trucker rating systems, to supply chain analytics, to routing optimization.

I would like to give a special thank you to my supervisor at Shipwell, Dr. Tyler Dorland; for sharing a wealth of experience and wisdom in data science, indulging technical questions both related and unrelated to this study, and trusting me to design a tool that has a clear impact on a major business.

Finally, I would like to thank my supervisors at the Vrije Universiteit Amsterdam, Dr. Oliver Fabert and Dr. Rene Bekker; for providing contextual insight on optimization, giving guidance and feedback on the steps and process of conducting research, and committing to coordinate with a student who is 5,000 miles and 9 timezones away.

Contents

Chapter 1

Introduction

Supply chain logistics is a sector that's seen sporadic, but rapid growth in the digital age with companies such as Amazon investing heavily in developing incremental advancements. The recent Covid-19 pandemic dramatically shifted the status quo for supply chain logistics, as links within existing supply chains were disrupted by lock-downs. The lock-downs exacerbated an existing trend for commerce to move online, forcing companies to re-strategize the distribution of their products. McKinsey Consulting surveyed businesses in 2021, and found an average of 88% across industries invested in digital supply chain technologies [1].

With numerous types of transportation options, especially for long journeys, the need forms for innovative routing algorithms to minimize the cost while handling the complexity of real world factors such as vehicle capacity, open hours of locations, and multi-vehicle order transfers. Managing these complexities leads to optimization problems that demand problem-specific, hybrid algorithms to solve, as research using traditional optimization algorithms struggles to handle these constraints effectively. As various optimization algorithms have become more robust in the face of real-world constraints, specialized optimization tools that work with the business rules of a company's logistics become more viable. This leaves an opportunity for any business taking optimization lightly to receive help, gaining efficiency in their existing supply chain components.

1.1 Shipwell

Shipwell is freight shipping logistics startup originally based in Austin, Texas, which has transitioned to operating fully remote. After launching in 2018, Shipwell weathered the changes caused by Covid-19 lockdowns and their impact on supply chains worldwide. Emphasizing a personal touch with their clients, and known for exceptional customer service, Shipwell maintains relationships with large companies to add lucrative logistics contracts that generate revenue. The clients are looking for a service that can handle every step of their logistics process, from planning routes, to tracking shipments, to finalizing contracts with third-party freight transportation firms. Shipwell works toward satisfying their clients needs by staying ahead of the industry, leveraging their strong software engineering culture to implement advanced data analysis, optimization, and language models in its user friendly **Transport Management System (TMS)** platform.

Shipwell recently raised another round of venture capital funding to continue their drive for growth by offering new features for their clients. An important service in development is the **Pool Distribution Optimization** tool, which is needed in particular by the new client Whole Foods Market. Whole Foods Market is a very important client for Shipwell, so the Pool Distribution Optimization tool is a high priority.

1.2 Whole Foods Market

Whole Foods Market is a large grocery store chain operating in the United Sates with over 500 locations, including a few in Canada and the UK. The company was purchased in 2017 by Amazon Inc. for 13.7 billion US Dollars. The pairing of Whole Foods Market and Amazon was an interesting one, as Amazon was known for efficiency and lowest cost, while Whole Foods Market had a reputation that inspired the moniker "Whole Paycheck" to describe it's pricing scheme [13]. Since the purchase, Whole Foods Market has integrated more technology into their stores, facilitating Amazon services like locker package pickups and returns, checkout-free shopping, and in-store sensory technology for advanced analytics [33].

With fealty to tech giant Amazon, Whole Foods Market looks to move toward more advanced methods of logistics planning. This means approaching optimization analytically, while it has previously been done by hand. With Shipwell as their logistics coordinator, leveraging tailored optimization strategies, Whole Foods Market hopes to make a jump in efficiency on the logistics side of their business.

1.3 Pool Distribution Freight Networks

Pool Distribution Optimization is a method of maximizing the value of **Pool Points**, which are intermediary locations operated by a business to benefit their supply chain. At pool points, orders can transfer from a **Full Truck Load (FTL) Vehicle** onto a designated **Pool Vehicle**. These designated vehicles are assigned a specific route, connecting the pool point to a **Distribution Center**, which is the sole destination for the orders on the designated pool vehicle. These routes are scheduled regularly, and so Shipwell is able to secure lower rates for these routes than for ad hoc deliveries. Routing orders through a pool point to get a discounted rate for the remainder of an order's journey lowers net shipping costs, so long as the optimizer can get the order to the pool point efficiently. Otherwise, it can be cheaper to skip the pool point.

A key benefit to the pool points is the opportunity for an FTL vehicle to pick up orders bound for disparate destination coordinates, which is only allowed when the truck can drop all its orders at a pool point. FTL vehicles are the primary freight vehicle type available for servicing all transportation not originating from a pool point, called FTL because the entire truck's capacity is paid for regardless of true utilization. The orders dropped at a pool point by an FTL vehicle are then placed on separate pool vehicles, each traveling directly to their respective destinations, after consolidating orders from other FTL vehicles. Alternatively, FTL trucks that skip a pool point must travel to a distribution center to drop off their contents. These centers are typically hundreds if not thousands of miles apart, so each FTL truck is only allowed to visit at most one. The opportunity for greater efficiency does come with drawbacks, as this system creates a real optimization challenge, because one set of vehicles' behavior affects the demand for another set of vehicles. This causes inter-dependency between the vehicle fleets, leading to

dynamic demand and synchronization issues [15].

The Pool Distribution Optimization tool looks to solve a highly constrained version of the **Pickup and Delivery Problem (PDP)**, which is an extension of the **Traveling Salesman Problem**, a classic NP-Hard problem. The problem addressed in this study is coined the **Pool Distribution Optimization Problem (PDOP)**. Although research on an optimization problem with all the specifics of the Pool Distribution Optimization Problem is sparse, each element has been addressed in some research within the Pickup and Delivery Problem umbrella. However, some approaches, which are effective in less constrained PDP, cannot effectively explore the solution space of the PDOP.

1.4 Objective and Research Questions

This goal of this research is to develop a tool which can consistently produce savings optimizing vehicle routes that service a set of orders, scheduled for pickup and delivery, while remaining computationally robust in scenarios with 100 or more orders. Savings is measured as the improvement on cost from using pool points over routing all orders using only FTL vehicles, while robustness is measured by the ability to find a feasible solution in each test case within the allotted time. Over the course of three sets of experiments; driving distance approximation methods, off-the-shelf optimization software, problem modeling decisions, optimization parameters, and problem-specific search operators are tested to reach the best practical routing optimization tool Shipwell can use to solve the PDOP and satisfy their client Whole Foods Market.

Research Question: *What is an effective and scalable approach for an optimization algorithm minimizing cost in the Pool Distribution Optimization Problem, which meets the capacity needs of major United States suppliers and distributors?*

The following sub-questions appear through the process of developing the PDOP optimization tool.

- *How can the optimization tool best predict real driving distances with minimal computational overhead and cost, while allowing for detailed visualization of routing solutions?*
- *How can PDOP optimization scenarios be modeled in a way that is solvable with multiple pool points?*
- *What methods effectively lower the cost of initial feasible solutions?*
- *Which Solution Strategies, Search Metaheuristics, and Destroy Operators are most effective when iteratively destroying and repairing solutions produced by the Standard OR-Tools Model?*

Research sub-questions are discussed more thoroughly in the coming chapters, with sub-question 1 addressed in Chapter 4, sub-questions 2 and 3 in Chapter 5, and sub-question 4 in Chapter 6. Each question builds on results from the previous, as the optimization tool hybridizes algorithms and optimization steps

Chapter 2

Related Work

This chapter gives an overview of research on traditional optimization approaches frequently applied to Pickup and Delivery Problems, and examples of hybrid algorithms applied to problem formulations most similar to the Pool Distribution Optimization Problem. Section 2.1 provides explanations on well substantiated methods for solving the Pickup and Delivery Problem, focusing on why they may be well suited to consider in the PDOP. These approaches range from Heuristic Routing Algorithms, to Evolutionary Computing, to Machine Learning. Section 2.2 discusses research on optimization problems most similar to the Pool Distribution Optimization Problem. Due to the use of intermediary points where orders exchange vehicles, these problems share some unique challenges in optimization with the PDOP. Particular attention is paid to the methods for exploration in these highly constrained problem formulations.

2.1 Pickup and Delivery Problem (PDP)

The Pickup and Delivery Problem is a well known NP-Hard optimization problem which builds on the earlier Travelling Salesman Problem (TSP). The fundamental components of the PDP are a set of nodes where a pickup is required and a set of nodes where a delivery is required. The PDP often uses a set of vehicles with limited capacity, becoming the more practical Capacitated Pickup and Delivery Problem. As the problem becomes more complex, the exact nature of the more primitive, greedy algorithms becomes computationally intractable [30]. The greedy algorithms that work well on the TSP struggle in a PDP because they

cannot predict the long term consequences of early choices made in the routing process that occur because of numerous constraints [9], but they computationally cannot explore the entire search space either. Methods for solving the PDP have means of exploring the solution space besides the optimal next individual route operation, using a variety of methods to guide the routing exploration into a larger search neighborhood of adjacent possible solutions.

2.1.1 Hybrid Heuristic Algorithms with Local Neighborhood Search

In solving complex routing optimization problems, Heuristic Algorithms work by making sequential changes to an existing set of routes until a stopping criterion is reached. The stopping criterion can be a time limit, a plateau in the solution's objective value, or a margin between the current objective value and a lower bound on the solution calculated by relaxing part of the problem's constraints. Hybrid algorithms beak the entire process into segments handled by different heuristics. The first heuristic component, also called a **First Solution Strategy**, searches for an initial feasible solution by prioritizing exploration. This is most often achieved with an insertion algorithm, which prioritizes adding nodes to routes over improving the configuration of nodes already in routes [2]. After obtaining an initial solution, a different set of heuristic algorithms, called a **Search Metaheuristic**, sequentially improve the solution while maintaining feasibility. As a search metaheuristic algorithm works, it produces a local search neighborhood after each change to the solution. The local search neighborhood contains all the solutions which can be reached through some operation on the current solution. Search Metaheuristics introduce randomness, memory, or a guided search to explore new regions of the solution space, reaching better solutions over time [30].

Local search algorithms explore slowly and would struggle to reach an initial solution on their own, but this combination of a First Solution Strategy and a Local Search Metaheuristic allows for a feasible solution to be found quickly, and for sequential improvements to navigate toward a strong optima in the solution space [26]. These two components can also contain sub-components. For example, popular off-the-shelf Mixed Integer Linear Programming optimization tool **Gurobi** forms a lower bound on the solution cost using the Barrier method. This algorithm is similar to the simplex algorithm, but penalizes arcs near the edge of

the feasible solution space when building a solution [16]. This method works because Gurobi relaxes the integer requirement on the values for each arc, showing how precise changes to the model formulation can lead to better solutions. To reach a feasible solution from the relaxation, Gurobi uses a Branch and Bound algorithm to break the problem into less dependent pieces, and rounding arc values to integers. Gurobi uses a complex first solution strategy compared to those implemented in open source library from Google, **OR Tools**, but the solution lower bound gives a helpful means of assessing the quality of solutions produced by the optimizer [16]. A major strength of Hybrid Heuristic Algorithms is their resilience in highly constrained problems. The search algorithms check feasibility constraints for every operation when building a local search neighborhood, rather than optimizing selection and checking feasibility second [31].

2.1.2 Genetic Algorithms (GA)

Genetic Algorithms look at the PDP in its entirety, creating a search space for the problem through a population of individuals that represent different possible solutions. All GA share the properties of survival, crossover, and mutation, however different types of GA represent the optimization problem as either floats or integers. As the population of individual solutions moves from one generation to the next, each solution's quality is assessed by a fitness score [12]. Then, a selection process, which can be deterministic or stochastic, uses the fitness scores to select which individuals in the population will reproduce. During the reproduction process, a crossover operator combines the information inside the individuals into a single offspring. However, more than two parents contributing to a child is possible, as in Differential Evolution. Children resulting from the crossover of individuals in the previous generation are then mutated. This occurs based on mutation temperature, which is the probability that any given allele in the new individual receives a mutation [12]. A difficult aspect of using GA for a highly constrained optimization problem such as the PDOP is ensuring the feasibility of offspring. Due to the fact individuals in the population can have completely different characteristics, two parents selected may have very little room to create a feasible solution from one another's components. Making a GA that can navigate a highly constrained solution space requires the use of feasibility-preserving operators for crossover and mutation, but the more constrained the problem, the more those operators dictate the evolutionary process [21], making a balance of

feasibility and exploration difficult to achieve.

Differential Evolution and Swarm Intelligence

In their research on unidirectional logistics distribution, Xu et al. tackled a problem with flow similarities to the PDOP using differential evolution. They used an encoding of nodes in the model to the ordinal position of floats in an array [43]. The floats were initialized in an array of size L, equal to the number of nodes in the model, and could take values anywhere from 0 to K, where K is the number of vehicles. Each vehicle's route was assigned all the indices in the array whose value $\in$ (k-1, k], where k is the vehicle number [43]. These indices are placed in descending order, so an index with value 1.22 would be visited by vehicle 2, but only after it has visited another array index whose value is 1.5. They used differential evolution, which by comparison to traditional GAs converges quickly, because new offspring only replace parents if they have a better fitness score. They also differ in the means of crossover, using perturbation vectors to change existing members of the population [32], which has the advantage of allowing offspring to obtain alleles that neither parent originally had, without needing mutation.

The problem with implementing such a model is again the difficulty of achieving feasible offspring. Xu et al. looked at models with up to 21 nodes, much smaller than those necessary at Shipwell, making more of the possible solutions feasible. Although other researchers have studied larger models, the amount of constraints in the PDOP explains why similar GA research uses smaller models. Genetic algorithms remain a popular way to solve combinatorial optimization problems, but their high compute cost, and poor performance in highly constrained problems make it less ideal for implementation of the PDOP.

Within Evolutionary Computing, a propitious approach to discrete optimization applies Swarm Intelligence. This approach increases the communication between individuals in the population, by allowing high performing individuals to send signals to other individuals, biasing them to move towards the high performers in the solution space [19]. However, solutions that rely on strategies like Particle Swarm Optimization or Ant Colony Optimization are not well suited for use with feasibility-preserving operators because these algorithms prioritize explo-

ration over strict adherence to constraints.

2.1.3 Learning-Based Optimization (LBO)

Recently, optimization strategies that leverage some form of machine learning have shown promise solving routing optimization problems. In a recent synopsis of LBO strategies, Li et al. found two main types of LBO applied to TSP type problems. These consisted of end-to-end approaches, and step-by-step approaches [23]. Step-by-step approaches work based off either supervised learning or reinforcement learning and use sequential route operations, similar to a local search algorithm, but require either huge sources of routing data or the compute power to train a reinforcement learning algorithm. Although heuristics like using an experience replay buffer can improve the efficiency of this process, either approach within step-by-step requires significant resources [23]. The other LBO approach gaining traction is end-to-end, where the original demands of the system form an input and the output is a complete solution. This works more similarly to genetic algorithms, although there is no iterative aspect to the end-to-end computation. The problem with the end-to-end approach is that it requires even more data than the step-by-step approach and struggles to generalize. This is because the end-to-end nature means the input and output of the solver's neural network are high-dimensional, requiring considerable parameters to tune within. So far, end-to-end models only performed well in small scenarios with few constraints such as the traditional TSP [23], but will likely gain popularity as their efficacy improves.

2.2 Intermediary Points

Research in the optimization of routing models that contain some kind of Intermediary Point, where orders transfer between vehicles, fall into two main categories. The first is Trans-Shipment models, where the intermediary points facilitate transfer of orders between vehicles of the same type. The second is Multi-Echlon models, where orders transfer from one transportation type to another to reach their final destination [5]. The Pool Distribution Optimization Problem's Pool Points are an intermediary point that shares characteristics with both types of intermediary point.

An element adding significant challenge to the PDOP is the existence of pool points visited by vehicles with two different sets of behavioral restrictions. A location with the exact qualities of a pool point is not well represented in PDP research. Pool points pose a challenge to routing algorithms, which represent the locations in the problem as nodes. Nodes in the routing model can only be visited once each, allowing routing algorithms to simplify the problem significantly. The need to visit nodes multiple times also exists in models with trans-shipments, and researchers in this sub-field found a clever solution.

2.2.1 Trans-Shipments and Large Neighborhood Search

Trans-shipments are a transfer of orders between two vehicles at a designated intermediary transfer point. Using such a transfer point helps trucks maximize their own value and efficiency. David Wolfinger solved a version of the PDP with time windows, split loads, and trans-shipments using a Large Neighborhood Search (LNS) metaheuristic [42]. The main difference between LNS and local neighborhood search is the size of components changed. Wolfinger removes entire routes from the solution based on a percentage of total routes to remove. Using LNS allowed him to make significant changes to the solution in the exploration process, which is essential due to the complexity increase that allowing intermediary points creates. In his formulation, entire routes were removed with operations such as Random Removal, or Worst Route Removal based on cost savings, among others. Subsequently, the solution was repaired using a set of insertion heuristics. The insertion heuristics take advantage of the unique problem formation by using operators such as Best Insertion With Transshipment, which only considers insertions that would transfer an order between two vehicles [42]. These operations are difficult to explore, because they can change the optimal route for the truck receiving the order as well. Figure 2.1 shows the insertion algorithm's logic, which is more granular than the destroy operator, examining individual orders.

Wolfinger also makes a critical modeling decision to create a problem solvable using a LNS algorithm: Modeling the intermediary points as multiple nodes each. Creating copies of intermediary nodes allows multiple vehicles to visit the same intermediary point without violating the mixed integer programming setup of the problem [42], because every node can still only be visited at most once. In parameter tuning, Wolfinger also made an important discovery about route destruction.

Input: set of uncovered requests U, set of transshipment locations T
stack of trips $S \leftarrow$ empty stack, best trip $\mathcal{P}^{best} \leftarrow$ empty trip, best cost $u \leftarrow \infty$;
foreach *uncovered request* $(r^+, r^-) \in U$ *and* $t \in T$ **do**
    evaluate insertion of pair (r^+, t), and consider it as inserted at its best position;
    push (r^+, t) onto the stack of trips S;
    while S *not empty* **do**
        trip $\mathcal{P} \leftarrow$ top(S), and pop(S);
        if $C(\mathcal{P}) < u$ **then**
            if $\mathcal{P}$ *ends at* r^- **then**
                $\mathcal{P}^{best} = \mathcal{P}$ and $u = C(\mathcal{P})$;
            else
                foreach $t' \in \{\bar{t} \in T \mid \bar{t} \notin \mathcal{P}\} \cup \{r^-\}$ **do**
                    extend $\mathcal{P}$ with t' to obtain trip $\mathcal{P}' = (\mathcal{P}, t')$, and push it onto S;
return $\mathcal{P}^{best}$

Figure 2.1: Pseudocode for the insertion algorithm used by Wolfinger [42]

He got significantly better results by removing 10-20% of the solution's routes before repairing [42], where the actual percentage was sampled from a uniform distribution $\in$ [0.1, 0.2] at each iteration, which was compared to ranges from [0.0, 0.1], [0.2, 0.3], [0.3, 0.4].

Like the PDOP, Wolfinger's optimization problem is highly constrained, because it tracks capacity, distance, and time while facilitating intermediary points. He goes as far to to dub the problem with the acronym: **PDPTWSLT**, making it clear the constraints largely define the problem and his LNS based solution fits under the umbrella of Hybrid Heuristic Algorithms.

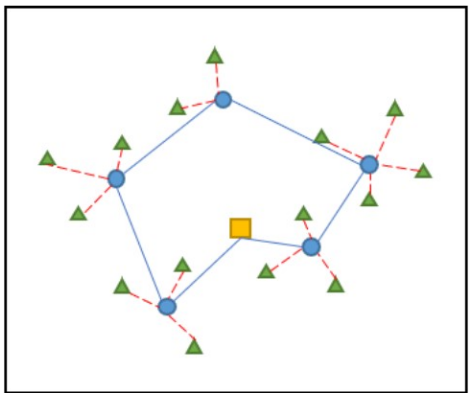
2.2.2 Multi-Echelon Vehicle Routing Problem

The Multi-Echelon VRP is a problem with two or more distinct components serviced by different vehicle types. These systems can be difficult to optimize because the results from one system affect the needs of another system within the model. Vakili et al. proposed a means of tackling such a problem by splitting it into two levels, and optimizing them separately [38]. In Vakili's research, the first layer optimizes depot placement and transportation routes, to supply each depot with orders. Then, the second layer optimizes the delivery from the depots to the respective destinations for the orders they contain. Breaking the optimization into

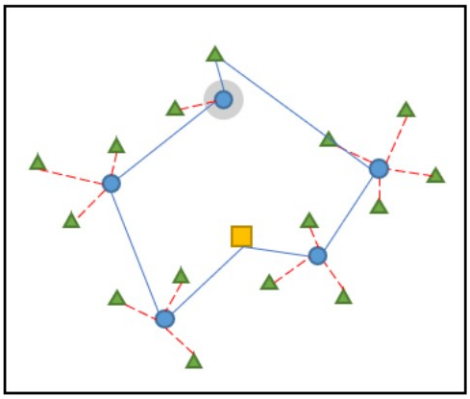
two components limits exploration of the true solution space, while drastically decreasing the problem complexity within each step [31].

Another interesting approach for modeling a two echelon routing optimization problem came from Lee et al., who used a novel heuristic approach based on memetic algorithms to solve a VRP with optional second-echelon delivery. In the problem, a distribution center has satellite depots, and all the deliveries to be made are assigned to the nearest depot. Then, a single vehicle finds the most efficient route to each depot as in the TSP, which forms the initial solution. In the start case, all deliveries are made by a second-echelon direct service. To incrementally improve, another algorithm goes through all the orders starting on the highest second-echelon delivery cost order, testing if delivering that order directly from the truck would be cheaper. During this test, genetic algorithms optimize the delivery truck's theoretical route and accept the change if the solution cost is reduced [22]. An important similarity between this problem and the PDOP, is the second echelon does not require additional optimization, because the routes are direct from the intermediary point. This allowed Lee at al. to solve the entire problem as a single model, although it is the culmination of four algorithms working together [22]. Figure 2.2 shows the solution in progress, where circles are local depots and triangles are the delivery locations. Each delivery is tested as part of the main vehicle's route, to see if it would be cheaper to deliver an order directly.

Although the second echelon pool trucks in the PDOP do not handle single orders, their behavior is deterministic given a set of demands from the assigned pool point, which also does not require its own optimization step. Unfortunately, the insertion algorithm Lee at al. used to assign orders to the delivery vehicle vs. second-echelon delivery is not subject to any capacity constraints, so it is not easily applied to the PDOP. This key difference relates back to the need for a robust constraint checking mechanism as part of the optimization process, a major benefit of using an off-the-shelf optimization tool.



(a) Route Graph Before Moving a Second Echelon Delivery to the Main Vehicle's Route



(b) The Graph after Insertion into the Main Route

Figure 2.2: Comparison Testing of Route Graphs during the Insertion Algorithm used by Lee et al. [22]

2.3 Key Takeaways

Developing a tool that solves the PDOP for Shipwell is about more than a theoretically optimal solution in testing. The final result must be a robust tool that is time efficient, consistent, scalable, and inexpensive to run. It is important to build off of an existing tool, such as Gurobi or OR Tools, to reduce the project's complexity and make it usable in production at Shipwell. Many intriguing choices made by researchers in modeling and search strategy for problems with important similarities to the PDOP could be applied to this problem. However, it is important to consider the differences in formulation between a given research problem and the PDOP, to determine if they invalidate the application of that strategy for the PDOP. Reviewing research in the field led to four important insights about multi-echelon, highly constrained routing optimization.

First, an algorithm that checks constraints before considering route options will prevent wasting time exploring the large infeasible areas in the solution space. When solving the PDOP, exploration is challenging because this problem is constrained in capacity dimensions and in vehicle behavior. For this reason, applying GAs might work within existing routes, but will explore the overall solution space very inefficiently. Some researchers apply GAs to solve sub-problems within a

Hybrid Heuristic Algorithm approach [35] such as rearranging a vehicle's route nodes, but it is inefficient for making global decisions in problems such as the PDOP. Given that GAs tend to strip constraints from the problem formulation, Learning-Based Optimizations are data-hungry, and end-to-end LBOs struggle outside the TSP; Hybrid Heuristic Algorithms are better suited for problems as constrained as the PDOP. Moreover, within the two most prominent professional tools for Hybrid Heuristic Algorithm routing optimization, GAs are not facilitated for local optimizations. This means using GAs would only be practical if building the optimizer around a GA focused tool, with individuals in the population representing solutions, which is a poor application of GAs for the PDOP.

Second, using duplicates of intermediary nodes, assigning one to each order, is a clever way of facilitating solutions otherwise not possible when using an optimization tool modeled on heuristic algorithms. Modeling decisions play a critical role in making the PDOP solvable using an off-the-shelf optimization tool. It allows the use of algorithms that rely on flow conservation constraints which prevent revisiting a node. In a problem where the second-echelon is optional, assigned intermediate node duplicates can even contain information about their assigned order. This could even facilitate optimizing both echelons concurrently without needing the custom-made insertion operators used by Wolfinger [42], although this is likely not scalable to the use case for Shipwell.

Third, calculating the cost of pool routes deterministically would simplify the solution space dramatically. Therefore, solving this model should take advantage of the simplicity in the second-echelon somehow. Leaning on Lee et al.'s problem formulation to eliminate the need for a second-echelon optimization step, by calculating the cost of pool routes deterministically, could simplify the solution space. Lee et al.'s second-echelon delivery working as a direct route only made the insertion cost independent of the overall solution, which allowed the cost to be checked during the optimization of the main vehicle. This is not possible in the PDOP because orders get consolidated on pool trucks, which is key to their efficiency. However, a deterministic heuristic for assessing the pool vehicle's cost could make the problem much more scalable, as Shipwell's need for the optimizer of 100-150 orders concurrently exceeds the maximum problem size in much of the research reviewed.

Fourth, Large Neighborhood Search allows highly constrained problems to explore a very tightly defined solution space. Within LNS, destroy operators do not violate capacity constraints, so they can work outside a highly constrained model. Insertion heuristics, on the other hand, can violate feasibility constraints and do not work well outside of an optimization tool that manages the problem's constraints. This opens the potential for problem-specific destroy operators and LNS destroy operators in general, which do not exist inside off-the-shelf professional routing optimization frameworks. This approach would allow application of Wolfinger's insight on highly constrained optimization to the PDOP, and for problem specific insight to influence the optimization tool's exploration through designed destroy operators.

Based on the related research, decisions on how to model the problem appear to be as important as methods for exploring the solution space. In particular, modeling decisions on ways to modify the problem's representation, attempting to make small alterations from the real-world problem while maximally reducing the routing model's complexity, will be a central contribution of this study.

Chapter 3

Data

This chapter explores the data provided by Whole Foods Market to help Shipwell build a tool to solve the PDOP, along with synthetic data used to create generalized test cases. Section 3.1 examines the real data, covering all the information provided with each order, along with steps to process additional features, combine similar orders and cluster the orders on a macro-level. Section 3.2 explains the process for generating synthetic data, used to improve the optimization tool's generalizability. Finally, section 3.3 gives an overview for creating scenarios used to develop and test the routing optimization algorithms in this study.

3.1 Whole Foods Market Data

Whole Foods Market's supply chain requires produce to be delivered from production centers to distribution centers across the United States. The contents are perishable, so they need to reach their destination quickly. However, there is opportunity to improve efficiency by consolidating orders at intermediary Pool Points, if possible within time constraints. Whole Foods Market operates six such pool points in areas with significant pickup demand for produce to be sold in Whole Foods Stores. Whole Foods Market has over a thousand produce orders to pick up daily, and so procurement for a single day must to be split into a group of scenarios which satisfy all the transportation needs for the day.

Figure 3.1 shows all produce orders that need to be shipped in one day, with the size of points corresponding to the log-scaled weight of the order. To develop

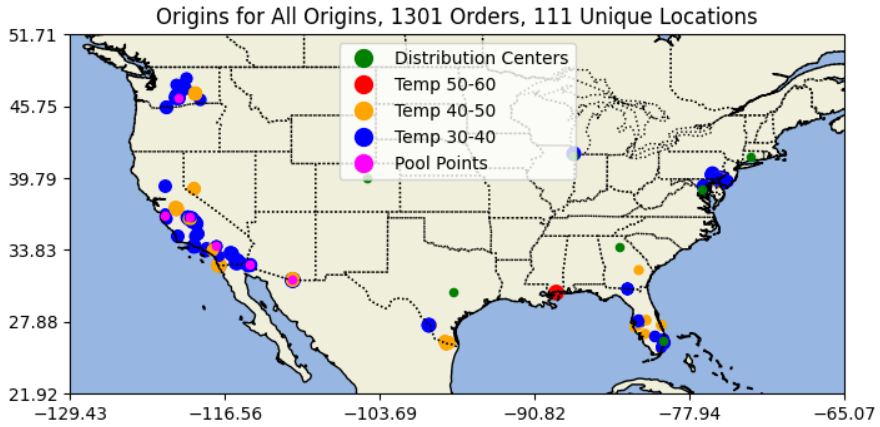


Figure 3.1: Origins of every order pickup for Whole Foods Market on Febuary 15th, 2024

an optimizer that can handle Whole Foods Market's needs, they provided complete order data for five of orders to fulfill, from February 7th, 8th, 9th, 12th, and 15th.

3.1.1 Order Sheets

Whole Foods Market's order sheets contain qualitative information about each order, along with its origin, destination, and shipping characteristics. The critical order information is the date, origin, destination, weight, volume, and temperature zone. Table 3.1 shows all the data entries provided, including qualitative descriptions that helped in the order bundling process.

3.1.2 Data Statistics

Whole Foods Market's order data follows a distribution that appears exponential at a glance. However, chi-square testing shows that an exponential distribution does not fit, with a p-value of zero, while the Kolmogorov-Smirnov test produced a p-value of 10.007e-83. Figures 3.2 and 3.3 show the distributions for weight and volume over all orders in the dataset, both are heavily skewed to the right.

Table 3.2 shows the inconclusive results testing for Exponential, Gamma, and Log-Normal distributions. The order data does not conform to any traditional parametric distribution. Between all five days order data, a total of 4487 orders

Table 3.1: All initial features for one Whole Foods Market shipping order

Field Name	**Data Sample**
Geo Range Name	PA_S
Commodity Group	Produce/Floral
Commodity Code	TROPICAL
Item Description	TROPICAL, BANANA, Yellow, SFG, 40lb
PO Status	Filled
Destination Account	WFM DCM Midwest Distribution Center
PO Master Name	5061010
Delivery Type	CPU
Origin Name	Earth University-DE1
Weight	40320
Units Ordered	960
Min Temperature	56
Max Temperature	58
Pallets	20
Shipping City	Wilmington
Pickup Date	2/15/2024
Delivery Date	2/17/2024
Shipping State	DE
Postal Code	19801

Note: All temperatures are in Fahrenheit.

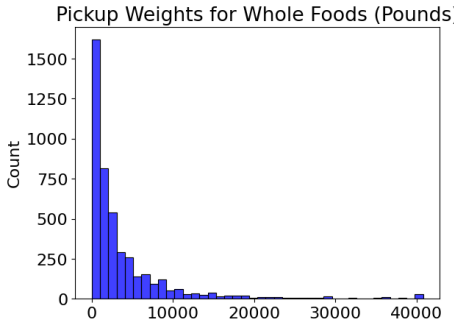


Figure 3.2: Order Weights in Pounds for WFM

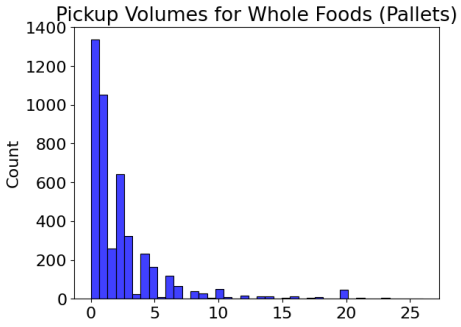


Figure 3.3: Order Volumes in Pallets for WFM

has a weight mean of 3886.49 Pounds with a median of 1764.00, while the mean volume is 2.35 Pallets with a median of 1.00.

Table 3.2: Test Statistics and P-Values for Order Weight and Volume Distributions

Test Type	**Order Weights**		**Order Volumes**	
	Test Statistic	**P-Value**	**Test Statistic**	**P-Value**
Chi-Square EXP	1.188e+16	0.000	2.948e+14	0.000
Kolmogorov-Smirnov EXP	0.146	1.006e-83	0.162	3.044e-103
Kolmogorov-Smirnov Gamma	0.051	1.130e-10	0.109	8.793e-47
Shapiro-Wilk Log-Normal	0.826	7.799e-57	0.934	1.033e-40

Although the data does not follow any known distribution, notably there is a high incidence of very small orders, and a low incidence of very large orders. Whole Foods Market often has small orders, but generally at locations where many similar orders need to be picked up concurrently.

3.1.3 Feature Processing

The raw data from Whole Foods Market needed key processing steps to facilitate scenario generation. Whole Foods Market operates six pool points in the Western half of the United States and ten Distribution Centers spread across the country. The first step was to obtain the coordinates for each Whole Foods Market distribution center and each pool point. These were used to append the provided order data with GPS coordinates for the destinations, by identifying an order's destination name and inputting the corresponding coordinates. The orders were then assigned to the nearest pool point based on euclidean distance as in equation

3.1.1 where p is the assigned pool point and P_c is the set of all pool point coordinates, with the exception of order pickups east of the Latitude Line at -105.0. Exception orders were too far from any existing pool point and were assigned to pool point "zero". Whole Foods Market's six pool points operate on the Western half of the United States, so pickups east of line -105.0 need not be included in Pool Distribution Optimization Scenarios.

$$p = \min_{y \in P_c} \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \tag{3.1.1}$$

3.1.4 Data Partitioning

Daily order volumes at Whole Foods Market are too high for a single optimization scenario. To create sets of orders that fit within the optimizer's capacity, while making the best opportunity to completely fill vehicles, orders must be partitioned to facilitate the lowest net cost to transport all orders. For example, if two orders have pickup locations on opposite sides of the United States, they cannot end up on the same truck; so there is little benefit to placing those orders in the same scenario. Orders can be partitioned into groups by geographic area, assigning orders to the nearest pool point, then optimizing scenarios with orders from individual pool points.

Whole Foods Market's produce origins have distinct areas, but these areas overlap in high population regions. This is exemplified by the Southern half of California containing four separate pool points. In these regions, orders from neighboring pool points may benefit from being optimized in the same scenario. This can increase efficiency, if a pool truck at a neighboring pool point has unused capacity while sending the order through the local pool point would require activating an additional truck.

3.1.5 Order Clustering

A notable fact for the produce order data at Whole Foods Market is that often, multiple small orders going from the same origin to the same destination appear on the same day. Groups of orders like this can count as high as ten, while overall taking up less than a single truck's capacity. Orders like these reduce the efficiency

of the optimizers. Their capacity is closely aligned with the total number of orders in the scenario, because the constraints in the optimizers scale either linearly or quadratically with the number of orders, depending on the implementation.

To solve this issue, small orders with identical origins and destinations can be bundled, while keeping the bundles small enough to be arranged inside a truck with other orders. This clustering works by first grouping orders with the same origins and destinations. Within these groups, orders are sorted by size in ascending order, then aggregated into separate bundles. Whenever one of these bundles exceeds 10% of a truck's total weight or volume capacity, it is finalized into a single order. Moving up the order list, once the individual orders reach 10% of a truck's capacity, the bundling stops and the remaining orders are finalized as they came. This bundling process reduced the number of orders by 67% on average, allowing an entire day's orders to be handled in a single optimizer scenario for each pool point.

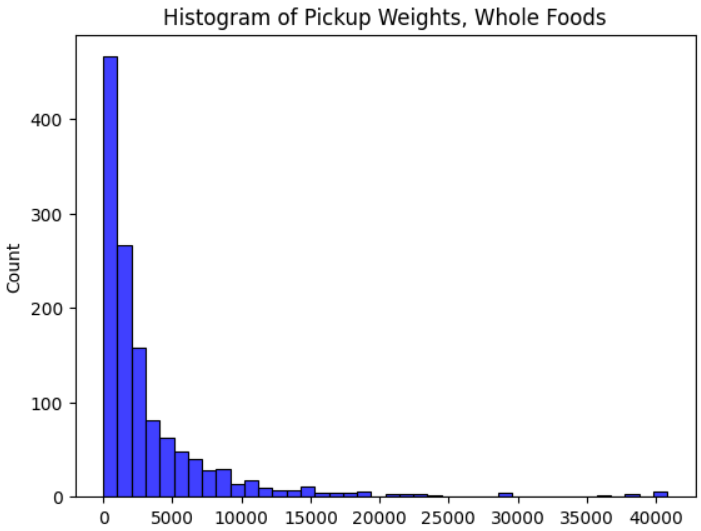


Figure 3.4: Order Weights in Pounds after clustering for each Whole Foods Market order on Febuary 8th, 2024

Figure 3.4 shows a histogram, giving a rough distribution on the order sizes by weight. Bundling the orders together produces a smoother-looking distribution that is less skewed, while reducing the number of orders significantly. As either weight or volume capacity can trigger the 10% clustering margin, many orders

remain under 10% of a truck's weight capacity of 42,000.

3.2 Synthetic Data

Although Whole Foods Market's needs as a client of Shipwell catalyzed development of the Pool Distribution Optimizer, Shipwell plans to offer other customers this service in the future. Anticipating the needs of other Businesses that need Pool Distribution Optimization necessitated synthetic data. The synthetic dataset allowed the testing of scenarios outside of Whole Foods Market's particular use-case. Key differences to test were: scenarios having fewer orders with wider distribution, pool points located in non-agricultural regions, and neighboring pool points with overlapping pickup distribution zones.

3.2.1 Pool Points and Destinations

Pool points and distribution centers need to be placed realistically, because the optimizers use heuristic estimated driving distances instead of haversine distance. This means the tendency that logistics hubs and major highways are near large cities must be represented in the synthetic data. Having a logistics hub in an unpopulated and poorly connected area in the US is both unrealistic and negatively affects the accuracy of driving distance estimation. To generate synthetic locations for the pool points and distribution centers, each location starts out as a random selection from one of the 200 most populated cities within the United States, in addition to the 25 largest Cities in sparsely populated states. The coordinate for the chosen city is taken, then displaced by a distance chosen from an exponential distribution, at a random angle from the original coordinate. The displacement mimics the tendency for these hubs to be located just outside large cities.

Figure 3.5 shows the locations of these cities on a Map of the United States, they are the same cities used to create a routing distance matrix between key points in the US.

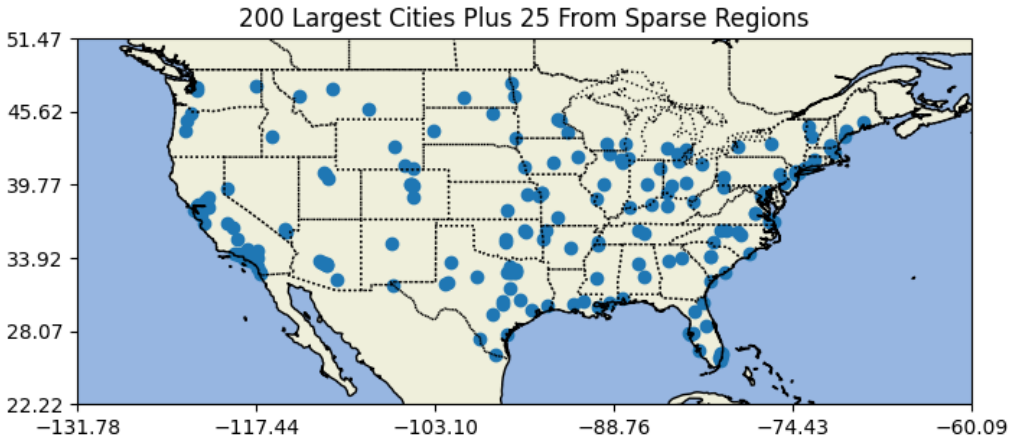


Figure 3.5: Geographic Coordinates for the Cities Chosen

3.2.2 Pickup Distributions

Pickup points in the synthetic data are clustered around a center point. This represents the expectation that a company will establish pool points near the center of areas with a high volume of orders. From this point, the pickup positions are randomly displaced according to an exponential distribution parameterized by:

$$f(x;\lambda) = 0.01535 \cdot e^{-0.01535x} \tag{3.2.1}$$

This distribution produces an expectation that 99% of orders will be within 300 miles of the center. This distance is applied from the center at an angle randomly selected from a uniform distribution. The center of each distribution is also a pool point; unless the scenario does not allow the use of pool points, then that point only acts to center the distribution of order pickups.

Any coordinate generated is checked to be within the United States, and reselected if not. This ensures no points are across a border, in the ocean, or on a lake. This step is particularly important, because the optimizers use real driving distance estimations and not the haversine or euclidean distance between coordinates. Figure 3.6 shows a distribution of order pickups created around a pool point outside Phoenix, Arizona. The pickup coordinates range enough in distance that some orders may not be routed through the pool point.

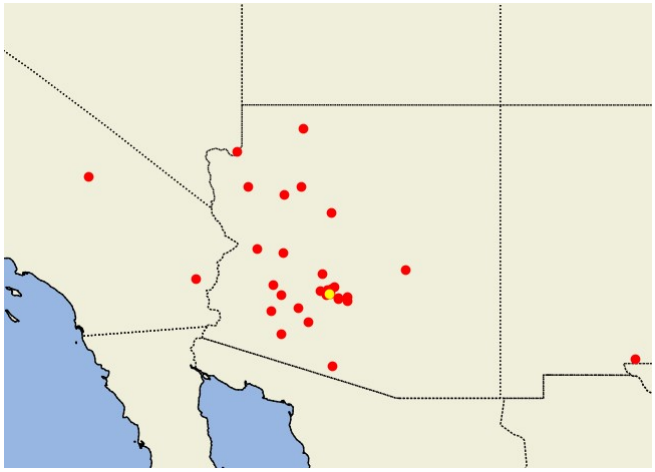


Figure 3.6: Geographic Coordinates from an exponential distribution of synthetic order pickups

3.2.3 Order Sizes

The synthetic orders need to have both a weight in pounds and a volume in pallets, which were initialized using exponential distributions to achieve a very high incidence of small orders and low incidence of large orders. In the Whole Foods Market use case, many orders shared the same origin, allowing efficient bundling for similar small orders. However, in the synthetic data, a generalized case where each order has a unique origin is used. The weights and sizes for these orders are sampled from the exponential distributions seen in equations 3.2.2 for weight, and 3.2.3 for number of pallets.

$$f(x;\lambda) = 0.0001096 \cdot e^{-0.0001096x} \tag{3.2.2}$$

$$f(x;\lambda) = 0.1645 \cdot e^{-0.1645x} \tag{3.2.3}$$

These distributions have their Cumulative Distribution Function's 99th percentile set to the capacity limit for standard American freight trucks, which is 42,000 Pounds and 28 Pallets respectively. Any value over this limit is clipped to the vehicle limit, and any value close to zero is rounded up to one. Although Whole Foods Market usually has numerous pickups from the same location, to develop a more generalized optimizer, orders in the synthetic data can be smaller

than the clusters made with Whole Foods Market data. This models that for some businesses, small orders need to picked up from unique locations.

3.3 Scenario Creation

Each optimization problem starts with a scenario. The scenario is modeled as an object, containing all the necessary variables to pass into the optimizer, including the distance matrix between all the points in the scenario. The distance matrix is populated with distances of routes that contain three legs; the haversine distance between between the coordinate and the nearest point on the major point distance matrix, the pre-calculated distance between the major points nearest to the origin and destination, and the haversine distance between the second major coordinate and destination. The logic behind this approximation method will be explained in the next chapter. Table 3.3 shows the variables contained in the scenario that are passed to the optimizer.

Table 3.3: Scenario Variables Passed to the Optimizers

Variable Name	Definition
Distance Matrix	Matrix containing distances between all locations, used to compute travel costs between nodes.
Number of Vehicles	Total number of vehicles available for routing.
Depot Node	The starting point, where all vehicles begin their routes.
Order Count	The total number of orders to be delivered, used to calculate pickup and delivery indices.
Valid Pool Points	List of pool points available for consolidating orders.
Pickup and Delivery Pairs	Pairs of pickup and delivery nodes for each order.
Max Travel Distance	Maximum allowable travel distance in miles for each vehicle.
FTL Price Matrix	Cost matrix for Full Truck Load routes between nodes.
Pallet Demand	Pallet demand at each location, representing the number of pallets to be picked up or delivered.
Demand at Nodes	The weight demand in pounds at each node, which vehicles must fulfill.
Vehicle Pallet Capacity	Maximum pallet space capacity of each vehicle.
Vehicle Capacities	Maximum capacities for each vehicle in terms of load they can carry.
Time Matrix	Matrix representing travel times between all locations.
Time Windows	Time constraints for each location, specifying when deliveries or pickups must occur.
Maximum Stops	The maximum number of stops each vehicle can make.
Pooling Enabled	Flag indicating if the pooling of orders at pool points is enabled.
Assigned Pool Points	Pre-assigned pool points for delivery orders if pooling is enabled.
Location Count	Total number of locations in the scenario, including depot, pickups, deliveries, and pool points.

Chapter 4

Driving Distances

This chapter explains the motivation and compares two approaches for quickly approximating the driving distance between any two coordinates in the United States. Both strategies use the underlying concept of creating a large node network representing the US highway system, routing important segments, storing just the routed segments, and fusing components together for a quick approximation during optimization. Section 4.1 explains the method for developing a routing heuristic, and compares two sets of coordinates as a basis for the pre-calculated route segments. Section 4.2 then explains the testing protocol and briefly covers the results, which dictate the final distance approximating heuristic tool's implementation. This tool is used during the remainder of the study, to calculate driving distance when testing optimization algorithms.

4.1 Driving Distance Approximation

Shipwell's customers may be sending orders across the United States, clearing major geographic obstructions and following the most efficient route through the US highway system. It is advantageous to move away from route optimization methods that rely on haversine distance between locations, as they can draw lines over areas that no highway crosses, or miss that a route is obstructed by a body of water. On the other hand, highly accurate tools like the Google Maps API are extremely costly for optimization, when N^2 routes must be calculated produce a distance matrix for an optimization scenario. Shipwell already spends tens of thousands annually on Google Maps API calls, even without using it for optimization.

To obtain near Google Maps API levels of accuracy without the cost, the author chose to develop a heuristic tool that combines haversine distance with predetermined routes between key points in the United States. With this estimator, the optimizer can map pickups, destinations and pool points to their nearest respective coordinates on a pre-calculated distance matrix. In this way, the optimizer can graft the direct distance between the start and finish of a route to their nearest coordinates in the distance matrix, leading to a three part route, where the middle section is a real route between two points connected by the US highway system.

With the use of open source tools, driving routes between points can be accurately computed without the use of costly API calls. However, this process takes significant time, memory, and computational power, making it impractical at the time of optimization. Instead, creating a node network representing the US highway system, deciding key points to include in a distance matrix, and computing the shortest route through the network between each point allows for quick results through reuse. This yields both an accurate distance between the points and a route that can be displayed when visualizing the solution to an optimization problem.

4.1.1 OSMnx Node Networks

To assemble a node network that creates realistic routes, a network that contains the most important roads for long distance travel while minimizing extraneous connections and low speed limit roads is essential. The open source tool **Open Street Maps** with **networkx** (OSMnx) has the capacity to construct node networks using road types and geographic areas, to filter out weakly connected parts of the resulting network, and to find the shortest path between two nodes in the network. These qualities make it an optimal selection as a routing tool. With distribution networks spanning the United States, a high level of granularity in the node network would produce an untractably large graph. For this reason, only highway roads need be part of the network. However, the US highway system has five types: Interstate Highway, U.S. Highway, State Highway, Highway Connector, and Trunk Road [7], so the best level of detail must be investigated. Figure 4.4 shows three different levels of detail outside Dallas, which is known for the large highway network surrounding it. Notably, figure 4.3 includes roads with

lower speed limits than the other highway types. Using this network to calculate routes requires optimizing travel time over distance traveled, making best route calculation more difficult.

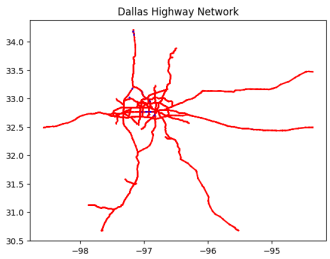


Figure 4.1: Three types

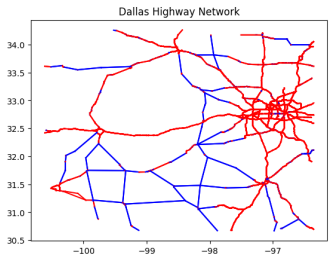


Figure 4.2: Four Types

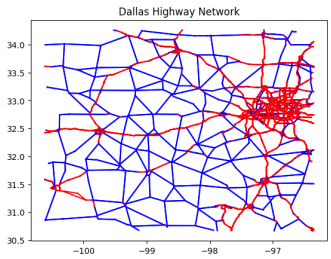


Figure 4.3: All Five Types

Figure 4.4: Node Networks for the Highways around Dallas at three levels of granularity

Very large node networks can be difficult to find the shortest path through, but finding routes between major points should not become its own difficult optimization problem. More roads alone does not imply better calculated routes, as the route calculation algorithm struggles in networks with too many connections. On the opposite end, a network too sparse can remove important links through weakly connected areas. Figure 4.5 shows what can happen when a route calculation between Dallas and New York goes off course, while figure 4.6 displays the correct route on a more sparse network.

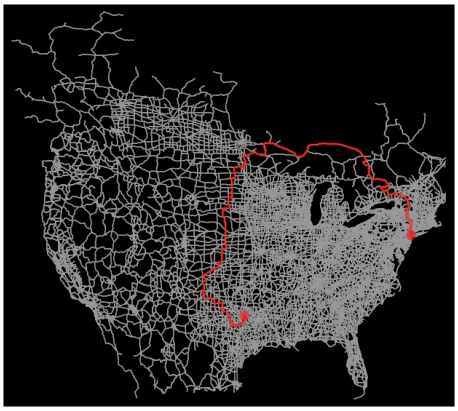


Figure 4.5: Route calculation error on a higher Granularity Network

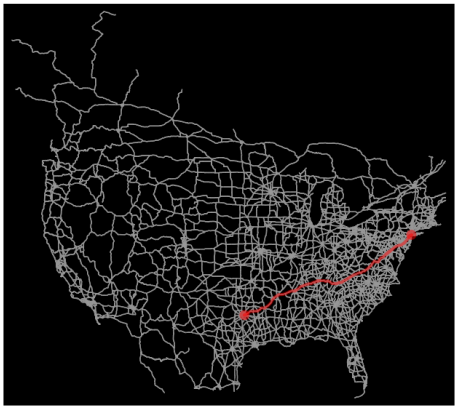


Figure 4.6: The same Points, accurately Mapped with less Granularity

OSMnx can calculate the shortest route between two points with the A* algorithm, which uses a euclidean distance heuristic in conjunction with Dijkstra's algorithm to focus the search towards its destination [17]. This allows OSMnx to find the shortest path through a large node network, whereas implementing Dijkstra's algorithm naively would have computational complexity of $O(V^2)$ [6], with a highway network consisting of 274,062 vertices.

The final road network is pictured in figure 4.6, including Interstate Highway, U.S. Highway, State Highway, and Highway Connector roads. With this network, a distance matrix and a route matrix can be produced between any two points in the United States. A key consideration is determining which set of coordinates will yield a matrix that generates the most accurate routes for real customer data. Plugging every coordinate combination and the highway node graph into OSMnx and using the A* algorithm produces all the routes needed for the matrix. Then, a summation of the edges in each route provides the route length.

4.1.2 Zip Code Clustering

The continental United States has 41,642 zip codes. Zip codes were designed with US Postal Service logistics in mind [37], so they each cover an area of similar population size, making them higher density in more populated areas. Zip codes are also located within geographic boundaries, so they are not centered on impassable areas [37]. Considering that the zip codes cover all the land in the US, an efficient clustering of these zip codes' coordinates would leave every conceivable point in the continental United States within range of the coordinate for its respective zip cluster center.

A distance matrix with even 1000 zip clusters would require calculating a million routes, each containing hundreds of node coordinates. The amount of computational power required to produce the matrix becomes a factor, in addition to the file size increasing to multiple gigabytes. Clustering the zip codes optimally requires balancing the computational load with the degree that each cluster center adequately represents the zip codes within. To find this balance, the gap statistic for within-cluster dispersion can be examined at different counts of clusters. The gap statistic is computed by comparing variance at different numbers of clusters

for the data with the variance in randomly generated noise with the same number of clusters [34]. This noise is created using the same range as the data, with a uniform distribution. As the number of clusters increases, the gap statistic should increase, but at some point, the the gap statistic's increase will slow. At that number of clusters, the optimal balance between minimizing total clusters and maximizing explained variance in the data is reached. In a plot of number of clusters against the gap statistic, the curve is expected to follow an elbow, where the bend in the elbow is that optimal number of clusters.

Testing then shows the distribution of zip code centroids does not show a clear peak in gap statistic, as the zip codes are intentionally dispersed so that none directly overlap in space. Although some zip codes are closer together than others, adding more centroids consistently increases the gap statistic so long as the number of clusters does not approach the actual number of zip codes. Instead, knowing that 200-300 locations is in the computationally feasible range for constructing a route matrix, the most significant drop in first derivative on the gap statistic is computed in that range. This point was at 214 clusters, giving the base dimension for the zip code cluster based route coordinate matrix, [214x214]. Figure 4.7 shows the gap statistic curve for testing up to 5,000 clusters. For each number of clusters, the data was clustered and compared to three different clustered sampled distributions of synthetic noise.

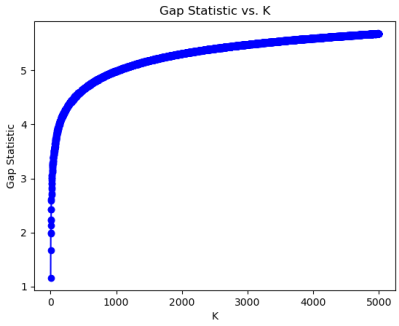


Figure 4.7: Gap Statistic curve for up to 5,000 clusters

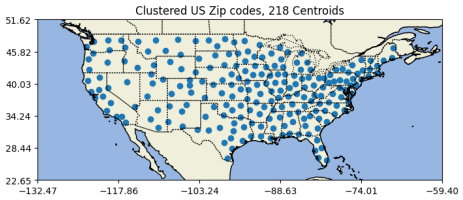


Figure 4.8: 218 Zip Code cluster centers mapped

Figure 4.8 shows the centroids that best explained variance in the coordinates for all continental US zip codes. The coordinates are distributed evenly with respect to their neighbors, but less populated regions in the western half do have

fewer assigned centroids. From this set of coordinates, a set of 47,524 routes is calculated.

4.1.3 Major City Matrix

Although clustering zip codes creates a well dispersed map that would perform well routing random coordinates, the order data shows a reasonable pattern of locating near larger cities. This is clear in the Whole Foods Market order origin map from figure 3.1, where most orders are located near large cities. Taking the largest cities in the United States as major coordinates for the distance matrix trades geographic dispersion for likelihood of proximity to real orders. This provides more precise routing in high population zones with numerous large cities. Viewing the 200 largest city coordinates does leave some states with no representation. However, adding 25 of the largest cities in low population areas greatly reduces the maximum distance an origin can be from a major coordinate. Figure 3.5 depicted the distribution of cities, which appealingly, has city clusters located near the dense order clusters in the data for Whole Foods Market from figure 3.1.

The distance and route matrices have to be tested against real order data and synthetic data, by comparing them to the routes produced by the Google Maps API. The optimizer doesn't need to know the coordinates in a route, but they are necessary to display solutions to the scenarios tested. Seeing the solution on a map is the quickest way of catching undesirable behavior.

4.2 Route Matrix Testing

The two route matrices are tested against the Google Maps API to determine which to use during development of the Pool Distribution Optimizer. For this test, 500 pickup and delivery coordinates were generated from synthetic data and 500 were sampled from Whole Foods Market orders. Figures 11.5 and 11.6 in the Appendix visualize the order positions for the synthetic coordinates and Whole Foods Market sampled coordinates.

Table 4.1 shows the similarity scores of the Zip Cluster Distance Matrix, the Major City Distance Matrix, and Haversine distance against the Google Maps API. Similarity is calculated by the formula $1 - \frac{abs(d_1 - d_2)}{d_1}$, where d_1 is the Google

Maps API distance and d_2 is the comparison distance. Notably, the major city matrix scored higher average similarity on both data types. Kernel Density Estimate figures 11.7 and 11.8 in the Appendix show the city matrix estimate similarities approach 100% more often than the zip cluster matrix. These results show the major city matrix is the more accurate distance matrix, but they both far outperform the haversine distance currently used by Shipwell's routing systems. Figure 11.9 in the appendix shows the confidence intervals on accuracy for each distance approximation, showing how far away haversine distance is in accuracy. Moving forward, it is clear that the Major City Distance Matrix is reliably the most accurate approximator, and should be used for future routing optimization experiments. Cutting combined average discrepancy from 17.46% to 6.52% in the routing tests, the Major City Matrix as a distance approximator is projected to greatly reduce the discrepancy between predictions and real driving distances.

Table 4.1: Similarity Comparison of Distance Approximation Strategies for Synthetic and WFM Data

	Synthetic		**WFM**	
Approximator	**Similarity (%)**	**Std Dev**	**Similarity (%)**	**Std Dev**
Haversine	81.75	0.2751	83.34	0.2097
Zip Cluster	91.84	0.6911	86.30	1.1322
Major City	93.82	0.6376	93.14	0.3536

Chapter 5

Modeling

Modeling the Pool Distribution Optimization Problem requires a theoretical mathematical formulation for the problem. However, creating a routing model based off a PDOP scenario requires additional modeling decisions on how to represent the solution space. From there, modifications to the original problem result in models that facilitate cost optimization on the scale needed by Shipwell. In this chapter, section 5.1 covers the terms and mathematical constraints at the core of the PDOP. Then, section 5.2 explains how the PDOP is modeled using the OR Tools Routing Library, paying special attention to the formulation of a two-stage optimization process achieved by splitting the vehicle types into two optimization steps. Section 5.3 proposes an external Large Neighborhood Search mechanism to further improve on the first OR Tools optimizer. And finally, section 5.4 details a more highly constrained model using Gurobi's MIP Solver, where both vehicle types are optimized simultaneously.

5.1 Mathematical Formulation

The mathematical formulation posed by the Pool Distribution Optimization Problem incorporates behavioral rules in terms of travel and contents for two types of vehicles through linear constraints. Different modeling techniques can lead to different abstractions, though they represent the same formulation seen below. Additionally, some of the models in this study modify the original problem to help find better solutions, but the question of solution's feasibility is always answered by satisfying the exact constraints provided in this chapter.

5.1.1 Sets, Parameters, and Variables

Table 5.1 lists the Sets, Parameters, and Variables that make up this optimization problem. The values for each parameter are known once the scenario has been constructed, as they are either present in the data used to construct the scenario, or can be calculated from the scenario's data without knowing the solution. The variables all stem from the values $x_{ij}^v \in X$, which are boolean variables that make up a tensor X representing all routes for all vehicles. In other words, the other seven variables are used to simplify the expression of constraints, but are all derived from a combination of the values for x and the scenario parameter values. Furthermore, the optimizers tested in this study all make routing decisions regarding the values of x, or an equivalent abstraction of X such as lists of nodes forming routes.

5.1.2 Objective Function

The overall goal of the routing optimizer is to find a solution which minimizes the expected cost of servicing all orders in the scenario. Although many metrics have a correlation with the objective function, ultimately, the solution to the optimization scenario and its cost can be represented as an element-wise product. This product is between a tensor of boolean variables X, which corresponds to the each vehicle and all connections between all nodes, and a matrix that represents the cost to travel between any two nodes in the scenario.

- Minimize Cost:

$$\sum_{v \in V} \sum_{i \in O_M} \sum_{j \in L} d_{ij} x_{ij}^v R + \sum_{p \in P} \sum_{j \in J} n_{pj} f_{pj}$$

Business Goals

The behavior of routes are subject to company analytics, with business goals outside the optimizer's objective function, that are used to compare solutions for a scenario obtained from different models. Including these metrics as variables in the optimizer, and penalizing the objective function when crossing undesired boundaries did not impact the behavior of the optimizer. However, these metrics are still of important business value to Shipwell and their clients. Additionally,

Table 5.1: Notations for the Problem Formulation

Sets	
L	All Locations, l $\in O \cup P \cup J$
M	Orders, m $\in M$
O	Unique Origins, o $\in O$
O_m	Order Origins, $o_m \in O_m$
V	FTL Vehicles, v $\in V$
V_2	Pool Vehicles, $v_2 \in V_2$
J	Distribution Centers (DC), j $\in J$
J_m	Order Destinations, $j_m \in J_m$
P	Pool Points (PP), p $\in P$
X	Connections Between Nodes for All Vehicles, $x_{ij}^v \in X$
Parameters	
f_{pj}	Fixed route costs from PP to DC, $\{p \in P, j \in J\}$
d_{pj}	Distance from PP to DC, $\{p \in P, j \in J\}$
d_{ij}	Distance between nodes i and j, $\{i \in L, j \in L\}$
t_{ij}	Travel time between nodes i and j
o_m	Origin for order m
j_m	Destination for order m
s_m	Demand of size units in order m
w_m	Demand of weight units in order m
w_v	Weight capacity for Truck $v \in V \cup V_2$
s_v	Size unit capacity for Truck $v \in V \cup V_2$
t_s	Wait Time at Vehicle Stops
R	FTL Rate per mile
S	Maximum pickup stops allowed
D	Vehicle max travel distance
T	Time limit
Variables	
x_{ij}^v	If node i and node j are connected by vehicle v, $\{i \in L, j \in L, v \in V \cup V_2\}$
v_m	FTL Truck v assigned to order m, $\{v_m \mid m \in M,\ v \in V\}$
v_{2m}	Pool Truck $v2$ assigned to order m, $\{v_{2m} \mid m \in M,\ v_2 \in V_2\}$
d_v	Distance traveled by Truck $v \in V$
l_v	Number of distinct locations visited by vehicle, $v \in V \cup V_2$
t_v	Route start time for vehicle, $v \in V \cup V_2$
p_v	Pool Point visited by truck v $\{p \in P\}$, 0 otherwise
n_{pj}	Number of pool trucks going from p to j, $\{p \in P, j \in J\}$

capacity utilization is used an an input for some destroy operators in the iterative version of the OR-Tools optimizer.

• Maximize peak truck weight capacity utilization per mile traveled (second sum is the pool routes):

$$\left[\sum_{v\in V}\sum_{m\in M|v_m=v} w_m\cdot\frac{d_v}{w_v} + \sum_{p\in P}\sum_{v\in V|p_v=p}\sum_{j\in J}\sum_{w_m|v_m=v,j_m=j} w_m\cdot\frac{d_{pj}}{w_v}\right] \div \left[\sum_{v\in V} d_v + \sum_{p\in P}\sum_{j\in J} n_{pj}d_{pj}\right]$$

• Maximize peak truck pallet capacity utilization per mile traveled (second sum is the pool routes):

$$\left[\sum_{v\in V}\sum_{m\in M|v_m=v} s_m\cdot\frac{d_v}{w_v} + \sum_{p\in P}\sum_{v\in V|p_v=p}\sum_{j\in J}\sum_{s_m|v_m=v,j_m=j} s_m\cdot\frac{d_{pj}}{w_v}\right] \div \left[\sum_{v\in V} d_v + \sum_{p\in P}\sum_{j\in J} n_{pj}d_{pj}\right]$$

• Minimize number of trucks that travel:

$$\sum_{v\in V}\min\left(1, \sum_{i\in O_M}\sum_{j\in L} x_{ij}^v\right) + \sum_{p\in P,\, j\in J} n_{pj}$$

Truck minimization is an important aspect of solutions, as pickups are usually located near pool points, but the destinations are spread out far from the Pool Point's general area. Reducing the number of trucks needing to leave the Pool Point zone towards a destination by just one generally improves the overall cost of the solution by thousands.

5.1.3 Constraints

The Pool Distribution Optimization problem is a PDP, with capacity measured in both metrics of weight (in pounds) and volume (in pallets). It is also a multi-echelon routing problem, where each vehicle type has specific constraints on the order and type of locations they may travel. These constraints are listed below as mathematical formula.

• All orders must be picked up by an FTL Vehicle once:

$$\sum_{v\in V}\sum_{i\in L} x_{ij}^v = 1,\ \forall j \in O_M$$

• All orders must be dropped off by a Vehicle once:

$$\sum_{v\in V\cup V_2}\sum_{i\in L} x_{ij}^v = 1,\ \forall j \in J_M$$

• FTL Trucks must only pickup orders bound for the same destination unless visiting a pool point:

$$j_{m_1} = j_{m_2}, \quad \forall m_1, m_2 \mid v_{m_1} = v_{m_2} = v, \ \ \forall v \in V \mid \sum_{i \in O} \sum_{j \in P} x_{ij}^v = 0$$

• FTL Trucks cannot drive between PP and DC, or from those to origins:

$$\sum_{i \in J \cup P} \sum_{j \in L} x_{ij}^v = 0, \ \forall v \in V$$

• FTL Trucks cannot make more than the allowed number of pickup stops:

$$\sum_{i \in O} \sum_{j \in O | d_{ij} > 0} x_{ij}^v < S, \ \forall v \in V$$

• Active FTL Trucks must visit either one destination or one pool point only:

$$\sum_{i \in O} \sum_{j \in J \cup P} x_{ij}^v \leq 1, \ \forall v \in V$$

• Pool Trucks cannot drive from DC to PP:

$$\sum_{i \in J} \sum_{j \in P} x_{ij}^v = 0, \ \forall v \in V_2$$

• Pool Trucks cannot drive to origins:

$$\sum_{i \in L} \sum_{j \in O} x_{ij}^v = 0, \ \forall v \in V_2$$

• All Truck routes must run within time limit, including 4 hour waits at stops:

$$t_v + \sum_{i \in 0 \cup P} \sum_{j \in 0 \cup J \cup P} t_{ij} x_{ij}^v + t_s l_v \leq T, \ \forall v \in V \cup V_2$$

• Trucks must not surpass max travel distance:

$$\sum_{i \in 0 \cup P} \sum_{j \in 0 \cup P \cup J} d_{ij} x_{ij}^v \leq D, \ \forall v \in V \cup V_2$$

• Trucks must not surpass Weight Capacity:

$$\sum_{m | v_m = v} w_m \leq w_v, \ \forall v \in V \cup V_2$$

• Trucks must not surpass Pallet Capacity:

$$\sum_{m | v_m = v} s_m \leq p_v, \ \forall v \in V \cup V_2$$

Behavioral Parameters

Setting up a routing optimizer requires real information about freight logistics rules, such as the standard capacity of freight vehicles and the maximum number of stops a business allows their trucks to make in one route. The parameters chosen for the optimization models in this study are a combination of business rules from Whole Foods Market and freight information from Shipwell's extensive experience in the field. The parameters are shown in Table 5.2, with all values static except maximum stops, which varies greatly depending on the business. This is due to the balance between efficiency and expediency. For example: Whole Foods Market, sourcing produce, cares more about the product arriving fresh and prefers to allow fewer stops for their trucks.

Table 5.2: Scenario Parameter Values

Parameter	Value
Truck Weight Capacity (Pounds)	42,000
Truck Volume Capacity (Pallets)	26
Vehicle Max Distance (Miles)	3,000
Maximum Stops per Truck	5-10
Pool Route Discount	10%
Truck Rate Per Mile (Dollars)	2.0

The routing optimizer is set up to allow a heterogeneous fleet of vehicles, as each vehicle's dimensional limitations (size, weight, and distance) can be input separately. However, optimizing with a variety of dimensional limitations will not be tested because in the use case for Shipwell, freight vehicles all have roughly the same limitations. The notable exception is maximum allowed stops, as this is part of an individual company's business rules. To allow for a wider variety of solutions, one FTL vehicle and one pool vehicle are added to the optimizer for each order. That way, potentially each order could travel on its own truck, if that would lead to the best objective value.

5.2 OR Tools

Google's OR Tools package has a **Constraint Programming** module that is both open source and frequently used by companies to optimize their logistics. Shipwell uses an OR Tools based optimizer to create freight routes without pool points, and wants an additional tool that can handle the complexity of Pool Distribution Optimization. The initial goal for the Pool Distribution Optimization project was to develop an optimizer that works using OR Tools, but the requirement was relaxed due to the difficulty of this optimization problem. However, most of the modeling and experimenting in this project remains focused on OR Tools.

5.2.1 Single Echelon Restructure

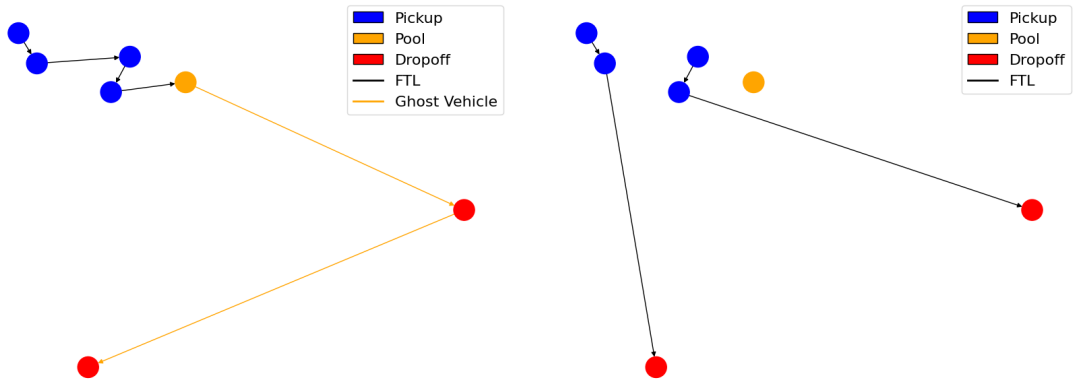
Solving this optimization problem is about more than modeling the listed constraints and pressing "play". The amount of constraints in this formulation prevents traditional optimization algorithms from exploring to find an acceptable solution. The optimizer Shipwell had in development tried to follow this approach, and was unable to find acceptable solutions for even simple scenarios with ten orders. This model was abandoned, after testing at the beginning of this research project proved the model wholly inadequate, prompting a new start from scratch for this study. Using constraints to create two different classes of vehicles leads to a highly constrained CVRP, whose solution space is very narrow, making any local search algorithm struggle to escape from local optima [26]. The problem formulated with these constraints is so difficult to solve, that modifying the formulation to better allow for solution exploration is critical for the model to computationally scale up to the needed size. To make a problem that's solvable at scale using constraint programming, the formulation must be modified so demand in the system is no longer dynamic.

To escape from dynamic demand, each of the two vehicle types can be optimized separately. This approach takes inspiration from the work of Lee at al. [22] where an estimation of the optional second-echelon delivery component simplified optimization of the primary vehicle's route. If given the routes of all FTL vehicles in advance, optimizing the pool vehicles' routes is rather straightforward. Each pool vehicle can only travel one leg, from a pool point to a distribution center, so only

the time window to operate each pool vehicle needs to be optimized. Therefore, if the role of the FTL trucks can be optimized reasonably well by itself, the solution will remain efficient, while allowing more exploration through local search. To optimize the FTL level of the routes, the optimizer needs an approximation for the cost of a pool truck route and a structure that allows orders to be fulfilled without the FTL vehicles delivering orders that went through a pool point.

Ghost Vehicles

The most difficult part of designing a model that can obtain an initial solution to the PDOP is optimizing the FTL vehicle routes before optimizing the Pool vehicle routes. This requires a mechanism for satisfying the demands at all delivery locations, despite Pool Vehicles not being a part of the model. Countless variations of model structuring and testing led to a viable solution by allowing unconstrained vehicles to make final deliveries from pool points. With this adaption, another necessary component is a cost approximation for the pool deliveries in the model, that the optimization algorithm can use to make routing decisions.



(a) A Ghost Vehicle freely completes all delivery requirements

(b) Two FTL Vehicles are needed without Pool Point usage

Figure 5.1: Illustration of Delivery Routes possible inside OR Tools model

To handle the delivery of items routed through a pool point, FTL vehicles that visit a pool point shed some constraints on FTL vehicles, changing vehicle type in the process. This is done by tracking a boolean variable g for each FTL vehicle, that determines if the vehicle has visited a pool point. This is the end of

the FTL truck's route, however the truck can continue onward acting as a Ghost Vehicle. Multiplying g's compliment by the FTL vehicle's destination constraint allows the new ghost vehicle to travel between destinations as in equation 5.2.1. Figure 5.1 shows an example of a delivery route started by an FTL vehicle and completed by a ghost vehicle from the pool point onward. The ghost vehicles perform the duties of the pool vehicles to come later, as they are allowed to travel to a first destination for the corresponding pool route cost. After reaching one destination, the trucks can travel between destinations for no additional cost, and without counting against their maximum travel distance. This facilitates necessary deliveries to satisfy the requirements of a viable solution, while estimating the cost of pool routes based off selecting one of the pool routes its contents must travel. The key behavior preserved is the ability of FTL vehicles to pick up orders that are all bound for different destinations if and only if they visit a pool point.

- FTL Trucks cannot drive between Destinations, Ghost Vehicles Can:

$$(1 - g) \sum_{i \in J} \sum_{j \in J} x_{ij}^{v} = 0, \ \forall v \in V \qquad [5.2.1]$$

5.2.2 Post-Processing

To find the true cost of the solution, the cost for all the FTL routes in the OR Tools solution is summed, removing the pool route cost approximations from the optimizer's objective value. After the optimizer has determined which orders to route through which pool points, post-processing on that solution stores information about which orders were on each truck visiting a pool point and what window of time can that truck be at the pool point without violating continuity constraints. Algorithm 1 shows the calculation used to find how many orders need to get from the pool point to each destination, and the minimum number of trucks that must be assigned to service these orders.

Analysis of the first solution provides a list of when trucks arrive at a pool point in chronological order. Then, each order on the truck is added to a queue of orders bound to a specific destination, where the time that the first order in the queue arrived is tracked. Once an order arrives to the queue, such that the sum of orders in the queue do not fit on one truck, all previous orders are placed on a Pool truck to the destination, the most recent order to arrive begins a new queue, and the tracking timer is reset. However, if no item arrives which fills the truck in the

allotted waiting time of 24 hours, the truck leaves with all the items in the queue. Thus, if a queue is empty when a new order arrives, the tracking timer will start when that order arrived. The combined cost for the FTL routes from the OR Tools solution and the calculated pool vehicle costs for each pool point used gives the true objective value for the solution.

5.2.3 Node Structure

The modified problem representation needs to have a node structure which permits all desired truck behaviors. To achieve maximum flexibility, it must be possible for all FTL trucks to visit the same Pool Point. The OR Tools constraint programming optimizer only allows each node to be visited by a single truck [3]. Working around this modeling convention requires a node for every truck at every pool point, each a proxy for the pool point itself, inspired by the implementation in Wolfinger's trans-shipment model with Intermediary points [42]. Since the number of FTL trucks in the model is equal to the order total, the node indices are selected by the pattern shown in figure 5.2. Order pickups and destinations alternate, so each odd number is a pickup and each even number is a destination. Once the node indices surpass twice the number of orders, each order is assigned one pool point node for each pool point, with coordinates at their respective locations. However, the trucks are free to visit whichever pool node is available. The quantity simply allows maximum flexibility for the truck behavior, especially in small order count scenarios. The pool nodes have zero penalty dis-junctions, meaning the optimizer does not need to visit any of them to find a viable solution. The dis-junctions dramatically reduce the computational impact of adding more pool nodes, so optimizing a scenario with x orders and 2 pool points takes a similar compute time as a scenario with the same number of orders and 3 pool points. This remains true despite total pool nodes in the model being equal to total orders times total pool points.

Algorithm 1 Post-Processing Vehicles for one Pool Point

Input: List of trucks' orders arriving at the Pool Point in chronological order, arrival time = m_t
Initialize an empty queue q ∈ Q, for each destination j ∈ J
Initialize tracking timer t_q to `null` for each queue
Set count for each Pool Route, $q_n = 0$
for each Vehicle arriving at the Pool Point **do**
    for each Order m on the vehicle's respective destination queue **do**
        if q = `null` **then**
            $t_q = m_t$
        end if
        if sum(q_w) + m_w > w || sum(q_s) + m_s > s **then**
            $q_n \leftarrow q_n + 1$
            $q \leftarrow \{m\}$
            $q_t = m_t$
        else if Time elapsed - m_t > 24 hours **then**
            $q_n \leftarrow q_n + 1$
            $q \leftarrow \{m\}$
            $q_t = m_t$
        else
            $q \leftarrow q \cup \{m\}$
        end if
    end for
end for
for q ∈ Q **do**
    if q ≠ `null` **then**
        $q_n \leftarrow q_n + 1$
    end if
end for
Net Pool Point Vehicle Cost = $\sum_{q \in Q} q_n * f_{pj}$

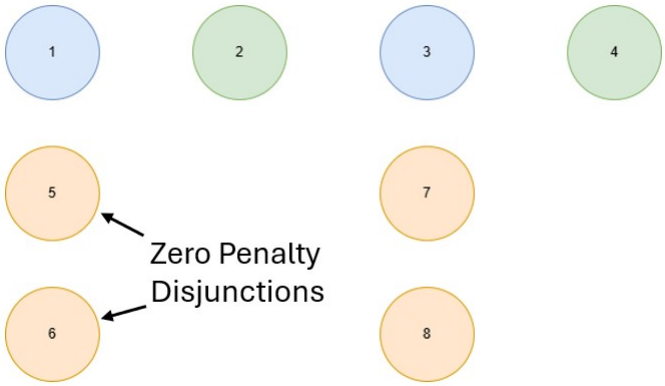


Figure 5.2: Structure for the nodes in an OR Tools optimization model with two orders and two pool points

5.3 Iterative OR Tools

To further improve on the OR Tools model formulation, an iterative version of the optimization model is developed. In this version, destroy operators which work on the route level, remove routes from the solution based on three different operators. The resulting partial solution is then repaired by running the OR Tools optimizer again, inputting the partial solution. This allows the optimizer to escape local optima by expanding the search neighborhood significantly. Highly constrained problems involving intermediary points, such as pool points, benefit from specialized search operators that take the distinction between FTL and Pool routes into account [42]. Such specialized operators help the algorithm search hard to reach areas in the solution space.

5.3.1 Optimization Loop

The iterative OR Tools optimizer runs through a loop for n iterations. With each iteration, an OR Tools model solution is optimized, then the complete solution is formulated through post-processing. Next a destroy operator is applied to the solution, removing between 10 and 20 % of the routes based on the criterion of the operator chosen. This incomplete solution is then given to the OR Tools solver to repair. During the repair phase, because the solution is nearly complete, solution

strategies within the optimizer that cannot compute a scalable solution from a cold start perform effectively. A greater selection of search strategies gives the solver more flexibility as it searches for a better solution. Figure 5.3 shows the iterative solution process, where the solution reconstruction occurs n times.

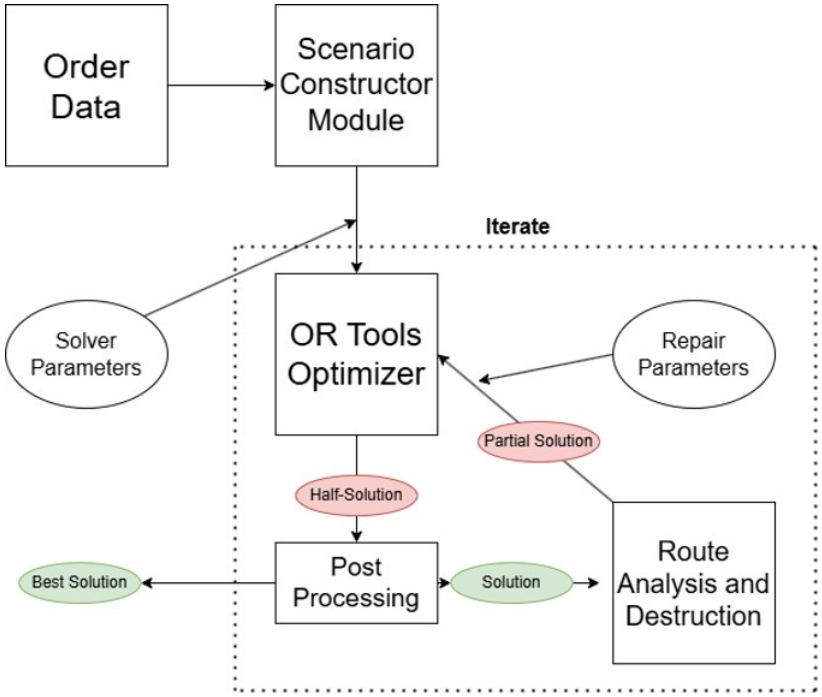


Figure 5.3: Graphic of the Iterative Solution Pipeline

5.4 Gurobi

Given the difficulty creating a formulation to the original optimization problem using OR Tools, Gurobi Mixed Integer Programming solver is also considered. This tool works fundamentally differently from OR Tools, as it relaxes the integer format of the boolean variables, allowing Gurobi to solve a continuous linear program efficiently. This creates a lower bound for the solution's objective value, allowing the algorithm to track the optimality gap between the lower bound and the current solution. It then works towards a feasible solution by searching within decomposed sub-problems to reduce the amount of violated constraints until it

reaches a feasible solution. One notable difference between OR Tools and Gurobi is that OR Tools is open source, while using Gurobi professionally is a very expensive service that includes a suite of PhD level optimization experts on retainer. The cost to use Gurobi is prohibitive, as it would need to have a truly dramatic difference in solution quality to merit such a high cost of use.

5.4.1 Pre-solved Scenarios

The author of this study met with an optimization engineer at Gurobi and discussed the application of their solver for the Pool Distribution Optimization problem. They remarked this was a very difficult problem to solve, and Gurobi would do best if given feasible solutions to improve on, as it struggles to find feasible solutions to improve on in highly constrained problems. To take advantage of initial solutions produced by OR Tools, a pipeline for formatting OR Tools solutions to fit a Gurobi formulation of the problem leverages the best qualities of both tools. In the Gurobi formulation, both FTL and Pool vehicles are optimized simultaneously. The key benefit to this parallel optimization is the true pool costs of the current solution are present during FTL route optimization. The drawback is the node structure of the problem in Gurobi, and the additional constraints needed to manage the Pool Vehicles.

5.4.2 Node Structure and Multi-Echelon Behavior

The node structure for the Gurobi model is defined more strictly, as the demand is not truly dynamic, but the need for Pool Trucks to visit specific pickup nodes is managed through constraints. In the Gurobi model, each order is assigned a specific dropoff node at each pool point, which corresponds to a specific order. Unlike in the OR Tools model, the pool node visited by the FTL truck must be the right one, because each pool dropoff node has a linked pool pickup node as shown in figure 5.4. A truck visiting an order's assigned node at a pool point activates a boolean variable; to activate a constraint stating a pool truck must visit the corresponding pool pickup node.

When a truck carrying multiple orders travels to a pool point, it will visit one node at that pool point for each order it carries. These nodes are all explicitly

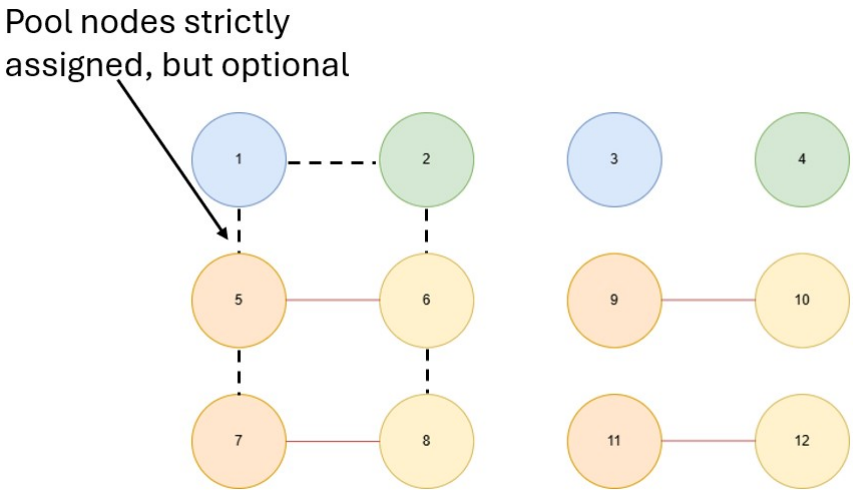


Figure 5.4: Structure for the nodes in a Gurobi optimization model with two orders and two pool points

assigned when forming the optimization model. Then, when a pool truck reaches the pool point to pick up orders, it will visit the pool pickup node assigned to each order that it must pick up. The Gurobi model handles the entire optimization problem as one component, but its difficulty finding initial solutions shows there is still no straightforward way to solve the entire problem at once. Gurobi is much more computationally intensive, particularly in terms of memory, and the number of constraints in the model scales quadratically with the number of orders. By comparison, the number of constraints in the OR Tools formulation scales linearly with additional orders.

Chapter 6

Optimization

This chapter explains the algorithms used to solve models in OR Tools and Gurobi for this study. Each library uses a hybrid algorithm approach to finding solutions. Section 6.1 covers the behavior of initial solution strategies and local search algorithms available inside OR Tools. This sectional also details the external destroy operators used in the iterative OR Tools solution model. Then, section 6.2 defines the steps Gurobi takes to improve on initial solutions inside the two-echelon model. And finally, section 6.3 breaks down the limitations intentionally imposed on this research, which drive the results toward a more practical and scalable solution.

6.1 OR Tools

OR Tools contains numerous algorithms for obtaining an initial solution, and local search to improve an existing solution. These two categories of optimization algorithm work together inside the optimizer to reach a feasible starting point, then make incremental improvements. The strategies and algorithms are derived from graph theory, where trees are spanned by connecting nodes to minimize cost, forming the basis of these optimization techniques.

6.1.1 First Solution Strategies

A **First Solution Strategy** is an algorithm designed to find an initial solution to a constraint programming optimization problem [28]. These algorithms focus on the nodes that need to be visited, prioritizing a feasible solution over an optimal

one. Notably, only one OR Tools first solution strategy reliably finds a feasible solution to an unsolved Pool Distribution Optimization problem. However, in a partial solution repair run, three total first solution strategies reliably produce solutions. Each of the three solution strategies is explained in the following subsections.

Parallel Cheapest Insertion (PCI)

The most robust first solution strategy available for the Pool Distribution Optimization Problem is Parallel Cheapest Insertion. It's also the only strategy available in the OR Tools optimizer able to consistently produce feasible solutions for optimization problems containing 50 or more orders. Parallel Cheapest Insertion builds multiple routes simultaneously, inserting the cheapest node at its cheapest position for each arc respectively. The number of concurrent arcs forming depends on the problem size, and the thread capacity of the hardware used [28]. Although parallel insertion algorithms can use heuristics like $\sqrt{n}$ to decompose the problem into a number of sub-trees based on number of graph vertices [39], the OR Tools algorithm views the entire system when forming arcs in parallel.

The cheapest insertion operation works by examining each in-sequence node pair (i,j) from a route, and checking the cost to insert an unvisited node k, until the lowest cost of insertion is found. Equation 6.1.1 shows the calculation for cheapest insertion, computed for each arc concurrently.

$$\min_{i,j,k} [c(i,k) + c(k,j) - c(i,j)] \qquad [6.1.1]$$

Out of the remaining unvisited nodes the cheapest node to insert, in its cheapest position, is added to the route. This process is only complete when every node without a disjunction is visited on a route.

Local Cheapest Insertion (LCI)

Differing in tree search order, the Local Cheapest Insertion algorithm takes node index into account when making insertion selections. This means that lower node indices will be checked earlier, making the algorithm more faster, but potentially less optimal than Parallel Cheapest Insertion. Node ordering in the optimization

scenarios places pool points after order pickups and dropoffs, so the search tendencies of this algorithm appear to bias towards skipping pool points. Nevertheless, this insertion algorithm is not able to consistently produce feasible solutions to large problems. The value of this algorithm comes when repairing partial solutions within the Iterative Run OR Tools Optimizer. This insertion method uses the same minimization equation, except that the index of k is selected in sequential order [28].

Global Cheapest Arc (GCA)

The Global Cheapest Arc strategy works by finding the cheapest connections between nodes, iteratively adding connections to the model until a feasible solution is found [28]. This strategy is not robust enough to find initial solutions for pool optimization scenarios of 10 orders or more, but is useful when repairing partial solutions. Unlike insertion strategies, global cheapest arc may produce route segments that cannot be connected while still satisfying the constraints in this problem formulation. It often builds segments cost-effective in isolation, but that cannot be easily integrated into a feasible overall solution.

6.1.2 Search Metaheuristics

OR Tools needs a Local Search Algorithm, also called a **Search Metaheuristic**, to incrementally improve the current solution during optimization. The metaheuristic takes control once an initial feasible solution is found using a first solution strategy. Each metaheuristic takes a different approach to creating a local search neighborhood and selecting which operations to use when altering a solution. OR Tools has a total of 13 local search operators [14], listed in Table 6.1. The optimizer views a feasible solution which can be created through one of these operations as a candidate solution. A candidate solution has to exist somewhere in the problem's feasible solution space, meaning that all constraints in the system are satisfied. The set of possible candidate solutions comprises the current solution's local search neighborhood [18]. The search metaheuristic navigates through the solution space, by iteratively selecting a candidate solution and forming a new search neighborhood around it. The five search metaheuristics available in OR Tools are explained in the following subsections.

Table 6.1: Local Search Operators in OR Tools

Operation	Description
Two-Opt	Reverses a segment of a route between two nodes.
Three-Opt	Removes three edges and reconnects them in a different way.
Relocate	Moves a node from one position to another in any route.
Exchange	Swaps two nodes between routes or within the same route.
Cross-Exchange	Exchanges sub-paths between two routes.
Or-Opt	Moves a chain of two or three nodes to another position in any route.
Make-Active	Activates a previously inactive node by adding it to a route.
Make-Inactive	Deactivates a node by removing it from a route.
Make-Unperformed	Removes a node from the solution entirely.
Reverse Sequence	Reverses the order of nodes in a route segment.
Chain Exchange	Exchanges chains of nodes between two routes.
TSP-Opt	Optimizes the sequence of nodes in a single route.

Greedy Descent (GD)

Greedy Descent is a deterministic algorithm that searches for a route operation to reduce the solution's objective value by the greatest amount. Its behavior is not based on exploration, and can only take the best available immediate next option [25]. This could have a problematic relationship with such a highly constrained problem, as greedy descent is prone to being stuck at local optima. Greedy Descent also uses only a subset of potential route operators, as considering all 13 would require too much calculation at each step.

Guided Local Search (GLS)

Guided Local Search builds on the greedy behavior of traditional local search by modifying the problem's objective function to explore different candidate solutions. This modification occurs when the optimizer has reached a local optimum. From this position, simple greedy behavior would prevent the search algorithm from moving any further. At local optima, guided local search ascribes features to the current solution, and initializes penalties for inclusion of the features at zero. In this problem these features come from specific arcs in the model, which represent a vehicle driving between two points [41]. Equation 6.1.2 shows how the algorithm assigns penalty utility U to features in the local optimum, where the features with the highest utility score are chosen to be penalized, increasing the

value of P_i. $I_i(x)$ is an indicator function for the feature being penalized and $c_i(x)$ is the arc cost, so expensive routes are seen as having a higher utility to penalize. If the inclusion of the derived features in the solution fails to improve the objective value, the penalty for including those features will increase. Over time, the penalties for features that do not improve the solution cause the algorithm to move away from the current local optimum [41]. The augmented cost function used to select new solutions is shown in equation 6.1.3. The modified cost function $g(x)$ is used to navigate the solution space, and over time $g(x)$ diverges from $f(x)$ enough to escape the local optimum. Coefficients for λ and a are selected by the OR Tools routing model. Generally, λ moderates the diversity of the search space, while a balances the size of the penalty function relative to previous improvements and the total number of features penalized [40].

$$U(x,i) \;=\; I_i(x)\,\frac{c_i(x)}{1+p_i} \qquad [6.1.2]$$

$$g(x) \;=\; f(x) + a\lambda \sum_{1\le i\le m} p_i\, I_i(x) \qquad [6.1.3]$$

Simulated Annealing (SA)

Simulated Annealing has an entropy approach to escaping local optima, which focuses on exploration in the early stage of optimization. This approach works by following a cooling schedule, similar the way some alloys are produced to remove defects by controlling the speed they cool to normal temperature after smelting. In this analogy, temperature $\in$ [0,1] and equates to the likelihood of the routing model selecting the next solution by accepting a random change, even if it increases the solution cost. As the search algorithm explores, iterating through new solutions, the temperature decreases. This makes the routing model less likely to accept a new solution if it does not improve cost minimization, progressively adopting more greedy behavior. A simulated annealing search algorithm puts higher priority on finding the best region of the solution space over finding the very best local optimum [20]. Prioritizing exploration seems a promising approach for Pool Distribution Optimization, as local search neighborhoods are quite small due to all the constraints in the problem.

Tabu Search (TS)

Another metaheuristic that modifies the problem to enhance exploration is Tabu Search. This algorithm works by encouraging exploration when a local optimum or a plateau is detected in the solution space. For example, when the OR Tools routing optimizer reaches a local optimum, it will detect that no improvement is possible from the local search neighborhood, and instead will accept a change that increases the solution cost. Tabu Search keeps three levels of memory to guide its exploration of the solution space. Short Term Memory is used to track which solutions have been visited recently, to avoid redundant exploration. Short term memory relates to modified objective values for candidate solutions $f(x')$, where candidate value becomes $f'(x') = f(x') + T(x')$, accounting for penalty $T(x')$ if x' is in the Tabu List. Intermediate Term memory biases the search towards promising areas in the solution space by tracking the arcs which appear frequently in good solutions. If $f(x')$ achieves a new best solution, x' is removed from the Tabu List to achieve this bias. Finally, Long Term memory drives the search into unexplored regions when the search gets stuck in a plateau [10]. Long Term memory assigns penalties to solutions that are in areas where significant exploration has occurred by updating the Tabu List $T(x)$, preventing the search from over-exploiting one region [11].

Automatic

The Automatic search metaheuristic leverages a combination of the other search metaheuristics, making it a robust choice as it can adapt when one search strategy is ineffective. However, because this selection process is adaptive, and the optimizer needs time to test and score the other metaheuristics before finding the best one for the problem. Given a long period of time, automatic is expected to find the most effective search metaheuristic, but scoring and selection can slow down optimization if solve time is limited [29].

6.1.3 Iterative Model

The Pool Distribution Optimization problem is highly constrained, trapping search algorithms in local optima, potentially far from the global optimum. In finding ways to explore larger regions in the solution space, external components to the

optimization process lead to further development for the OR Tools optimizer. An iterative run version of the OR Tools Pool Distribution Optimizer allows analysis of a feasible solution to guide major changes to the solution though an iterative optimization process. A key benefit to this approach is the ability to change the solution significantly in a single step, preventing the algorithm from being trapped in a subspace within the problem's feasible solution space. The use of destroy operators to remove whole routes results in a Large Neighborhood Search.

Destroy Operators

Removing parts of the existing solution, then repairing it helps to push existing solutions outside their current search neighborhood. The route modification operators inside OR Tools have limitations in terms how large of a change they can make in a single step. The capacity for these operators to change the current solution defines the local search space [35], which is small in this problem, leading to the local optima issue. Defining operators that can make more significant changes opens up the search space to better explore possible solutions. Destroy operators are used to remove routes from the solution after post-processing. One destroy operator randomly selects routes to remove from the solution, the other, novel destroy operators are shown in equations 6.1.4 and 6.1.5.

• Worst Trip Removal takes the higher peak of weight or size utilization for each vehicle, and chooses the lowest among the vehicles:

$$\min_{v \in V} \left[\max \left(\sum_{v \in V} \sum_{m \in M | v_m = v} \frac{w_m}{w_v} , \sum_{v \in V} \sum_{m \in M | v_m = v} \frac{s_m}{w_v} \right) \right] \qquad [6.1.4]$$

This operator works to push out inefficient routes, with the possibility that orders on the destroyed route can be added to existing routes, increasing efficiency and ideally decreasing the number of active trucks. Capacity utilization is a metric that some Shipwell clients value in addition to the absolute cost for all routes, so removing low utilization routes can improve the solution beyond objective value.

• Worst FTL Trip Removal takes the higher peak of weight or size utilization for each vehicle not visiting a pool point, and chooses the lowest among the vehicles:

$$\min_{\substack{v \in V \\ \sum_{i \in O} \sum_{j \in P} x_{ij}^{v} = 0}} \left[\max \left(\sum_{v \in V} \sum_{m \in M | v_m = v} \frac{w_m}{w_v} , \sum_{v \in V} \sum_{m \in M | v_m = v} \frac{s_m}{w_v} \right) \right] \quad [6.1.5]$$

This operator has similar benefits to Worst Trip removal. However, Worst FTL Trip Removal actively reduces the number of trucks bypassing the pool point, tending the solution towards routing orders through that pool point. More trucks arriving at a pool point gives the pool trucks more opportunity to increase utilization given the time window to fill one pool truck. When the resulting repaired solution lowers the net solution cost compared to the previous iteration, the new solution is saved to move forward. If the new solution is not an improvement, the previous solution is retained.

Adaptive Selection

Two of the three destroy operators select routes to destroy deterministically. This may cause a loop where the repaired solution is worse than the previous, so the previous solution is reused next iteration, and the same routes are subsequently removed. To prevent this, a combination of all three operators is tested. An **Upper Confidence Bound** selection algorithm is employed to adapt selection based on past performance. The UCB algorithm computes the upper bound of the confidence interval on mean reward for each operator. Selections are scored based on fractional improvement of the solution. With each selection and iteration, the value for each UCB changes, shifting the balance of UCB scores and leading to a dynamic selection process. The UCB for operator o is computed in equation 6.1.6, where i is the solution iteration, n is the number of times the operator has been selected, c is the solution cost, and t is the exploration temperature. At each iteration, improvement for the current operator o is calculated as: $\Delta_o = \frac{c^{i-1} - c^i}{c^{i-1}}$, which is used to update the average performance of that operator.

$$UCB = \frac{\sum \Delta_o}{n} + t * \sqrt{\frac{\log(i)}{n}} \quad [6.1.6]$$

The Iterative OR Tools optimizer resulting from this selection algorithm further reduces solution cost after the first run of the OR Tools model. Figure 11.10 in the Appendix shows an example of a scenarios where this iterative optimizer worked very well to reduce cost. Figures 11.11 and 11.12 depict solutions where all

orders were either required to travel by FTL only, or where all orders must route through a pool point. These two extremes produce similar objective value in the solution. However, Figure 11.13 shows running the single run OR Tools optimizer produced a solution with roughly 10% reduction in cost. Then, Figure 11.14 shows that the Adaptive LNS is able to find an additional 5% savings making for a total 15% savings on the solution. This is the best case scenario that, which through robust design may be repeated in real world situations.

6.2 Gurobi

Although Gurobi works with a similar model formulation to OR Tools, its problem solving approach is fundamentally different, focusing on computing models into sub-problems within the scenario over exploration via route operations. Although Gurobi natively implements first solution heuristics, they are slower for this problem and less robust than Parallel Cheapest Insertion used by OR Tools. Gurobi's overall approach tries to solve the model, focusing on breaking the model into pieces and determining components of the solution with certainty, rather than trying to navigate the solution space through route operations.

6.2.1 Presolved Scenarios

To speed up optimization and get better incumbent solution starting points, solutions are generated in OR Tools with a single run of the routing optimizer. Using a Mixed Integer Programming (MIP) start produced from a faster model helps Gurobi avoid exploring unpromising branches. Each solution is then transformed into a boolean tensor that represents a MIP start for Gurobi, using a module developed to equate solutions between the two models. The key component is a formula to compute which pool pickup nodes need to be visited by the pool vehicles, as in the last chapter we saw that the Gurobi model strictly assigns pool nodes to specific orders. Calculating the corresponding pool point pickup or dropoff node for a specific order follows equation 6.2.1, where n is the total orders, o is the order number, p is the total pool points, and p_i is the index for the pool point visited.

$$ind = (2n) + (2po) + (2p_i) + 1 \qquad [6.2.1]$$

Equating the nodes in the two models yields a set of routes that also satisfy the constraints in Gurobi. They simply need to be equated to a $[V, N, N]$ boolean tensor, where each arc represents the connection between two nodes for a specific vehicle. This will become the incumbent solution within the model, which is the current best feasible solution. Feeding the tensor solution as a start into Gurobi skips the initial solution finding phase, moving directly into a Branch-and-Bound algorithm.

6.2.2 Branch-and-Bound

The Gurobi solver looks to break the problem down into smaller components by fixing the value of some arcs in the model. This is called Branching, and it simplifies the problem, allowing the solver to look for the best solution within more manageable search spaces [36]. Branches that do not have promising solutions are pruned, further simplifying the model. After the branches are formed, Gurobi begins Bounding, where it relaxes the integer format of the solution to find the lower bound on the solution to the sub-problem. By allowing variables to take fractional values, Gurobi can compute an optimal solution for the relaxed, continuous version of the problem. The gap between the lower bound through this relaxation and the incumbent solution found constitutes the Optimality Gap. This gap is the percentage difference between the current best solution and the lower bound determined by an integer relaxation of the problem [24]. If a node's lower bound is higher than the current upper bound on the problem, it is pruned entirely to prevent wasted exploration. During the solving of the problem relaxation, additional constraints called Cutting Planes help remove fractional values from solutions without removing feasible integer solutions, which accelerates convergence. Figure 6.1 shows a diagram of a branch and bound algorithm's progression. It can be seen to identify unfeasible branches of the solution space, prune them, and move to search other feasible branches. This algorithm is a rigorous solution approach, in that it tries to search every branch of the tree that cannot be pruned. This ensures that no feasible solution is overlooked, but may also lead to daunting computational demands in large or complex instances.

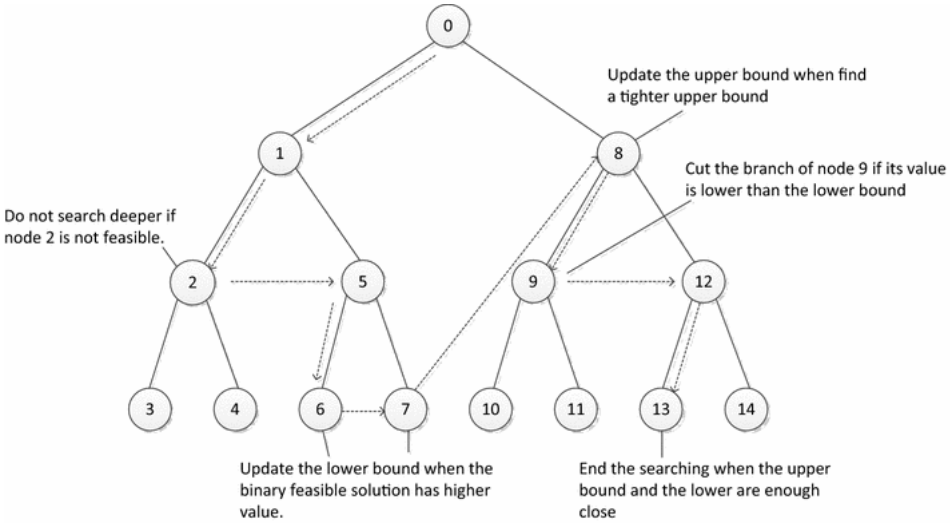


Figure 6.1: A diagram of the Branch and Bound algorithm [27]

6.2.3 MIP Heuristics

As Gurobi searches to improve solutions, it uses two MIP specific strategies to search the solution space. The first is a Feasibility Pump, where fractional values in the problem relaxation are rounded to make a feasible solution. It works iteratively, rounding values and then modifying the solution to satisfy any constraints that were violated due to rounding [8]. The feasibility pump takes advantage of the problem relaxation to find integer values which clearly improve the solution. The other heuristic Gurobi employs is a Relaxation Induced Neighborhood Search, which works by comparing the incumbent solution to the linear relaxation solution. It fixes values on the arcs where the two solutions are almost identical, establishing where the incumbent solution and the relaxation solution suggest approximately the same value [4]. In this problem, those values always correspond to either 1 or 0. Gurobi continues to apply heuristics, improving the incumbent solution until either the time limit is reached or the optimality gap drops below a threshold chosen based on the model.

6.3 Self-Imposed Limitations

The optimization algorithms in development need to work efficiently so Shipwell can optimize the orders of their clients without significantly impacting the company's cloud systems. Limitations on both time and and compute power are imposed to put emphasis on scalable solutions. Customers expect to get an optimized route set shortly after inputting their order data, so an upper limit of 15 minutes is imposed on solve time. To simulate a low-cost compute cluster, a memory limit of 32GB and 3.0GhZ processor with 24 cores are used to compute solutions. These limitations ensure that a solution developed in a research setting is not outside the practical capabilities of the Transport Management System.

Chapter 7

Experimental Setup

To determine the best routing optimization conventions for solving the PDOP in Shipwell's use case, two series of testing serve to refine the parameters and implementation for the models in this study. Section 7.1 catalogs the first main round of optimization testing, in which solver parameters for repairing solutions in the iterative OR Tools optimizer are tested. Then, section 7.2 details the final, comprehensive round of testing for all the optimization tools and benchmarks developed in this study across a variety of scenarios. The results of this testing will dictate the final implementation decisions for the Pool Distribution Optimizer.

7.1 OR Tools Iterative Run Parameter Testing

The iterative optimizer has the capacity to use multiple first solution strategies and search metaheuristics when repairing partial solutions. Each optimizer parameter is tested to determine the potential for use in the repair process. Furthermore, the three destroy operators are tested separately to compare performance. Each destroy operator must improve solutions reliably to be included in the final model. The testing protocol for three levels of optimizer parameters: First Solution Strategy, Search Metaheuristic, and Destroy Operator freezes two parameters of the model, while comparing all viable types within the chosen category. For example, when testing destroy operators, the first solution strategy is set to parallel cheapest insertion, and search metaheuristic is set to automatic, which are the most robust settings respectively.

Testing involves optimizing ten different synthetically generated scenarios of fifty orders, two pool points, and 5 distribution centers. Each scenario is optimized ten separate times to examine variance within the solver with the selected parameters. In preliminary testing, three first solution strategies consistently reconstructed a feasible solution when repairing partial solutions. Any strategies that failed to consistently produce results were eliminated as the optimizer needs robust performance. Parallel Cheapest Insertion, Local Cheapest Insertion, and Global Cheapest Arc are compared in a variety of scenarios to determine which is best. Table 7.1 shows the schematic for these tests. Each iterative run of the solver starts with an initial solution, produced using the same robust settings as the Single OR Tools Optimizer Run benchmark; parallel cheapest insertion and automatic search metaheuristic. Initial solutions are generated with 90 seconds of solve time. The optimizer then iterates twenty times, destroying and repairing parts of the solution. Each solution repair is given 30 seconds to run, making the full computation time roughly 12 minutes. The results of these tests inform the settings for testing the **Upper Confidence Bound Selection Large Neighborhood Search (UCB LNS)** Algorithm in the following section. However, detailed results from these tests are included in the results section of the report.

Table 7.1: Configurations of First Solution Strategy, Search Metaheuristic, and Destroy Operator in Each Test Group

Parameter Tested	First Solution Strategy	Search Metaheuristic	Destroy Operator
Parallel Cheapest Insertion	Parallel Cheapest Insertion	Automatic	Random Removal
Local Cheapest Insertion	Local Cheapest Insertion	Automatic	Random Removal
Global Cheapest Arc	Global Cheapest Arc	Automatic	Random Removal
Greedy Descent	Parallel Cheapest Insertion	Greedy Descent	Random Removal
Guided Local Search	Parallel Cheapest Insertion	Guided Local Search	Random Removal
Tabu Search	Parallel Cheapest Insertion	Tabu Search	Random Removal
Simulated Annealing	Parallel Cheapest Insertion	Simulated Annealing	Random Removal
Automatic	Parallel Cheapest Insertion	Automatic	Random Removal
Random Removal	Parallel Cheapest Insertion	Automatic	Random Removal
Worst Trip Removal	Parallel Cheapest Insertion	Automatic	Worst Trip Removal
Worst FTL Trip Removal	Parallel Cheapest Insertion	Automatic	Worst FTL Trip Removal

7.2 Upper Confidence Bound Selection Large Neighborhood Search Algorithm Testing

The second main round of optimization testing compares the Upper Confidence Bound selection Large Neighborhood Search algorithm against other benchmark optimization tools using Whole Foods Market and synthetic data for scenarios of various sizes. The main goal is to show the UCB LNS algorithm is an effective optimization method. However, revealing patters in the ability of different models across the variety of scenarios could emerge from the results, driving further insight into better implementation of the final optimization tool.

7.2.1 Benchmarks

The UCB LNS algorithm has three benchmark algorithms to compete with across a variety of scenarios. The FTL Only and OR Tools Single Run algorithms will give an idea of the improvement from modeling with pool points, and from using an iterative version of the optimization model. The third benchmark, Gurobi, only works in small scenarios but is highly effective. It acts as a solution cost lower bound for small scenarios, indicating approximately the lowest cost solution possible in them for comparison and therefore the remaining potential efficiency.

FTL Only Solutions

Shipwell uses a routing optimizer for clients who do not operate pool points, it runs using OR Tools. This PDP is much easier to solve and OR Tools is well suited to the task without additional help. A similar optimizer to the version used by Shipwell is relatively simple to implement, with solutions generated by this optimizer making an excellent benchmark for the solution costs that come from pool solutions. The difference between the two represents the savings made by using pool points. Furthermore, any pool solution with a higher cost than the FTL only solution produced by the benchmark is a good indicator that the optimizer is not working properly or the scenario is not suited for leveraging pool points at all.

To set up the FTL Only Solution, information for the pickup and delivery nodes are passed to the optimizer, but all pool point related nodes and data are left out.

This way, the same scenarios can be used with the optimizer, by slicing the needed sections from appropriate lists and matrices of scenario data. This model is fast, and very robust, so it is used as a benchmark for every scenario type.

OR Tools Single Run Solutions

A step up in complexity from the FTL Only solution is an OR Tools Single Run solution, these solutions are closer to the results produced by the more advanced Gurobi and Iterative Run OR Tools solutions. This benchmark serves as a means to test the two advanced optimization methods' ability to improve on a feasible solution with pooling. Although less robust than the FTL only optimizer, this benchmark is also used for every test scenario type.

Gurobi Optimized Solutions

The solutions from Gurobi, even with pre-solved route inputs, are much slower to produce than OR Tools. This factor alone makes it a poor candidate for Shipwell's Pool Distribution Optimizer, which needs to produce solutions within a few minutes. Furthermore, the Gurobi solver does not scale well compared to OR Tools, so testing with every scenario type is not feasible. For example, a scenario with just 30 orders and one pool point crashes a compute cluster with 32GB RAM, running out of memory. Even smaller scenarios such as 20 orders and one pool point take 30 minutes to optimize. These issues relocate the Gurobi optimizer to use as a benchmark, rather than a final candidate.

The Gurobi model acts as an lower bound for solution cost in small scenarios, because it searches more exhaustively than OR Tools. Gurobi solutions cannot be verified as optimal because this is an NP-Hard problem, but scenarios small enough to optimize with Gurobi help set a target for optimizer performance.

7.2.2 Testing Schematic

A variety of scenarios test the quality of the Iterative Run OR Tools Optimizer with UCB LNS. Each scenario is solved three times and the scores are averaged. Solutions are compared based on their average score for a scenario type, and their percentage improvement over the three benchmark optimizers' results. In

the synthetic data, each scenario parameter set is used to generate ten unique optimization problems. In the real data case, 95.37% of the orders provided by Whole Foods Market were optimized after being partitioned into scenarios.

Synthetic Scenarios

Synthetic data is critical to developing generality for optimization algorithms solving the PDOP, as the data from Whole Foods Market only covers a specific use case. Synthetic data also serves to test the optimizer in small scenarios, where a more thorough algorithm like Gurobi can operate and provide a lower bound on solutions from other algorithms. Smaller scenarios in the range of 10-20 orders are solvable using Gurobi, the best performing, but least scalable algorithm used in the final testing phase. Due to strict stopping criterion and a thorough solution approach, Gurobi either finds a very good solution or fails to improve its incumbent solution at all, depending on computational resources relative to the problem size. This means Gurobi is a great source for a rough cost lower bound, when it can find a solution. Since the PDOP is deep in the NP-Hard category of problems, having a rough lower bound is a valuable comparison tool to quantify room for improvement within a scenario. All synthetic scenario sizes and the solver parameters used with them are listed in table 7.2.

Table 7.2: Testing Schematic for O orders, P pool points, and D unique destinations. Solver time reported in seconds.

(O, P, D)	Gurobi Pre-Solved		OR Tools FTL Only		OR Tools Single Run		UCB LNS OR Tools	
	Tested	Time (s)	Tested	Time (s)	Tested	Time (s)	Tested	Time (s)
(10, 1, 3)	✓	1200	✓	90	✓	90	✓	360
(10, 2, 3)	✓	2400	✓	90	✓	90	✓	360
(20, 1, 5)	✓	1800	✓	90	✓	90	✓	360
(30, 2, 5)			✓	90	✓	90	✓	360
(50, 2, 8)			✓	150	✓	150	✓	520
(100, 1, 10)			✓	150	✓	150	✓	780
(100, 2, 10)			✓	150	✓	150	✓	780
(100, 3, 10)			✓	150	✓	150	✓	780

Examples of these scenarios are seen in figure 7.1 and 7.2, where each cluster of orders is roughly centered at a city and so are the destinations. Order clusters and by extension, pool point service regions, can overlap; giving some orders multiple pool point options.

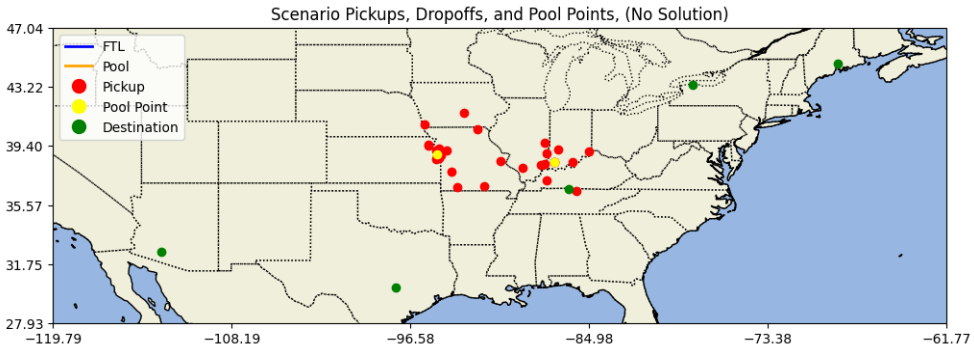


Figure 7.1: Node Coordinates from synthetic scenario with 30 Orders, 2 Pool Points, and 5 Destinations

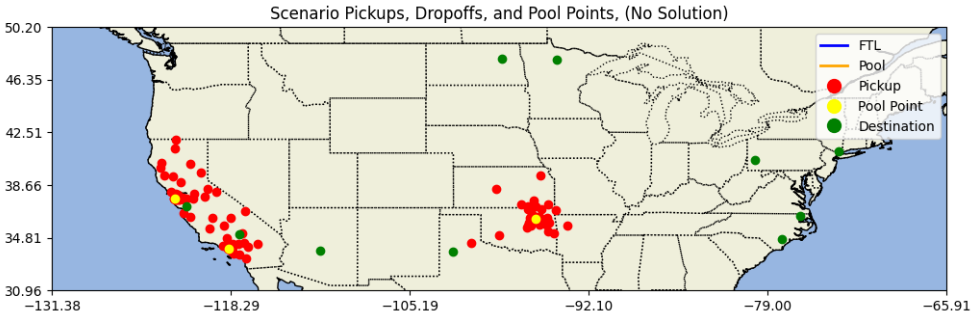


Figure 7.2: Node Coordinates from synthetic scenario with 100 Orders, 3 Pool Points, and 10 Destinations

Whole Foods Market Scenarios

Testing scenarios using Whole Foods Market data are designed to mimic the real use case for the company. Only the largest category of scenario the optimizers can reliably handle is used, where 100 to 150 orders are optimized simultaneously. Table 7.3 provides the number of orders after clustering for each pool point or pool point pair. The lesser of the optimizer's capacity and the total number of orders determines the actual number of orders used in a single scenario. On a total of 7 out of 25 days, some orders were left out to reach the capacity limit of

150 orders. This was done for consistency in comparison, but in practice, those scenarios over 150 orders would be split into two groups to fit into the optimizer. In total, 95.37% of the orders for one week are used in the test schematic, so the net cost savings will provide an idea of the cost savings per week Whole Foods Market stands to gain from using the best pool point routing model and optimizer.

Table 7.3: Number of orders optimized for each Whole Foods Market Pool Point and Date is the minimum between the Order Count and Capacity.

Pool Point/s: (Capacity)	**Feb. 15**	**Feb. 12**	**Feb. 9**	**Feb. 8**	**Feb. 7**
Pool Point 4: (100)	61	82	66	63	31
Pool Point 5: (100)	82	76	79	88	54
Pool Point 2: (150)	158	166	105	145	69
Pool Point 6: (150)	150	196	156	172	124
Pool Points 1,3: (150)	144	175	139	156	50

Table 7.4: Testing Schematic for 100 and 150 Order Scenarios. Solver time is reported in seconds.

Scenario	OR Tools FTL Only	OR Tools Single Run	OR Tools Iterative Run
(100)	180	180	780
(150)	240	240	1440

Figure 7.3 shows the origins for a single day's orders nearest to pool point 6. A single day's worth of orders near this pool point after clustering 150, which the routing optimizer is able to compute in a single instance. The destinations are located both across the country and within a day's drive.

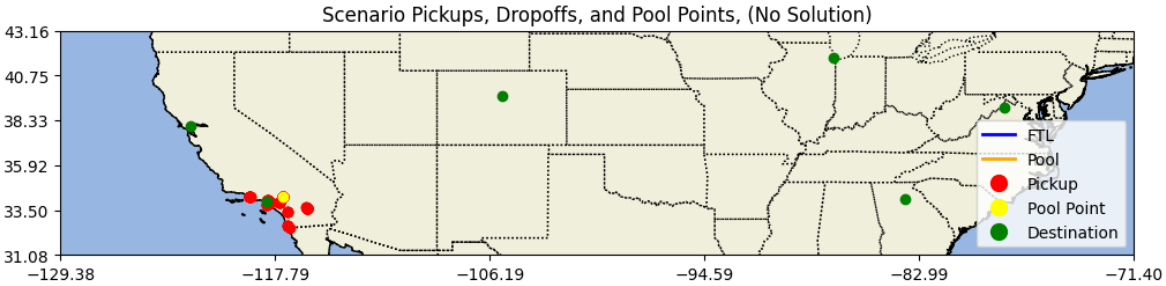


Figure 7.3: Order pickups clustered around a WFM Pool Point, along with their destinations.

Chapter 8

Results and Discussion

The testing results for optimizer parameters, and optimization algorithms, are covered in this chapter. In section 8.1, results from the Iterative OR Tools Optimizer solver parameter testing are discussed, focusing on evidence to clearly show one or multiple parameters are optimal for a general use case. Then, section 8.2 dives into the results from the final round of testing, where complete optimization tools are compared directly, resulting in an unexpected discovery about the best optimization implementation for Shipwell. This section also discusses important patters seen in the testing solutions and insights on the PDOP that come from a qualitative assessment of the results.

8.1 Optimizer Parameters

In testing the Iterative Run OR Tools optimization model, scenarios are reconstructed using the same strategy, metaheuristic, destroy operator for each iteration. This test gives insight into the long term performance of each parameter, to better understand which configuration gives the model the best chance to find a high quality solution. Testing the routing model in this way serves to eliminate approaches that fail to explore the feasible solution space, with the goal of boiling the configuration options down to the most useful parts.

8.1.1 First Solution Strategies

Three First Solution Strategies showed robust performance in partial solution reconstruction. The results of testing Parallel Cheapest Insertion (PCI), Local Cheapest Insertion (LCI), and Global Cheapest Arc (GCA) are stated in Table 8.1, showing Local Cheapest Insertion as the top performer. When examining the confidence interval on overall cost, Figure 8.1 shows that at the 95% confidence level, Local Cheapest Insertion significantly outperforms Parallel Cheapest Insertion. However, when comparing percent improvement on the initial solution, Local Cheapest Insertion significantly outperforms all other first solution strategies. The evidence points to Local Cheapest Insertion as the best first solution strategy when repairing partial solutions. Interestingly, Global Cheapest Arc scored significantly lower on cost improvement then the others, despite having a lower average cost than Parallel Cheapest Insertion. This suggests the initial solutions fed into the Global Cheapest Arc test were generally higher cost solutions, despite using the same scenarios and the initial solutions being generated with the same optimizer settings in all test cases.

Table 8.1: Average Cost and Solution Improvement for Solution Strategies

Solution Strategy	**Average Cost**	**Average Solution Improvement (%)**
Parallel Cheapest Insertion	72522	7.96
Local Cheapest Insertion	65456	11.87
Global Cheapest Arc	67049	4.35

To get a sense for the effect the iterative reconstruction has on existing solutions, figure [**fig:kde˙solution**] shows a kernel density estimate for the distribution of improvements on the initial solution in each optimization run. Global Cheapest Insertion fails to improve the initial solution in 35% of instances, while Local Cheapest Insertion appears to have a higher ceiling on solution improvement over the other first solution strategies. Some spikes in the distributions are likely due to solving each scenario tens time total, leading to a similar optimization path occurring multiple times. With Local Cheapest Insertion the clear top performer, testing for the UCB Algorithm version of the Iterative OR Tools Optimizer will use this First Solution Strategy as the default when repairing partial solutions.

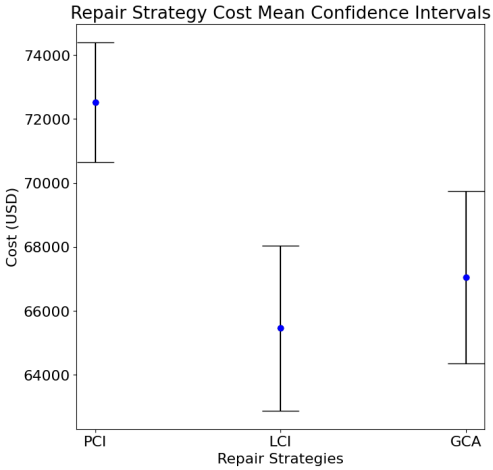


Figure 8.1: 95% Confidence Interval on the Mean Cost for each First Solution Strategy

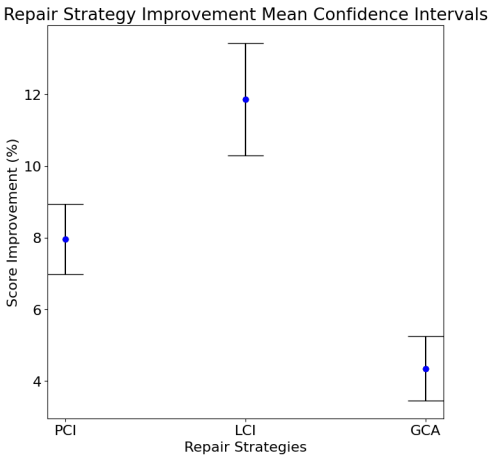


Figure 8.2: 95% Confidence Interval on the Mean Improvement on Initial Solution for each First Solution Strategy

8.1.2 Search Metaheuristics

As seen in Table 8.2, the results from testing Search Metaheuristics are less conclusive than the testing from first solution strategies. Considering the solution strategy testing results indicate solutions have significant variance, even within specific scenario parameters, these results suggest search metaheuristic during the solution reconstruction process has little impact on the optimizer's ultimate solution. Figure 8.4 shows that all five 95% confidence intervals on mean cost overlap. Furthermore, figure 8.5 shows that four of the metaheuristics overlap in improvement confidence interval, with Simulated Annealing performing significantly worse than the other four.

Table 8.2: Average Cost and Solution Improvement for Metaheuristics

Metaheuristic	**Average Cost**	**Average Solution Improvement (%)**
Automatic	67683	5.90
Greedy Descent	67386	6.26
Guided Local Search	67851	5.17
Simulated Annealing	69904	2.51
Tabu Search	68264	6.60

Figure 8.6 displays the improvement distribution for each Metaheuristic, where

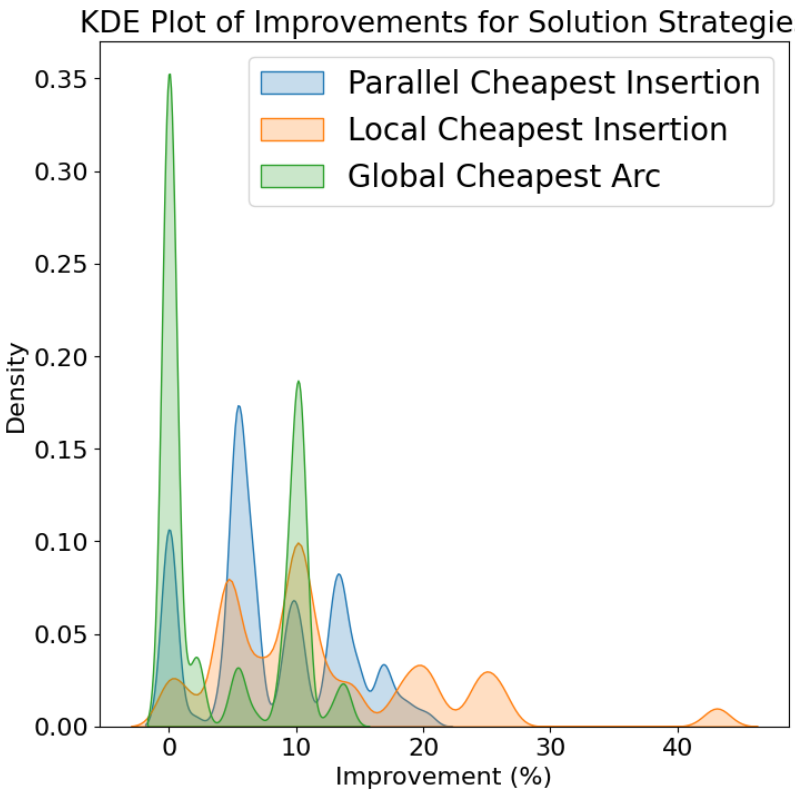


Figure 8.3: Estimated Distributions on Percent Improvement of Solutions for each First Solution Strategy

Simulated Annealing failed to improve the initial solution for half of instances. Automatic shows the highest ceiling, with one cost improvement even reaching 35%. Automatic is an interesting metaheuristic because it is free to move between different the other search metaheuristics, which sounds like a generally better performing approach. However, with only a few seconds to repair a solution within each iteration for the optimizer, the search selection process might slow down convergence in the little time available. Based on the results of this experiment, that concern proved to be unnecessary. With little indication of one Search Metaheuristic differing significantly in performance; the generalized nature of the Automatic setting is theoretically appealing, while the higher ceiling on improvement in testing adds a compelling argument for its use as the default setting when repairing partial solutions during the UCB Algorithm version of the Iterative OR Tools Optimizer tests.

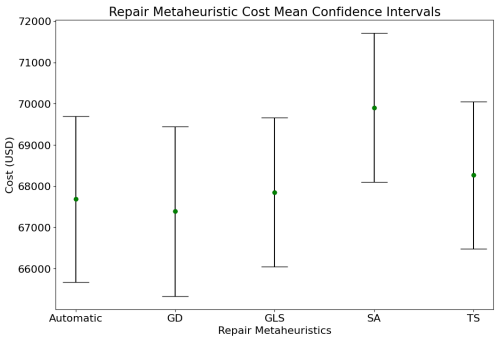


Figure 8.4: 95% Confidence Interval on the Mean Cost for each Search Metaheuristic

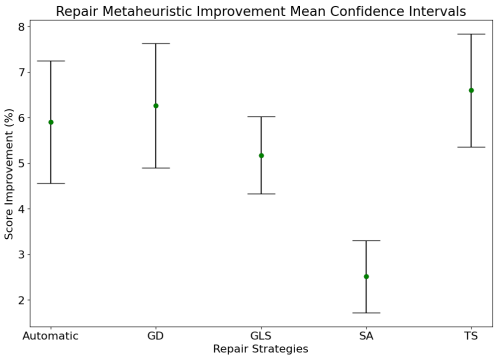


Figure 8.5: 95% Confidence Interval on the Mean Improvement on Initial Solution for each Search Metaheuristic

8.1.3 Destroy Operators

The Destroy Operator experiment results in table 8.3 show that all three are somewhat successful in improving incumbent solutions. Two of these operators were developed specifically for the PDOP, figure 8.7 shows both score within the 95% confidence interval for average solution cost as Random Removal, vindicating their selection as destroy operators. The average cost improvements in Figure 8.8 does show significantly better cost improvement using the Random Removal operator. However, this is not the full story, as the Worst Trip Removal and Worst FTL Trip Removal operators showed rapid improvement during the first few iterations of the optimizer. In Figure 8.9, the kernel density estimate for each operator's solution cost improvement over the tests shows the max value for improvement for Worst Trip Removal and Worst FTL Trip Removal is over 40%. The highest improvement from Random Removal was under 30%, so each operator demonstrated strength in either average or best performance.

Table 8.3: Average Cost and Solution Improvement for Destroy Operators

Destroy Operator	**Average Cost**	**Average Solution Improvement (%)**
Random Removal	66102	9.94
Worst FTL Trip Removal	68883	5.97
Worst Trip Removal	67600	7.61

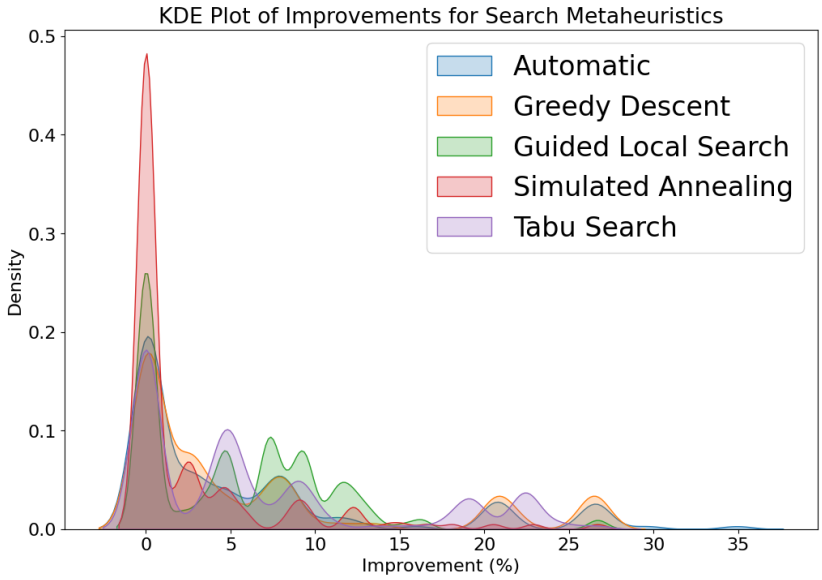


Figure 8.6: Estimated Distributions on Percent Improvement of Solutions for each Metaheuristic

One pattern seen with the two destroy operators that analyze trip efficiency is an optimization loop, where the routing model fails to obtain a better solution, causing the deterministic behavior of the destroy operator to repeat itself. This leads to the same partial solution being fed into the routing model at each iteration. This loop weakens the performance of the iterative run optimization tool overall, making it necessary to introduce randomness into the solver's behavior. This is the motivation for balancing the destroy operators in the final version of the solver, using a selection algorithm that will favor other operators if it gets trapped in a loop with the same operator.

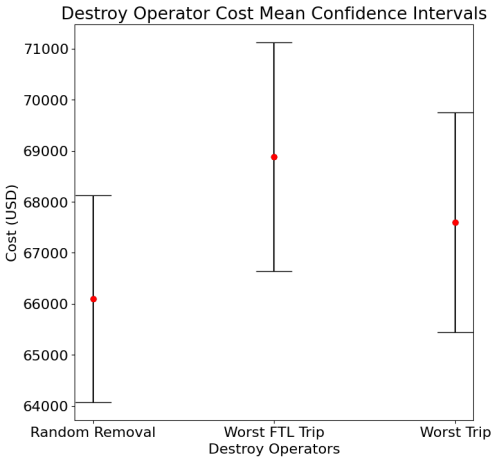


Figure 8.7: 95% Confidence Interval on the Mean Cost for each Destroy Operator

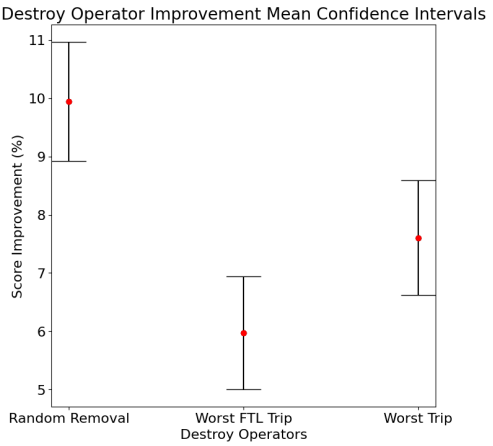


Figure 8.8: 95% Confidence Interval on the Mean Improvement on Initial Solution for each Destroy Operator

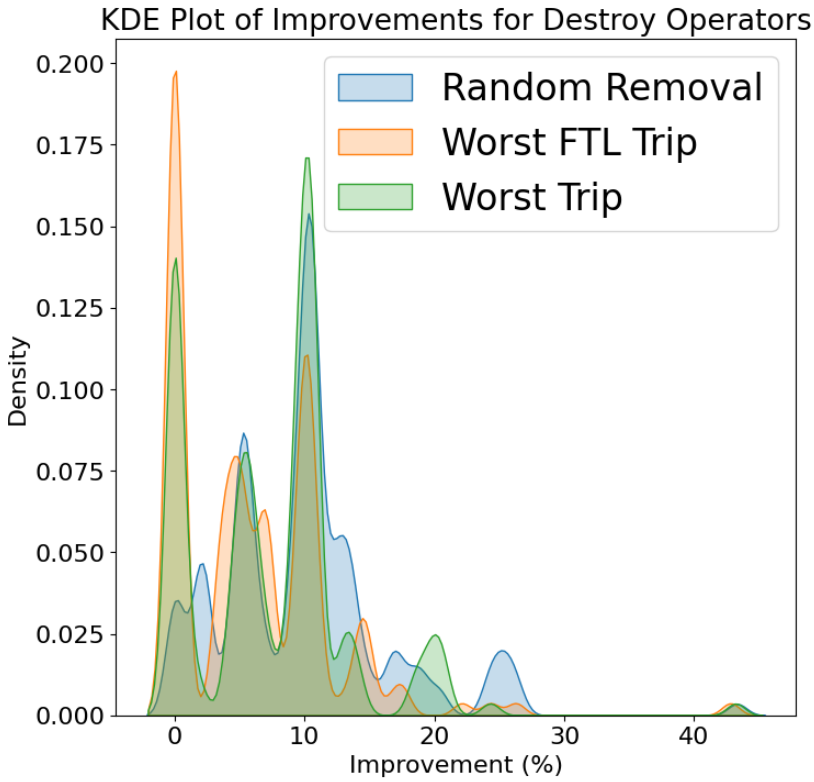


Figure 8.9: Estimated Distributions on Percent Improvement of Solutions for each Destroy Operator

8.2 Pool Distribution Optimization Performance

This section examines the results of a full array of testing scenarios in both synthetic and real data, comparing the UCB LNS Algorithm to benchmarks FTL Only, Single Run OR Tools, and Gurobi. The patterns in performance over different scenario types are discussed, and important takeaways from the testing are detailed.

8.2.1 Final Testing Results

In analysis of the UCB LNS algorithm testing results, significant variance in performance between algorithms, depending on the scenario parameters, quickly becomes clear. Although frequently achieving similar cost solutions, the FTL Only benchmark algorithm and the UCB LNS algorithm each outperform the other in some instances. It is important to consider that the FTL Only algorithm works very quickly, solving a simpler model but without the ability to use Pool Points. Despite this, the FTL Only algorithm can beat the UCB LNS algorithm in scenarios where the use of Pool Points is not advantageous. In the use case for Shipwell, the difference in computational resources between running the UCB LNS algorithm alone or running it while also running the FTL Only algorithm is negligible. Therefore, leveraging both optimization approaches concurrently, and selecting the best solution yields the best results, while hardly increasing computational demands. Results from testing in this section are compared against a **Bagged (FTL and UCB LNS)** algorithm solution, where the better of results from FTL Only and UCB LNS is used for each scenario instance.

Synthetic Data

The results from testing synthetic data reveal a number of patterns related to the circumstances where each algorithm performs best. First looking to small scenarios, Figure 8.10 indicates Gurobi outperforms the other algorithms in scenarios with one pool point only. Although Gurobi was fed solutions from the Single Run OR Tools algorithm as an incumbent solution, its performance is actually worse in the model using two pool points. This is an unexpected outcome, showing that the Gurobi model becomes too complex when multiple pool points are involved.

Thinking back to the modeling section for optimization scenarios in Gurobi, doubling the number of pool points doubles the number of pool nodes in the model. Although this is also true in OR Tools model, OR Tools allows for disjunctions which tell the optimizer those points can be bypassed when finding a solution, whereas Gurobi's Barrier solution search algorithm does not support this kind of disjunction, often resulting in an impracticable attempt to link the entire graph structure.

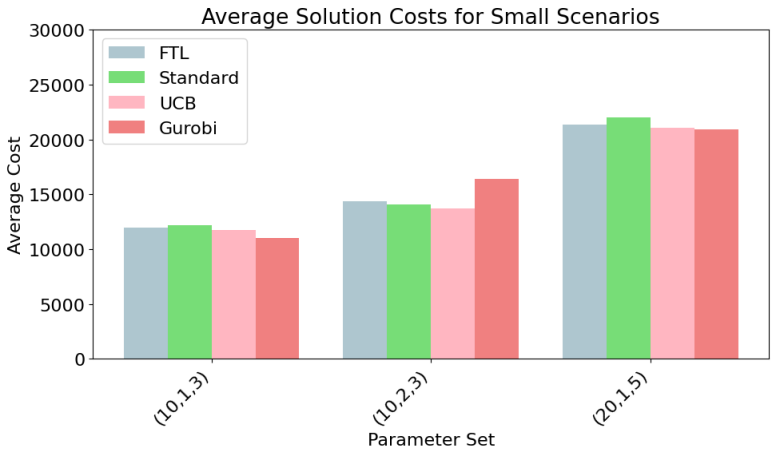


Figure 8.10: Performance of the Optimizers on Synthetic Data Parameters Gurobi can compute

Viewing the single pool point scenarios, Gurobi solutions act as a lower bound on solution cost, as we cannot know the true optimal solution to an NP-Hard problem. Table 8.4 shows how much Gurobi reduced solution costs compared to the FTL Only benchmark and the UCB LNS Algorithm. In the smallest instance of (10,1,5), Gurobi is able to improve on the UCB LNS Algorithm by nearly 6%, but as the problem scales up, Gurobi loses the ability to find much better solutions. In general, as scenarios scale up and stress the computational limits of the OR Tools model, the gap between the true global optimum and the local optima found by the optimizers should increase. This is represented in the relationship between FTL Only performance, which is leagues ahead in speed and efficiency, and the performance of the UCB LNS Algorithm in larger scenarios.

When comparing solutions for large scenarios of 50 or more orders, the UCB LNS Algorithm is affected negatively by scenarios with multiple pool points. Fig-

Table 8.4: Comparative Reduction in Average Solution Cost with Gurobi

Instance	FTL Only (%)	UCB LNS (%)
(10,1,3)	8.06	5.79
(10,2,3)	-13.90	-19.50
(20,1,5)	2.14	0.74

ure 8.11 compares the average solution cost between the Algorithms over each scenario parameter set, showing that FTL Only averaged a lower cost than UCB LNS in every large scenario type that includes 2 or more pool points. However, even within a scenario parameter set where either FTL Only or UCB LNS generally performs better, the other optimization algorithm can potentially find the lower cost solution. This means the Bagged solution, taking the better of these two in each instance, always beats the lowest average cost against the FTL Only, Single Run OR Tools (referred to as Standard), and UCB LNS Algorithms individually.

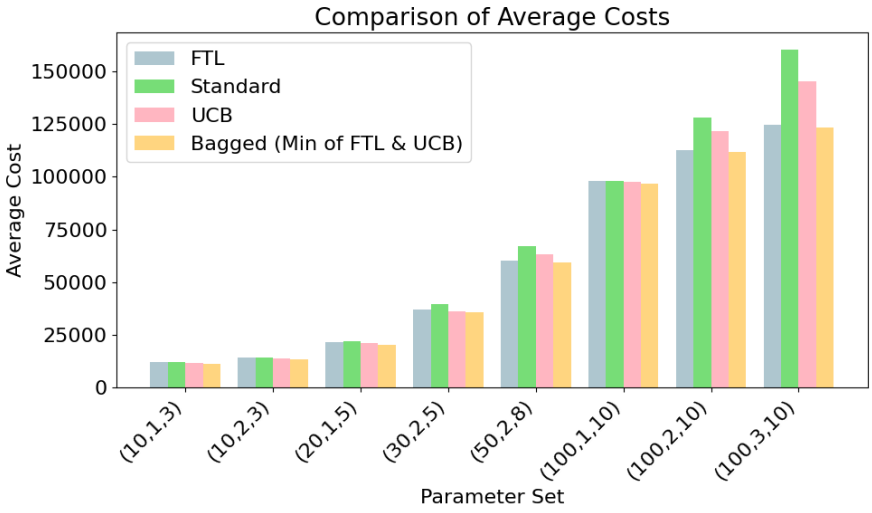


Figure 8.11: Performance of the Optimizers on Synthetic Data for Different Scenario Parameters

Any improvement over an FTL Only solution represents real savings a Shipwell client would gain by using one of the PDOP algorithms. Therefore, the true value of these algorithms is the net improvement over using just the FTL Only algorithm. Figure 8.12 shows the improvement percentage by comparing the best optimization setup, Bagged, with the FTL Only solution. Because the Single Run OR Tools optimizer forms the initial solution for the UCB LNS Algorithm, it

is not included when choosing the Bagged algorithm solution. The percentage improvement in smaller scenarios is much greater, although the average savings in scenarios with 100 orders still results in over 1000$ per scenario, which would amount to thousands in savings daily for a company with a large logistics network. At the 95% confidence level, scenarios with 100 orders and 1 pool point performed significantly better with Bagged than an FTL Only solution. This scenario type is most similar to the scenarios created from Whole Foods Market data, which bodes well for the Bagged optimizer earning the company significant savings. A smaller company, with fewer orders to optimize in a single day could also benefit from the large improvements seen in small scenarios.

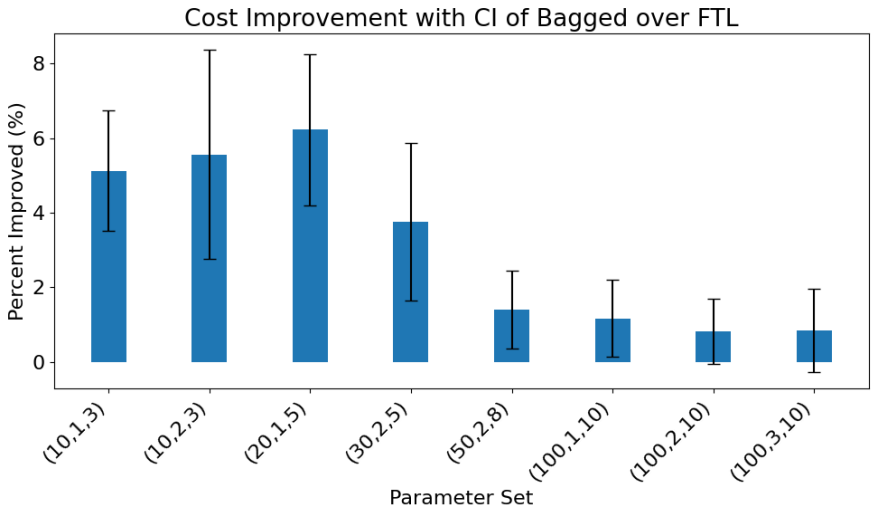


Figure 8.12: Cost Improvement of the Bagged Optimizer Solutions over FTL Optimization Costs alone for Synthetic Data, with a 95% Confidence Interval

Whole Foods Market Data

As in the Synthetic Data, results from Whole Foods Market data testing showed variance in algorithm performance between different pool points. Notably, the better algorithm for a scenario varied between days, even at the same pool point. Figure 8.13 shows the average cost for each pool point and pool point pair tested. FTL Only is seen to perform better than UCB LNS on average at pool points 2 and 5, while approximately tying at the combination of pool points 1 and 3. UCB LNS finds lower average solution cost at pool points 4 and 6. Though roughly tying on average cost in scenarios with pool points 1 and 3, the Bagged solution

beats FTL Only and UCB LNS, so both perform best on at least one of the five days. In fact, none of the pool points had an individual algorithm as the best performer on all five days tested. Figure 8.14 shows a comparison between daily solutions for pool point 6, where UCB LNS achieved its highest improvement over the FTL Only algorithm. The daily results for the remaining pool points are included in the Additional Figures section.

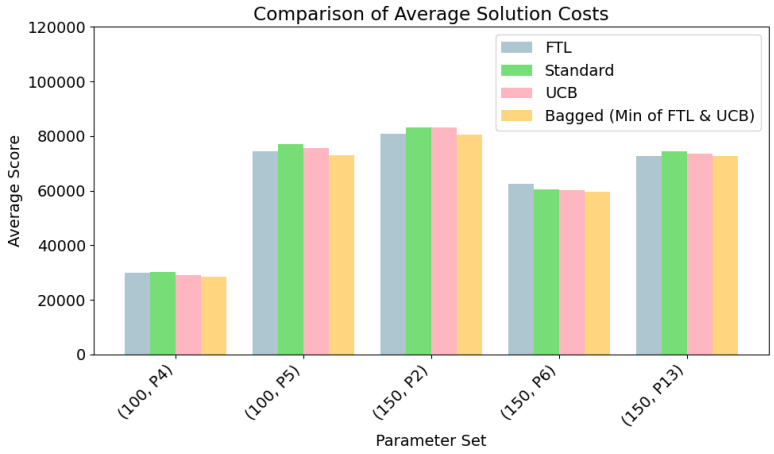


Figure 8.13: Performance of the Optimizers on Data from Whole Foods Market Pool Points

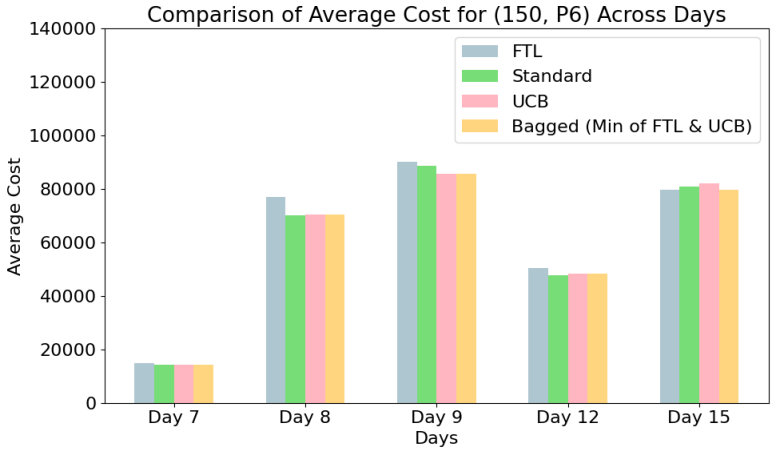


Figure 8.14: Performance of the Optimizers for Pool Point 6

Overall cost improvement ranged greatly between pool points, with Table 8.5 showing a range from 0.26% to 5.65% on average improvement over the FTL Only solution. Examining these improvements more closely, Figure 8.15 shows the mean

improvement with a 95% confidence interval. The intervals confirm the Bagged algorithm significantly improved solution costs for pool points 4, 5, and 6. The net improvement in cost over the five days optimized is 88,885$, which accounts for 95.37% of orders during the 5 day period, one business week. Since each scenario is optimized in three separate instances, this is divided by three, leading to a savings of 29,628$ per week, or 1,540,673$ annually. In percentages, the net improvement over FTL Only is 1.849%. This percentage can be applied to the extrapolated annual expense for Whole Foods Market's freight shipping logistics needs, taking the expected net cost from 84,859,614$ down to 83,318,941$.

Table 8.5: Percent Cost Over FTL Only Improvements for Bagged Algorithm

Scenario Parameters	**Cost Improvement Over FTL Only (%)**
(100, P4)	5.65
(100, P5)	2.20
(150, P2)	1.42
(150, P6)	4.23
(150, P13)	0.29

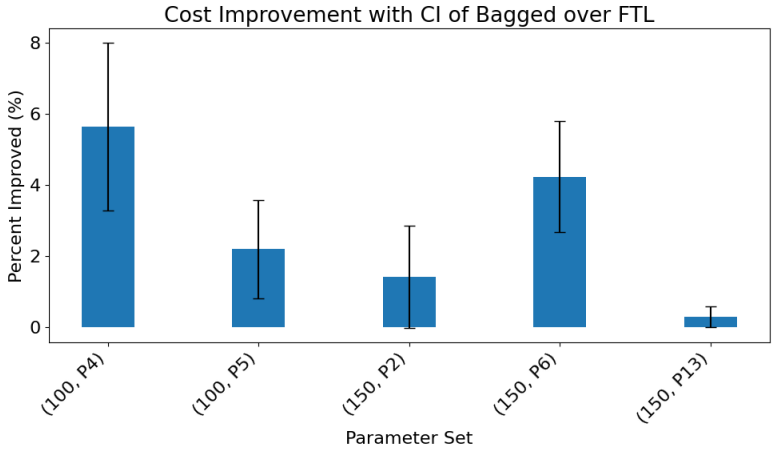


Figure 8.15: Cost Improvement of the Bagged Optimizer Solutions over FTL Optimization Costs alone for Whole Foods Market Data

8.2.2 Solution Patterns

Through testing real and synthetic data scenarios, correlations in both the general behavior of routes and the quality of solutions are visible. These patterns are

related to qualitative observations made through inspection of countless plotted solutions and data-frames containing information about each order's journey in the solution.

Subjective analysis shows a clear pattern from the solutions in the relationship between distance of destination and pool point use. Some scenarios, having destinations that are ten times greater than the typical distance between the orders and their local pool point, benefit from routing every order through a pool point. Figure 8.16 shows an example of a solution where each order routes through a pool point due to the destinations being thousands of miles away. Conversely, in a scenario where the destinations are very near the pool point, significantly more orders travel by FTL only. Figure 8.17 shows a solution to a scenario where the destinations are interspersed with pickup coordinates.

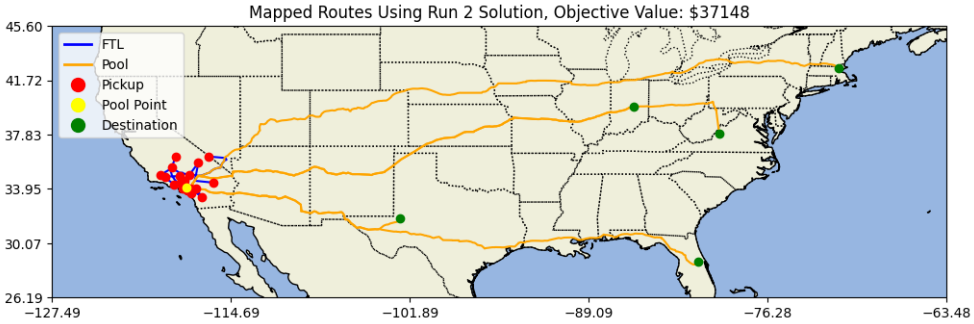


Figure 8.16: A Solution with every Order Routed through the Pool Point

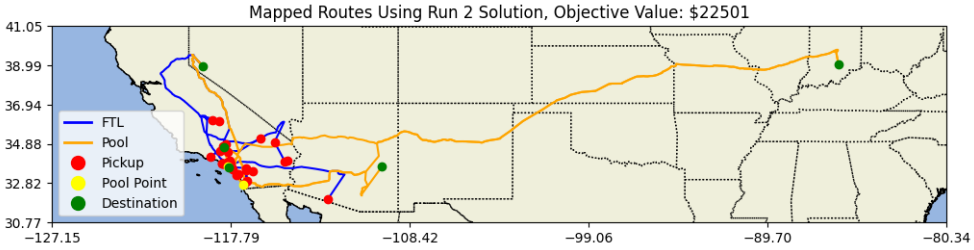


Figure 8.17: A Solution with most orders traveling by FTL only

Results for the UCB LNS Algorithm from optimization scenarios including 2 or 3 pool points, and over 30 orders, underperformed compared to the FTL Only

model. Further corroborated by Gurobi's inability to find acceptable solutions in models with multiple pool points, the added complexity of managing multiple intermediary points can negate the potential for maximizing efficiency. In other words, although placing two pool points in a model increases the solution space, potentially creating a new global optimum, it increases the algorithm's difficulty to navigate that space and find a good solution. Considering that in Whole Foods Market's data, the scenarios with a combination of pool points 1 and 3 saw the least improvement over FTL Only, their expected annual savings is likely even higher if pool point 1 and 3 are optimized separately.

8.2.3 Research Insights

Through answering the main research question and sub-questions of this study, unanticipated insights from test results will serve to improve the optimization algorithm's integration with Shipwell's Transport Management System and potentially further improve solution quality. There were three main unexpected takeaways.

First, results from both real and synthetic data tests on a variety of scenarios reveal that ultimately; a Bagged algorithm using FTL Only and UCB LNS models, aggregating the most complex and fastest algorithms respectively, is the best way to obtain a solution that beats any one algorithm in all scenario instances. Looking at the use case at Shipwell, any approach that ignores algorithms that may find the best solution is leaving savings untouched. Although the UCB LNS algorithm is the best overall performer, the FTL Only algorithm requires less than 10% of the time and compute power, so Bagging the algorithms is certainly worth the extra route cost savings.

Second, the unified optimization model, inside OR Tools, exhibits bias toward routing orders through a pool point. This is shown by the FTL Only algorithm results outperforming the UCB LNS algorithm in some instances, despite that the UCB LNS algorithm can feasibly produced the same solutions as the FTL Only algorithm. Failure to find superior solutions the FTL Only algorithm does find shows the UCB LNS algorithm's tendency to favor routing through a pool point. Nonetheless, the behavior is not surprising, due to the unified model's cost simplification on routes running from pool points to distribution centers. Vehicles

in the OR Tools model visiting a pool point become Ghost Vehicles, which can visit multiple distribution centers for the cost of visiting just one. This modeling decision allows OR Tools to select the lowest distance to a distribution center out of those a truck's contents are bound for, to represent the pool vehicle's cost. That prediction will always sit at or below the true cost if the optimizer is making insertion decisions optimally. Looking forward, the preference for pool routes could be mitigated by testing different discounts to pool routes in the OR Tools model, while maintaining the same discount when calculating the solution cost during post-processing allocation of pool vehicles. For example, increasing the cost of pool routes by 10% inside the model could counter the bias towards visiting pool points, leading to better solution costs without actually changing the method for calculating the solution's true cost.

Third, optimizing multiple pool points at once is not worthwhile in large scenarios where the PDOP becomes computationally challenging. Figure 11.15 in the Appendix shows an example of a scenario where the complexity of multiple pool point caused the UCB LNS algorithm to find a suboptimal solution compared to FTL Only. It is generally apt to split multiple pool points into separate optimization scenarios, unless the pool points are near one another and have 30 or fewer orders to route between them. This criterion is not met by the data from Whole Foods Market, so until a Shipwell client has a logistics use case that fits this description, pool points should always be optimized separately.

Chapter 9

Conclusion and Future Research

This study's underlying purpose is to develop a tool that meets Shipwell's need for solving the Pool Distribution Optimization Problem at scale. Research questions and sub-questions split this task into components, each designed to yield insight for developing the final algorithm to use in production. Section 9.1 of this chapter covers important conclusions from the research done, which are also recommendations for how Shipwell should set up their optimization tool. Then in section 9.2, concerns and limitations of the current model, with regards to implementation in the Transport Management System are discussed along with future avenues for research that build on the results of this study.

9.1 Conclusion

This study set out to answer the research question: *What is an effective and scalable approach for an optimization algorithm minimizing cost in the Pool Distribution Optimization Problem, which meets the capacity needs of major United States suppliers and distributors?* The answer is best encapsulated in a set of research conclusions, and recommendations for Shipwell both in optimizer implementation and future research avenues. Six main research conclusions as described next answer the research question and sub-questions of this study.

First, making incremental improvements to routing solutions requires accurate driving distance estimates, as improvements in cost would be otherwise overshadowed by uncertainty in the actual distance between coordinates in the routing model. The best way to achieve this for very little computation or financial cost is by using an augmented route matrix based on major cities, where pre-calculated distances between key locations are added to the haversine distance between the actual model coordinates and the nearest major city coordinates. Major cities better reflect the likely coordinates for a pickup or dropoff than evenly dispersed geographic coordinates.

Next, out of the off-the-shelf professional routing optimization tools, Google's OR Tools is the most robust and therefore practical for solving the PDOP. Despite Gurobi performing very well in small scenarios, the difficulty of this highly constrained optimization problem is clear in Gurobi's inability to scale to near the practical needs of Shipwell. OR Tools uses algorithms that navigate through the solution space, whereas Gurobi is stymied decomposing the initial model into countless sub-problems. However, even OR Tools struggles greatly when multiple pool points are introduced to the model because the total nodes in the model then increases significantly, so it is better to separate pool points into their own separate models with nearby orders only.

Third, finding an initial feasible solution to the PDOP is challenging in large scenarios, and the best approach is to modify the problem so that the most flexible component, the FTL Vehicle, is optimized separately from the Pool Vehicle component. This is achieved using an approximation for the cost of pool routes, based on the contents of trucks arriving at a pool point, then finding the true cost of the pool routes during a post-processing step after OR Tools outputs a solution. This prevents the need for managing a set of Pool Vehicles inside the model, which would dramatically increase the model's constraints in an already highly constrained problem. FTL Vehicle behavior affects the demands of the Pool Vehicles, but not vice versa.

Fourth, Automatic is the best choice for a Search Metaheuristic when using OR Tools, although the difference between the viable choices is small. Figure 8.6 shows that using the Automatic Search Metaheuristic found the highest im-

provement in any scenario, at 35%, showing the possible advantage from the only metaheuristic that can change between the other metaheuristics. Unlike Search Metaheuristic, the optimal First Solution strategy differs when finding initial solutions or repairing partial solutions. Parallel Cheapest Insertion is by default the optimal strategy for initial solutions because none of the other strategies can consistently find a solution to large scenarios over 30 orders. However, during the partial solution repair process, the Local Cheapest Insertion strategy is the better parameter. Figure 8.2 shows significantly more improvement on initial solutions compared to the Parallel Cheapest Insertion or Global Cheapest Arc, this parameter selection has an impact.

Fifth, to facilitate partial solution reconstruction, a mechanism for Large Neighborhood Search, similar to in the research done by Wolfinger [42] on highly constrained models with trans-shipments, is necessary to escape local optima. The operators should introduce randomness while also focusing on the pool point aspect of the model. This is best achieved by using multiple operators, selected at each iteration through dynamic scores based on past performance. The operators respectively introduce randomness, analyze inefficient routes, and focus on vehicles either visiting or skipping a pool point.

Finally, a Bagged version of the optimization algorithm, which includes the UCB LNS and FTL Only algorithms, ensures the best cost improvement possible regardless of the scenario, with minimal additional compute requirements. The FTL Only algorithm acts as a check against poor UCB LNS solutions, and to efficiently handle scenarios where pool points turn out to be unsuitable.

9.2 Limitations and Future Research

This study is conducted with the mindset of creating a tool that Shipwell can build into its TMS without needing to change the design in a major way. However, one component present in TMS not feasible to test, was operating hours data for each location in the model. Handling the additional constraints from specific operating hours per location will certainly affect solutions, so it is recommended to conduct further tests, especially comparing the results from scenarios tested in

this study, to determine if the solutions change dramatically, which could offset the balance between each algorithm in terms of optimality. Another potential limitation not tested was the splitting of orders around a single pool point. For example, if a pool point has over 200 orders to optimize for the day, it needs to be split into two scenarios. The question of how to most efficiently split such a group up could strongly impact final solution costs. Perhaps geographically clustering the distribution of pickup coordinates into two groups, or using analysis to compute likeness between orders, would make a better segmenting approach. This is a question future optimization experts at Shipwell could research to further improve the optimization tool.

Limitations aside, there are avenues for further optimizer improvement Shipwell could research. Although this study tests many optimization parameters, there is not enough time to test everything. Future researchers at Shipwell can tune some of the parameters there are not closely tested in this study. First, there is the order clustering threshold. Allowing the optimization model to cover more orders at once, the most similar orders are grouped into clusters. The threshold for these clusters was 10%, so once the cluster is over 10% of a single vehicle's capacity, that is taken as an order to use in the model. Modifying the cluster size could have a big impact on solutions. Intuitively, the trade-off when selecting the cluster threshold would be between capacity for orders and potential for a better global optimum in the solution space. Another candidate parameter for further testing is the pool route discount calculated in the optimizer. During this study, the pool route costs used to approximate pool routes inside the optimizer and the discount applied during post processing are the same value. However, creating a disparity where inside the OR Tools optimizer the pool routes are estimated as slightly higher than their actual cost, could counteract the UCB LNS Algorithm's bias toward using pool points. Whether this change improves the overall Bagged Algorithm, or weakens UCB LNS component's solution quality, is yet to be seen.

During future research, Shipwell's data scientists could experiment with larger compute clusters than those available for this study. This would clarify whether extra power allows OR Tools to optimize a larger model, or if the complexity reaches a point where the tool itself is the limit, rather than compute power. They should also obtain data from more Shipwell customers, as each company has

unique elements from business rules to distribution patters in order, pool point, and distribution center location. These factors cannot be replicated synthetically without access to the real data initially, and may lead to emergent patterns in optimizer behavior that inspire further model adjustments.

Finally, to further improve the UCB LNS model, a destroy operator that biases optimization away from using pool points should be tested as part of the UCB selection algorithm. The results from testing showed UCB LNS exhibited bias towards routing through pool points. Furthermore, two of the destroy operators used are agnostic to pool points, while the third "Worst FTL Trip Removal" biases the model towards routing through pool points. A fourth operator that biases the model away from pool point use could compensate for the apparent bias in the model towards pool points.

Bibliography

[1] K. Alicke et al. "How COVID-19 is reshaping supply chains". In: *McKinsey Company* (Nov. 2021). URL: https://www.mckinsey.com/business-functions/operations/our-insights/how-covid-19-is-reshaping-supply-chains.

[2] O. Bräysy et al. "Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms". In: *Transportation Science* 39.1 (2005), pp. 104–118. DOI: 10.1287/trsc.1030.0056.

[3] T. Cuvelier et al. "OR-Tools' Vehicle Routing Solver: A Generic Constraint-Programming Solver with Heuristic Search for Routing Problems". In: *24e Congrès Annuel de La Société Française de Recherche Opérationnelle et d'aide à La Décision*. Feb. 2023. URL: https://hal.science/hal-04015496.

[4] E. Danna et al. "Exploring relaxation induced neighborhoods to improve MIP solutions". In: *Mathematical Programming* 102.1 (2005), pp. 71–90. DOI: 10.1007/s10107-004-0544-5.

[5] E. Demir et al. "Last mile logistics: Research trends and needs". In: *IMA Journal of Management Mathematics* 33.4 (2022), pp. 549–561. DOI: 10.1093/imaman/dpac006.

[6] E. W. Dijkstra. "A Note on Two Problems in Connexion with Graphs". In: *Numerische Mathematik* 1 (1959), pp. 269–271. DOI: 10.1007/BF01386390.

[7] FHWA. *National Highway System—Planning—FHWA*. https://www.fhwa.dot.gov/planning/national_highway_system/.

[8] M. Fischetti et al. "The Feasibility Pump". In: *Mathematical Programming* 104.1 (2005), pp. 91–104. DOI: 10.1007/s10107-004-0570-3.

[9] D. Ghosh et al. "Tolerance-based greedy algorithms for the traveling salesman problem". In: (Nov. 2008). DOI: 10.1142/9789812813220_0005.

[10] F. Glover. "Future Paths for Integer Programming and Links to Artificial Intelligence". In: *Computers & Operations Research* 13.5 (1986), pp. 533–549. DOI: 10.1016/0305-0548(86)90048-1.

[11] F. Glover et al. *Tabu Search*. Boston, MA: Kluwer Academic Publishers, 1997. DOI: 10.1007/978-1-4615-6089-0.

[12] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. DOI: 10.5555/534133.

[13] Gonzalez. "Amazon to buy Whole Foods for $13.7 billion in bid to become major grocer". In: *The Seattle Times* (June 2017). URL: https://www.seattletimes.com/business/amazon/amazoncom-buys-whole-foods-for-137-billion/.

[14] Google. *OR-Tools Constraint Programming Documentation*. 2024. URL: https://developers.google.com/optimization/routing/constraint_programming.

[15] P. Grangier et al. "An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization". In: *European Journal of Operational Research* 254.1 (2016), pp. 80–91. DOI: 10.1016/j.ejor.2016.03.040.

[16] *Gurobi Optimizer Reference Manual*. Gurobi Optimization, LLC, 2024. URL: https://www.gurobi.com/documentation/.

[17] P. E. Hart et al. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[18] H. Hoos et al. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, 2005. DOI: 10.1016/B978-155860872-6/50000-4.

[19] J. Kennedy et al. "Particle swarm optimization". In: *Proceedings of ICNN'95 - International Conference on Neural Networks*. Vol. 4. IEEE, 1995, pp. 1942–1948. DOI: 10.1109/ICNN.1995.488968.

[20] S. Kirkpatrick et al. "Optimization by Simulated Annealing". In: *Science* 220.4598 (1983), pp. 671–680. DOI: 10.1126/science.220.4598.671.

[21] S. Koziel et al. "Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization". In: *Evolutionary Computation* 7.1 (1999), pp. 19–44. DOI: 10.1162/evco.1999.7.1.19.

[22] M. Lee et al. “Flexible Delivery Routing for Elastic Logistics: A Model and an Algorithm”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.7 (2022), pp. 6864–6882. DOI: 10.1109/TITS.2021.3063195. URL: https://doi.org/10.1109/TITS.2021.3063195.

[23] B. Li et al. “An Overview and Experimental Study of Learning-Based Optimization Algorithms for the Vehicle Routing Problem”. In: *IEEE/CAA Journal of Automatica Sinica* 9.7 (2022). DOI: 10.1109/JAS.2022.105677.

[24] J. Lysgaard et al. “A New Branch-and-Cut Algorithm for the Capacitated Vehicle Routing Problem”. In: *Mathematical Programming* 100.2 (2004), pp. 423–445. DOI: 10.1007/s10107-003-0514-8.

[25] S. N. Medvedev. “Greedy and Adaptive Algorithms for Multi-Depot Vehicle Routing with Object Alternation”. In: *Automation and Remote Control* 84.3 (2023), pp. 305–325. DOI: 10.1134/S0005117923030086.

[26] S. Muñoz-Herrera et al. “Constrained Fitness Landscape Analysis of Capacitated Vehicle Routing Problems”. In: *Entropy* 24.1 (2024), p. 53. DOI: 10.3390/e24010053.

[27] ResearchGate. *An illustrative example of the branch-and-bound algorithm.* https://www.researchgate.net/figure/An-illustrative-example-of-the-branch-and-bound-algorithm_fig1_286510562.

[28] *Routing Options - OR-Tools.* https://developers.google.com/optimization/routing/routing_options.

[29] R. Ruiz-Torrubiano. “Modeling Local Search Metaheuristics Using Markov Decision Processes”. In: *Ar5iv* (2024). DOI: 10.1007/3-540-16761-7_92.

[30] F. F. S. Sánchez et al. “Comparative Study of Algorithms Metaheuristics Based Applied to the Solution of the Capacitated Vehicle Routing Problem”. In: *Novel Trends in the Traveling Salesman Problem.* IntechOpen, 2020. DOI: 10.5772/intechopen.91972.

[31] T. Stefański et al. “Modeling and Optimization of Multi-echelon Transportation systems—A hybrid approach”. In: *Proceedings of the Federated Conference on Computer Science and Information Systems.* 2017, pp. 1057–1064. DOI: 10.15439/2017F80.

[32] R. Storn et al. “Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces”. In: *Journal of Global Optimization* 11.4 (1997), pp. 341–359. DOI: 10.1023/A:1008202821328.

[33] K. Tarasov. "Amazon bought Whole Foods five years ago for $13.7 billion. Here's what's changed at the high-end grocer". In: *CNBC* (Aug. 2022). URL: https://www.cnbc.com/2022/08/25/how-whole-foods-has-changed-in-the-five-years-since-amazon-took-over.html.

[34] R. Tibshirani et al. "Estimating the number of clusters in a data set via the gap statistic". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63.2 (2001), pp. 411–423. DOI: 10.1111/1467-9868.00293.

[35] V. Tomar et al. "Metaheuristic Algorithms for Optimization: A Brief Review". In: *Engineering Proceedings* 59 (2024). DOI: 10.3390/engproc2023059238.

[36] P. Toth et al. *Vehicle Routing: Problems, Methods, and Applications*. 2nd ed. Society for Industrial and Applied Mathematics, 2014. DOI: 10.1137/1.9781611973594.

[37] United States Postal Service, Office of Inspector General. *The Untold Story of the ZIP Code*. https://www.uspsoig.gov/reports/white-papers/untold-story-zip-code. 2013.

[38] R. Vakili et al. "Multi-echelon green open-location-routing problem: A robust-based stochastic optimization approach". In: *Scientia Iranica* (2020). DOI: 10.24200/sci.2020.52149.2564.

[39] P. Varman et al. "A parallel vertex insertion algorithm for minimum spanning trees". In: *Automata, Languages and Programming*. Ed. by L. Kott. Springer, 1986, pp. 424–433. DOI: 10.1007/3-540-16761-7_92.

[40] C. Voudouris. "Guided Local Search—An illustrative example in function optimisation". In: *BT Technology Journal* 16.3 (1998), pp. 46–50. DOI: 10.1023/A:1009683510033.

[41] C. Voudouris et al. "Guided local search and its application to the traveling salesman problem". In: *European Journal of Operational Research* 113.2 (1999), pp. 469–499. DOI: 10.1016/S0377-2217(98)00263-8.

[42] D. Wolfinger. "A Large Neighborhood Search for the Pickup and Delivery Problem with Time Windows, Split Loads and Transshipments". In: *Computers & Operations Research* 126 (2021), p. 105110. DOI: 10.1016/j.cor.2020.105110.

[43] H. Xu et al. “Differential Evolution Algorithm for the Optimization of the Vehicle Routing Problem in Logistics”. In: *2012 Eighth International Conference on Computational Intelligence and Security*. 2012, pp. 48–51. DOI: 10.1109/CIS.2012.19.

Additional Figures

Whole Foods Market Results by Pool Point

The following graphics depict scores for the optimization algorithms at each pool point or pair from the data provided by Whole Foods Market. Notably, from one day to the next, the best performing algorithm changes even at the same pool point.

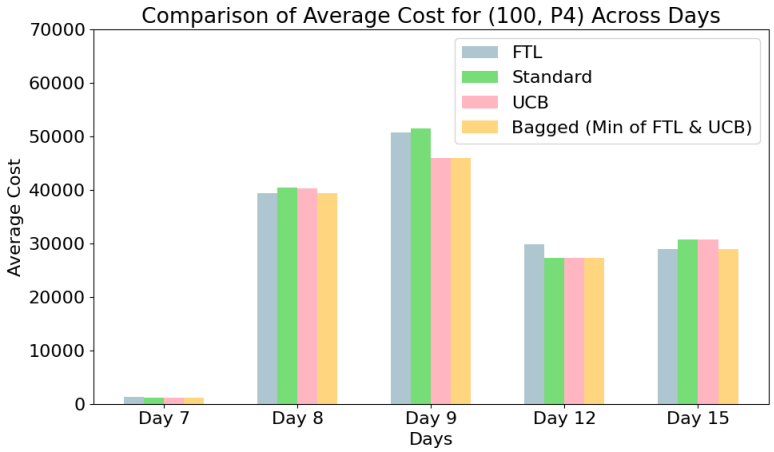


Figure 10.1: Performance of the Optimizers for Pool Point 4 over 5 dates in February 2024

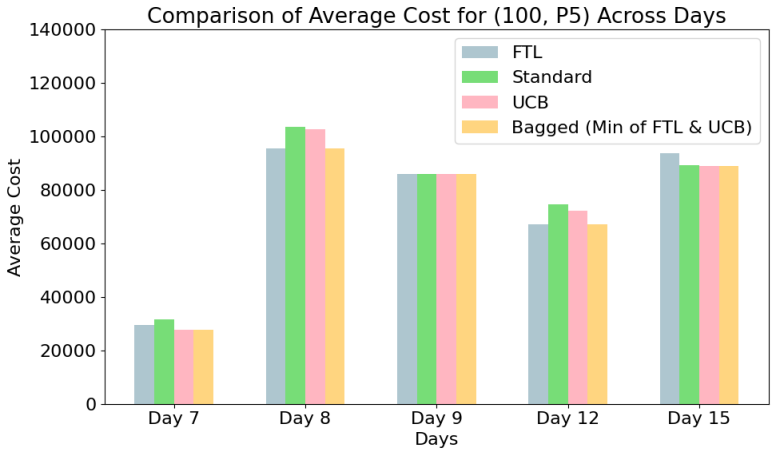


Figure 10.2: Performance of the Optimizers for Pool Point 5 over 5 dates in February 2024

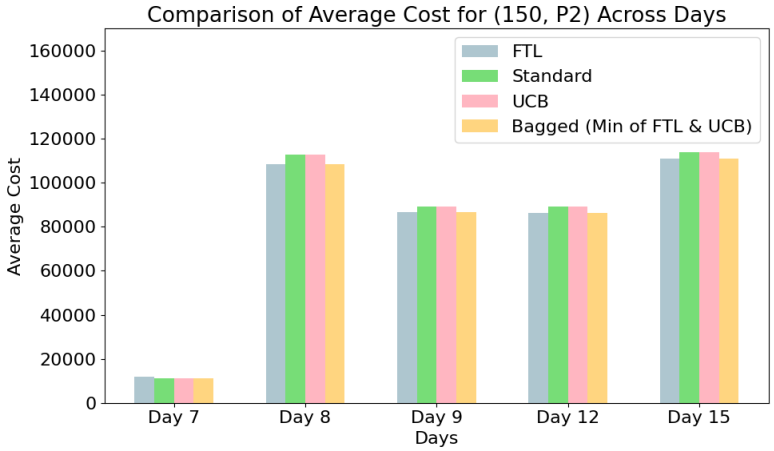


Figure 10.3: Performance of the Optimizers for Pool Point 2 over 5 dates in February 2024

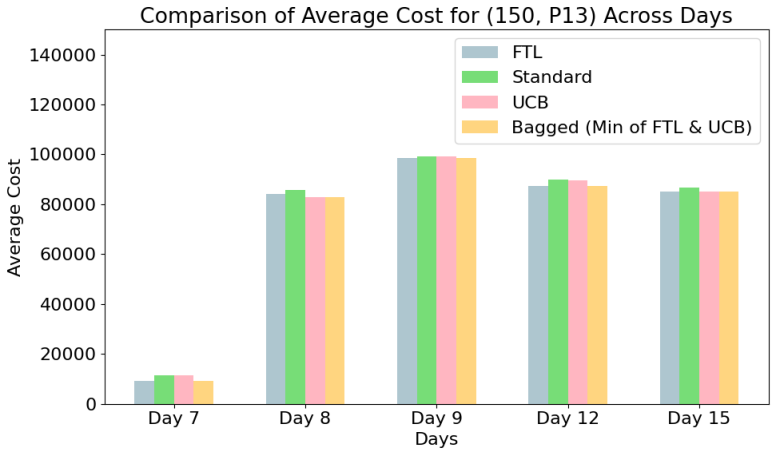


Figure 10.4: Performance of the Optimizers for Pool Points 1 and 3 combined over 5 dates in February 2024

Appendix

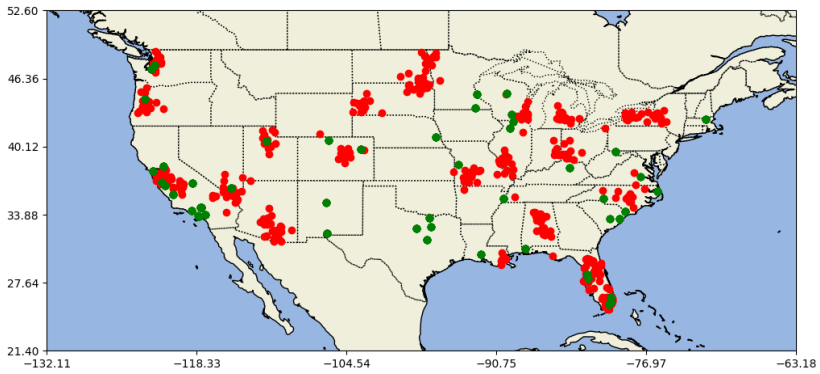


Figure 11.5: Pickup Locations in Red and Dropoff Locations in Green for Synthetic Data used in Distance Approximation Testing

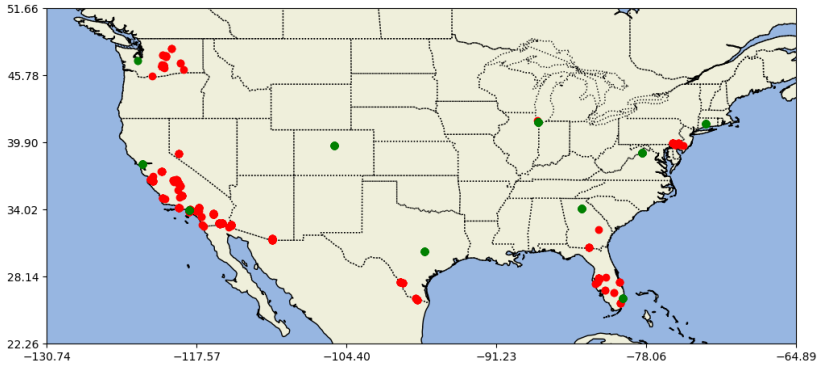


Figure 11.6: Pickup Locations in Red and Dropoff Locations in Green for WFM Data used in Distance Approximation Testing

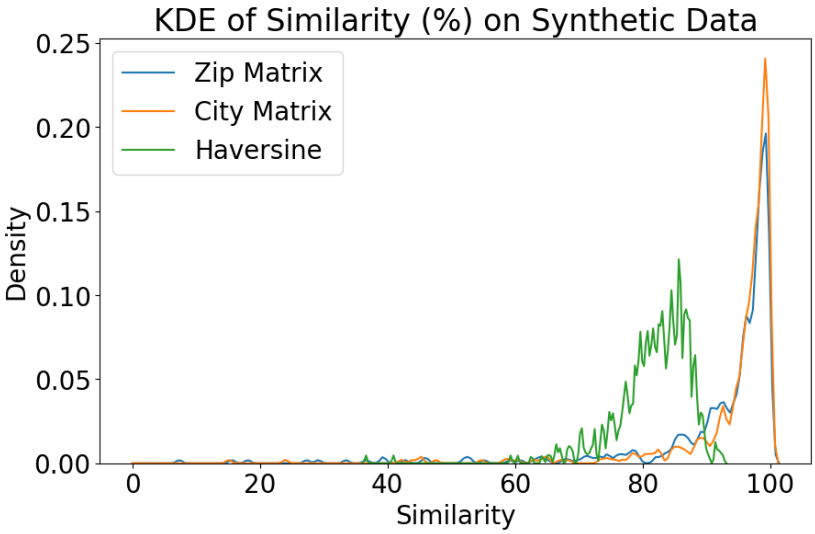


Figure 11.7: Similarity of Each Distance Matrix to Google Maps API Predictions

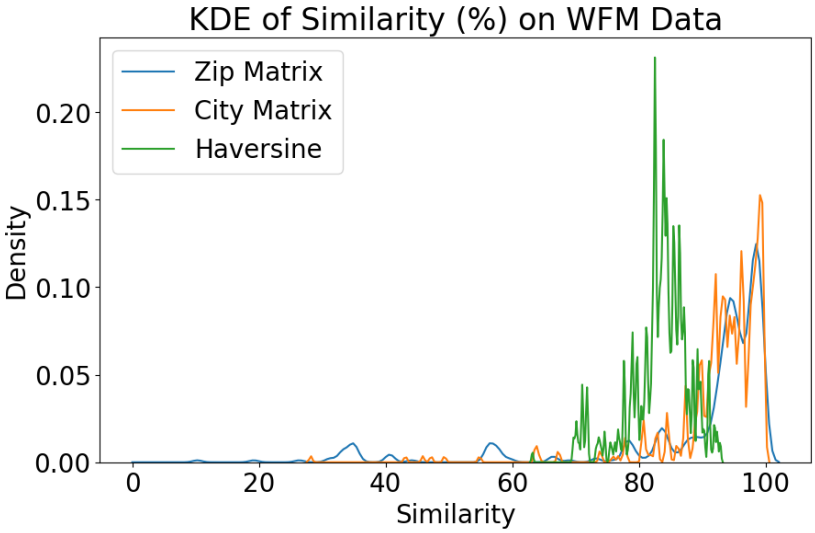


Figure 11.8: Similarity of Each Distance Matrix to Google Maps API Predictions

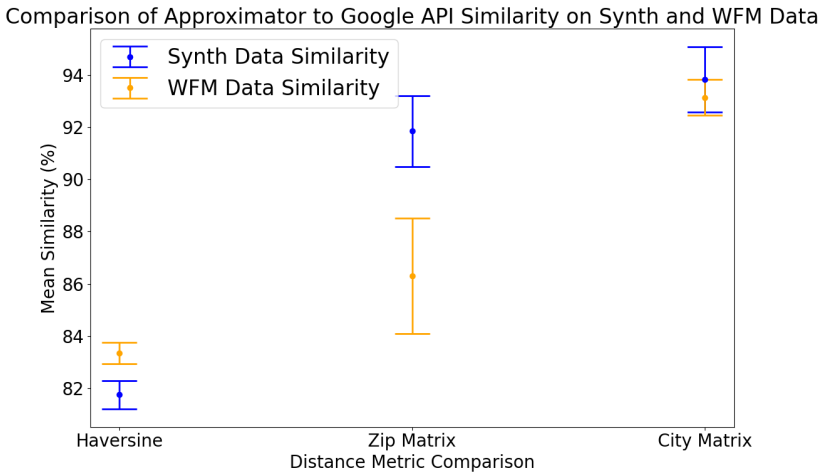


Figure 11.9: 95% Confidence Interval on Similarity

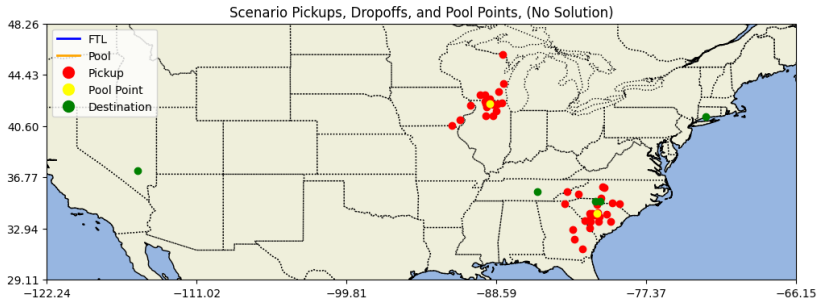


Figure 11.10: Optimizer Scenario with 50 Orders and 2 Pool Points

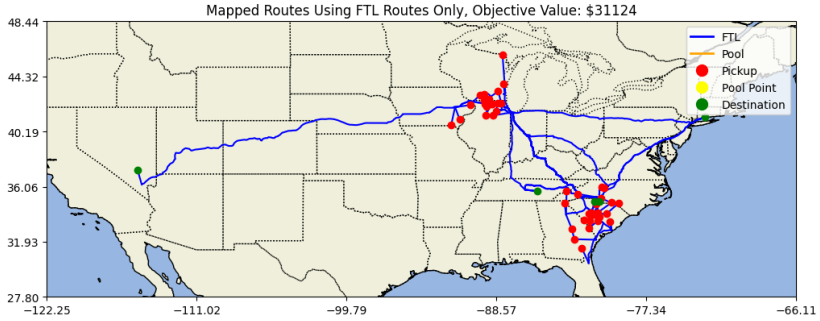


Figure 11.11: Optimizer Solution using FTL Only Algorithm

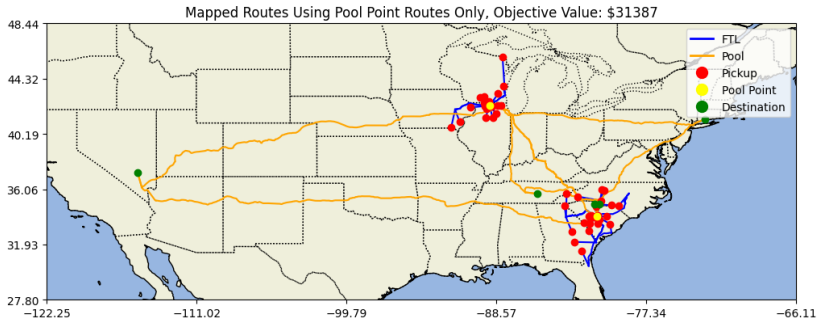


Figure 11.12: Optimizer Solution with All Routes Required to Travel through a Pool Point

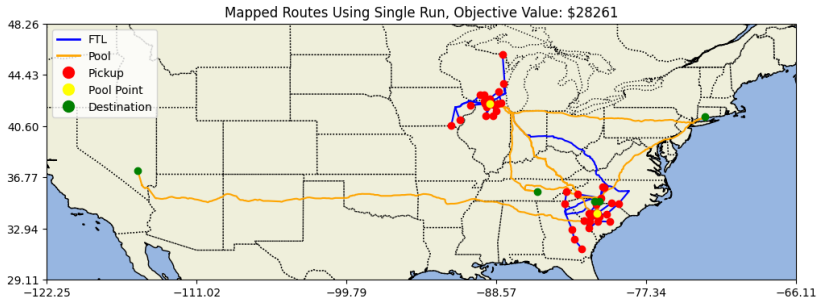


Figure 11.13: Optimizer Solution after a single run of the OR Tools model

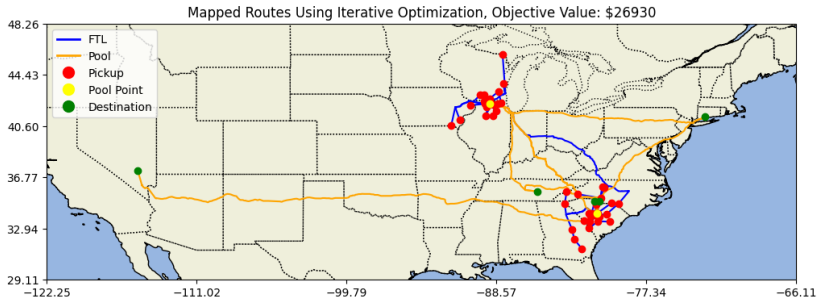


Figure 11.14: Optimizer Solution after using the Iterative UCB LNS Algorithm

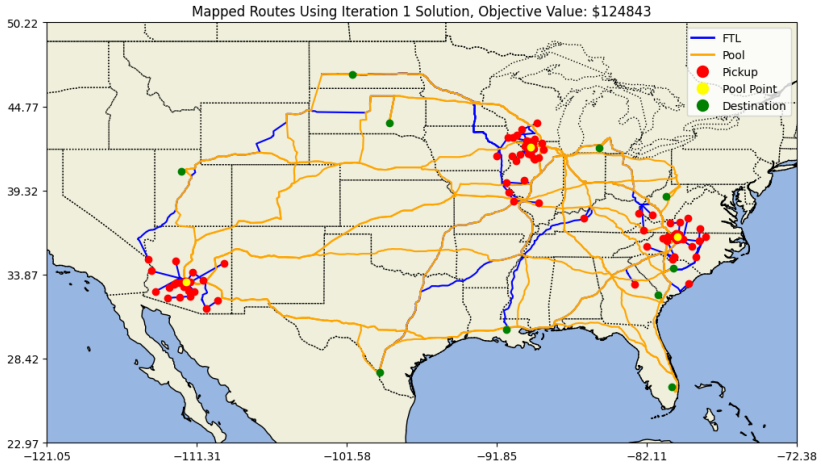


Figure 11.15: UCB LNS Algorithm Solution to a Scenario with 3 Pool Points