Meal forecasting with capacitated vehicle routing

Algoritmi di Ottimizzazione Exam Project

Author:
Lorenzo Rossi

Academic Year:
2022/2023

Contents

1 Project description **1**

2 Meal Forecasting **3**

2.1 Details 3

2.2 Results 4

3 Data Generation **6**

3.1 Ingredient composition 6

3.2 Distribution center locations 7

4 Vehicle Routing **9**

4.1 Polynomial model 9

4.2 Subtour elimination model 10

4.3 Column generation model 12

4.3.1 Initial routes 13

4.3.2 Subproblem 13

4.3.3 Branch and Price 15

5 Results **17**

Chapter 1

Project description

The aim of this project is to address a delivery problem for a meal delivery company that operates across multiple cities and regions. The company has several distribution centers, and its first challenge is to forecast the meals that each center will need to produce for the upcoming week, based on historical data of meal purchases aggregated by week. Once the meals are forecasted, the company's next crucial task is to distribute the necessary ingredients from a central depot to the distribution centers. The primary objective of this task is to minimize the distance traveled during transportation. Overall, the project aims to streamline the meal delivery company's operations by forecasting meal requirements accurately and optimizing ingredient distribution, ultimately enhancing customer satisfaction and improving business efficiency.

The company demand data was provided by Edwin U Kannanaikkal [4] on Kaggle with a DbCLv1.0 license. To solve the forecasting task the project implements two metrics, one is a simple average of N previous weeks projected in the future, the other is a more complex integration of the popular library for forecasting, prophet [3]. Additional data not provided in the original files, like meal ingredients and facility locations, have been generated using a mix of statistics and machine learning to be as plausible as possible. Finally, different solvers for the VRPC problem have been tried, a polynomial one, one with subtour-elimination, and the last one implemented by column generation and a custom branch and price algorithm.

The rest of this report is divided as follows: Chapter 2 explains our tactics for forecasting meal demands, Chapter 3 describes how we generated the data that was not included in our data source, Chapter 4 presents our implementation of the VRPC solver, Chapter 5 confronts the achieved results of the project.

Chapter 2

Meal Forecasting

Based on the historical data of meal purchases aggregated by week, spanning 145 weeks for each center, the initial objective of this project is to forecast the quantity of each food item that every distribution center will require. This prediction will involve analyzing the demand trends for each food item across the different centers, and identifying any patterns or seasonality in the data. The accurate forecasting of meal orders for each center will enable the company to efficiently plan and allocate resources, ensuring that they can meet the anticipated demand and minimize waste.

2.1 Details

The first reference metric is to take the average of N previous weeks and project it into the future, this is both easy to implement and quite accurate.

The second approach involves utilizing the widely used library, Prophet [3], to capture the seasonality and trends in the demand data and project them into the future. In order to simplify the problem, we did not consider other additional data such as pricing, product features, or promotions. Instead, we solely focused on the number of orders per week. While this simplified approach negatively impacted the performance and precision of the model, we decided that it was the best option given the project's time constraints.

Given the high quantity of meal-facility pairs to analyze, the project also implements

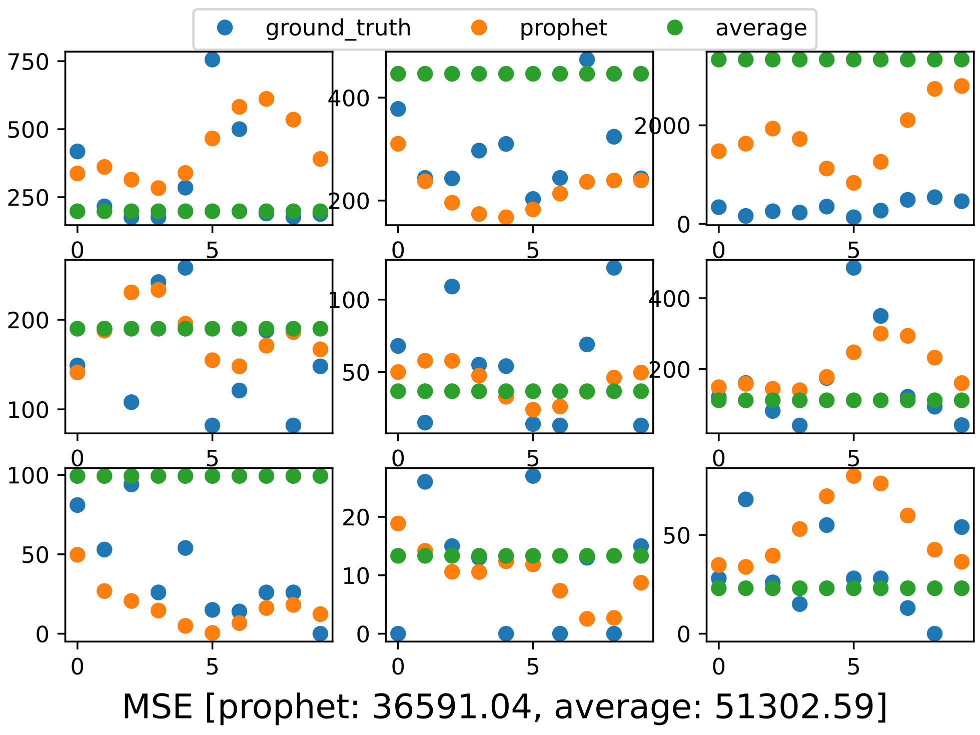

Figure 2.1: Meal Forecast results, 10 weeks

a multi-processing optimization that splits the computation load across the present CPU cores.

2.2 Results

To compare the results we will divide the data between training and testing, after that we compare the Mean Square Error of the whole dataset for the various strategies. To offer a better interpretation, 9 randomly selected meal-facility pairs have been plotted.

The only hyper-parameter involved in the first strategy is the number of weeks to average the demand data. After conducting several manual trials, we optimized this parameter to three weeks.

As demonstrated in Figure 2.1, the Prophet forecasting strategy outperforms the simple average approach in terms of accuracy, even without the use of auxiliary data. The Prophet model is able to capture the underlying trends and patterns in the demand

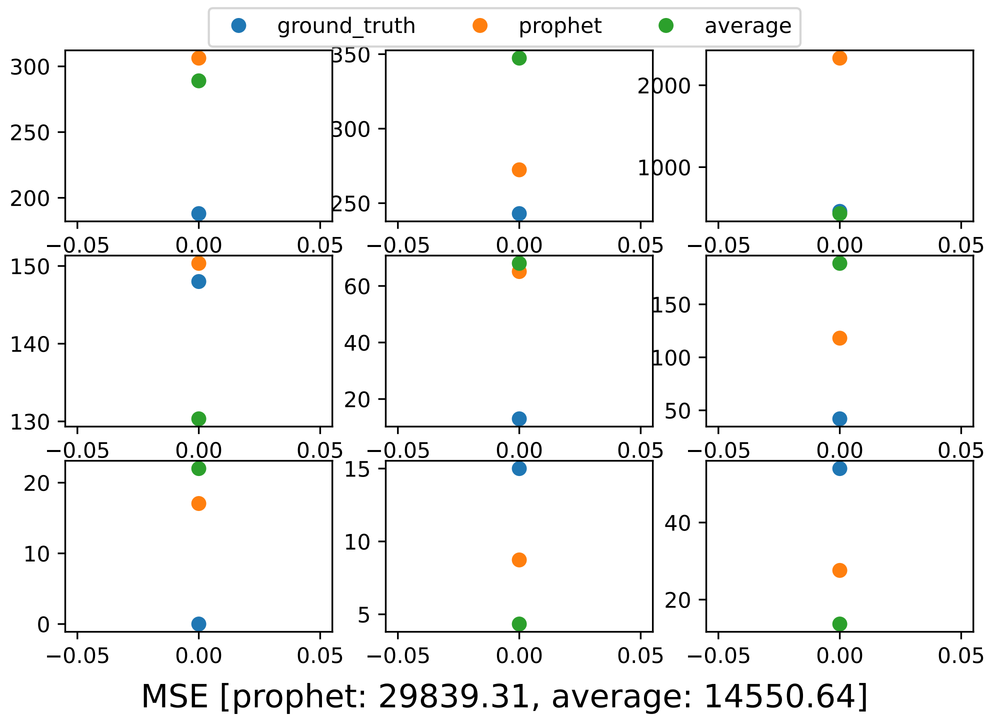

Figure 2.2: Meal Forecast results, 1 week

data more effectively, and provide more accurate forecasts for future weeks. However, this advantage is not as significant when it comes to single-week forecasting.

However, as depicted in Figure 2.2, when forecasting the outcome for a single week in advance, the average solution appears to provide better results. Although this could be attributed to inadequate utilization of the library or the presence of data outliers, we opted to proceed with the data that produced lower error since that part falls out of the project scope.

Chapter 3

Data Generation

The obtained data solely included information about meal orders, without any additional details about the distribution centers' locations or the ingredients incorporated in each meal. To effectively model the VRPC step, additional data was required. To address this data gap, both machine learning and statistical methodologies were utilized to generate a suitable dataset. The generation process utilized the hints provided in the original dataset to create data that was as plausible as possible.

Section 3.1 outlines the methodology utilized to generate a realistic ingredient list for each meal, Section 3.2 details the approach employed to randomly generate distribution center locations.

3.1 Ingredient composition

In order to generate the ingredient composition for each meal, we employed the use of ChatGPT, a sophisticated language model that is capable of generating text based on a given prompt. We sent the name for each meal and we asked how many grams of each ingredients are required, in order to ensure the accuracy and plausibility of the generated data, we implemented a human supervision step.

While the generated data did contain some errors, expecially in the weight of the ingredients, the supervisor could easily find and correct these, producing an high quality, mostly plausible ingredient list directly in CSV format. The following paste will contain an example of the generated data:

```
meal_id,ingredient,weight_grams
Pasta al Pesto,Pasta,100
Pasta al Pesto,Basil Pesto,50
Pasta al Pesto,Parmesan Cheese,20
Pasta al Pesto,Pine Nuts,20
Pasta al Pesto,Garlic,10
Chicken Rice Bowl,Chicken,200
Chicken Rice Bowl,White Rice,150
Chicken Rice Bowl,Red Bell Pepper,50
Chicken Rice Bowl,Cucumber,50
Chicken Rice Bowl,Carrot,50
Chicken Rice Bowl,Onion,20
Chicken Rice Bowl,Soy Sauce,15
Chicken Rice Bowl,Honey,10
```

3.2 Distribution center locations

The original data provides only partial information about the location of each facility, specifying only the city and region through anonymized IDs. Despite this limitation, we use the available data to generate plausible location positions for the centers. Instead of generating random positions for the centers directly, we first generate regions, then cities, and finally centers. This hierarchical approach enables us to use the known city and region information to inform the positions of the centers by relative positioning.

To ensure that the generated locations are feasible, we have implemented two additional constraints. Firstly, we ensure that the regions, cities, and centers are a minimum distance apart. If this criterion is not met, the positions are regenerated until the minimum distance is achieved. Secondly, we sample the distribution of centers from a normal distribution, with the variance varying based on the size of the city they are located in. This means that larger cities have more sparse facilities, while smaller cities have facilities located closer to their center.

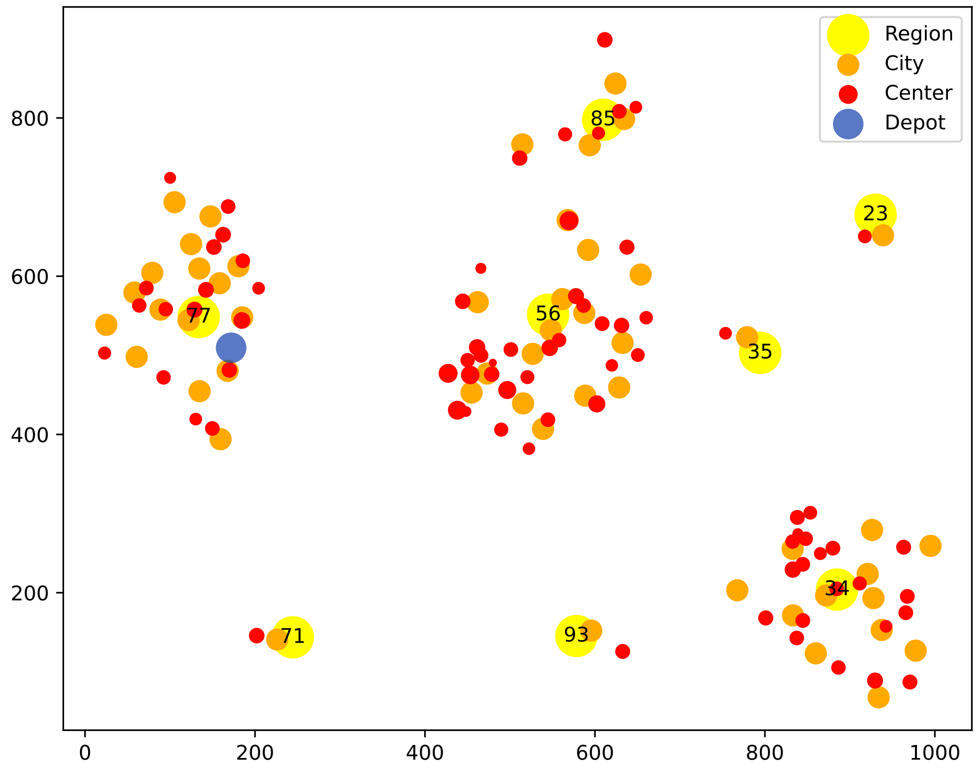

Figure 3.1: Locations of the centers in a square 2D grid

The central depot is located using a similar method to that used for the centers. However, instead of choosing a specific city for the depot, we randomly select a city from among the available options, with a bias towards larger cities. This ensures that the central depot is strategically placed to enable efficient distribution to the centers located throughout the region.

The resulting distribution can be visualized in Figure 3.1, depicting both regions, cities, centers and the central Depot.

Chapter 4

Vehicle Routing

The final phase of our process is to determine the most efficient route for delivering the ingredients from the central depot to the delivery centers. We have modeled this problem as a Vehicle Routing Problem with Capacity constraints (VRPC). Given that there are 77 facilities, developing an optimal solution is a challenging task. As a result, we have explored three different solutions to address this issue. These include a polynomial approach, a method with subtour elimination, and a third approach that uses column generation and a custom branch and price model.

Of particular note is that while most formulations are independent of the precision of the numbers, the column generation model has a subproblem of cost $\mathbb{O}(Q^2N^2)$ where Q is the maximum capacity of the vehicles, thus, to have faster subproblem iterations the granularity of the vehicle capacity has been reduced, a similar reason is also used for the granularity of the distance. To offer better comparison between the different methodologies, the same granularities have been applied to the polynomial and subtour-elimination approaches.

4.1 Polynomial model

The first method that we explored is a polynomial approach, which was the most straightforward solution. Despite its simplicity, this method is effective at identifying a feasible route for delivering ingredients to the various delivery centers.

$$
\begin{aligned}
\min \quad & \sum_{i,j} d_{i,j} x_{i,j} \\
\text{s.t.} \quad & \sum_{i \neq j} x_{i,j} = 1 \quad \forall j \in V && (1)\\
& \sum_{j \neq i} x_{i,j} = 1 \quad \forall i \in V && (2)\\
& \sum_{i \neq 0} x_{0,i} - x_{i,0} = 0 && (3)\\
& c_0 = 0 && (4)\\
& c_j - c_i \geq w_j - M(1 - x_{i,j}) \quad \forall (i,j) \in A | i \neq j \wedge j \neq 0 && (5)\\
& c_i \leq Q \quad \forall i \in V && (6)\\
& x_{i,j} \in \{0,1\} \\
& c_i \in \mathbb{N}
\end{aligned}
$$

Where w_i is the material need of node i, $d_{i,j}$ is the distance from node i to node j, V is the set of nodes and A is the set of arcs. The variables are: $x_{i,j}$ that is 1 if and only if the arc from node i to node j is selected, and c_i that contains the accumulated capacity at node i. The model minimizes the total distance, constraints 1, 2 and 3 are the flow constraints, while inequality 4, 5 and 6 constrain the capacity of the vehicles.

Constriant 5 activates when the arc (i,j) is selected and constrains the capacity of the new node, as the inequality can be rewritten as $c_j >= c_i + w_j$. When node capacities are positive this inequality also imposes cycle elimination as c_i must be strictly monotonic along the path.

4.2 Subtour elimination model

The second tested model is based on subtour elimination, also known as row generation. The general idea is that the model has a polinomial number of variables but an exponential number of constraints that are added incrementally during the exploration of the solution space. Usually the solution does not need to generate too many constraints, rendering this methodology usually faster than a polynomial formulation.

$$
\begin{aligned}
\min \quad & \sum_{i,j\in A} d_{i,j} \sum_{k\in K} x_{i,j,k} \\
\text{s.t.} \quad & x_{i,i,k} = 0 \quad \forall i \in V, k \in K && (1)\\
& x_{i,0,k} = 0 \quad \forall i \in V, k \in K && (2)\\
& x_{n+1,i,k} = 0 \quad \forall i \in V, k \in K && (3)\\
& \sum_{j\in V, k\in K} x_{i,j,k} = 1 \quad \forall i \in C && (4)\\
& \sum_{i\in C} d_i \sum_{j\in V} x_{i,j,k} <= Q \quad \forall k \in K && (5)\\
& \sum_{j\in V} x_{0,j,k} = 1 \quad \forall k \in K && (6)\\
& \sum_{i\in V} x_{i,n+1,k} = 1 \quad \forall k \in K && (7)\\
& \sum_{i\in V} x_{i,h,k} = \sum_{k\in V} x_{h,j,k} \quad \forall h \in C, k \in K && (8)\\
& \sum_{(i,j)\in A} x_{i,j,k} \le |S| - 1 \quad \forall S \subset V, k \in K && (9)\\
& x_{i,j,k} \in \{0, 1\}
\end{aligned}
$$

Here the nodes are a bit different than the previous model, node 0 and $n+1$ are the depot node, and while C is the set of customers ($n = |C|$), V is the set of all nodes ($V = C \cup \{0, n+1\}$). The other used set is K the set of usable vehicles, while Q, as in the other models is the maximum capacity for each vehicle. In this formulation we have only one three-dimensional variable, $x_{i,j,k}$ that is 1 if and only if vehicle k uses the arc (i, j).

The first three ineqaulities are just to remove unwanted arcs, 1 is to avoid 1-cycles, 2 and 3 remove cycles in and out of source and sink nodes. Constraint 4 allows each customer to be visited and cosntraint 5 limits the capacity for each vehicle. Flow constraints are enforced through ineqaulities 6 to 8.

The last constraint purpose is to remove cycles in each tour, but needs to be enforced for each possible cycle. To render this efficient we relax the problem and only add the constraints encountered when the problem solution contains cycles.

We adopted two additional details to increase the model performance: the first optimization is to use lazy constraints to speed up the optimization. The second one is less obvious, when a cycle is found we do not only remove that cycle but we remove the cycle also with the other k (the other vehicles), this speeds up by a lot the convergence, otherwise the algorithm would try the same cycle with another vehicle.

4.3 Column generation model

Column generation is a more complex approach, as such we have implemented the method described by Desrochers, Desrosiers and Solomon [2] simplified to remove time constraints and with a custom subproblem solution.

$$
\begin{aligned}
\min \quad & \sum_{r \in R} c_r x_r \\
\text{s.t.} \quad & \sum_{r \in R} a_{i,r} x_r \geq 1 \quad \forall i \in C && (1)\\
& \sum_{r \in R} x_r = X_d && (2)\\
& \sum_{r \in R} c_r x_r = X_c && (3)\\
& x_r \in \{0,1\} \\
& X_d, X_c \geq 0, integer.
\end{aligned}
$$

C is the set of customers while R is the set of all possible routes, in this model the R set grows exponentially with the number of clients.

The variable x_r is 1 iff the route is used, X_d is the number of vehicles used and X_c is the total distance traveled. Parameter c_r is the total cost of route r and $a_{i,r}$ counts how many times customer i is visited by route r. The system imposes integrality on the distance traveled, as such, the weights of the arcs should be integral too. To maintain precision while performing distance integralization, we defined a distance granularity parameter.

Constraint 1 requires all customers to be statisfied, while constraints 2 and 3 constrain the variables X_c and X_d to adhere to their definition, these are useful in the Branch and Price phase.

To resolve this problem without using an exponential number of variables, the system is first relaxed of its integrality constraints, and is then initialized with a feasible set of variables. After that the linear problem is solved and the dual variables are used to find better routes to add as variabels. This process is repeated until an optimal solution for the relaxed problem is found. To then find the integral solution another method called Branch and Price is required.

4.3.1 Initial routes

There are various strategies to select initial routes, one possible solution is to select the Identity matrix, interpreted as using one vehicle to visit each customer, while this does work it leads to inefficient first steps and additional subproblem cycles. In this implementation the initial routes are selected with a custom heuristic that explores the closest nodes until the vehicle is filled, the pseudocode is written in Algorithm 1

4.3.2 Subproblem

The used subproblem is a variation of what was described in the original paper, we revisited it to remove the time constraints and to be more intuitive.

Given the reduced costs of each arc: $\overline{c}_{i,j} = (1 - \pi_c)c_{i,j} - \pi_i$ where $\pi_0 = \pi_d$, the subproblem requires to find negative cost routes (from the depot back to the depot) with capacity constraints traversing the graph. The graph can also contain negative loop cycles, making the problem NP hard in the strong sense.

Our solution is a simple Dynamic Programming algorithm similar to the Pulling

Algorithm 1 Algorithm to find an initial solution

```
$routes \leftarrow \{\}$
$visited \leftarrow \{\}$
$current\_node \leftarrow 0$
$current\_route \leftarrow \{\}$
$current\_capacity \leftarrow 0$
**while** there are still nodes to visit **do**
    $n \leftarrow closest\_unexplored\_node(current\_node)$
    Add n to visited
    **if** $current\_capacity + m_n \geq Q$ **then**
        Close path and add it to routes
    **end if**
    Add $n$ to $current\_route$
    $current\_node \leftarrow n$
    $current\_capacity \leftarrow current\_capacity + m_n$
**end while**
Close the last path and add it to routes
```

Algorithm described in the paper, of complexity $\mathbb{O}(Q^2N^2)$

$$C_0(0) = 0$$
$$C_j(q) = \min_{i \in V}\{G_i(q\prime) + \bar{c}_{i,j} | q\prime + q_j \leq Q\}$$

This computes the minimal paths for each of the nodes, available in $C_i(Q) \forall i \in V$.

In addition to this the implementor will want to keep a predecessor matrix to keep track of the real computed route. This also helps in determining the 2-cycle elimination algorithm, the idea is to keep track of the best two routes for each state, one that is the best route, and the second one that is the best route using a different predecessor node.

$$C_0(0) = 0$$

$$P_j(q) = arg\, min_{i \in V}\{[C_i(q') + \overline{c}_{i,j}\text{if}j \neq P_i(q), C'_i(q') + \overline{c}_{i,j}] | q' + q_j \leq Q\}$$

$$C_j(q) = C_{P_j(q)} + \overline{c}_{i,P_j(q)}$$

$$C'_j(q) = \min_{i \in V}\{[C_i(q') + \overline{c}_{i,j}\text{if}j \neq P_i(q), C'_i(q') + \overline{c}_{i,j}] | q' + q_j \leq Q \wedge i \neq P_j(q)\}$$

A simple interpretation of this algorithm is that $C_i(q)$ offers the best route to arrive at node i with node capacity q, $C'_i(q)$ offers the second best route and $P_i(q)$ contains the predecessor node to obtain $C_i(q)$. In a real algorithm $P'_i(q)$ should also be computed to reconstruct the real paths. $C_i(q)$ selects the best paths using capacities less than $q - q_j$, also checking that the path doesn't to 2-cycles. By iterating the problem with every node at every capacity we obtain the minimal route to arrive at each node.

The subproblem is a critical component of the algorithm as it is where the model will spend most of the time, to have faster iterations we implemented the subproblem using Cython [1], while all of the other project is developed using Python. This alone has achieved a 10x speedup for the solution of every subproblem.

Since the complexity of the subproblem higly depends on Q, the maximum capacity for each vehicle, as a speed optimization we offer a capacity granularity, by having less granularity the results are less precise but the subproblem speed decreases quadratically. All of the results have a capacity granularity of 1000 Kg per customer where $Q = 10000Kg$.

4.3.3 Branch and Price

The Branch and Price algorithm is the same as described in the original algorithm, we implemented some optimizations to deduplicate paths and other minor implementation details. The exploration direction in the algorithm is parametrized but by default a mixed exploration is selected as it quickly arrives at an integral solution and it keeps the number of unexplored nodes comparatively low.

Developing a Branch and Price technique is quite error-prone for a student, as such we integrated a debug mode that enables exploration of the Branch and Price tree after the optimal solution has been found, keeping track of the Gurobi model and added constraints. This online debugging strategy has been a helpful way to locate and remove multiple coding mistakes.

Chapter 5

Results

While the problem istance is too big to find an optimal solution, the models have been given ample time to compute feasible qualitative routes. Unexpectedly, the subtour elimination model is less efficient than the polinomial model, even in smaller instances. Even more unexpectedly, the polinomial formulation seems to be more efficient than the custom Column Generational approach, finding better solutions in orders of magnitude less time.

We have run the two models on two datasets each: one for the forecasted data and the other for the real data. The polynomial approaches have been time-limited by the memory of the calculator as they seem to consume far more than what the running computer had to offer, thus their time will be lower than those for column generation. The results are displayed in Figure 5.1, Figure 5.2, Figure 5.3 and Figure 5.4.

As can be observed, the polynomial model consistently yields superior results in significantly less time compared to our Branch and Price method. This observation is a testament to the impressive work done by the Gurobi team in developing Mixed Integer Programming (MIP) solving solutions that can efficiently handle even the most straightforward problem formulations.

It is worth noting, however, that we may have achieved even better results had we implemented more efficient subproblem formulations. With the benefit of hindsight, we recognize that given the number of clients required, a metaheuristic solution might have been a more appropriate approach. Nonetheless, the results obtained through

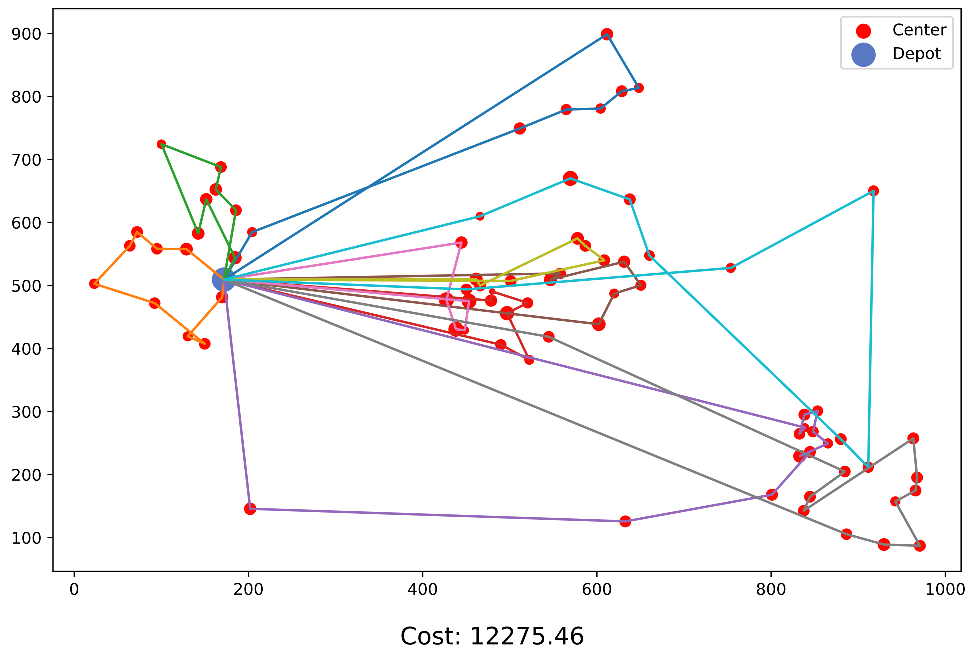

Figure 5.1: method: column generation, data: average, runtime: 11h

our selected approach provide valuable insights into the problem and pave the way for further optimization efforts in the future.

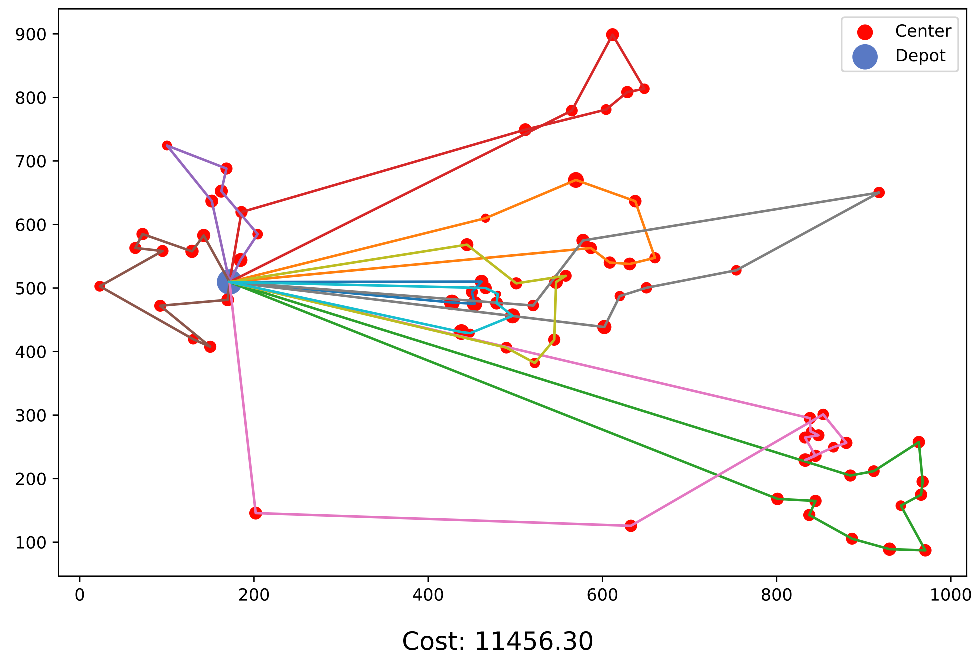

Figure 5.2: method: column generation, data: real, runtime: 22h

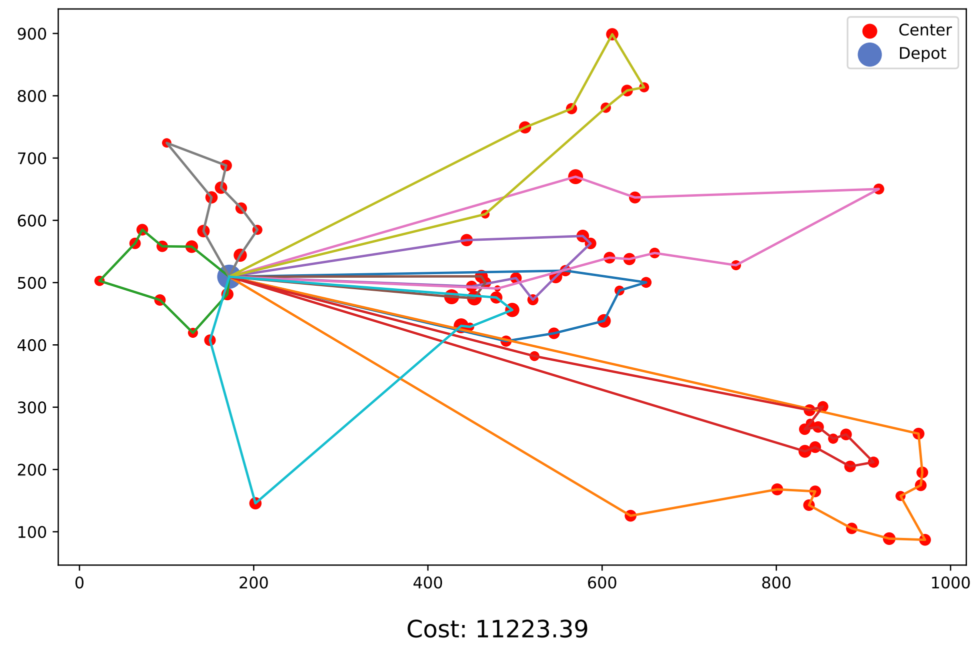

Figure 5.3: method: polynomial, data: average, runtime: 7h

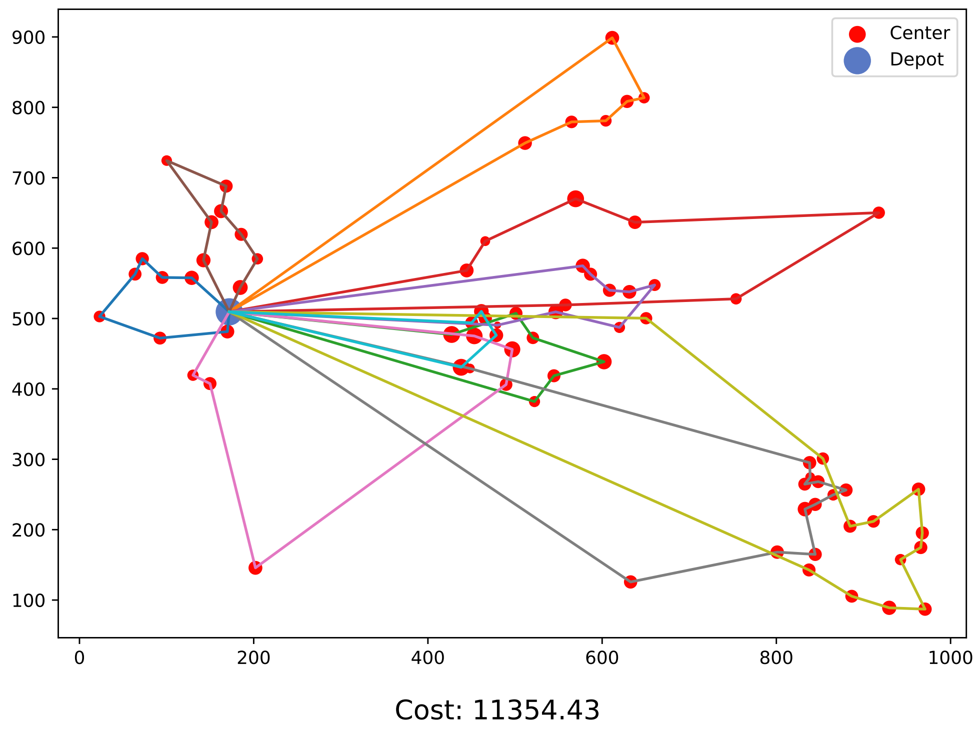

Figure 5.4: method: polynomial, data: real, runtime: 5h

Bibliography

[1] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39, 2011.

[2] Martin Desrochers, Jacques Desrosiers, and Marius Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations research*, 40(2):342–354, 1992.

[3] Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[4] Analytics Vidhya. Food demand forecasting, 2020.