Report of ISyE 603 Final Projects on Two-Stage Stochastic TSP with Time Window and Random Arc Wrights Bread Delivery Scenario

ISyE 603: Advanced Optimization Modeling

May 6th, 2025

Team Members:

Eddy Yeung

Ajeenckya Mahadik

Problem Description

This project addresses the challenge of optimizing a bread delivery system in a small town, with the aim of ensuring timely and cost-effective delivery to customers regarding real-world uncertainties. The delivery process originates from a central bakery shop and move to multiple customer locations, each with specific delivery time windows between 3:00 PM and 7:00 PM. The route times from a customer location to another customer location change variously due to real-world event: traffic jam, road construction, car accident, and many more. A penalty is applied if a delivery is early or late for the customer's designated time window.

The core problem is approached in two stages:

1. **First Stage – Planned Route Optimization**:
   The initial stage models the delivery as a Traveling Salesman Problem (TSP), identifying the most efficient fixed route that visits all customer locations exactly once. This route represents the ideal delivery path under normal conditions, without any disruptions.

2. **Second Stage – Route Adjustment to Real World Events**:
   Once delivery begins, the model enters a stochastic decision-making phase where real-world disruptions—such as traffic congestion, road closures, or other unforeseen delays—may occur. In response, the model decides whether the delivery vehicle should wait for the disruption to clear reroute to another customer location. These adjustments are made to maintain delivery efficiency, minimize penalties, and ensure that all orders are successfully completed within the allowed time window.

By combining deterministic planning with stochastic adaptability, the model aims to optimize both the reliability and cost-effectiveness of the delivery process under uncertainty.

Model and Solution Approaches

Assumptions

There are several assumptions within this model, reducing the size of the model to a controllable size. The first assumption is customer will neither cancel nor move their delivery window throughout the entire delivery. Also, there won't be an increment of customer orders during delivery. The customer demand is viewed as satisfied once the driver arrives at the customer location, and the time for delivering the bread on foot to the customer house is ignored. The truck has enough capacity to satisfy all the customers in one trip. The last assumption is the roads between the customer location are always the direct routes.

Solution Approach

To address the challenges of delivery routing under uncertainty, we developed a two-stage optimization model tailored to a bread delivery system operating within a small town. The model integrates both deterministic and stochastic elements to ensure timely, cost-effective deliveries even in the presence of disruptions.

First-Stage Model: Planned Delivery Route

In the first stage model, each customer is located at a known point on a map, and the delivery cost is based on the time it takes to travel between these points. Travel time is calculated using grid distance, with the assumption that the delivery vehicle travels a fixed distance per time unit. Using this information, the model computes the travel time between every pair of customer locations. It then selects the most efficient route that starts and ends at the bakery, visits each customer exactly once, and minimizes the total travel cost. The route must be continuous, meaning the vehicle cannot return to the same location or skip any customers.

In this stage, we focus on planning the delivery route under normal conditions, before any disruptions happen. The problem is modeled as a Traveling Salesman Problem (TSP), where the goal is to find the shortest and most efficient path that starts at the bakery and visits every customer exactly once.

Set and Parameter

The bakery dataset provides the spatial coordinates of customers along with their computed distance from the central bakery location. Each customer is identified by a unique label and their (X, Y) position. The dataset enables calculation of travel times and routing paths between the bakery and delivery points. These distances are essential for evaluating route efficiency and cost in the delivery optimization model.

Decision Variable

Traveling Salesman Problem (TSP) using binary decision variables `x_ij`, which indicate whether the route goes directly from location `i` to location `j`.

Objective

The objective is to minimize the total travel cost, computed as the product of the fixed cost per time unit.

Constraints

The first constraint ensures that from each customer location `i`, the vehicle departs to exactly one other location, enforcing that each customer is visited once. The second constraint ensures that each customer location `j` is entered exactly once, which complements the first constraint to ensure a complete tour. The third constraint guarantees that the route begins by leaving the bakery and going to exactly one customer location, establishing the starting point of the route. We also applied the delayed constraint generation method as used in HW2 to solve the subtour elimination.

Second-Stage Model: Planning Route with Time Window, Random Demand, and Driver Waiting

The second-stage model extends the initial routing problem by introducing adjustments to simulate uncertainties and customer delivery constraints in the real world. Unlike the first stage, the second-stage model includes scenario elements to simulate potential disruptions or scheduling complexities. The model includes constraints that manage customer location entry and exit flows, enforce delivery time windows, and prevent sub-tours. This approach allows the evaluation of multiple realistic scenarios, optimizing the driver decisions in response to varying customer availability and travel conditions.

Set and Parameter

Besides inheriting similar data set from the first stage such as customer location, the second stage model includes a time element. The customer availability data was structured to reflect realistic delivery windows across a 4-hour day divided into 10-minute intervals. Each customer is assigned exactly one time window of 5 consecutive periods, randomly positioned to simulate varying service preferences. The start of the window avoids the very first and last time slots to allow for practical routing flexibility. These windows may overlap between customers, creating diverse scheduling scenarios.

For the scenario, we decided to start with two scenarios and gradually increase to ten scenarios. Considering the average driver's wage in Wisconsin to be $31/hour, the truck fuel cost is $8/hour, and truck maintenance and insurance-related costs are roughly $10/hour, totaling $49/hour. To cover other unexpected costs, we have set the travel cost to be $10 per period.

To introduce variability in travel times across scenarios, we implemented a procedure that perturbs arc travel durations randomly. For each scenario, a new dictionary is created to store modified travel times between customer pairs. A subset of arc indices is randomly selected to remain unchanged, preserving their base travel times. For the remaining arcs, a random value scaled by a predefined maximum increase is added to simulate delays or disruptions.

Decision Variables

The optimized x_ij values are stored as xvals variable from the first stage model and used as a reference route in the second stage. We have included several new decision variables in the model. v_ijts is a binary variable representing the driver's decision to go from customer location i to customer location j at time t in scenario s. y_jts is a binary variable indicating whether a delivery for customer j is successful at time t in scenario s. z_jts is a binary variable that indicates whether a delivery for customer j is unsuccessful at time t in scenario s. w_jts is a binary variable that shows the driver's decision to take a break at customer location j at time t in scenario s. r_ij is a binary variable showing customer location i before customer location j.

Objective Function

$$\min \frac{1}{|S|} \sum_{s \in S} \left(\sum_{t \in T} \sum_{i \in L} \sum_{j \in L: i \neq j} C * d_{ij}^{s} * v_{ijt}^{s} + \left(\sum_{j \in L: j \neq \text{"Bakery"}} \sum_{t \in T: t < P_j[0]} \lambda_{jt} * y_{jt}^{s} \right) + \left(\sum_{j \in L: j \neq \text{"Bakery"}} \sum_{t \in T: t > P_j[2]} \lambda_{jt} * z_{jt}^{s} \right) \right)$$

Figure 1: The Objective Function

The objective function (Figure 1) is first summed over each scenario and divided by the cardinality of scenarios to get the average value. The first summation is period t and customer location i and j. This calculates the transportation cost by multiplying the driver's decision on a route with the time it takes to travel the route (d_ijs) and multiplying the cost per hour (C). The second summation is to penalty for early delivery. Any successful deliveries before a customer j's delivery window (t<P_j[0]) are considered as early delivery, so we give a penalty. On the other hand, any not successful deliveries after a customer j's delivery window (t>P_j[2]) are considered as late delivery, so we give a penalty.

Constraints

More constraints are involved in the second stage model, but a few similar constraints exist. The first two constraints, Leave and Enter, are the two constraints that limit the inflow and outflow of a node to be more than one. Using more than and equal to sign relaxes the model, allowing the model to visit a node again if necessary. The third constraint, StartPt, ensures that the driver will leave the bakery to another customer location by the beginning of the period. The fourth constraint, SuccessOnce, is to limit the number of successful deliveries in the customer time window to less than or equal to 2. In summation with t, we limit the t to be each customer j's time window. To have the number of successful deliveries less than or equal to two is to relax the constraint further. The fifth constraint, Successful, ensures the delivery succeeds once the driver decides to go to the customer's location j. We have included –1e-3 to have the flexibility of the constraint. The sixth constraint, SucessOrNot, is ensuring that a customer's delivery status can only be successful or not.

```
@constraint(ef, DriverBalance[j in CustLoc, s in Scen, t in period; t >= M - 1],  # M is the maximum possible time distance travel
        sum(v[i,j,t - TDist_Sec[s][(i,j)], s] for i in CustLoc if i != j) + w[j, t - TDist_Sec[s][(i,j)], s]
        == sum(v[j,k,t,s] for k in CustLoc if k != j) + w[j,t,s])
```

Figure 2: The Original Seventh Constraint

In the beginning version of the seventh constraint (Figure 2), we notice a bug that the i in the w[j, t – TDist_Sec[s][(i,j)], s], where TDist_Sec is the time distance traveling between customer i and j, are not in the iterator of the constraint. We cannot remove the summation and move the i to the iterator section of the constraint because it will over-constrain the system.

```
# Enforce travel time between arrivals and departures, allow waiting, disallow subtours
for j in CustLoc, s in Scen, t in period
    for i in CustLoc
        if i != j && haskey(TDist_Sec[s], (i,j))
            travel_time = round(Int, TDist_Sec[s][(i,j)])
            t_prev = t - travel_time
            if t_prev in period
                @constraint(ef,
                    v[i,j,t_prev,s] + w[j,t_prev,s] >= sum(v[j,k,t,s] for k in CustLoc if k != j) + w[j,t,s] - 1
                )
            end
        end
    end
end
```

Figure 3: The Fixed Seventh Constraint

We used a for-loop to do the iteration section (Figure 3). The if-statements ensure that i is not equal to j, (i,j) route is in the traveling route, and t – traveling time is still in the period. The constraint is slightly changed due to the over-constraint. The summation allows the model to choose more than one route to k, but the model will only choose one route due to minimizing cost objective function. Subtracting one on the RHS of the constraint gives the model more flexibility while guaranteeing the driver is at the t – traveling time. By having RHS be – 1, the driver takes no action, so the driver is in a state of waiting. This RHS also allows the LHS to be zero, meaning the driver did not choose a route or wait before the period t. A similar LHS situation also occurs when RHS is 0. It means the driver at period – traveling time can choose to not to do anything, while the driver at period t can take an action to either drive to a new location or stay at period t. However, this scenario is less likely to happen due to the first two constraints of the model. At the same time, RHS cannot be one because of the eighth constraint. The eighth constraint, OneStatus, is to ensure the sum of all the arcs coming out from customer location i plus the waiting in customer location i is no more than 1. This stops the constraint seventh RHS from being 1.

The ninth constraint is a subtour elimination constraint. To run this constraint, we create a new continuous decision variable u with a lower bound of 0 and an upper bound of the size of all the customers. The decision variable u is used for tracking the visiting position of customer i in scenarios. For the constraint, the u_is subtracts u_js to ensure the i order is before j. If this order is valid, we will have a negative value, and this negative value will

multiply with a big M, in this case, the size of all customers. The big M is controlled by the arc between customer locations i and j. The LHS is big M – 1 to ensure there will not be any formulation of subtours. If the u_is is always before the u_js, the RHS will give a value smaller than the LHS. If not, RHS will be too big for the LHS.

The tenth constraint, FlexChange, is a reference constraint where the decision variable r_ij will try to create a flexibility for the xvals from the first stage. By having plus two on the RHS, r_ij variables can more freely choose their route under the original routes from stage 1. The eleventh constraint, FlexToV, is connecting the r_ij variable with the v_ijst variable. However, we have included the wait time to give the v_ijst variable more freedom to change.

Alternative Modeling Approaching

Linear Penalty

In the linear penalty experiment, the model applies penalties for early or late deliveries that are directly proportional to the deviation from the desired delivery window. These penalties are scaled by customer-specific demand window, ensuring that some customers are prioritized more than others as their demand windows are earlier in the timeline. The structure ensures that the model encourages timely deliveries without harshly penalizing small deviations. This setup is appropriate when moderate delivery accuracy is sufficient and computational speed is important.

Quadratic Penalty

The quadratic penalty experiment increases the penalty for delivery time deviations nonlinearly through the function of x^2, with larger deviations incurring disproportionately higher costs. This method emphasizes tight adherence to time windows, particularly for critical customers. Since quadratic terms introduce nonlinearity. This formulation is beneficial when lateness or earliness causes severe degradation in service quality, such as spoilage or customer dissatisfaction. It prioritizes service quality over solver simplicity.

High Vehicle Operation Cost

In the high vehicle operation cost scenario, transportation costs are set high to simulate budget-constrained or fuel-sensitive conditions. The model responds by minimizing travel, possibly reducing the number of active routes or consolidating deliveries. It reveals the structure of efficient routing under financial stress and challenges the system's ability to

maintain service levels under tight constraints.

Low Vehicle Operation Cost

By contrast, the low vehicle operation cost scenario reduces transportation cost weight, enabling the model to prioritize service quality over route efficiency. This leads to more flexible delivery paths and often better adherence to customer time windows. Since traveling is inexpensive, the model is more likely to explore redundant or longer routes to improve on-time performance. It helps identify where strict cost control may be compromising service outcomes.

Also, this model works for multiple time windows of the same customer.

Result

First Stage Results

In the first stage model result, the model estimates the total cost of this delivery trip to be $410. In the resulting diagram (Figure 4), we can see that the model decides to go a far distance to O first and travel around, satisfying the customer. An interesting route that the model decides is N->A->G->S (Total distance is 17.245) rather than N->G->A->S (Total distance is 14.384).

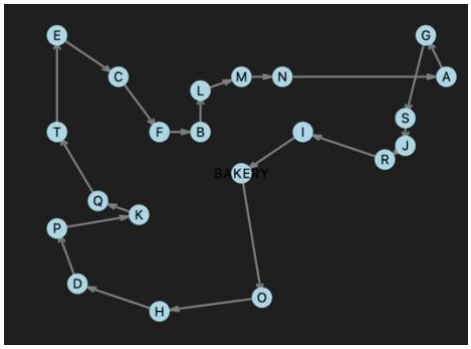

Figure 4: First Stage Model Result

Second Stage Results with the constant penalty value

When the model was run with a constant penalty, it gave the following routes: Each route was almost different for each scenario. This function's objective value is $175.5. As shown in the graph (Figure 5) below, the model mostly chooses a path from E to C, the same as the first-stage solution, and G to A, the opposite of the first-stage solution.

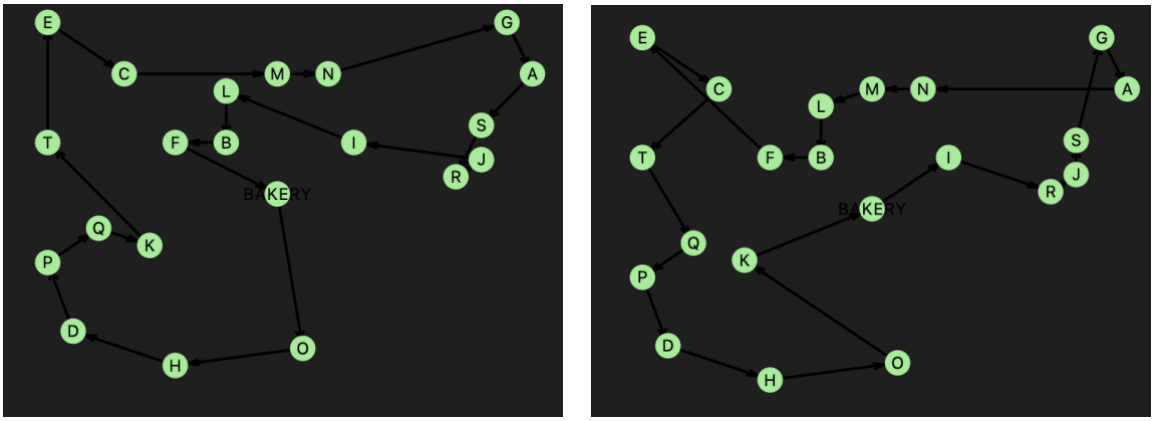

Figure 5: Second Stage Model Result

Linear Penalty Results

When the model was run with a linearly increasing penalty, it gave the following routes: each route is almost different for each scenario. This function's objective value is $349.2. As shown in the graph (Figure 6) below, the model mostly avoids the long routes; instead, it uses the shortest path in between.

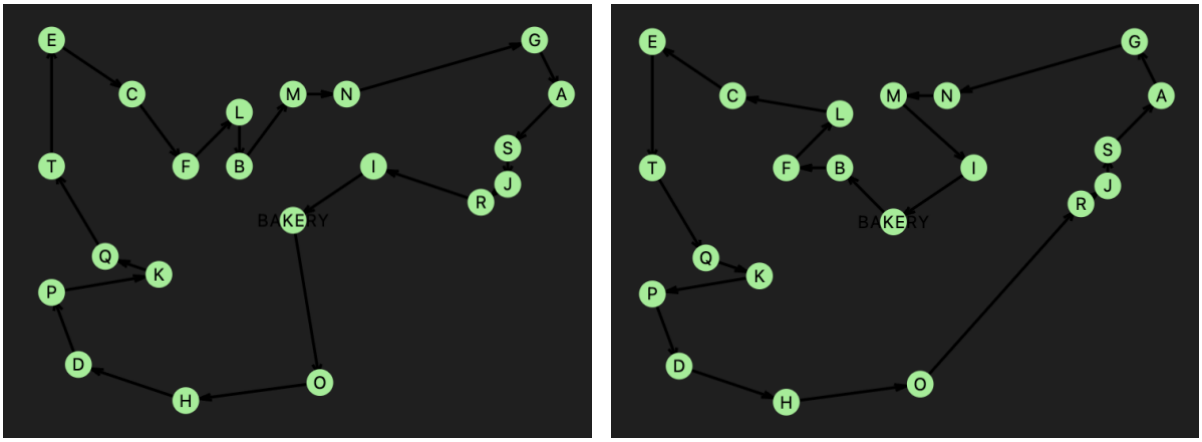

Figure 6: Linear Penalty Model Result

Quadratic Penalty Results

In the quadratic penalty model result (Figure 7), the model estimates the total cost of this delivery trip to be $371.57. In some scenarios, the quadratic penalty model might make a route decision similar to the linear penalty model.

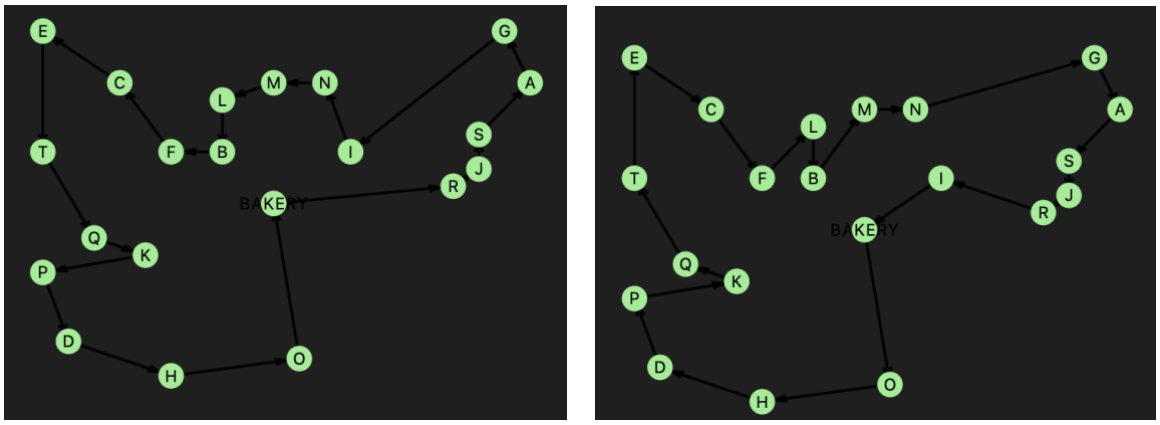

Figure 7: Quadratic Penalty Model Result

High Vehicle Operating Cost Results

The high cost of vehicle operation model (Figure 8) does not show any major changes in the route; it mostly sticks with the route as the linear penalty model. The objective value changes to $690.2. As this experiment was with a linear penalty, it won't change much.

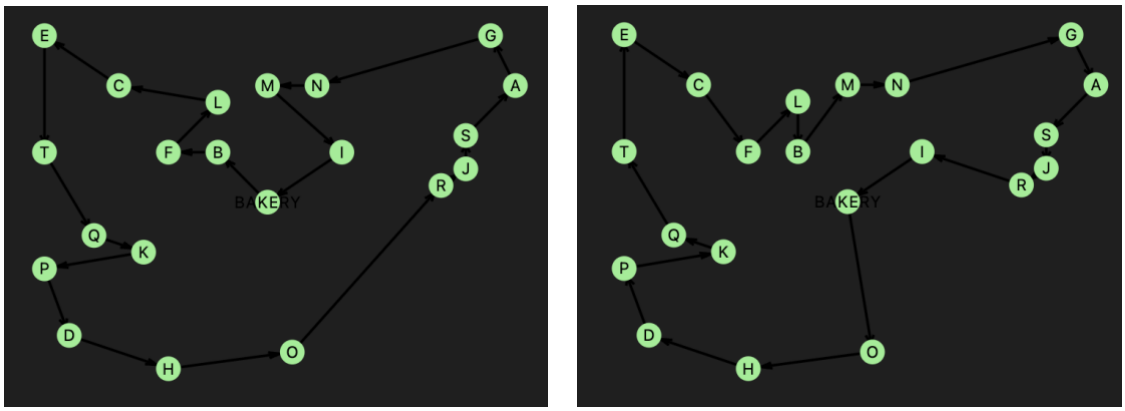

Figure 8: High Vehicle Operating Cost Model Result

Low Vehicle Operating Cost Results

The low cost of vehicle operation model (Figure 9) does not show any major changes in the route; it mostly sticks with the route as the linear penalty model. The objective value changes to $177.9. As this experiment was with a linear penalty, it won't change much.

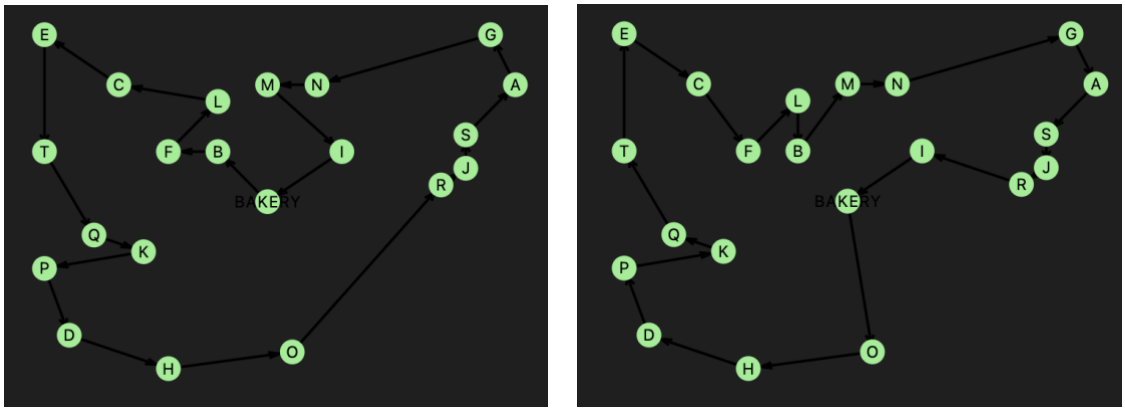

Figure 9: Low Vehicle Operating Cost Model Result

Conclusion

With our different methods of approaches, we notice that when we make any monetary-related change to the model, the models react similarly across all four different approaches. These reactions are commonly shared with trying to reach the shortest distance possible. This might be because these four parameter changes are all related to the objective function parameters, and the model takes the same approaches to solve these models. However, the change in vehicle cost does not affect much as we are still penalizing the model with a linearly increasing penalty.

So, we can conclude that the model shows similarities between the experiments, but it changes somewhat in some scenarios to avoid the higher penalties. It always takes the shortest path when it is behind time, and initially, it takes the long routes.

Next Step

Due to limited time, we cannot successfully tune our model to its optimal status. We have created a series of plans in the future to improve the model. We will continue fine-tuning the model by reducing some relaxation on the model's constraints and strengthening the inequality and RHS value. Meanwhile, we will seek opportunities to remove any unnecessary constraints that might have led to over-constraining the model. For example, we notice the eighth constraint might be over-constraining the seventh constraint, so we might remove the eighth constraint and modify the seventh constraint.

We plan to apply the delayed constraint generation to replace the MTZ method in the model, reducing the running time for the second stage model.