Optimal Package Delivery Strategy

The University of Texas at Austin

Rushil Nakkana

October 23, 2023

Business Objective

Our logistics company is tasked with improving the cost-effectiveness of numerous package deliveries per day by optimizing the distance traveled. There are 10 available trucks and a central distribution center within the city where each truck starts and ends its route at this center. Our goal is to determine the most suitable truck for each package and to optimize the distance in the delivery routes for each truck. Given the increasing complexity of the city routes, determining the truly optimal solution can become computationally infeasible. However, we can utilize two approximation algorithms, k-means clustering and simulated annealing, to provide near-optimal solutions, as well as Gurobi to provide the optimal routes and enforce constraints that prevent inefficient circular routes and ensure that the distribution center must have exactly as many connections (links) as there are trucks.

Part 1: Minimizing Total Distance Traveled

The objective is to minimize the total distance traveled from the distribution center to different delivery locations and back, using all 10 trucks to deliver 500 packages on the specified business day. Two different methodologies will be utilized to tackle this objective: clustering and 3-change simulated annealing.

1. *Clustering Approach*

This is a two-step process: First, we grouped the delivery points into clusters and solved a Traveling Salesperson Problem (TSP) for each cluster, which represents each truck. Using a **k-means clustering** approach, we were able to create 10 clusters for each of the trucks:

a. The data is initially prepared with using the ‘x’ and ‘y’ coordinates of the 500 delivery points for clustering, which are then grouped into one of 10 clusters, representing the 10 trucks.
b. A K-Means clustering model is created and fit to the data, which identify cluster centers and designates each data point to the nearest cluster.
c. The cluster labels assigned by the K-Means modeled are then stored into the original data frame as ‘cluster_label’ to be used later when filtering our data by cluster.
d. The clustering results can be visualized in **Exhibit A**, where each point is colored based on its assigned cluster label and the black marker in the middle represents the distribution center at the coordinates (0,0)

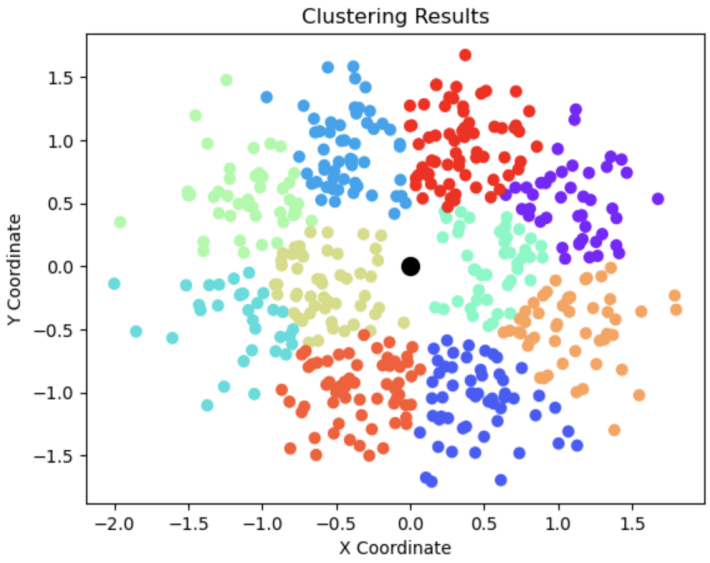
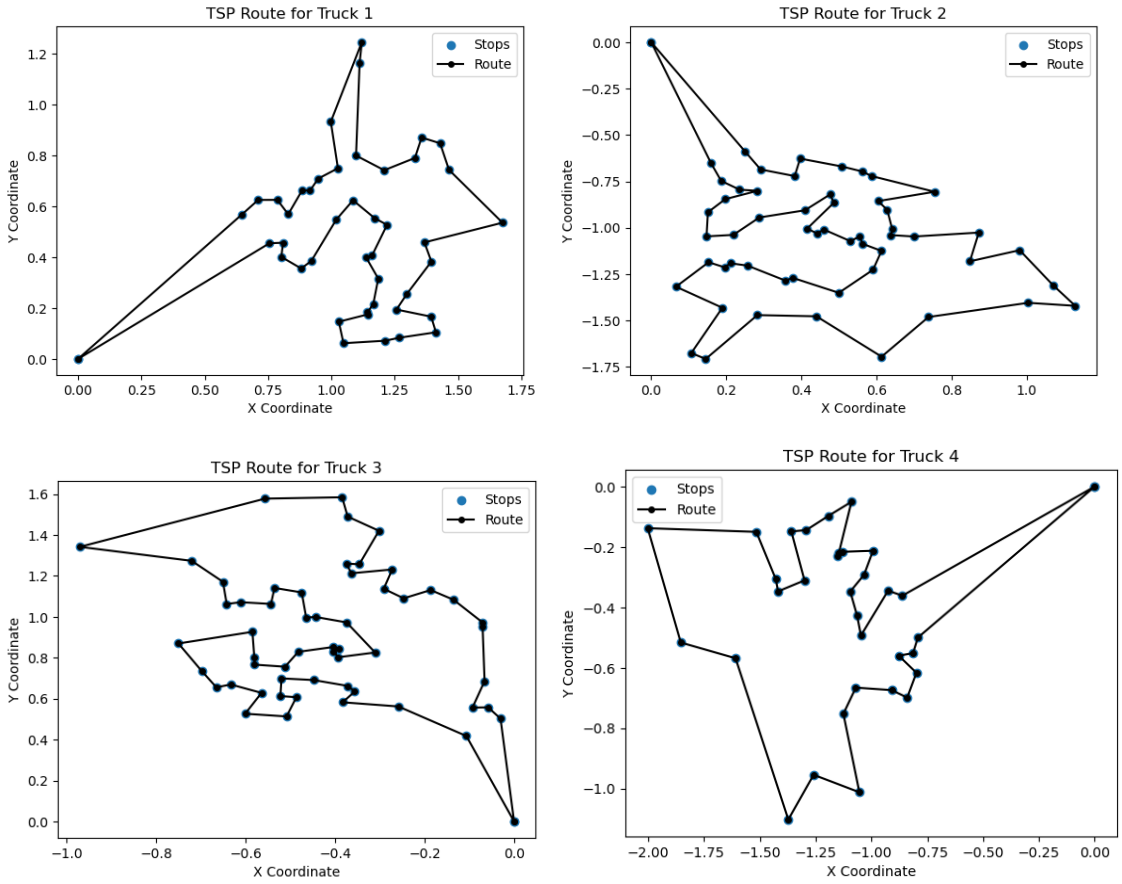
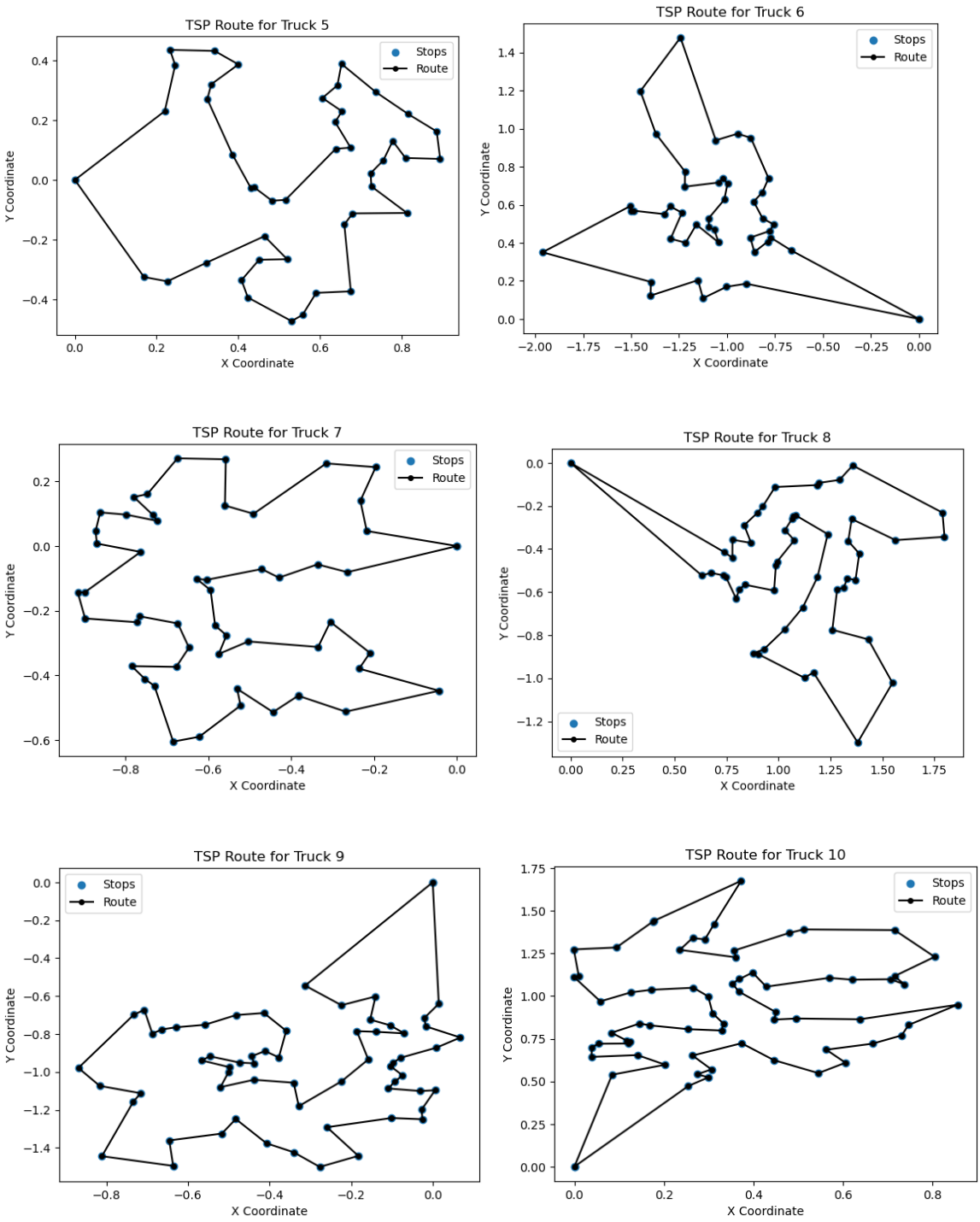


Exhibit A: K-means Clustering Results

We then assigned a truck to each cluster and conducted a TSP for each truck, yielding a **total distance of approximately 72.9 miles**. The routes for each truck are displayed below.





2. *Clustering + 2-Change Simulated Annealing*

This process uses the same clustering method as the TSP to assign the packages to the trucks; however, it makes random changes to the established paths to determine the ideal rather than a traditional TSP. This is done by changing the path of a random trucks in one of two ways by the flip of a coin.

The **transport method** takes a subsection of a random truck's path and puts it in another portion of that same truck's path. The function to perform this task is shown below (returning the truck number and its new path):

```
def transport(trucks):
    random_ix = np.random.randint(0, len(trucks))
    path = trucks[random_ix]
    nx = len(path)
    startstop = np.random.choice(nx,2,replace=False)
    if startstop[0] > startstop[1]:
        remove_path = np.append(path[startstop[0]:],path[:(startstop[1]+1)])
        closed_path = path[(startstop[1]+1):startstop[0]]
    else:
        remove_path = path[startstop[0]:(startstop[1]+1)]
        closed_path = np.append(path[0:startstop[0]],path[(startstop[1]+1):])
    nc = len(closed_path)
    if nc > 0:
        paste = np.random.choice(nc,1)[0]
        newpath = np.append(closed_path[0:(paste+1)],remove_path)#,closed_path[(paste+1):])
        newpath = np.append(newpath,closed_path[(paste+1):])
    else:
        newpath = path
    return [random_ix,newpath]
```

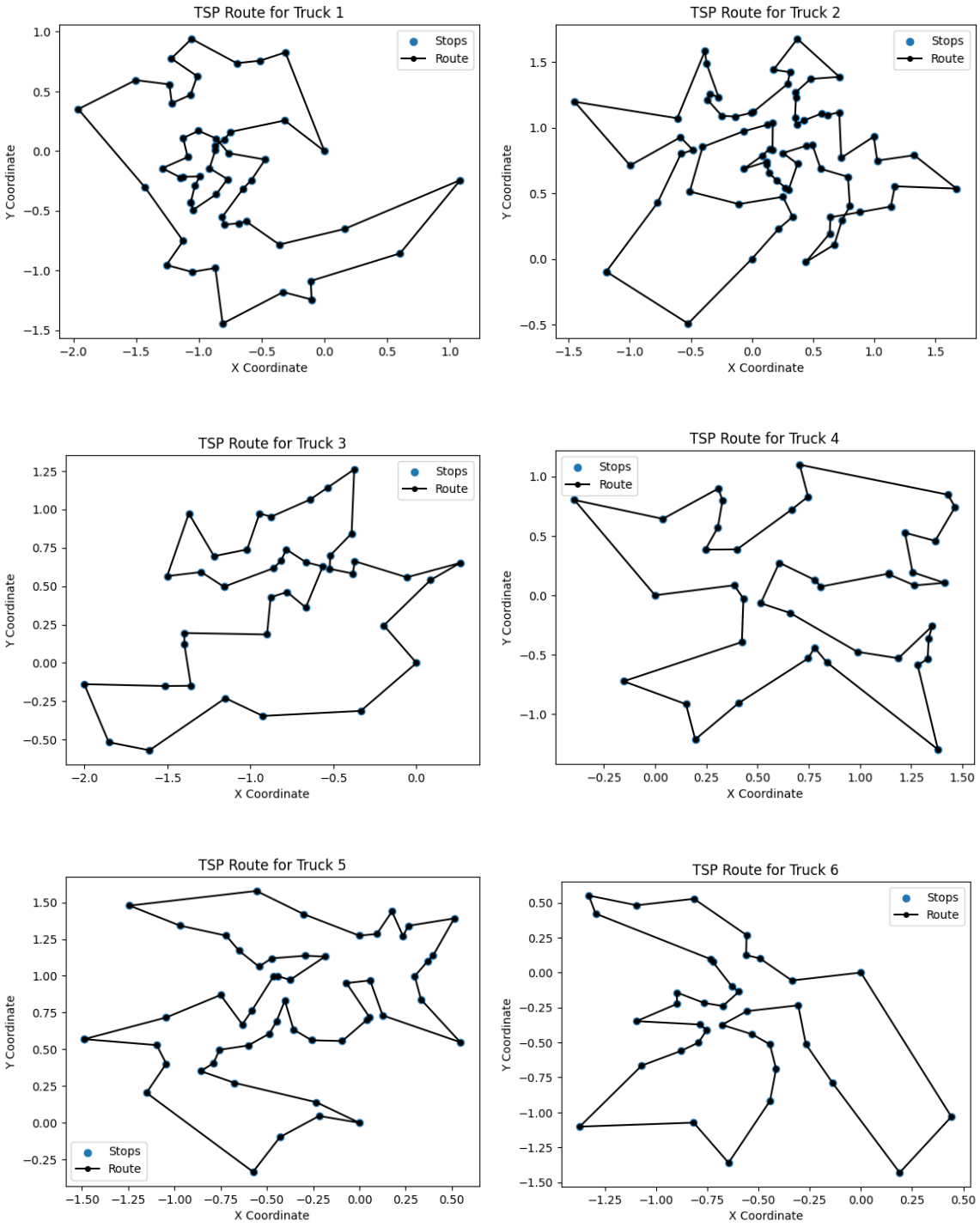
The **reversal method** takes a random section of the path of a random truck, reverses it, and inserts it in the same place it was before. With this method, the subpath still has the same entry and exit points as it did before it was reversed. The function to perform this task is shown below (returning the truck number and its new path):

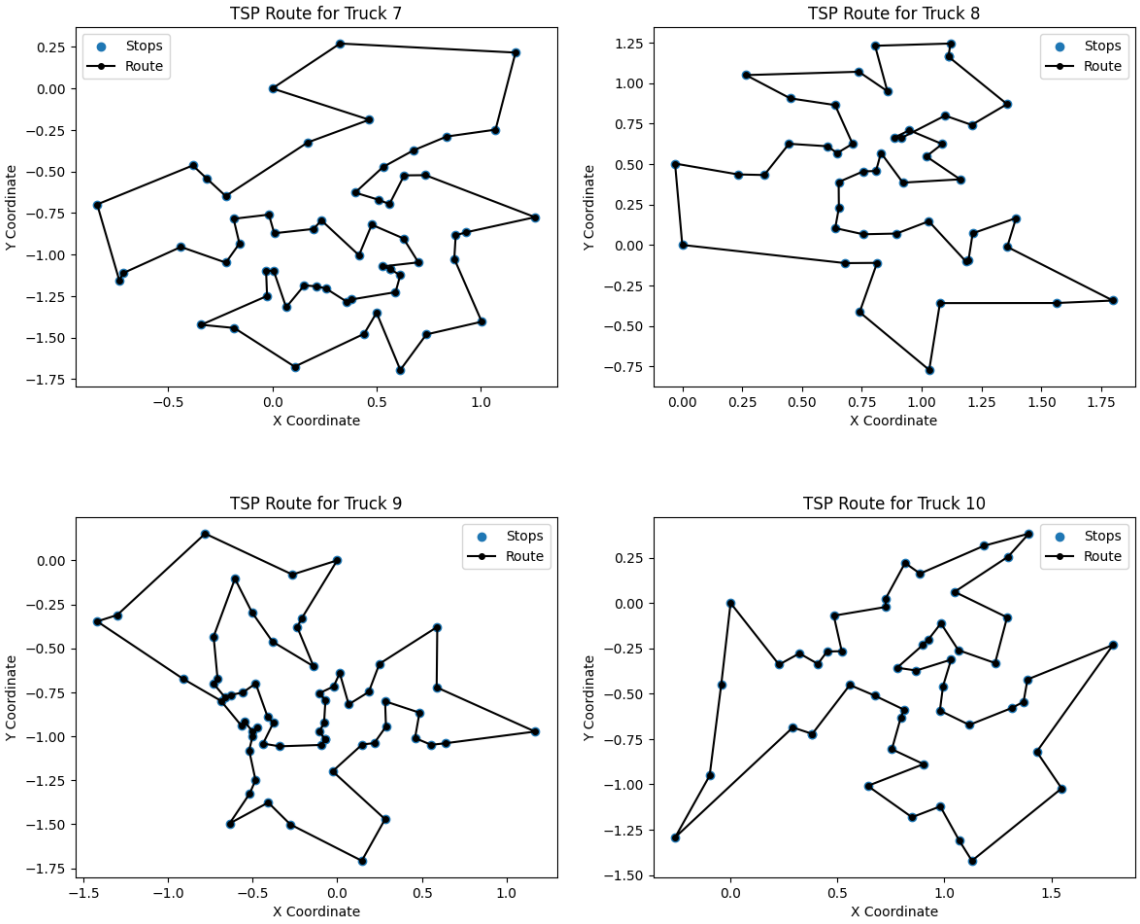
```
def reversal(trucks):
    random_ix = np.random.randint(0, len(trucks)) # chooses random truck
    path = trucks[random_ix] # retreives path of truck
    nx = len(path)
    startstop = np.random.choice(nx,2,replace=False) # chooses random start and stop
    if startstop[0] > startstop[1]: # if start is greater than stop, does this
        subpath = np.append(path[startstop[0]:],path[:startstop[1]+1]) # Start + anything after, anything before stop + stop
        subpath = np.flip(subpath) # Flips the path
        newpath = np.append(path[startstop[1]+1:startstop[0]],subpath) # Everything between start and stop appended by reversed path
    else: # stop is greater than start
        subpath = path[startstop[0]:startstop[1]+1] # Everything between stop and start
        subpath = np.flip(subpath)
        newpath = np.append(path[:startstop[0]],subpath) # Everything before start put before subpath
        newpath = np.append(newpath,path[startstop[1]+1:]) # Everything after start put after subpath

    return [random_ix,newpath] # returns truck number and it's new path
```

With these two possible changes, we then make a for loop of 100k iterations to continually try making either the transport change or reversal change. For each attempt, it has a 50/50 chance of choosing either the transport change or reversal change. The change is then made, and the squared distance is calculated. If the distance is better than the previous attempt, it is recorded as the new best distance, and that path is saved. If it is worse that our previous attempt, our previous best distance could be replaced by this new worse path A weighted coin, starting at 0.95 to keep the worse path, is flipped to determine whether the path is chosen, with an associated penalty for the difference in squared distance. With each iteration, the coin's weight is shrunk by 99%. After

100k iterations, the program takes the square root of the best distance and has the associated best path saved.

This methodology yielded a **total distance of approximately 39.39 miles,** and the routes for each of the trucks are displayed below:





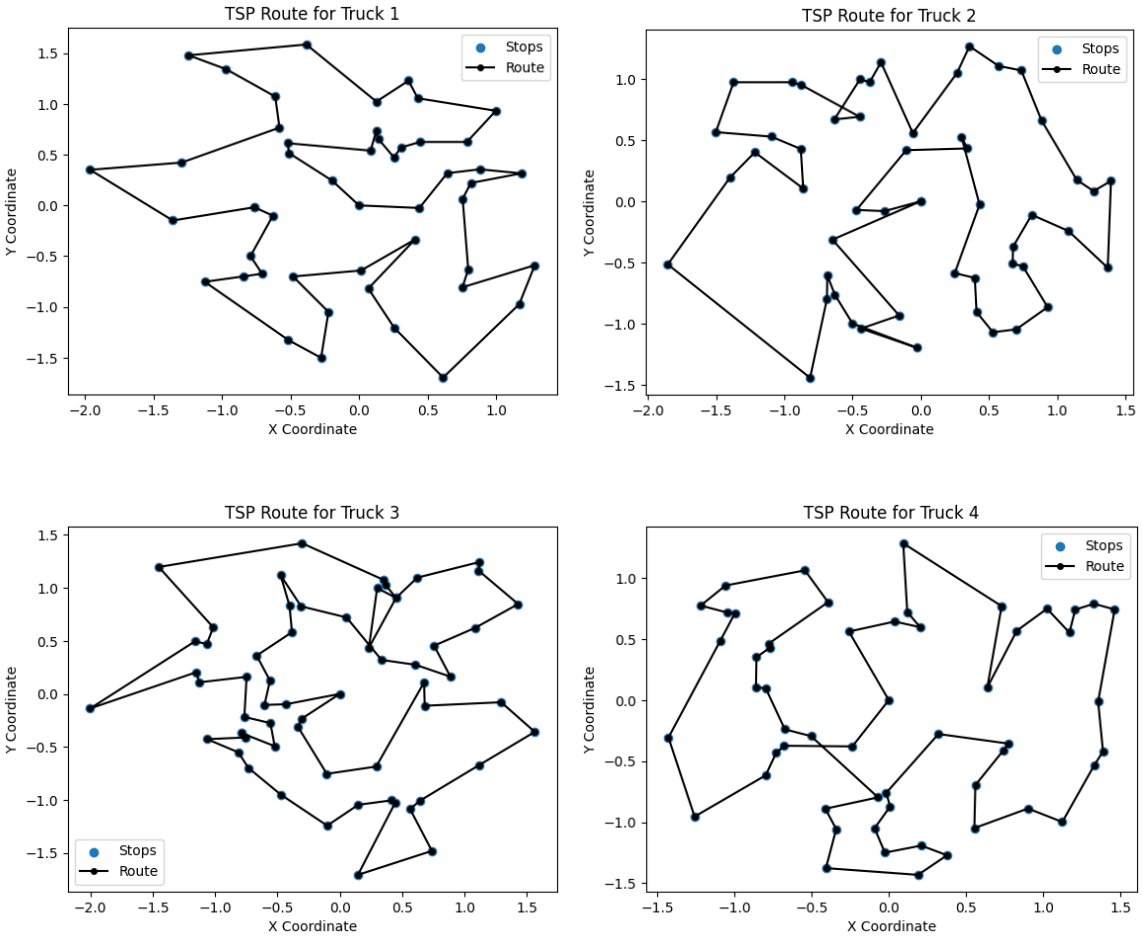
3. *3-Change Simulated Annealing Approach*

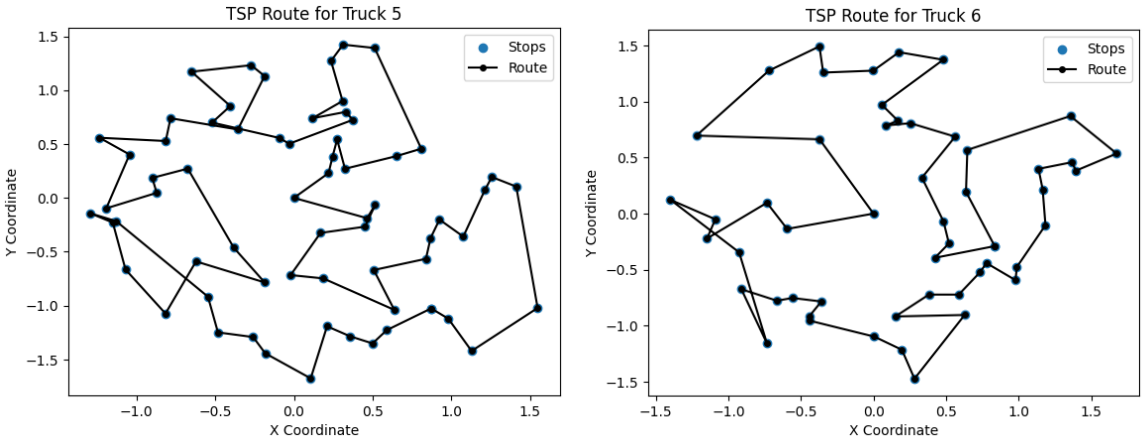
This is a two step process. Unlike the clustering approaches, we first assign the packages to the trucks completely randomly. Meaning, a truck can have any package, and it can contain any number of packages by random chance. After assigning the packages, the trucks go through the same process as the 2-change annealing method, but has an additional change that is equally likely to occur: taking a package from one truck and putting it on another. This is done with this additional function:

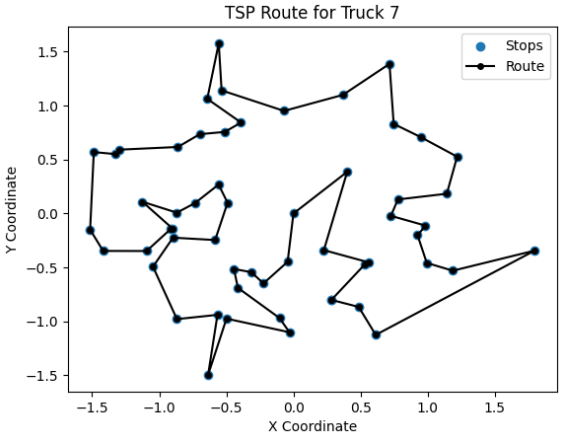
```
def package(t):
    random_steal = np.random.randint(0, len(trucks)) # random truck to steal from
    random_pack = 0
    while random_pack == 0: # loop to prevent the chosen package from being the distribution center (0,0)
      random_pack = np.random.randint(0, len(t[random_steal])) # Index of Package you're taking off truck
    random_give = np.random.randint(0, len(trucks)) # truck that receives it
    random_ix = np.random.randint(0, len(t[random_give])) # where in the route it's going
    package = t[random_steal][random_pack] # the package itself
    newpath = np.insert(t[random_give], random_ix, package) # puts package in the new truck's path
    return [random_give,newpath] # truck number + its new path
```

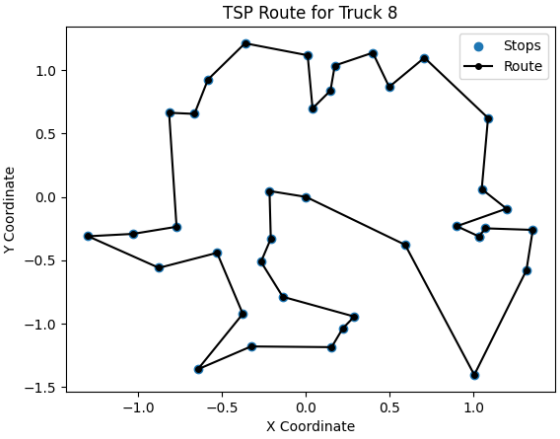
With this new function, the same loop is performed as before, but instead of choosing from the reverse and transport method, it chooses from the reverse, transport, and package functions. When performing this test, we looped this test 100,000 times, taking the new path if it performed better than the previous or took a worse distance based on the result of our weighted coin.

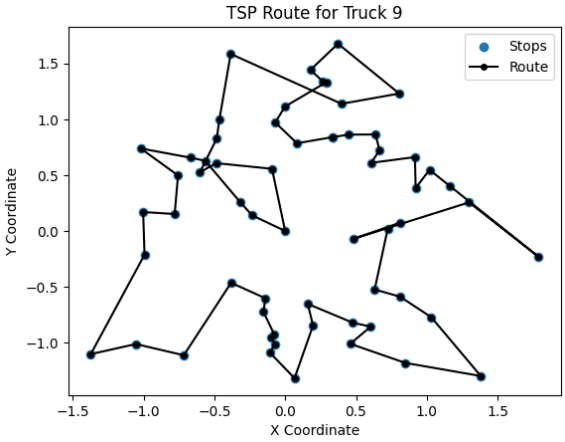
This methodology yielded a **total distance of approximately 59.98 miles,** and the routes for each of the trucks are displayed below:

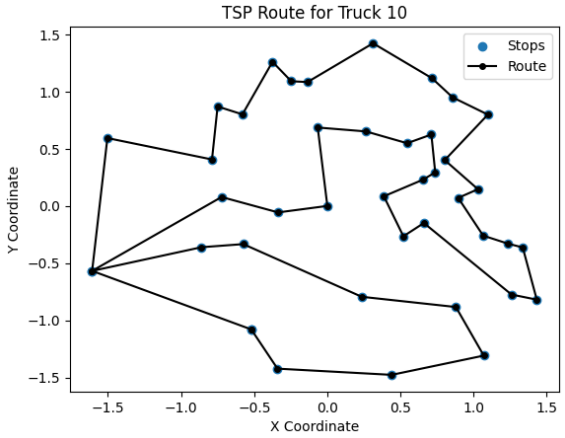












Result: Across all three methods, clustering + 2-change annealing performed the best. With a total distance across all 10 trucks of 39.39 miles, it is significantly more efficient than our other 2 models.

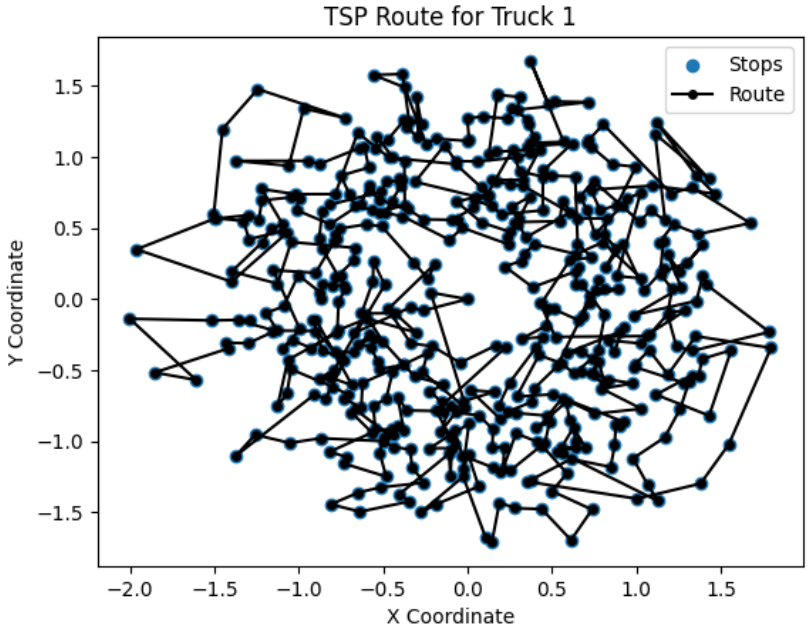
Part 2: Minimizing Costs

For determining the most cost efficient routes, we used the clustering + 2-change annealing method to determine routes as it performed the best previously. To determine the ideal number of trucks, we use our 100k iterations of our changes with 10 differently clustered data sets. This is done by having an outer loop that assigns our 500 packages to i number of trucks (out of 10), and then doing the annealing methods loop of 100k iterations. This produces the lowest possible total distance across all routes. We then take the square root of the best distance to calculate the cost. With one truck, this is calculated by adding the best distance (which is the same as the mileage cost of $1 per mile) and $300 for the driver. Otherwise, we use this formulation to determine the cost, as there is a penalty for drivers who work 10% more than average (abnormal path distances):

$$Cost = mileage + \Sigma\, 0.5 * (abnormal\ path\ distance - average\ path\ distance) + 300 * \#\ of\ trucks$$

After running this program, we find that the ideal number of trucks is 1, traveling a distance of 95.69 miles and costing the firm approximately $395.69. While this is the most total mileage traveled from all truck possibilities, it is still ideal cost wise because the cost of having the truck out significantly outweighs the cost per mile. To make having more trucks viable, the cost per distance would need to increase and the cost of just having the truck staffed would need to decrease.

Ideal Path:



Part 3: New Set of Packages

1. *Consistent Routing*

There is an incentive to make the truck routes for the new set of packages consistent with the initial routes. We used a nearest centroid classifier to generate clusters from the new set of packages that are as close as possible to the initial clusters.

```
X1 = data1[['x', 'y']]
X2 = data2[['x', 'y']]
num_clusters = 10 # num clusters = num trucks

# location1 clusters
kmeans = KMeans(n_clusters=num_clusters)
y_predict = kmeans.fit_predict(X1)

# find centroids of locations1 clusters
clf = NearestCentroid()
clf.fit(X1, y_predict)
centroids = clf.centroids_

# create and fit a K-Means model to cluster locations2 data based on centroids from locations1 data
kmeans = KMeans(n_clusters=num_clusters, init=centroids, max_iter=1, n_init=1)
kmeans.fit(X2)
data2['cluster_label'] = kmeans.labels_
```

First, k-means clustering is used to generate clusters based on the initial set of packages. Next, a nearest centroid classifier is fitted to this initial cluster data. The resulting centroids are set as the initial cluster centroids for a new k-means clustering model. This new model is used to generate the centroids for the new set of packages.

The clustering results using the nearest centroid classifier are displayed in **Exhibit B** below:

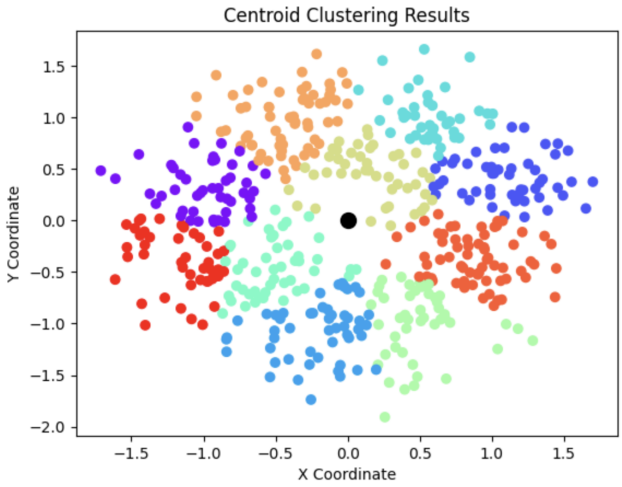


Exhibit B: Centroid Clustering Results

Using these new clusters of truck routes, we applied the **clustering plus 2-change simulated annealing method** from Part 1 to generate the optimal truck routes since it resulted in the least distance traveled by the trucks.

This approach yielded a total distance of approximately **22.02 miles** and total cost of **$3,045.16.**

2. *Routing from Scratch*

Another way to generate routes for the new set of packages is to apply the **clustering plus 2-change simulated annealing method** from Part 1 from scratch.

The clustering results are displayed in **Exhibit C** below:

Exhibit C: Clustering Results

This approach yielded a total distance of approximately **36.65 miles** and total cost of **$3,075.56**.

Result: Starting clustering and route generation from scratch causes an increase in cost by approximately $30, which is only a marginal difference. Moreover, maintaining consistency in routes can have intangible benefits such as increased driver satisfaction and reduced turnover since drivers would prefer to drive similar routes everyday.

Conclusion and Remarks

While using an algorithm to plan delivery routes eliminates the time spent on manually sorting packages and scheduling routes by automating the process, TSP optimization's costs outweigh the benefits since it is a time-consuming process to generate optimal routes for a large number of packages. However, the clustering with 2-change simulated annealing approach is an alternative that not only automates the process of scheduling delivery routes but also is more time-efficient compared to TSP optimization. Hence, this is a greedy approach that can decrease overall costs.