DELHI TECHNOLOGICAL UNIVERSITY

DST PROJECT REPORT
Last Mile Drone Delivery System

Archit Agarwal (2k19/IT/030)
Ayush Goyal (2k19/IT/036)

Table Of Contents

Delhi Technological University

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

Acknowledgement

I would like to convey our heartfelt thanks to our teacher Ms. Swati Sharda for her ingenious ideas, tremendous help and cooperation.
I am extremely grateful to my friends who gave valuable suggestions and guidance for completion of my project. The cooperation and healthy criticism came handy and useful with them.
Finally, I would like to thank all the above-mentioned people once again.

Delhi Technological University

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, (Ayush Goyal), Roll No – 2K19/IT/036, and (Archit Agarwal), Roll No – 2K19/IT/030 student of B.Tech (Information Technology), Delhi Technological University hereby declare that the project "Last Mile Drone Delivery System" which is submitted to the Department of Information Technology, Delhi Technological University, New Delhi in partial fulfillment of the requirement for the end term examination of 3rd semester, is original and not copied from any source without proper citation.
Place: Delhi

Date: 26-11-2020 Archit Agarwal & Ayush Goyal

Delhi Technological University

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

Abstract

We would be solving the last-mile delivery issue. We have taken inspiration from the works of various startups and companies like Amazon’s Prime-Air, UPS Drones, other unicorns working on it. We have discussed several approaches to solve this problem using the Travelling Salesman Problem, with different approaches and heuristics. Later we will also discuss mTSP (VRP) Solution to this problem. A truck will leave the depot with the packages to be delivered along with two drones. It will travel along a TSP tour and stop at the centroid of each cluster of delivery locations. After the truck reaches a cluster, both the drones will be dispatched with the packages to be delivered. These drones will cover all the delivery locations in the cluster following a VRP tour and return back to the truck. The truck will now move to the next cluster and the same process will be repeated again until all delivery locations have been covered. Finally, the truck will return back to the depot to collect the next slot of packages.

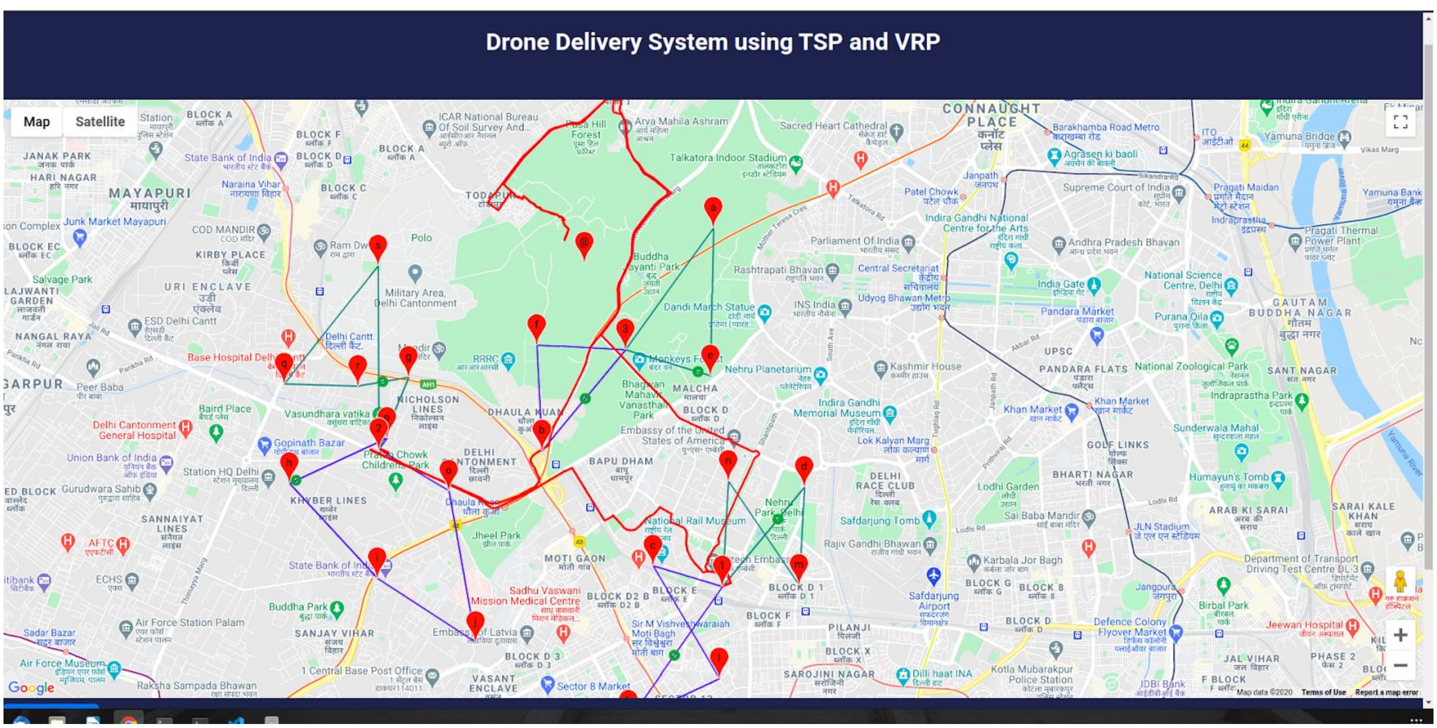
Introduction:

- Solve the last-mile delivery issue.
- Amazon Prime-Air, UPS Drones, other unicorns working on it
- We have discussed several approaches to solve this problem using Travelling Salesman Problem, with different approaches and heuristics
- Later we will also discuss mTSP (VRP) Solution to this problem.

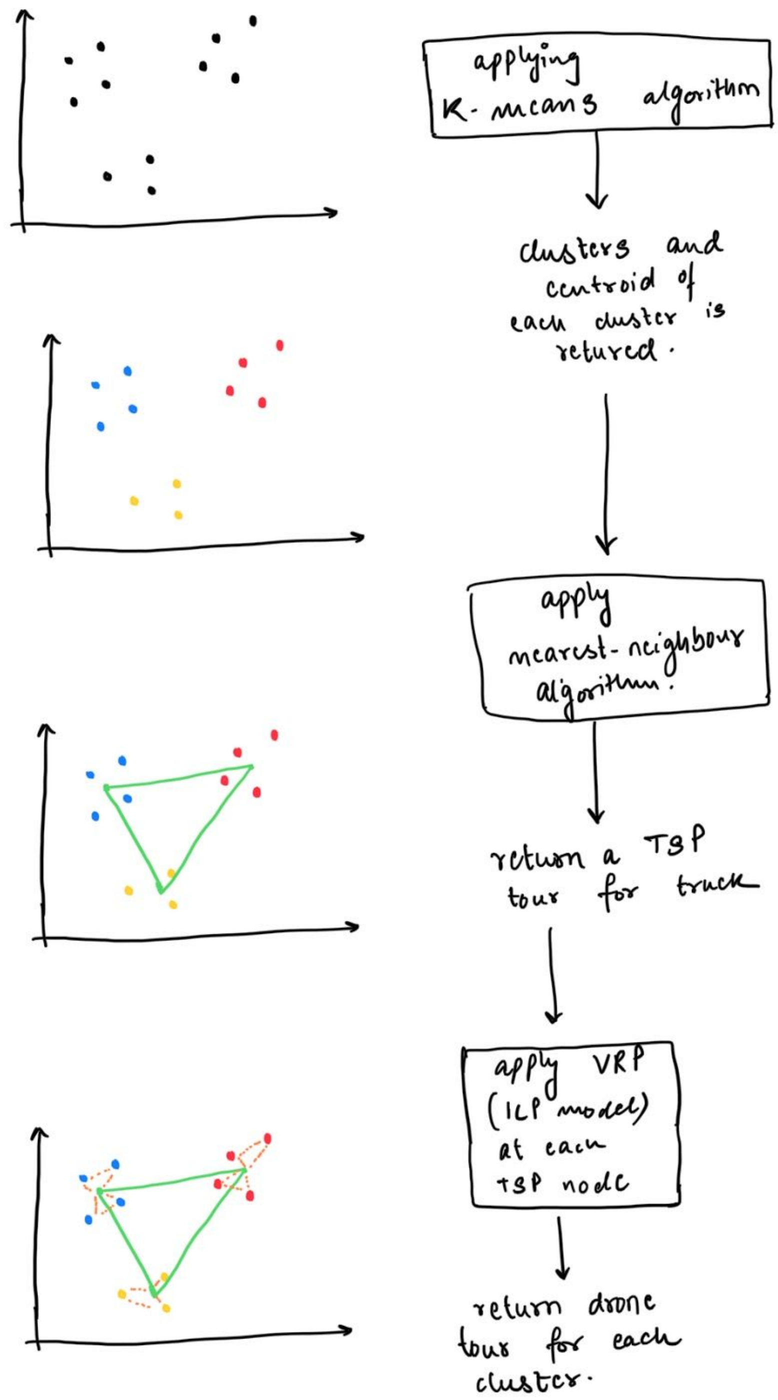
Problem Definition

Let there be m drones located at a depot. Then a single depot mTSP consists of finding tours for m drones such that all of them start and end at the depot, each other node is located in exactly one tour, the number of nodes visited by a drone lies within a predetermined interval, and the overall cost of visiting all nodes is minimized.

Working Screenshot -



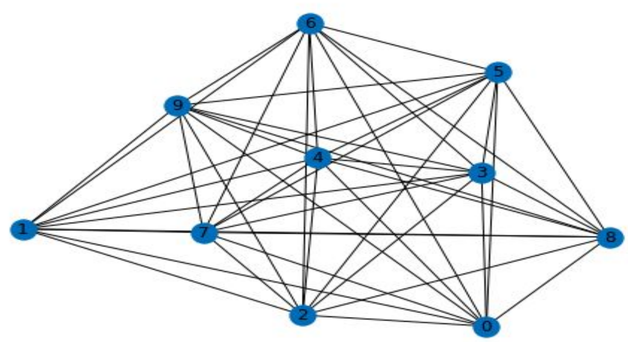
FlowChart


applying K-means algorithm
clusters and centroid of each cluster is retured.
apply nearest-neighbour algorithm.
return a TSP tour for truck
apply VRP (ILP model) at each TSP node
return drone tour for each cluster.

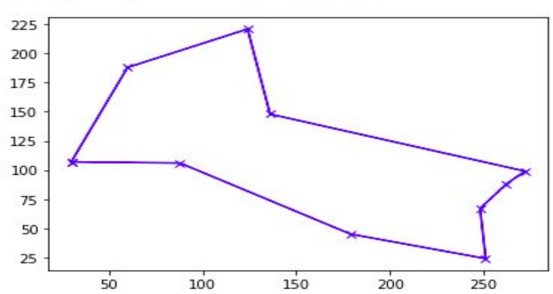
TSP

- Classic Problem of TSP. NP-Hard
- Bruteforce Solution by trying all combinations O(n-1)!
- Optimize it using DP Based Solution to reduce time complexity to O(2^n * n^2) by finding the Hamiltonian Path
- ILP to solve this mathematical regression
- Approximation Methods to solve the equations faster using different heuristics and metaheuristics
- The objective is to minimize operational costs including total transportation cost and one created by waste time a vehicle has to wait for the other.
- Different heuristics used are Nearest Neighbor and 2-Opt Algorithm

```python
# Consider the following 10 points.
coordinates = [(88, 106), (248, 67), (251, 24), (124, 221), (136, 148), (
# Create a corresponding graph.
g = get_graph(coordinates)
# Compute an optimal Hamiltonian path using Integer Linear Programming:
cycle = ilp(g)
print(cycle)
# Plot the resulting cycle
plot_cycle(coordinates, cycle)
```




```
the minimal cycle length is  702.6934098934813
[9, 7, 3, 4, 8, 5, 1, 2, 6, 0]
```



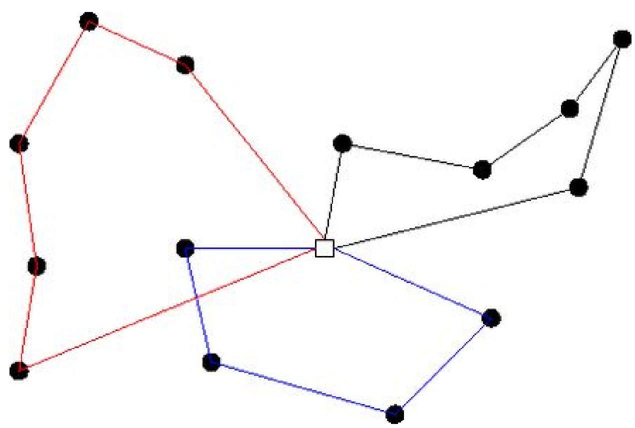
TSP and VRP

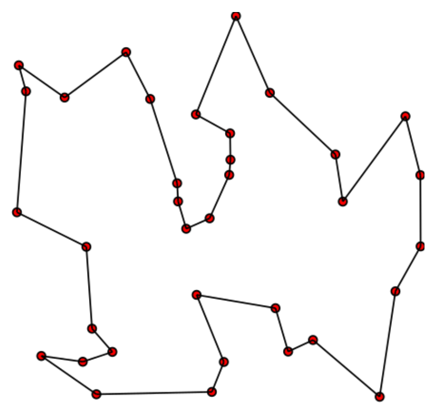
TSP asks the following question:

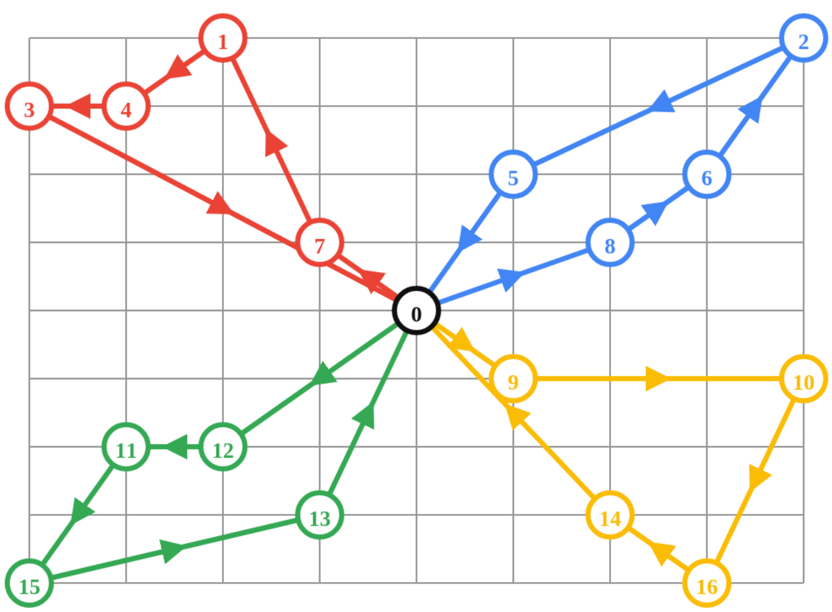
"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

VRP asks the following question:

"What is the optimal set of routes for a fleet of vehicles to traverse in order to deliver to a given set of customers?"







K-Means Clustering Algorithm

K-means is one of the simplest unsupervised learning algorithms that solve the clustering problems. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters). The main idea is to define k centers, one for each cluster.
The algorithm clusters the data into k clusters, even if k is not the right number of clusters to use. Therefore, when using k-means clustering, users need some way to determine whether they are using the right number of clusters.

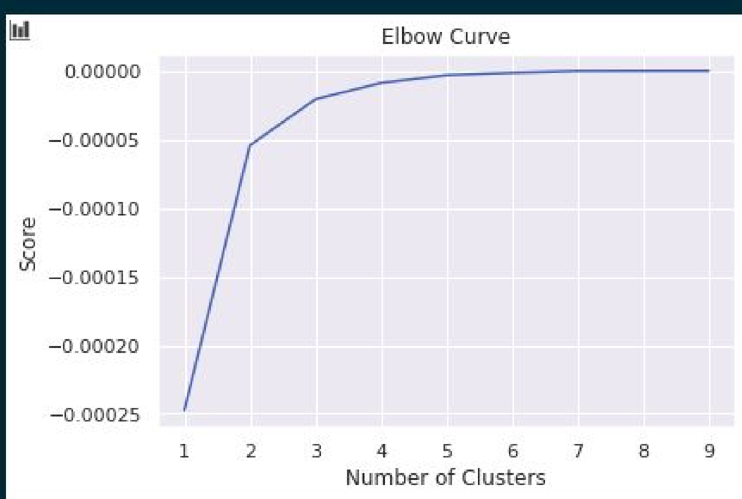
One method to validate the number of clusters is the elbow method. The idea of the elbow method is to run k-means clustering on the dataset for a range of values of k

```
plt.plot(K_clusters, score)
plt.xlabel('Number of Clusters')

plt.ylabel('Score')

plt.title('Elbow Curve')

plt.show()
```



We can thus make a 2-d graph using the given coordinates and process the information by making clusters for the same, and thus, applying VRP to individual clusters.

```python
def k_means():
    """Uses K-Means Clustering Algorithm and returns the centroid of each cluster.
    It also save the coordinates with the cluster label

    Returns:
        [list]: Truck Nodes (Centroids of Clusters)
    """
    df = pd.read_csv('coordinates.csv')
    X = df[1:]  # removed the depot
    # print(X.head(10))
    K_clusters = range(1, 10)  # exp b/w 1 and 10 clustrers
    # we generate model for each "k" clusters
    kmeans = [KMeans(n_clusters=i) for i in K_clusters]
    Y_axis = df[['latitude']]
    X_axis = df[['longitude']]
    score = [kmeans[i].fit(Y_axis).score(Y_axis) for i in range(len(kmeans))]

    # elbow ka score. Ideal elbow score at k = 3 (graph's monotonicty / elbow curve)
    print('elbow score → ', score)
    kmeans = KMeans(n_clusters=3, init='k-means++',
                    max_iter=1000)  # max iteration parameter
    # Compute k-means clustering. # Compute k-means clustering.
    kmeans.fit(X[X.columns[1:3]])

    X['cluster_label'] = kmeans.fit_predict(X[X.columns[1:3]])
    centers = kmeans.cluster_centers_  # Coordinates of cluster centers.

    # Labels of each point (0,0,1,1,2,2)
    labels = kmeans.predict(X[X.columns[1:3]])
    clustered_data = df.merge(X, left_on='City', right_on='City')
    clustered_data = clustered_data.drop(['latitude_y', 'longitude_y'], axis=1)
    clustered_data = clustered_data.rename(
        columns={"latitude_x": 'latitude', "longitude_x": "longitude"})
    centers = kmeans.cluster_centers_
    clustered_data.to_csv('coordinates_kmeans.csv', index=None, header=True)
    # for debugging purposes
    # X.plot.scatter(x='latitude', y='longitude', c=labels, s=50, cmap='viridis')      You, a mo
    # plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
    return centers
```

Research Paper

Consider a complete directed graph G = (V, A), where V is the set of n nodes (vertices), A is the set of arcs and C = (cij) is the cost (distance) matrix associated with each arc (i, j) in A. xij as a binary variable equal to 1 if arc (i, j) is in the optimal solution and 0 otherwise

Transportation Research Part C 86 (2018) 597–621

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

On the min-cost Traveling Salesman Problem with Drone

Quang Minh Ha[a], Yves Deville[a], Quang Dung Pham[b], Minh Hoàng Hà[c,*]

[a] ICTEAM, Université catholique de Louvain, Belgium
[b] SoICT, Hanoi University of Science and Technology, Viet Nam
[c] University of Engineering and Technology, Vietnam National University, Hanoi (VNU), Viet Nam

A R T I C L E I N F O

Keywords:
Traveling Salesman Problem with Drone
Minimize operational cost
Integer programming
Heuristic
GRASP

A B S T R A C T

Over the past few years, unmanned aerial vehicles (UAV), also known as drones, have been adopted as part of a new logistic method in the commercial sector called "last-mile delivery". In this novel approach, they are deployed alongside trucks to deliver goods to customers to improve the quality of service and reduce the transportation cost. This approach gives rise to a new variant of the traveling salesman problem (TSP), called TSP with drone (TSP-D). A variant of this problem that aims to minimize the time at which truck and drone finish the service (or, in other words, to maximize the quality of service) was studied in the work of Murray and Chu (2015). In contrast, this paper considers a new variant of TSP-D in which the objective is to minimize operational costs including total transportation cost and one created by waste time a vehicle has to wait for the other. The problem is first formulated mathematically. Then, two algorithms are proposed for the solution. The first algorithm (TSP-LS) was adapted from the approach proposed by Murray and Chu (2015), in which an optimal TSP solution is converted to a feasible TSP-D solution by local searches. The second algorithm, a Greedy Randomized Adaptive Search Procedure (GRASP), is based on a new split procedure that optimally splits any TSP tour into a TSP-D solution. After a TSP-D solution has been generated, it is then improved through local search operators. Numerical results obtained on various instances of both objective functions with different sizes and characteristics are presented. The results show that GRASP outperforms TSP-LS in terms of solution quality under an acceptable running time.

We propose the following integer linear programming formulation for the mTSP defined above.

$$\text{minimize} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{j=2}^{n} x_{1j} = m, \tag{2}$$

$$\sum_{j=2}^{n} x_{j1} = m, \tag{3}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 2, \ldots, n, \tag{4}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 2, \ldots, n, \tag{5}$$

I. Kara, T. Bektas / European Journal of Operational Research 174 (2006) 1449–1458 1451

$$u_i + (L-2)x_{1i} - x_{i1} \leqslant L-1, \quad i = 2, \ldots, n, \tag{6}$$

$$u_i + x_{1i} + (2-K)x_{i1} \geqslant 2, \quad i = 2, \ldots, n, \tag{7}$$

$$x_{1i} + x_{i1} \leqslant 1, \quad i = 2, \ldots, n, \tag{8}$$

$$u_i - u_j + Lx_{ij} + (L-2)x_{ji} \leqslant L-1, \quad 2 \leqslant i \neq j \leqslant n, \tag{9}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in A. \tag{10}$$

This formulation is valid when $2 \leqslant K \leqslant \lfloor (n-1)/m \rfloor$ and $L \geqslant K$. When $K \geqslant 4$, constraints (6) and (7) do not allow the situation $x_{1i} = x_{i1} = 1$, i.e., constraint (8) becomes redundant when $K \geqslant 4$. Thus, we need constraint (8) only for the cases $K = 3$ or $K = 2$.

```python
def ilp_model(sites, coordinateFile) → None:
    """Kara, I., & Bektas, T. (2006). Integer linear programming formulations of multiple salesman problems
        and its variations. European Journal of Operational Research, 174(3), 1449-1458.
        doi:10.1016/j.ejor.2005.03.008
    """
    # a handful of sites
    # sites = ['Depot', 'a', 'b', 'c', 'd']
    print('filename', coordinateFile)
    latlng = ['latitude', 'longitude']
    position = pd.read_csv(coordinateFile, index_col="City")
    flighttime = pd.read_csv('./time.csv', index_col="City")
    distance = pd.read_csv('./distance.csv', index_col="City")
    positions = dict(
        (city, (position.loc[city, 'longitude'], position.loc[city, 'latitude'])) for city in sites)
    print('===== POSITIONS → ', positions)
    distances = dict(((s1, s2), distance.loc[s1, s2])
                     for s1 in positions for s2 in positions if s1 ≠ s2)
    print('distances ', distance)
    K = 2
    prob = LpProblem("vehicle", LpMinimize)
    # indicator variable if site i is connected to site j in the tour
    # xij as a binary variable equal to 1 if arc (i, j) is in the optimal solution and 0 otherwise
    x = LpVariable.dicts('x', distances, 0, 1, LpBinary)
    # dummy vars to eliminate subtours
    # ui is the number of nodes visited on that travelerÕs path from the origin up to node i (i.e., the visit number
    u = LpVariable.dicts('u', sites, 0, len(sites)-1, LpInteger)
    # the objective (minimize signma c_ij, x_ij)
    cost = lpSum([x[(i, j)]*distances[(i, j)] for (i, j) in distances])
    prob += cost

    # constraints
    for k in sites:
        cap = 1 if k ≠ 'Depot' else K
        # inbound connection
        # we can only enter a node once (Paper pg 2, eq 4), i variable, j contant.
        prob += lpSum([x[(i, k)] for i in sites if (i, k) in x]) == cap
        # outbound connection
        # we can only exit a node once (Paper pg 2 eq 5)
        prob += lpSum([x[(k, i)] for i in sites if (k, i) in x]) == cap
```

Google Maps

The Maps JavaScript API lets you customize maps with your own content and imagery for display on web pages and mobile devices. The Maps JavaScript API features four basic map types (roadmap, satellite, hybrid, and terrain) which you can modify using layers and styles, controls and events, and various services and libraries.

```
function calculateAndDisplayRoute(directionsService, directionsDisplay, o, d, w) {
  directionsService.route(
    {
      origin: o,
      destination: d,
      waypoints: w,
      optimizeWaypoints: true,
      travelMode: 'DRIVING',
    },
    function (response, status) {
      if (status === 'OK') {
        directionsDisplay.setDirections(response);
      } else {
        window.alert('Directions request failed due to ' + status);
      }
    }
  );
}

// Use the DOM setInterval() function to change the offset of the symbol at fixed intervals.
function animateCircle(line) {
  let count = 0;
  window.setInterval(function () {
    count = (count + 1) % 200; // change this to 1000 to only show the line once
    let icons = line.get('icons');
    icons[0].offset = count / 2 + '%';
    line.set('icons', icons);
  }, 50); // change this value to change the speed
}
```

References

- https://developers.google.com/maps/documentation/javascript/overview
- https://stackoverflow.com
- https://www.geeksforgeeks.org/traveling-salesman-problem-tsp-implementation/
- https://sci-hub.do/10.1016/j.ejor.2005.03.008
- https://www.geeksforgeeks.org
- https://coin-or.github.io/pulp
- https://developers.google.com/optimization/routing/vrp