# Motivation

The motivation for this approximation project stems from the need to solve real-world transportation and logistics problems that involve connecting multiple locations with minimal cost or distance while considering other constraints such as frequently visited locations. By finding an approximate solution to the problem, you can achieve the following:

**Cost savings:** Minimizing the total distance or cost of the road network can lead to significant savings in construction, maintenance, and transportation expenses for governments, businesses, and individuals.

**Environmental benefits:** Reducing the total distance traveled contributes to lower fuel consumption and emissions, which is beneficial for the environment and public health.

**Time efficiency:** By finding a road network that minimizes travel distances, you can improve the efficiency of transportation systems, which in turn reduces the time it takes to travel between locations for businesses, deliveries, and commuters.

**Resource optimization:** With an optimal or near-optimal road network, the available resources (such as construction materials, manpower, and budget) can be allocated more efficiently, ensuring that the network is built and maintained in the most effective way possible.

**Scalability:** In real-world scenarios, the number of locations can be quite large, and finding an exact solution to the problem might be computationally expensive or even infeasible. Approximation algorithms can provide reasonable solutions in a relatively short amount of time, making them suitable for large-scale applications.

**Flexibility:** Approximation algorithms can be adapted to accommodate additional constraints or requirements, such as considering frequently visited locations, which makes them suitable for solving complex and dynamic transportation problems.

By developing an approximation algorithm for this problem, you can address these needs and contribute to improved transportation systems that benefit society as a whole.

# Model

To model the given problem as an Integer Linear Programming (ILP) problem, we can use a variation of the ILP formulation of the Travelling Salesman Problem (TSP). In this case, we aim to minimize the total distance traveled while visiting all nodes exactly once.

Problem Formulation as ILP:

Let G = (V, E) be a complete graph, where V is the set of nodes representing locations and E is the set of edges representing distances between pairs of nodes. Let w(i, j) be the distance between nodes i and j. We introduce binary decision variables:

x[i, j] = { 1, if the edge (i, j) is in the tour; 0, otherwise }

Objective function:

Minimize the total distance traveled: minimize ∑\_(i in V) ∑\_(j in V) w(i, j) \* x[i, j]

Subject to the following constraints:

1. Each node must be visited exactly once: ∑\_(j in V) x[i, j] = 1 for all i in V
2. Each node must be exited exactly once: ∑\_(i in V) x[i, j] = 1 for all j in V
3. Subtour elimination constraints (using Miller-Tucker-Zemlin formulation): u[i] - u[j] + |V| \* x[i, j] ≤ |V| - 1 for all i, j in V, i ≠ j and i, j ≠ 1

where u[i] are continuous variables representing the position of node i in the tour.

The ILP formulation above can be solved using an integer programming solver. However, it is worth noting that solving ILP problems is generally NP-hard, and the TSP is a well-known NP-hard problem. Therefore, finding an exact solution can be computationally expensive for large instances. In practice, approximation algorithms or heuristics like the 2-opt method are often used to find near-optimal solutions more efficiently.

# Problem Formulation

## MST

The problem formulation for this approximation project can be stated as follows:

Given a weighted graph G = (V, E), where V is the set of nodes (locations) and E is the set of edges (connections between locations) with associated weights (distances or costs), the goal is to find a connected subgraph T = (V, E') of G such that:

T is a tree (i.e., it is connected and acyclic).

The total weight of the edges in T is minimized.

The tree T includes the most frequently visited locations, based on a given set of constraints or preferences.

The problem can be considered as a variant of the Minimum Spanning Tree (MST) problem, with an additional requirement to prioritize the inclusion of frequently visited locations. Since finding an optimal solution to the MST problem can be computationally expensive, especially for large graphs, the goal is to develop an approximation algorithm that can provide a near-optimal solution in a relatively short amount of time.

The output of the algorithm should be a tree T that connects all nodes in the graph while minimizing the total edge weight and considering the constraints related to frequently visited locations.

## TSP

Given a weighted graph G = (V, E), where V is the set of nodes (locations) and E is the set of edges (connections between locations) with associated weights (distances or costs), the goal is to find a Hamiltonian cycle (a cycle that visits each node exactly once and returns to the starting node) such that the total weight of the cycle is minimized.

While the TSP and the problem you're considering both involve finding an optimal route on a graph, they have different objectives. The TSP focuses on finding the shortest cycle that visits all nodes exactly once, while the problem you're working on aims to find a tree that connects all nodes while minimizing total edge weight and considering the constraints related to frequently visited locations.

However, it is possible to adapt the TSP to incorporate the constraints and preferences related to frequently visited locations. In this case, the problem would involve finding a Hamiltonian cycle that minimizes the total weight while prioritizing the visitation of the most frequently visited locations. Note that this would still be a different problem than the one you initially formulated, as the output would be a cycle instead of a tree.

## 2-OPT TSP

we can formulate it as a 2-opt TSP problem by considering each location as a node and the distance between each pair of nodes as the edge weight. The objective is to find an approximate optimal tour that visits all the nodes exactly once and returns to the starting node, while minimizing the total distance traveled.

Problem Formulation as 2-opt TSP:

1. Input: A complete graph G = (V, E) where V is the set of nodes representing locations, E is the set of edges representing the distances between pairs of nodes, and a weight function w: E → ℝ+ that assigns a positive real number to each edge (distance).
2. Output: An approximate optimal tour T = (V, E') that visits all the nodes exactly once and returns to the starting node, such that the total distance (sum of edge weights) is minimized.
3. Optimization: Use the 2-opt heuristic to iteratively improve the initial tour by swapping pairs of edges to reduce the total distance. The 2-opt heuristic is not guaranteed to find the global minimum, but it can find a good approximation of the optimal tour in practice.

To apply the 2-opt heuristic for TSP to this problem, follow these steps:

1. Create an initial tour (a Hamiltonian cycle that visits all nodes exactly once and returns to the starting node).
2. Repeatedly swap pairs of edges in the tour and check if the new tour has a lower total distance. If it does, update the current tour with the new tour.
3. Terminate the algorithm when no further improvements can be made or after a certain number of iterations, and return the best tour found.

# Algorithm

## Kruskal MST

A. Sort edges and initialize MST

H ← G, w' ← w, Sort edges based on w'(edge) + w'(u) + w'(v)

Initialize empty MST, E\_MST ← ∅

B. Add edges without creating cycles

while |E\_MST| < |V| - 1:

Pick the smallest edge e(u,v) not in E\_MST from the sorted list

if adding e(u,v) to E\_MST doesn't create a cycle:

E\_MST ← E\_MST ∪ {e(u,v)}

C. Return the MST

return (V, E\_MST)

## TSP

A. Initialize the tour

tour ← [start\_node], unvisited\_nodes ← V - {start\_node}

current\_node ← start\_node

B. Find the closest unvisited node

while unvisited\_nodes ≠ ∅:

next\_node ← argminu∈unvisited\_nodes w(current\_node, u)

tour ← tour + [next\_node]

current\_node ← next\_node

unvisited\_nodes ← unvisited\_nodes - {next\_node}

C. Connect the last node back to the start node and return the tour

tour ← tour + [start\_node]

return tour

## 2-OPT TSP

A. Initialize the current tour

current\_tour ← initial\_tour, improvement ← true

B. Perform 2-opt swaps to find better tours

while improvement = true:

improvement ← false

for i = 0 to |V| - 2:

for j = i + 1 to |V| - 1:

new\_tour ← 2-opt swap of current\_tour at i and j

if length(new\_tour) < length(current\_tour):

current\_tour ← new\_tour

improvement ← true

C. Return the optimized tour

return current\_tour