

Ramdeobaba University, Nagpur
Department of Computer Science and Engineering

Session: 2025-26

Subject: Design and Analysis of Algorithms (DAA) Lab Project

III Semester

LAB PROJECT REPORT

SUBMITTED BY-

- 1) Soumya Wasule(A1-38)
- 2) Sharvan Kotharu(A1-43)
- 3) Sujal Tembhare(A1-49)

Github repository

https://github.com/SoumyaWasule/Vector_path.git

TITLE : LOGISTICS ROUTE MANAGER

OBJECTIVES

- ❖ Minimize Total Delivery Cost and Time
- ❖ Optimize Route Planning
- ❖ Maximize Vehicle and Resource Utilization
- ❖ Automate and Visualize Decision Making
- ❖ Improve Scalability and Reliability
- ❖ Enhance User Experience and Communication:
 - ❖ Bridge Theory and Practice



INTRODUCTION

In today's fast-paced e-commerce environment, companies need efficient ways to deliver goods to their customers quickly and at a low cost. The process of planning, organizing, and carrying out deliveries across multiple warehouses and customer locations is called supply chain logistics. Many challenges can arise during deliveries, such as delays, high costs, or poor use of delivery vehicles.

Our project aims to solve these problems by using advanced computer algorithms. We have created a web application that helps plan the best delivery routes, decide which packages should go on each trip, and make sure every delivery vehicle is working at its full capacity. The system uses well-known algorithms like Multi-Stage Graph, Traveling Salesman Problem (TSP), and Fractional Knapsack. The app also provides an interactive and animated visual interface so users or firms can clearly see how deliveries are optimized. This project will help e-commerce companies save money, deliver faster, and make their supply chain much more effective.

ALGORITHMS/TECHNIQUES USED

1) Travelling Salesman Problem Algorithm (*shortest and most efficient delivery route for a vehicle that needs to visit multiple delivery points or customer locations and return to the starting point*)

2) Multistaged graph Algorithm (*optimize the delivery routes across a series of mandatory warehouses or distribution centers arranged in sequential stages*)

3) Fractional Knapsack (*optimize the loading of delivery vehicles by selecting packages with the highest value-to-weight ratio to maximize overall profit within the vehicle's capacity constraints*)

Algorithms/Pseudocode :

TSP

```
function nearestNeighborTSP(distanceMatrix):  
    n = number of cities  
  
    visited = array of boolean initialized to False  
  
    route = list initialized with starting city (e.g., 0)  
  
    visited[0] = True  
  
    currentCity = 0  
  
    totalDistance = 0  
  
    for i in range(1 to n-1):  
  
        nearestCity = None  
  
        shortestDistance = infinity  
  
        for city in range(n):  
  
            if not visited[city] and  
            distanceMatrix[currentCity][city] < shortestDistance:  
  
                nearestCity = city  
  
                shortestDistance = distanceMatrix[currentCity][city]  
  
        route.append(nearestCity)
```

```
    visited[nearestCity] = True

    totalDistance += shortestDistance

    currentCity = nearestCity

totalDistance += distanceMatrix[currentCity][0] # Return to
start

route.append(0) # Complete cycle

return route, totalDistance
```

Time Complexity : $O(n^2)$

This is because for each city, the algorithm checks distances to all unvisited cities to find the nearest one, which takes $O(n)$ time per city. Since this process is repeated for all n cities, the total is $O(n^2)$. This makes it much faster than exact algorithms, although it does not always guarantee the optimal solution.

MSG

```
function multiStageGraph(graph, stages, n):  
    // n = total number of nodes  
  
    // stages = list of stages with nodes  
  
    // graph[i][j] = cost from node i to node j (infinity if no  
    edge)  
  
    cost = array of size n initialized to infinity  
  
    path = array of size n to store next node in optimal path  
  
    // Start from the last stage (destination node)  
  
    destination = last node in final stage  
  
    cost[destination] = 0  
  
    // Process stages from second-last to first (backward)  
  
    for stage from (number of stages - 2) down to 0:  
  
        for each node u in current stage:
```

```

minCost = infinity

nextNode = None

for each node v in next stage:

    if graph[u][v] exists and graph[u][v] + cost[v] <
minCost:

        minCost = graph[u][v] + cost[v]

        nextNode = v

        cost[u] = minCost

        path[u] = nextNode

// Trace the optimal path from source to destination

optimalPath = []

currentNode = source (first node in first stage)

optimalPath.append(currentNode)

while currentNode != destination:

    currentNode = path[currentNode]

    optimalPath.append(currentNode)

```

```
return optimalPath, cost[source]
```

Time Complexity : O(n×e)

The time complexity of the Multi-Stage Graph (MSG) algorithm using dynamic programming is typically $O(n \times e)$, where n is the number of nodes and e is the number of edges between stages. The algorithm processes each node in the graph and computes the minimum cost by examining all outgoing edges from that node to the next stage. This ensures an efficient calculation of the optimal path by building solutions incrementally from the destination back to the source, avoiding redundant calculations and making it suitable for stage-wise structured problems like supply chain logistics



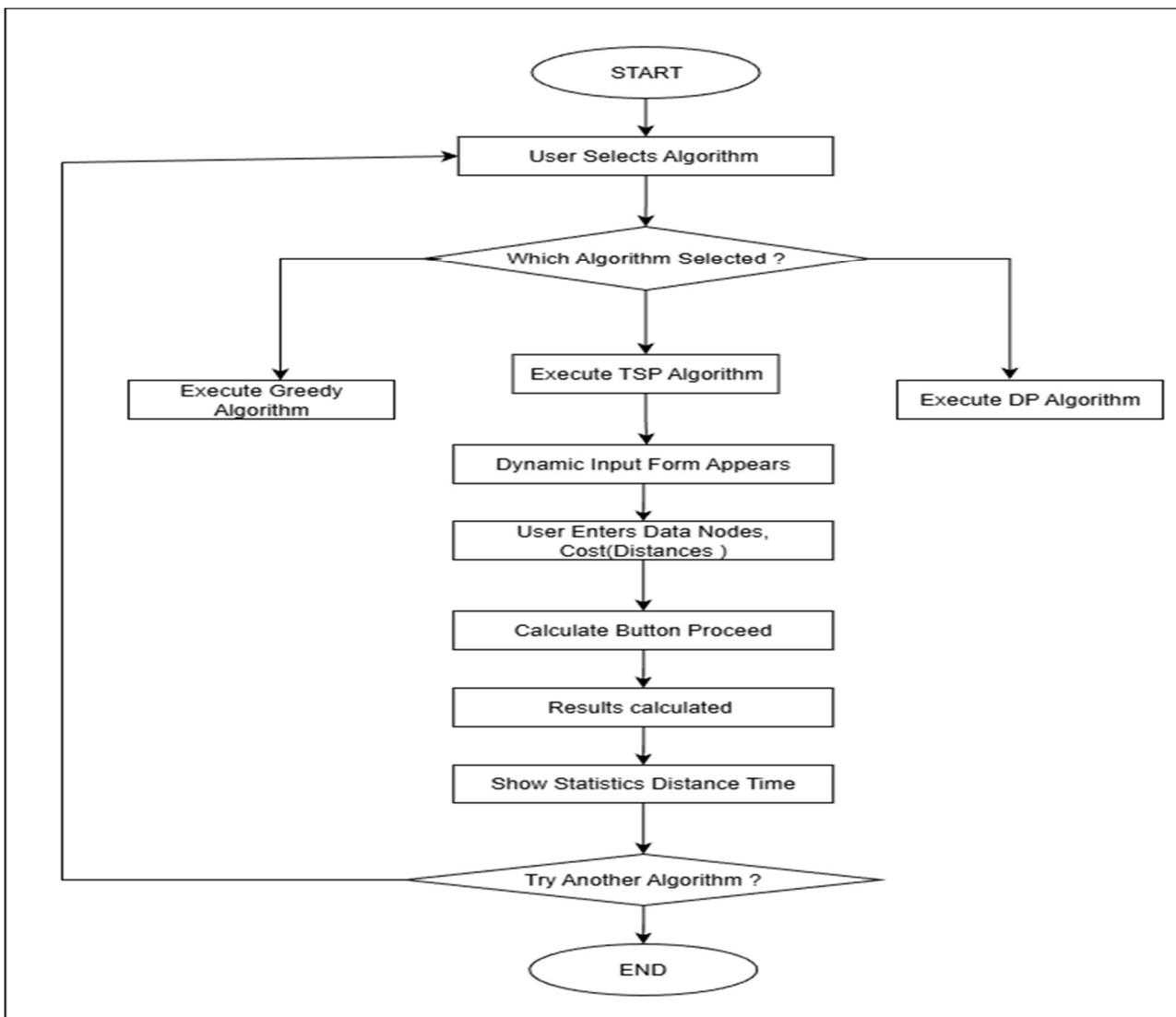
Fractional Knapsack

```
function fractionalKnapsack(values, weights, capacity):  
    n = number of items  
  
    items = array of (value, weight, value/weight)  
  
    for i in 0 to n-1:  
        items[i] = (values[i], weights[i], values[i]/weights[i])  
  
    sort items by value/weight ratio in descending order  
  
    totalValue = 0  
  
    remainingCapacity = capacity  
  
    for item in items:  
        if item.weight <= remainingCapacity:  
            totalValue += item.value  
            remainingCapacity -= item.weight  
  
        else:  
            fraction = remainingCapacity / item.weight  
            totalValue += item.value * fraction  
            break  
  
    return totalValue
```

Time Complexity

The Fractional Knapsack algorithm has a time complexity of $O(n \log n)$. This is because sorting items by their value-to-weight ratio takes $O(n \log n)$, and the subsequent single pass to select items adds $O(n)$. Sorting dominates, so the overall complexity is $O(n \log n)$.

LOGISTICS SUPPLY FLOW



RESULTS

The screenshot shows the home page of the Vectorpath Intelligent Delivery Network. At the top left is the logo and name. A blue button on the right says "Get Started". Below the logo is a section titled "Advanced Algorithm Solutions" with the heading "Optimize Your Delivery Network". A sub-section below it states: "Transform your e-commerce logistics with cutting-edge algorithms. Minimize costs, maximize efficiency, and deliver faster across multiple warehouses and distribution centers." Another "Get Started" button is located here. To the right are four performance metrics in cards: "45% Cost Reduction", "2.3x Faster Delivery", "98% Route Efficiency", and "34% Load Optimization". Below these is a "Performance Index" section with a real-time optimization bar chart consisting of 15 blue segments.

Home Page UI

1) MSG: Finds optimal path connecting warehouses

The screenshot shows the Algorithm Dashboard under the "Multi-Stage Graph" tab. The title is "Choose Algorithm". It features three options: "Multi-Stage Graph" (selected), "TSP Route", and "Knapsack Loading". The "Multi-Stage Graph" section includes input parameters: "Number of Stages" (set to 4), "Nodes per Stage" (set to 3), and "Edge Costs" (with a list: 0-1:2, 0-2:3, 0-3:4, 1-4:6, 1-5:4, 1-6:5, 2-4:4, 2-5:3, 2-6:5, 3-4:5, 3-5:6, 3-6:4, 4). A "Live Animation" section shows a graph with 8 nodes across 4 stages, with edges and costs labeled. Buttons at the bottom include "Calculate Route" and "Reset".

Users/Firms have the option to select a particular algorithm depending on their use case

2) TSP:-Finds the shortest path connecting locations of different customers

Algorithm Dashboard
Interactive Optimization Tools

Multi-Stage Graph
Optimal path through stages

TSP Route
Shortest tour visiting all cities

Knapsack Loading
Maximize value within capacity

Input Parameters
Configure parameters for TSP Route

Number of Locations

Delivery destinations including warehouse

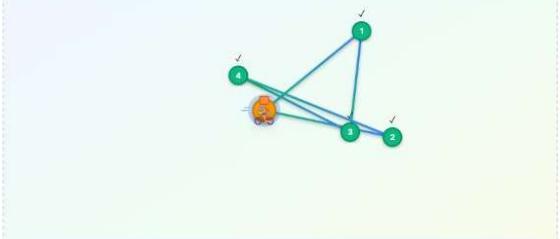
Distance Matrix

```
0-1:10, 0-2:15, 0-3:20, 0-4:25, 1-2:35, 1-3:25, 1-4:30, 2-3:30, 2-4:20, 3-4:20
```

Format: location1:location2:distance (comma-separated)

Calculate Route **Reset**

Live Animation
Watch the optimization in action



6 Returning to Warehouse
Step 6 of 6

Progress **100%**

3) Fractional Knapsack: Finds the best possible combination of goods for efficient packaging of goods

Algorithm Dashboard
Interactive Optimization Tools

Multi-Stage Graph
Optimal path through stages

TSP Route
Shortest tour visiting all cities

Knapsack Loading
Maximize value within capacity

Input Parameters
Configure parameters for Knapsack Loading

Vehicle Capacity (kg)

Maximum weight the vehicle can carry

Package Details

```
10:60:electronics, 20:100:furniture, 15:120:appliances, 20:100:television , 30:75:was
```

Format: weight:value:name (comma-separated)

Calculate Route **Reset**

Live Animation
Watch the optimization in action



Loading packages...

appliances 15.0kg \$120	electronics 10.0kg \$60	furniture 20.0kg \$100
television 5.0kg \$25		

Progress **100%**

CONCLUSION & FUTURE SCOPE

This project successfully demonstrates how advanced algorithms like Multi-Stage Graph (Dynamic Programming), Traveling Salesman Problem (TSP), and Fractional Knapsack can optimize a complex e-commerce supply chain. By efficiently planning delivery routes, scheduling vehicle trips, and maximizing package loading, the system minimizes overall delivery costs and time. The interactive application with animations helps visualize the logistics process clearly, making it easier for users and stakeholders to understand and apply these optimizations in real-world scenarios.

Future Scope:

- Integrate real-time traffic and weather data to dynamically adjust routes for better accuracy and responsiveness.
 - Expand the model to support multiple vehicles and simultaneous deliveries using advanced vehicle routing problem (VRP) algorithms.
 - Incorporate machine learning-based demand forecasting to proactively plan logistics resources.
 - Enhance UI with map APIs (Google Maps, Mapbox) for precise geolocation and delivery tracking.
 - Develop mobile apps for on-the-go monitoring and route adjustments by field drivers.
 - Explore IoT integration to automatically track vehicle status, deliveries, and optimize loads in real time.
-