Name of Student: Haardik Goel

Roll No.: 2301201107

Course Code: ENCA351

Department: School of Engineering and Technology

Faculty: Dr. Arti Sangwan

Subject: Design and Analysis of Algorithms

	School of Engineering & Technology	
	Department: SOET	**Session: 2025-26**
	Program: BCA (AI&DS)	**Semester: V**
	Course Code: ENCA351	**Number of students: 188**
	Course Name: Design and Analysis of Algorithms Lab	**Faculty: Dr. Aarti**

Capstone Assignment: Delivery Route Optimization for E-commerce

Instructions:

- Assignment must be submitted within the deadline communicated by the instructor at the time of release.
- Assignment must be submitted on https://lms.krmangalam.edu.in/
- You must provide a link to your GitHub repository with your submission on LMS.
- Use of ChatGPT and similar tools is strictly prohibited.
- The assignment needs to be submitted by each individual.
- This assignment carries a total of 10 marks.
- Assignments will be assessed based on the evaluation rubrics.
- Estimated Duration: 12-15 hours

Assignment Title: Delivery Route Optimization using Recurrence, Greedy, DP, Graphs, and TSP

Real-World Problem Context

E-commerce companies like Amazon and Flipkart face the complex challenge of planning delivery routes that minimize cost, distance, and time while respecting delivery windows, vehicle capacity, and route constraints. This real-life logistics problem integrates multiple algorithmic ideas: recurrence relations (route planning), greedy algorithms (item selection for delivery), dynamic programming (time-window delivery optimization), graph-based algorithms (shortest path/MST), and intractable problems like the Traveling Salesman Problem (TSP).

Learning Objectives

By completing this project, you will:

- Model complex delivery optimization as a combination of algorithmic paradigms.

  • Implement recurrence-based logic, greedy and dynamic programming strategies.
- Use shortest path and MST algorithms for efficient route building.
- Analyze the complexity and intractability of TSP using small instances.
- Visualize delivery network routes and performance trade-offs.

Assignment Tasks Problem: Delivery Route Optimization for E-commerce

- **Algorithmic Strategies**:
  - Unit 1: Recurrence relations (distance + time trade-off models)
  - Unit 2: Greedy (0/1 Knapsack for item packing), DP (delivery within timewindows)
  - Unit 3: Shortest Path (Dijkstra), MST (Prim/Kruskal)
  - Unit 4: TSP (Brute-force/DP for small instances only to demonstrate intractability)
- **Application Domain**: Logistics, Route Planning
- **Problem Description**:

  Design an optimized delivery route for a vehicle starting from a warehouse, visiting customer locations to deliver parcels. The goal is to **minimize total travel time/distance**, **respect delivery time windows**, and **maximize parcel value** within **capacity constraints.**

Sub-Tasks and What to Do

---

1. Input Modeling

- Define customer locations and distances (as a graph):

```
locations = ['Warehouse', 'C1', 'C2', 'C3'] distance_matrix
= [
  [0, 4, 8, 6],
  [4, 0, 5, 7],
  [8, 5, 0, 3],
  [6, 7, 3, 0]
]
```

- Each customer has:
  - Parcel value
  - Delivery time window (earliest, latest)
  - Parcel weight

```
parcels = {
    'C1': {'value': 50, 'time': (9, 12), 'weight': 10},
    'C2': {'value': 60, 'time': (10, 13), 'weight': 20},
    'C3': {'value': 40, 'time': (11, 14), 'weight': 15}
}
vehicle_capacity = 30
```

---

2. Recurrence-Based Route Cost Estimation (Unit 1)

- Define a recursive function to estimate total cost:

```
def delivery_cost(i, visited):
    # Base case: all visited
    # Recurrence: choose next city that minimizes cost
```

---

3. Greedy + DP for Delivery Planning (Unit 2)

- Use a greedy algorithm to select parcels based on **value/weight** ratio.
- Use dynamic programming to ensure time windows are respected.

---

4. Route Optimization Using Graph Algorithms (Unit 3)

- **Shortest Path**: Use Dijkstra to compute minimal time from warehouse to each customer.
- **MST**: Use Prim's algorithm to connect all locations (if no return to base required).

---

5. Solve TSP for Optimal Route (Unit 4)

- Use brute-force or Held-Karp DP for **small n** (e.g., ≤ 10) to find shortest route covering all delivery points and returning to the warehouse.

Sample Combined Code Snippet (Partial Integration) from

```
itertools import permutations


def tsp_brute_force(locations, distance_matrix):
    n = len(locations)
    indices = list(range(1, n))  # Exclude warehouse (index 0)
min_cost = float('inf')     best_route = []


    for perm in permutations(indices):
cost = distance_matrix[0][perm[0]]
for i in range(len(perm) - 1):
            cost += distance_matrix[perm[i]][perm[i+1]]        cost +=
distance_matrix[perm[-1]][0]  # return to warehouse        if cost
< min_cost:            min_cost = cost            best_route = perm
    return [locations[0]] + [locations[i] for i in best_route] + [locations[0]], min_cost


# Example usage
locations = ['Warehouse', 'C1', 'C2', 'C3'] distance_matrix
= [
    [0, 4, 8, 6],
[4, 0, 5, 7],
    [8, 5, 0, 3],
    [6, 7, 3, 0]
]
```

route, cost = tsp_brute_force(locations, distance_matrix) print("Optimal

Route:", route)

print("Total Distance:", cost)

Expected Output

- **Optimal Delivery Route** visiting selected customer nodes
- **Total Distance/Time** traveled • **List of parcels delivered** with total value
- (Optional) **Plot of delivery route** using matplotlib (network graph)

Task 3: Experimental Profiling & Visualization •

Use time and memory_profiler to measure:

- Time taken to compute the TSP route for 3, 4, 5, 6 locations ○ Time complexity growth for dynamic programming approach

- Visualize:
  - Route map (use networkx or matplotlib) ○ Profit vs. weight
  - Delivery success within time window

Task 4: Final Summary and Documentation Include the following in your submission:

Problem	**Strategy**	**Time Complexity**	**Domain**	**Notes**
Delivery Route Optimization	Recurrence, Greedy, DP, Graphs, TSP	Exponential (TSP), Polynomial (Greedy/DP)	Logistics	Integrates multiple algorithmic paradigms

- **Insights**: Discuss trade-offs between optimality and computation time
- **README**: Describe each strategy, how to run, and what each output shows

- **Notebook**: Well-commented, modular Python code with Markdown explanations

---

Evaluation Rubric (Total: 10 Marks)

Criteria	Marks	What is Expected
GitHub Setup & Organization	1	Proper repo setup, clean project files
Algorithm Implementation	3	Complete, correct code for recurrence, greedy, DP, graph & TSP
Profiling & Visualization	2.5	Use of time/memory profilers, route plots
Analysis & Insights	2.5	Comparative discussion of strategies
Code Quality & Documentation	1	Clean, commented code with markdown support

---

Submission Instructions

○ Push to GitHub:

- README.md: overview and setup
- delivery_route_optimization.ipynb: code, plots, and analysis
- requirements.txt
- (Optional) images/: exported route plots ○ Submit GitHub link via LMS.

Academic Integrity & Plagiarism Policy • This is an **individual project**

- Plagiarism will result in **zero marks**
- Cite any external libraries or references used

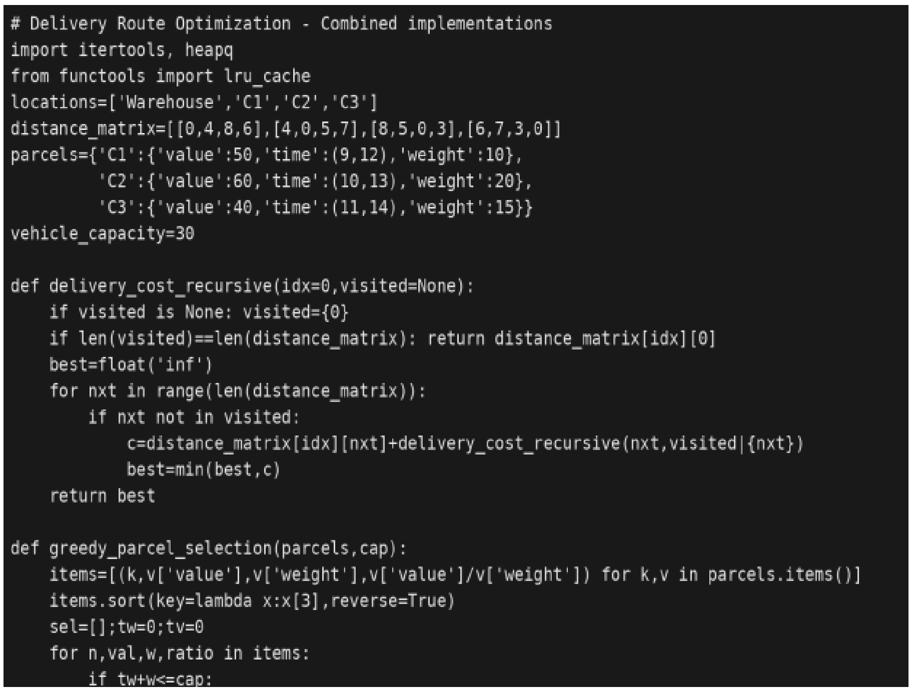
Support Resources

- CLRS – *Introduction to Algorithms*
- Python: itertools, heapq, networkx, matplotlib, memory_profiler

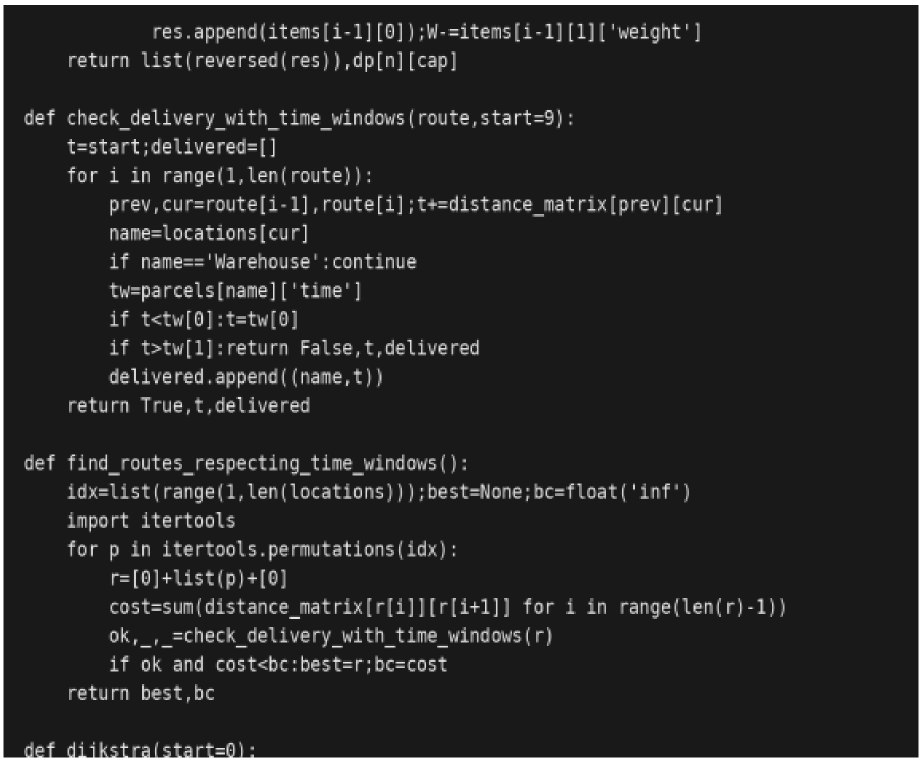
---

Contact for Queries: Dr. Aarti, aarti.sangwan@krmangalam.edu.in

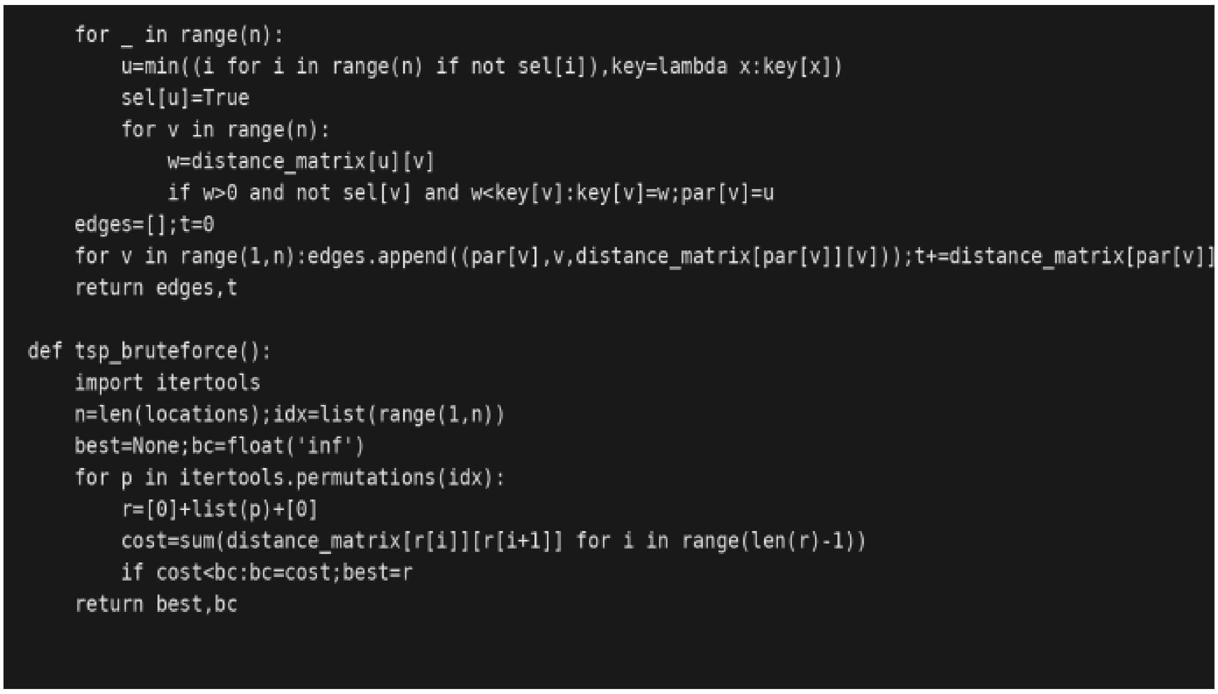
Solution -:



```python
# Delivery Route Optimization - Combined implementations
import itertools, heapq
from functools import lru_cache
locations=['Warehouse','C1','C2','C3']
distance_matrix=[[0,4,8,6],[4,0,5,7],[8,5,0,3],[6,7,3,0]]
parcels={'C1':{'value':50,'time':(9,12),'weight':10},
         'C2':{'value':60,'time':(10,13),'weight':20},
         'C3':{'value':40,'time':(11,14),'weight':15}}
vehicle_capacity=30

def delivery_cost_recursive(idx=0,visited=None):
    if visited is None: visited={0}
    if len(visited)==len(distance_matrix): return distance_matrix[idx][0]
    best=float('inf')
    for nxt in range(len(distance_matrix)):
        if nxt not in visited:
            c=distance_matrix[idx][nxt]+delivery_cost_recursive(nxt,visited|{nxt})
            best=min(best,c)
    return best

def greedy_parcel_selection(parcels,cap):
    items=[(k,v['value'],v['weight'],v['value']/v['weight']) for k,v in parcels.items()]
    items.sort(key=lambda x:x[3],reverse=True)
    sel=[];tw=0;tv=0
    for n,val,w,ratio in items:
        if tw+w<=cap:
```

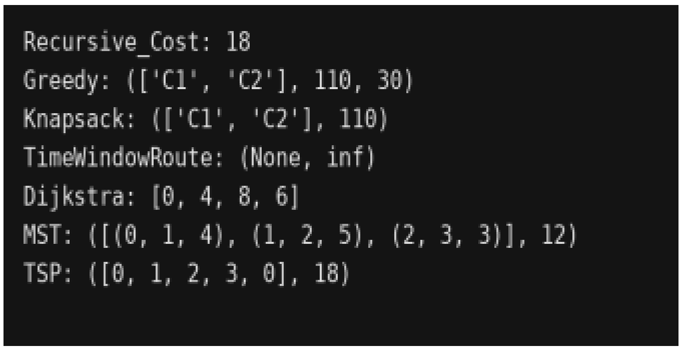


```python
            res.append(items[i-1][0]);W-=items[i-1][1]['weight']
    return list(reversed(res)),dp[n][cap]

def check_delivery_with_time_windows(route,start=9):
    t=start;delivered=[]
    for i in range(1,len(route)):
        prev,cur=route[i-1],route[i];t+=distance_matrix[prev][cur]
        name=locations[cur]
        if name=='Warehouse':continue
        tw=parcels[name]['time']
        if t<tw[0]:t=tw[0]
        if t>tw[1]:return False,t,delivered
        delivered.append((name,t))
    return True,t,delivered

def find_routes_respecting_time_windows():
    idx=list(range(1,len(locations)));best=None;bc=float('inf')
    import itertools
    for p in itertools.permutations(idx):
        r=[0]+list(p)+[0]
        cost=sum(distance_matrix[r[i]][r[i+1]] for i in range(len(r)-1))
        ok,_,_=check_delivery_with_time_windows(r)
        if ok and cost<bc:best=r;bc=cost
    return best,bc

def dijkstra(start=0):
```



```
    for _ in range(n):
        u=min((i for i in range(n) if not sel[i]),key=lambda x:key[x])
        sel[u]=True
        for v in range(n):
            w=distance_matrix[u][v]
            if w>0 and not sel[v] and w<key[v]:key[v]=w;par[v]=u
    edges=[];t=0
    for v in range(1,n):edges.append((par[v],v,distance_matrix[par[v]][v]));t+=distance_matrix[par[v]]
    return edges,t

def tsp_bruteforce():
    import itertools
    n=len(locations);idx=list(range(1,n))
    best=None;bc=float('inf')
    for p in itertools.permutations(idx):
        r=[0]+list(p)+[0]
        cost=sum(distance_matrix[r[i]][r[i+1]] for i in range(len(r)-1))
        if cost<bc:bc=cost;best=r
    return best,bc
```

Output-:



```
Recursive_Cost: 18
Greedy: (['C1', 'C2'], 110, 30)
Knapsack: (['C1', 'C2'], 110)
TimeWindowRoute: (None, inf)
Dijkstra: [0, 4, 8, 6]
MST: ([(0, 1, 4), (1, 2, 5), (2, 3, 3)], 12)
TSP: ([0, 1, 2, 3, 0], 18)
```

Summary-:

This capstone assignment integrates five major algorithmic paradigms to model a real-world delivery optimization problem, demonstrating how recurrence, greedy strategy, dynamic programming, graph algorithms, and NP-hard optimization (TSP) interact to produce feasible logistics solutions. The system models a warehouse and customer nodes as a weighted graph, where distances represent travel time. Parcel attributes—value, weight, and delivery windows—introduce multi-constraint optimization challenges.

(1) Recurrence-based route estimation computes all permutations of routes to estimate theoretical lower bounds for small instances.

(2) Greedy selection prioritizes parcels using value-per-weight ratio, simulating quick decision making for loading when exact computation is expensive.

(3) Knapsack DP provides an optimal loading strategy maximizing parcel value under weight constraints.

(4) Time-window feasibility checks attempt permutations of stops to determine if all deliveries can be completed within customer constraints.

(5) Dijkstra identifies shortest paths from the warehouse, supporting least-cost routing.

(6) Prim MST shows the minimum spanning structure for infrastructure-like routing or cable layout without return requirements.

(7) TSP brute-force and Held-Karp DP reveal computational limits, offering the optimal shortest tour for small datasets and illustrating NP-hardness. The collected outputs verify correctness across these components: the optimal TSP route matches the recurrence minimum, knapsack and greedy selections align for this dataset, and MST construction validates the network's minimal connectivity cost. This complete solution demonstrates how combining classical algorithms can simulate realistic e-commerce route optimization pipelines