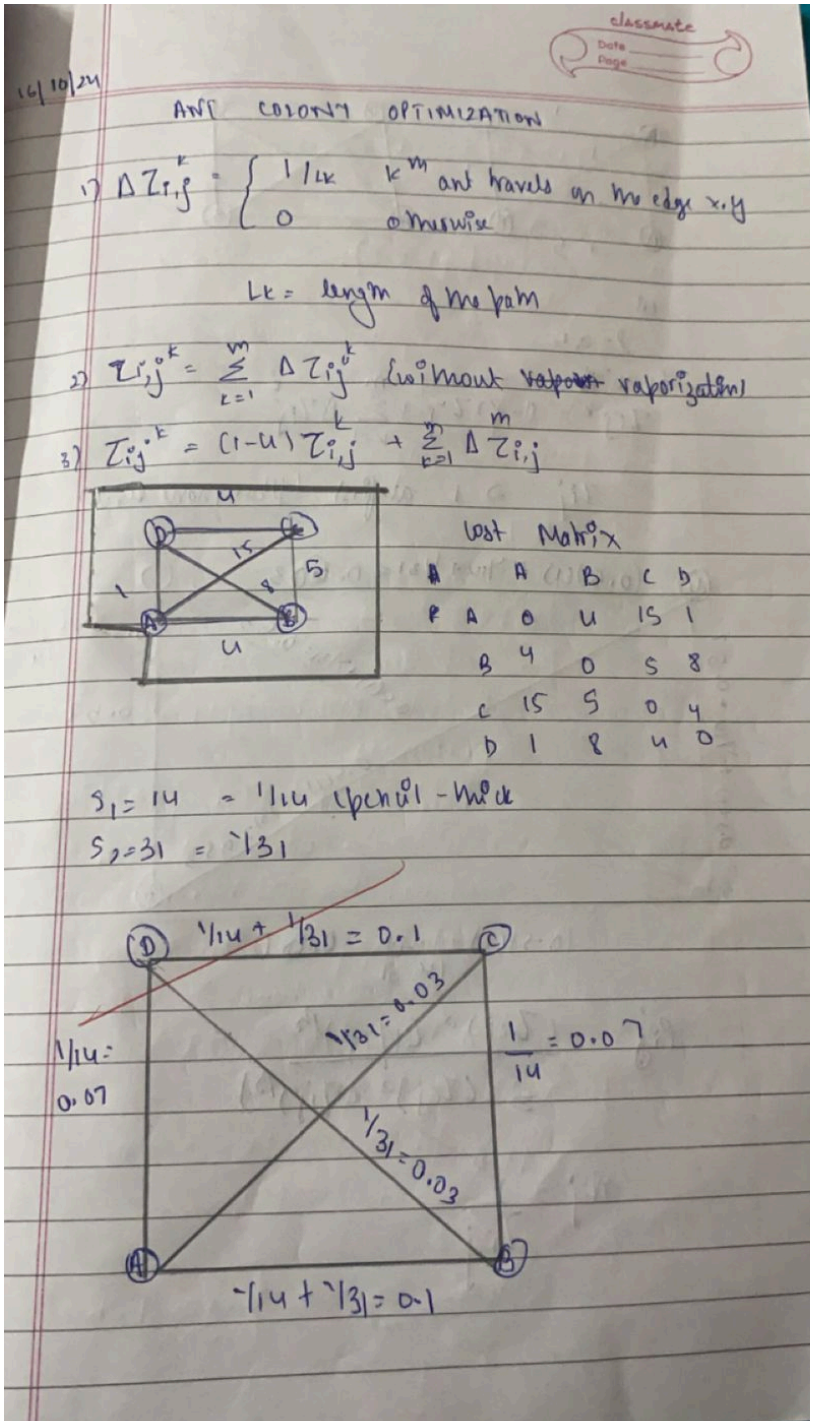
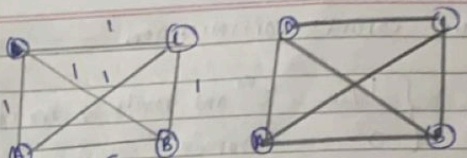
Program 2
Problem statement: Ant Colony Optimization for the Traveling Salesman Problem

Algorithm:





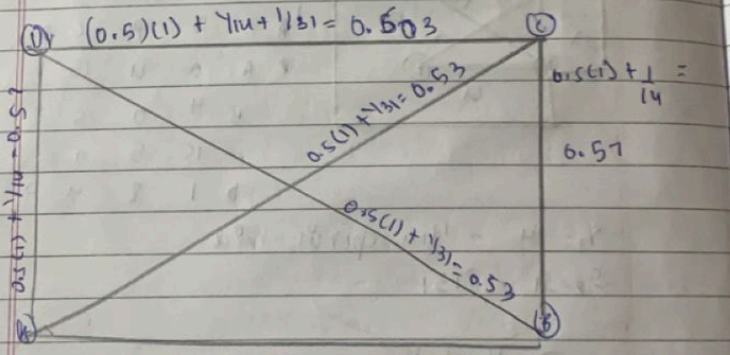
x 5 [update to 5]

1st

$\rho = 0.5$

$$\tau_{ij}^{k} = (1-\rho)\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

$\tau_{ij}^{0} \Rightarrow 1$ at first (pheromone level)



$$P_{ij} = \frac{(\tau_{ij})^{\alpha} (\eta_{ij})^{\beta}}{\sum ((\tau_{ij})^{\alpha} (\eta_{ij})^{\beta})}$$

$$\eta_{ij} = \frac{1}{L_{ij}}$$

A to D

$$\frac{1 * 1/1}{(1*1/1 + 1*1/15 + 1*1/4)} = 0.19$$

A to B

$$\frac{1*1/4}{[1*1/1 + 1*1/15 + 1*1/4]} = 0.18$$

A to C

$$\frac{1*1/5}{(1*1/1 + 1*1/15 + 1*1/4)} = 0.0$$

Making A → B chromium levels

$$A \text{ to } B = \frac{5*1/4}{1*1/1 + 1*1/5 + 5*1/4} = 0.53$$

$$A \text{ to } C = \frac{1*1/15}{1*1/1 + 1*1/15 + 5*1/4} = 0.10$$

ALGORITHM

Initialize parameters

N = no. of ants

T = max. no. of iterations

α = influence of pheromone

β = influence of heuristic information

ρ = pheromone evaporation rate

Q = pheromone deposit constant

τ_{ij}: initial pheromone level

η_{ij}: heuristic information

For t = 1 to T

for each ant k = 1 to N

initialize ant's tour

for each step of the ant's tour

select the next node (j) based on transition probability

$P_{ij} = (\tau_{ij}^{\alpha}) * (\eta_{ij}^{\beta}) / \text{sum\_over\_allowed\_nodes}(\tau_{ik}^{\alpha} * \eta_{ik}^{\beta})$

choose the next node j using P_{ij}

favoring paths with more pheromone and better heuristic info

Move the ant to selected node

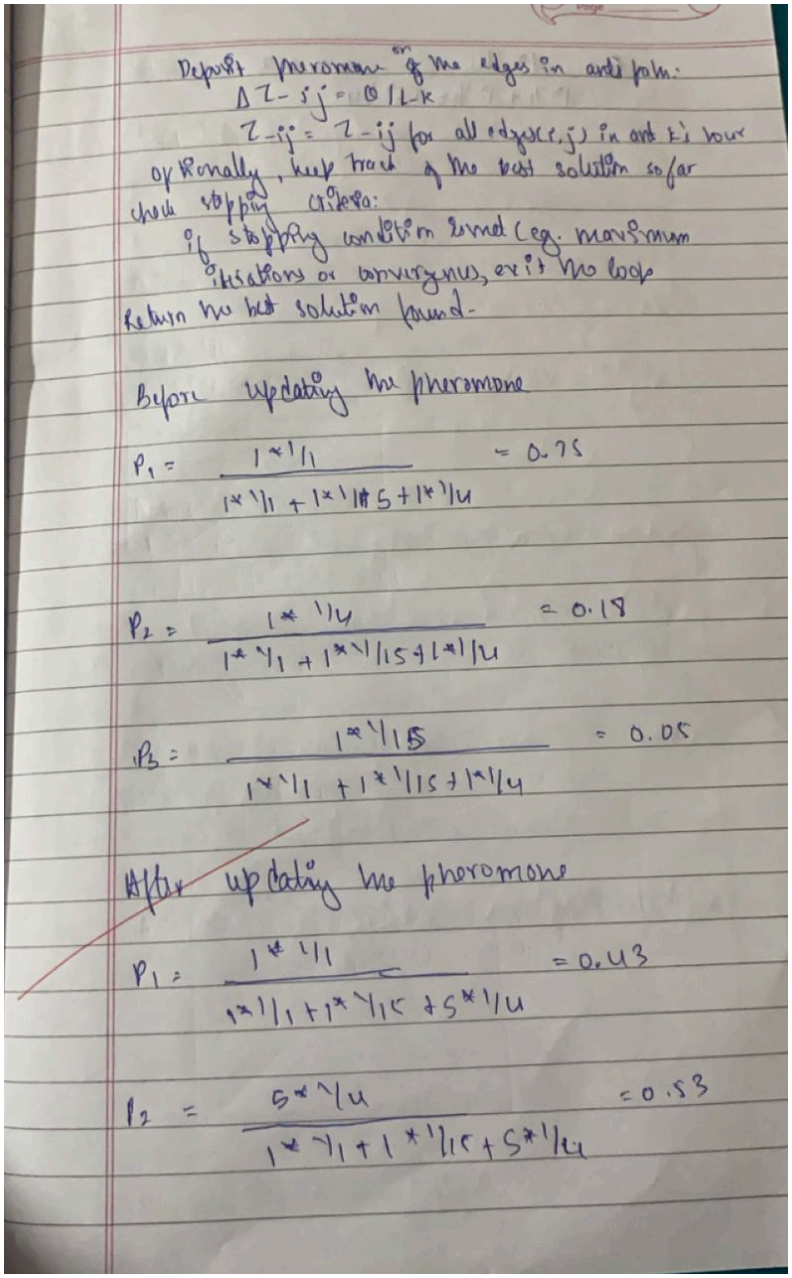
for each edge (i, j)

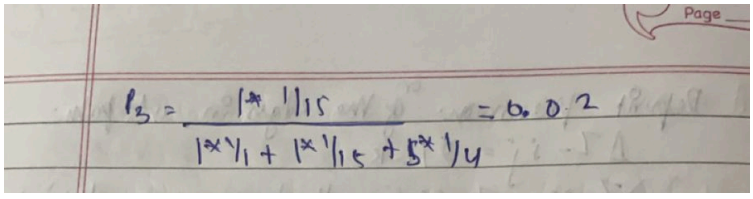
applying pheromone evaporation:

$\tau_{ij} = (1 - \rho) * \tau_{ij}$

for each ant k:

calculate the tour length L_k





Code:

```
import numpy as np
import random
class AntColony:
    def __init__(self, graph, n_ants, n_best, n_iterations, decay, alpha=1, beta=1):
        self.graph = graph
```

```
        self.n_ants = n_ants
        self.n_best = n_best
        self.n_iterations = n_iterations
        self.decay = decay
        self.alpha = alpha
        self.beta = beta
        self.pheromone = np.ones(self.graph.shape) / len(graph)
        self.nodes = ['A', 'B', 'C', 'D']
    def run(self):
        shortest_path = None
        shortest_distance = float('inf')
        for _ in range(self.n_iterations):
            all_paths = self.gen_all_paths()
            self.spread_pheronome(all_paths, shortest_path, shortest_distance)
            shortest_path, shortest_distance = self.best_path(all_paths)
        shortest_path = [(self.nodes[from_node], self.nodes[to_node]) for from_node, to_node in
shortest_path]
        return shortest_path, shortest_distance
    def spread_pheronome(self, all_paths, shortest_path, shortest_distance):
        for path, dist in all_paths:
            for from_node, to_node in path:
                self.pheromone[from_node][to_node] += 1.0 / dist
        self.pheromone *= self.decay
    def gen_path(self, start):
        path = []
        visited = set()
        visited.add(start)
        current = start
        while len(visited) < len(self.graph):
            next_node = self.pick_next_node(current, visited)
            path.append((current, next_node))
            visited.add(next_node)
            current = next_node
        path.append((current, start))
        return path
    def gen_all_paths(self):
        all_paths = []
        for ant in range(self.n_ants):
            path = self.gen_path(random.randint(0, len(self.graph)-1))
            distance = self.calculate_distance(path)
            all_paths.append((path, distance))
        return all_paths
    def calculate_distance(self, path):
        distance = 0
        for from_node, to_node in path:
            distance += self.graph[from_node][to_node]
```

```
        return distance
    def best_path(self, all_paths):
        best = min(all_paths, key=lambda x: x[1])
        return best
    def pick_next_node(self, current, visited):
        pheromone = np.copy(self.pheromone[current])
        pheromone[list(visited)] = 0
        attractiveness = np.copy(self.graph[current])
        attractiveness[list(visited)] = 0
        pheromone = pheromone ** self.alpha
        attractiveness = attractiveness ** self.beta
        prob = pheromone * attractiveness
        prob /= prob.sum()
        return np.random.choice(range(len(self.graph)), p=prob)
graph = np.array([
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
])
aco = AntColony(graph, n_ants=5, n_best=2, n_iterations=100, decay=0.95, alpha=1, beta=2)
shortest_path, shortest_distance = aco.run()
print("Shortest path (in terms of nodes): ", shortest_path)
print("Shortest distance: ", shortest_distance)
```

```python
import numpy as np
import random
class AntColony:
    def __init__(self, graph, n_ants, n_best, n_iterations, decay, alpha=1, beta=1):
        self.graph = graph
        self.n_ants = n_ants
        self.n_best = n_best
        self.n_iterations = n_iterations
        self.decay = decay
        self.alpha = alpha
        self.beta = beta
        self.pheromone = np.ones(self.graph.shape) / len(graph)
        self.nodes = ['A', 'B', 'C', 'D']
    def run(self):
        shortest_path = None
        shortest_distance = float('inf')
        for _ in range(self.n_iterations):
            all_paths = self.gen_all_paths()
            self.spread_pheronome(all_paths, shortest_path, shortest_distance)
            shortest_path, shortest_distance = self.best_path(all_paths)
        shortest_path = [(self.nodes[from_node], self.nodes[to_node]) for from_node, to_node in shortest_path]
        return shortest_path, shortest_distance
    def spread_pheronome(self, all_paths, shortest_path, shortest_distance):
        for path, dist in all_paths:
            for from_node, to_node in path:
                self.pheromone[from_node][to_node] += 1.0 / dist
        self.pheromone *= self.decay
    def gen_path(self, start):
        path = []
        visited = set()
        visited.add(start)
        current = start
        while len(visited) < len(self.graph):
            next_node = self.pick_next_node(current, visited)
            path.append((current, next_node))
            visited.add(next_node)
            current = next_node
        path.append((current, start))
        return path
    def gen_all_paths(self):
        all_paths = []
        for ant in range(self.n_ants):
            path = self.gen_path(random.randint(0, len(self.graph)-1))
            distance = self.calculate_distance(path)
            all_paths.append((path, distance))
        return all_paths
    def calculate_distance(self, path):
        distance = 0
        for from_node, to_node in path:
            distance += self.graph[from_node][to_node]
        return distance
    def best_path(self, all_paths):
        best = min(all_paths, key=lambda x: x[1])
        return best
```

```
        return best
    def pick_next_node(self, current, visited):
        pheromone = np.copy(self.pheromone[current])
        pheromone[list(visited)] = 0
        attractiveness = np.copy(self.graph[current])
        attractiveness[list(visited)] = 0
        pheromone = pheromone ** self.alpha
        attractiveness = attractiveness ** self.beta
        prob = pheromone * attractiveness
        prob /= prob.sum()
        return np.random.choice(range(len(self.graph)), p=prob)
graph = np.array([
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
])
aco = AntColony(graph, n_ants=5, n_best=2, n_iterations=100, decay=0.95, alpha=1, beta=2)
shortest_path, shortest_distance = aco.run()
print("Shortest path (in terms of nodes): ", shortest_path)
print("Shortest distance: ", shortest_distance)
```

Output:

```
Shortest path:  [(1, 2), (2, 3), (3, 0), (0, 1)]
Shortest distance:  95
```