**Ant Colony Optimization for Traveling Salesman Problem (TSP)**

**1. 📦 Importing Required Libraries**

python

CopyEdit

import random

import numpy as np

import matplotlib.pyplot as plt

* random → Random selections (for cities, ants).
* numpy → Efficient array and matrix operations (distance matrix, probabilities).
* matplotlib.pyplot → Plotting the final best tour.

**2. ⚙️ Defining Key Parameters**

python

CopyEdit

NUM\_CITIES = 10

NUM\_ANTS = 20

NUM\_ITERATIONS = 100

ALPHA = 1 # Importance of pheromone (trail strength)

BETA = 5 # Importance of distance (visibility)

EVAPORATION = 0.5 # Pheromone evaporation rate

Q = 100 # Pheromone deposit factor

* **NUM\_CITIES** → How many cities are there? (10)
* **NUM\_ANTS** → Number of ants running per generation (20).
* **NUM\_ITERATIONS** → Number of times ants explore (100 loops).
* **ALPHA** → How much ants trust *pheromones*.
* **BETA** → How much ants trust *shorter distances*.
* **EVAPORATION** → How quickly old pheromone trails fade.
* **Q** → How much pheromone an ant drops after a tour.

**3. 🗺️ Create the Cities and Distance Matrix**

python

CopyEdit

city\_coordinates = np.random.rand(NUM\_CITIES, 2) \* 100

* Random (x, y) coordinates between 0 and 100 for each city.

python

CopyEdit

distance\_matrix = np.zeros((NUM\_CITIES, NUM\_CITIES))

for i in range(NUM\_CITIES):

for j in range(NUM\_CITIES):

if i != j:

distance\_matrix[i][j] = np.linalg.norm(city\_coordinates[i] - city\_coordinates[j])

* For every pair of cities (i, j), calculate **Euclidean distance**.
* Distance between city and itself = 0.

**4. 🧠 ACO Helper Functions**

**4.1 Initialize Pheromone Levels**

python

CopyEdit

def initialize\_pheromone():

return np.ones((NUM\_CITIES, NUM\_CITIES))

* At the start, all paths have pheromone = 1.

**4.2 Probability of Choosing a City**

python

CopyEdit

def probability(i, j, pheromone, visited):

if j in visited:

return 0

pheromone\_val = pheromone[i][j] \*\* ALPHA

distance\_val = (1 / distance\_matrix[i][j]) \*\* BETA

return pheromone\_val \* distance\_val

* If city **j** already visited → probability = 0.
* Otherwise, combine:
  + **pheromone strength** between i and j
  + **inverse of distance** (closer is better).
* Apply **ALPHA** and **BETA** exponents.

**4.3 Select Next City Based on Probability**

python

CopyEdit

def select\_next\_city(current\_city, pheromone, visited):

probs = [probability(current\_city, j, pheromone, visited) for j in range(NUM\_CITIES)]

total = sum(probs)

if total == 0:

return random.choice([j for j in range(NUM\_CITIES) if j not in visited])

probs = [p / total for p in probs]

return np.random.choice(range(NUM\_CITIES), p=probs)

* If all probabilities = 0, randomly pick unvisited city.
* Otherwise:
  + Normalize probabilities.
  + Randomly **choose next city** based on the normalized probabilities.

**4.4 Calculate Total Distance of a Tour**

python

CopyEdit

def total\_distance(tour):

dist = 0

for i in range(NUM\_CITIES):

dist += distance\_matrix[tour[i]][tour[(i + 1) % NUM\_CITIES]]

return dist

* Add up distance from:
  + City 0 to 1, 1 to 2, ..., 9 back to 0 (full circle).

**4.5 Update Pheromone Trails**

python

CopyEdit

def update\_pheromone(pheromone, all\_tours):

pheromone \*= (1 - EVAPORATION)

for tour, dist in all\_tours:

for i in range(NUM\_CITIES):

from\_city = tour[i]

to\_city = tour[(i + 1) % NUM\_CITIES]

pheromone[from\_city][to\_city] += Q / dist

pheromone[to\_city][from\_city] += Q / dist

return pheromone

* **Evaporate** all pheromones first (multiply by 1 - EVAPORATION).
* For each ant:
  + Add **new pheromone** on the path based on Q / total\_distance.
  + Better tours (lower distance) → deposit more pheromones.

**5. 🚀 Main Ant Colony Optimization Loop**

python

CopyEdit

def ant\_colony\_optimization():

pheromone = initialize\_pheromone()

best\_tour = None

best\_distance = float('inf')

* Start with fresh pheromone matrix.
* Track best tour found.

**Main Loop:**

python

CopyEdit

for iteration in range(NUM\_ITERATIONS):

all\_tours = []

for ant in range(NUM\_ANTS):

visited = []

current\_city = random.randint(0, NUM\_CITIES - 1)

visited.append(current\_city)

* Each ant starts at a random city.

python

CopyEdit

while len(visited) < NUM\_CITIES:

next\_city = select\_next\_city(current\_city, pheromone, visited)

visited.append(next\_city)

current\_city = next\_city

* Ant **builds full tour** by probabilistically choosing next cities.

python

CopyEdit

dist = total\_distance(visited)

all\_tours.append((visited, dist))

* Calculate the full tour distance.

python

CopyEdit

if dist < best\_distance:

best\_tour = visited

best\_distance = dist

* Update if a new **shorter tour** is found.

python

CopyEdit

pheromone = update\_pheromone(pheromone, all\_tours)

print(f"Iteration {iteration+1}, Best Distance: {best\_distance:.2f}")

* After all ants finish → **update pheromones** based on all tours.
* Print best distance after each iteration.

**Final Return**

python

CopyEdit

return best\_tour, best\_distance

* Return the best tour and its distance found.

**6. 📊 Plotting the Best Tour**

python

CopyEdit

def plot\_tour(tour):

tour\_coords = [city\_coordinates[i] for i in tour] + [city\_coordinates[tour[0]]]

x, y = zip(\*tour\_coords)

plt.figure(figsize=(10, 6))

plt.plot(x, y, 'bo-')

plt.title("Best TSP Tour Found using ACO")

for i, (x\_i, y\_i) in enumerate(city\_coordinates):

plt.text(x\_i + 1, y\_i + 1, f"{i}", fontsize=12)

plt.xlabel("X")

plt.ylabel("Y")

plt.grid(True)

plt.show()

* **Plot cities** and connect them according to the tour.
* Shows how ants have minimized the path visually.

**7. 🏁 Final Execution**

python

CopyEdit

best\_tour, best\_distance = ant\_colony\_optimization()

print("\nBest Tour:", best\_tour)

print("Best Distance:", round(best\_distance, 2))

plot\_tour(best\_tour)

**Important Theory: Ant Colony Optimization (ACO) for TSP**

**1. What is ACO?**

* **ACO (Ant Colony Optimization)** is a **nature-inspired** optimization algorithm.
* It mimics the **foraging behavior of real ants** who find the shortest path between food and nest **using pheromones**.
* Artificial "ants" **cooperate** and **share knowledge** (via pheromone trails) to **find optimal solutions** to combinatorial problems like TSP.

**2. Why ACO for TSP?**

* The **Traveling Salesman Problem (TSP)** asks:

Find the **shortest possible tour** that visits each city exactly once and returns to the starting city.

* TSP is **NP-hard**, meaning it’s computationally expensive to solve exactly for large numbers of cities.
* ACO provides a **heuristic**, **probabilistic**, **collaborative** approach to find **near-optimal** tours efficiently.

**3. Key Concepts in ACO**

| **Concept** | **Explanation** |
| --- | --- |
| **Ants** | Agents that build solutions (tours) step-by-step. |
| **Pheromone Trails** | Virtual "smell" laid on paths; stronger pheromone → more attractive path. |
| **Heuristic Information** | Typically, **inverse of distance** between cities (closer cities are preferred). |
| **Pheromone Update** | After all ants complete tours, pheromones are evaporated (reduced) and reinforced (increased) based on tour quality. |
| **Evaporation** | Prevents unlimited accumulation of pheromone, helping exploration. |
| **Probabilistic Choice** | Next city is chosen based on a probability combining pheromone strength and heuristic closeness. |

**4. Mathematical Formulations**

1. **Probability of Moving from City i to City j**:

Pij={(pheromoneij)α×(1distanceij)β∑k∉visited(pheromoneik)α×(1distanceik)βif j∉visited0otherwiseP\_{ij} = \begin{cases} \frac{(\text{pheromone}\_{ij})^\alpha \times (\frac{1}{\text{distance}\_{ij}})^\beta}{\sum\limits\_{k \notin \text{visited}} (\text{pheromone}\_{ik})^\alpha \times (\frac{1}{\text{distance}\_{ik}})^\beta} & \text{if } j \notin \text{visited} \\ 0 & \text{otherwise} \end{cases}Pij​=⎩⎨⎧​k∈/visited∑​(pheromoneik​)α×(distanceik​1​)β(pheromoneij​)α×(distanceij​1​)β​0​if j∈/visitedotherwise​

Where:

* α\alphaα controls **pheromone influence**.
* β\betaβ controls **heuristic (distance) influence**.

1. **Pheromone Update Rule**:

pheromoneij=(1−evaporation)×pheromoneij+∑antsQdistance traveled\text{pheromone}\_{ij} = (1 - \text{evaporation}) \times \text{pheromone}\_{ij} + \sum\_{\text{ants}} \frac{Q}{\text{distance traveled}}pheromoneij​=(1−evaporation)×pheromoneij​+ants∑​distance traveledQ​

Where:

* **Q** is the amount of pheromone an ant deposits.
* **Evaporation** prevents stagnation (keeps ants exploring new paths).

**5. Parameters Explained**

| **Parameter** | **Meaning** | **Effect** |
| --- | --- | --- |
| NUM\_CITIES | Number of cities in the TSP | Size of problem |
| NUM\_ANTS | Number of ants per iteration | More ants → more exploration |
| NUM\_ITERATIONS | How long to run ACO | More iterations → better solution |
| ALPHA | Weight of pheromone in decision-making | Higher → ants favor pheromone trails more |
| BETA | Weight of heuristic (distance) in decision-making | Higher → ants prefer closer cities |
| EVAPORATION | Pheromone evaporation rate | Controls memory of system |
| Q | Pheromone deposit factor | Bigger Q → stronger reinforcement of good paths |

**6. Step-by-Step Working of ACO for TSP**

1. **Initialization**:
   * Randomly place pheromones equally on all paths.
2. **Each Iteration**:
   * Each ant builds a tour:
     + Starts at a random city.
     + Repeatedly chooses the next city based on probability (pheromone and distance).
   * After all ants complete tours:
     + Update pheromone matrix:
       - **Evaporate** some pheromone.
       - **Add** pheromone on paths ants traveled, stronger for shorter tours.
   * Keep track of the **best tour found** so far.
3. **After All Iterations**:
   * Output the **best tour** and its **total distance**.

**7. Strengths and Limitations**

**Strengths:**

* Good for **large search spaces**.
* **Distributed** (parallelizable).
* **Adaptable** to dynamic environments.

**Limitations:**

* May **converge prematurely** (all ants follow the same path too early).
* Needs careful **parameter tuning**.

**8. Real-Life Inspiration**

* Real ants drop **pheromone** on paths to food sources.
* Over time, **shorter paths** are **reinforced** more because they are traveled more frequently.
* ACO captures this **emergent collective intelligence** behavior.