```
!git checkout -b main
```

```
fatal: A branch named 'main' already exists.
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remo
```

```
import os

# path in your Google Drive where your repo will be stored
project_root = "/content/drive/MyDrive/metaheuristics-colab"
os.makedirs(project_root, exist_ok=True)

# move into that folder
os.chdir(project_root)

# clone my repo
!git clone https://github.com/reyamm/Metaheuristic-Optimization-for

# move inside the repo folder
os.chdir("/content/drive/MyDrive/metaheuristics-colab/Metaheuristic

!pwd
!ls
```

```
fatal: destination path 'Metaheuristic-Optimization-for-a-Real-Worlc
/content/drive/MyDrive/metaheuristics-colab/Metaheuristic-Optimizati
 eil76.tsp  ' Metaheuristic-Optimization-Proj'   README.md
```

```
from getpass import getpass
import datetime

# basic info for my repo on github
github_username = "reyamm"
repo_name = "Metaheuristic-Optimization-for-a-Real-World-Problem"
branch_name = "main"

# i type the token once and colab keeps it in memory only
github_token = getpass("enter my github token (not shown): ")

def git_commit_and_push(message=None):
    # small helper to add changes, commit and push in one go
    if message is None:
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%
        message = f"colab auto commit {timestamp}"

    print("adding files...")
    !git add .

    print("committing...")
    !git commit -m "{message}" || echo "no changes to commit."

    print("pushing to github...")
    !git push https://{github_username}:{github_token}@github.com/{

    print("done.")
```

```
enter my github token (not shown): ··········
```

```
# creating a simple README file
with open("README.md", "w") as f:
    f.write("# Metaheuristic Optimization Project\nThis repo is sync

# check that the file exists
!ls -l
```

```
total 19
-rw------- 1 root root   801 Dec  4 23:39  eil76.tsp
-rw------- 1 root root 17816 Dec  5 00:07 ' Metaheuristic-Optimizati
-rw------- 1 root root    75 Dec  5 00:07  README.md
```

```
git_commit_and_push("Initial test push from Colab")
```

```
Adding files...
Committing...
[main 55196d0] Initial test push from Colab
 1 file changed, 1 insertion(+), 1 deletion(-)
 rewrite  Metaheuristic-Optimization-Proj (99%)
Pushing to GitHub...
To https://github.com/reyamm/Metaheuristic-Optimization-for-a-Real-W
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/reyamm/Metahe
hint: Updates were rejected because the remote contains work that yo
hint: not have locally. This is usually caused by another repository
hint: to the same ref. You may want to first integrate the remote ch
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for de
Done.
```

```
os.chdir("/content/drive/MyDrive/metaheuristics-colab/Metaheuristic
!pwd
```

```
/content/drive/MyDrive/metaheuristics-colab/Metaheuristic-Optimizati
```

```
import math
import random
import numpy as np
import matplotlib.pyplot as plt

random.seed(42)
np.random.seed(42)

print("Imports OK")
```

```
Imports OK
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files eil76.tsp

- **eil76.tsp**(n/a) - 801 bytes, last modified: n/a - 100% done

```
Saving eil76.tsp to eil76 (1).tsp
```

```
!ls
```

```
'eil76 (1).tsp'   eil76.tsp  ' Metaheuristic-Optimization-Proj'   RE
```

```
!mv eil76.tsp "/content/drive/MyDrive/metaheuristics-colab/Metaheur
```

```
mv: 'eil76.tsp' and '/content/drive/MyDrive/metaheuristics-colab/Met
```

```
!ls "/content/drive/MyDrive/metaheuristics-colab/Metaheuristic-Opti
```

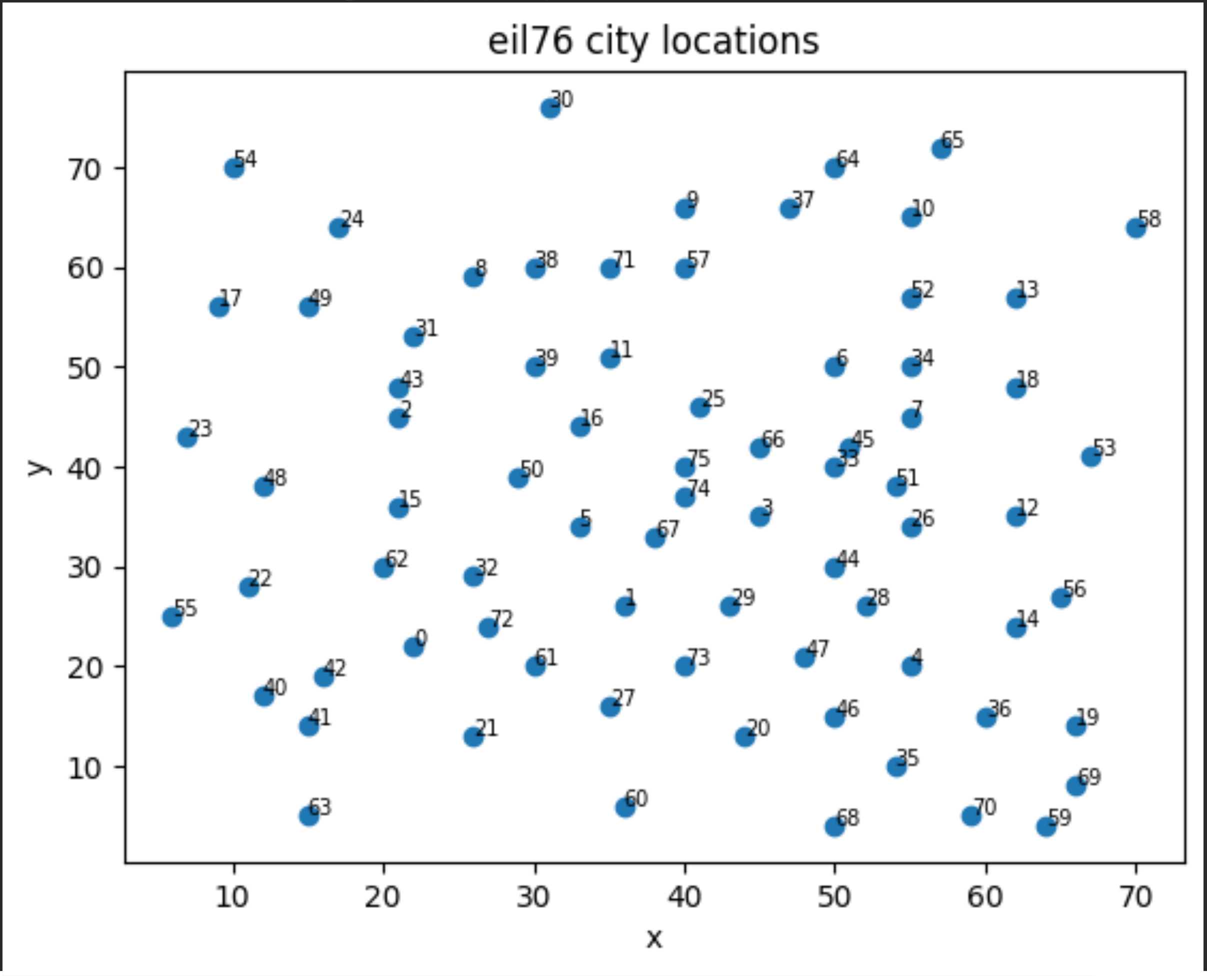
```
'eil76 (1).tsp'   eil76.tsp  ' Metaheuristic-Optimization-Proj'   RE
```

```
# make sure we are inside the repo folder
os.chdir("/content/drive/MyDrive/metaheuristics-colab/Metaheuristic

# load eil76.tsp (tsplib euclidean tsp) and return the city coordir
def load_tsplib_euc2d(path):
    """returns a list of (x, y) coords for each city"""
    coords = []
    with open(path, "r") as f:
        lines = f.readlines()

    node_section = False
    for line in lines:
        line = line.strip()
        if line == "" or line.upper().startswith("EOF"):
            continue
        if line.upper().startswith("NODE_COORD_SECTION"):
            node_section = True
            continue
        if not node_section:
            continue

        parts = line.split()
        if len(parts) >= 3:
            # format: index  x  y
            _, x, y = parts[:3]
            coords.append((float(x), float(y)))

    return coords

# function to build the distance matrix
def compute_distance_matrix(coords):
    """build a full distance matrix using rounded euclidean distanc
    n = len(coords)
    dist = np.zeros((n, n))
    for i in range(n):
        x1, y1 = coords[i]
        for j in range(n):
            if i == j:
                continue
            x2, y2 = coords[j]
            dist[i, j] = round(math.hypot(x1 - x2, y1 - y2))
    return dist

# tour length helper (used in both ga and aco)
def tour_length(tour, dist_matrix):
    total = 0.0
    for i in range(len(tour)):
        j = (i + 1) % len(tour)
        total += dist_matrix[tour[i], tour[j]]
    return total

# simple plot helper to visualize the cities
def plot_cities(coords):
    xs = [c[0] for c in coords]
    ys = [c[1] for c in coords]
    plt.figure()
    plt.scatter(xs, ys)
    for i, (x, y) in enumerate(coords):
        plt.text(x, y, str(i), fontsize=7)
    plt.title("eil76 city locations")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.show()

# simple plot helper to visualize a tour
def plot_tour(tour, coords, title="best tour"):
    xs = [coords[i][0] for i in tour] + [coords[tour[0]][0]]
    ys = [coords[i][1] for i in tour] + [coords[tour[0]][1]]
    plt.figure()
    plt.plot(xs, ys, marker="o")
    plt.title(title)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.show()

print("setup cell ran correctly")
```

```
setup cell ran correctly
```

```
# check that the file is really here
!ls

coords = load_tsplib_euc2d("eil76.tsp")
distance_matrix = compute_distance_matrix(coords)

print(f"cities: {len(coords)}")
print("distance matrix shape:", distance_matrix.shape)

# visualize the cities
plot_cities(coords)
```

```
'eil76 (1).tsp'   eil76.tsp   ' Metaheuristic-Optimization-Proj'   RE
cities: 76
distance matrix shape: (76, 76)
```



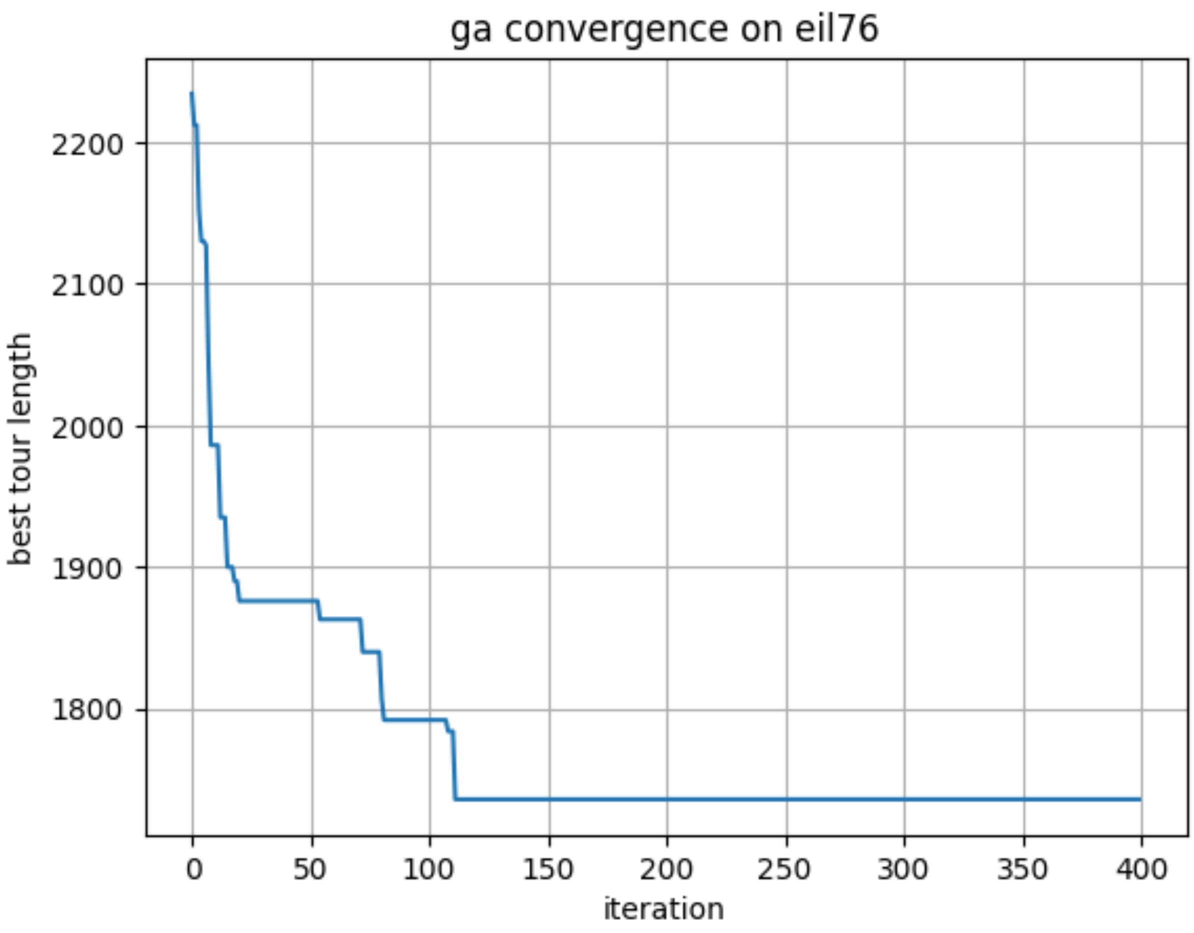
```
# genetic algorithm for tsp

num_cities = len(coords)

def create_random_tour():
    # make a random permutation of all cities
    tour = list(range(num_cities))
```

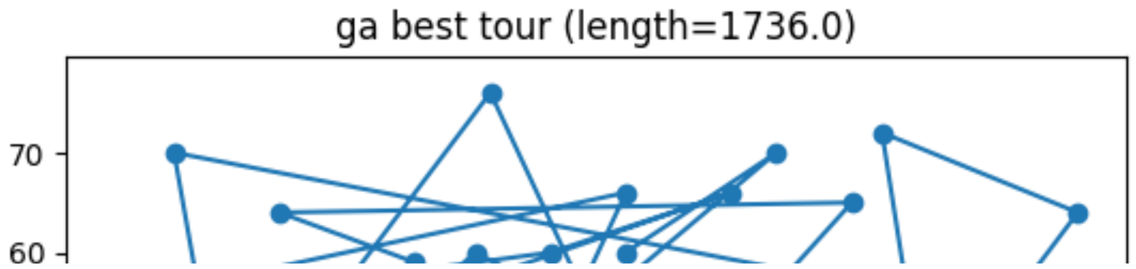
```
        random.shuffle(tour)
        return tour

    def initial_population(pop_size):
        # create the first generation of random tours
        return [create_random_tour() for _ in range(pop_size)]

    def tournament_selection(population, fitnesses, k=3):
        # pick k random individuals and return the best one
        indices = random.sample(range(len(population)), k)
        best_idx = min(indices, key=lambda idx: fitnesses[idx])
        return population[best_idx]

    def ordered_crossover(parent1, parent2):
        # ordered crossover (ox) keeps the relative order of cities
        n = len(parent1)
        a, b = sorted(random.sample(range(n), 2))
        child = [None] * n

        # copy a slice from parent1
        child[a:b+1] = parent1[a:b+1]

        # fill the remaining positions with cities from parent2 in orde
        p2_idx = 0
        for i in range(n):
            if child[i] is not None:
                continue
            while parent2[p2_idx] in child:
                p2_idx += 1
            child[i] = parent2[p2_idx]
        return child

    def swap_mutation(tour, mutation_rate):
        # randomly swap cities with a given probability
        tour = tour[:]  # copy
        for i in range(len(tour)):
            if random.random() < mutation_rate:
                j = random.randint(0, len(tour) - 1)
                tour[i], tour[j] = tour[j], tour[i]
        return tour

    def run_ga(
        dist_matrix,
        pop_size=80,
        generations=400,
        crossover_rate=0.9,
        mutation_rate=0.03,
        tournament_k=3
```

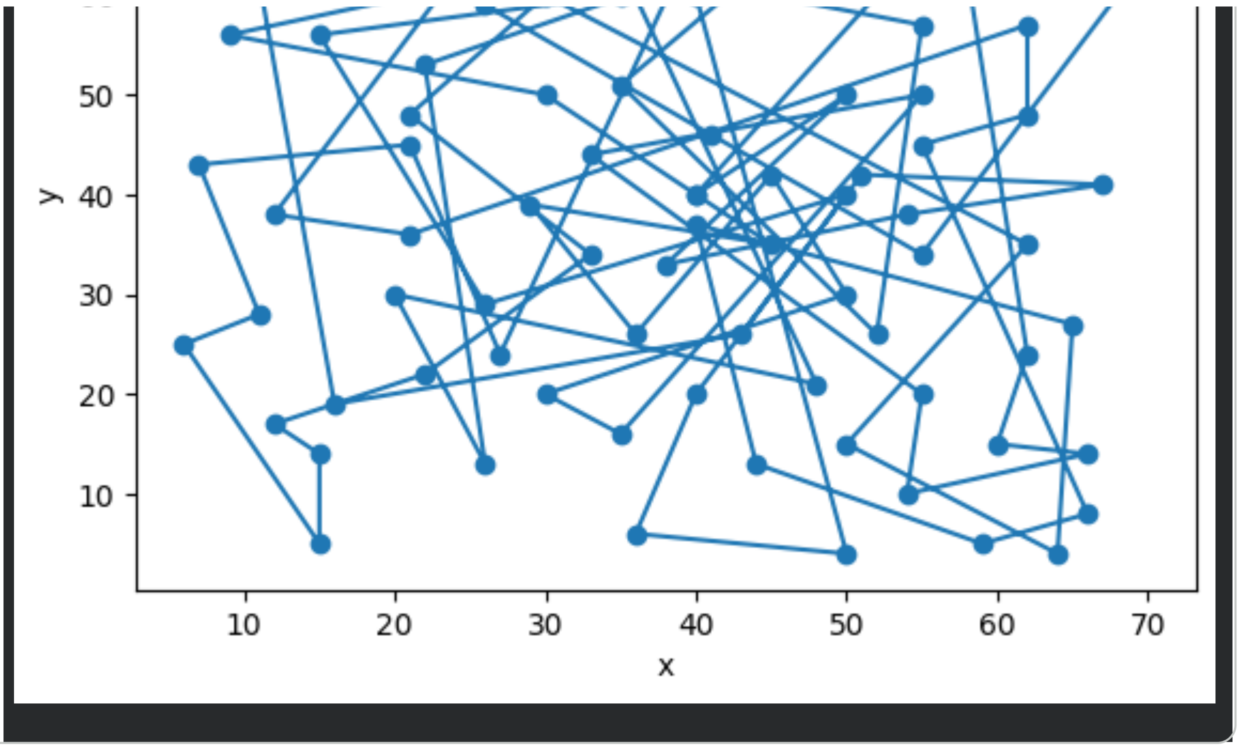
```
):
    # make initial population
    population = initial_population(pop_size)
    fitnesses = [tour_length(ind, dist_matrix) for ind in populatic

    best_lengths = []
    best_tour = None
    best_len = float("inf")

    for gen in range(generations):
        # update current best
        for i, ind in enumerate(population):
            if fitnesses[i] < best_len:
                best_len = fitnesses[i]
                best_tour = ind[:]

        best_lengths.append(best_len)

        # build next generation
        new_population = []
        while len(new_population) < pop_size:
            parent1 = tournament_selection(population, fitnesses, k
            parent2 = tournament_selection(population, fitnesses, k

            if random.random() < crossover_rate:
                child = ordered_crossover(parent1, parent2)
            else:
                child = parent1[:]

            child = swap_mutation(child, mutation_rate)
            new_population.append(child)

        population = new_population
        fitnesses = [tour_length(ind, dist_matrix) for ind in popul

        if gen % 50 == 0 or gen == generations - 1:
            print(f"generation {gen+1}/{generations} - best length

    return best_tour, best_len, best_lengths

best_tour_ga, best_len_ga, history_ga = run_ga(distance_matrix)
print("ga best length:", best_len_ga)
```

```
generation 1/400 - best length so far: 2234.0
generation 51/400 - best length so far: 1876.0
generation 101/400 - best length so far: 1792.0
generation 151/400 - best length so far: 1736.0
generation 201/400 - best length so far: 1736.0
generation 251/400 - best length so far: 1736.0
generation 301/400 - best length so far: 1736.0
generation 351/400 - best length so far: 1736.0
generation 400/400 - best length so far: 1736.0
ga best length: 1736.0
```

```python
def plot_convergence(history, title="convergence"):
    plt.figure()
    plt.plot(history)
    plt.xlabel("iteration")
    plt.ylabel("best tour length")
    plt.title(title)
    plt.grid(True)
    plt.show()

plot_convergence(history_ga, "ga convergence on eil76")
plot_tour(best_tour_ga, coords, f"ga best tour (length={best_len_ga
```





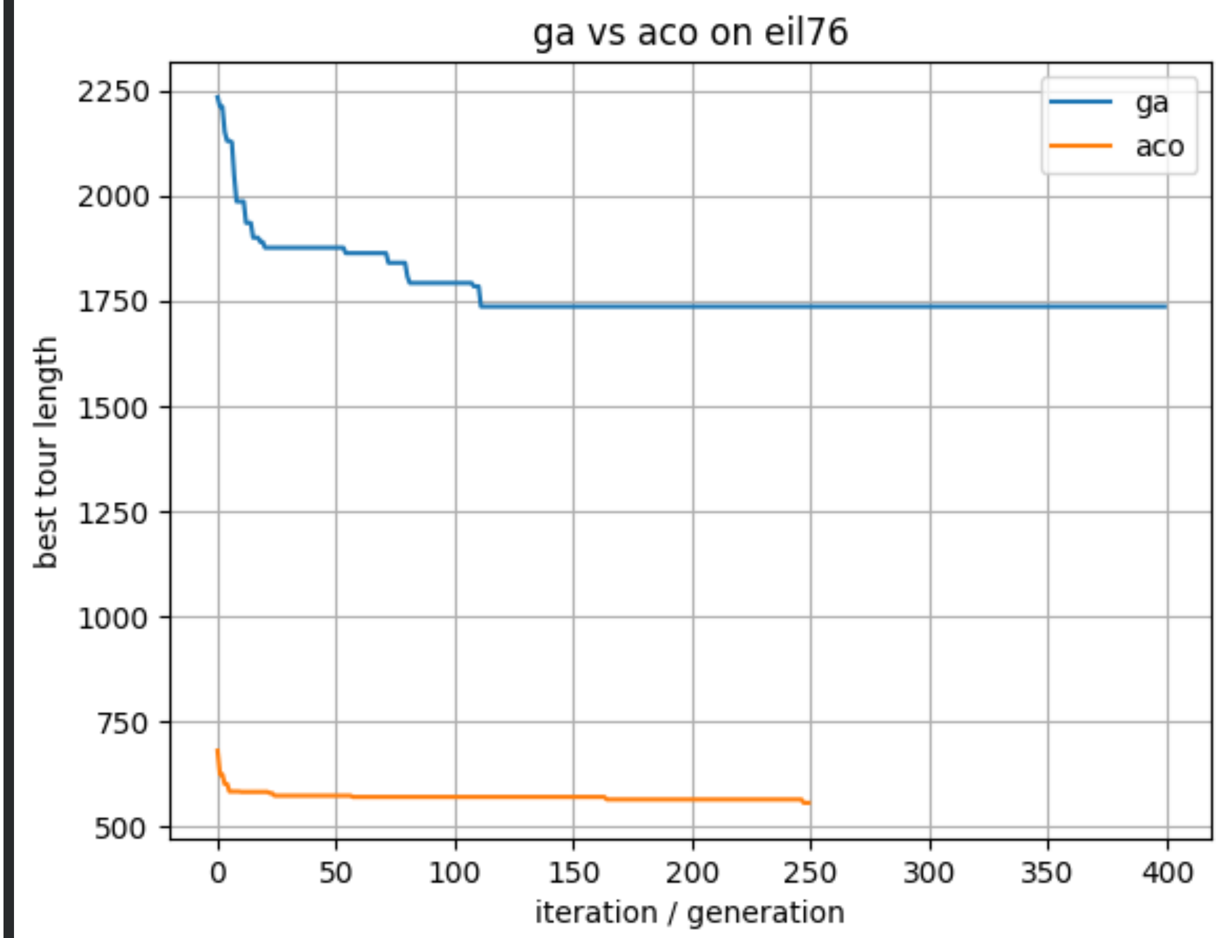


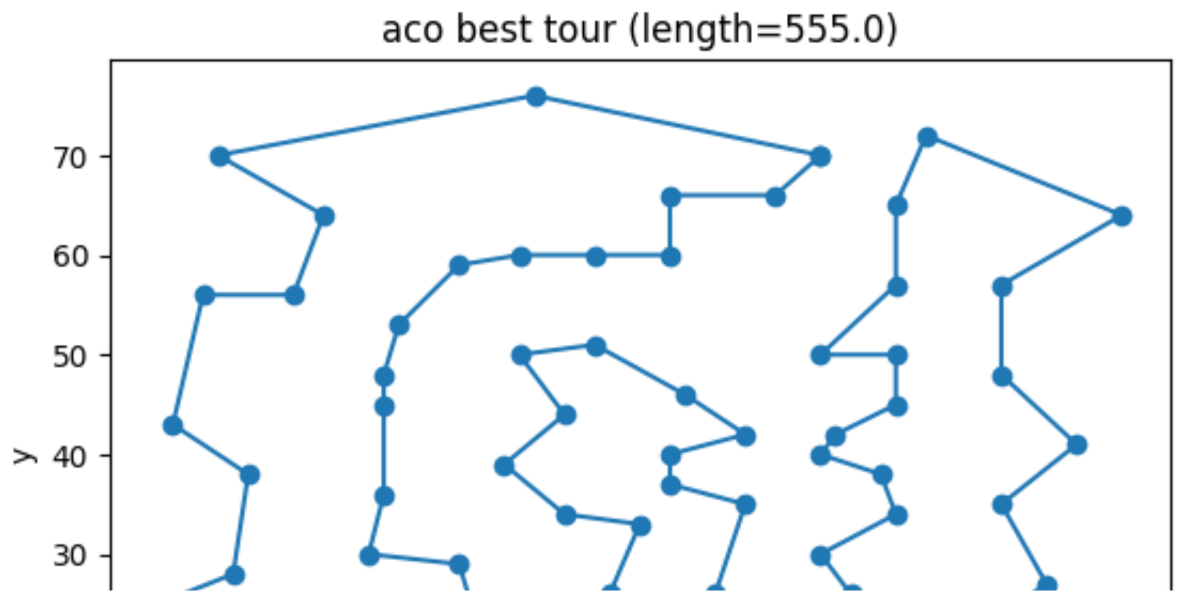
```
# ant colony optimization for tsp

num_cities = len(coords)

def construct_ant_tour(dist_matrix, pheromone, alpha=1.0, beta=5.0)
    """
    builds a tour for one ant using the probabilistic transition ru
    """
    n = dist_matrix.shape[0]
    start_city = random.randint(0, n - 1)
    tour = [start_city]
    unvisited = set(range(n))
    unvisited.remove(start_city)

    while unvisited:
        i = tour[-1]

        # compute probabilities for next city
        probs = []
        for j in unvisited:
            tau_ij = pheromone[i, j] ** alpha
            eta_ij = (1.0 / (dist_matrix[i, j] + 1e-9)) ** beta
            probs.append((j, tau_ij * eta_ij))

        total = sum(p for _, p in probs)
        if total == 0:
            next_city = random.choice(list(unvisited))
        else:
            r = random.random() * total
```

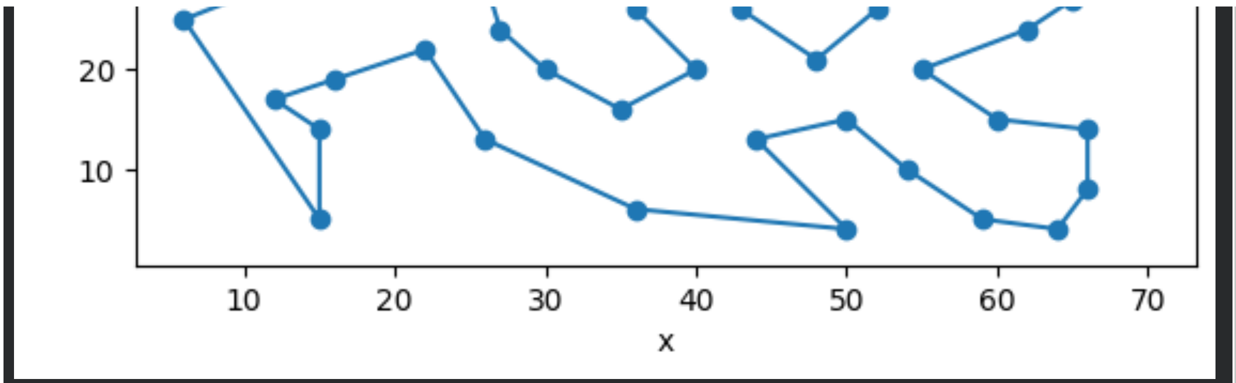
```
                cum = 0.0
                next_city = None
                for j, p in probs:
                    cum += p
                    if r <= cum:
                        next_city = j
                        break

            tour.append(next_city)
            unvisited.remove(next_city)

        return tour

    def update_pheromones(pheromone, ant_tours, ant_lengths, rho=0.5, Q
        """
        evaporate old pheromone and add new pheromone from ants
        """
        # evaporation
        pheromone *= (1.0 - rho)

        # deposit pheromones from each ant
        for tour, L in zip(ant_tours, ant_lengths):
            deposit = Q / L
            for i in range(len(tour)):
                j = (i + 1) % len(tour)
                a = tour[i]
                b = tour[j]
                pheromone[a, b] += deposit
                pheromone[b, a] += deposit  # symmetric tsp

    def run_aco(
        dist_matrix,
        num_ants=30,
        iterations=250,
        alpha=1.0,
        beta=5.0,
        rho=0.5,
        Q=100.0,
    ):
        """
        main aco loop: build tours, update pheromones, track best tour
        """
        n = dist_matrix.shape[0]
        # initial pheromone level based on average distance
        tau0 = 1.0 / (n * np.mean(dist_matrix[dist_matrix > 0]))
        pheromone = np.full((n, n), tau0)

        best_tour = None
        best_len = float("inf")
        best_lengths = []

        for it in range(iterations):
            ant_tours = []
            ant_lengths = []

            # each ant builds a tour
            for _ in range(num_ants):
                tour = construct_ant_tour(dist_matrix, pheromone, alpha
                L = tour_length(tour, dist_matrix)
                ant_tours.append(tour)
                ant_lengths.append(L)

                if L < best_len:
                    best_len = L
                    best_tour = tour[:]

            best_lengths.append(best_len)

            # update pheromone trails
            update_pheromones(pheromone, ant_tours, ant_lengths, rho=rh

            if it % 25 == 0 or it == iterations - 1:
                print(f"iteration {it+1}/{iterations} - best length so

        return best_tour, best_len, best_lengths

    best_tour_aco, best_len_aco, history_aco = run_aco(distance_matrix)
    print("aco best length:", best_len_aco)
```

```
iteration 1/250 - best length so far: 679.0
iteration 26/250 - best length so far: 572.0
iteration 51/250 - best length so far: 572.0
iteration 76/250 - best length so far: 569.0
iteration 101/250 - best length so far: 569.0
iteration 126/250 - best length so far: 569.0
iteration 151/250 - best length so far: 569.0
iteration 176/250 - best length so far: 563.0
iteration 201/250 - best length so far: 563.0
iteration 226/250 - best length so far: 563.0
iteration 250/250 - best length so far: 555.0
aco best length: 555.0
```

```
plt.figure()
plt.plot(history_ga, label="ga")
plt.plot(history_aco, label="aco")
plt.xlabel("iteration / generation")
plt.ylabel("best tour length")
plt.title("ga vs aco on eil76")
plt.legend()
plt.grid(True)
plt.show()

plot_tour(best_tour_aco, coords, f"aco best tour (length={best_len_

print("final ga best:", best_len_ga)
print("final aco best:", best_len_aco)
```







```
final ga best: 1736.0
final aco best: 555.0
```

```
!git push https://reyamm:{github_token}@github.com/reyamm/Metaheuris
```

```
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 2 threads
Compressing objects: 100% (17/17), done.
Writing objects: 100% (18/18), 240.90 KiB | 3.13 MiB/s, done.
Total 18 (delta 6), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (6/6), done.
To https://github.com/reyamm/Metaheuristic-Optimization-for-a-Real-W
 + 3728cec...9716d16 main -> main (forced update)
```