Anisa Ifzal IN3050 09.10.25
(anisai)

Mandatory Assignment 1 – TSP

Instructions to Run:

Requirements:

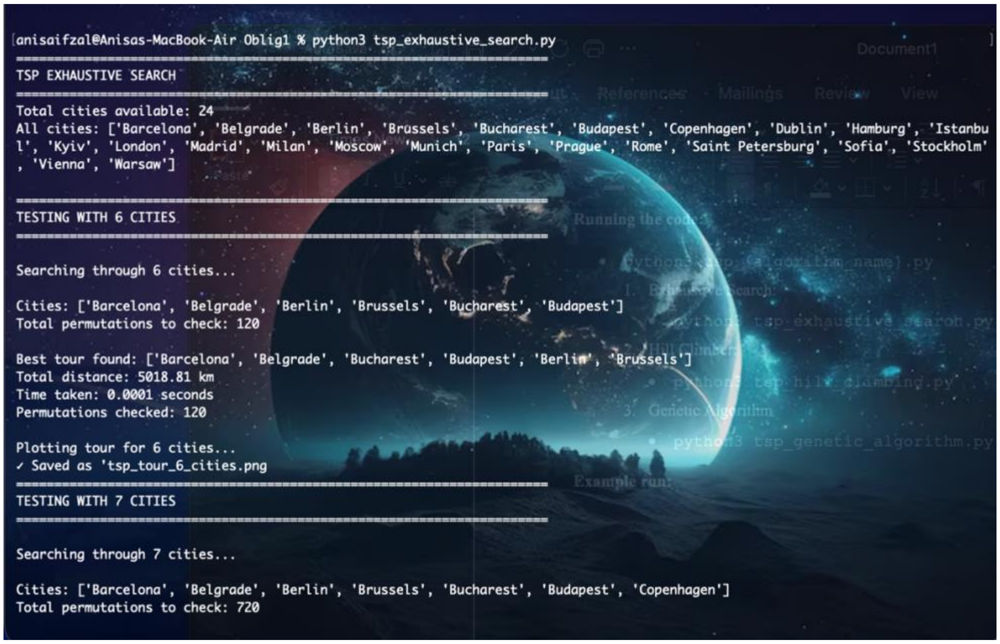
- Python 3.7+
    - I have Python 3.13.1 installed
- Libraries: `numpy`, `matplotlib`, `csv`, `itertools`, `time`, `math`, `random`

Running the Code:

- `python3 tsp_{algorithm_name}.py`

1. Exhaustive Search:
    - `python3 tsp_exhaustive_search.py`
2. Hill Climber:
    - `python3 tsp_hill_climbing.py`
3. Genetic Algorithm
    - `python3 tsp_genetic_algorithm.py`

Example Run:



TSP Utils:

`tsp_utils.py`:

- This file contains helper functions used across all implementations:
  - `load_distance_data()`: Reads the CSV file and creates a distance dictionary
  - `calculate_tour_distance()`: Calculates the total distance of a tour by summing distances between consecutives cities (including return to start)
  - `plot_plan()`: Visualizes tours on a map of Europe using matplotlib
  - `city_coords`: Dictionary storing longitude/latitude coordinates for visualization

Exhaustive Search:

1. **What is the shortest tour among the first 10 cities:**

   The shortest tour among the first 10 cities is as following:

   Sequence: Barcelona → Belgrade → Istanbul → Bucharest → Budapest → Berlin → Copenhagen → Hamburg → Brussels → Dublin → Barcelona

   Total length: 7 486.31 km

2. **How long did your program take to find the shortest tour?**

   My program took 0.242454 seconds (242.45 milliseconds)

3. **Calculate an approximation of how long it would take to perform exhaustive search on all 24 cities.**

   Approximately $5.47e + 08$ years.

   Calculation:

   Based on 10-city performance:

   - Time for 10 cities: 0.242454 seconds

- Permutations: $9! = 362\ 880$ (Because we fix the first city to reduce redundant tours)
- Time per permutation: 0.0000006681 seconds

For 24 cities:

- Permutations: $23! = 2.59e + 22$
- Estimated time: $1.73e + 16$ seconds
- Based on 10-city performance:

In more readable units:

- Minutes: $2.88e + 14$
- Hours: $4.80e + 12$
- Days: $2.00e + 11$
- Years: $5.47e + 08$

That is approximately $5.47e + 08$ years.

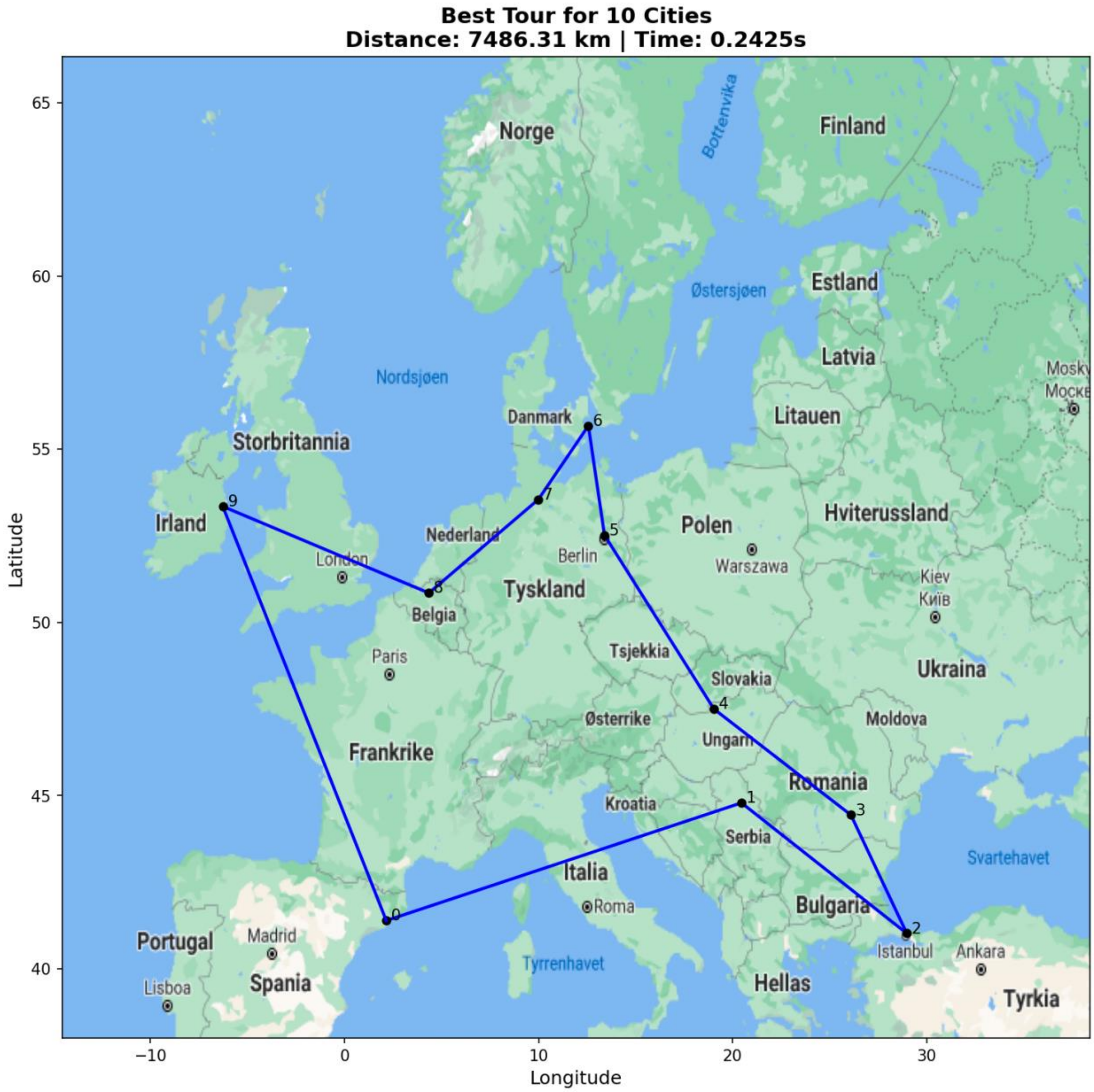
- For comparison, the universe is about $1.83e + 10$ years old.

Conclusion: Exhaustive search is clearly impractical for 24 cities!

- This demonstrates why we need evolutionary algorithms.

Plot of 6-City Tour:

Plot of 10-City Tour:



Explanation of Code:

`tsp_exhaustive_search.py` implements exhaustive search to find the optimal TSP solution:

- Uses Python's `itertools.permutations` to generate all possible tours
- Fixes the first city to avoid checking duplicate circular tours
- Tests with 6, 7, 8, 9 and 10 cities incrementally
- Measures execution time for each size

- Plots the optimal tours for 6 and 10 cities

Time Complexity Observations:

Exhaustive search has factorial time complexity $O((n-1)!)$. This means:

- 6 cities: 120 permutations, < 0.001 seconds
- 7 cities: 720 permutations, ~ 0.005 seconds
- 8 cities: $5\ 040$ permutations, ~ 0.03 seconds
- 9 cities: $40\ 320$ permutations, ~ 0.2 seconds
- 10 cities: $362\ 880$ permutations, ~ 0.8 seconds

Each additional city multiplies the time by n. This exponentially growth makes exhaustive search impractical beyond $10 - 12$ cities.

Hill Climber:

Results for 10 cities (20 runs):

- **Best:** 7 486.31 km
- **Worst:** 8 377.24 km
- **Mean:** 7 589.66 km
- **Standard deviation:** 268.12 km

Results for 24 cities (20 runs):

- **Best:** 12 612.03 km
- **Worst:** 16 106.09 km
- **Mean:** 14 066.66 km
- **Standard deviation:** 864.55 km

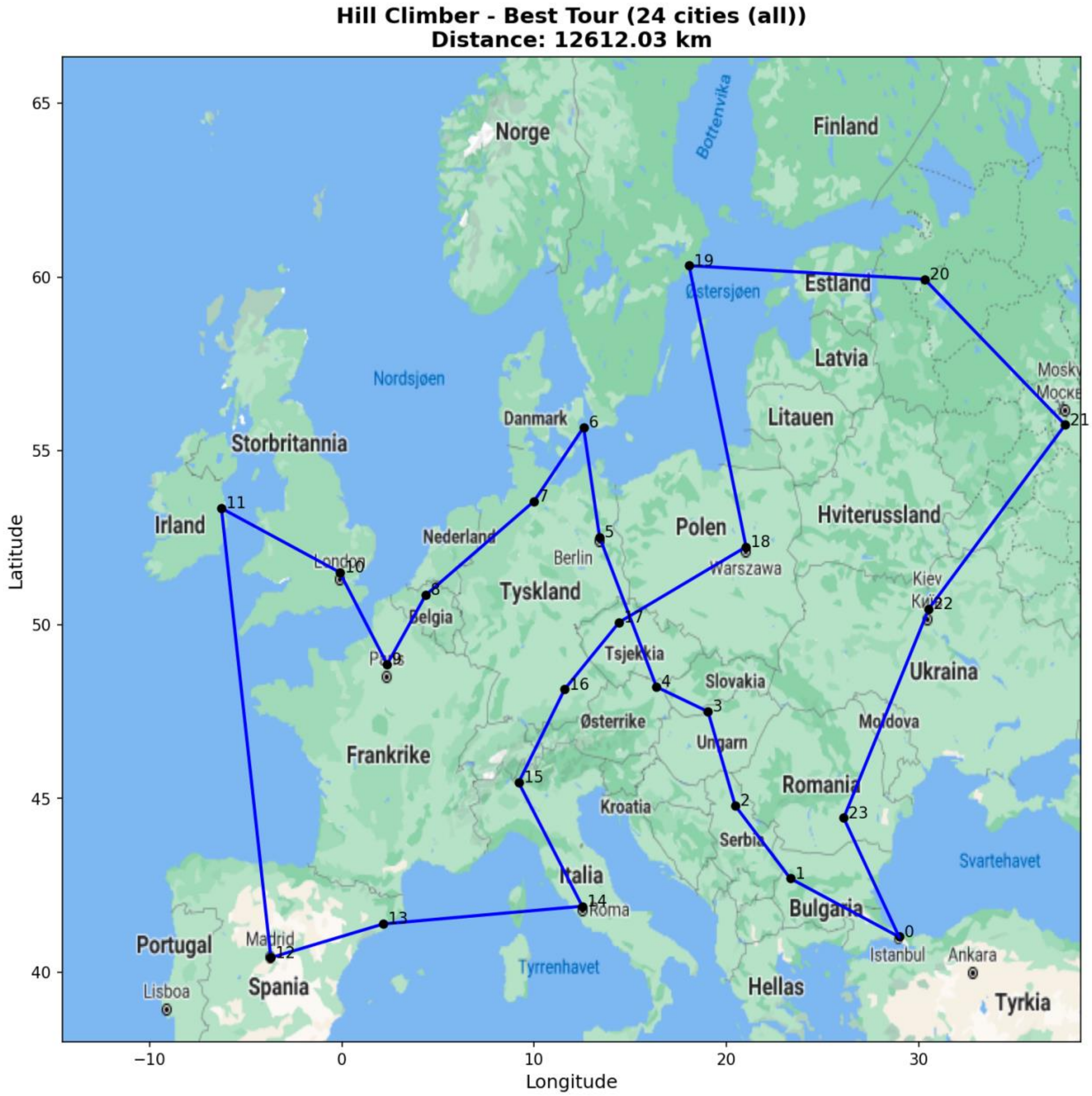
Comparison with Exhaustive Search:

- **Exhaustive (optimal):** 7 486.31 km
- **Hill Climber best:** 7 486.31 km

- **Difference:** -0.0 km

Plot of 10-City Tour:

Plot of 24-City Tour:



Explanation of Code:

`tsp_hill_climbing.py` implements a simple hill climbing algorithm:

- Starts with a random tour
- Generates neighbors using a 2-opt swap (swapping two cities)
- Moves to the best neighbor if it improves the tour
- Stops when no improvement is possible (local optimum)
- Runs 20 trials with different random starting tours (stochastic algorithm)

- Tests on both 10 and 24 cities
- Reports best, worst, mean, and standard deviation across 20 runs
- Compares performance with exhaustive search optimal for 10 cities
- Demonstrates that hill climbing can get stuck in local optima

Performance and Time Complexity:

The hill climber is much faster than exhaustive search:

- For each iteration, it evaluates $n(n-1)/2$ neighbors (all possible swaps)
- Typically converges in a few hundred iterations
- Time scales roughly as $O(n^2)$ per iteration
- Much faster than exhaustive search, but solution quality varies

Genetic Algorithm:

1. **Did GA find the shortest tour (10 cities) compared to exhaustive search?**
    - **Exhaustive optimal (10 cities):** 7 486.31 km
    - **GA best (10 cities):** 7 486.31 km
    - Yes, GA found the optimal tour.
2. **Running time comparison.**
    - **For 10 cities:**
        - **Exhaustive search:** 0.242454 seconds
        - **GA (average – population size: 100):** 0.29 seconds
    - **For 24 cities:**
        - **Exhaustive search:** $5.47e+08$ years
        - **GA (average – population size: 100):** 0.57 seconds
        - GA is astronomically faster; it is practical while exhaustive is impossible
3. **How many tours were inspected GA vs. exhaustive search?**
    - I don't know if I did this correctly, but I somehow did not count how many tours that were inspected by GA for 10 and 24 cities. I did for exhaustive search, and based on how long both algorithms take to run, I would say GA inspected

significantly less tours than exhaustive search, as GA completes search for 24 cities.

Results for Population size 50 (20 runs, 10 cities):

- **Best:** 7 486.31 km
- **Worst:** 7 503.10 km
- **Mean:** 7 487.15 km
- **Standard deviation:** 3.66 km

Results for Population size 100 (20 runs, 10 cities):

- **Best:** 7 486.31 km
- **Worst:** 7 503.10 km
- **Mean:** 7 487.15 km
- **Standard deviation:** 3.66 km

Results for Population size 200 (20 runs, 10 cities):

- **Best:** 7 486.31 km
- **Worst:** 7 486.31 km
- **Mean:** 7 486.31 km
- **Standard deviation:** 0.00 km

Results for Population size 50 (20 runs, 24 cities):

- **Best:** 12 325.93 km
- **Worst:** 13 206.95 km
- **Mean:** 12 724.47 km
- **Standard deviation:** 269.49 km

Results for Population size 100 (20 runs, 24 cities):

- **Best:** 12 287.07 km
- **Worst:** 13 228.29 km
- **Mean:** 12 705.52 km
- **Standard deviation:** 262.40 km

Results for Population size 200 (20 runs, 24 cities):

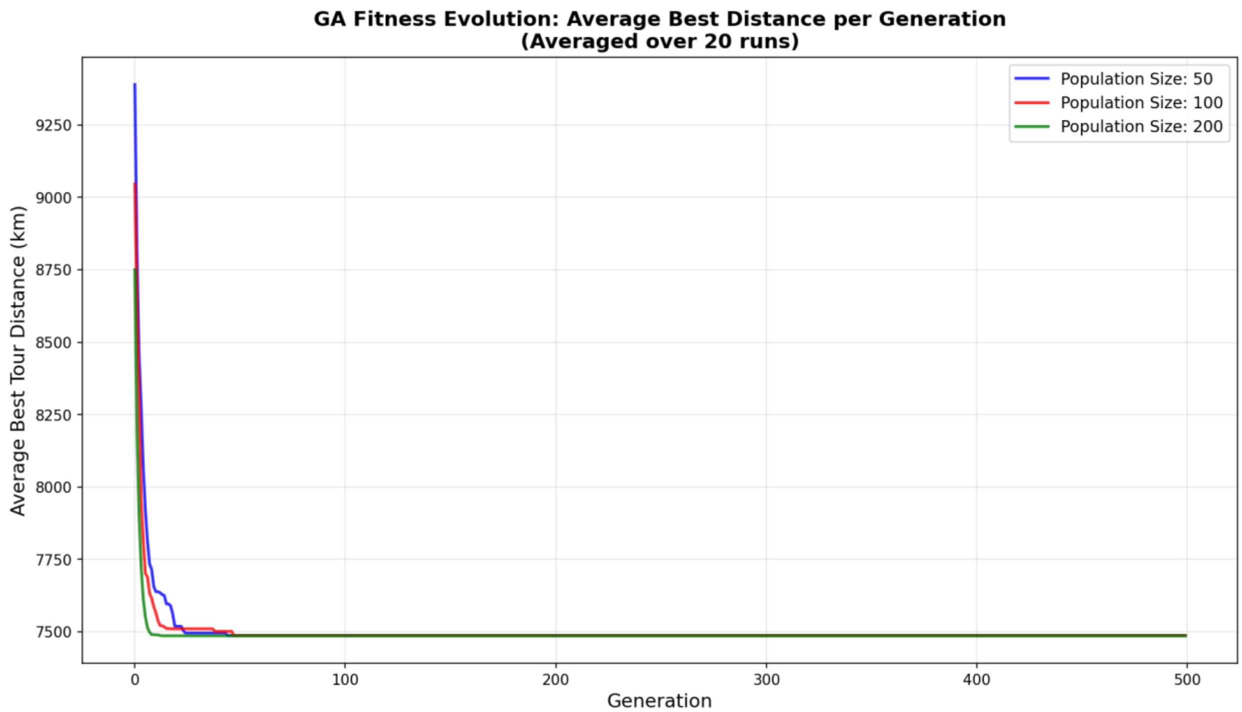
- **Best:** 12 287.07 km
- **Worst:** 13 100.34 km
- **Mean:** 12 533.26 km
- **Standard deviation:** 191.07 km

Plot of 10-City Tour – Population Size 50:

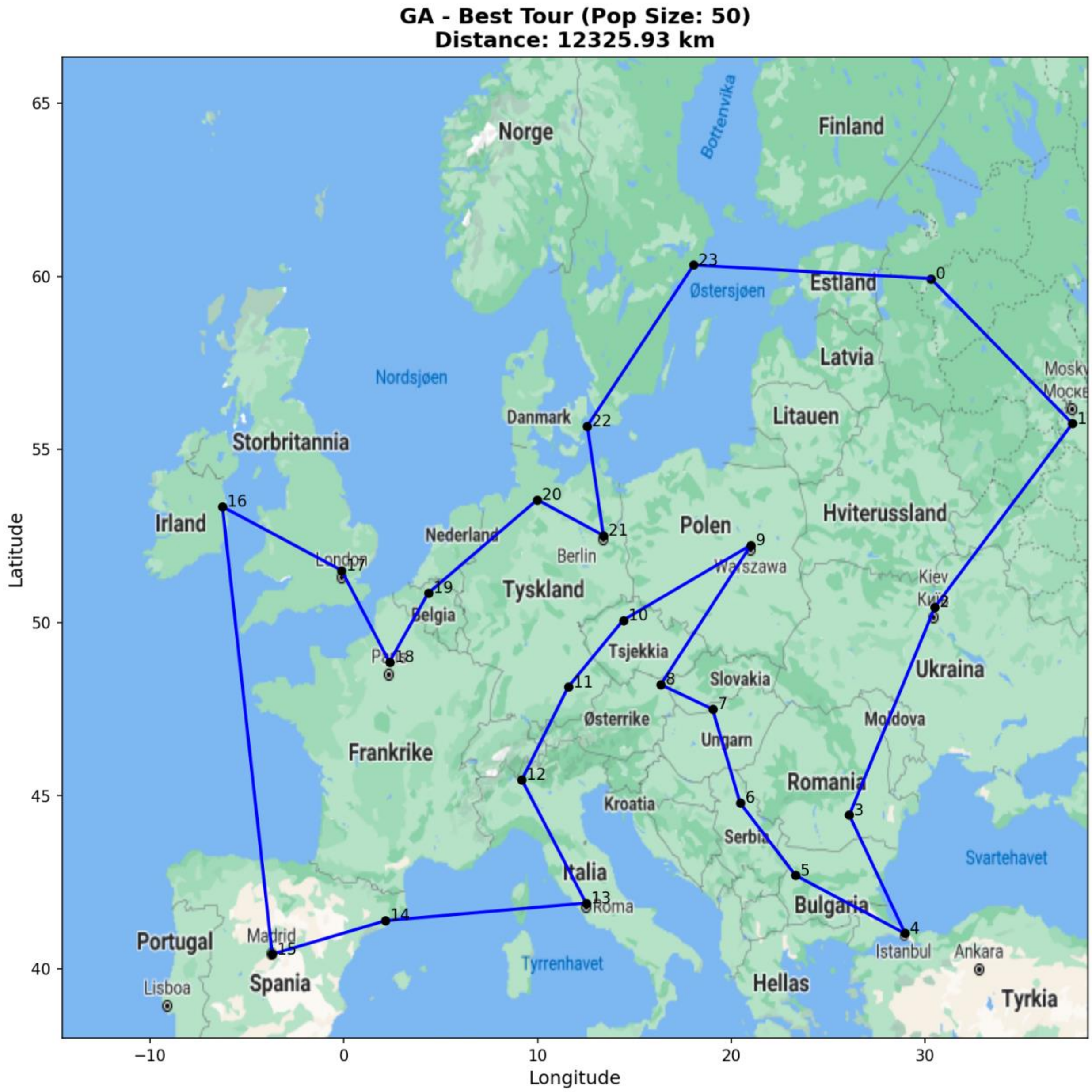
Plot of 10-City Tour – Population Size 100:

Plot of 10-City Tour – Population Size 200:

Fitness Evolution 10 Cities:



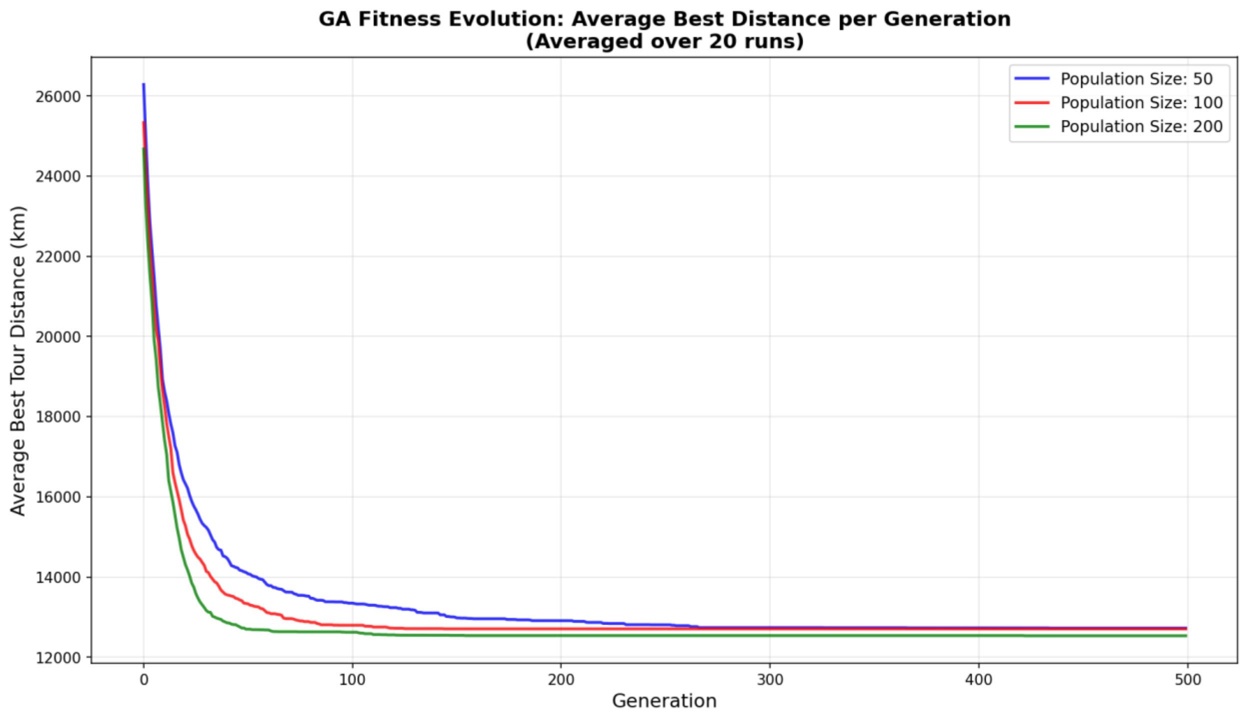
Plot of 24-City Tour – Population Size 50:



Plot of 24-City Tour – Population Size 100:

Plot of 24-City Tour – Population Size 200:

Fitness Evolution 24 Cities:



Explanation of Code:

`tsp_hill_climbing.py` implements a genetic algorithm for TSP:

- Uses Order Crossover (OX) from Eiben & Smith textbook
- preserves relative order of cities, suitable for permutation problems like TSP
- Uses Inversion Mutation - reverses a segment of the tour, effective for untangling crossed paths in TSP
- Uses Tournament Selection for parent selection ($tournament\ size\ =\ 5$)
- Implements elitism (keeps 2 best individuals each generation)
- Tests three population sizes: 50, 100, and 200
- Runs 500 generations for each configuration
- Performs 20 independent runs for each population size
- Reports statistics (best, worst, mean, std dev) for all configurations
- Tracks and plots fitness evolution over generations
- Compares which population size performs best in terms of tour quality and convergence speed

Which Population Size is Best?

- Best population size (by mean distance): 200
  - Mean distance: 12533.26 km
  - Best distance: 12287.07 km

Time Complexity:

- The GA evaluates a fixed number of tours per run:
- $population\_size \times generations \approx 100 \times 500 = 50{,}000$ tours
- This is independent of whether we have 10 or 24 cities
- Time per tour evaluation increases with n, but total tours stay constant
- This makes GA practical even for large problem instances