Homework 2 (Simulated Annealing Algorithm on TSP)

Introduction:

This report details the implementation and performance of the Simulated Annealing algorithm designed to solve the Traveling Salesman Problem (TSP). The algorithm was evaluated on cities of varying sizes, with a focus on optimizing parameters such as starting temperature, final temperature, temperature decrement and iterations per temperature to achieve efficient solutions and assess its scalability.

Methodology:

Algorithm Overview: (Implementation Link)

The SA algorithm was implemented in Python. The algorithm simulated the annealing process to probabilistically explore the solution space, allowing for occasional "uphill" moves to escape local optima. Key components included:

- **Temperature Schedule**: Used exponential decay to adaptively control the annealing process.
- **Neighbor Generation:** Generated by swapping two randomly chosen cities in the current path.
- **Acceptance Criterion:** Transitioned to a new solution based on the Metropolis criterion, considering both the distance change and current temperature.

The following parameters were varied to study their impact:

- Initial temperature: $T_{initial}$
- Final temperature: : T_{final}
- Total iterations: : $iterations_{total}$
- Temperature decay rate: Adaptive exponential schedule.

Parameter Tuning and Experiments

Parameter Setup

The experiments were conducted with the following parameter ranges:

- $T_{initial}$: 100 to 1000
- T_{final}: 1 to 10
- $iterations_{total}$: 1000 to 10,000

Experimental Observations

- Scalability: The algorithm showed a significant increase in computation time as the number of cities increased. The below table shows the running time increment over a difference of approximately 10k cities each (initial temperature – 1000, final temperature – 1, iterations – 100):

Num Cities	Time Taken in seconds
1000	4.223
10000	188.59
20000	608.752

- Temperature Schedule: Faster decay rates resulted in premature convergence, leading to suboptimal solutions, while slower decay rates provided better results at the cost of increased runtime.

Results

Below table shows different experiments that I conducted. I have only included results with considerable time (6000 cities or more) taken during the experiment.

Num Cities	Initial Temperature	Final Temperature	Iterations	Best Distance	Running Time (s)
6000	100	1	1000	133013	71.8295
	100	1	5000	92663	76.5396
	100	1	10000	72779	79.617
	100	5	1000	127222	58.00844407
	100	5	5000	92229	67.79318619
	100	5	10000	75003	80.37513733
	100	10	1000	134910	55.34175992
	100	10	5000	93956	66.06293488
	100	10	10000	75667	77.09729028
	500	10	1000	136025	54.78381371
	500	10	5000	104158	64.8523643
	500	10	10000	89625	75.71150684
10000	100	1	1000	229209	149.7237792
	500	5	5000	195645	160.477546
	1000	10	10000	175349	189.3164992
20000	100	1	1000	477743	595.9421835
	500	5	5000	432241	650.8086491
	1000	10	10000	409762	702.9572515

Observations

Based on the results from the **Simulated Annealing (SA)** experiments and comparisons with the **Genetic Algorithm (GA)** analysis from Homework 1, the following refined observations can be made:

1. Impact of Initial Temperature ($T_{initial}$):

- **Higher Initial Temperatures Lead to Better Results:**
  Higher $T_{initial}$ values allow the algorithm to explore the solution space more thoroughly in the initial stages, avoiding premature convergence. For example:
    - For 10,000 cities, $T_{initial}$ =1000 yielded a best distance of **175,349** with 10,000 iterations, which outperformed lower initial temperatures ($T_{initial}$ =100).
- **Comparison with GA:**
  While GA uses diverse initial populations to ensure exploration, SA relies solely on the initial temperature for global exploration. This makes SA more sensitive to $T_{initial}$, whereas GA can balance exploration and exploitation through crossover and mutation strategies.

2. Impact of Final Temperature (T_{final}):

- **Higher Final Temperatures Help Long Runs:**
  Higher T_{final} values (T_{final}=10) result in smoother transitions and better convergence in longer runs. For example:
    - For 6,000 cities with $T_{initial}$=100, increasing T_{final} from 1 to 10 improved the best distances for 1,000 iterations (from **133,013** to **134,910**) and for 10,000 iterations (from **72,779** to **75,667**).
- **Comparison with GA:**
  In GA, parameters like mutation rate act similarly to T_{final}, ensuring diversity in the final stages. However, GA's adaptability to population evolution provides more robustness against stagnation compared to SA's rigid temperature decay.

3. Impact of Iterations:

- **Longer Runs Yield Better Solutions:**
  Increasing iterations improves solution quality, as the algorithm has more opportunities to explore and refine the solution. For instance:
    - For 6,000 cities with $T_{initial}$=100 and T_{final}=1, increasing iterations from 1,000 to 10,000 reduced the best distance from **133,013** to **72,779**.
- **Comparison with GA:**
  GA scales better with iterations, as it processes multiple solutions (population) per iteration. SA processes a single solution at a time, making it less efficient for large datasets.

4. Scalability with Number of Cities:

- **Exponential Growth in Runtime:**
  As the number of cities increases, both the best distance and runtime grow exponentially. For example:
    - For 20,000 cities with $T_{initial}$=500, T_{final}=5, and 5,000 iterations, the runtime was **650.81 seconds**, compared to **160.48 seconds** for 10,000 cities with the same parameters.
- **Comparison with GA:**
  GA demonstrated a more consistent scaling pattern in Homework 1, thanks to its ability to process multiple solutions simultaneously. SA's single-solution approach makes it slower for larger datasets.

5. Overall Robustness:

- **SA's Sensitivity to Parameters:**
  SA's performance is sensitive to parameter tuning, particularly $T_{initial}$, T_{final}, and iterations. Minor changes in these parameters can drastically affect the results.

- Example: For 10,000 cities, $T_{initial}$=100, T_{final}=1, and 1,000 iterations produced a distance of **229,209**, whereas $T_{initial}$=500, T_{final}=5, and 5,000 iterations improved the distance to **195,645**.

- **Comparison with GA:**
  GA showed less sensitivity to mutation and elitism parameters in Homework 1, providing more robust performance across a wider range of datasets. However, GA's larger memory and computational demands may offset this advantage for smaller datasets.

Conclusion

The Simulated Annealing (SA) algorithm was successfully applied to the Traveling Salesman Problem (TSP), with the following key takeaways:

- SA performs well for bigger datasets (e.g., up to 10,000 cities) when finely tuned.
- Its single-solution approach provides faster convergence for fewer cities, but its reliance on temperature schedules and iteration count makes it less adaptable than GA for larger datasets.
- SA's runtime grows exponentially with the number of cities, making it less suitable for larger datasets. But still it gives much faster results compared to sequential execution of Genetic Algorithms.

Appendix: I have uploaded the plot figures in the same repo I have uploaded the code. It can be found under plot/sim_ann folder. The reason I haven't attached here due to the fact that MS Word was not producing the best resolution for the plots when attempting to attach and re-configure space.