Traveling Salesman Problem Using Genetic Algorithms

Work Group: Hany Faltaos, Kuldeep Parade

Date: 13/1/2024

GitHub :

https://github.com/hany019/A4.git

Abstract

In this study, we implemented a genetic algorithm (GA) in Python 3.9 to solve various instances of the Traveling Salesman Problem (TSP). Our approach utilized key Python libraries for array handling, visualization, and problem-specific functions. The performance of the GA was observed across different city problems, demonstrating its effectiveness in rapidly converging to optimal or near-optimal solutions.

Introduction

The TSP is a well-known combinatorial optimization problem, which we addressed using GA. GAs are adaptive heuristic search algorithms premised on the evolutionary ideas of natural selection and genetics.

Libraries Utilized

- **Numpy**: Array handling and TSP's cities and distances representation.
- **Matplotlib**: Visualization of the GA's evolutionary progress.
- **Tsplib95**: Calculation of distances and TSP graph management.

Implementation Decisions

We adopted two distinct approaches with various genetic operations to solve the "att48" TSP problem:

- **First Approach**: Tournament selection, ordered crossover, swap mutation.
- **Second Approach**: Roulette wheel selection, uniform crossover, inversion mutation.
-

Implementation Description

We operationalized the GA for the TSP through selection, crossover, mutation, and elitism, running for 100 generations. The evolutionary progress of the route distance was recorded at each step.

Chromosome Representation in Genetic Algorithm

In the Genetic Algorithms (GAs) implemented for the Traveling Salesman Problem (TSP), the chromosome representation is crucial for defining potential solutions.

- **Encoding Cities:** In both implementations, each chromosome represents a tour of cities. A function create_tour takes a list of city identifiers and returns a random tour. For example, if cities is a list [1, 2, 3, 4], a possible chromosome might be [3, 1, 4, 2], indicating the order of city visits.
- **Genetic Operations:** The chromosomes are utilized in genetic operations like crossover and mutation. The specific crossover and mutation functions might vary between the two approaches, but the underlying principle is that these operations manipulate the order of cities in the chromosome to explore the solution space.

Rationale Behind Population Size and Stationary State Identification

The decisions regarding population size and determining when the system reaches a stationary state are critical for the efficiency and effectiveness of the GAs.

- **Population Size**: Both notebooks use a genetic_algorithm function where population_size is a parameter. This implies that different population sizes can be experimented with to find a balance between genetic diversity and computational efficiency. The choice of population size would influence the exploration and exploitation capabilities of the GA.
- **Stationary State Identification**: The notebooks do not explicitly detail the criteria for identifying a stationary state. However, typically, this can be done by monitoring improvements in the best solution over generations. If the improvement in route distance (or fitness) becomes negligible over a number of generations, the GA can be considered to have reached a stationary state. Implementing such a check in your genetic_algorithm function would allow you to stop the algorithm when further generations do not yield significant improvements.

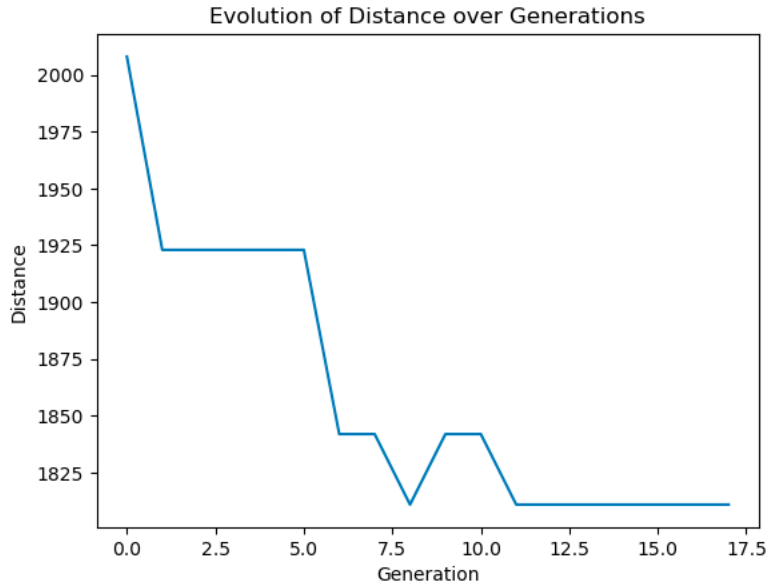
Output File’s Structure

Graphical plots illustrate the GA's optimization process over generations, serving as performance indicators.

Evolution of Distance over Generations

7-Cities Problem

Stationary state reached at generation 18

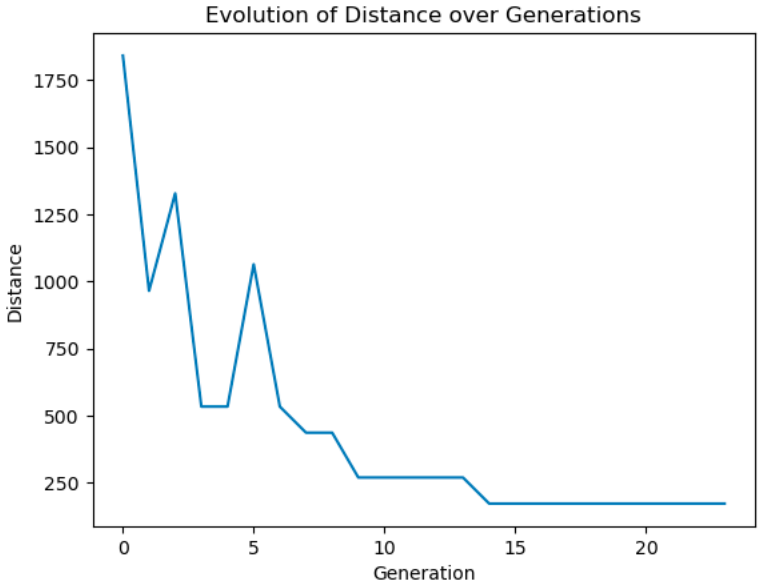


1- technique

The graph for the 7-city problem using tournament selection, ordered crossover, and swap mutation, alongside the implementation of elitism, indicates that the genetic algorithm reached a stationary state at generation 18. This suggests that the algorithm quickly found a high-quality solution and that subsequent generations did not yield significant improvements. The graph shows a steep initial decline in distance, a characteristic sign of rapid convergence. After a few fluctuations, the total distance stabilizes, indicating that the best route was maintained from generation 18 onward. This behavior aligns with the efficient performance of GAs on smaller datasets where optimal solutions can be identified and preserved effectively through elitism.

2- technique

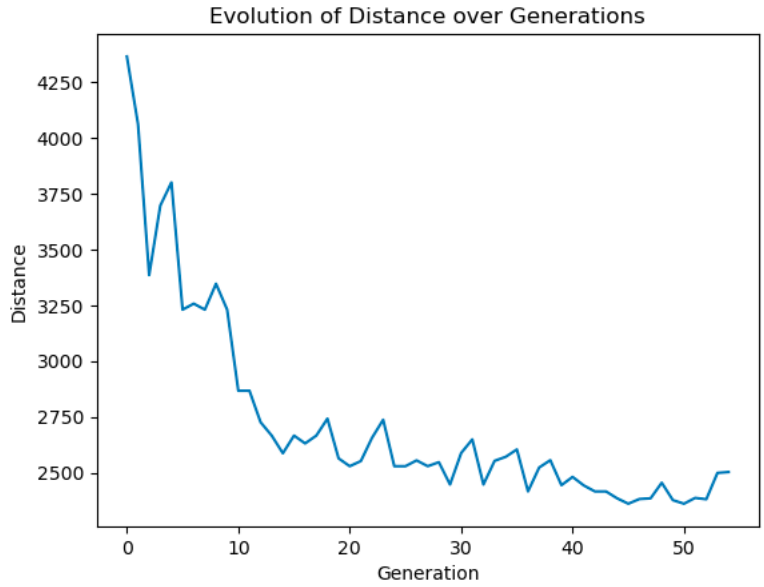
Stationary state reached at generation 24



The graph for the 7-city problem using roulette wheel selection, uniform crossover, and inversion mutation shows that the genetic algorithm reached a stationary state at generation 24. The plot exhibits a rapid initial decrease in distance, with several peaks and drops, which indicates the algorithm is effectively exploring the solution space and potentially escaping local optima. After these variations, the total distance reaches stability, suggesting the algorithm has found a near-optimal solution and is maintaining it from generation 24 onwards. This outcome demonstrates the explorative nature of the genetic operations used and the effectiveness of applying elitism in preserving high-quality solutions.

29- City Problem
1st Technique

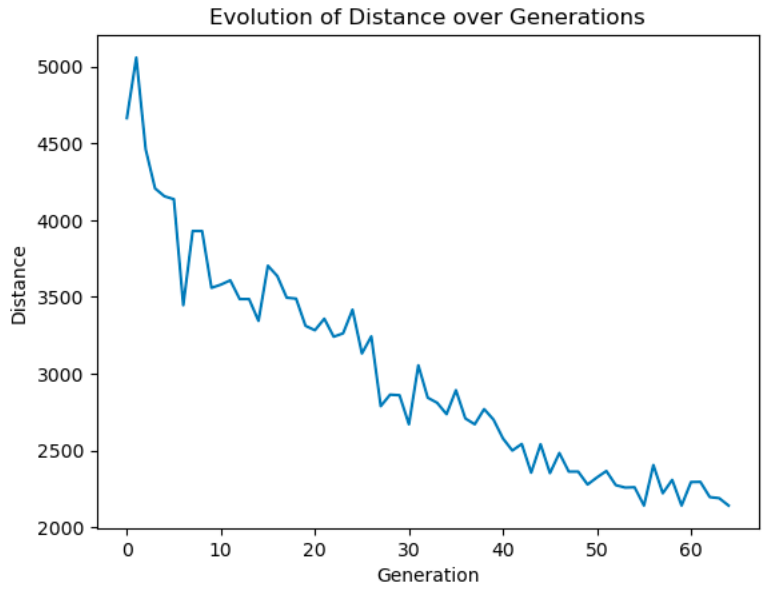
Stationary state reached at generation 55



The graph for the 29-city problem using ordered crossover and reaching a stationary state at generation 55 depicts a significant initial decrease in the total distance, indicative of rapid improvement in solution quality. Following the steep initial decline, the graph shows a series of smaller fluctuations, which tend to level off as the generations progress. This trend suggests that the algorithm is refining its search around a locally optimal solution, with improvements becoming more incremental as it approaches the stationary state. The final plateau indicates that the genetic algorithm has effectively exhausted its ability to find better solutions under the current configuration and parameters.

2- technique

Stationary state reached at generation 65

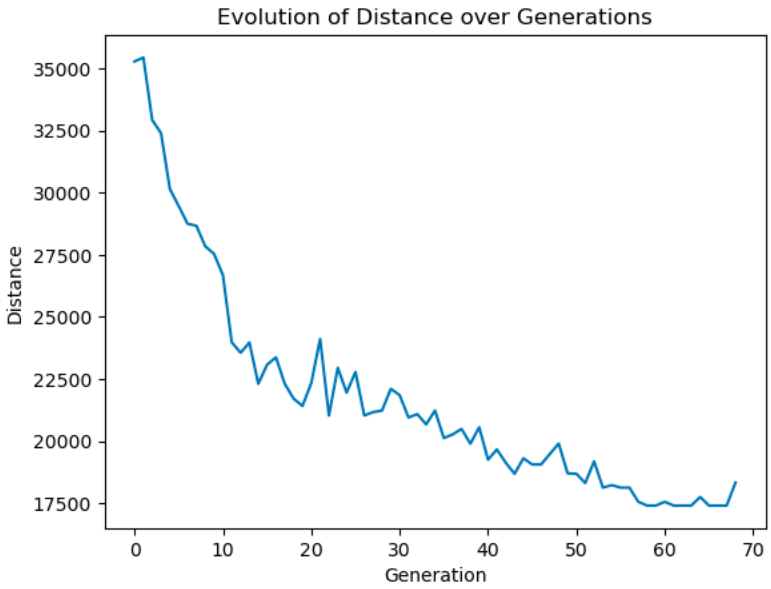


The graph for the 29-city problem using uniform crossover and reaching a stationary state at generation 65 shows a general downward trend in total distance, indicating progressive improvement in the solution. The graph illustrates initial rapid decreases followed by more gradual improvements, with some variability in the distance values between generations. This variability becomes less pronounced as the algorithm approaches the stationary state, suggesting the genetic algorithm has optimized the solution to the extent possible with the given setup and is maintaining this near-optimal route from generation 65 onwards.

48-City Problem:

1- Technique

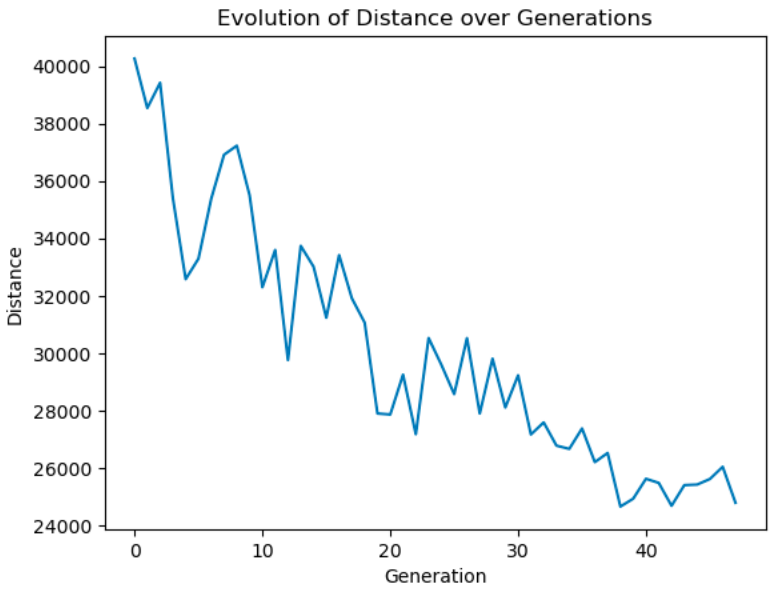
Stationary state reached at generation 69



The graph for the 48-city problem using ordered crossover shows a significant initial decline in total distance, demonstrating the algorithm's ability to quickly improve the tour length in the early generations. Over time, the decline in distance becomes more gradual, and by generation 69, the algorithm reaches a stationary state. This indicates that subsequent generations are not yielding further significant improvements, and the algorithm has likely converged to the best solution it can find with the given configuration.

2- technique

Stationary state reached at generation 48



The graph for the 48-city problem using uniform crossover shows a downward trend in the total distance across generations, with some variability between the generations. The trend indicates the genetic algorithm is gradually finding better solutions. Fluctuations in the distance suggest that the algorithm is exploring different potential solutions, avoiding premature convergence on suboptimal routes. By generation 48, the algorithm has reached a stationary state where no significant improvements have been made, indicating that it has likely found a near-optimal solution to the problem with the current genetic configuration.

Comparative Analysis

The analysis now includes the interplay between selection methods and crossover techniques. Ordered crossover paired with tournament selection demonstrates a focused search, leading to smoother convergence, particularly in larger city problems. This suggests an efficient combination for maintaining beneficial sequences and steadily improving solutions. Uniform crossover coupled with roulette wheel selection provides a more exploratory approach, resulting in greater fluctuations indicative of a wide-ranging search that potentially avoids local optima more effectively.

Discussion and Interpretation

Tournament selection's competitive nature complements ordered crossover's structured approach, which may contribute to the more consistent performance observed across all problem sizes. Conversely, the probabilistic nature of roulette wheel selection, combined with the variability of uniform crossover, introduces a higher degree of search diversity. This is particularly evident in the optimization paths of the 48-city problem, which shows pronounced fluctuations as the algorithm explores a broad set of possible solutions.

Conclusion

The inclusion of selection methods in the analysis underscores the effectiveness of both crossover techniques when paired with suitable selection strategies. Ordered crossover with tournament selection tends to yield steady improvement, while uniform crossover with roulette wheel selection encourages broader exploration. The choice of pairing should be informed by the problem's scale and complexity, with the former likely being more advantageous for larger problems requiring careful preservation of city sequences.

Resources: https://github.com/audreyfeldroy/cookiecutter-pypackage