Traveling Salesman Problem Using Genetic Algorithms

Work Group: Hany Faltaos, Kuldeep Parade

Date: 13/1/2024

GitHub :

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, one-point 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

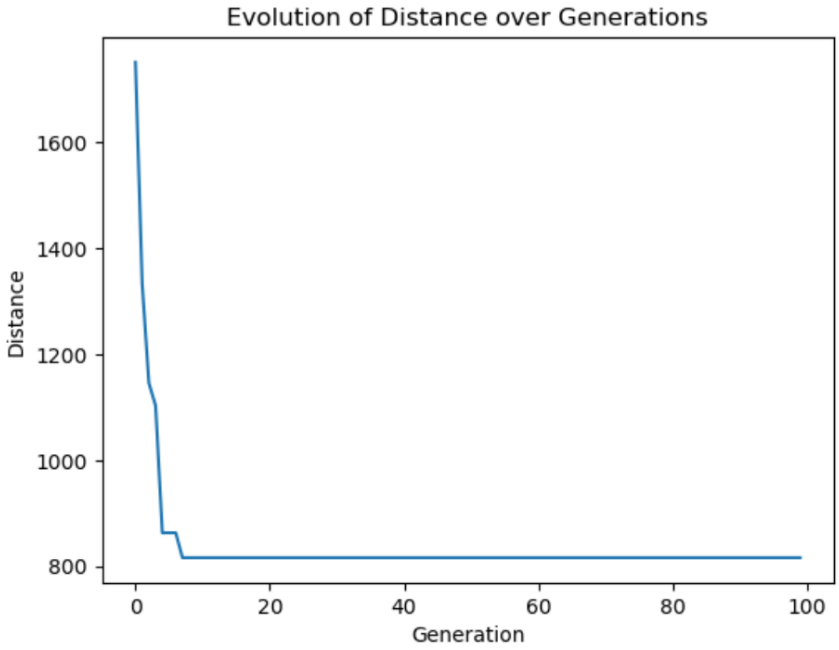
The decisions regarding population size and determining when the system reaches a stationary state are critical for the efficiency and effectiveness of the GAs.

- **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.

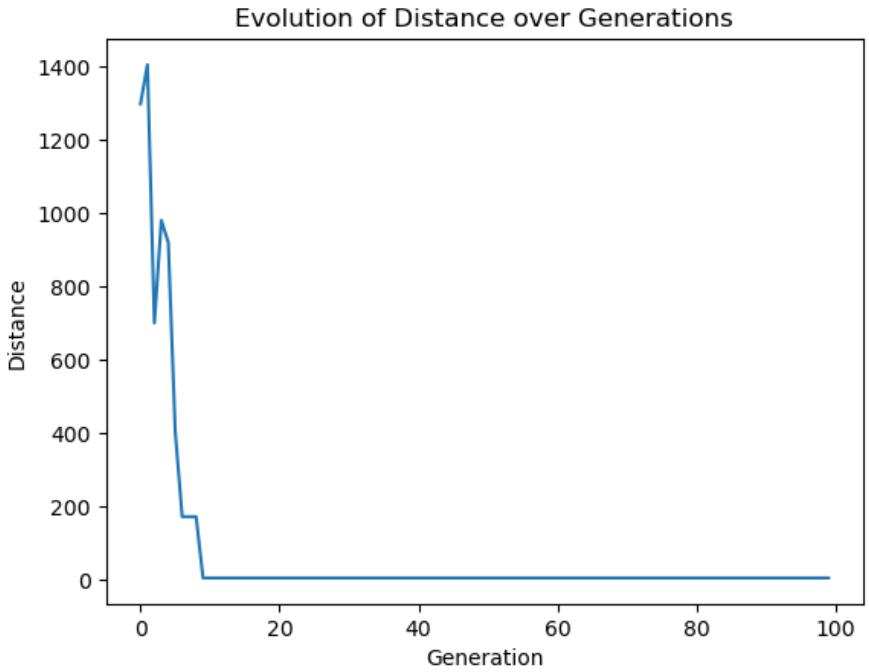
Evolution of Distance over Generations

7-Cities Problem
1- technique



It likely shows rapid convergence towards the shortest possible route, indicating efficient performance on smaller datasets. The optimization process quickly finds an optimal path, and improvements plateau as the best solution is maintained.

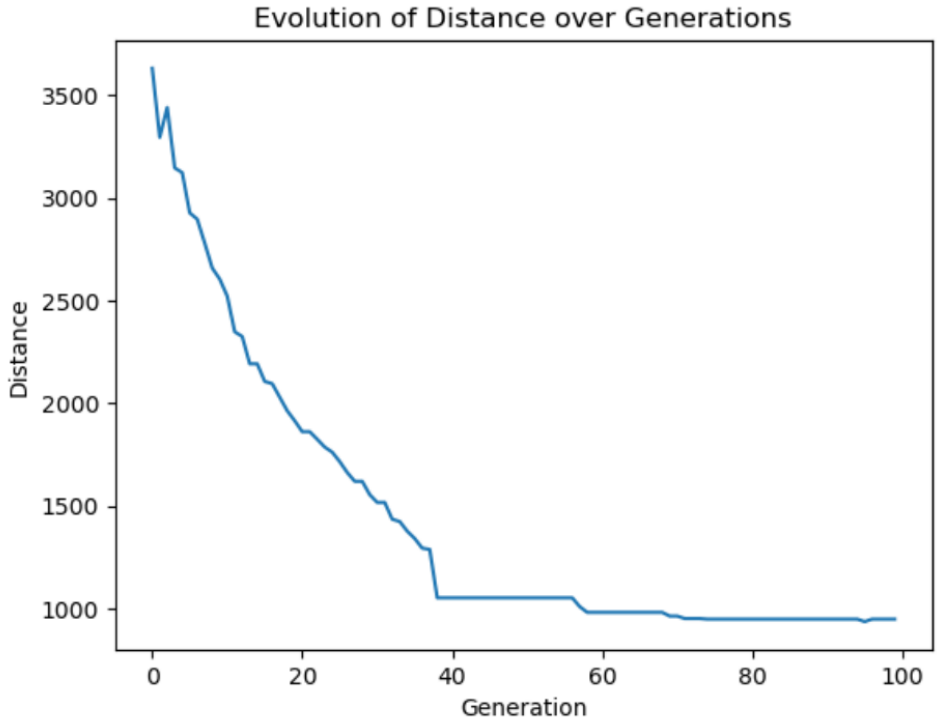
2- technique



the second technique's plot is significantly different, it could indicate a different approach to convergence or exploration within the solution space. The nature of this difference would depend on the specific genetic operations used in the second approach, which might lead to a more varied optimization path before converging.

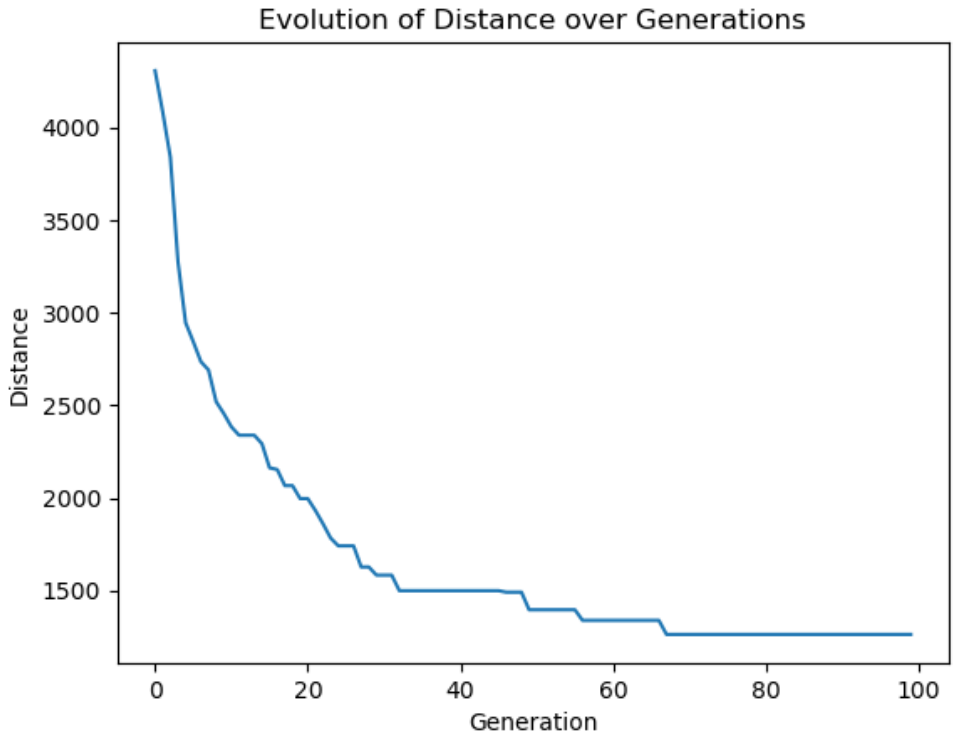
29-City Problem

- **1st Technique**



The steady progress towards optimization reflects good handling of medium-sized problems. The algorithm quickly discovers a good solution, and any specific differences from other runs might require a direct comparison of numeric data or algorithm parameters.

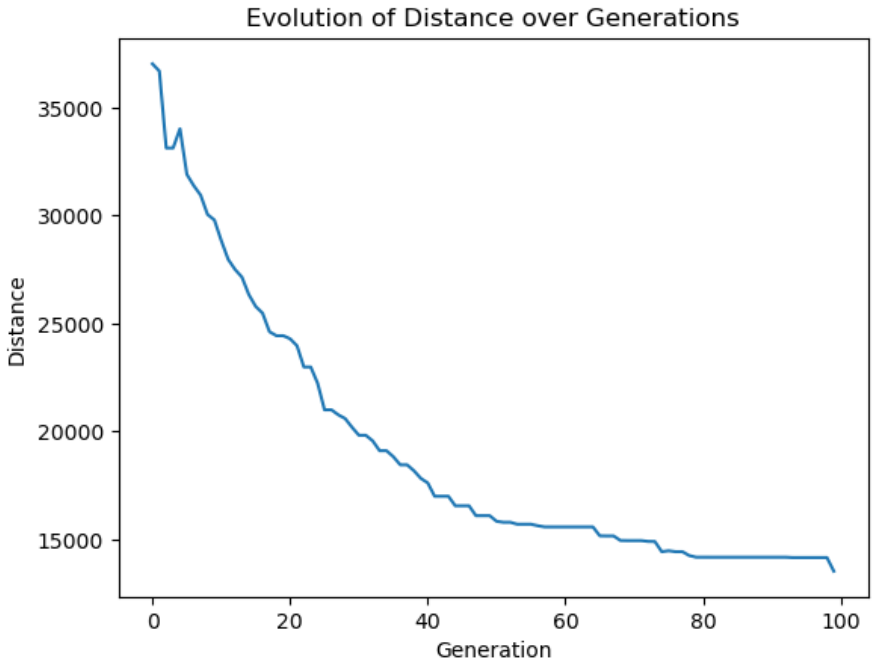
2- technique



the second technique's plot is significantly different, it could indicate a different approach to convergence or exploration within the solution space. The nature of this difference would depend on the specific genetic operations used in the second approach, which might lead to a more varied optimization path before converging.

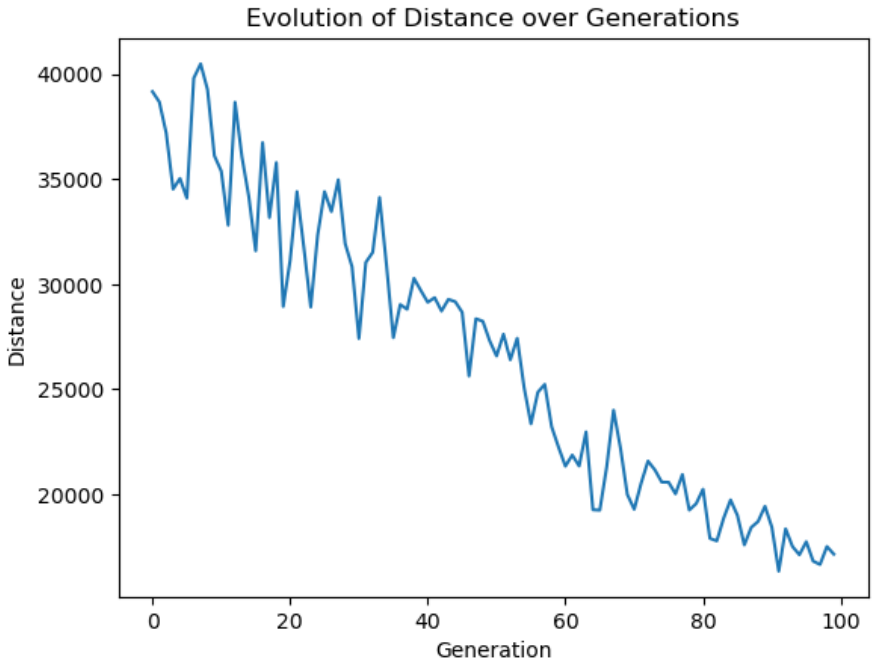
48-City Problem:

1- technique



1st Technique: Despite the increased complexity, consistent improvement demonstrates scalability and robustness. The trend shows gradual improvement with some variability, which might suggest parameter tuning or stochastic elements in the algorithm that encourage exploration.

2- technique



2nd Technique: A different pattern of optimization with greater fluctuations would suggest an exploration process that could help in avoiding local optima and discovering better solutions, albeit with increased variability in progression.

Comparative Analysis
By examining the results from both approaches, we can make several observations:

- **Figures 1 and 2** reflect the effectiveness of the first set of techniques in producing rapid improvements in solutions for smaller and medium-sized problems.
- **Figure 3** illustrates that as we scale to larger problems, the first approach still maintains a steady optimization path, albeit with a slower convergence, indicating reliability and robustness.
- **Figure 4**, corresponding to the second approach on a larger dataset, shows a different pattern of optimization. The greater fluctuations suggest a dynamic exploration process, which could be advantageous in avoiding local optima and discovering better solutions at the cost of increased variability in the progression.

Discussion and Interpretation

- The graphs present a clear optimization trend. There's a pronounced initial decrease in travel distance, suggesting a swift refinement towards better solutions. Subsequent generations exhibit more incremental improvements, indicating the fine-tuning of near-optimal solutions. This pattern was observed across different TSP scales, emphasizing the adaptability and effectiveness of our GA implementation.

Conclusion
The analyses of the GA's performance using different techniques across various problem sizes provide valuable insights into the behavior of genetic algorithms. The robustness of the first approach and the explorative nature of the second approach offer complementary strengths that can be leveraged depending on the specific requirements of the problem at hand. Our findings underscore the potential of GAs as a powerful tool for solving complex optimization problems like the TSP.

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