# 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, 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.

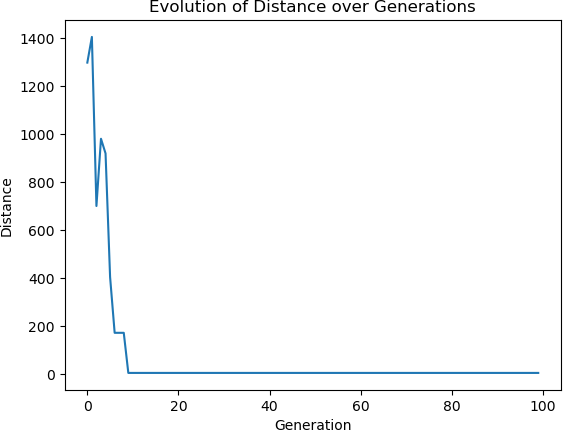
# Evolution of Distance over Generations 7-Cities Problem

# A graph with blue lines Description automatically generated

1. **technique**

7-city problem using ordered crossover ,The updated graph you have provided demonstrates the performance of the genetic algorithm using ordered crossover for the Traveling Salesman Problem. The plot shows the evolution of the total distance of the tour over 100 generations. Initially, there is a sharp decline in distance, indicating rapid convergence towards more optimal solutions. Following this, there are several fluctuations in distance, which suggest the algorithm is exploring the solution space and escaping local optima. Eventually, the distance stabilizes, showing that the algorithm has likely converged to a near-optimal solution.

# technique



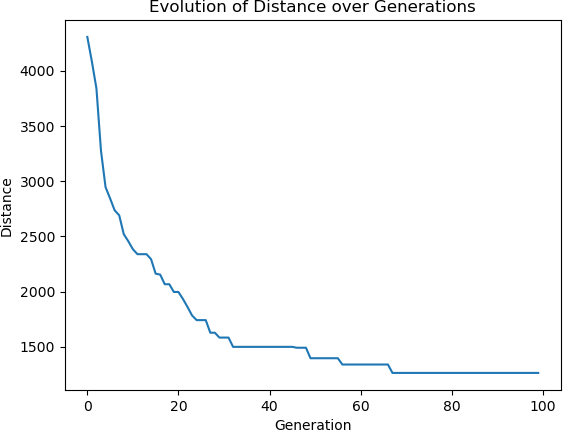
7-city problem using uniform crossover indicates a very swift convergence to an optimal or near-optimal solution within the first few generations. There's an immediate drop in the total distance, which suggests that the genetic algorithm quickly found a short tour. After this drop, the distance remains stable across the generations, implying that the algorithm is maintaining the best solution found without further improvements. This behavior is typical in smaller instances of the TSP, where the search space is limited and optimal solutions can be found rapidly.

### City Problem

# 1st Technique

# A graph with blue lines Description automatically generated

The graph for the 29-city problem using ordered crossover shows a gradual and consistent decrease in total distance across generations. The trend indicates that the genetic algorithm is effectively exploring and exploiting the solution space. There are fluctuations throughout the generations, which suggest the algorithm is avoiding premature convergence and exploring various routes to refine the solution further. The final distance stabilizes toward the later generations, suggesting that the algorithm has likely converged to a near-optimal solution after thoroughly searching the space.

**2- technique**

The graph for the problem using uniform crossover shows a consistent and steep decline in the total distance of the tour over the initial generations. This steep descent indicates that the genetic algorithm was able to quickly find much shorter tours early on. The curve flattens out as the generations progress, which suggests that the algorithm reached a point where further significant improvements are hard to come by. This plateauing effect is typical for genetic algorithms as they approach an optimal or near-optimal solution and the opportunities for further drastic improvements diminish.

# 48-City Problem:

1. **technique**

A graph of a graph

Description automatically generated

The graph for the 48-city problem using ordered crossover shows a strong initial decrease in total distance, indicating that the genetic algorithm quickly finds better solutions at the start. As the generations progress, there are noticeable fluctuations in distance, illustrating the exploration of the solution space and the algorithm's efforts to escape local optima. Towards the later generations, the distance values show less variability, suggesting that the algorithm is converging to a stable solution. This pattern demonstrates the effectiveness of ordered crossover in managing larger and more complex TSP instances.

# A graph of a graph showing the evolution of a number of generations Description automatically generatedtechnique

The graph for the 48-city problem using uniform crossover shows a more erratic decline in total distance, indicative of the algorithm exploring a diverse set of solutions. The trend demonstrates a gradual optimization with several peaks and troughs, suggesting the algorithm is actively avoiding local optima and searching for better paths. This behavior can be beneficial in complex problems like a 48-city TSP, as it allows the algorithm to explore more of the solution space before converging on a near-optimal solution.

**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.

**Top of Form**

**Bottom of Form**

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