Ankara University Department of Computer Engineering 22290021 Ahmet Atakan DEMIR

Solving the Traveling Salesman Problem Using GA and PSO

1. Introduction

Definition of "Traveling Salesman Problem"

The Traveling Salesman Problem (TSP) is a classic problem in combinatorial optimization. It involves a set of cities and the distances between them, and the task is to determine the shortest possible route that visits each city exactly once and returns to the starting city. The problem is widely studied because it has numerous practical applications, such as logistics, route planning, and circuit design. The TSP is a well-known NP-hard problem, meaning that finding an optimal solution in a reasonable amount of time becomes increasingly difficult as the number of cities grows.

Difficulties and Requirements

The primary difficulty in solving the TSP is the exponential growth of the solution space as the number of cities increases. For \(n \) cities, there are \((n-1)! \) possible routes, making the problem computationally infeasible for large instances. This growth makes finding an optimal solution increasingly challenging as the number of cities exceeds 20-30.

To solve the TSP effectively, the following are required:

- **Optimization of Route**: TSP aims to minimize total distance or travel time.
- **Exact vs. Approximate Solutions**: Exact solutions are accurate but costly; approximations are faster but less precise.
- **Scalability**: TSP needs efficient algorithms for large problem sizes.
- **Efficiency**: Efficient algorithms deliver results faster, even for large instances.

2. Algorithms

Genetic Algorithm (GA)

Genetic Algorithm (GA) is a search method inspired by natural selection, used to solve optimization problems. It works by evolving a population of candidate solutions through processes like selection, crossover, and mutation. Over multiple generations, the population improves, aiming for an optimal or near-optimal solution.

GA is effective for complex problems like the Traveling Salesman Problem (TSP), where the solution space is large. It balances exploration of new solutions with refining the best ones, providing high-quality solutions in a relatively short time, though not always guaranteeing the optimal result.

Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a computational method inspired by the social behavior of birds flocking or fish schooling. It optimizes a problem by simulating a swarm of particles, each representing a potential solution. These particles move through the solution space, adjusting their positions based on their own experience and the experience of their neighbors, gradually converging toward the optimal solution.

PSO is particularly effective for optimization problems like the Traveling Salesman Problem (TSP), where the goal is to find the best route among many possibilities. The algorithm balances exploration of the solution space and exploitation of the best-found solutions, making it efficient for complex problems. Though it does not always guarantee an optimal solution, PSO can often find good solutions quickly, especially for large-scale problems.

3. Algorithm Applications

Solution With Genetic Algorithm

I am using a genetic algorithm to solve the Travelling Salesman Problem (TSP). Due to the nature of this problem, I preferred to store the genomes using permutation encoding, as the genomes represent the order in which cities are visited. Since I am using permutation encoding, the mutations involve a swap operation.

I selected the best two genomes from the previous generation using tournament selection and applied elitism to preserve these genomes. Additionally, without specifying a mutation probability, I produced 3 offspring for each genome. These offspring were generated through mutation. However, due to the circular nature of permutation representation and the TSP problem, genomes can emerge that represent the same result but with different city orders. This is a problem, but it has not currently hindered me in solving the problem, so I am ignoring it.

Below is the pseudocode that illustrates how my algorithm works.

```
Start:
  Define cities and the distance matrix.
  Set population size and number of iterations.

  Generate 8 random city orderings for  genomes.

Main
  For each iteration: (100)
    Evaluate each individual (total distance).
    Select parents (tournament selection).
    Generate new individuals (swap mutation).
    Track best and worst fitness values.

After the final iteration:
  Record the results.

Display Results:
  Print the best path and total distance.
  Plot fitness values.

End.
```

Solution With Particle Swarm Optimization:

This time, I am using the Particle Swarm Optimization (PSO) algorithm to solve the Traveling Salesman Problem. As in GA, each particle here represents a permutation, and the velocity values represent swap operations on the permutation. In my algorithm, during each iteration, each particle generates two swap operations based on its personal best and global best values and applies them to its velocity.

However, multiple swap operations can sometimes cause the particle to be misdirected, or two swap operations may neutralize each other, causing the particle's movement to stop. I have tried several methods to solve this issue, but I believe that this situation could actually be useful for discovering new solutions.

After a certain number of particles and iterations, my algorithm presents the global best value as the solution. This solution represents the best-found permutation.

Below is the pseudocode that illustrates how my algorithm works.

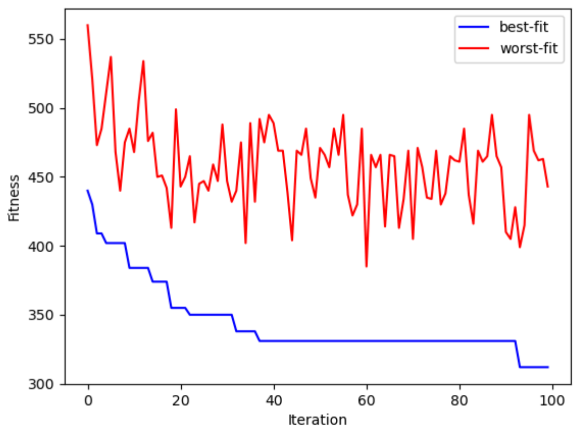
```
Start:
  Define cities and the distance matrix.
  Set number of particles and iterations.

Main:
  Initialize 8 particles (random city orderings)
  and velocities (random swap operations).
  Evaluate initial fitness of particles.

  Set global best and personal bests

  For each iteration: (100)
    For each particle:
      Apply velocity to the particles.
      Evaluate fitness. (total distance)
      Update pBest and gBest
    Update velocities based on pBest and gBest.

Display Results:
  Print the best path and total distance.
  Plot fitness values.

End.
```

4. Results and Evaluation

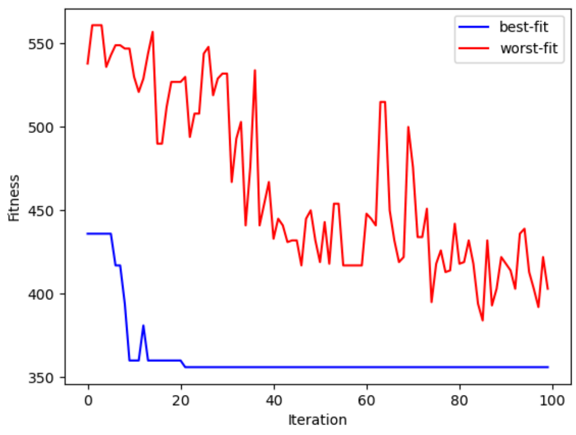
Performance Comparison

Before comparing the performance of the algorithms, let's examine the best fitness values that change throughout the processes of both algorithms.

Genetic Algorithm Fitness Graph:



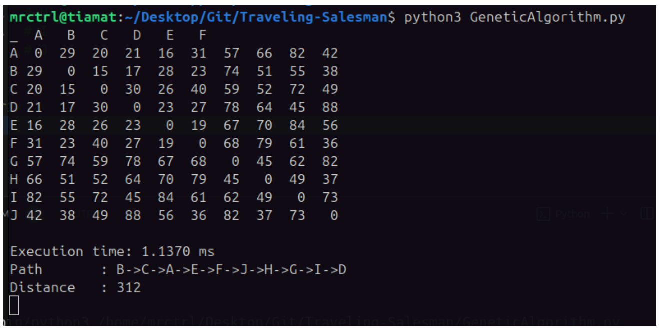
PSO Algorithm Fitness Graph:



The shape of these graphs is largely dependent on the random values that the algorithms start with. However, it is still possible to make a general assessment of the algorithms from these graphs. As seen, the PSO algorithm reaches the optimal result much faster than the GA. The main reason for this is that, unlike GA, which attempts to reach the best solution through random guesses, PSO uses velocity values to perform its mutations in a controlled and beneficial manner. On the other hand, the GA exhibits more random behavior. This allows the GA to easily overcome local minimum points in the fitness function and reach a better fitness value than the PSO.

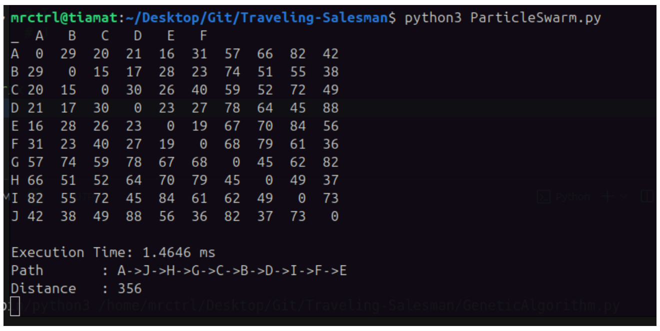
It also brings all individuals of the PSO closer to the best state by following the worsfit values. From here, we can see that the PSO focuses its search process around the best particle.

Genetic Algorithm Performance: 1.1370ms



```
mrctrl@tiamat:~/Desktop/Git/Traveling-Salesman$ python3 GeneticAlgorithm.py
_  A   B   C   D   E   F
A  0  29  20  21  16  31  57  66  82  42
B 29   0  15  17  28  23  74  51  55  38
C 20  15   0  30  26  40  59  52  72  49
D 21  17  30   0  23  27  78  64  45  88
E 16  28  26  23   0  19  67  70  84  56
F 31  23  40  27  19   0  68  79  61  36
G 57  74  59  78  67  68   0  45  62  82
H 66  51  52  64  70  79  45   0  49  37
I 82  55  72  45  84  61  62  49   0  73
J 42  38  49  88  56  36  82  37  73   0

Execution time: 1.1370 ms
Path        : B->C->A->E->F->J->H->G->I->D
Distance    : 312
```

PSO Algorithm Performance: 1.4646ms



```
mrctrl@tiamat:~/Desktop/Git/Traveling-Salesman$ python3 ParticleSwarm.py
_  A   B   C   D   E   F
A  0  29  20  21  16  31  57  66  82  42
B 29   0  15  17  28  23  74  51  55  38
C 20  15   0  30  26  40  59  52  72  49
D 21  17  30   0  23  27  78  64  45  88
E 16  28  26  23   0  19  67  70  84  56
F 31  23  40  27  19   0  68  79  61  36
G 57  74  59  78  67  68   0  45  62  82
H 66  51  52  64  70  79  45   0  49  37
I 82  55  72  45  84  61  62  49   0  73
J 42  38  49  88  56  36  82  37  73   0

Execution Time: 1.4646 ms
Path        : A->J->H->G->C->B->D->I->F->E
Distance    : 356
```

There may be several reasons for the performance (execution time) difference between the algorithms. In this scenario where both algorithms work with the same number of iterations, it is observed that PSO performs better. The main reason for this is that the number of iterations is a parameter that works in favor of GA and the cost of PSO's velocity calculations is high. If we had stopped the algorithms with a system that follows the monotonicity in the fitness value instead of stopping the process using the number of iterations, PSO would have been much more successful than GA. We can easily reach this information from the graphs we have given before.

Also, the programmer's coding performance is an important factor affecting the overall program performance and should not be ignored.

Comparison Result

Performance Difference Between PSO and GA

Particle Swarm Optimization (PSO) generally produces faster and more efficient results than Genetic Algorithm (GA), offering quick solutions with simple parameters and low computational cost. However, PSO can struggle with large search spaces and may get stuck in local optima if not properly designed.

Advantages and Disadvantages of Genetic Algorithm (GA)

GA is effective in exploring wide solution spaces by mimicking biological processes, promoting diversity in the population. However, it comes with higher computational costs and complex parameter tuning, which can hinder efficiency, especially for large-scale problems.

Advantages and Disadvantages of Particle Swarm Optimization (PSO)

PSO provides fast solutions with simple parameter adjustments and low computational cost. Its main disadvantage is the potential for getting stuck in local minima, especially if problem encoding or parameters are not chosen effectively. PSO also has limited exploration capabilities in complex or large search spaces.

Which Algorithm to Choose Depending on the Problem

The choice between PSO and GA depends on the problem's needs. PSO is ideal for fast convergence and efficiency, while GA is better suited for complex and large search spaces, despite its higher computational cost.