Genetic Algorithm for the Traveling Salesman Problem

Devarshi Goswami - 730054389

November 15, 2023

Abstract

The Travelling salesman problem is renowned in the domain of combinatorial optimization. There exists no particular efficient method to solve this problem to give the best result. Problems like these are called NP-complete and many algorithms are used to solve the traveling salesman problem. A genetic algorithm is a heuristic method that is ideally used to improve the solution space for any problem and it can be applied to a varied range of problems. Some algorithms can provide an optimal solution to such a problem but heuristic methods like Genetic Algorithms provide the nearest optimal solution within a fairly reasonable time. Optimization is the process of making something better which improves the solution or finds the better solution amongst all available solutions in the solution space. In this project, we have worked on a plethora of parameters we can feed to a genetic algorithm to provide the most optimal result.

Introduction

The idea of the problem is that there exist N vertexes and the distance between those vertexes. the traveler has to traverse through all cities exactly once and return to the same city from where he/she started with the minimal cost of traversing. The main difficulty of this problem is the humongous number of possible tours:

$(n-1)!/2$ for n cities.

Our goal is to find the minimum Hamilton cycle that starts from any vertex and visits all vertexes only once before returning to its original location.

Genetic algorithms are probabilistic search algorithms used to solve complex problems like the traveling salesman. It is used for searching the solution space in the field of problem optimization and is based on the core idea of Darwin's "survival of the fittest". Genetic Algorithms find the best solution with a large number of iterations and can be easily parallelized. There exists an analogy in the genetic algorithm that if you have a chimpanzee randomly typing on a typewriter for an infinite amount of time, eventually, it will produce a complete work of Shakespeare or any other specific piece of text. analogy represents the search process in a solution space. The chimpanzee's random typing corresponds to the generation of random solutions to a problem. Over time, through a process of selection and recombination (analogous to natural selection and genetic crossover in genetic algorithms), the solutions that are closer to the desired solution are more likely to be preserved and passed on to the next generation.

Input Pre-processing

Before starting to work on the solution I would like to give some context on the data we are using for testing our genetic algorithm. we have been provided with maps for two specific sets of cities, one in Brazil (brazil.xml) and the other in Burma (burma.xml). These maps include details about the cities and the associated travel costs between them, all provided in an XML format. The xml.etree.ElementTree module implements a simple and efficient API for parsing and creating XML data and we have used that to parse the input XML files to create a n x n matrix where n is the size of the vertexes. Each element in the matrix represents the distance between two locations. For Brazil.xml we have 58 vertexes starting from 0 to 57 and for Burma, we have 14 vertexes starting from 0 to 13. This would be our input for the fitness function.

Experiment

A genetic algorithm finds all possible solutions to the problem and applies various operators like fitness evaluation, selection, crossover, and mutation. All possible solutions to the problem are called the population and each individual solution is called a chromosome. Each value in a chromosome is referred to as a gene. Hence with the context build, let us examine the methodology used in this project by understanding the steps in the genetic algorithm process.

I have chosen an object-oriented methodology to code my genetic algorithm where each instance of a class is a genetic algorithm with specific parameters:

$$[3, 51, 26, 9, \ldots, 5, 57, 40]$$

Initial Population Generation

The genetic algorithm initializes by creating a population. The initial population is generated randomly by our algorithm. Our $generate_permutation$ function does that when a class is instantiated. the $_{init}$ method mentioned above calls that function where we are taking a 1D vector of inputs using $input_string = [iforiinrange(distance_matrix.shape[0])]$ this gives us an array of numbers having values 0 to 57. We are then using the random library in Python to shuffle the digits in the input array and add a condition where if the sequence not has already been created, we are appending it to the initial population

article listings

Listing 1: GeneticAlgorithm Class

```
class GeneticAlgorithm:
    def __init__(self,
        input_vector, n,
        tournament_size,
        distance_matrix,
        mutation_point,
        crossover_type='fix'):
         self.mutation_point =
             mutation_point
         self.input_vector =
             input_vector
         self.n = n
         self.tournament_size =
             tournament_size
         self.distance_matrix =
             distance_matrix
         self.locations =
             len(input_vector)
         self.crossover_type =
             crossover_type
         self.candidates =\
         self.generate_permutations()
```

Fitness Evaluation

This step assigns every individual chromosome produced in the last stage. In our case, we are using the fitness value of 1/l where l is the total path length for an individual solution. This means solutions that have the least traversal cost will be rewarded.

$$\text{cost} = \sum_{i=1}^{n} D[C[i]][C[i+1]] + D[C[n]][C[1]]$$

Parent Selection

In this phase, we select the fittest chromosomes to further process the algorithm. We are supposed to observe the fitness of each individual and select the individual with the best fitness for the next phase.

Tournament Selection is a widely used and robust selection mechanism which is what we have implemented in our experiment. The selection pressure of tournament selection is directly proportional to the tournament size where the larger the contenders in the tournament, the higher the selection pressure. Selection pressure means the degree to which the better individuals are preferred. The higher the selection pressure, the more the better individuals are favored.

For our use case, we use the random. a sample method of the Python random library that Generates n unique samples (multiple items) from a sequence without repetition.

Crossover

The operator defines child production via a combination of the genetic information of two parent solutions to create one or more offspring solutions. In our use case we have used two types of crossover operations:

Fixed Point Crossover

A single crossover point is selected and the genetic material beyond that point is exchanged between two parent solutions.

Ordered Crossover

This involves selecting a subset of genes from one parent and filling in the remaining positions with genes from another parent, which maintains the order of occurrence in the parents. In our implementation, we Randomly select two crossover points (indices) after which we create an empty offspring with the same length as the parents copying a segment from the first parent to the offspring and then creating lists of elements remaining in parent2 (excluding the copied segment) then Iterating through the offspring and fill in the remaining elements from parent2 and parent1

Mutation

A mutation operator introduces random changes to the individual solutions of the population. It is used to explore new states and helps us avoid a local optima maintain genetic diversity within the population and avoid premature convergence.

Replacement

Finally, we are replacing the chromosomes in the current population generated with the offspring. replacement refers to the process of selecting individuals from the current population to be replaced by new individuals (offspring) generated through genetic operators such as crossover and mutation. Although there are multiple ways to achieve this like Rank based, St In our experiments, we have achieved this by the implementation of a generational replacement where the current population is replaced by a new set of offspring in each generation. This is being done by getting the candidates that have the highest cost of traversal using an idxmax operator, and then replacing it with the offspring.

Figures

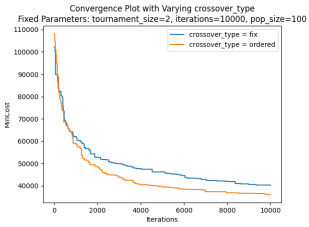

Figure 1: Convergence Plot for fixed Crossover Type

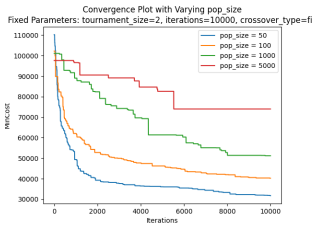

Figure 2: Convergence Plot for fixed population size

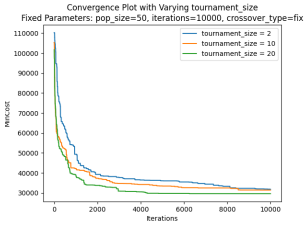

Figure 3: Convergence Plot for fixed tournament size

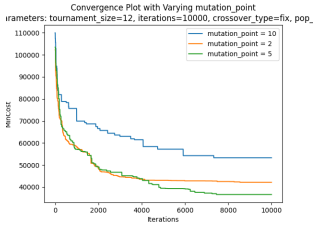

Figure 4: Convergence Plot for fixed mutation point

Conclusions

Some of the conclusions from the above charts are: An ordered crossover converges to better Minima in the case of Brazil A rapid decrease in cost early in the optimization process indicates exploration, while slower changes later on suggest exploitation. As the population size increases, the marginal improvement in solution quality often diminishes. Beyond a certain point, increasing the population size might not lead to a proportional increase in the algorithm's ability to find better solutions.

Questions

Question 1

Which combination of parameters produces the best analysis?

Brazil: The Minimum cost we arrived upon is **28218** which we arrived upon with the following parameters: tournament size of 20, population size of 50, single-point mutation, and ordered crossover. **Burma**: The Minimum cost we arrived upon is **3323** which we arrived upon with the following parameters: tournament size of 20, population size of 5000, single-point mutation, and fixed crossover.

Question 2

What do you think is the reason for your findings in Question 1?

A large tournament size enables a diversified set of chromosomes to compete. that promotes exploration in the search landscape. This allows the algorithm to discover a broad range of solutions. In contrast, a smaller population size encourages exploitation, which allows more focus on the best individuals found till now. Striking an apt balance between exploration and exploitation can lead to optimal solutions.

Ordered Crossover: The selection of ordered crossover can lead to a high impact because this

method will reserve the relative order of cities in the parent chromosomes that will maintain the good characteristics of both parents in the generated offspring.

Single Point Mutation: The use of single-point mutation introduces small changes to the populace, which helps explore the neighboring solutions. But, in some cases, we have observed that in some cases doing this has led to prematurely arriving at a sub-optimal solution.

Question 3

How do each of the parameter settings influence the performance of the algorithm?

The performance of any Genetic Algorithm is dependent on the parameters that have been fed and each parameter plays a critical role in the output of the algorithm.

Population Size: Increasing the population size ideally improves global exploration but may slow down convergence due to the fact that more computations need to be performed. Smaller populations may converge faster but can also lead to arriving at sub-optimal solutions

Tournament Size: A large tournament size can help in promoting diversity which may allow us to escape local optima but again, would require more computational resources.

Iterations: We have observed that if we increase the number of iterations to more than 10000, with the same set of parameters we are getting better solutions. This is because more iterations can allow us longer explorations but convergence should be monitored because it might just so happen that we have already arrived on the global optima but computation is still going on.

Crossover Type: this would determine how the genetic material is exchanged between the parents and offspring. Having a fixed convergence would enable us to arrive at faster convergence while ordered will preserve structural information hence enhancing exploration.

Mutation Point:

Overall Considerations: We have observed that there are always trade-offs between exploration and exploitation. Parameters that affect exploration include larger populations and larger tournament sizes enabling the ability of the algorithm to discover diverse solutions while the parameters that influence exploitation include smaller populations and fixed crossover may accelerate convergence toward a local optima.

Question 4

Can you think of a local heuristic function to add? A valuable local heuristic function that can be added to enhance the performance of our genetic algorithm is the 2-Opt heuristic. It works by iteratively removing two edges from the current tour and then reconnecting the paths to possibly form a better, shorter tour. If the tour distance becomes lower then we accept the solution, we keep repeating this process.

Another local heuristic function that is used in synch with the Travelling Salesman problem is the Lin-Kernighan algorithm. It is known for finding high-quality solutions by applying a series of transformations iteratively to identify pairs of potential exchanges to calculate paths that lead to an increase in fitness.

Question 5

Can you think of any variation for this algorithm to improve your results? Explain your answer.

One variation that we can add to the algorithm is to use dynamic crossover strategies based on the characteristics of the population. We can dynamically switch between ordered and cycle crossover methods. This choice of crossover strategy can depend on the diversity of the population or the historical performance of every strategy being used.

Question 6

boldDo you think of any other nature-inspired algorithms that might have provided better results? Explain your answer.

Optimization of Ant Colonies (ACO):

Ants' methods of foraging serve as an inspiration for ACO. It portrays the issue as a graph with potential paths to resolution for each edge. Ants leave pheromones behind on the paths they take, and the quantity of pheromones left behind affects the likelihood of selecting a particular path. By utilizing artificial ants' collective intelligence to find high-quality pathways quickly, ACO has successfully solved TSP. SA, or Simulated Annealing, is:

The metallurgical annealing process served as the model for Simulated Annealing. Because it is a probabilistic optimization algorithm, it can escape local optima by accepting poorer solutions with a given probability. SA gradually lowers its high exploration rate (high temperature) as it approaches extraction.

References

[1] Anitha Rao, Sandeep Kumar Hegde, *Literature Survey On Travelling Salesman Problem Using Genetic Algorithms*, International Journal of Advanced Research in Eduation Technology (IJARET), Vol. 2, Issue 1 (Jan. - Mar. 2015).

[2] P. LARRANAGA, C.M.H. KUIJPERS, R.H. MURGA, I. INZA and ˜ S. DIZDAREVIC, *Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators*, Artificial Intelligence Review 13: 129–170, 1999.

[3] Heinrich Braun, *On Solving Travelling Salesman Problems by Genetic Algorithms*, Journal Name, 2020.

[4] Brad L. Miller,David E . Goldberg *Genetic Algorithms, Tournament Selection, and the Effects of Noise*, Complex Systems 9 (1995)

[5] A.J. Umbarkar1 and P.D. Sheth2, *crossover operators in genetic algorithms: a review*, ictact journal on soft computing, october 2015, volume: 06, issue: 01.

[6] David S. Johnson, *Local Optimization and the Traveling Salesman Problem* , AT&T Bell Laboratories Room 2D- 150 Murray Hill, NJ 07974, USA

[7] Goldberg, D.E., Holland, J.H, *Genetic Algorithms and Machine Learning*, Machine Learning 3: 95-99, 1988.