Tackling the Traveling Salesman Problem with Genetic Algorithms

Mauro Vázquez Chas
Dániel Mácsai

2024. 11. 10

Abstract

Abstract goes here.

Contents

1 Introduction

1.1 Problem setup

1.2 Background

Provide background information on the topic, including relevant literature and the motivation behind the project.

1.3 Objectives and scope

Outline the main objectives and goals of the project. Define the scope and limitations of the study.

2 Methodology

2.1 An overview on Genetic Algorithms

Explain the basics of genetic algorithms, including selection, crossover, and mutation.

2.2 Crossover techniques

Describe the POS operator, its implementation, and its role in the genetic algorithm.

2.3 Mutation Operators

Detail the mutation operators used, such as Exchange, Insertion, and Inversion Mutation (IVM).

3 Implementation

3.1 Environment Setup

Describe the tools, libraries, and environment used for the implementation.

3.2 Code Structure

Provide an overview of the code structure, including key modules and their functionalities.

3.3 Key Algorithms

Include code snippets and explanations of the core algorithms implemented.

3.3.1 Position-Based Crossover

```python
import random

class Crossover:
    def __init__(self, parent1, parent2, number_of_pos):
        self.parent1 = parent1
        self.parent2 = parent2
        self.number_of_pos = number_of_pos

    def POS(self) -> tuple[list[int], list[int]]:
        # Select random positions for the subset
        positions = random.sample(range(len(self.parent1)), self.number_of_pos)

        # Initialize offspring with None
        offspring1 = [None] * len(self.parent1)
        offspring2 = [None] * len(self.parent2)

        # Copy the selected positions from parents
        for pos in positions:
            offspring1[pos] = self.parent2[pos]
            offspring2[pos] = self.parent1[pos]

        # Function to fill the remaining positions
        def fill_offspring(offspring, parent):
            current_index = 0
            for city in parent:
                if city not in offspring:
                    while current_index < len(offspring) and offspring[current_index] is not None:
                        current_index += 1
                    if current_index < len(offspring):
                        offspring[current_index] = city
            return offspring

        # Fill the remaining positions for both offspring
        offspring1 = fill_offspring(offspring1, self.parent1)
        offspring2 = fill_offspring(offspring2, self.parent2)

        return offspring1, offspring2
```

Listing 1: Position-Based Crossover Implementation

4 Results

4.1 Experimental Setup

Describe the experiments conducted, including parameters and datasets used.

4.2 Performance Analysis

Present and analyze the results, using figures and tables as necessary.

4.3 Discussion

Interpret the results, discussing their implications and any observed patterns.

5 Conclusion

5.1 Summary

Summarize the key findings and contributions of the project.

5.2 Future Work

Suggest potential areas for future research or improvements.

Appendix A Include any supplementary material, such as additional code or data.