UNIVERSITAT ROVIRA I VIRGILI

A4: Optimization

Researcher: Kamilla Ten
Date: 12/01/24

Description of Implementation

A chromosome in this genetic algorithm is essentially a sequence that represents a possible solution to the Traveling Salesman Problem (TSP). It's like a list of cities in a particular order, denoting the route a salesman might take. Each city in the list has its own coordinates, pinpointing its location.

As the algorithm goes through its iterations, it's trying to find the route with the shortest total distance. Here's how it's adapted to enhance performance:

Selection: Elitism is part of the process, meaning the best routes get a free pass to the next generation. This helps in keeping the good solutions around while keeping things diverse.

Mutation: This part introduces some randomness into the population. It's like shuffling the order of cities to see if a better route comes up. It keeps things fresh but doesn't shake up the basic genetic structure too much.

Crossover: While the specifics aren't laid out here, there's a hint of a crossover step, which is all about mixing and matching parts of two good routes to hopefully create an even better one, much like how mixing genes works in nature.

The number of routes in the population and how likely they are to mutate are set to strike a balance. You don't want the population too small because you might get stuck in a less than the best solution, but make it too big, and you'll need a lot of computing power. The mutation rate is also a balancing act. it's high enough to keep things interesting but not so high that it messes up the good solutions that have already been found.

- Github link: https://github.com/kamillok505/A4.git

Execution Instructuions

Chromosome Representation:

In this implementation of the TSP, a chromosome is represented as a vector or list of City instances, which corresponds to a specific route through all the nodes in the graph.

Each City instance holds x and y coordinates, and the distance method is used to calculate the Euclidean distance to another city.

Selection Techniques:

Parent Selection: Utilizes a fitness proportionate selection where parents are chosen based on their fitness score. The parentSelection function creates a mating pool by randomly selecting routes from the population, with a preference for routes with a higher fitness score.

Survival Selection: Ensures that a certain number of the best routes (eliteSize) are carried over to the next generation without alteration to preserve good traits. This is the concept of elitism.

Mutation Techniques:

The mutate function applies a mutation to a given route with a certain probability (mutationProbability). The mutation swaps the position of two cities in the route to introduce variability.

Crossover Techniques:

The crossover function produces new routes (children) by combining segments of two parent routes. It ensures that each city appears only once in the child route.

An insertGene function aids in this process to ensure that the child routes are valid and do not contain duplicate cities.

Population Size and Stationarity:

The population size (popSize) is chosen to balance between diversity (larger populations) and computational efficiency (smaller populations).

The algorithm checks for a stationary state by observing the best route distance. If there's little to no improvement over several generations, it may indicate that the population has reached a convergence.

Description and Link to the Selected Dataset

Determining the shortest path that permits a salesman to visit each city in a given list once before returning to the starting point is the classic computational difficulty posed by the Traveling Salesman Problem (TSP). The goal is to find the best ordered list of cities that will result in the shortest travel time or distance. The problem is shown as a graph with cities as nodes and distances as edges.

The use of a genetic algorithm is one efficient method for combating the TSP. The solutions are shown as vectors, or "chromosomes," each of which describes a path that passes through every node (or city) in the network. In order to reduce the overall trip distance that each of these chromosomes represents, the genetic algorithm iteratively refines a population of them.

The measurement, calculation, and handling of TSPLIB95 Documentation Site.

To conduct analysis and test the genetic algorithm, one can obtain datasets from multiple online repositories, including the one hosted on John Burkardt's FSU webpage. A range of TSP datasets in a format specifically designed for the TSPLIB library are hosted by this repository. These datasets include a variety of city network problems, from easy to complicate, enabling thorough testing and performance benchmarking of the evolutionary algorithm at various TSP scales. Visit to investigate these datasets:

John Burkardt's TSP Datasets

Evaluation of Results

2. Results for Different Problem Sizes:

The following steps should be documented for different problem sizes:

a. Dataset Description:

Describe the source of your data, for example, "The dataset, sourced from 'st70.txt', consists of 70 city coordinates."

Explain the data format and how it was loaded into the program.

b. Parameter Combinations:

Document the results obtained with at least six different combinations of parameters like population size, mutation rate, and elitism count.

Detail the impact of each parameter change on the algorithm's performance and the distance outcomes.

At each iteration, the population undergoes selection, crossover, and mutation to form a new generation.

The distances of the routes in the population are tracked, and the shortest distance at each iteration is printed.

Final Observations:

The output suggests that the genetic algorithm finds shorter paths as the number of iterations increases.

For instance, the best distance improves from the initial population's 3124.62 units to lower values in subsequent iterations.

The optimal path is the sequence of cities that corresponds to this shortest distance found.

When documenting these results, make sure to include graphs that visually represent the evolution of the algorithm's performance and to discuss the implications of the different parameter settings on the outcomes. Additionally, compare the results across different problem sizes to demonstrate the scalability of the genetic algorithm.

Initial Observation: The genetic algorithm began with an initial population that had a best route distance of 3124.62 units. This initial value serves as the benchmark for the algorithm's performance.

Performance Over Iterations: The algorithm's performance fluctuated over the iterations, which is expected behavior due to the stochastic nature of genetic algorithms. In the initial

stages, the distances showed slight increases and decreases, indicating the exploration of the solution space.

Mid-Evolution Observations: By iteration 100, the best distance was 3303.79 units, showing a general trend of optimization as the algorithm progressed. There was a significant drop in the best distance to 2977.04 units by iteration 222, suggesting the algorithm found a substantially better route through the mutation or crossover processes.

Late-Evolution Observations: The algorithm continued to explore the solution space with the best distance varying around the 3300 to 3500 range for most of the middle to late iterations. This behaviour suggests that the algorithm might have been trapped in a local optimum or that the diversity of the population was not sufficient to escape it.

Final Outcome: By the final iteration, the best distance found was 3383.86 units. The optimal path identified by the algorithm included a sequence of coordinates representing the cities in the order they should be visited to achieve this distance.

Conclusions

The genetic algorithm showed improvement from the initial solution, demonstrating its ability to optimize the path for the Traveling Salesman Problem. However, the fluctuations in the best distances indicate the potential for further optimization, possibly through parameter tuning, enhanced mutation strategies, or by incorporating additional genetic operators.