Escuela Superior de Cómputo
Instituto Politécnico Nacional
Ingeniería en Sistemas Computacionales

Evolutionary Computing

Practice 4: GA's for Combinatorial Optimization

Professor: Jorge Luis Rosas Trigueros
Student: Ayala Segoviano Donaldo Horacio
Creation date: October 3th 2021
Delivery date: October 8th 2021

1 Introduction

1.1 Genetic Algorithms

Genetic algorithms (GAS) are **numerical optimisation algorithms inspired by both natural selection and natural genetics**. Genetic algorithms encode the decision variables of a search problem into finite-length strings of alphabets of certain cardinality, these strings which are candidate solutions to the search problem are referred to as **chromosomes**, the alphabets are referred to as **genes** and the values of genes are called **alleles**.

To evolve good solutions and implement natural selection, we need a measure for distinguishing good solutions from bad solutions, it could be an objective function that is a mathematical model or even a computer simulation that allows to choose better solutions over worse ones. This functions is referred to as **fitness**.

Unlike traditional search methods, genetic algorithms rely on a population of candidate solutions. Once the problem is encoded in a chromosomal manner and a fitness measure for discriminating good solutions from bad ones has been chosen, we can start to evolve solutions to the search problem using the following steps:

1. **Initialization:** The initial population of candidate solutions is usually generated randomly across the search space. However, domain-specific knowledge or other information can be easily incorporated.

2. **Evaluation:** Once the population is initialized or an offspring population is created, the fitness values of the candidate solutions are evaluated.

3. **Selection:** it keeps more copies of the solutions with higher fitness values and imposes the survival-of-the-fittest mechanism on the candidate solutions. The main idea of selection is to prefer better solutions to worse ones, and many selection procedures have been proposed to accomplish this idea, including **roulette-wheel** selection, stochastic universal selection, ranking selection and tournament selection.

4. **Recombination:** Recombination combines parts of two or more parental solutions to create new, possibly better solutions. There are many ways of accomplishing this.

5. **Mutation:** While recombination operates on two or more parental chromosomes, mutation locally but randomly modifies a solution. There are many variations of mutation, but it usually involves one or more changes being made to an individual's trait or traits.

6. **Replacement:** The offspring population created by selection, recombination, and mutation replaces the original parental population. Many replacement techniques such as **elitist replacement**, generation-wise replacement and steady-state replacement methods are used in genetic algorithms.

1.2 0-1 Knapsack problem

The **0-1 knapsack problem** is the following. A thief robbing a store finds n items. The i_{th} item is worth i dollars and weighs w_i pounds, where i and w_i are integers. The thief wants to take as valuable a load as possible, but he can carry at most W pounds in his knapsack, for some integer W . Which items should he take? It is called the 0-1 knapsack problem because for each item, the thief must either take it or leave it behind; he cannot take a fractional amount of an item or take an item more than once.

The problem can also bee seen as: given a set of n items numbered from 1 up to n, each with a weight w_i and a value v_i, along with a maximum weight capacity W maximize $\sum_{i=1}^{n} v_i w_i$ subject to $\sum_{i=1}^{n} w_i x_i \leq W$ where $x_i \in \{0, 1\}$.

1.3 Travelling Salesman Problem

The **Travelling Salesman Problem** objective is to find the shortest route for a travelling salesman who, starting from his home city, has to visit every city on a given list exactly once and at the end, return to his home city. It can be decscribed mathematically as follows:

Given an integer $n \geq 3$ and an $n \times n$ matrix $C = (c_{ij})$ where each c_{ij} is a non negative integer. Which cyclic permutation π of the integers from 1 to n minimizes the sum $\sum_{i=1}^{n} c_{i\pi(i)}$?

2 Material and equipment

Following are the hardware and software used during the realization of this practice. ***Google Colaboratory*** was the tool used for the development of this practice and the next list shows the specifications of the hardware and software provided by Google Colab.

- **Hardware:**
  - **CPU:** Intel(R) Xeon(R) CPU @ 2.30GHz
  - **Memory:** 12GB
  - **Disk:** 108GB
- **Software:**
  - **Platform used:** *Google colaboratory* was used for this practice
  - **Programming language:** Python 3.7.11

3 Development

3.1 Genetic algorithm for the 0-1 knapsack problem

The first thing to do is to determine a way to represent possible solutions to be the chromosomes. In this case, since the problem is naturally binary, that is, take one item or leave it, the representation is going to be a binary string of length n where n represents the number of items, and the i_{th} bit represents taking he i_{th} element if it is 1 and leaving it otherwise. Another thing to consider is the total weight and the total value of each combination of items, so these characteristics will be included in the chromosomes.

For this practice, only 20 items will be used, weights and values are randomly generated, the values are integers between 0 and 100 and weights are floating point numbers between 0 and 1, and the capacity of the knapsack will be 1.

In this case, there are two similar approaches to determine the fitness of chromosomes. For this problem here is a weight restriction, so that needs to be considered while determining a chromosome's fitness, to deal with this problem, a penalization factor will be set to decrease the fitness of a chromosome every time it exceeds the capacity of the knapsack, and the two approaches depend on how big this factor is, if it is small, it can still consider invalid solutions (the ones that exceed the knapsack's weight) as possible candidates, but if the factor is large, then those chromosomes will be discarded. In this case, he big factor will be used to discard non valid solutions.

The approach used for selection is the roulette-wheel, where the fittest elements will have more chances to survive. Crossover will be used for genetic recombination. Mutation will change one bit of the chromosomes created by crossover. And for replacement the elitist approach will be used, where he fittest element of the population will pass by default to the next generation.

3.1.1 Implementation and testing

The genetic algorithm along with the different strategies mentioned above will be coded in Python programming language. Once implemented, the program will be tested to verify that it yields a right and valid solution. Code implementation can be found by clicking **here**.

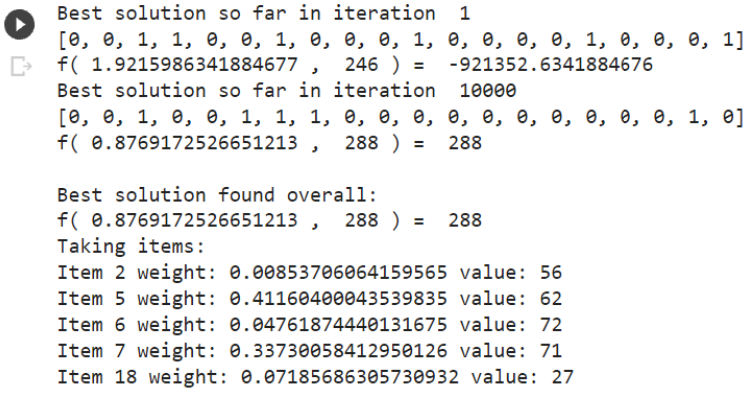

Figure 3.1: GA's output for 0-1 knapsack problem.

Figure 3.1 shows the execution of the genetic algorithm to optimize the knapsack problem. It shows the first and last best solutions found so far. It can be observed that the result obtained in the last iteration is far better than the found in the first iteration, and this is where the big penalization factor comes into play, because in the first operation, the capacity of the knapsack is exceeded, so the fitness value of that chromosome is negative.

It can be observed that the last solution obtained is valid, and it produces a total value of 288 and it shows the different items that need to be taken to achieve this result. The parameters used for this run were only 10 chromosomes, 1000 iterations and a mutation probability of 100%.

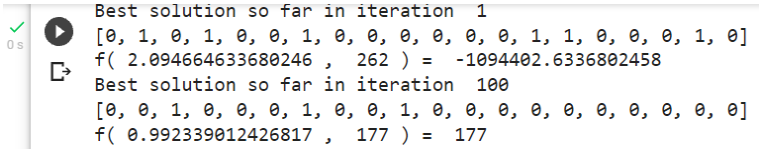

Figure 3.2: GA's output for 0-1 knapsack problem with more chromosomes.

Figure 3.2 shows the output given for the knapsack problem using 100 chromosomes. It can be observed that it gives a valid weight and that it is including different items in the knapsack. So, it can be concluded that the genetic algorithm works correctly.

3.2 Genetic algorithm for the traveling salesman problem

The first step is to determine a viable way to to represent possible solutions, i.e. the chromosomes. In this case, representing the series of cities of a path can be easily represented with a list, where elements represent the number of cities, so, the first position represents the city in which the salesman should start, second city represents the city

to where the salesman has to go first, and so on.

The fitness function for these chromosomes will be the cost of following that sequence of cities. Costs can be stored in a matrix and so to get the cost of going from city i to city j, it is necessary to check the value in the matrix at $M[i][j]$. The sum will be calculated by getting the sum of going from the city at i to the city at $i+1$ with $0 \leq i \leq n-2$ and finally, adding the value of going from the position $n-1$ to the position 0, that indicates the value of returning to the starting point.

Similar to the knapsack problem, for this problem, the approach used for selection is the roulette-wheel, where the fittest elements will have more chances to survive. In this case a special version of crossover will be used, since operations in with this chromosomes are not closed, the approach used will be the **Cycle Crossover**. Mutation will swap two elements in the chromosomes permutation every time new chromosomes are created to generate more diversity. And for replacement the elitist approach will be used, where he fittest element of the population will pass by default to the next generation.

3.2.1 Implementation and testing

The genetic algorithm will be coded in python. Now, the program will be tested to verify that it works correctly and that it gets right solutions. To test the algorithm, 7 cities will be used, and the distances between them, the cities selected are: Mexico City, Montreal, Moscow, New York, Paris, Rio de Janeiro and Rome. The implementation can be found by clicking **here**.

Cities	Mex. City	Montreal	Moscow	New York	Paris	Rio de Jan.	Rome
Mex. City	0	2318	6663	2094	5716	4771	6366
Montreal	2318	0	4386	320	3422	5097	4080
Moscow	6663	4386	0	4665	1544	7175	1474
New York	2094	320	4665	0	3624	4817	4281
Paris	5716	3422	1544	3624	0	5699	697
Rio de Jan.	4771	5097	7175	4817	5699	0	5684
Rome	6366	4080	1474	4281	697	5684	0

Table 1: Resulting table from the execution of the change making problem algorithm.

Table 1 shows the distances between the 7 cities chosen. That matrix will be used to calculate the fitness of each chromosome.

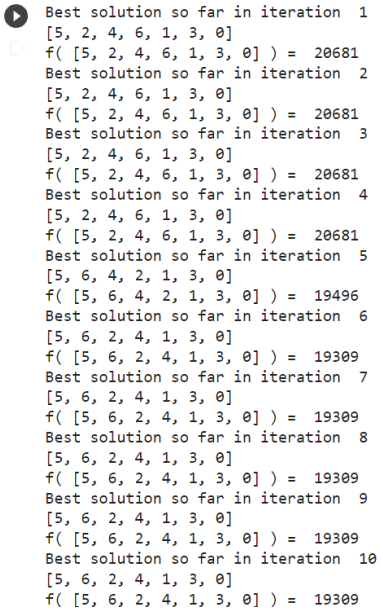

Figure 3.3: GA's output for TSP problem.

Figure 3.3 shows the output of the execution of the genetic algorithm to calculate the shortest path to visit every city only once and returning to the initial one. For this run, 10 chromosomes were used, with 10 iterations and a mutation probability of 50%. After running the program several times, it could be observed that the program always converged to the same value, which is 19309, so it was concluded that this value is the best option to visit all seven cities with the minimum distance.

This way, it was concluded that for the given configurations, the genetic algorithm for the travelling salesman problem, is able to find optimal solutions.

4 Conclusions

The development if this practice resulted in a better understanding of genetic algorithms, it was understood the main parts of a genetic algorithm, and got familiar with them. Also, learned that depending on the type of problem, the chromosomes can be represented in many ways.

It was also concluded that the parameters used for running genetic algorithms can drastically impact on the results obtained and also in the performance of the program. During this practice, it could be observed that some changes impacted in the time taken to execute the programs.

Also, while experimenting with the genetic algorithms, the chromosomes of the knapsack problem were printed to visualize the evolution, and it was easy to see how the chromosomes evolved, and how the genes of the fittest chromosomes are spread to the other chromosomes.

5 Bibliography

- **Larrañaga P., Kuijpers C., Murga R. (1999) Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. Artificial Intelligence Review.**
- **Coley David. (2009). An introduction to genetic algorithms for scientists and engineers. New Jersey. University of Exeter World Scientific.**
- **Thomas Cormen, Charles Leiserson, Ronald Rivest. (2009). Introduction to algorithms. United States of América: MIT Press.**