UNIVERSIDAD DE SEVILLA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

- ARTIFICIAL INTELLIGENCE -

PRACTICAL ASSIGNMENT

Genetic Algorithms for TSP and VRP

Fernando Méndez Requena
José Luis González Chacón

Index

Introduction

This practical assignment consists in develop an implementation of genetic algorithms for solving the Travelling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP).

Travelling Salesman Problem. Find the optimum itinerary for a salesman that needs to visit a set of cities, visiting each city exactly once, except the city where the trip started, that must be the last city to visit.

Vehicle Routing Problem. Find routes for shipping supplies to a set of customers having different demands. The routes should be adjusted to the available fleet of trucks in order to get minimum costs.

Both problems will be analyzed, developed and implemented with Python (Version 3 in our case).

For each problem, we will analyze the genetic operators in the first part. In the second part we will modify the standard of genetic algorithms by some variants and finally we will study the obtained results for giving some interesting conclusions about performance of different algorithms.

First part - Genetic Operators

For this part we have needed to implement a full standard genetic algorithm. It could be found in the attached code of the actual practical assignment.

A general standard genetic algorithm contains the following components:

- INITIATE population
- EVALUATE each individual of the population
- Repeat until HALTING-CONDITION
    - SELECT parents
    - COMBINE pairs of parents
    - MUTATE offspring
    - EVALUATE new individuals
    - SELECT individuals for the next generation
    - RETURN the best individual from the last generation

But in our case, we need to define clearly each aspects of our two assigned problems, because they are differents. These will be defined in the following sections.

Travelling Salesperson Problem (TSP)

Genes

In our case we will consider each city as a different gen. A gen will be composed only by a number that represent the code of each city. In the Andalusian problem, we will consider as many different genes as provinces exists, numbers from 0 to 7

Possible values of genes in Andalusian_TSP {0..7}

Chromosomes

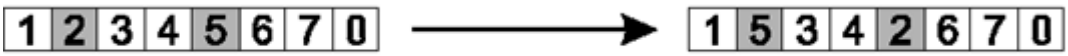
The chosen type of chromosome in our case will be lists of different genes of same size as possible differents cities in our problem. In the Andalusian problem we will consider the size of the chromosome the same as different provinces in Andalucía, 8 in our case.

Andalusian_TSP chromosome example: [0,1,3,2,4,6,5,7]

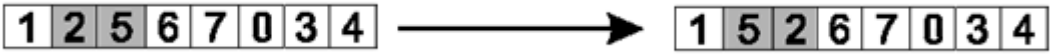
Mutation

There are many different options for choosing our mutation method. We have try two:

In our first case, it has been the random swapping method. This method exchanges the position of two random genes on each given chromosome as we show now:

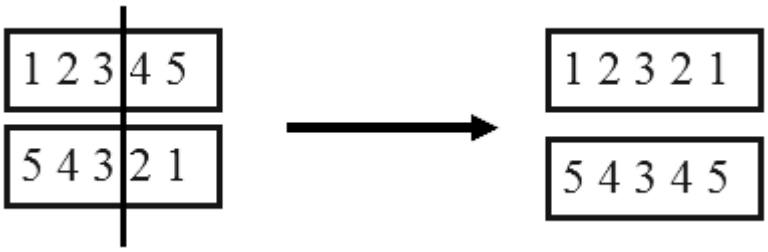


In the second case we have try the mutation by inversion. We need to select a random part of our chromosome, and invert the order of genes of this part. By this method we have found better solutions that with swapping mutation.

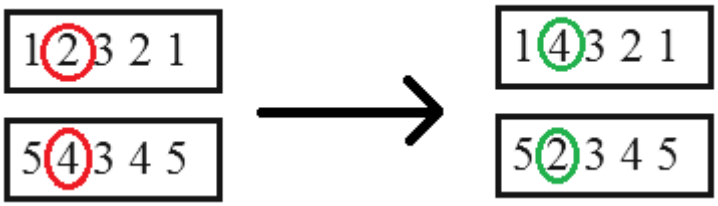


Crossover

Crossover chosen method has been 1 point crossover with ordering fix. Its works by choosing a random place for trunk in two parts the parents chromosomes, then generate two different childs by exchanging those parts by this manner:



By this manner we can generate invalid individuals, because some of them could represent incorrect travels where salesperson goes two different times by the same city. By this reason we need to fix those errors on our two childs generateds with **ordering** crossing.

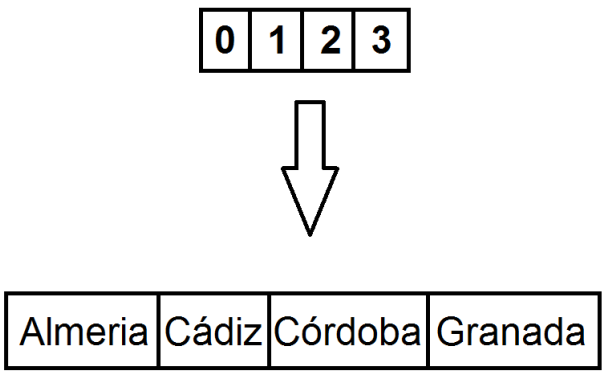


It consists in replacing incorrect genes with next gen available in the part not inherited of the other parent.

Decode

The decode function is only responsible for returning the names of the cities that represent the genes of a given chromosome. Its translate the code of genes from a given dictionary as this:

```
cities = {
     0:'Almería',
     1:'Cádiz',
     2:'Cordoba',
     3:'Granada',
     4:'Huelva',
     5:'Jaen',
     6:'Málaga',
     7:'Sevilla'}
```



Fitness

The fitness function returns the value for each chromosome. This value is a number that represent the ranking of kindness of each chromosome.

For each gen its calculate the totaly of weights. In addition, this function uses the penalty function for calculating this ranking.

The better gens will have a better fitness, so in this case it will be selected first to the next generation of chromosomes.

In this section we asume that the chromosome dont repeat gens, in other wars, the complety path doesnt have any city repeated. See the section “Crossover”.

The fitness function that we use in TSP algorithm is:

$$\text{Pena}lty(Path) = 100\ x\ |Gens\ - Path|$$

$$\text{Fitness}(Path) = 2\ x \sum_{i,j=0}^{7}\left(w_{i,j}\right) + 50\ x\ Penalty(Path)$$

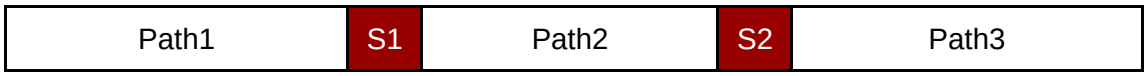
Vehicle Routing Problem (VRP)

Genes

In our case we will consider each city as a different gen. A gen will be composed only by a in a tuple with the number that represents the code of the city and the quantity of requirements of that city. In the Andalusian problem, we will consider as many different genes as provinces exists, tuples with numbers from (0, X) to (7, X) where X is the requirement of each city.

Possible values of genes in Andalusian_TSP {(0, X) . . (7, X)}

In this case of problem (VRP) we need to consider in addition some genes that represents separators of different routes in our problem. For example, if we have 3 deliveries men we need to represent the 3 path, one for each of delivery man. This can be represented by a chromosome that have N-1 separators, where N is the total of deliveries men. (2 separators)



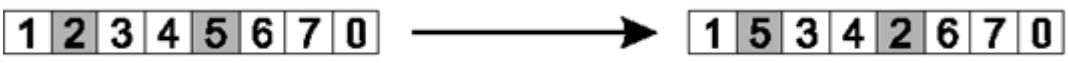
Chromosomes

An chromosome in VRP problem will be a list of tuples where each tuples with previously meaning given of each gen. The size of the chromosome will be the sum of all possible different cities and the separators that we need to use. N-1 separators for N differents path as we explain in the previous part.

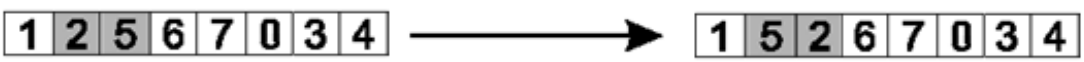
Mutation

There are many different options for choosing our mutation method. We have try two:

In our first case, it has been the random swapping method. This method exchanges the position of two random genes on each given chromosome as we show now:

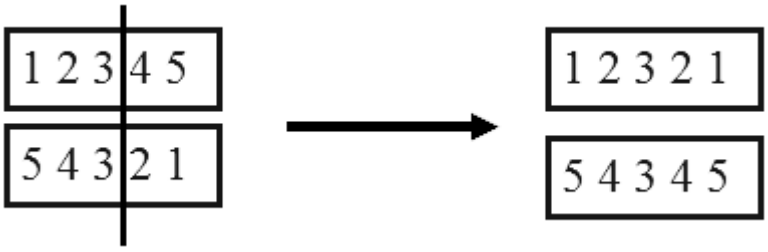


In the second case we have try the mutation by inversion. We need to select a random part of our chromosome, and invert the order of genes of this part. By this method we have found better solutions that with swapping mutation.

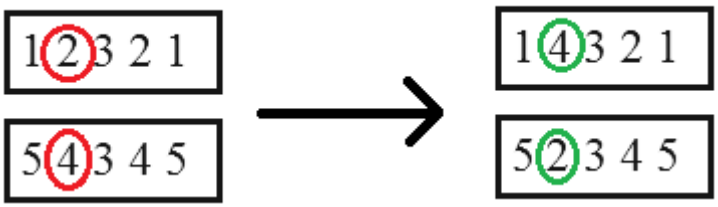


Crossover

Crossover chosen method has been 1 point crossover with ordering fix. Its works by choosing a random place for trunk in two parts the parents chromosomes, then generate two different childs by exchanging those parts by this manner:



By this manner we can generate invalid individuals, because some of them could represent incorrect travels where salesperson goes two different times by the same city. By this reason we need to fix those errors on our two childs generateds with **ordering** crossing.



It consists in replacing incorrect genes with next gen available in the part not inherited of the other parent.

Consider each gen of example as a different tuple. Between city and requirements.

Decode

The decode function is only responsible for returning the names of the cities that represent the genes of a given chromosome. Its translate the code of genes from a given dictionary as this.

We can use the decode function with any chromosome, buy only is needed to understand the meaning of codes of cities. We have explained on the previous part how works the decode function.

Fitness

The fitness function is very different in VRP problem. Now, It needs to evaluate each chromosome for choose it in the next generation and filter bad chromosomes.

In VRP problem, the fitness function, we need to consider if there are exactly N-1 separators and accept the following capacity restriction:

$$Capacity_{city} = (w_{i,j}) \leq Capacity\_trucks$$

The fitness function that we use in VRP algorithm is:

$$\text{Penal}ty_{capacity}(Path) = 100 * Overloads$$

We must assume a Penalty Capacity in order to filter bad chromosomes.

$$\text{Fitness}(Path) = 2\ x \sum_{i,j=0}^{7} (w_{i,j}) + 50\ x\ Penalty_{capacity}(Path)$$

Second part - Variants over the standard GA

In this sections, we are going to study how to introduce some variants over the genetic algorithms previously exposed. We will choose the variants about genetic algorithms called "***Varying population size***". It consists to introduce the concept of "ageing" in TSP and VRP chromosomes.

We must to study in which cases, a chromosome has been generated by a winner parent in a previously tournament selection.

In our case, we will consider chromosomes with old ageing.

To implement it we use a dictionary compose with a pair of tuples, where the "key" is the gen, and his value is a an age.

For initial population we set all values to 1, but in next functions we need to update ages in tournament, to update names in crossover and to update tuples in mutation, deleting each time all old chromosomes not necessaries from our dictionary, form example, chromosomes parents or precious copies after mutate them.

For this purpose we have created a new class called `genetic_algorithm_t2` both for TSP and VRP. It has a new parameter for his initialization, a void dictionary ({})

The function in both problems is the same, but updating our dictionary previously explained generation after generation.

With this method the algorithm prefer to choose chromosomes for next generation with lower age. By that we can know how much punctuation have the best chromosome chosen.

Third part - Experimentation

For this part we have prepared two tests for both problems (TSP and VRP), and for their variants about ageing too.

First we show TSP results for 10 iterations of same test of genetic problem.:

Probe	Best Chromosome P1	Fitness	Best Chromosome P2	Fitness	Age
1	[3,0,6,1,4,7,2,5]	1269	[2,0,3,5,6,4,7,1]	1528	3
2	[6,0,3,5,2,4,7,1]	1273	[5,3,0,6,1,4,7,2]	1269	3
3	[2,4,7,1,6,3,0,5]	1303	[3,0,6,4,7,1,2,5]	1359	2
4	[1,7,4,2,5,0,3,6]	1303	[2,5,0,3,6,7,4,1]	1381	3
5	[2,5,3,0,6,1,4,7]	1269	[1,7,4,2,0,5,3,6]	1440	2
6	[1,4,7,2,5,3,0,6]	1269	[4,1,3,6,0,5,2,7]	1419	1
7	[5,2,4,7,1,6,0,3]	1273	[5,3,0,6,7,4,1,2]	1351	1
8	[2,5,3,0,6,1,7,4]	1273	[2,6,0,3,5,1,7,4]	1414	4
9	[1,4,7,2,5,3,0,6]	1269	[7,4,1,2,5,0,3,6]	1381	2
10	[7,1,2,6,0,3,5,4]	1448	[0,3,4,7,1,6,2,5]	1446	4

Total time for 10 iterations of first part: 27.321 secs, and second part: 37.567 secs

Now we show the results for VRP algorithm and results for 10 iterations in our test:

Probe	Best Chromosome 1st P.	Fitness	Best Chromosome 2nd P.	Fitness	Age
1	['Ma','Se','Ja','Al', '---------','Gra', 'Co','Hu','Ca']	961	['Gra','Se','Hu', 'Al','Ma','Co', '------','Ja','Ca']	1338	1
2	['Gr','Hu','-----', 'Ja','Se','Al','Co', 'Ca', 'Ma']	1064	['Ja','Co','Hu', 'Ma','Se','-----', 'Ca','Gr','Al']	1402	2
3	['Gr','Se','Ma','Al', '--------- ','Co','Ja','Ca', 'Hu']	1031	['Ma','Se','Gr', '--------','Ca', 'Hu','Al','Co','Ja']	1460	1
4	['Gr','Hu','Ja','Co', 'Se','Al','--------- ', 'Ca', 'Ma']	1121	['Ja','Hu','Al', 'Ma','Se','------', 'Gra','Co','Ca']	1233	3
5	['Ma','Se','Ja','Al', '------','Co','Gr', 'Ca', 'Hu']	979	['Gr','Co','Ca','Ja' ,'Sevilla','-----', 'Al','Ma','Hu']	1280	3
6	['Gr','Hu','Ma','Al', 'Se','Co','-------', 'Ca', 'Ja']	1029	['Ja','Gr','Ca', 'Ma','Se','Al', '----','Hu','Co']	1298	2
7	['Ma','Se','Jaen', 'Co', '------', 'Gr','Ca','Hu','Al']	1074	['Ja','Co','Gra','Ma ','Se','----', 'Al', 'Hu','Ca']	1211	3
8	['Malaga', 'Sevilla', 'Jaen', 'Cordoba', '--------', 'Granada', 'Almeria', 'Huelva', 'Cadiz']	1062	['Granada', 'Jaen', 'Malaga', 'Cordoba', 'Sevilla', '--------', 'Almeria', 'Cadiz', 'Huelva']	1287	1
9	['Malaga', 'Sevilla', 'Huelva', 'Jaen', '-------', 'Almeria', 'Granada', 'Cadiz', 'Cordoba']	1132	['Granada', 'Huelva', 'Sevilla', 'Malaga', 'Cadiz', '---------', 'Almeria', 'Jaen', 'Cordoba']	1314	2
10	['Malaga', 'Sevilla', 'Jaen', 'Almeria', 'Cadiz', 'Granada', '---------', 'Huelva', 'Cordoba'],	1013	['Cordoba', 'Huelva', 'Jaen', 'Malaga', '---------', 'Granada', 'Almeria', 'Cadiz', 'Sevilla']	1196	4

Total time for 10 iterations of first part: 28.930 secs, and second part: 45.415 secs

Bibliography

NP- Problems: https://en.wikipedia.org/wiki/BQP

TSP Problem: https://en.wikipedia.org/wiki/Travelling_salesman_problem

GAVaPS - a Genetic Algorithm with Varying Population Size:

http://dlia.ir/Scientific/IEEE/iel2/1125/8059/00350039.pdf

https://pdfs.semanticscholar.org/9db9/2c22e6e34ac3616dc28b89725869c3d780a0.pdf

Cellular Genetic Algorithms : http://tracer.lcc.uma.es/tws/cEA/documents/AGTR02.pdf7

VRP PROBLEM : https://thesai.org/Downloads/Volume2No7/Paper%2019-Solving%20the%20Vehicle%20Routing%20Problem%20using%20Genetic%20Algorithm.pdf

Slides Unit 4:
https://www.cs.us.es/docencia/aulavirtual/pluginfile.php/5152/mod_page/content/4/unit-04-2014-15.pdf

StackOverFlow Python: http://stackoverflow.com/tags/python