Instituto Politécnico Nacional
Escuela Superior de Cómputo
Evolutionary Computing Dynamic Programming
Student: Rodríguez Coronado Ricardo Professor: Jorge Luis Rosas Trigueros
Realization: 25/feb/20 Delivery date: 03/mar/20

Theoretical framework

Genetic Algorithm (GA)

Metaheuristic inspired by the process of biological evolution in a genetic algorithm, a population of candidate solutions (called individuals) to an optimization problem is evolved toward better solutions.

Each candidate solution has a set of properties (it's chromosomes or genotype) which can be mutated and altered.

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each interaction called generation.

In each generation, the fitness of every individual in the population is evaluated, the fitness is usually the value of the objective function in the optimization problem being solved. The move fit individuals (best phenotypes) are stochastically select from the current population, and each individual's gnome is modified (recombined and randomly mutated) to from a new generation.

Commonly the algorithm terminates when either a maximum number of generations has been produced, or satisfactory fitness level has been reached.

A GA requires:

- A representation of the solution.
- A fitness function.

Initialization

Typically contains as many individuals as possible. Often, the initial population is generated randomly allowing the entire range of the possible solutions. The selections may be "selected" in areas of the search species where optimal solutions are likely to be found

Selection

A portion of the existing population is selected to bring a new generation.

Fitter solutions are typically more likely to be selected. An example could be an weighted roulette wheel.

Genetic Algorithm's for combinatorial optimization.

A GA is designed as a combination of patterns in encoding generation, selection, joining, replacement and stopping criteria.

In a typical optimization problem, there is a range of points named search the range and the goal is to find a point leads us to the optimal solution.

An essential characteristic of GA is the encoding of the variables that describe the problem. It is often possible to encode the problem using a set of ones and zeros, but other ideas have been explored.

One of the most important components of a GA is the crossover operator. Two main options remain popular:

Point crossover: The string of the parent’s chromosomes is randomly cut on one or more points of the parent’s chromosomes a new child is created.

Uniform crossover: Any genes of both parent’s chromosomes have a chance to become a part of the child’s chromosomes

Mutation: Acts like an insurance policy against premature less if important information.

A mutation probability can be assigned to each individual and each gene.

The children of the old generation can be replaced with the newly generated children. One possible scheme us to have M existing chromosomes and generate L new children.

The chose M individual from the argued population of (M+L) chromosomes.

Knapsack problem

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. [1]

change-making problem

The change-making problem addresses the question of finding the minimum number of coins (of certain denominations) that add up to a given amount of money. It is a special case of the integer knapsack problem and has applications wider than just currency. [2]

Material and equipment:

Python 3.0+
Python libraries:

- Math
- Random
- Functools

A text editor
PC or Laptop with: Windows 7+/Linux/MacOS X

Practice development
Changing-Making Problem
The objective of the practice is to solve the CTM, but this time using genetic algorithms.

For this problem, the input variables will be, Total (T) = 9; amount of money to be given in exchange, and denominations (D) = [5,2,1]; the value that coins can take. The objective is to give the exact change, with the least amount of coins possible, assuming that the amount of coins of each denomination is infinite.

The problem mentions that three denominations of coins will be used; 5, 2 and 1, this means that we need to somehow represent those values in a string of symbols; a chromosome, for this, we will use a chromosome of 9 bits of length, where each 3 bits will be the amount of coins of the denominations, as shown in Figure 1.

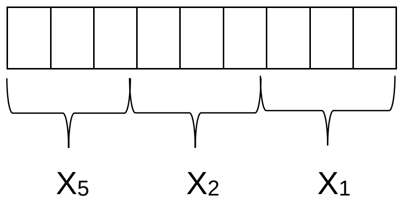


Figure 1, chromosome

Where:

- X_5 Represents the amount of coins of value 5 in binary.
- X_2 Represents the amount of coins of value of value 2 in binary .
- X_1 Represents the amount of coins of value of value 1 in binary.

In this way, we can calculate the total amount of coins that are trying to be given and what value each one is.

The next step is to design a fitness function, this allows the algorithm to determine how suitable a chromosome is and, therefore, how close it is to the optimal solution. The script is intended to result in an amount of coins equal to the one requested (exact 9), so a difference between T and the amount of change that the script is giving, will be used; T(x), as shown in Function 1.

$$f(x) = T - T(x)$$

Function 1. Fitness Function

Defining the function T (x), we would have something like in Function 2

$$T(x) = 5*X_5 + 2*X_2 + X_1$$

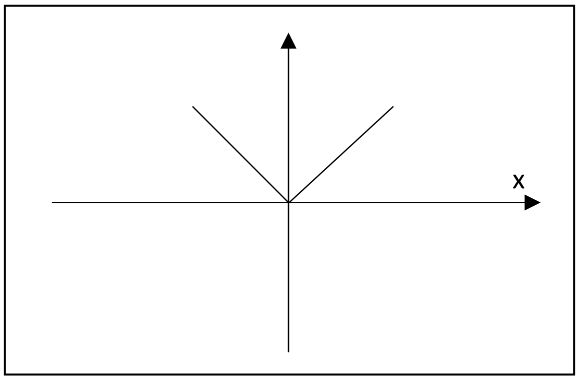
Function 2. Total Function

It is import mentioning that the values of X_5, X_2 & X_1 must be converted to decimal before being multiplier.

In this way we will have a number, which if it is close to 0, it will be closer to be the solution, but it may be the case that the amount of coins giving by the algorithm is larger than the amount of change, resulting in a negative number, so the absolute value of this function will be used, Function 3. In this way we guarantee a positive number and, graphically, a function which has a local minimum, Graph 1.

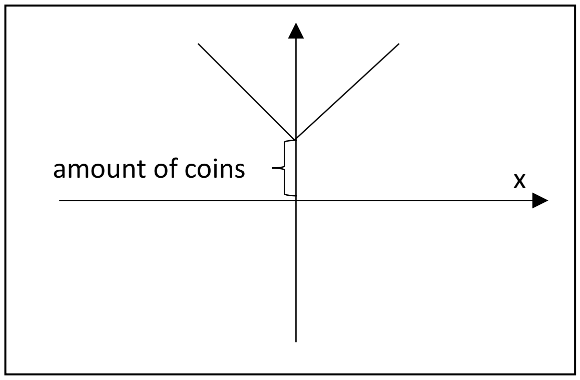
$$f(x) = T - T(x)$$

Function 3. Fitness Function with abs



Graph 1. Fitness Function with abs

At this point, the algorithm will give us solutions which T(x) and T are equal, but the resolution of the topic of using the least amount of coins is still missing. Something must be added to the function, but still search for 0, so it is proposed to add the sum of a scalar to the function, this modifies the graph so that the smaller the number, the graph will get closer to 0, Graph 2. The number to add would be the total number of coins on the chromosome, the final function would be expressed as in Function 4.



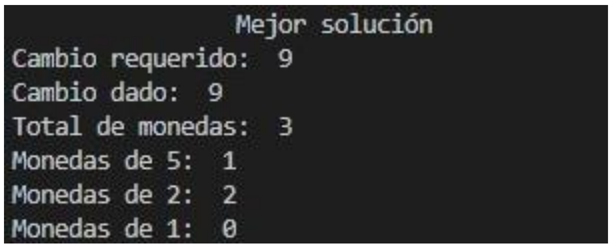
Graph 2. Final Fitness Function.

$$f(x) = T - T(x) + (X_5 + X_2 + X_1)$$

Function 4. Final Fitness Function

It is import mentioning that the values of X_5, X_2 & X_1 must be converted to decimal before being multiplier.

Output:



```
                Mejor solución
Cambio requerido:  9
Cambio dado:  9
Total de monedas:  3
Monedas de 5:  1
Monedas de 2:  2
Monedas de 1:  0
```

Knapsack Problem
The problem of the backpack gives us as input data the capacity of the backpack; C = 11, and the objects, witch will be represented as a list of tuples, where the first value will be the weight and the second the value of an object; ob = [(3.3), (4.4), (5.4), (9.10), (4.4)].

The analysis will be similar to the previous problem, but with some changes in the functions and now instead of looking for a minimum value, a maximum value will be searched.

The chromosomes will have a length equal to the number of objects; in this case 5, in which each bit will represent whether an object enters the backpack or not, 1 - if it entered, 0 - if it did not, as shown in Figure 2.1.

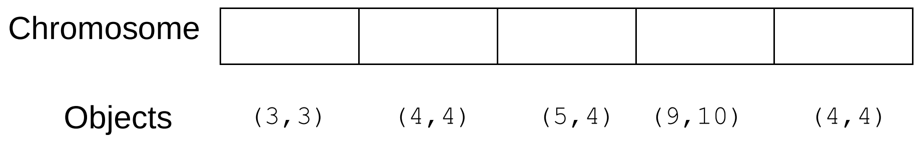


Figure 2.1, chromosome

The next step is the fitness function, in this we must obtain the largest possible value inside the backpack, without exceeding the capacity, that is why the function 2.1 is proposed.

```
if (totalw(chromosome)<=C):
   return totalvalue(chromosome)
else:
   return -1
```

Function 2.1 – Fitness Function

In this case, a conditional is used that ensures that the weight of the chromosomes does not exceed the knapsack capability, because, if the opposite happens, the chromosome will be penalized, giving a value of -1. Otherwise, the total value will simply be returned.

The decoding functions totalw(chromosome) and totalvalue(chromosome) will be defined as follows, Formula 2.2 and 2.3 respectively.

```
def totalvalue(chromosome):
    global C, ob
    r = 0
    i = 0
    for cr in chromosome:
        r += (cr *ob[i][1])
        i+=1
    return r
```

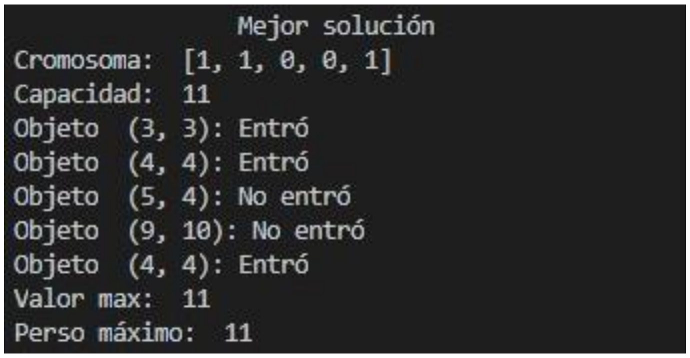
Function 2.2 Total Value of a chromosome

```
def totalw(chromosome):
    global C, ob
    r = 0
    i = 0
    for cr in chromosome:
        r += (cr *ob[i][0])
        i+=1
    return r
```

Function 2.3 - Total weight of a chromosome

In the value function a multiplication is made between the value of one of the cells of a chromosome, by the value of an object related by the same cell, ending with a sum of all the multiplications of a chromosome. In the case of the weight function, it is similar, with the change that the weight of an object is now used instead of its value.

Output.



```
                Mejor solución
Cromosoma:  [1, 1, 0, 0, 1]
Capacidad:  11
Objeto  (3, 3): Entró
Objeto  (4, 4): Entró
Objeto  (5, 4): No entró
Objeto  (9, 10): No entró
Objeto  (4, 4): Entró
Valor max:  11
Perso máximo:  11
```

Conclusions

The objective in the analysis of a genetic algorithm is to represent a possible solution through a chain of symbols; In this case only one chain of binaries was necessary, where the cell number is vital, since with this, we can assign a value to a chromosome which shows how close it is to the optimal solution, and in this way give it priority or penalize it, changing its importance in future populations.

Bibliography

[1] Knapsack Problem. (2020). In Wikipedia. *Retrieved* 03 February 2020 https://en.wikipedia.org/wiki/Knapsack_problem.

[1] Change-Making Problem. (2020). In Wikipedia. *Retrieved* 03 February 2020 https://en.wikipedia.org/wiki/Change-making_problem