Cairo University
Faculty of Computers & Artificial Intelligence
Department of Computer Science
Soft Computing Course - 2020

Lab 1 – Canonical Genetic Algorithm

What is the "Genetic Algorithm"?

The genetic algorithm (GA) is a **search heuristic** and a mathematical **simulation** of the process of **natural evolution**. It uses approximate calculations to solve both constrained and unconstrained **optimization** problems.

Genetic algorithms are commonly used to generate high-quality solutions (not necessarily optimal solutions) to optimization and search problems using biologically inspired **operators** such as mutation, crossover and selection.

The basic steps performed by the canonical GA are:

- Initialization (of the population)
- *Looping over generations and performing:*
    - Fitness evaluation
    - Selection (of parents for reproduction)
    - Crossover
    - Mutation
    - Replacement

Some terminologies used in GA:

Before you understand how to solve optimization or search problems using GA, there are a few terminologies that you need to know:

Chromosome	an individual in the population and consists of an array of genes. It models **one particular solution** to the problem we're trying to solve.
Gene	a particular bit in the chromosome. It models a particular trait in the solution.
Population	a constant number of chromosomes. It represents the **pool of solutions**.
Genotype	the genetic structure (how bits are organized) of a particular solution.
Phenotype	the actual solution itself. It can be encoded to or decoded from genotype.

How to solve problems using GA:

Firstly, you need to think how you can appropriately map solutions to chromosomes. Once you're done with that, you can easily apply the canonical GA. *(We will explore each step in detail)*

- ***Initialization:***

The initial population is generated **randomly**, allowing the entire range of possible solutions (the search space). Occasionally, the solutions may be **"seeded"** in areas where optimal solutions are likely to be found. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions.

- ***Fitness Evaluation:***

The fitness function can be the objective function you're trying to maximize (or its inverse if your objective is to minimize a function). The fitness function is evaluated for every chromosome and stored to be used in the selection process.

- ***Selection:***

In this step, we are trying to select some fit parents to breed. A lot of different selection methods exist like **"Roulette Wheel"** which is one of the popular selection methods used in GA. In the roulette wheel selection, the probability of choosing an individual for breeding is proportional to its fitness; the better the fitness is, the higher chance for that individual to be chosen.

- ***Crossover:***

Crossover is a process of **recombination**. This genetic operator is used to combine the genetic information of two parents to generate new offspring. Crossover is performed with a certain probability ($\mathbf{P_c}$) and at a single random crossover point. (Note: We can also perform **k-point** crossover.)

- ***Mutation:***

Mutation alters one or more gene values in a chromosome from its initial state to maintain diversity. It is also performed according to a certain probability ($\mathbf{P_m}$).

- ***Replacement:***

There are several different replacement schemes such as:

- **Generational replacement:** reproduce enough offspring to replace all population.
  Drawbacks: possibility of losing good genes.
- **Steady-state replacement:** a number of individuals are selected to reproduce, and the offspring replace their parents.
  Drawbacks: possibility of losing of good chromosomes.
- **Elitist Strategy:** like steady-state replacement but allows keeping the best so far.
  Drawbacks: after some time, chromosomes will become a copy of each other.

Exercise:

How can we use the GA to solve the **"Knapsack Problem"**?

Given:

- A knapsack that can carry weights up to **W**
- **N** items; each item (**x_i**) has a weight (**w_i**) and a value (**v_i**)

Objective:

- Select the items to carry in the knapsack in order to maximize the total value.
  (**max.** $\sum_{i=1}^{N} x_i \, . \, v_i$)

Constraints:

- The selected items must fit inside the knapsack.
- The entire item is either selected or not.
  ($\sum_{i=1}^{N} x_i \, . \, w_i <= W$ **and** $x_i = 0$ **or** 1)

Sol.:

We can represent a solution (the items selected) as a chromosome containing **N bits**; each bit (i) corresponds to item (i) where 1 means that the item is selected and 0 means that it is not selected.

Assume W = 10, N = 5 and we are given a list of pairs containing each items weight and value: [(5,4), (4,4), (2,1), (2,7), (4,6)]

Step 1: Let's say we started with following randomly initialized chromosomes:

0	1	0	1
1	1	0	0
1	0	1	0
0	0	0	0
1	0	0	0
C1	C2	C3	C4

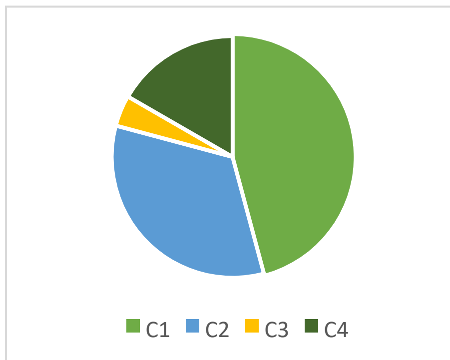
Note: To make sure your mapping works, you can easily decode these genotypes into phenotypes. For example:
C1: C1[0]=0 -> item 1 is not selected, C1[1]=1 -> item 2 is selected, C1[2]=1 -> item 3 is selected, C1[3]=0 -> item 4 is not selected and C1[4]=1 -> item 5 is selected.

Step 2: Let's evaluate the fitness of each chromosome using the objective function:

Chromosome	*Fitness*
C1	11
C2	8
C3	1
C4	4

Step 3: Let's select the parents! First, we need to calculate the cumulative fitness function:

Chromosome	*Fitness*	*Cumulative Fitness*
C1	11	11
C2	8	19
C3	1	20
C4	4	24



Note: Another option is to **normalize** the fitness first by dividing it over the total fitness sum to make it range between 0 and 1 (representing the **probability** of selection) and then calculate the cumulative probabilities.

Second, generate a random number (**r1**) between 0 and 24:
If **0 <= r < 11**, choose **C1**
If **11 <= r < 19**, choose **C2**
If **19 <= r < 20**, choose **C3**
If **20 <= r < 24**, choose **C4**

Assume **r1** = 2.32, therefore C1 is the first parent selected. Generate a random number (**r2**) between 0 and 24 and assume **r2** = 16, so select C2 as the second parent.

Step 4: Let's perform crossover between C1 and C2:
First, generate a random integer ($\mathbf{X_c}$) between **1 and len(C)-1** to be the crossover point.
Second, generate a random number ($\mathbf{r_c}$) between 0 and 1:
If $\mathbf{r_c} \mathbf{<=} \mathbf{P_c}$, perform crossover at $\mathbf{X_c}$**.**
If $\mathbf{r_c} > \mathbf{P_c}$, no crossover. (O1 = C1 and O2 = C2)

Assume $\mathbf{X_c} = 2$, $\mathbf{r_c} = 0.5$ and $\mathbf{P_c} = 0.6$, so this means that we will perform crossover as follows:

0	1	1	0	1

1	1	0	0	0

0	1	0	0	0

O1

1	1	1	0	1

O2

Step 5: Let's perform mutation on the offspring:
Iterate over **each bit** in **each offspring** chromosome and:

- Generate a random number (**r**) between 0 and 1.
- If $\mathbf{r} \le \mathbf{P_m}$, flip that bit.

Question: What will happen if we never performed mutation?

Step 6: Replace the current generation with the new offspring using any of the replacement strategies explained earlier, go to step 2 and repeat the process.