Introduction to Genetic Algorithms

Satyam K & Kamesh K

November 4, 2019

Overview

1. About Genetic Algorithms
   - What are Genetic Algorithms ?
   - Why Genetic Algorithms ?
2. Foundations of Genetic Algorithms
   - Biological Terminologies
   - Principles of GA
3. Algorithmic Approach of GA
   - Defining the problem statement
   - Elements and Operators of Genetic Algorithms
   - Algorithm Flowchart
4. Code Implementation
   - Data Structure and functions
   - Operator functions of GA
   - References

About Genetic Algorithms

What are Genetic Algorithms ?

- A Genetic algorithm (or GA) is a search technique used in computing to find true or approximate solutions to optimization and search problems
- Inspired by the process of natural evolution - **John Holland** introduced genetic algorithms in 1960 based on the concept of Darwin's theory of evolution
- GA's are the most prominent example of "evolutionary computation"
- The Algorithm is adaptive and evolutionary in nature, implying that they continue to perform well in a changing environment as well

Why Genetic Algorithms ?

- Many computational problems require searching through a huge number of possibilities for solutions, making it computational intensive and maybe infeasible in other algorithms
- Particularly well suited for hard problems where little is known about the underlying search space
- Actual problem may take infinite time to obtain the exact global optimal solution
- Wide range of applications are available. Example include Recurrent Neural Networks (RNN's), optimization in operations management including Travelling salesman problem

Foundations of Genetic Algorithms

Biological Terminologies

- The solution space is formulated as a population of living organisms
- Each living organism will have chromosomes which serve as a "blueprint" for the organism
- These chromosomes are further divided to genes and further to alleles, where the union of all the alleles is genes
- Generation refers to the population of organisms at a given time frame
- As time progresses two important processes happens:
  - Mating of organisms resulting in off springs and hence increasing the population size
  - Weak organisms die leading to decrease in population size
- In case of GA, we need to make sure that the population size doesn't increase over generations

Fundamental principles of GA

- The core idea of GA is inspired from evolution theory, the basic principles involves
  - **Selection**
  - **Heredity**
  - **Variation**
- The optimal solution is achieved over generations as the weak organisms die leading to a strong or fit organism for the environment, which is nothing but the optimal solution
- We will see how each of the basic principles are incorporated in GA.

Algorithmic Approach of GA

Problem Statement

- We will consider the **infinite monkey theorem** which states that: A monkey hitting keys randomly will surely type every possible finite text an infinite number of times given infinite amount of time
- The idea is simulate this setup and try to match a given target string using GA Approach
- For this example we are considering the target string to achieved as **"POPCORN"**

Elements of Genetic Algorithms

Population

- We randomly generate the initial population for the given problem to setup the solution space for the GA
- The initial population should capture sufficient variation so that convergence to optimal solution can be achieved faster
- For this case our initial population is:
  - "UNICORN"
  - "POPJORM"
  - "AAAHAAA"
  - "FIRETIN"
  - "DEEDFUL"
  - "PIGOLET"

Elements of Genetic Algorithms

Fitness Function

- The Fitness of an organism is determined by the closeness (or farness) from the optimal solution
- In this case we define our fitness to be proportional to number of matching characters in the strings
- For the initial population the fitness are as follows:
  - "UNICORN" $\longrightarrow$ 4
  - "POPJORM" $\longrightarrow$ 5
  - "AAAHAAA" $\longrightarrow$ 0
  - "FIRETIN" $\longrightarrow$ 2
  - "DEEDFUL" $\longrightarrow$ 0
  - "PIGOLET" $\longrightarrow$ 1
- In this case we will select top 50% fittest population for crossover (Survival of the fittest !)

Operators in Genetic Algorithms

Selection

- A proportion of the existing population is selected based on the fitness parameter to produce off-springs forming a new generation.
- Popular and well-studied selection methods include **rank selection, roulette wheel selection** and **tournament selection**.
- In case of Rank Selection, organisms with best fitness are straight away selected
- For roulette wheel selection the probability of getting selected is given by: $P_i = \frac{f_i}{\Sigma f_i}$
- In case of tournament selection, the following approach is used:

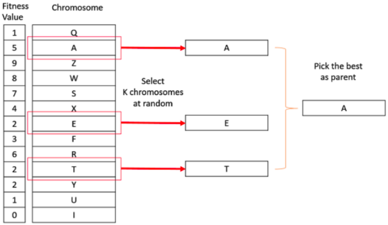

Figure: Tournament selection

Operators in Genetic Algorithms

Crossover → *Heredity*

- To form the next generation, We directly select some percent of the best fitting parents and remaining elements we get by crossover in order to keep the population constant.
- In this step an offspring is generated from two selected parents
- The most common method is single point crossover where you swap the alleles of parents from a fixed locus point. The locus point is selected randomly.
- There are many different types of crossover method like K-Point, **discrete** (implemented), multivariate, average, uniform crossovers.

Single point & Discrete Crossover Table

Parents	Offspring 1	Offspring 2	Offspring(D)
"FIRETIN" "POPJORM"	"FIRETIM"	"POPJORN"	"POPJORN"
"POPJORM" "UNICORN"	"POPCORN"	"UNIJORM"	"POICORM"

Operators in Genetic Algorithms

Mutation → *Variation*

- Mutation is fairly simple. You just change the selected alleles based on what you feel is necessary and move on. Mutation is, however, vital in ensuring genetic diversity within the population.
- The probability of mutation is usually between 1 and 2 tenths of a percent.

Mutation table

Before	**After**
"FIRETIN"	"FIRCTIN"
"UNICORN"	"UNIBORN"

Algorithm Flowchart

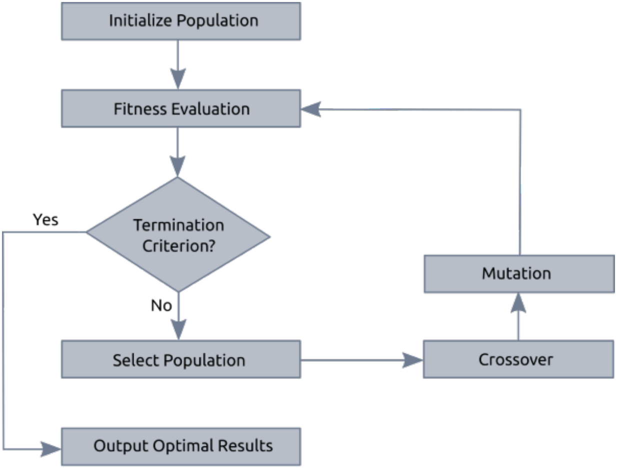

Figure 2: Basic structure of Genetic Algorithm

Code Implementation

Data Structure for Organism

```c
// Strucuture representing organism in population
struct organism
{
    char chromosome[100];
    int fitness;
};
void organism_init(struct organism *org, char chromosome[])
{
    int len=strlen(chromosome);
    for(int i=0;i<len+1;i++)
        org->chromosome[i]=chromosome[i];
    org->fitness = calc_fitness();
};
```

Fitness function

```
// In this case fitness score is the number of matching
    characters in the strings
int calc_fitness(struct organism *org)
{
    int len = strlen(target);
    int fitness = 0;
    for(int i = 0;i<len;i++)
    {
        if(org->chromosome[i] == target[i])
            fitness++;
    }
    return fitness;
};
```

Crossover function

```c
// Perform mating and produce new offspring
void crossover(struct organism par1,struct organism par2,struct
    organism *offspring)
{
    char child_chromosome[100]; // chromosome for offspring
    int len = strlen(par1.chromosome);
    for(int i = 0;i<len;i++)
    {   // random probability p
        float p = gen_rand(0, 100)/100;
        // Selection of Gene happen with this probability
        if(p < 0.45)    // if p < 0.45, gene from parent 1
            child_chromosome[i]= par1.chromosome[i];
        // if p > 0.45 and 0.90, gene from parent 2
        else if(p < 0.90)
            child_chromosome[i]= par2.chromosome[i];
        else // otherwise mututate gene
            child_chromosome[i]= mut_gene();
    }
    child_chromosome[len]='\0';
    organism_init(offspring,child_chromosome);
};
```

Mutation function

```
// Create random genes for mutation
char mut_gene()
{
    int len = strlen(genes);
    int r = rand()%len;
    return genes[r];
}
```

References I

[1] Melanie Mitchell. **An Introduction to Genetic Algorithms**. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262631857.