UNIT 5 U5/0/2024

Machine Learning

COS4852

Year module

Department of Computer Science

School of Computing

CONTENTS

This document contains the material for UNIT 5 for COS4852 for 2024.

1 OUTCOMES

In this Unit you will learn about GENETICALGORITHMS. Specifically you will study:

1. A basic class of GENETICALGORITHMS.
2. The encoding of the chromosomes from a problem.
3. The function and structure of the fitness function and the various operators used in GENETICALGORITHMS.
4. The theoretical basis of GENETICALGORITHMS.
5. Different variants of GENETICALGORITHMS and other types of genetic operators.
6. More advanced versions of GENETICALGORITHMS, specifically their application to real-valued problems.

After completion of this Unit you will be able to:

1. Understand and recognise appropriate learning problems that can be solved using genetic algorithms.
2. Encode a learning problem for use in genetic algorithms.
3. Understand and describe genetic operators.
4. Understand and describe the fitness function and the process of selection.
5. Solve a given problem using the genetic algorithm technique.
6. Understand and describe how genetic algorithms search the hypothesis space.

2 PREPARATION

2.1 Textbooks

One of the definitive textbooks on GENETICALGORITHMS was written by Melanie Mitchell. The book goes into a lot of detail, and will serve well in a standalone course on GENETICALGORITHMS.

2.2 Online material

The notes from the Johannes Kepler University Institute for Mathematical Methods in Medicine and Databased Modeling provides another excellent, detailed set of notes on GENETICALGORITHMS. This is much shorter than Melanie Mitchell's book and was developed for a course on GENETICALGORITHMS. It starts with a simple class of GENETICALGORITHMS (very similar to the example below), with four detailed examples. It continues into more variants of GENETICALGORITHMS, and discusses GENETICALGORITHMS for continuous-valued problems in Chapter 5. The rest of the notes are more advanced, and is not covered in this module.

3 INTRODUCTION

GENETICALGORITHMS are a type of optimisation algorithm, which attempt top find the minimum (or maximum) of a function. GENETICALGORITHMS are part of a broader field in Machine Learning, called Evolutionary Computing. GENETICALGORITHMS imitate the concept of biological evolution and natural selection to find individual solutions that are the 'fittest'. The process in GENETICALGORITHMS are essentially random, but with control over the structure, population, which random processes to use, and measures to search for better solutions.

3.1 A basic algorithm for GENETICALGORITHMS

Most GENETICALGORITHMS consist of the following components:

- A *population* of candidate solutions, encoded in what are termed *chromosomes*, which can reproduce to create new members of the population.
- A *fitness function* that gives a measure of how close to a good solution a *chromosome* is.
- A *selection mechanism* to choose *chromosomes* that will *reproduce* to create new population members. This relies on the *fitness function*.
- Techniques to introduce randomness into the new generations. The most common techniques are *crossover* and *mutation* that either swop bits of chromosome from the parent, or randomly change bits of the chromosome. These have the effect of creating new candidate solution.

A basic algorithm for GENETICALGORITHMS is given in Figure 1.

To get from one generation to the next, there are four basic steps, called operators:

Selection: Use the fitness function to pick a number of individuals in the population for reproduction.

Crossover: Merge the chromosomes of two parent individuals to generate two new individuals. There are various mechanisms to do this.

Figure 1: A basic algorithm for GENETICALGORITHMS

$t \leftarrow 0$
compute initial population β_0
while *stopping condition not met* **do**
    select individuals for reproduction
    create offspring by crossover
    select and mutate some individuals
    generate and select new generation
end while

Mutation: As in real evolution, there are essentially random processes at work (radiation, cancer, viruses, aging,etc.)that cause mutations in genetic material. This same concept is used in GENETICALGORITHMS to randomly make changes in chromosomes to enhance diversity, and thereby increase the probability of finding a solution.

Sampling: Again, as in the real world, individuals die. Sampling is another random process. To keep the algorithm efficient the population is usually kept at a constant number, so that as many individuals are removed as were created during the crossover operation.

3.2 A simple example

Let's look at a simple optimisation task to see how GENETICALGORITHMS works. Consider the problem of maximising (finding the maximum value) the following function (example from Goldberg):

$$f(x) = \frac{-x^2}{10} + 3x$$

where,

$$x \in \mathbb{Z}$$

$$x \in [0, 31]$$

(x is an integer between 0 and 31).

Figure 2 shows this function.

The choice of range and integer numbers here is deliberate. This is to illustrate that you can use a binary encoding of the chromosome. We can encode the x-values as binary strings of 5 bits, from 00000 to 11111, which covers the complete domain of $x \in [0, 31]$. For example, $x = 5$ encodes to the chromosome 00011. Every one of the 32 chromosomes in this case can be a possible solution to this optimisation task.

The first step in the algorithm is to create an initial population. Set the population size to 10, and select 10 chromosomes at random for the initial population.

Table 1 shows the initial population in the population column.

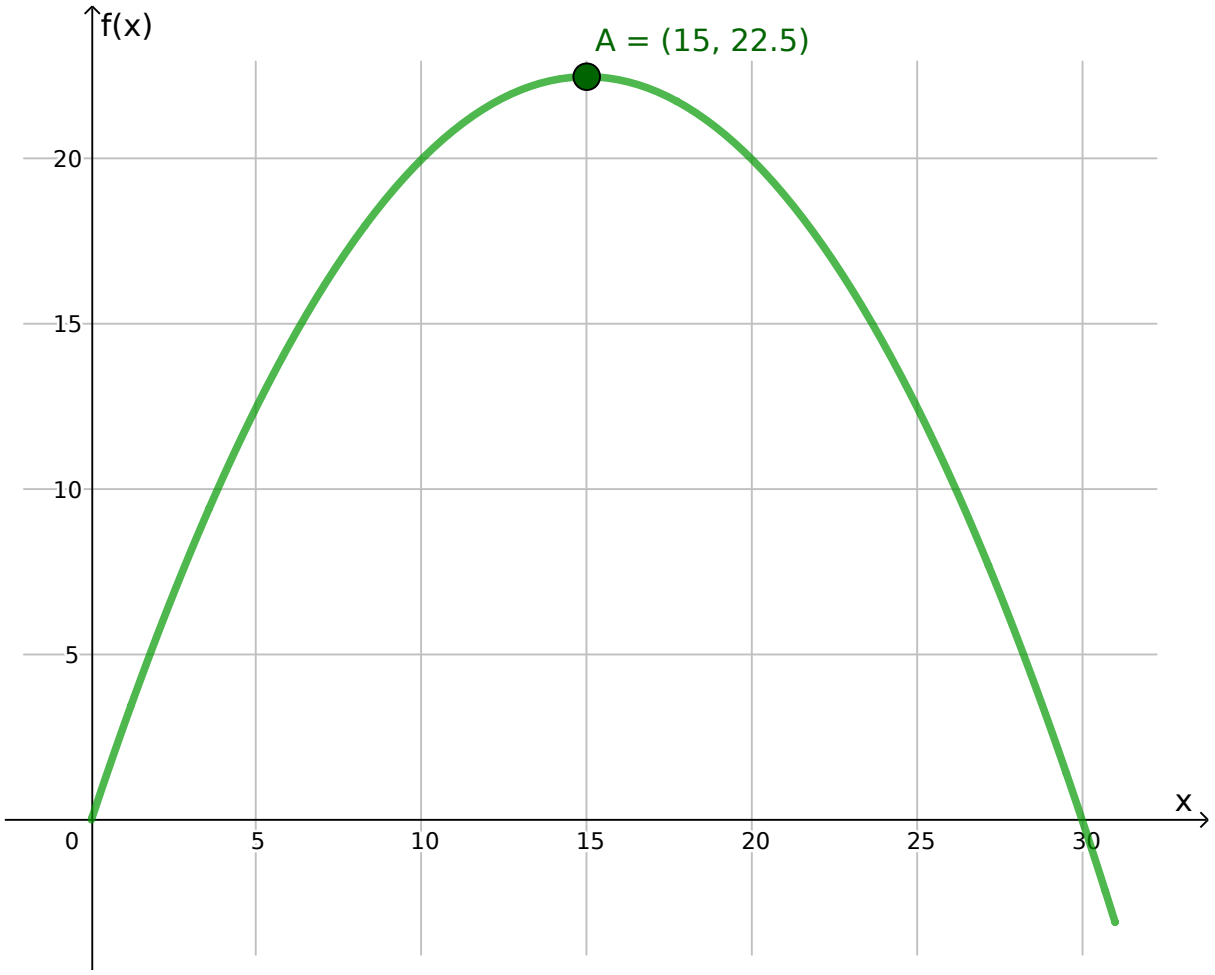

Figure 2: $f(x) = \frac{-x^2}{10} + 3x$

We also define a fitness function that determine how close an individual is to the solution. In this case the fitness function is simply the function we are trying to maximise, $fit(x) = f(x)$. We also need to select which chromosomes will reproduce. Fitter individuals should have a higher probability of being selected. A simple way to do that is to weight the probability by the fitness value:

$$P(i) = \frac{fit(x_i)}{\sum_{k=1}^{10} fit(x_i)}$$

The fitness values and the likelihood of an individual being selected to create the next generation is also shown in Table 1.

We can set a parameter to determine the ratio of individuals to select for reproduction. For simplicity again, we can choose that we want to replace the entire population. To generate a new population of 10 members, and each pairing produces 2 new members, we create 5 pairs randomly. The first column in Table 2 shows the chromosome pairs selected to reproduce.

The next thing to do is to decide on the crossover point in each pair of chromosomes. In this case this is a number from 0 to 4, indicating the position in the chromosome. A crossover point at 0

i	chromosome	x_i	$fit(x_i)$	$P(x_i)$
1	01011	11	20.9	0.1416
2	11010	26	10.4	0.0705
3	00010	2	5.6	0.0379
4	01110	14	22.4	0.1518
5	01100	12	21.6	0.1463
6	11110	30	0	0
7	10110	22	17.6	0.1192
8	01001	9	18.9	0.1280
9	00011	3	8.1	0.0549
10	10001	17	22.1	0.1497
		Sum	147.6	
		Average	14.76	
		Max	22.4	

Table 1: The initial population and calculations.

i	crossover point	mating pairs	new population	x_i	$fit(x_i)$
5	2	01|100	01010	10	20.0
2		11|010	01000	28	5.6
4	4	0111|0	01111	15	22.5
8		0100|1	01000	8	17.6
9	4	0001|1	00110	10	20.0
2		1101|0	11011	27	8.1
7	0	10110	10110	22	17.6
4		01110	01110	14	22.4
10	3	100|01	10010	17	22.1
8		010|01	01010	9	18.9
			Sum	174.8	
			Average	17.48	
			Max	22.5	

Table 2: The next generation

means that the parents are effectively cloned (as with chromosomes 7 and 4 here) (or this could be interpreted as the parents not having any offspring, but surviving to the next generation – the result is the same).

Once we have generated our new population we need to mutate some of them. This is also a parameter that we can set - 0.001 is often a good value here, meaning that 0.001 of the available bits will be swopped from a 0 to 1, or vice versa. This is another random choice. For a population of 10 with 5-bit strings there are 50 bits available to mutate. For our ratio of 0.001 that equals 0.05

bits. To maintain this mutation rate we have to randomly flip a bit once every 20 generations, on average. For illustration purposes, we will flip a random bit in the first generation – the third bit of chromosome 9, in this case.

Here we also see that random selection may result in the same chromosome being selected as the parent of more than one offspring (chromosomes 2 and 8 here), while some don't get selected at all (chromosomes 1, 3 and 6).

In GENETICALGORITHMS with more complex problems, and larger chromosomes the algorithm would typically run for thousands of generations until it is stopped. One stopping criterion could simply be a certain number of generations. A more appropriate criterion would be if the fitness values does not improve for a number of generations. In this case we see that by the second generation both the maximum fitness and the average fitness has increased. The maximum fitness value here is $fit(x_4) = 22.5$, which happens to be the solution (in a real problem we won't know this). So, running the algorithm for more generations will not increase the maximum fitness, but is likely to improve the average fitness. This could be another stopping criterion – when the average fitness approaches the maximum fitness arbitrarily close.

4 ACTIVITIES

4.1 TASK 1 - STUDY THE NOTES

Download the notes mentioned under Online Material above, and do the following:

- Study Chapters 1 and 2 in detail.
- Read Chapter 3 and discuss the relation between the analysis done there and Tom Mitchell's approach to hypotheses.
- Read Chapter 4 to learn more about other variants of GENETICALGORITHMS.
- Study Chapter 5, Sections 1 and 2 on how to implement GENETICALGORITHMS for real-valued problems.

4.2 TASK 2 - IMPLEMENT GENETICALGORITHMS

Find a number of integer-valued and real-valued problems that will be suitable for GENETICALGORITHMS and implement the algorithm to solve these. Pay specific attention to the design of chromosome, the fitness function, and the choice of operators.