Lab 8: Genetic Algorithms and the Max Ones Problem

CMPS 3560 Artificial Intelligence

Alberto C. Cruz, Ph.D.

Department of Computer and Electrical Engineering and Computer Science

Document version February 15, 2015

Prelab

No prelab for this lab.

Introduction

Goal

- Implement a genetic algorithm
- Apply the genetic algorithm to solve the max ones problem

Background – Random Chance

This program is not deterministic. That is, it has a random behavior. In many instances you will need to implement a routine that will on happen only happen a certain percent of the time. For example, let's say that some routine `foo()` should only fire 5% of the time when called. One way of implementing this is to generate a random number, say, between 1 and 100. If the random number is equal to or less than 5, then `foo()` operates normally. Otherwise, the call to `foo()` will do nothing. This will be useful in the algorithm described bellow.

Background – Genetic Algorithms

The max ones problem is a task where a program must fill a bit string with ones. We will apply a genetic algorithm to solve this task. Your program should proceed as follows:

1. Initialize a population of individuals. Each individual is represented by a chromosome. With genetic algorithms, the chromosome must specifically be a bit string. The set of all individuals is called a population. Let the number of individuals in a population be N.

2. **Fitness phase.** In the fitness phase, the fitness of each individual is calculated. The fitness is a function of a single individual. This is a metric that indicates how well an individual achieves the goal. Since we are attempting to fill a bit string with ones, given some individual x_0, the fitness of x_0 is the number of ones. Consider the following population:

Individual x_0									
0	1	1	0	1	0	0	0	1	1
Individual x_1									
1	0	1	1	1	0	1	0	1	0
Individual x_2									
1	0	1	0	1	0	1	1	1	1

So, the fitness of x_0 is 5, the fitness of x_1 is 6 and the fitness of x_2 is 7.

3. Perform a check to see if the termination criteria are satisfied. In our application, this condition is met when a bit string is filled with 1's. If the termination criteria are not satisfied, then proceed. Otherwise, terminate the run.

4. **Mating and Mutation phase**. Create a new, empty population. For each individual, carry out **selection**, **crossover** and **mutation**, and place the offspring of these operations in the new population. The following steps should be carried on an individual per individual basis on the old population. The current individual being iterated over will be called x_i.
   - **Selection**. With a certain percent chance p_c, the individual will select another individual for mating. Initially you should set $p_c = 1$ and experiment with values later on. There are many selection methods. This lab will use tournament selection. In tournament selection, a random subset of the population is selected of size k. k should be much less than the length of the old

population N, e.g. $k = 3$. x_i *should not consider itself for mating*. In the random subset of the population, the individual with the highest fitness is selected for mating with x_i.

- **Crossover**. The two individuals will exchange parts of their chromosome, creating two or more offspring. In our application we will use a method called one-point crossover. In a one-point crossover, you combine one half of individual one and the other half of individual two. E.g., consider two individuals that have been selected by crossover:

Individual 1									
0	1	1	0	1	0	0	0	1	1

Individual 2									
1	0	1	0	1	0	1	0	1	0

A one point crossover of these two individuals will result in the following offspring.

Individual 1-2									
0	1	1	0	1	0	1	0	1	0

Individual 2-1									
1	0	1	0	1	0	0	0	1	1

These are the two offspring that will be placed in the *new* population.

- **Mutation**. In mutation, there is a slight chance that the gene values change randomly of the two offspring will randomly change. For each offspring, let there be some chance p_m that a single bit in a random position will flip.
- Place the two offspring into the new population

5. Repeat step 4 until the length of the new population is equal to N, and replace the old population with the new population.

Technical Approach

Create code that will run a genetic algorithm solving the max ones problem. Some tips:

- An individual is represented by a binary array and an integer representing the fitness of the individual.
- The population should be a list or vector of individuals.
- Let N be 25 and the size of the chromosome is 32 bits.
- Let probability of mutation P_M be 0.01.
- Start with P_c as 1 and experiment with other values later.

You should output the status of the population after each generation:

```
*************************************************
Individual 1: [0010101110] and fitness is 5
Individual 2: [1110000100] and fitness is 4
Individual 3: [0001011010] and fitness is 4
Individual 4: [1011000010] and fitness is 3
...
Individual 25: [1000000110] and fitness is 3
*************************************************
Generation 25: Best individual's fitness is 5,
continuing with next generation!
*************************************************
```

This lab must be completed from scratch without using any evolutionary computation libraries or packages.

Additional Exercises

You are only required to implement the algorithms described in the background section for full credit. However, if you finish early you may want to consider the following tweaks. When implementing the tweaks, you will want to consider how the tweak impacts the number of generations it takes to converge, over a few runs.

Elitism

Elitism is the process where older individuals can enter the next generation if their fitness value is still highly ranked. In textbook only the new individuals—those directly resulting from a crossover—were allowed to form the next generation. One way to implement elitism is to push the old population into the new population at step 5 and trim the population to N members, retaining the individuals with the highest fitness. This heuristic allows our run (simulation) to converge to the solution faster by ensuring that the best individuals move on to the next generation.

Other selection methods

Tournament selection is not the only selection method. You are free to implement a selection method of your choice. Other methods include stochastic universal or fitness proportionate selection (described in the book).

Other crossover methods

We implemented one-point crossover. Even with this method, the point where crossover occurs can vary. There can also be n-point crossovers.