

CAP5512: Evolutionary Computation

Homework 1

http://ivan.research.ucf.edu/classes/CAP5512_Spring2007/

1 Objective

The primary goal of this assignment is to introduce you to the genetic algorithm (GA) and to show you how GA performance can be affected by its structure and parameter settings.

2 Preliminaries

A Genetic Algorithm (GA) is a particular Evolutionary Algorithm. A GA tries to find a solution to a problem by maintaining a balance between the need to discover the new, better solutions (exploration) and the need to make use of existing information (exploitation) about the search domain.

A solution, in a GA, is represented by a string that provides the necessary information. A GA maintains a set of solutions and evaluates how good each of them are. A new set of solutions are formed based on the quality of the existing solutions using selection and crossover operators. A mutation operator is also used to alter some parts of the new solutions. The new solutions are evaluated again and the set of operations are repeated until an acceptable solution is found.

2.1 Components of the Genetic Algorithm

2.1.1 Population

A GA population is composed of a set of individuals. The size of a population is typically predetermined and configurable. Each individual in a population is usually randomly initialized.

Individual (chromosome): Each solution is encoded as a string that includes all of the necessary information pertaining to the solution. GAs usually use binary strings for to represent a solution(i.e. individual) but other representation techniques are also being used to improve the solution quality. The size of each solution can be either variable or fixed.

2.1.2 Fitness Function

“Fitness” of an individual describes how good that particular solution is. (The better the solution the higher the fitness is.) The fitness function is essentially a black box that calculates the goodness of each solution and varies based on the problem type we would like to solve.

2.1.3 Selection

Selection is used to propagate the useful information to new generations. More fit individuals are favored and have greater probability to be chosen to be parents for the next generation. Selection enables the characteristics of the good solutions to be kept in the population. The most common three selection methods are:

Fitness proportional selection: The probability of a solution to be selected as a parent for the next generation, $P(x)$, is proportional to its fitness, $f(x)$.

$$P(x) = \frac{f(x)}{\sum_{i=0}^n f(i)} \quad (1)$$

Tournament selection: We select n number of individuals from the population randomly and keep the most fit one as a parent. The same process is repeated until the selected number of parents reach the population size. Tournament size for binary tournament selection is $n = 2$.

Elitist selection: A predetermined percentage of the top best individuals are kept and copied directly into the next generation. The remaining part of the population are generated from parents chosen by one of the other selection methods.

2.1.4 Operators

Genetic operators are used to explore the search space and introduce new information while maintaining the useful information gained from the previous generations.

Crossover: Two parents are chosen to produce two new children. The most common crossover operators are one-point and two-point crossover.

One-point crossover : Randomly pick a crossover point on the two parents. Combine the left portion of the first parent and right portion of the second parent to produce first child. Combine the right portion of the first parent and left portion of the second parent to produce second child as shown in Figure 1.

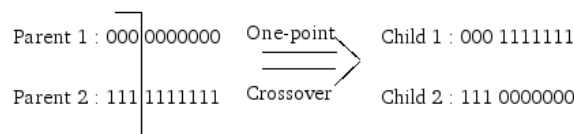


Figure 1: One point crossover example

Two-point crossover : Randomly pick two crossover points on the two parents. Combine the first, middle and last portions of the first, second and first parent respectively to produce the first child. Combine the first, middle and last portions of the second, first and second parent respectively to produce the second child as illustrated in example in Figure 2.

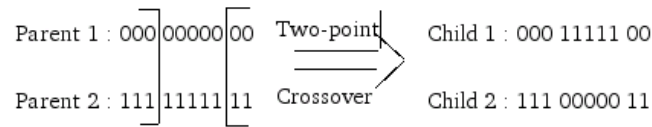


Figure 2: Two point crossover example

Mutation: The child that is produced after crossover operation undergoes the mutation, Figure 3. For binary representation of a solution, every bit has potential of being flipped. We do not ever select a bit to be mutated, instead every bit is flipped with a certain probability predefined as mutation rate.

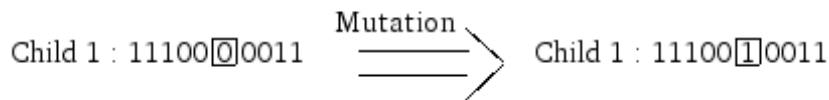


Figure 3: Bit-wise mutation example

3 Coding Options

You are encouraged to write your own code for a simple GA either from scratch or using code samples from the textbook. Your simple GA will be extended in the next homework and it should become the basis for your final project. Alternatively, you can download any of the following open source EC systems, but be prepared to expend time understanding the source code well enough to be able to make code changes and extensions:

- **UCF ECLab Basic GA Code (Java):** Code instructions: (1) Unzip/Untar-ungzip; (2) compile "javac Search.java"; (3) run "java Search onemax.params". Pros: simple to use. Cons: not very well documented, only GA. http://ivan.research.ucf.edu/classes/CAP5512_Spring2007/stringer.ga.tar.gz (or) http://ivan.research.ucf.edu/classes/CAP5512_Spring2007/stringer.ga.zip
- **De Jong's Very Simple EC System (Java):** see Appendix A from the textbook, download code here. Pros: simple to use, well documented, basis for any EA. Cons: very minimal <http://mitpress.mit.edu/evolutionary-computation/DeJong>
- **Luke's ECJ (Java):** ECJ 14 and 15. Pros: complete complex large system, relatively well documented, mostly used for GP but basis for any EA. Cons: complex large system <http://cs.gmu.edu/~eclab/projects/ecj/>
- **Garibay's EC System (C):** Available to students upon request

4 Problem Statement

The goal in OneMax Problem is to maximize the number of 1's in a binary string. We have chosen such a simple problem in order to focus on the basics of the GA rather than on the problem itself. Starting with a randomly initialized set of individuals of binary

strings of length 64, the goal is to obtain at least one single optimal individual of all 1's and to minimize the number of generations required to find a single optimal solution.

Simple GA pseudocode is given below:

```
SimpleGA()
{
    Initialize each individual in the population randomly
    Evaluate the fitness of each individual in the population
    While ( (Generation != Maximum number of generations)
           OR ( Best Fitness of the population != Optimal fitness ) )
    {
        Select parents from the current population
        Crossover between each parent pair w/crossover probability
        For each child
            For each bit of a child
                Mutate bit w/ mutation probability
        Evaluate fitness of each child in the new population
        Copy current children to parents
        Increase Generation number
    }
}
```

A sample GA code (UCF ECLab) that solves the OneMax problem is located at:
<http://ivan.research.ucf.edu/classes/CAP5512.Spring2007/stringer.ga.tar.gz>

- (1) Unzip/Untar-ungzip;
- (2) compile "javac Search.java";
- (3) run "java Search onemax.params".

4.1 Program Input

Individuals in the first generation are initialized randomly. Set the other parameters to the following values unless otherwise stated.

Individual length	: 64 (Binary string)
Population size	: 200
Parent Selection Method	: Tournament, size:3
Crossover type	: two-point
Crossover rate	: 0.7
Mutation rate	: 0.025 (per bit)
Max number of generations	: 300

4.2 Program Output

1. Since the GA is not a deterministic algorithm, it's important to run the GA sufficient number of times to understand the average behavior. Run the GA for 50 times. For each run record the first generation number an individual with the optimal fitness value of 64 is found. Find the minimum, maximum and the average of these 50 values.
2. For each generation the best fitness and the corresponding individual, the average fitness value of each generation and the standard deviation is found and printed in a file named *run.X.genbest*, *X* being the number of the run. Each line of the file should contain the following information:
Gen. num. Averagefitness Standard deviation Best fitness Best individual

Plot generation number versus average fitness and generation number versus best fitness for one of the runs.

3. For each generation calculate the averaged statistics over all 50 runs and write them to file named *RUN.AVG.GENSTAT*. Using the averaged best fitness and averaged average fitness values in this file, plot generation number versus average and best fitness.
4. Keep all the parameters same and run the GA with crossover turned off. (i.e. crossover rate = 0.0). Plot generation versus average/best fitness averaged over 50 runs. Write your observations on how the GA performance is affected by turning off the crossover.
5. Keep all the parameters as stated in section 3.1 except for the population size. Run the GA by setting the population size to 50 and then run it again using population size of 400. Describe the difference you observe in the GA behavior by changing the population size. Support your conclusion by using the plots of generation versus fitness values.
6. Keep all the parameters as stated in section 3.1 except for the mutation rate. Run the GA by setting the mutation rate to two more different values, 0.0025 and 0.25. Describe the difference you observe in the GA behavior by changing the mutation rate. Support your conclusion by using the plots of generation versus fitness values.
7. Using the default parameter settings given above, set the individual length to 128. How does it affect the optimum fitness? State how GA behavior changes and how the results are affected. What kind of tunings can be done for the parameter values (population size, max number of generations, mutation rate) in order to obtain similar behavior as in the case of individual length of 64?

5 Deliverables

- Write up:
 1. Include all the graphs/plots for the program output requirements described in section 3.2.
 2. State your observations and conclusions on the behavior of the GA for each different configuration parameter setting.
- Output files for each scenario explained in section 3.2 and the parameter configuration file.

6 Grading

Plots/graphs of different scenarios: 50%
Interpretation of the output : 50%