CS 361 – Lab #13 – Genetic Algorithms

Today we will experiment with genetic algorithms. We will look at two programs. The first one will be the "digit" program we reviewed in class. The second program will be a sorting algorithm.

Part 1 – Digit program

First, create a folder called Digit, and download the Java source files in the "lab13" folder on the class Web site. Add a copy constructor to the Individual class. Then, modify the last two assignment statements in main() to use the copy constructor. To avoid aliasing errors that are hard to debug, it is probably safer to use the copy constructor when copying objects to objects.

Compile and run the program. The results you get are probably not optimal, so we will experiment. There are many parameters to a genetic algorithm that we can modify. You should try many different modifications, and be on the lookout for ones that achieve higher levels of "fitness", i.e. individuals that are closer to an optimal solution.

Consider the following types of changes to the algorithm. You should not change the fitness function or the number of genes per individual.

- Number of generations (initially 300)
- Mutation threshold (initially 0.001)
- What to mutate (initially we arbitrarily select the first gene)
- Size of population (initially 20)
- How parents are selected (initially: randomly only among the top half)
- How alleles of a child are determined

For the Digit program, fill in the following table to document your findings.

# gens	Mutation threshold	What mutates	Pop size	How parents selected	Other changes?	Top fitness value

Part 2 – Sorting

Can you believe we can use a genetic algorithm to sort? Let's try it and see if it can be done. Create a new folder called Sort. Copy the source code of your Digit program to the new Sort program. The basic structure will be the same because it is still going to be a genetic algorithm. The goal of this program is to sort the numbers 0 to 14 in a 15-element array. Begin your population by setting each individual to a different shuffle (permutation) of the numbers 0-14. Of course, you will need to change the fitness function. For example, an elegant fitness criterion is counting how many cells have a[i] == i.

Once again, it will be necessary for you to tweak the parameters several times in order to get an outcome that is closer to (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14).

For the Sort program, fill in the following table to document your changes and findings.

# gens	Mutation threshold	What mutates	Pop size	How parents selected	Other changes?	Top fitness value