CSE496/896 AI and Heuristics in Software Engineering
Homework II
Handed out: Monday, February 1st, 2016
Due: Monday, February 15th, at 10PM
Late Assignments will be accepted until Wednesday, February 17th at 10PM. A 10% late deduction will apply until Tuesday at 10PM and 20% if submitted by Wednesday at 10PM.

There are two pieces to this assignment. You will implement an algorithm to solve a small problem and answer a set of questions related to this. You must complete and submit both parts.

Assignment: Assume you are testing a program and want to cover a particular branch of the program using a GA. You would normally generate a population of solutions and then run the application and calculate a fitness based on control flow/dependence (as we have seen in the literature). For this homework we are simplifying the problem to make it tractable in a short time frame. You will write a GA for testing a program, but the fitness is going to be based on a known target string that will lead to covering the specific branch you are interested in covering. You will use a chromosome consisting of 17 letter strings. The alleles for each gene are the same – they are the ASCII characters between 'A' and 'z' -- i.e. all upper and lower case letters and the symbols in between these. **The optimal solution in this case is known since we have simplified the problem**. It is the string "**HeuristicSE_andAI**". Write a genetic algorithm to search for the optimal (in this case known) solution.

Part I: The initial parameters you should use for your genetic algorithm are as follows:

- Encode your chromosome as an 17 integer array. Each gene can take one of 58 possible alleles. (You can either use the actual ASCII values or start at 0 or 1. This is your choice)

- Use an initial population of 32 (this population will only be used as a an initialization step). Generate the chromosomes randomly.

- Every generation after the first will have a population of 16.

- At each iteration, select the **best 8** (rank order) and mate using a **one-point crossover** to create two new individuals. Pairing should be done by alternating the individuals matched so that even and odds are matched (i.e. 1 with 3, 2 with 4, etc.). The others (below the best 8) are killed off. At the end of each mating session, you should have a population of 16 again.

- Use a mutation rate of .03

- Use two different fitness functions as follows (both have optimal fitness of 0):

    - The first function, tests the distance of each chromosome from the optimal solution. It adds the square of this distance for all chromosomes. For instance in a string of length 2 with an optimal solution of "no" and a current solution of "os" would give you a fitness of (1+16=17).

  - The second fitness function uses a binary decision for each distance. If the allele matches it is a 0, otherwise it is a one. The sum of the distances is the fitness of this individual. In the example above the fitness would be 2.

- Run this algorithm 10 times for each of the fitness functions. Record the number of generations required to converge on the optimal solution for each run. Report the max, min and standard deviation for the two fitness functions.

- Remove the mutation step. Run the same experiments. Hint: You may need to set a maximum number of generations as a termination for this part!

Questions:

1) Which of the fitness functions works better on this particular dataset? (you can run more iterations if you want to get cleaner data). If they are both the same you should comment on this as well. When you answer this question you should quantify if your measurement for "better" is in terms of number of generations, reliability of convergence, CPU time, population variation over generations, etc.

2) Comment on the effect of removing the mutation operator in the first part

3) Run some experiments to see how a random generation would do for this problem. Explain how you set up this experiment.

II. Experimenting with the Algorithm

In this section you will manipulate some of the parameters in this algorithm and try to determine what effect they have on the running of the algorithm. You should **choose at least 2** of the parameters for this algorithm from the following list:

- Encoding, fitness function, selection, mutation, crossover, pairing, population size

- For each of the two parameters that you have chosen, select one alternative method or representation. For instance, in selection you may try tournament selection and for mutation you may decide to vary the mutation rate over time. For crossover, you may choose a different crossover and for population you may decide to use a different size. The goal is to pick something that you think will have an impact on the ability of your search to converge (or not) to an optimal solution. You may want to informally try a few things first.

Questions to Answer:

1) For each of the parameters you have selected, run experiments to determine what the effect of manipulating these has on your algorithm. You can select the better of the two fitness functions above as the basis for these experiments. Comment with respect to the specific metrics you have chosen. Explain how you have implemented the alternative parameters.

Visualization:

Create a visualization of the running of your algorithm for at least 2 different settings. You can do this in either 2 or 3 dimensions. (for this part you can choose to visualize the actual algorithm or you may use one of the visualizations we have seen to show population diversity at different stages).

Question to Answer:

1) Comment on what information, if any, the visualization provides to the developer of this algorithm.

Guidelines:

- Your program just needs to satisfy the functionality and clearly show that it does. I am not interested in a fancy/GUI interface, **but** it should clearly print out a solution when it is found or let me know if one is not found.
- Include a short README file with your program giving me directions for compiling and running. List any known issues (i.e. if it isn't working properly, doesn't always converge, etc.)
- Your program must run on the cse server. It can be written in any language that exists on cse as long as I have clear directions for compiling/running. **You may not use any GA libraries for this assignment.** The algorithm must be your own.
- For this program you do not need to spend time working on run time efficiency.
- Your program should be written in a modular fashion so that it is easy to read/comprehend. Submit all source code.

Handing In Your Program:
1) Submit your source code and answers using the cse webhandin: http://cse.unl.edu/~handin/.