CSCI/CMPE 4350 Artificial Intelligence

Spring 2015 **Dr. Laura Grabowski**

Homework 1: Genetic Algorithms – *Max Ones*

DUE DATE: Tuesday, February 17, 2015, 11:00 PM (through Blackboard)

For this program, you will implement a simple genetic algorithm that maximizes the number of 1s in a binary string. Beginning with a population of individual binary strings of length 32, your goal is to obtain at least one optimal solution (an individual of all 1s) and to minimize the number of generations needed to find the single optimal solution. Details of the components of the genetic algorithm (GA) follow. The simplicity of the problem will allow you to focus on the details of implementing the GA rather than on the problem itself.

GA Pseudocode

```
Initialize each individual in the population randomly
Evaluate the fitness of each individual in the population
Initialize generation number
WHILE ( (Generation != Max number of generations)
       AND (Best fitness of the population !=  Optimal Fitness) ) {
       Select parents from current population
       Crossover between each parent pair with crossover probability
       FOR each child
              FOR each bit in child
                     Mutate bit with mutation probability
       Evaluate fitness of each child in new population
       Choose the N best individuals to survive to next generation
       Increase generation number
}
```

Program Parameters

Individual length	32 (binary string)
Population Size (N)	75
Parent selection method	Tournament, size 3
Crossover type	2-point
Crossover probability	0.7
Mutation rate	0.025 (per bit)
Survivor selection	Fitness-based (the 75 best individuals survive to the next generation)
Max number of generations	200

Fitness

For this problem, the optimum individual consists of all 1s (*i.e.,* 32 1s). In general, individuals with more 1s are more fit than those with fewer 1s. Create a fitness function that encapsulates this property and measure the "goodness" of each individual. This can be as simple as using the count of 1s in the representation, from 0 to 32.

Tournament selection

From the population of 75, you will select 16 individuals as a "mating pool," using *tournament selection:*

```
Set current_member = 1
WHILE (current_member <= 16) {
       Pick 3 individuals randomly, with or without replacement
```

```
        Select the best of these 3 by comparing their fitness values
        Denote that chosen individual as i
        Set mating_pool[current_member] = i
        Set current_member = current_member + 1
}
```

Once the 16 individuals have been selected, you may use any method you want to try for deciding the pairs of parents (pick randomly, use order chosen, rank by fitness, *etc.*).

Two-point Crossover

Decide if crossover will be used. For each set of parents, generate a random value *r* in [0, 1). If *r* <= 0.7 (the crossover probability or rate), then use crossover to create offspring. Otherwise, simply copy each parent to a child (*i.e.,* the children are identical to the parents).

Do crossover. When crossover will occur, you will use *two-point crossover* (illustrated following). For each set of parents, randomly pick two crossover points on the two parents. Combine the first and last portions of the first parent and the middle portion of the second parent for the first child, and the first and last portions of the second parent and middle portion of the first parent for the second child. You will create 16 children from the 8 sets of parents.

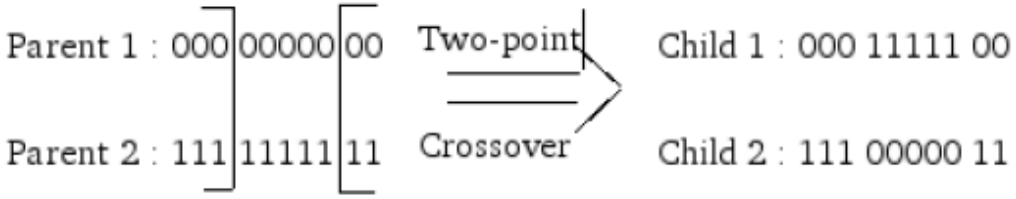

Figure 1: Two point crossover.

Mutation

For each bit in each new child, generate a random value *r* in [0, 1). If the *r* <= 0.025 (the mutation rate), flip the current bit. Otherwise, the bit is unchanged.

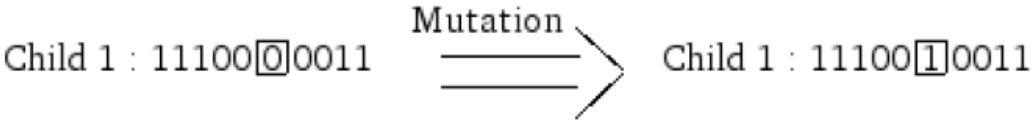

Figure 2: Bit-wise mutation.

Survivor selection

After children are created and mutated, evaluate their fitness. Keep the 75 highest fitness individuals for the new generation (or throw out the 16 worst – same thing, different view).

PROGRAM OUTPUT AND ANALYSIS

Because the GA is not deterministic, you need to run it a number of times to understand its average behavior. **You will run your completed GA 25 times**. For each run, print the first generation number an individual with the optimum fitness value (*i.e.,* a solution) was found and the solution representation vector to an output file (NOTE: be sure your output file is set up to be

appended, not to be overwritten). **Submit the output file as generated by your 25 experimental runs along with your source code and discussion.**

- Find the maximum, minimum, and average of these 25 values. These calculations will require some post-processing of the output file. You may write a short program to do that yourself, or use a tool such as Matlab or Excel. You do not have to submit your code for the analysis.
- Create a scatterplot of the 25 values with the run number (1-25) on the x-axis and the generation number on the y-axis. **Include the maximum, minimum, and average generation numbers when the solutions were found, identified directly on the plot or in a caption for your plot**.
- Put your plot and other information into a word-processed file and include it in your zip file with your program source code.

WHAT YOU WILL TURN IN

You will turn in all of the following files, in a **zipped folder:**

1. Program code – 70%.
2. Output file – 10%.
3. Discussion (.pdf, .doc, ,docx, .rtf formats are all acceptable), including the scatterplot – 20%.

A FEW MORE NOTES AND SUGGESTIONS

- Do not expect every population to produce a solution. Some runs of the algorithm will not find a solution.
- You may occasionally generate a solution in the initial population. If it happens a lot, you need to make sure that you are really randomly generating initial individuals. Remember that you need to start with random **GENOTYPES**, as opposed to random **PHENOTYPES**.
- Make sure that your randomly generated events and choices are indeed random (or as close as we get in the computer).
- If your algorithm is not finding any solutions and you have carefully tested all the steps of the algorithm for correctness, try adjusting the parameters. Adjust parameters in this order: 1) number of generations, 2) population size, 3) mutation rate. Feel free to talk to me if you encounter problems with your algorithm failing completely.

GENERAL REQUIREMENTS

1. You must implement your program in C or C++. **Files must compile on Linux using gcc or g++. I will not compile code in MS Visual Studio. YOU MAY NOT USE ANY ADDITIONAL COMMAND LINE ARGUMENTS, INCLUDING THE EXTENSIONS FOR C++ 11 (see below).** You **are** permitted to use the source code from the course textbook. The source code is available in Blackboard if you don't have the book.
2. Programs will be turned in through the Blackboard assignment. Be sure that you are aware of the due date and time for the assignment. Please turn in **only 1 file (zipped files if you are turning in multiple source files).** When naming your file, **please include your name in the file name**. **Also please be sure your name is in the content of the file itself (in comments for source code, at the beginning of the file for other file types).**
3. Part of your grade will be based on good programming practice, such as use of appropriate data structures, constants, and variables, commenting, and code readability. These issues are handled by way of deductions: the program is first graded for the quality of the solution, then appropriate deductions are taken in the following areas.
   a. *Data structures and other programming practices (up to 20 points deduction):* Program data must be read from files and stored in an appropriate data structure. **Program data must NEVER be hard-coded in a program!** Hard-coded data will result in a significant grade deduction. You are expected to follow good programming practice, such as proper use of constants, avoidance of global variables, and making your program as generalized as possible.
   b. *Comments (up to 20 points):* At a minimum, comments must include a head comments section with general information (programmer name, summary of the program, *etc*.), and comments in the body of the code for major code sections. Please do not hesitate to ask me if you have any questions about my expectations for documentation. Deductions for poor commenting may range up to 20 points.
   c. *Readability (up to 15 points):* You are expected to use meaningful identifiers, and organize and format your code well.

Quick g++ Information

Compiling using g++ on Linux (or Mac) is pretty straightforward.

1. In a Linux environment (or in a Terminal on Mac), navigate to the directory where your source code is located.
2. **IMPORTANT COMPILATION NOTE: Please be sure that your code compiles with the gcc or g++ compiler on the Linux machines in the CS lab, ENGR 2.212. Please DO NOT use any extra command line arguments for compiling. Compiling at the command line without extra arguments looks like this:**

```
g++ mycode.cpp -o myexecutable
```

YOU MAY NOT USE EXTENSIONS FOR C++ 11.

You can find a short tutorial here: http://www.cprogramming.com/g++.html. If you need more help, see me in my office or visit the open lab (ENGR 2.212).