CS 420/594 Biologically-Inspired Computation
Project 4: Genetic Algorithms
Due December 11, 2007

Bruce J. MacLennan

November 27, 2007

Note: *You have two weeks, and we cannot accept late projects and get grades turned in on time. Start right away!*

In this project you will implement a basic genetic algorithm (see Flake, p. 344, Table 20.1). Your program should accept as input:

1. number of genes in the genetic string (ℓ); try $\ell = 20$ to start;
2. population size (N); try 30 to start;
3. mutation probability (p_m, the probability of mutating each bit); try 0.033 to start, and keep it approximately inversely proportional to N;
4. crossover probability (p_c, the probability of single-point crossover); try 0.6 to start;
5. seed for random number generator;
6. number of generations G to run the simulation; try $G = 10$ to start. (A generation replaces all the individuals in the population; hence if a "breeding cycle" replaces two individuals, then there will be $N/2$ breeding cycles per generation. For each breeding cycle select two individuals probabilistically based on their fitness. From these generate two new individuals to add to the new population.)

In this experiment you will use the following real-valued fitness function:

$$F(s) = (x/2^{\ell})^{10}, \tag{1}$$

where x is the integer that results from interpreting s as an unsigned binary number. This is, of course, a very simple fitness function, and we can see that the optimum genotype is all 1s. However, the problem is hard in the sense that the optimum is narrow. (Plot it over $x \in [0, 2^{\ell} - 1]$ to see its shape.) Your fitness function can be either wired into the program, or included from a separate file.

Your program should produce a plot showing:

- average fitness of the population,
- fitness of the best individual,
- the number of correct (1) bits in the best individual,

as a function of time (generation number). Note which bits are most likely to be wrong in the best individual in the final few generations.

Explore the behavior of the GA with various parameter settings:

- population size N,
- mutation probability p_m,
- crossover probability p_c,
- genetic string length ℓ (e.g., try $\ell = 5, 10, 20, 30, 40$),
- number of generations G.

For each set of parameters you will have to do several runs to determine typical behavior for those settings.

Discuss the effect of the various parameters and your algorithm's overall ability to find the best solution. If some of the parameters seem to interact with each other, describe the interactions in your discussion.

For graduate credit (CS 594) or extra credit for undergraduates (CS 420): In addition to the preceding, you will explore the interaction of inheritance and learning ("nature and nurture"). Design your program to run in two modes, *nonlearning* and *learning*. In nonlearning mode the program operates exactly as described in part 1 with $\ell = 20$. In learning mode, every other bit (say, all the odd bits) is determined by evolution, while the remaining (even) bits are determined by guessing (trial and error learning). Thus, the genome size is $\ell = 10$, and the genetic operators determine the inherited bits, but the phenotype, which is tested for fitness, has $\ell = 20$. Each individual is given 20 tries to guess the remaining 10 (non-inherited or learned) bits (each trial guesses all 10 non-inherited bits). An individual's fitness is determined by its best guess over these 20 trials. (Note that it doesn't matter how you initialize the non-inherited bits when you generate new individuals, because they will be guessed, and therefore replaced, before they are tested.)

Your program will also have to be able to switch from the original fitness function (Eq. 1) to this one (Eq. 2), after a specified number of generations:

$$F_2(s) = (1 - x/2^\ell)^{10}. \tag{2}$$

It's easy to see that all 0s is the best genotype. The switch models a sudden change in the environment.

First, from your runs in part 1, pick the parameter setting that seem to work best with $\ell = 20$. In particular, pick a population size and number of generations G^* that leads to good performance, but not complete convergence (all individuals identical).

1. Run you program several times in nonlearning mode to verify its performance.

2. Next, run your program in learning mode (with several random seeds), and compare its performance to that in nonlearning mode. Discuss the similarities and differences. You may also want to compare learning/nonlearning performance with several population sizes.

3. Next we will investigate population adaptability with and without learning. In both learning and nonlearning modes, switch from F to F_2 after G^* generations, and investigate how many additional generations are required for the performance of the population and its best individual to recover (i.e., to return to the fitness levels they had achieved before the switch).

4. Discuss the interaction of learning and evolution, as modeled in your experiments, and discuss its applicability to biological systems.