Travis Rowe

9/10/18

5463 Natural Computing

Report on GA using Bitsets

By implementing a genetic algorithm on bitsets of varying sizes - in this case 20, 40, 60, 80, 100 - we can find interesting information on evolutionary programming with random probabilities. In this instance we use a Population class which has a fixed size of 50. This Population has a vector of bitsets - our Chromosomes - which have various sizes based on the iteration of the program.

We start with bitsets of size 20. The Population is filled with 50 random Chromosome, each bit randomly selected using rand() % 2. Each Chromosome has its fitness calculated by adding the number of ones in the bitset. Note that we are using a seed based on time, that is srand(time(NULL)).

The next Population is generated using a Roulette Selection method. Each Chromosome is given a section of the roulette wheel based on the Chromosome's fitness divided by the total fitness of the entire Population (Chromosome.fitness / totFitness). A random number, r, is generated based on the Population's total fitness (r = rand() % totFitness). Using a running total of the population's fitnesses, each Chromosome fitness is added to the running total until runningFitness >= r. This Chromosome is used to cross with another roulette-selected Chromosome in a one-point Crossover method. Both of the offspring are then run through a Mutation method and added to the next generation. New generations are created using this method until convergence, which is when a Chromosome is created with fitness = Chrom_size. For example, if the size of Chromosomes is 20, then the program will stop when any one Chromosome has a fitness of 20, or when a generation limit is reached, preventing an infinite program.

Initially, I implemented the chance of mutation, pmut, as .01. Each bit in a Chromosome is iterated. On each iteration a random number, r, is generated using rand() / RAND_MAX, giving us a random number between 0 and 1. If r < pmut, then the current bit is flipped, causing a mutation in the Chromosome. With a Chromosome size of 20, convergence is reached on average at generation 51, and the average fitness of converged Populations is 15. With a Chromosome size of 40, the generation limit of 1,000 was reached most of the time. In fact out of 100 times the program ran, only 5 times did the generation not get reached. The average fitness of convergence was 31 and the average generation of these five was 653. At a Chromosome size of 60, there was no convergence at all.

After this I decreased the chance of mutation, pmut, to .005. With a Chromosome size of 20, this made the average generation of convergence 47 with the average fitness of the Population at 16 at the time of convergence. However, at Chromosome size 40, we see a significant increase in convergence. The program reached convergence 29 out of 100 times with a lower pmut. The average generation of convergence was 620 and the average fitness of the Population was 33. Still, at size 60, there was no convergence over 100 iterations of the program.

At a .001 chance for mutation, Chromosomes of size 20 tend to take longer to reach convergence (128 generations), but they have a higher average fitness (18). At size 40, There was a 40% convergence rate at an average of generation of 672 and average fitness of 36. At Chromosome size 60, I still do not see convergence before the generation limit of 1000. At a generation limit of 5000 and over 50 iterations, only 2 convergences were reached, and they were both past generation 2000.

The data seems to show that convergence is difficult with larger strings of bits. The perfect organism is not so easy, it seems. The mutation chance is either so high that good bits get flipped too often, and convergence takes longer to reach. Otherwise, it’s too low and chromosomes become nearly identical, so bad bits don’t get changed often enough. Still, I think that a lower mutation chance tends to have a higher success rate.