Bio-Inspired Artificial Intelligence
Report - Assignment 1 - 2/10/17
Implementing a Simple Genetic Algorithm
Dr. Prof. J. Marshall
Alexander Jermann

Introduction

The goal of this assignment was to implement a simple genetic algorithm, which would apply evolutionary principles on a population of genomes to generate a new population with the goal of finding an 'optimal' genome over several generations. The population consisted of randomly generated 20-bit long binary strings (the genomes), and the goal was to generate a genome with a binary string sequence consisting of only one's, as the optimal solution. The evolutionary principles used to generate a new population (next generation) were: Crossover, mutation, and copying of individuals (not every generation was completely wiped out and replaced). The evolutionary principles were applied probabilistically on the population using inputs by the user.

The Program

The main function of the program is the 'RunGA' function, which takes as input the population size, the crossover rate, the mutation rate and a log file name. After every loop it prints the current generation, the average fitness, and the best fitness. At the end it returns the generation at which the optimal string was found and terminates.
It is important to note that a crossover rate or mutation rate of a certain percentage does not mean that there will be a predefined amount of bits exchanged or mutated. Rather the rate gives a probability with which two genomes will be crossed-over or a genome bit mutated. If the mutation rate was 50% for example, not every second bit would be mutated, instead every bit would be flipped with a 50% probability. The difference seems to be minimal but is in fact of paramount importance to constitute diversity in the population.

Experiments

The experiments were designed to measure the influence that evolutionary parameters had on the outcome of the program, and thus to test the efficiency of the program to find the optimal solution. The experiment was done by only altering one variable and isolating all others to clearly see the effect of a parameter.
For the first experiment the function 'RunGA' was executed 50 times. After each execution of 'RunGA' the amount of generations needed to find the optimal solution were written to a log file, which after termination of the program was plotted using gnuplot. The average generations needed was then printed as well.

Dataset 01: Population size: 100, Crossover rate: 0.7, Mutation Rate: 0.01
Dataset 02: Population size: 100, Crossover rate: 0.0, Mutation Rate: 0.01

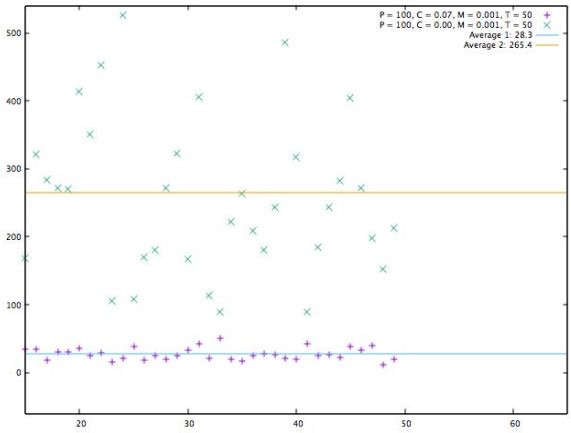


Fig. 01

In the first instance of the experiment, two datasets were compared. Dataset 01 with a crossover rate of 70% and dataset 02 with crossover turned off (thus 0%). Fig. 01 shows that the average generations needed to find the optimal string in dataset 01 was 28.3 generations and thus significantly lower than the average of 265.4 generations needed in dataset 02. The question arises why crossover has such a big influence on average generations needed.
One possible answer would be that crossover recombines large sections of the genome and thus increases the possibility of one genome in the population having a large fitness. This would be consistent with the data collected on one execution of the 'RunGA'.

1	10.63	15.00
2	10.72	15.00
3	11.07	15.00
4	11.61	15.00
5	12.25	16.00
6	12.79	16.00
7	12.92	16.00
8	12.84	17.00
9	12.79	17.00
10	12.66	17.00
11	12.80	17.00
12	13.52	17.00
13	14.00	18.00
14	14.52	18.00
15	14.51	17.00
16	14.84	19.00
17	15.13	18.00
18	15.43	19.00
19	15.58	19.00
20	15.91	20.00

Dataset 01 - RunGA 01

1	10.78	15.00
2	11.11	15.00
3	11.31	15.00
4	11.68	15.00
5	12.03	15.00
6	12.41	15.00
7	12.33	15.00
8	12.52	15.00
9	12.51	15.00
10	12.81	15.00
11	12.83	15.00
12	12.90	15.00
13	13.05	15.00
14	13.34	15.00
15	13.65	16.00
16	13.63	15.00
17	13.68	15.00
18	13.78	15.00
19	14.02	15.00
20	14.04	15.00

Dataset 02 - RunGA 02

First column: Generations, Middle Column: Average Fitness Population, Last Column: Max Fitness

We can see that the average fitness in ‘Dataset 02 - RunGA 02’ is consistently close to the maximum fitness, meaning that the whole population approaches the maximum fitness at a similar rate. This requires a lot of computing power which is redundant if the goal is only to find the maximum fitness and not generate the highest average fitness. Over 20 generations the maximum fitness stays largely the same.
In ‘Dataset 01 - RunGA 01’ the maximum fitness increases a lot faster and in turn also contributes to a higher average towards the end, but there is still a significant distance between the average fitness and the maximum fitness.

Results

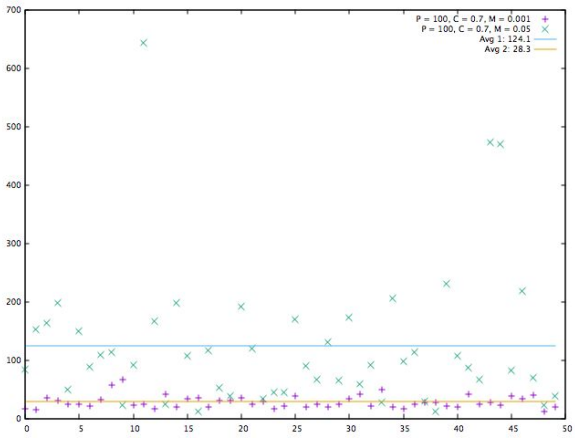
Exper. Nr.	1	2	3	4	5	6	7	8	9
Population	100	100	100	100	100	100	100	100	100
Crossover	0.7	0.0	0.7	0.7	0.5	0.2	0.7	0.7	0.9
Mutation	0.001	0.001	0.01	0.05	0.001	0.001	0.0	0.0001	0.001
Avg. Gen.	28.3	265.4	27.52	124.1	36.24	69.86	undef.	30.24	22.44

No Mutation

When mutation was turned off, the average generations needed to find the optimal solution seemed to be very low (around 20 generations on average), until in one run it seemed to get stuck. The function had already calculated 1400 generations and the maximum fitness didn’t seem to go above 19. The conclusion can be drawn that the function reached a local maxima. Since there was no mutation and thus no variety, there was no way out.

High mutation rate

The experiment shows that a high mutation rate results in a higher average of generations needed to find the optimal genome. This result could be explained by the fact that a high probability of mutation changes the fitness of a genome with such a high frequency that there is a high degree of randomness added to the function. If we observe the figure we see how scattered the the single solutions are in the function with high mutation rate.



The best results were obtained with a high crossover rate and a low (but not zero) mutation rate.