
Machine Learning (CS 391L) : Spring 2013
Homework 6: Genetic Algorithms

Vineet Keshari **EID**: vk3226 **mailto**:vkeshari@cs.utexas.edu

---

1. Problem Description

Genetic Algorithms emulate natural evolution by maintaining, evaluating and breeding populations of a species through several generations. Each individual is encoded in the form of a genome, which represents its behavior in the environment. The individuals interact with the environment as per their genomes and are assigned a fitness value based on their performance. The fittest individuals in each generation then survive to reproduce and create the next generation through genome crossover and mutation. Over many generations, the individuals of the species may adapt to their environment, which is observed in the form of higher fitness values. Genetic algorithms are useful in dynamic environments, where the environment may change over time. In co-evolution, the environment itself evolves over generations, thereby acting as a "parasite" that competes for higher fitness values against the species. In this experiment, we will use a Genetic Algorithm to evolve a population of Sorting Networks, which can sort a given set of input numbers by sequentially comparing pairs as defined by the network's genome. The networks will be co-evolved against an evolving set of inputs. Our goal is to tune the parameters of the Genetic algorithm to facilitate evolution and obtain an "evolved" network with reasonably high fitness and as small a length as possible.

2. Approach

We begin with randomly generated networks and inputs. The inputs are of length 16 each, whereas the networks can be of variable length. The genetic algorithm used for evolving each of the two species is defined by parameters as described in Table 2.1.

	Genome Length	Population	Breeders	Survivors
Input species	16	64	16	0
Network species	64-1024	256	64	16

Table 2.1: Parameters used for evolving the network and input populations. **Genome Length:** The no. of units in the genome; **Population:** Total no. of individuals in each generation; **Breeders:** No. of individuals that participate in reproduction; **Survivors:** No. of individuals that are duplicated in the next generation.

During reproduction, there is crossover between the genomes followed by mutation, both of which are described in detail in Section 3. We also discuss the fitness functions and selection strategy in Section 3, followed by results in Section 4.

3. Fitness, Evaluation and Reproduction

3.1 Genomes

The input's genome is a sequence of integers in the range $[1, 256]$. The sorting network's genome is a set of ordered pairs, each containing the indices of the numbers in the input that will be compared. The comparisons on the numbers in the input are performed in the order they appear in the sorting network's genome. For inputs of length $n = 16$, the length of the sorting network's genomes is restricted between $n \log n = 64$ and $n^2 \log n = 1024$.

3.2 Fitness

The "sort score" for a {network, input} pair is given by the fraction of ordered pairs in the output that are in the correct order. For input size $n = 16$, there are a total of $n(n-1)/2 = 120$ pairs.

The fitness of a sorting network is its average sort score over all inputs. To keep the length of the sorting networks low, this value is multiplied by $log\,(M-l)\,/\,log\,(M-m)\;\varepsilon\,[0,1]$, where l is the length, $m = 64$ is the minimum length and $M = 1024$ is the maximum length of the network. The logarithmic function decreases slowly at m and rapidly at M, thereby punishing an increase in length at higher values more than that at lower values, in addition to rewarding smaller lengths overall.
For inputs, the fitness is the average sort score on the individual over all sorting networks.

In addition, we also calculate the "actual fitness" of the champion sorting network as the fraction of all binary permutations (2^{16}) of length $n = 16$ it sorts correctly. While this value does not affect the selection process, it is useful to know how well the champion network generalizes to all inputs.

3.3 Selection

In each generation, individual i from the population may be selected for reproduction ("breeder") with probability $P(i) = f(i) / \sum_i f(i)$, where $f(i)$ is the fitness of the individual:

```
[creatureFitness creatureIndices] = sort(creatureFitness, 'descend');
creatureFitness = creatureFitness ./ sum(creatureFitness);
creatureFitness = cumsum(creatureFitness);
for i = 1:creatureBreeders
      r = rand();
      index = find(creatureFitness >= r, 1);
      creatureBreed (:,:,j) = creatures (:,:,creatureIndices(index));
end
```

At the same time, individuals with the highest fitness values ("survivors") from each generation are duplicated in the next generation. There are no survivors in the input population, since we want more variation in each new generation.

```
newCreatures = zeros(2,maxCreatureLength,creaturePopulation);
for j = 1:creatureSurvivors
      newCreatures(:,:,j) = creatures(:,:,creatureIndices(j));
end
```

The other members of the new generation are created by crossover and mutation from randomly selected individuals from the "breeders".

3.2 Crossover

Crossover occurs by splitting the genomes at random points and combining the first part of one individual with the second part of the other. For inputs, the points of split are the same, so that the resulting individual is of the same length. For sorting networks, the points are random, but when combining the two parts we reject a crossover if its resulting length is outside the bounds on length as given in Table 2.1.

3.3 Mutation

In each generation, a mutation may occur in some of the genomes with probability 0.1 for each genome. During mutation, a single unit of the genome (integer for inputs, pair of indices for sorting networks) is changed to a random value.

4. Results

4.1 Evolution

We run the genetic algorithm for 1000 iterations with parameters set to values given in Table 2.1 and selection and evolution strategies as described in Section 3. The change in fitness over generations is shown in Figure 4.1, while the corresponding change in length is shown in Figure 4.2. Figure 4.3 shows the champion network as a result of the algorithm, which has an actual fitness of 0.9978 and a length of 155.

4.2 Champion Fitness

As per Figure 4.1, we see that the fitness increases rapidly for the first 50 generations, and then takes steps towards the optimal value of 1. Note that the champion fitness can both increase and decrease over generations, since the inputs are co-evolving. After each generation, we may also update the actual champion to the champion with the highest actual fitness, calculated as described in Section 3.3, in all generations so far. We do this if the current champion has higher fitness than the actual champion, or if it has the same fitness but is shorter. The actual champion's fitness increases monotonically, and reaches a value close to 1 in 1000 iterations for the example shown.

4.3 Champion Length

Similar to the Champion Fitness, the Champion Length both increases and decreases across generations due to co-evolving inputs, as seen in Figure 4.2. We see that the actual champion length increases initially, since longer networks are more likely to generalize to all inputs, and then gradually decreases to a give a network of shorter length with equal or higher actual fitness. The same trend is also observed for the average population length, indicating that shorter genomes have better fitness values in later generations.

Compared to the minimum possible length of $n \log n = 64$, the champion network we get is just over twice as long and is hence of length $cn \log n$ or $O(n \log n)$.

4.4 Parasite Fitness

The input fitness over generations for the same example is shown in Figure 4.4. As expected, the parasite fitness decreases sharply initially, and then gradually decreases towards zero, indicating that more of the inputs get sorted almost completely by the evolving networks in later generations.

4.4 Restricting Crossover

To prevent losing a major part of the genome, we can restrict the random split point for the sorting network genomes between 1/3rd and 2/3rds of their lengths. The fitness curve we obtain for this technique seems to follows the same trend as before, indicating that both techniques generate similar results due to their overall random nature. We therefore do not explore this option further.

4.5 Varying Mutation Probability

We vary Mutation probability between 0 and 1, and plot the actual champion fitness after 100 generations for each value in Figure 4.5. We see that the fitness decreases as we increase the mutation probability. This is because while mutation allows the exploration of unknown solutions, it can slow down the exploitative nature of selection based on fitness. The fitness after a fixed no. of generations will therefore decrease as mutation probability increases, but the long-term fitness achieved is likely to increase as a larger state space gets explored.

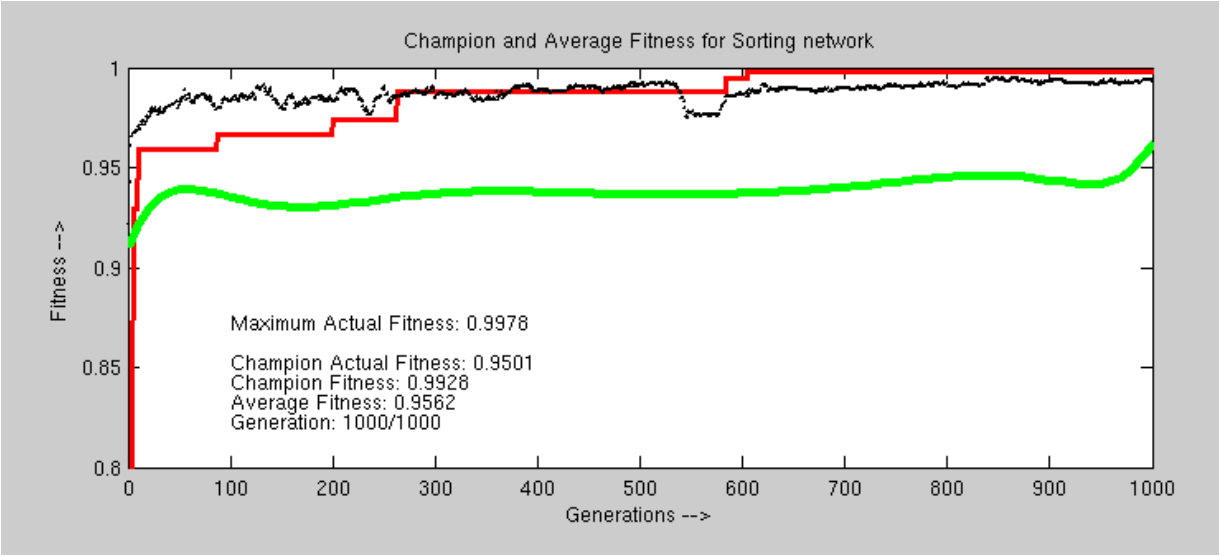

Figure 4.1: Network fitness over 1000 generations. Black dots indicate the champion fitness in each generation, while the red line is the maximum actual champion fitness seen so far. The average population fitness is in green.

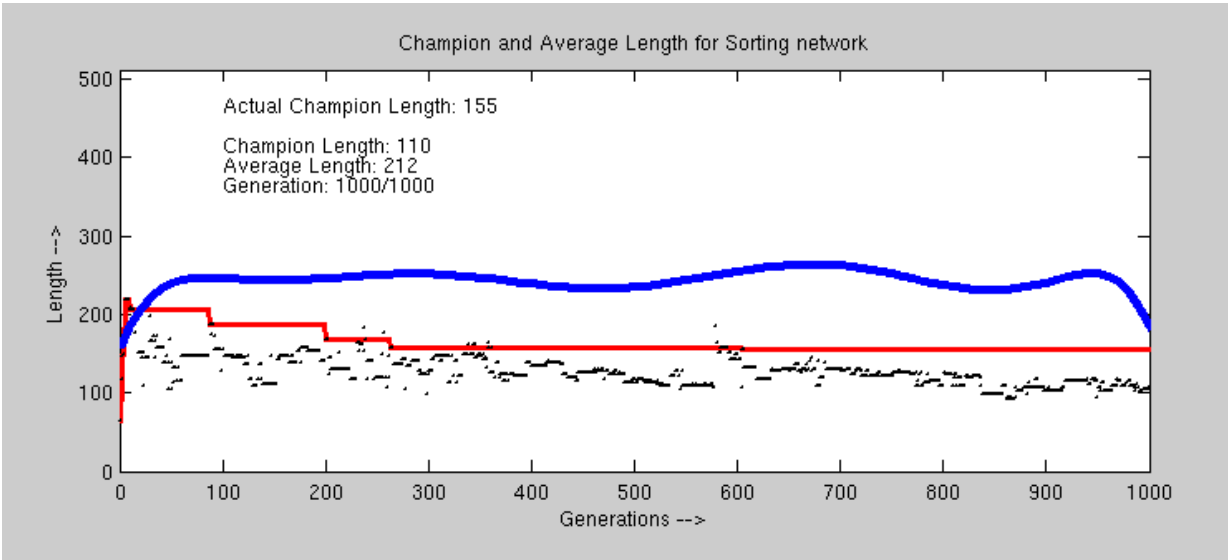

Figure 4.2: Network length over 1000 iterations. Black dots indicate the champion length in each generation, while the red line is the length of the actual champion as plotted in Figure 4.1. The average population length is in blue.

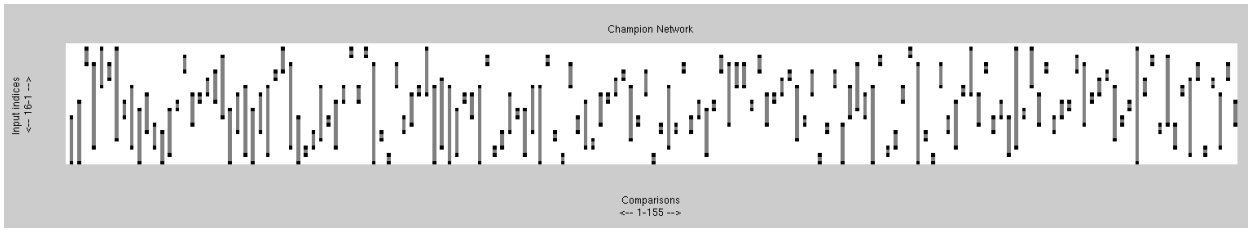

Figure 4.3: The champion network obtained after 1000 iterations. The network has length 155 and an actual fitness value of 0.9978 (on all binary permutations of length 16).

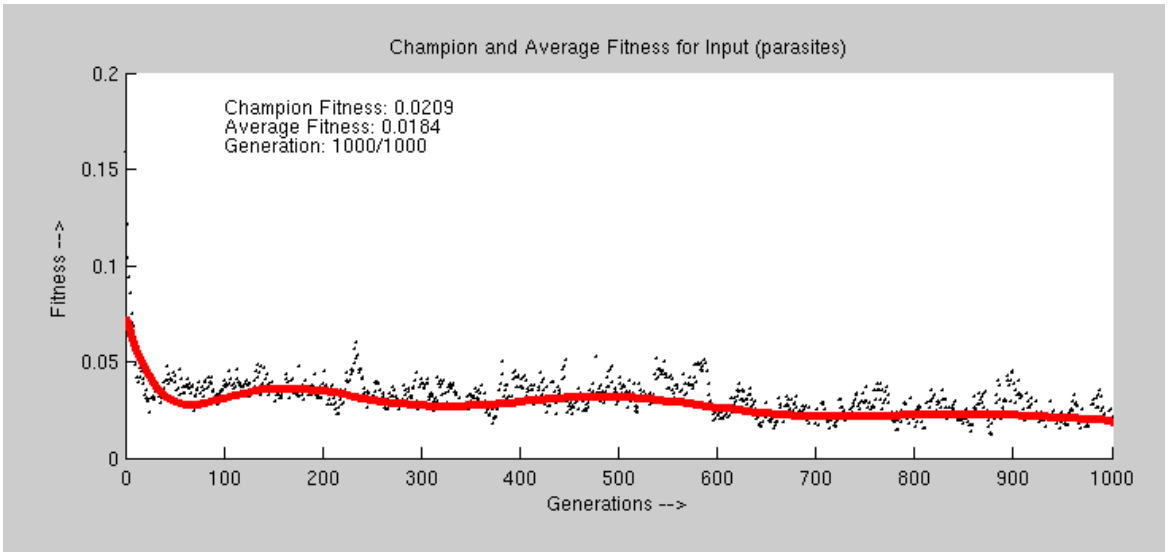

Figure 4.4: Input (parasite) fitness over 1000 generations. The black dots indicate the champion fitness in each generation. The red line is the generation's average population fitness.

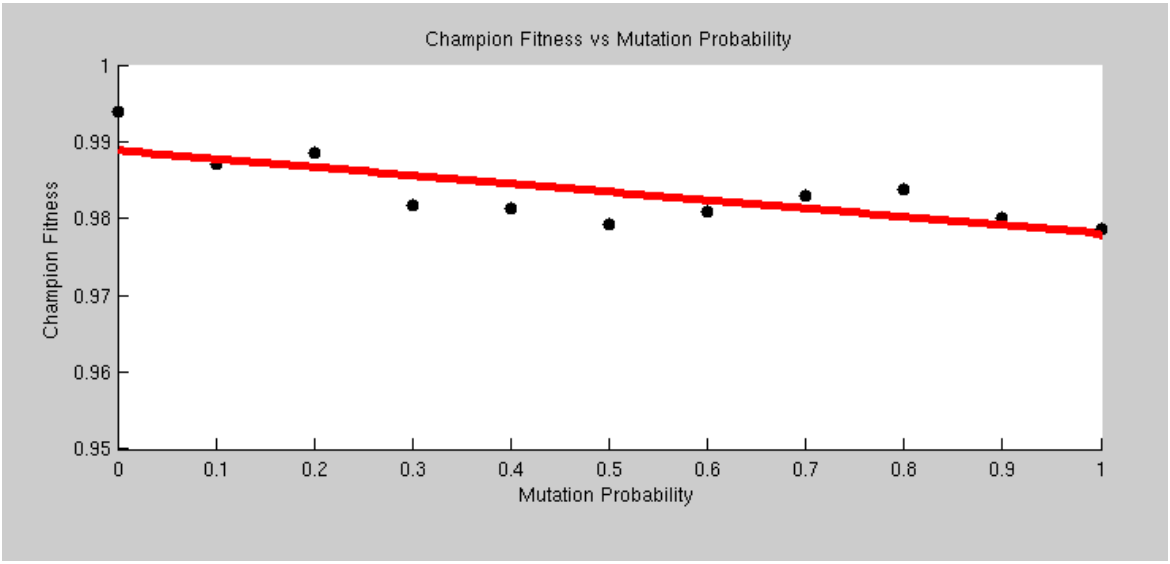

Figure 4.5: Change in champion fitness obtained after 100 iterations for various values of mutation probability. All data points are averaged over 10 runs. The red line shows the linearly decreasing trend.