Akanksha Bansal (ab45563) **Assignment 6**
bansal@cs.utexas.edu **May 12, 2015**

Genetic Algorithm A genetic algorithm (GA) is a search heuristic that mimics the process of natural selection. This heuristic is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

Algorithm The code takes in as input :

- No of solutions to generate in each iteration (n)
- No of inputs to test the solutions in each iteration
- Weight to be given while imposing penalty to the length of the solution while computting fitness
- No of iterations we want the solutions to evolve.

In order to generate the random solutions , I create a pair such that the left value is smaller than the right value. Not doing so would unnecessarily increase the length of the solution set. I also make sure that the swaps are not between the same index. Both these actions reduce the reduce the length of the swap sequence.

One point Crossover was chosen, as the best possible way to generate numerous solutions in a quick manner, with solutions length varying. When the fitness function didn't have any constrain on the

In order to ensure that sorting net is able to sort more accurately the fitness function needs to give credit for each correct swap made and penalize when the sorting nets sequence is too big. I played around a lot with the fitness function and its affect on the final sorting net accuracy. Here are the details :

- When the Fitness function does not penalize for the sequence length, the solutions length tends to explode a lot and however the accuracy of the resulting sorting sequence is very high on all the possible test cases.
- When the fitness function imposes a small penalty for the sequence length. This penalty is imposed by make the fitness function penalized for the normalized the length differences between the sorting networks in the solutions pool, the function still tends to generate sorting sequences of longer length. The conference of the algorithm becomes random. With different weights given for the penalty, the algorithm, might converge or it might not. Then the length of the sorting net in most of the cases would still be more than what is desirable. This is because with a longer sorting sequence the code is able gain a lot more credit in fitness value and shorting the sorting sequence is never beneficial for the algorithm.
- A higher penalty for the sorting sequence length forces the algorithm to pick sorting networks which have smaller lengths. the higher penalty is imposed by increasing the weight of the penalty applied with each increase in the the length of the sorting sequence. However, in some cases because of randomness the code might not converge on good solutions. In order to remove that, I added another piece in the logic where the best solution of the previous iteration

is carried forward to the next. This helped in retaining the correct solutions, and improved the fitness of the overall solution set. In some cases the length of the solution dropped to below the optimal solution length. if there weight parameter is too high. In order to enable the code to recover such scenarios, I make use of mutation parameter. (explained later in detail)

- The better fitness function after experimentation provide to one which gives credit for sorting the test inputs partially and is panelized by squaring the the difference in the length of the solutions and the shortest length in the solution pool. This method is able to proved to be a much better fitness function, as the algorithm slowly tries prunes away the excess swaps in the solutions. And the length of the solutions tends to reduce gradually over generations..
- The best fitness function was where the penalty of length was cube of the length which is greater than the ideal solution length (64 swaps) when used with a < 1 1 weight parameter.. This ensured that the whole set of solution's length is smaller.

Mutation is done with a very low probability(0.01). I tried with couple of variants on mutation. However none of the mutation's I experimented with had any significant change in the result. Therefore my mutation is very simple, the code iterates over every solutions, every swap and with the probability of the mutation frequency generates it again. I did make use of ability of mutation with one more thing. My code has a lot of parameters and because of that in some cases the code might not be able to move forward because the solutions are trying to shorten the sequence length by a huge amount and the solutions tend to become very small, thus it becomes very difficult for the sequences to be able to learn much from one another. If that becomes the case, the mutation probability increases for one generation sequence, so that the solution pool gets in some randomness and the learning algorithm can make progress.

In order make sure that the generation algorithm doesn't only train itself to sort the sample of generated test cases. Therefore, I generate the sample test cases with every iterations. I spent a lot of time on determining the best fitness function and therefore lost time on working on co-evolving the test cases. Ideally, I wanted to add another matrix with the tests cases and make sure that the hard test cases are kept for the nest generation sequence.

Experiments and Results Final testing is done on the code with more no of inputs is more helpful rather than training the code for more no of iterations.

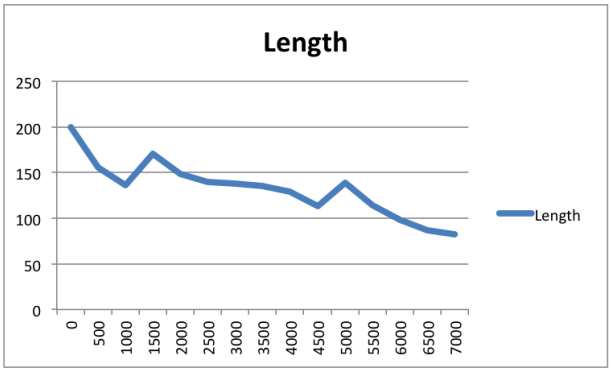

Figure 1: Graph representing the evolution of length with generations.

As can be seen from the graph Figure 1 with increase in generation no. length of the sorting sequence reduces. However, it is not necessary that the decrease in length would be consistent, nor

is it necessary that each step is going to show such a reduction in the sequence length. The reason for this behavior is the fact that there is a to of randomness in the way solutions are generated. The fitness function which is used to determine the fitness of the solution being generated at crossover is not very accurate. As it is not able to penalize the solution much for the length, because it doesn't know that the length of rest of the solutions.

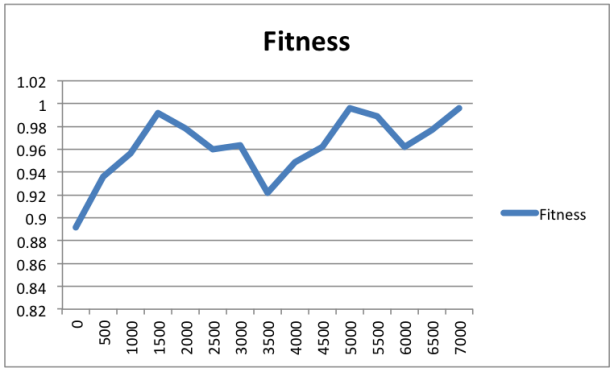

Figure 2: Graph representing the change in fitness with generations.

As can be seen from the graph Figure 2 the model is able to keep the fitness of the solutions above a certain threshold. This is because again the fitness functions try to get the best possible fitness out of the parents. It is not necessary that the fitness of the most optimal solutions would always be on an increase. This could be changed if there is another check on the generative solution is to have higher fitness than its parents.

My final final successful solution had the final accuracy of : 0.908518 and is of the length 82.