Jacob Kilver CS7641 Assignment 2 19 Oct 2014

1. Introduction

Randomized optimization algorithms are useful when analyzing functions that are not well behaved (no derivative) or have many inputs and are thus very complex. In this paper, randomized optimizations algorithms are explored in a number of ways. First, they are used to train a neural network using a data set analyzed in a previous study. The results are compared with the previous study. Second, the performance of several randomized optimization algorithms are examined using several newly constructed fitness functions. The advantages and disadvantages of the algorithms are discussed.

2. Neural Network Training with Randomized Optimization

In the first experiment, randomized optimization algorithms were used to train a neural network using ABAGAIL. The neural network was fixed with 7 input nodes, 2 hidden layers, and 1 output node. The results are presented in Figure 1 and Figure 2 below. Each point represents the number of iterations or the amount of time required to have zero training error for one run. Table 1 lists the averages of these values over 25 runs. The normalized error is explained later.

It should be noted that the number of iterations was bounded for these experiments (1500 for randomized hill climb and simulated annealing and 500 for the genetic algorithm) so these limits did have an effect on the values. However, a close analysis of Figure 1 shows that when the neural network was able to classify all the training instances correctly, it did not approach the limit of the iterations. It can be inferred then that in the cases where the neural network was unable to classify all instances correctly within the iteration limit, the neural network was stuck at a local minima and would not have converged regardless of the number of iterations.

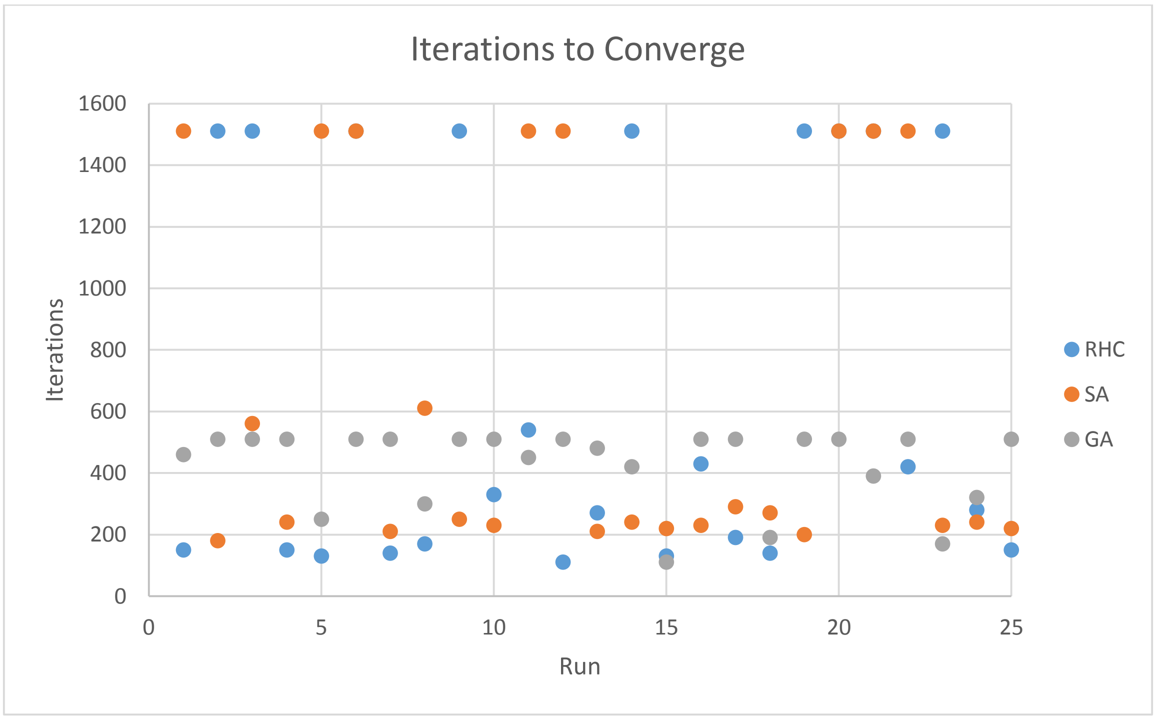

Figure 1: Iterations required to correctly classify all training instances

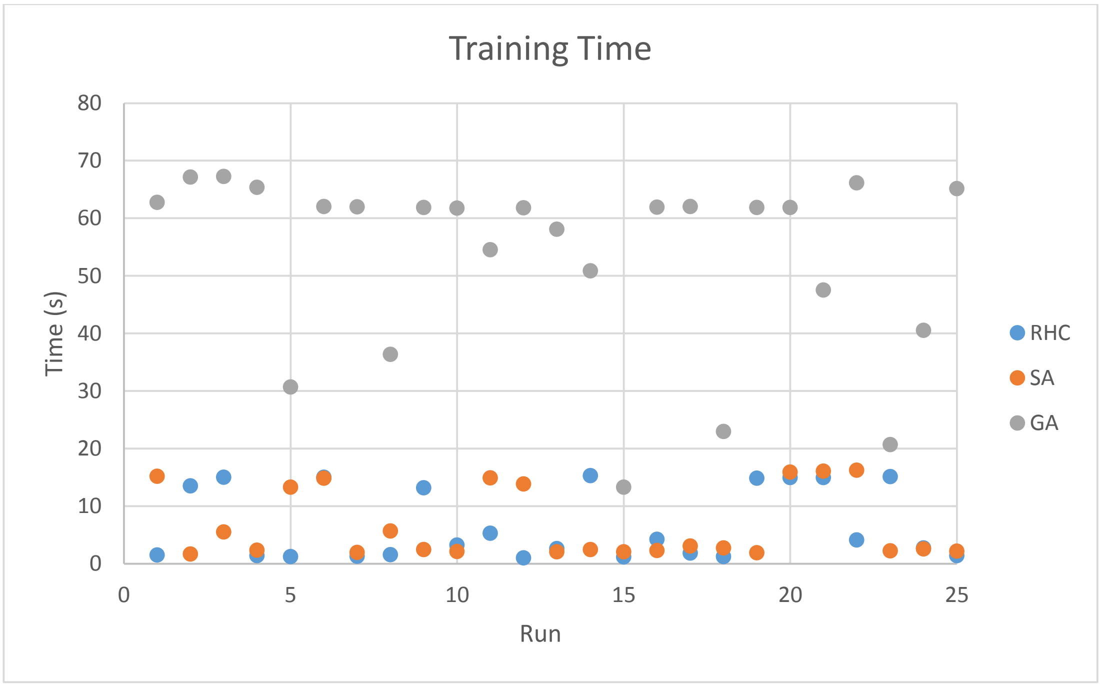

Figure 2: Time required to correctly classify all instances

Table 1: Average performance of randomized optimization algorithms with results from Assignment 1 for comparison

	Iterations	Training Time (s)	Training Error (%)	Normalized Error (%)
RHC	693	6.76	.1458	.4051
SA	668	6.67	.0671	.2099
GA	427	53.10	.4345	.776
Backprop	-	2.17	.4049	.4049

Since randomized optimization was used to train the neural network, consistent results were not expected for each run as in Assignment 1. For the first assignment, the results from backpropagation are deterministic, but the results of randomized optimization are not deterministic, and thus the results of neural network training using randomization could not be deterministic. Therefore, the results from Assignment 1 will be compared in several ways.

First, note that the neural network in Assignment 1 did not have zero training error when the full data set was used. This is the last row in Table 1. In contrast, there were numerous runs where the randomly optimized neural network had zero training error. While this does not make one training method superior to the other, it does show a fundamental difference between the two. With backpropagation training, there is the advantage of consistency. With randomized optimization training, there is not only the chance that the neural network performance exceeds that of backpropagation training but also, as described in the next paragraph, the chance that performance does not exceed that of backpropagation.

Second, the cases where the randomly optimized neural network did not converge are examined. The last column of Table 1 shows the Normalized Error, which in this case simply takes the runs where the neural network did not converge and averaged the training error. As can be seen, the normalized training error of backpropagation is less than all of the randomly optimized neural networks except for one, simulated annealing. However, when the training error is averaged with the cases where there was zero training error, the training error drops significantly.

Finally, the training time will be compared. Backpropagation took the least amount of time to train as compared to all the random optimization algorithms. This is in part due to the assumptions made at the beginning about how the random optimization algorithms would be graded and averaged. It was decided that when a neural network was not able to correctly identify all the training instances within the iteration limit, the full iteration and training time would be used. This is as opposed to running the full iteration, then searching backwards until the first iteration where the local minimum upon which the neural network converged was found and using this data instead. Since even the neural network trained using backpropagation did not have zero training error, this is a valid metric, but it was not used here in order to show that performance of the random optimization network is not the same every time. As such, for the cases where the randomly optimized neural network did not converge with zero training error, the training time and iterations were much larger than the cases where it converged with zero training error. As a result the averages were pushed higher. If the alternate grading scheme were implemented, the random optimization results would be much more comparable to that of backpropagation.

3. Optimization Problems

For the second experiment, three fitness functions were devised to compare Simulated Annealing, a Genetic Algorithm, and MIMIC optimization algorithms. The fitness function assumed bit strings of length N as the input. ABAGAIL was used to test the performance of each of the algorithms with the given fitness function. It should be noted that the number of iterations was bounded for each of the optimization algorithms. Where these bounds significantly affect the results will be presented in the pertinent sections.

a. Fitness Function One

The first fitness function was devised to highlight the advantages of the MIMIC algorithm. Its goal was to find the dependency tree where every right node had the same value as the root of the tree. It was assumed that the inputs were stored in an array, so the fitness function assumed the ordering of nodes as seen in Figure 3. As can be easily determined, this fitness function had two global maxima: one where every even numbered node has a 1 and one where every even number node has a zero, assuming the root has the same value.

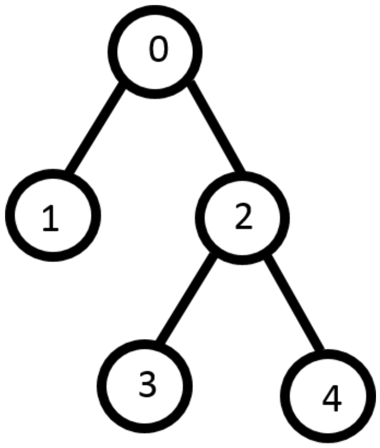

Figure 3: Translating nodes of decision tree to array indices

Figure 4 and Figure 5 present the average number of iterations and the average execution times that were required for the various algorithms to converge as the number of inputs was varied. While the number of iterations was bounded, none of the algorithms approached this limit before they converged. Figure 4 shows that Simulated Annealing took the most iterations to converge for all numbers of inputs. However, the genetic algorithm never required significantly more iterations to converge than MIMIC. In fact, for some input sizes, it actually took fewer iterations. Furthermore, Figure 5 shows that MIMIC takes the most time of all the algorithms to converge.

While it was thought that MIMIC would perform the best with this fitness function, the results do not appear to be that conclusive. The genetic algorithm outperforms MIMIC consistently on execution time and, for some input sizes, the number of iterations. Furthermore, simulated annealing takes the least time of all the algorithms to converge. While MIMIC usually has the least iterations, it always takes the most time to converge. This is because MIMIC generates significantly more information than the genetic algorithm or simulated annealing, but that information is not important when only searching for the maximum value of the fitness function. Nevertheless, the best performing algorithm depends on what is more important: a few computationally intense iterations or many computationally easy iterations.

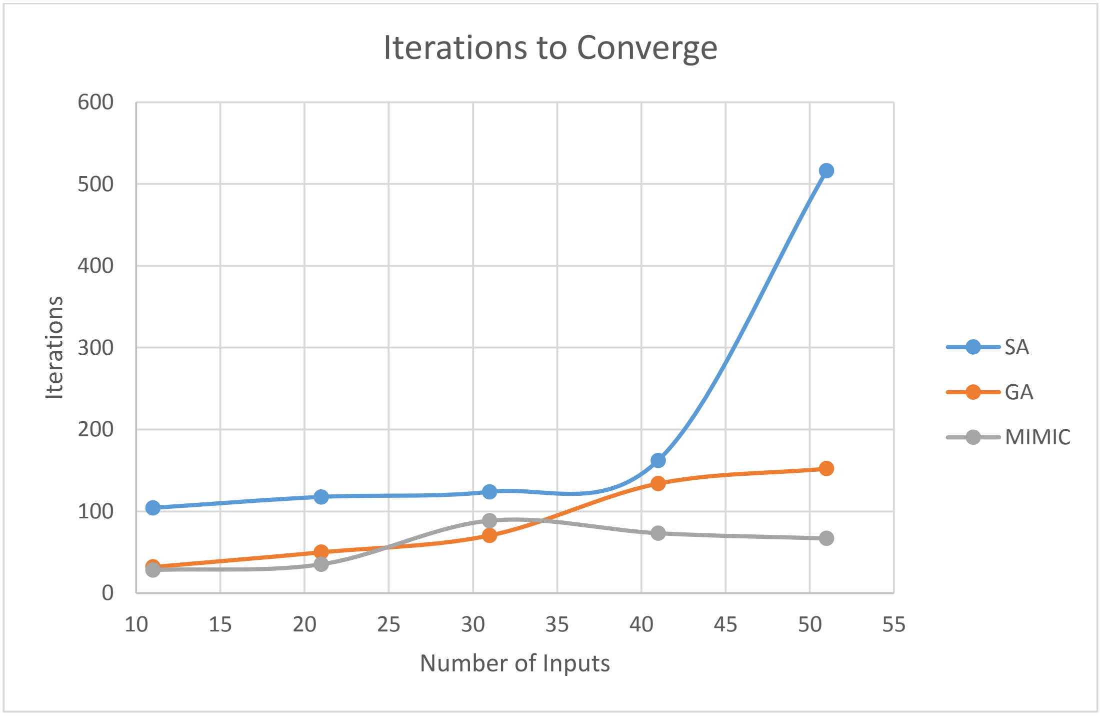

Figure 4: Iterations required for algorithms to converge with fitness function 1

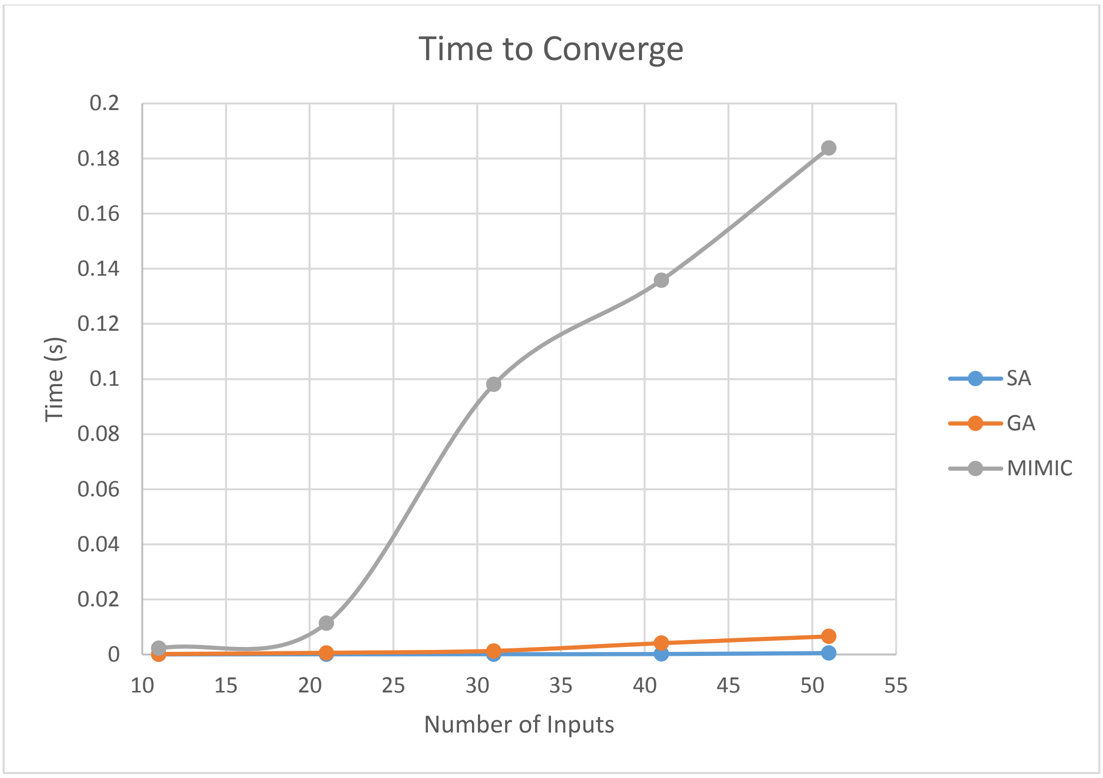

Figure 5: Time required for algorithms to converge with fitness function 1

b. Fitness Function 2

The second fitness function seeks to find the input string with the highest number of consecutive digits that are the same. For example, the input 11101 would have a fitness of 3 while an input of 00000 would have a fitness of 5. As such, the fitness function as 2 global maxima: one where the input is all zeros and one where the input is all ones.

Since the fitness does not depend on structure as the first fitness function did, it was not expected that MIMIC would have any definitive advantage. Figure 6 and Figure 7 reinforce this thinking. MIMIC consistently takes the fewest number of iterations, but as the input size grows, the wall clock time to find the maximum grows rapidly. Meanwhile, the execution time for simulated annealing and the genetic algorithm barely grows.

Once again, the “best” randomized optimization algorithm depends on which criteria are most important. Simulated always takes the least amount of time, but the most iterations. MIMIC typically requires the fewest iterations but the most time. The genetic algorithm seems to strike the best middle ground between number of iterations and time. It takes approximately the same number of iterations to converge as MIMIC while taking approximately the same time as simulated annealing. As such, it combines the best features of both algorithms. However, it can be seen that the performance of the genetic algorithm begins to diverge as the input size increases, so it is unknown if these conclusions hold for larger input spaces.

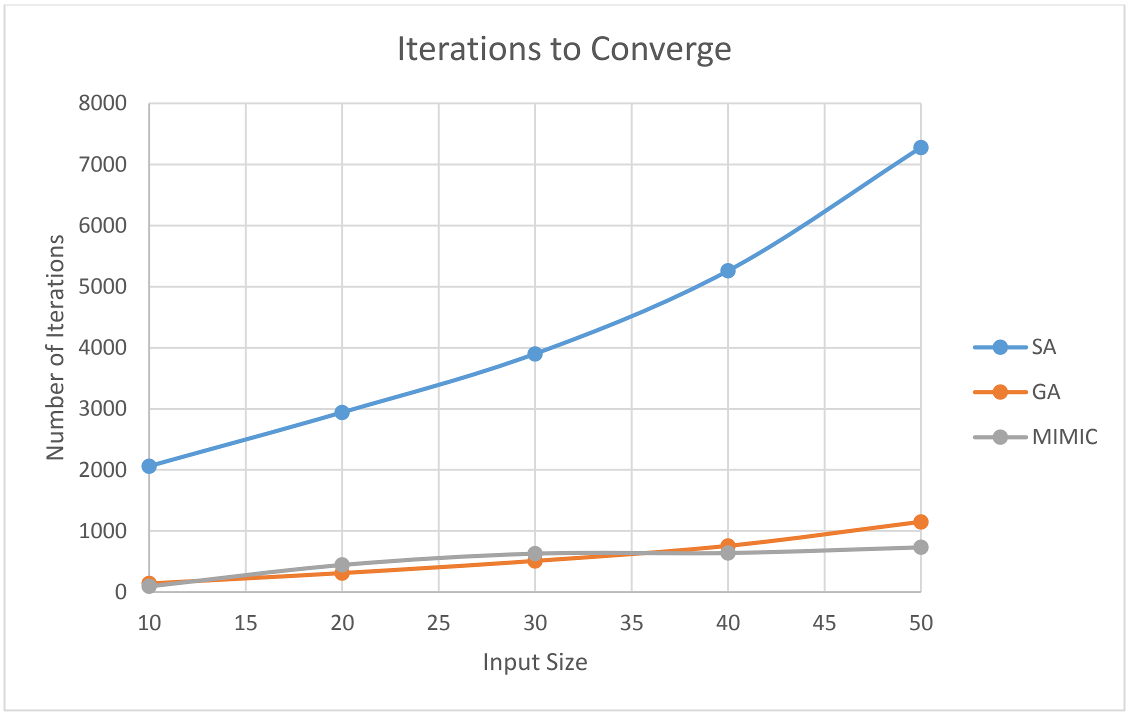

Figure 6: Iterations for algorithms to converge with fitness function 2

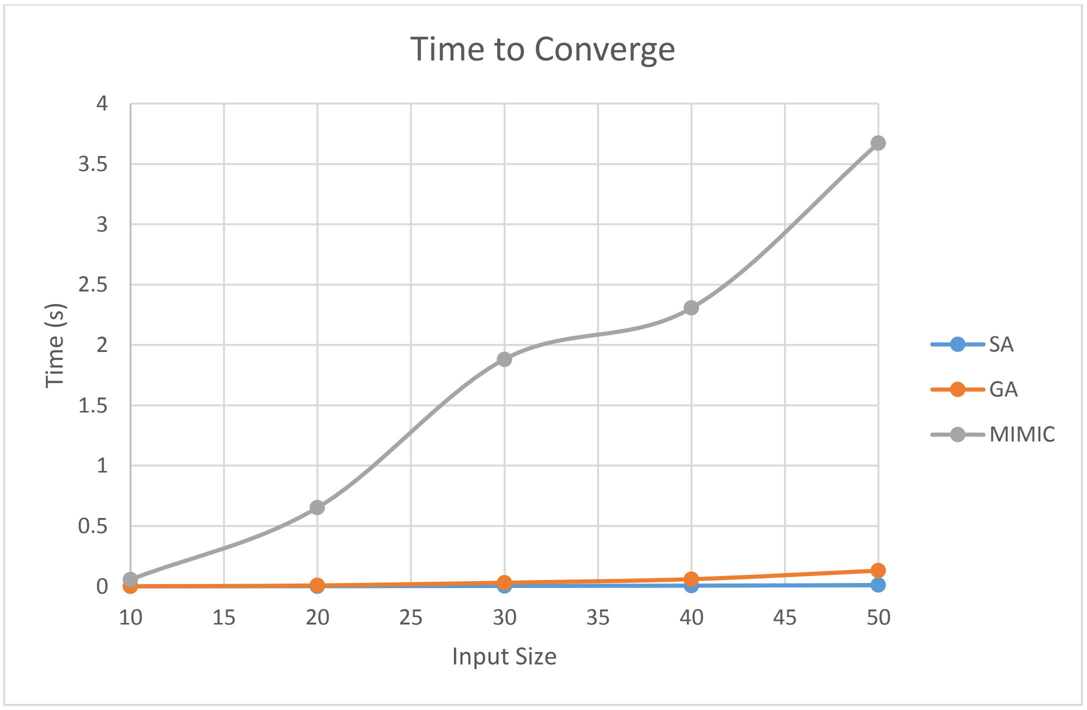

Figure 7: Time required for algorithms to converge with fitness function 2

c. Fitness Function 3

The third and final fitness function that was tested seeks to maximize the number of ones in the input string while also being concerned about the first and last inputs. The output value of this fitness function is the number of ones present in the input strings plus the length of the input string if the first and last digits are the same. It is easy to determine that the fitness function has a single global maximum at the point where the entire string is all ones. At this point, the fitness value is two times the length of the input string. Nevertheless, there are many local maxima that the optimization algorithms can get trapped in as detailed below.

Figure 8 and Figure 9 present the performance results of the algorithms with this fitness function. They clearly show that the genetic algorithm and simulated annealing are the worst performing algorithms of the group. The poor performance is due to how the algorithms were graded when they did not converge. For the other fitness functions, the algorithms never consistently reached the bound on the number of iterations, so the bounds did not affect the results. However, for this fitness function, both simulated annealing and genetic algorithm frequently reached the limit having failed to converge. It was decided that in order to graphically show that these algorithms did not converge, the full time and iteration limit was used in the construction of these figures. Different metrics could have been used, such as running until the iteration limit and then determining when the algorithm first reached the local minimum from which it never escaped. However, a metric such as this would have skewed the results even further because the assumption would have been made that when the algorithm plateaus, then that value was the maximum which is simply not the case.

Examining the individual runs that were averaged together to construct the data points shows that for the genetic algorithm and for simulated annealing either the algorithm converged relatively quickly or it did not converge at all but did settle on a value, albeit not the maximum, quickly. Thus, it appears that the algorithms were able to find local maxima very quickly, but had difficulty searching the input space to find the single global maximum.

The MIMIC algorithm outperforms all the other algorithms on this fitness function. In fact, it is the only algorithm that converged on the maximum value every time. MIMIC required several orders of magnitude fewer iterations than the other algorithms to converge on the maximum value. The time required for MIMIC to converge was comparable to simulated annealing and grew at a much slower rate than the genetic algorithm while having the advantage of actually converging on the maximum value. This agrees with the results in [1]. Since this fitness function is similar to that used in the 4 and 6 peaks problem, it is reasonable that MIMIC would outperform the other algorithms here as it did in [1].

The reason MIMIC outperforms the other algorithms is because it is able to identify the relationship between the first and last inputs. Simulated annealing does not model any relationships between the various inputs. The genetic algorithm is able to model some relationships between inputs via crossover, but it seems the choice of crossover for this genetic algorithm is not able to model the relationship between the first and last input consistently.

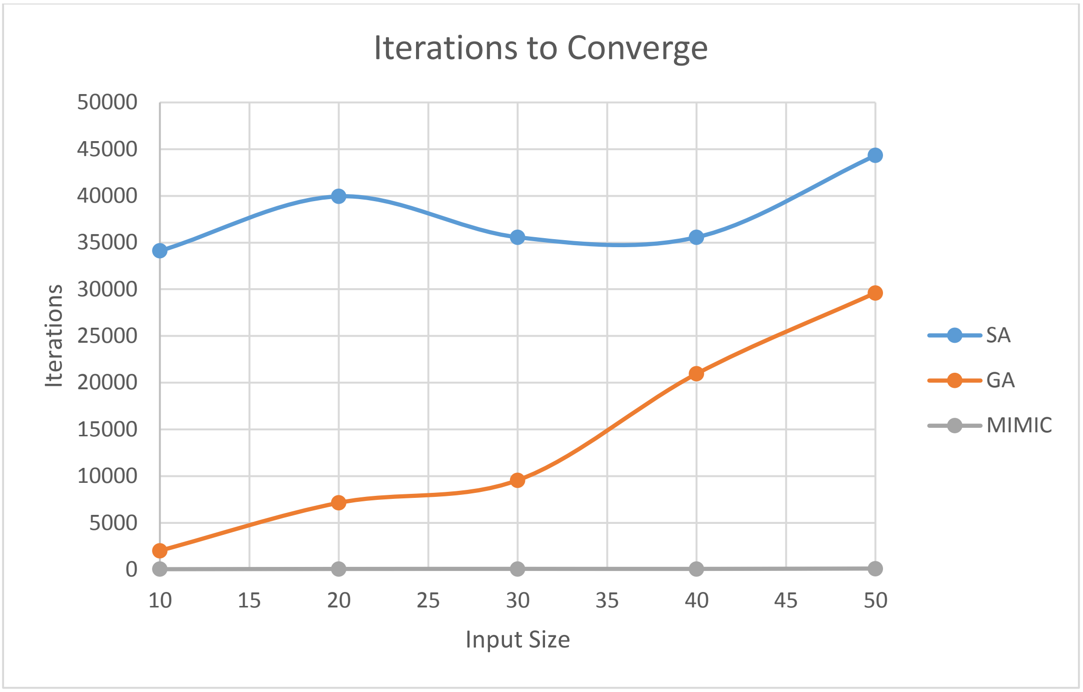

Figure 8: Iterations for algorithms to converge with fitness function 3

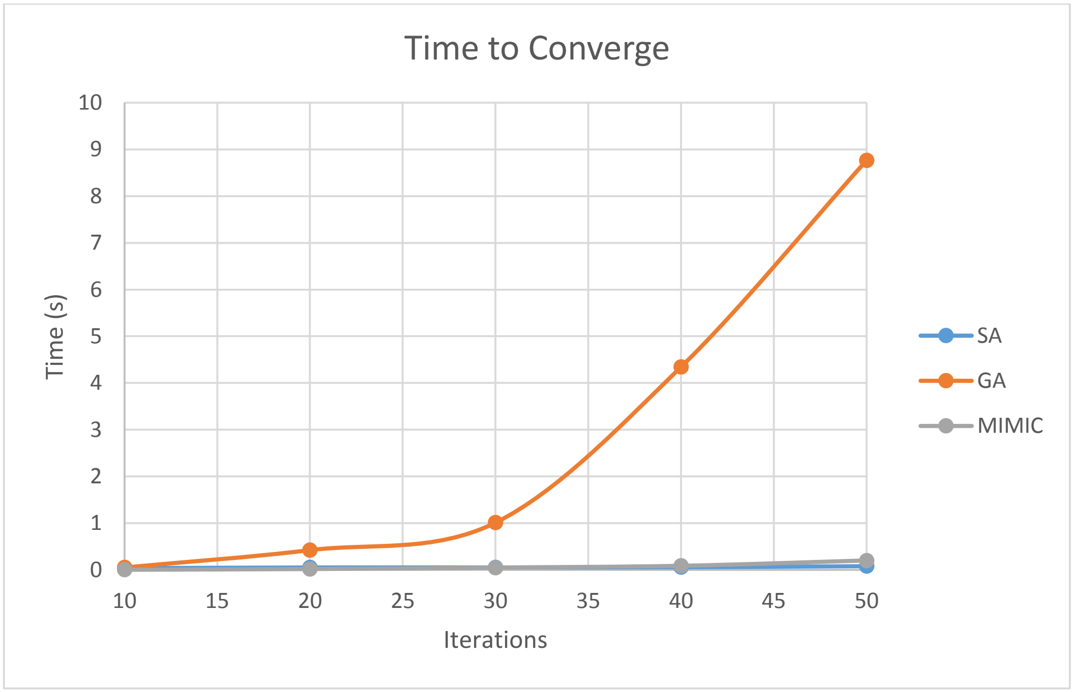

Figure 9: Time for algorithms to converge with fitness function 3

4. Conclusion and Further Work

With the data set from Assignment 1, randomized optimization neural network training is comparable and in some aspects superior to backpropagation training. When averaged over all the runs, training error using randomized optimization was less than backpropagation. However, randomized optimization took longer to train and has the potential to suffer a consistency problem.

The choice of fitness function can greatly affect the optimization algorithm performance. It was shown that for some fitness functions, either the algorithm converged right away, or became trapped in a local minimum and never converged. This appears to be part of the nature of randomized optimization; consistently locating a global maximum is not always to be expected. However, it was also shown that some algorithms did not get trapped in these same local maxima.

It was further concluded that there is generally no single best randomized optimization algorithm for a given fitness function. For instance, while MIMIC consistently beat the other algorithms in the number of iterations required to find the maximum, it was itself consistently beaten the other algorithms in the amount of time required to converge. There appears to be a fundamental tradeoff between the number of iterations and the time required to converge.

Further work involves studying how changing the parameters of the various optimization algorithms affects performance. Some tweaking was done during the tests, such as for the third fitness function, in order to get the genetic algorithm to converge, but nothing was done in a systematic way. These values can drastically affect the performance. For instance, with the genetic algorithm, increasing the population size may lead to a more diverse population, but will also increase the amount of time and iterations required to converge.

5. References

[1] C. L. I. Jr., "Randomized Local Search as Successive Estimation of Probability".