Randomized Optimization

Overview

Four randomized optimization algorithms were implemented and analyzed for this report. These algorithms are Randomized Hill Climbing, Simulated Annealing, Genetic Algorithms, and Mutual Information Maximizing Input Clustering (MIMIC).

In the first part, these algorithms are implemented (excluding MIMIC) to determine the best weights for a neural network and compare results to backpropagation, which was implemented in Assignment 1. In the second part, all four algorithms were implemented on a set of common optimization problems including the Continuous Peaks Problem, Flip Flop Problem, and Travelling Salesman Problem. Results between different algorithms were compared to determine the best performer for each problem.

Be aware that in this report, randomized optimization algorithms may be referred to by their acronyms for brevity (RHC = Randomized Hill Climbing, SA = Simulated Annealing, GA = Genetic Algorithm, MIMIC = Mutual Information Maximizing Input Clustering)

Neural Network

Randomized optimization algorithms were used to determine optimal weights for a neural network. The dataset that is used to train the neural networks is the Segmentation dataset used in Assignment 1. Each instance in this dataset represents a 3x3 pixel segment of a randomly selected outdoor picture. 19 continuous valued features contain information about the visual characteristics of the segment. There are 7 classes corresponding to what is depicted in the segment, and the dataset is balanced.

Methodology

Four algorithms for determining neural network weights are examined in this section: backpropagation, randomized hill climbing, simulated annealing, and a genetic algorithm. The backpropagation algorithm was implemented as a control to compare randomized optimization algorithms results. Each algorithm was run for 11 trials to convergence using the same convergence criteria of 5000 iterations without fitness value improvement. In our analysis, plots are created from the trial that resulted in the median test accuracy.

In the case of simulated annealing, the cooling exponent was varied between 0.15 and 0.95. For the genetic algorithm, population size was set to 50 and the number of the population to mate and mutate were varied in combinations of 10 and 20. For these algorithms, the hyperparameters that produced the best results for average test accuracy were used in this report.

For all algorithms, including backpropagation, neural networks were implemented with 3 hidden layers with 38 nodes each which was found in Assignment 1 to be the optimum number of layers and nodes for this classification problem. In further analysis, optimization algorithms were further implemented on a 10 node per hidden layer network to compare results.

Results

Table 1 shows the performance of the 3 randomized optimization algorithms in relation to backpropagation. The results of backpropagation are very similar to accuracies achieved in Assignment 1. We can see right away that the results are relatively poor for all 3 algorithms. We see lower training and test accuracy and much higher number of iterations to reach convergence. In fact, as can be seen in Figure 1, backpropagation converges in only a few iterations.

Table 1: Performance of Backpropagation and Randomized Optimization Algorithms (values are averages taken over 11 trials)

Training Algorithm	Nodes per Hidden Layer	Training Accuracy (%)	Test Accuracy (%)	Iterations	Time (seconds)
Backpropagation	38	97.38	90.36	5001	247
RHC		49.32	47.89	27535	397
SA		56.44	54.89	38355	482
GA		56.41	56.00	18058	5058
RHC	10	89.24	85.99	91221	224
SA		93.41	89.90	100000	242
GA		50.24	49.89	16022	732

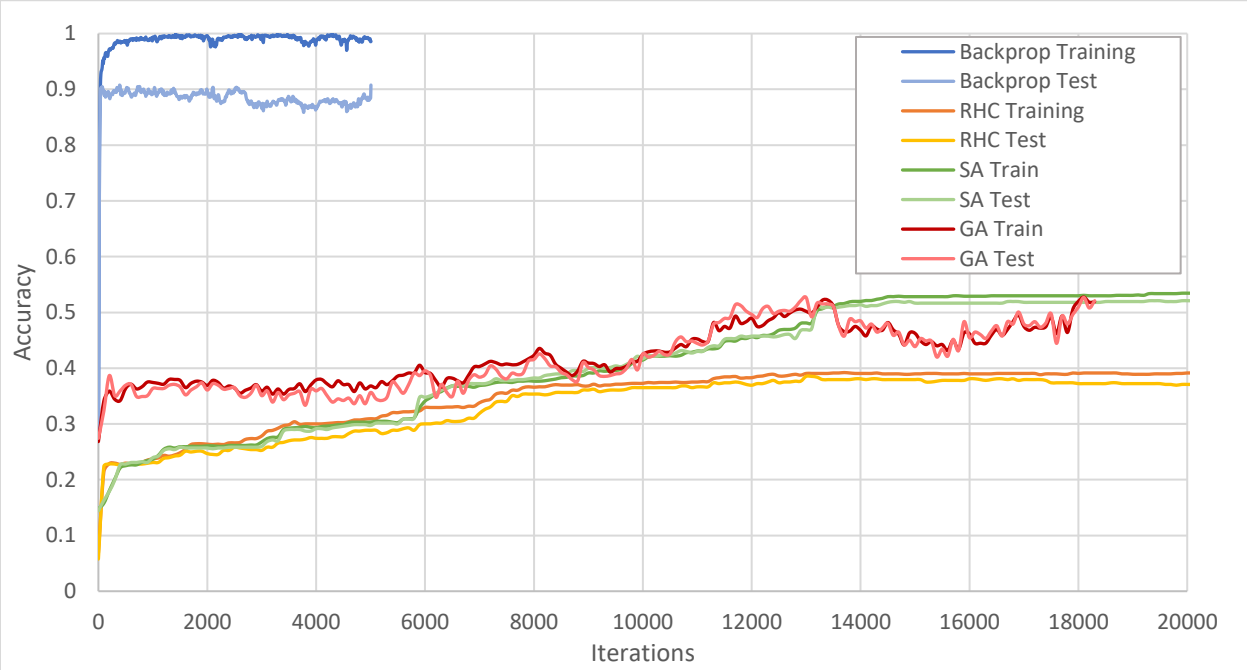

Figure 1: Neural networks trained by randomized optimization algorithms (38 nodes per hidden layer) - Accuracy vs Iterations. Backpropagation neural network accuracy included for comparison (38 nodes per hidden layer)

In Figure 1, the trial that achieved the median test accuracy out of the 11 trials for each algorithms was plotted to help us understand the typical performance. Compared to backpropagation, we see that the optimization algorithms take many iterations to converge in and when they do converge it is at a lower

accuracy. We see that SA breaks away from RHC in higher iterations which suggests that RHC gets stuck in a local optimum while the SA algorithm can restart and explore for better optima.

The genetic algorithm quickly finds higher fitness values in lower iterations but fluctuates in higher iterations. This may be due to the large problem space where finding a relatively small population that allows us to improve fitness consistently is unlikely. Also, despite the low iterations to convergence for GA, there is more work per iteration in comparison to the hill climbing algorithms. Every iteration has the GA evaluate and rank the fitness of its population and perform crossover and mutation. This leads to a much longer runtime than the other algorithms tested, as seen in Table 1.

A reason for the relatively poor performance of all randomized optimization algorithms in comparison to backpropagation is the fact that we are working with network weights which are continuous values rather than discrete. This makes the problem space infinitely large and there may be narrow basins of attraction that random searching algorithms are extremely unlikely to find.

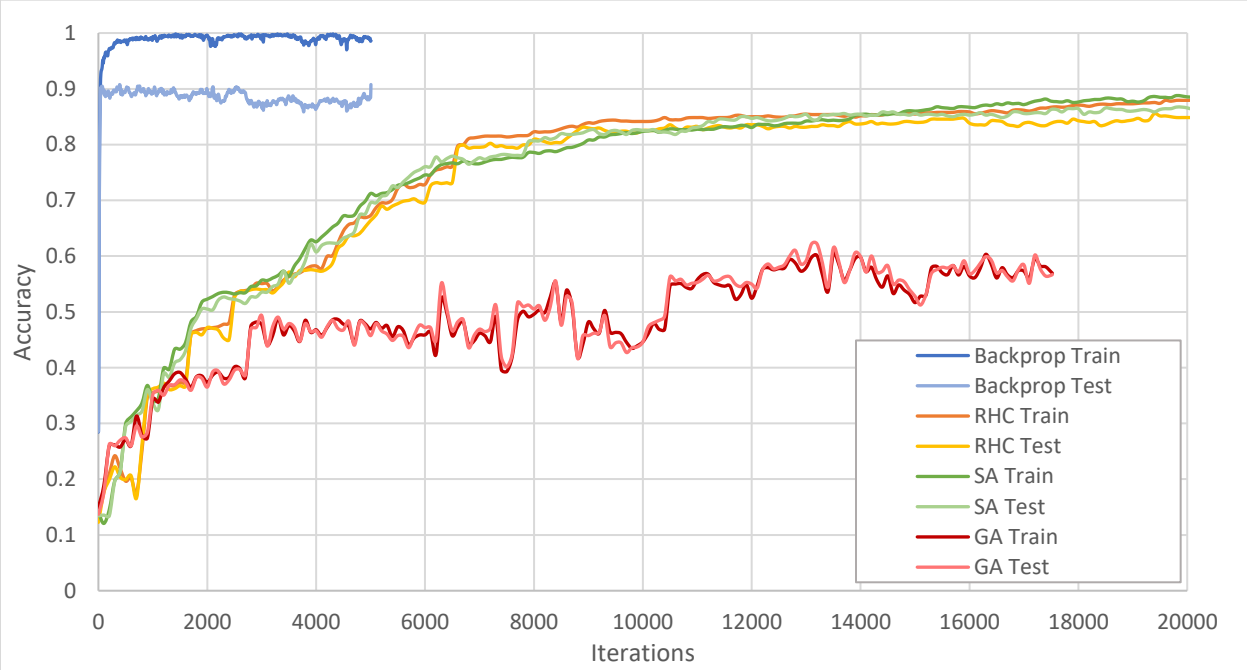

Figure 2: Neural networks trained by randomized optimization algorithms (10 nodes per hidden layer) - Accuracy vs Iterations. Backpropagation neural network accuracy included for comparison (38 nodes per hidden layer)

To further investigate this theory, optimization algorithms were implemented on neural networks with 10 nodes per hidden layer to reduce the number of network weights to train and thus to reduce the hypothesis space. Figure 2 shows the accuracy of these networks vs iterations. Notice that the accuracy reached by the RHC and SA algorithms is much higher. The genetic algorithm however saw minimal change in fitness, but much better training time, as seen in Table 1.

We can see that in smaller problem sizes some optimization algorithms become competitive with backpropagation both in accuracy and runtime. Simulated Annealing in particular achieved accuracy and runtime very close to backpropagation. With further tweaking of problem complexity and iterations we may be able to reach results that beat backpropagation for this particular problem.

One more interesting point is that all optimization algorithms exhibit very low overfitting in comparison to backpropagation. This is apparent from the margins between the training and test accuracy for each algorithm in the proceeding plots. A possible explanation for this is that backpropagation is gradient based and seeks iterative improvements based on previous weights. These weights are based on training data, so we can say that backpropagation relies and adheres closely to training data, sometimes too closely which can result in overfitting. Conversely, the stochastic nature of the optimization algorithms implemented means that they do not retain information on training data from iteration to iteration and are thus able to avoid fitting the training data too closely.

Randomized Optimization Problems

For the second part of this report three types of optimization problems were examined: Continuous Peaks, Flip Flop, and the Traveling Salesman Problem. For each problem, four algorithms were implemented to determine the optimum fitness value: Randomized Hill Climbing, Simulated Annealing, Genetic Algorithms, and MIMIC.

Methodology

Each algorithm was implemented to convergence over 5 trials on each of the problem domains. The same convergence criteria of 1000 iterations without seeing a significant improvement in fitness values was used across all algorithms. Results from each trial were averaged and used to create the plots included in this report.

The same SA and GA hyperparameters were explored in this section as were the previous section. The MIMIC algorithm m-value was varied between 0.1 and 0.9. This is a Bayesian smoothing parameter – a value of 0 for 'm' allows a true dependency tree to be constructed while increasing values of 'm' allow for more deviation from the true dependency tree probability distribution where points with weaker dependencies have a higher probability of being selected. As in the previous section, hyperparameters that produced the best average fitness were used in this report.

Continuous Peaks Problem

The Continuous Peaks Problem assigns higher fitness values for sequences of unbroken 0s and 1s in the bit string and assigns a bonus when the max sequence of both 0s and 1s is greater than a preassigned value of T. The global optimum is 2N - T - 1 but for higher values of T this can be a difficult optimum to find and the algorithms tend to settle in local optima. The value of T for these experiments was set to 2/5 of N, which presents a somewhat narrow basin of attraction for finding the global optima.

Results for the optimal fitness values found by each algorithm as a function of bit string length can be seen in Figure 3 We can see that, in general, the optimum fitness value (as a percentage of the global optimum value) found by each algorithm decreases as the problem size increases. This is a characteristic that holds true for all problems examined here. Much like our findings in the Neural Network section of this report, we see that as the size of the problem domain increases, the likelihood to find the global optima decreases because the algorithms are required to search a larger space. In the case of bit string problems, the number of possible hypotheses is 2^N so the hypothesis space increases exponentially with increasing values for N.

It appears that MIMIC performs very well at lower bit string lengths. Here we se that for an N value of 20, MIMIC is the only algorithm to find the global optimum. This may be because the MIMIC algorithm retains information about the problem space and is able to iteratively estimate better distributions of bit strings based on this knowledge. However, as the N Value increases it proves to be one of the worst performing algorithms. At this point the problem size may be too large for MIMIC to effectively determine the feature dependencies that yield the best estimated probability distribution. Genetic Algorithms and Simulated Annealing perform best at higher values for N. GA be may benefitting from crossover to allow better fitness values. We can also see that SA breaks away from RHC and improves fitness in larger problem sizes suggesting that it is avoiding getting stuck in local optima, which is a pitfall of RHC. but in general, these problem domains prove to be difficult for all algorithms as exhibited by the low fitness values relative to optimum. This is most likely due to the T value creating a relatively small basin of attraction for the global optima.

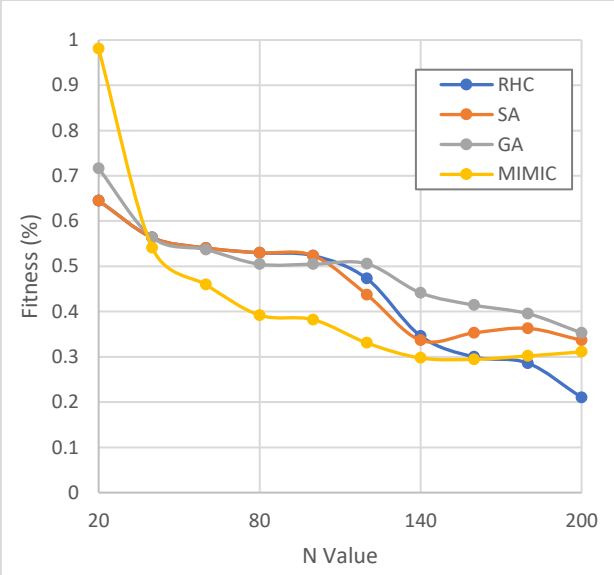

Figure 3: Continuous Peaks Problem – Fitness (as a percentage of optimum value) vs bit string length

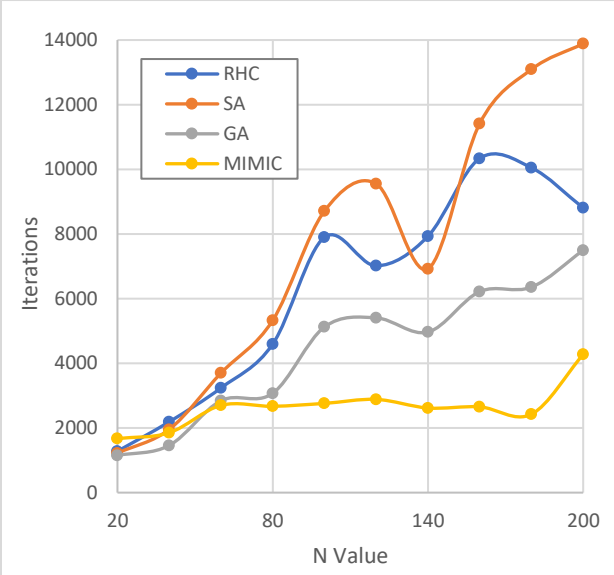

Figure 4: Continuous Peaks Problem – Iterations to reach convergence vs bit string length

In Figure 4 we can see that the number of iterations to reach convergence vary greatly between algorithms. As should be expected Simulated Annealing and Randomized Hill Climbing required the most iterations as these algorithms both use hill climbing to incrementally converge to optima. We can also see in Figure 6 that these algorithms use the least amount of function evaluations because these hill climbing algorithms perform only one evaluation per iteration. Conversely iterations are lower in the Genetic Algorithm and much lower in MIMIC. These algorithms perform more work in each iteration which is shown in Figure 6 where we can see that even with lower iterations the total number of function evaluations performed are 1 to 2 orders of magnitude greater that with RHC and SA.

In Figure 5 we can see that the total time to reach convergence for MIMIC is massive in comparison to the other algorithms. This is due to the greater amount of function evaluations required and the internal construction of dependency trees and population distribution estimation for each iteration. Although the Genetic Algorithm has roughly the same amount of fitness evaluations, the crossover and mutation functions are much less computationally intensive than then inner workings of MIMIC just described. The Simulated Annealing and Randomized Hill Climbing algorithms were very quick in comparison. These

algorithms are computationally simpler, evaluating only one neighbor per iteration and updating the current hypothesis if the fitness function evaluation at the neighbor is greater. Simulated Annealing typically ran longer than RHC due to the explorative nature of SA and the possibility of resetting the hypothesis to a new point in the problem domain.

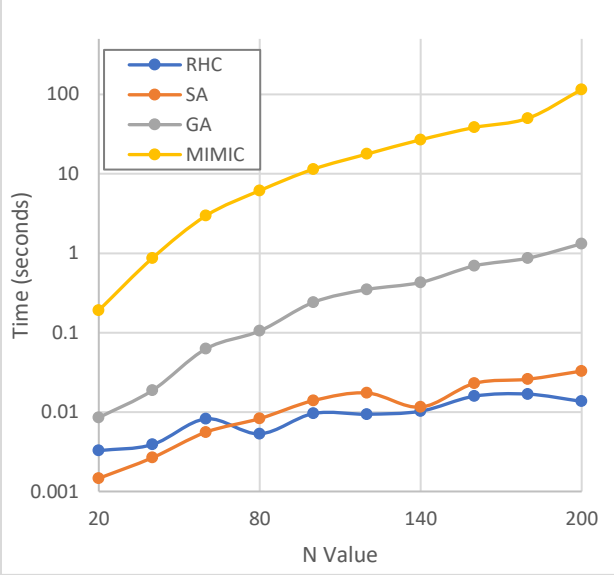

Figure 5: Continuous Peaks Problem – Time to reach convergence vs bit string length (logarithmic scale)

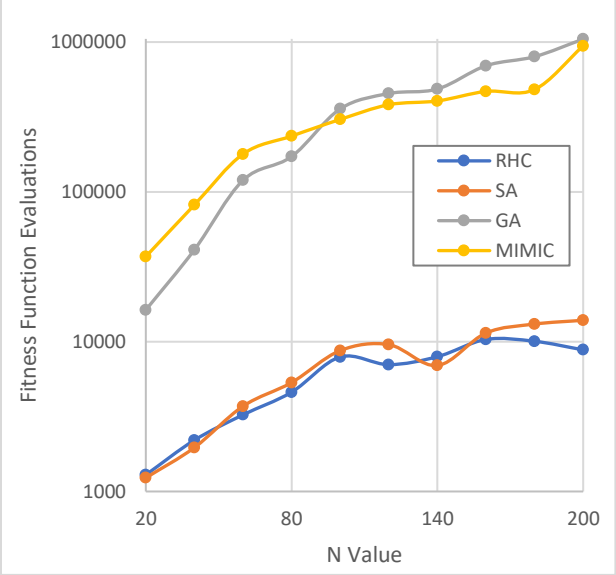

Figure 6: Continuous Peaks Problem – Fitness function evaluations to reach convergence vs bit string length (logarithmic scale)

The best performing algorithm here is debatable. All algorithms performed relatively closely with regards to optimum fitness found. An impressive point of note is that only MIMIC was able to find a global optimum even if it was on the smallest problem size. Given more time for playing with hyperparameters and T value we may find that MIMIC can perform better than other algorithms consistently but there will always be a runtime trade-off. GA performs best with larger values of N and given more time for tweaking of hyperparameters, this too may be found to perform even better but again we see long runtimes in larger problem domains. Simulated Annealing appears to be the most well-rounded algorithm for this problem. It performs nearly as well as GA in finding the best fitness and has a quick runtime.

Flip Flop Problem

The Flip Flop Problem is the simplest problem explored in this report. Here the evaluation function assigns a higher value with bit strings that alternate between 0 and 1. The longest uninterrupted sequence of 0s and 1s sets the fitness value. The global optima for this problem is N-1.

From Figure 7 we can see that all algorithms performed relatively well. Simulated Annealing performed the best, especially in higher values for N. Counterintuitively, RHC did relatively poorly compared to SA. From this we can conclude that the Flip Flop Problem contains a large number of local optima of various peak fitness values. In this situation, RHC will get stuck in in whichever basin it randomly starts at, resulting often in poor fitness values. SA, on the other hand, is able to jump to and "explore" other randomly selected basins in lower iterations. This highlights the exploration vs exploration trade-off between SA and RHC. For the Flip Flop Problem, the ability for the algorithm to explore leads to better fitness values.

Both GA and MIMIC performed well but saw slightly diminished fitness values as problem size increased. This may be due to the fact that with a simple problem like Flip Flop Problem there is little structure and not much gained in the more intensive computations of probability distribution generation and with an increasing problem size these processes struggle to find the best fitness values and performance is hindered.

Figure 8 shows that Simulated Annealing had by far the most iterations. Meanwhile RHC iterations are lower by about half in all problem sizes. This further suggests that RHC is quickly getting stuck in local optima, explaining the lower fitness values observed. Simulated Annealing shines in this type of problem with its tendency to explore the problem space in early iterations, allowing it to avoid local minima more effectively than RHC. GA and especially MIMIC exhibit low iterations needed to converge which is typical for these algorithms, the reason for which was discussed in the Continuous Peaks problem section.

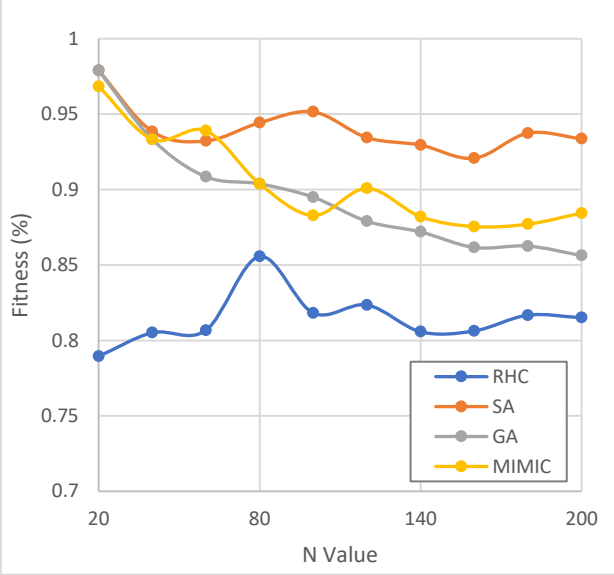

Figure 7: Flip Flop Problem – Fitness (as a percentage of optimum value) vs bit string length

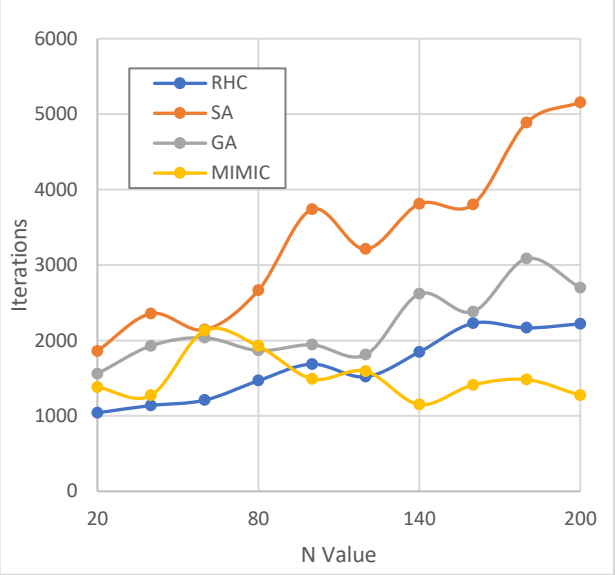

Figure 8: Flip Flop Problem – Iterations to reach convergence vs bit string length

As in the other problems examined here, the time to convergence for Simulated Annealing and RHC are much lower than that of GA and MIMIC. We see in Figure 9 that in higher iterations SA breaks away from RHC in time to convergence which can be explained by the higher amount of iterations and greater number of function evaluations seen in Figure 10. GA takes about an order of magnitude more time to converge and MIMIC is even slower by at least another order of magnitude. The sluggish performance of these algorithms is explained by the number of function evaluations performed as seen in Figure 10 and beyond this MIMIC takes more time to construct a dependency tree for every iteration.

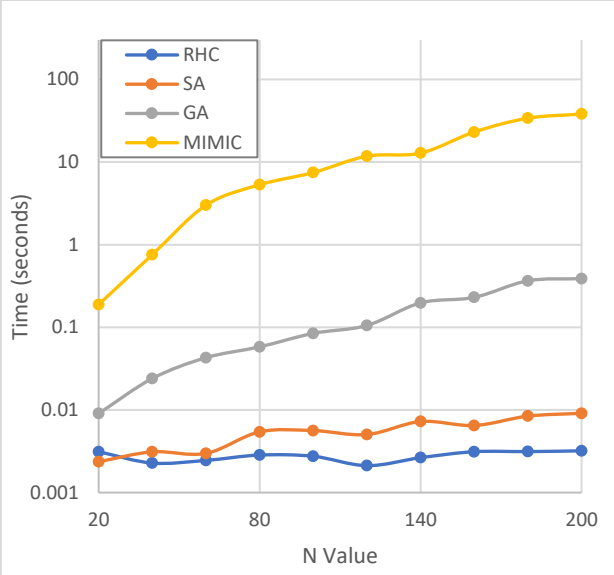

Figure 9: Flip Flop Problem – Time to reach convergence vs bit string length (logarithmic scale)

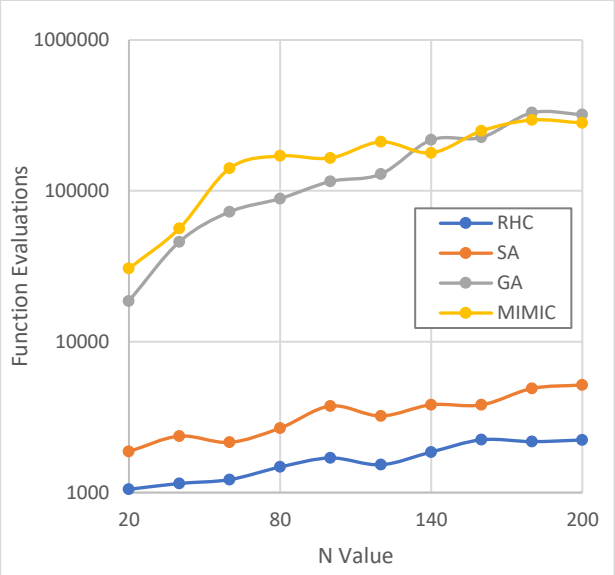

Figure 10: Flip Flop Problem – Fitness function evaluations to reach convergence vs bit string length (logarithmic scale)

The best performing algorithm here is clearly Simulated Annealing with little to no tradeoffs. This algorithm achieves a higher fitness value than other algorithms, especially with larger problem sizes. The algorithm performs very well regarding runtime as well, which is to be expected of an algorithm with a constant runtime with increasing problem size.

Traveling Salesman Problem

The Traveling Salesman Problem is a problem in which the goal is to connect a set points, or 'cities', with the shortest path possible and return to the origin city without visiting any other point more than once. For any set of points N there are N! possible routes, making this a problem domain that quickly becomes very large with increasing N. For this report, the number of points were varied between 10 and 100 and the (x, y) location of these points were randomly generated after which all algorithms were ran on the same set of points.

We can see in Figure 11 that Genetic Algorithms achieve very good fitness results, especially with higher numbers of points. Crossover plays a big role in the success of this algorithm and the ABAGAIL library contains a genetic algorithm crossover function specifically designed for the Traveling Salesman Problem. A child path is created by selecting a random starting point and the next point in the path is selected by seeing what point the parents connected to and picking the one with the shortest distance. In addition, the mutation function ensures that other paths are explored that might not be present in the current population. For these reasons, GA is able to iteratively find better and better fitness values for the Travelling Salesman Problem.

Mimic does comparatively poorly regarding fitness values and appears to be getting stuck in local optima. Simulated Annealing and RHC perform similarly but generally worse than GA except in cases of low number of points. All algorithms perform similarly well with lower point numbers. In this case the problem

domain may be small enough for the algorithms like SA and RHC to stumble upon high fitness values fairly easily.

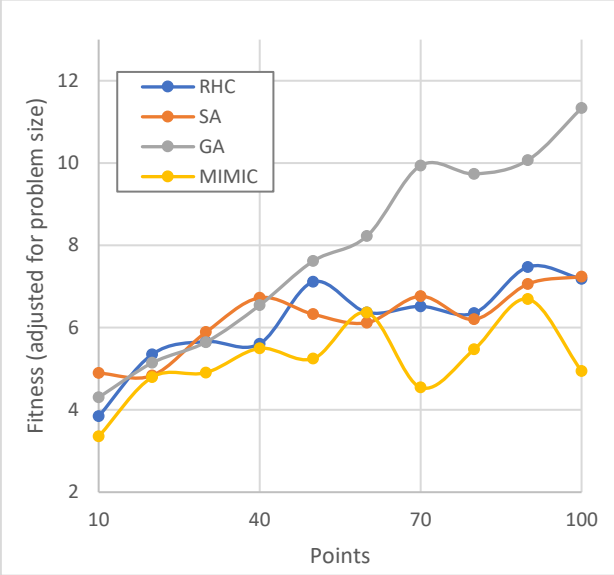

Figure 11: Traveling Salesman Problem – Fitness (scaled by number of points) vs number of points

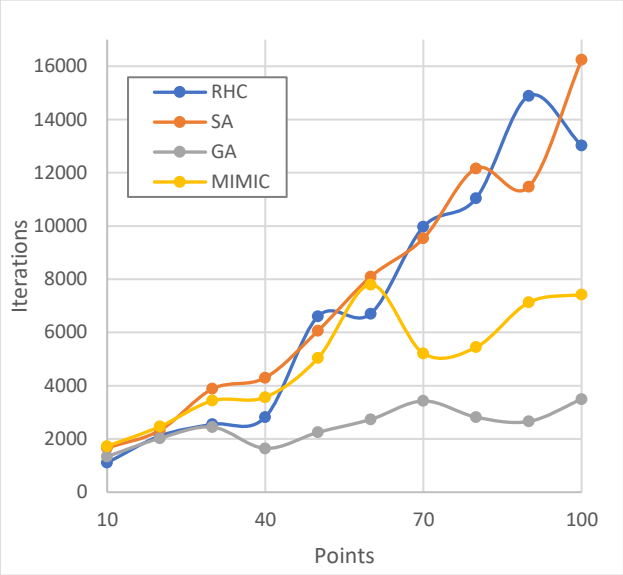

Figure 12: Traveling Salesman Problem – Iterations to reach convergence vs number of points

Figure 12 shows us the iterations it took each algorithm to converge based on the number of points in the problem domain. As in previous problems we see that the number of iterations to converge for SA and RHC are greatest in larger problem domains. For each iteration in these hill climbing algorithms, a random network of points is generated and evaluated so again you have one fitness function evaluation per iteration as shown in Figure 14. Unlike other problems in this report, we see that the Genetic Algorithm takes the lowest number of iterations to converge among almost all problem sizes. This suggests that the algorithm is effective enough, through crossover and mutation, to reach an optimum fitness value quickly. Conversely MIMIC sees iterations that are a bit higher than typical and we can see why with the large amount of function evaluations exhibited by this algorithm in Figure 14.

As can be expected given the number of iterations, we see in Figure 13 that MIMIC performs very slowly in comparison to other algorithms, taking more than 3 orders of magnitude more time in larger problem sizes. The Genetic Algorithm, on the other hand performs well regarding time due to the low number of iterations. Again, SA an RHC are the fastest to converge but the trade off is that they get stuck in local optima in larger problem sizes.

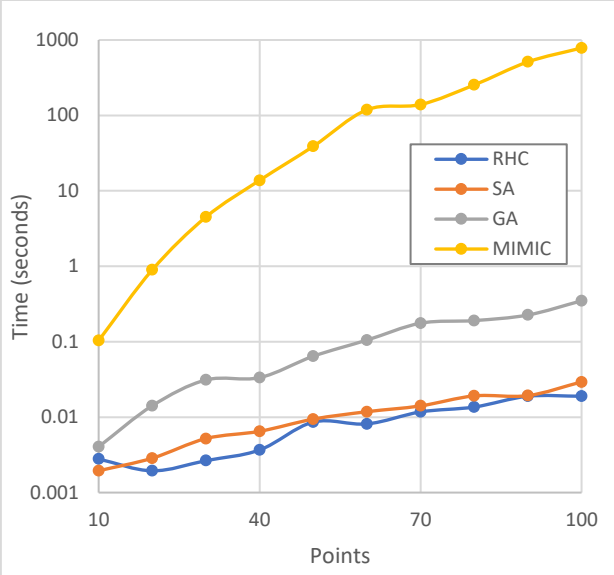

Figure 13: Traveling Salesman Problem – Time to reach convergence vs number of points (logarithmic scale)

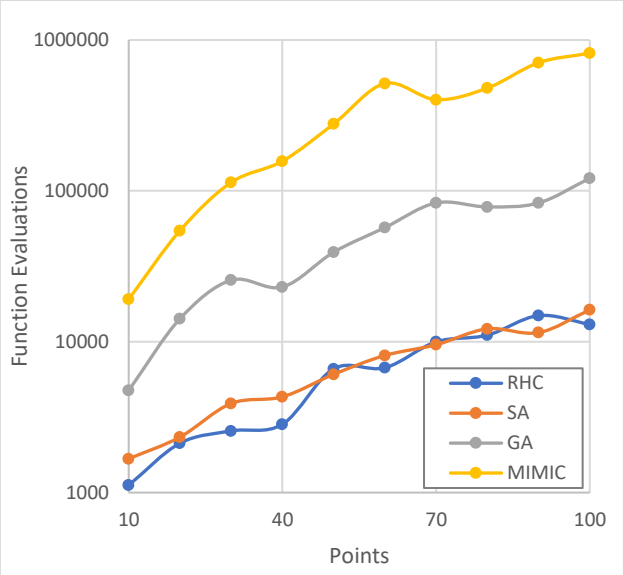

Figure 14: Traveling Salesman Problem – Fitness function evaluations to reach convergence vs number of points (logarithmic scale)

Genetic Algorithms can be considered the best algorithm for the Traveling Salesman Problem if we are willing to accept a runtime that is about an order of magnitude longer than the quick hill climbing algorithms. The optimum fitness value found by GA is significantly higher in larger problem sizes. SA and RHC could be considered in cases where time is important. MIMIC performed all around poorly and should be avoided here based on the results of this experiment.

Conclusion

We have seen in this report that certain randomized optimization algorithms are better suited for certain problems. In the case of neural networks, we see that optimization algorithms do not perform well in large problem domains with continuous values and backpropagation presents a more efficient algorithm for finding network weights. In certain cases, however, we may be able to reduce the problem domain size by using fewer network nodes. In the experiments here, a high accuracy was achieved with the hill climbing algorithms, comparable to backpropagation with the added bonus of less overfitting.

In the second part of this report we found that certain problems are able to be successfully navigated by algorithms suited for them. In the case of Continuous Peaks Problem, we found that MIMIC performed well at low problem sizes because of its accounting of problem structure. Genetic algorithms performed well in larger problem domains here for similar reasons. The Flip Flop Problem was dealt with handily by the Simulated Annealing algorithm due to its simplicity and its tendency to avoid local optima by way of exploration. Finally, the Traveling Salesman Problem was solved best by genetic algorithms thanks to an effective crossover and mutation function.

Each algorithm clearly has its strengths and weaknesses and it helpful for the individual implementing them to have a firm grasp on which problems they are better suited for.