Abhay Dalmia

11th March 2018

CS 4641 - Project 2 (Randomized Optimization)

Part 1: Training neural net with randomized optimization

For this part, the **spambase** dataset from project 1 was used.

This dataset uses 57 attributes to predict whether an email is considered "spam" or not. With the hundreds of important emails that we get every week, it is important for email services to sort out the spam. This dataset counts the frequency of word considered to appear in abnormal frequency in spam emails (like "credit" and "money") along with other important features, like the frequency of special characters and the frequency of capital words, to decide if the email is considered spam.

The whole spambase dataset contains 4601 instances that was split into training and testing sets. An 80/20 split was used with the training set having 3681 instances and the test set having 920 instances.

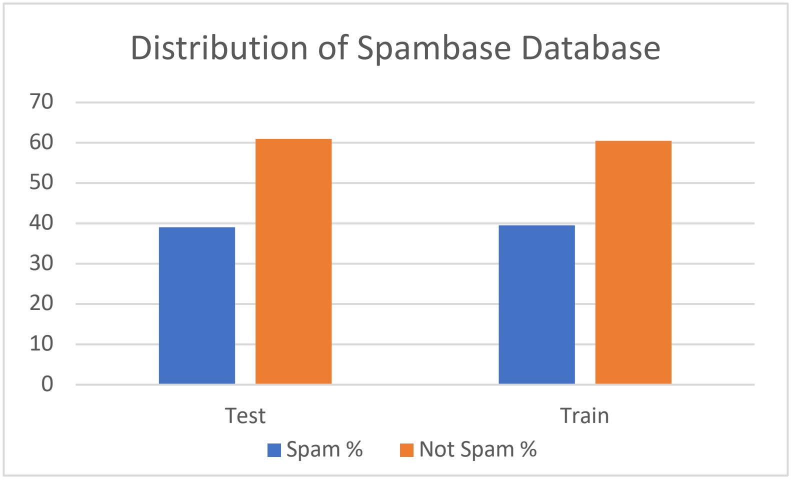

Figure 1. Distribution of labels for training and testing data set

The distribution of this dataset is pretty even. About 40% of all instances are spam instances, and 60% are not spam. This, in part, speaks to the fairness of the testing and training datasets. This dataset has a large number of attributes, compared to the number of instances. This might mean that not all attributes will be represented well by the instances and learning models might choose to ignore some attributes completely, because their importance in determining the final state is not apparent in the dataset.

Note: In the proceeding analysis, test and train sets are used instead of cross-validation because of time considerations.

1.1 Back-propagation

From the first assignment, for back propagation, the optimal neural net was trained with 2 hidden layers and 8 nodes per layer, learning rate of 0.3 and momentum of 0.2 which resulted in a train accuracy of 93.5449% and test accuracy of 92.5886%. However, in for the first assignment, results were derived by using cross-validation techniques. The experiment was repeated with the above train and test sets above for an even/just basis for comparison. The results are summarized below.

Training Size	Number of Layers	Learning Rate	Momentum	Training Accuracy	Test Accuracy	Training Build Time (s)	ROC Area
3681	2	0.3	0.2	94.0043	90.2133	19.82	0.915

For the randomized optimization algorithms below, the number of hidden layers used was also 2 with 8 nodes per layer. Instead of backpropagation, feed-forward was used in conjunction with the algorithms.

The ABAGAIL library was used to implement and test these three algorithms. The testing was largely derived from the AbaloneTest file provided in the ABAGAIL library.

1.2 Randomized Hill Climbing

For this algorithm, the number of iterations was varied from 500 to 10,000 and the testing and training accuracies were recorded. 3 trials were conducted for each number of iterations and accuracies were averaged out. This was done due to the "random nature" of this algorithm. With averaged trail, there is greater confidence in the result, and any random performances of the algorithm (whether lucky or unlucky) are mitigated.

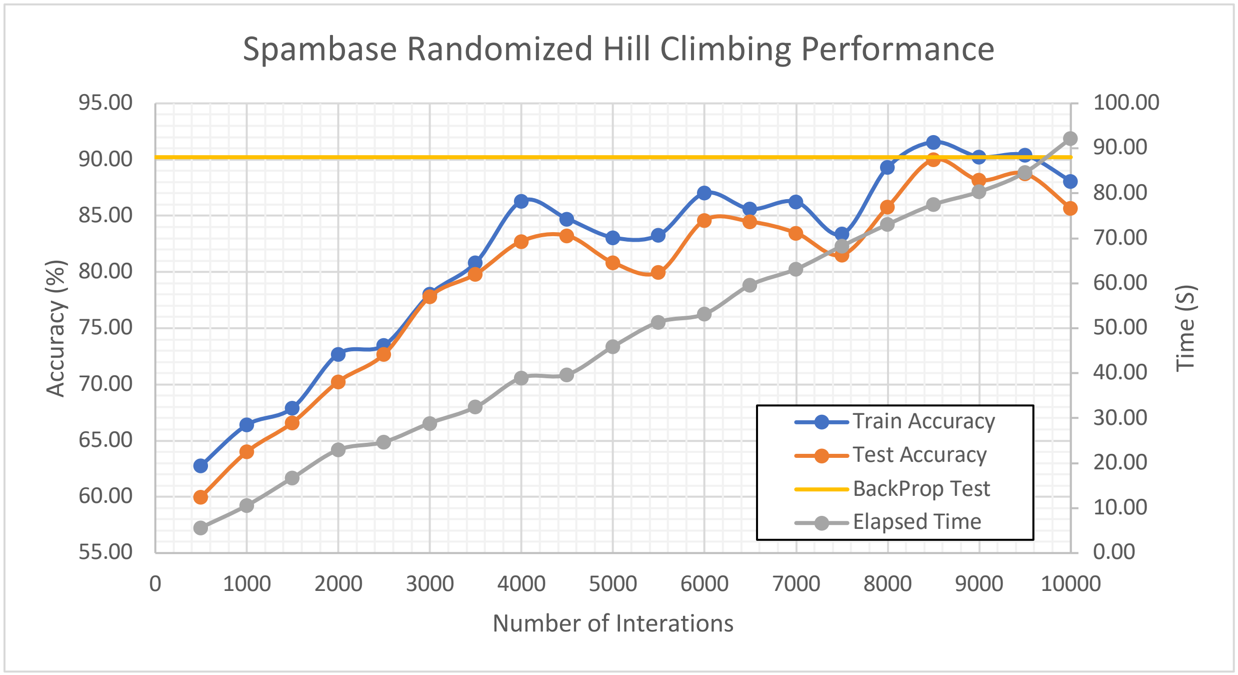

Figure 2. Performance of RHC based Neural Net on Spambase Dataset

It is clear from the graph that the performance of the algorithm trends upwards as the number of iterations is increased. This is as expected because with each iteration (and by extension random restart), the algorithm's likeliness of incorrectly getting "stuck" at a local minima/maxima decreases. Hence, the training and testing accuracies increase quite dramatically from around 60% to more than 90%.

It is also interesting to note, that for specific trials of the iterations, the trials across all iterations tend to converge to similar values of accuracy. One trial from both 2000 iterations and 9000 iterations exhibits the same accuracy. This pattern is seen amongst other trials as well. The fact that these seem to converge to similar values indicated that the spambase dataset has several local maximas of similar value that the algorithm is identifying as the global maxima. It also validates the importance of multiple trials in this case, so that the average performance of the algorithm is evaluated.

The time taken to run this algorithm was much shorter than that of genetic algorithms (although it depends on the number of iterations) and it seemed to linearly increase with the number of iterations.

1.3 Simulated Annealing

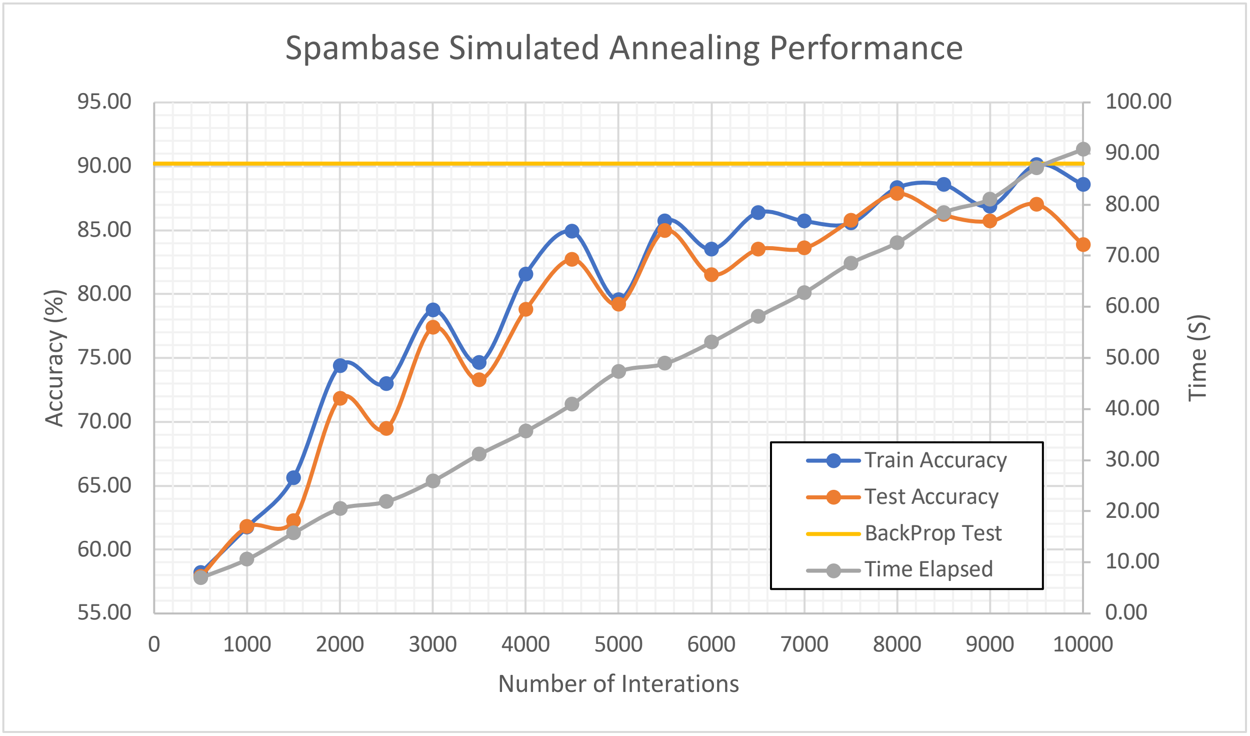

Figure 3. Performance of Simulated Annealing Neural Net on Spambase Dataset

The main hyper parameters in simulated annealing that significantly affect the algorithms runtime performance are temperature and cooling factor. Several values were tested and experimented against for a fixed number of iterations (6000 iterations in this case). From all the different permutations, it the best combination was found to be 1E9 and 0.92. The lower cooling rate could indicate the presence of many possible local maximas. However, the temperature factor is similarly low. Hence, the local maximas were unlikely to be much higher than the rest of the dataset. If the maximas were high, the temperature required would be high for increased "exploitation and exploration". Hence, the dataset is likely to have multiple low maximas. Using these parameters, the training and testing accuracies were evaluated for different number of iterations from 500 to 10,000. Similar to randomized hill climbing, for these results, the performance across 3 trails were averaged.

Similar to randomized hill climbing (RHC), the test and train accuracy steadily increased with the number of iterations. Like RHC, even for this algorithm, it follows from the fact that since there are more random restarts, the likelihood of getting stuck on a local minimum is lower (since you are exploring more) and hence, a better model is trained.

The most interesting conclusion however, is that this algorithm has similar performance to RHC. By virtue, this algorithm is a more refined version of RHC and hence should perform better. Simulated annealing explores "riskier" neighborhoods and exploits these explorations while training the model, further reducing the probability of the algorithm getting stuck at a local minimum, and increasing model performance. Since this refinement does not seem to have made a significant difference to RHC in this case, we can conclude that the dataset did not have exceedingly high local maximums. So simulated annealing did not perform much better than RHC.

The training times for both algorithms (shown in the figures as grey lines) were very similar and increased linearly with number of iterations.

1.4 Genetic Algorithms

The main hyper parameters in genetic algorithms that significantly affect the algorithms runtime performance are the population ratio, mate ratio and mutate ratio. For this algorithm I tried all permutations of the set of population ratios {0.10, 0.15, 0.20, 0.25, 0.3}, mate ratios {0.01, 0.02, 0.04} and mutation rate of {0.02, 0.04}. The best combination found was population ratio of 0.1, mate ratio of 0.04 and mutation rate of 0.04. Mutate ratio for this case seemed to make a huge difference. From this we can deduce that the search space for this dataset is complex and large. This is as expected because this dataset has a lot of attributes, very little instances, and most attributes are continuous (in the real number domain).

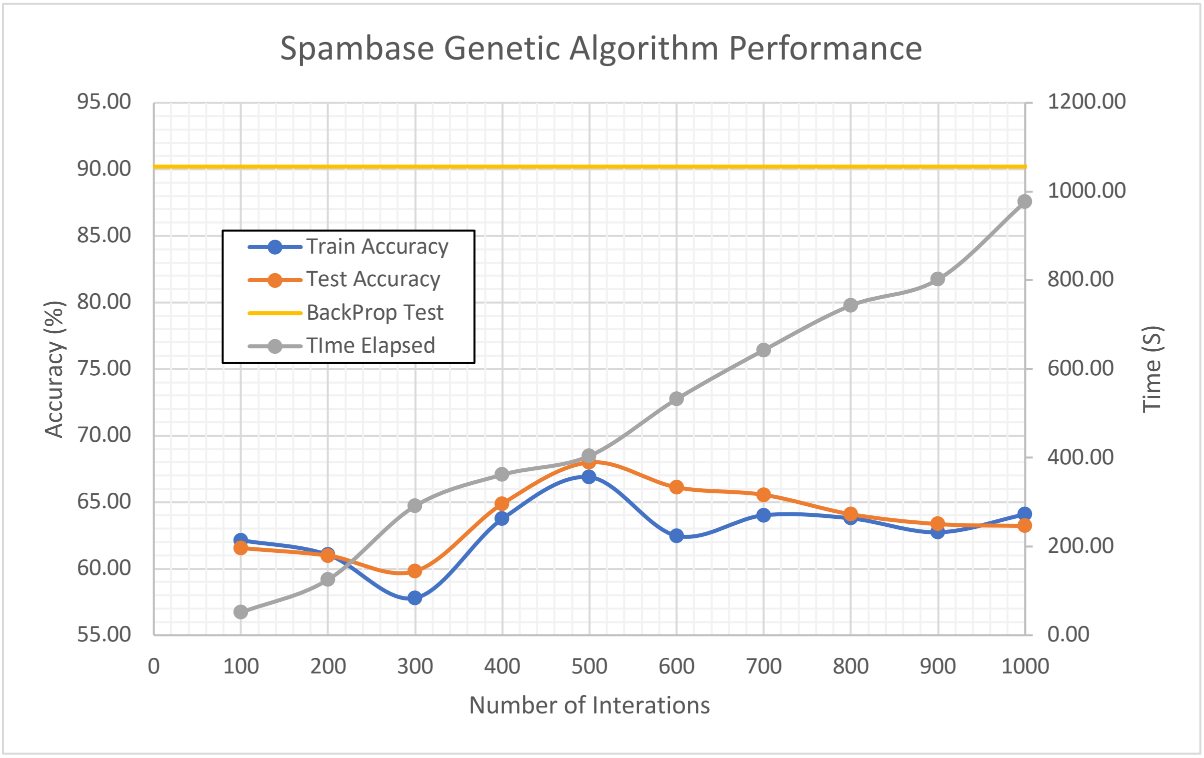

Figure 4. Performance of Genetic Algorithm (GA) based Neural Net on Spambase Dataset

The first thing we notice is the training time. Genetic algorithms take a significant amount of time to train. It is orders of magnitude slower than both simulated annealing and randomized hill climbing. This is due to the fact that the computation conducted by the genetic algorithm is much more complex than the other two. The entire population is first evaluated for fitness, then it is mated and mutated and reevaluated to make better models (have a fitter population). While RHC and SA took around 10 seconds for 1000 iterations, GA took around 1000 seconds.

The genetic algorithm also seems to reach a maximum at some number of iterations and becomes worse at higher iterations. This is a classic sign of overfitting, where the fittest model is too closely related to the training set and cannot perform as admirably over the test set.

The genetic algorithm has higher accuracy than the other two algorithms for 1000 iterations (by about 7%). This can be explained by the fact that in GA, for each iteration, only the fittest instances are kept, and the unfit ones are replaced with genetically modified ones. From this we can deduce that GA would reach the optimum model with fewer number of iterations. However, the training time with this strategy was 100 times more than the other two algorithms (as mentioned before).

1.5 Overall Comparison and Last Thoughts

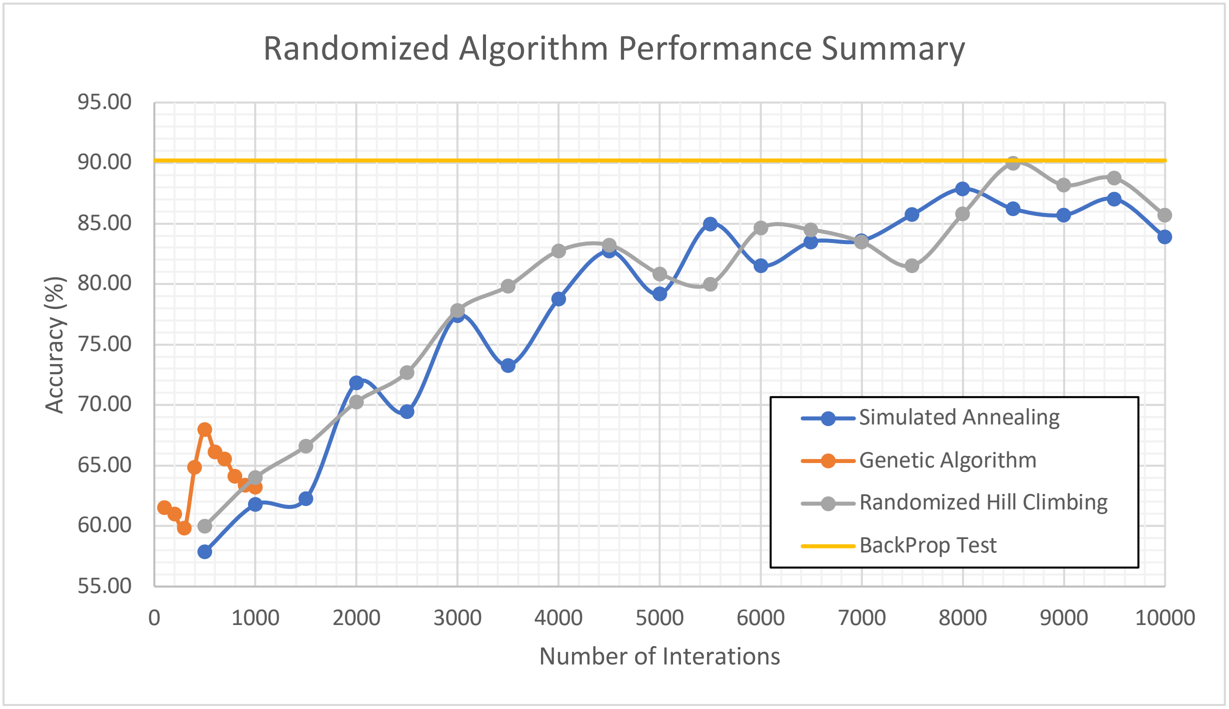

Figure 5. Comparison of test accuracies for all 3 algorithms and back-propagation

For this dataset, we can see that backpropagation provides better, if not, comparable results for all iterations considered. Both RHC and SA increase steadily, as explained above and genetic algorithms increase and then decrease (also explained above). Back-propagation performs better than any of the other algorithms because for this dataset, it is a continuous differentiable search space. With gradient descent, it is no surprise that the backpropagation converges faster than other algorithms. The randomized optimization algorithm's advantage will be apparent in a non-differentiable search space as they utilize random guess and check approach to find the global minima.

Part 2: Exploring various optimization problems

2.1 Flip-Flop Problem (Simulated Annealing)

Problem Statement

This problem takes as an input a bitstring (like “0101”) and outputs the number of flip-flops between 0 and 1 for the string. In this case, it would return 4, as it also includes the flip/flop for the first bit compared to the 0th bit that doesn’t exist.

This is a unique problem as it exposes the algorithm to some unique constraints and conditions. The maximum number of flips is the length of the bitstring and algorithms would try to find an optimum value in this neighborhood. Hence, the local maximum increases as the length of the bitstring increases, and this pattern delineates the performance of some algorithms from others.

Here, the optimization problem is to try to return a bit string with constantly alternating bits, such that the solution of the function is maximum (equal to length of the bitstring).

Experiment and Results

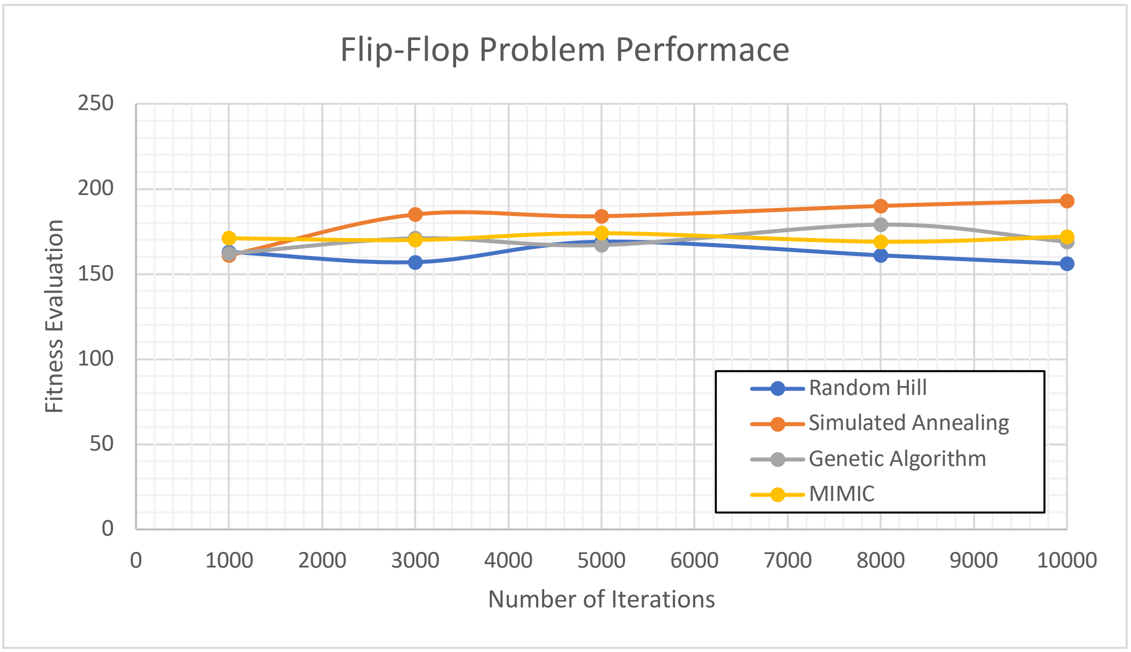

Figure 6. Performance of the 4 algorithms in solving the flip-flop problem

For this problem, the algorithms were given N = 200, and it tried to maximize the number of flip-flops it could create. This was repeated over several iterations to compare the results.

As seen from the graph, simulated annealing outperforms all other algorithms. This problem presents unique conditions which highlight the strengths of simulated annealing, as it has a discrete search space and local maxima are close to each other.

GA performs the worst because it explores a “small” neighborhood and most likely, gets stick at a local maximum. Any mutations from this state doesn’t lead to much better models/populations being created. RHC would also get stuck at a local maximum if no gain in fitness was seen due to flipping some bits.

On the other end of the spectrum, simulated annealing does the best. Initially, at high temperatures, it takes jumps in large increments and crosses any local maximums. As the temperature starts to cool, the jump size decreases and it converges to a steady state value. An additional experiment was conducted for simulated annealing where the cooling factor was changed. The cooling factor is a measure of how fast the temperature decreases, where a higher cooling factor means that the temperature was decreased slowly. With increasing cooling factor, the performance increased, as expected, as the algorithm could jump out of additional local maximums. This also required more iterations, however, to reach its steady state value.

The wall-clock time required to train and evaluate each algorithm was in the 0.5 second range other than MIMIC which took considerably longer (close to 20 seconds). This is due to the fact that MIMIC tries to keep track of the complex search space. Hence, the time taken increases exponentially with complexity of the search space. This solidifies simulated annealing's advantage over the other algorithm for this problem.

2.2 Traveling Salesman Problem (Genetic Algorithm)

Problem Statement

This is a famous NP hard problem, where the aim is to find the shortest route that passes through all points given, only once. (The best route to visit each country once, as an example). Each route has a certain cost associated with it (like the cost of the airplane flight). In order to get a function to maximize, the inverse of the shortest route is considered, as maximizing this in essence minimizes the route taken.

Experiment and Results

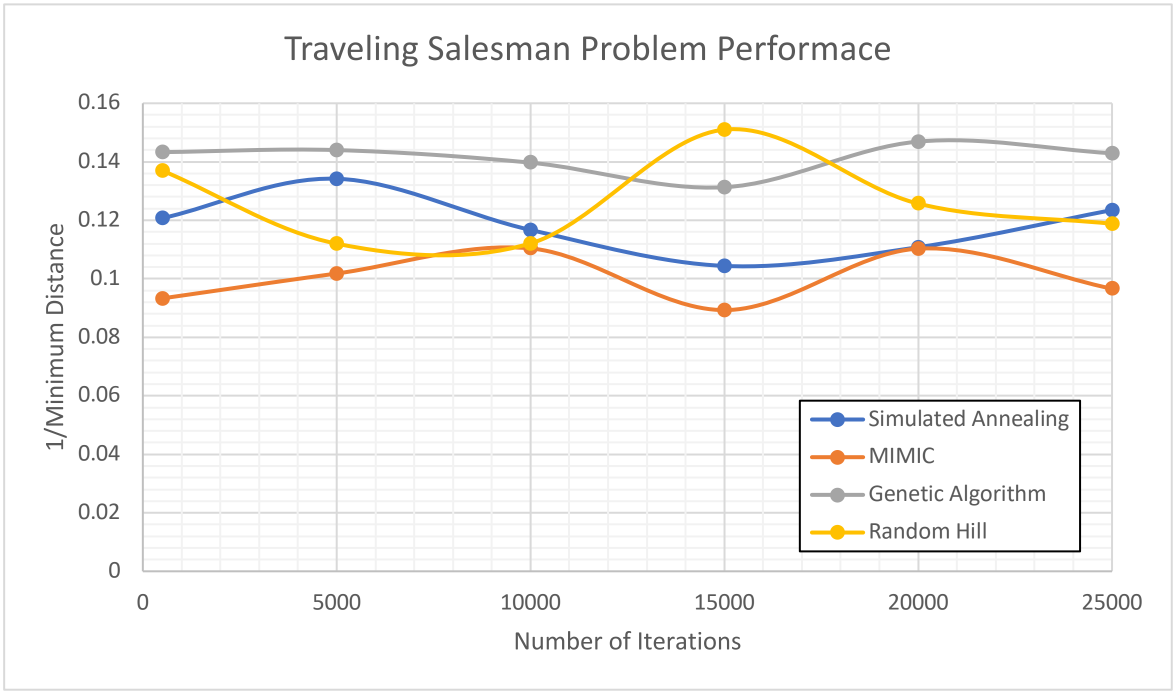

Figure 7. Performance of the 4 algorithms in solving the salesman problem

All the algorithms were tested with the same node/edge and graph structure (complex example was taken from the internet). The number of iterations of each algorithm was varied between 500 and 25,000 to evaluate the performance.

We can see that GA is the clear winner from all the algorithms. Only RHC achieves a higher score for a certain number of iterations, but RHC has an extremely volatile curve for this problem and seems to increase and decrease "randomly" with number of iterations, and hence is unreliable.

This problem is specifically suited to GA because it has a search space that grows exponentially with N, which is non-continuous and non-differentiable. GA also takes into account all possible solutions and proceeds with those that are genetically modified to perform the best on the fitness function evaluation. In this case, it can be envisioned as a breadth first search across all nodes. The reason GA performs better than other additionally, is because as the search space increases, the number of local maxima also increases and GA explores multiple good points/"offsprings" and picks the best ones to explore further. This couples with a mutation rate means that GA is likely to get closer to the global maxima and not get stuck in a complex search space.

As an aside, MIMIC performs the worst as it keeps track of the entire complex search space and tries to create a distribution and train a model from this distribution. Hence, MIMIC would be a poor choice for this problem.

2.3 4 Peaks Problem (MIMIC)

Problem Statement

The four peaks problem is a problem that has 4 peaks (2 global maximums and 2 local minimums). The fitness function is defined as the following:

$$f(\vec{X},T) = \max\left[tail(0,\vec{X}), head(1,\vec{X})\right] + R(\vec{X},T)$$

$$\begin{aligned} tail(b,\vec{X}) &= \text{number of trailing } b\text{'s in } \vec{X} \\ head(b,\vec{X}) &= \text{number of leading } b\text{'s in } \vec{X} \end{aligned}$$

$$R(\vec{X},T) = \begin{cases} N & \text{if } tail(0,\vec{X}) > T \text{ and } head(1,\vec{X}) > T \\ 0 & \text{otherwise} \end{cases}$$

Attributed to Charles Isbell's paper on this topic.

With a string of size N, a optima is found when there are T+1 leading 0s before a 1, or T+1 leading 1s before a 0. The two local minimums, and hence suboptimal solutions are a string of all 0s or 1s. The problem gets more difficult with increasing T.

Experiment and Results

For this problem, I fixed the value of N to be 200 and T to be 40. We can see from the results that MIMIC easily outperformed all the other algorithms. It matches the other algorithm's best performance with 1/10th the number of iterations.

It is a good choice with problems with a relatively homogenous search space. This is due to the fact that MIMIC begins with a uniformly distributed number and concentrations of random points, and then updates these probability densities after each iteration. It other words, it uses previously searched data points and their values to generate a probabilistic concentration map of the optimization domain.

However, it also takes significantly longer than the other algorithms to perform. In fact, the other algorithms can match the same performance with a greater number of iterations but in less time than MIMIC. The question then becomes about the cost to evaluate the fitness function. If the cost to evaluate the fitness function (costs other than time also to be considered) is relatively high, then MIMIC would be the best as it performs relatively little number of iterations and wouldn't evaluate the fitness as many times as the other algorithms. In this case, MIMIC is clearly the better option.

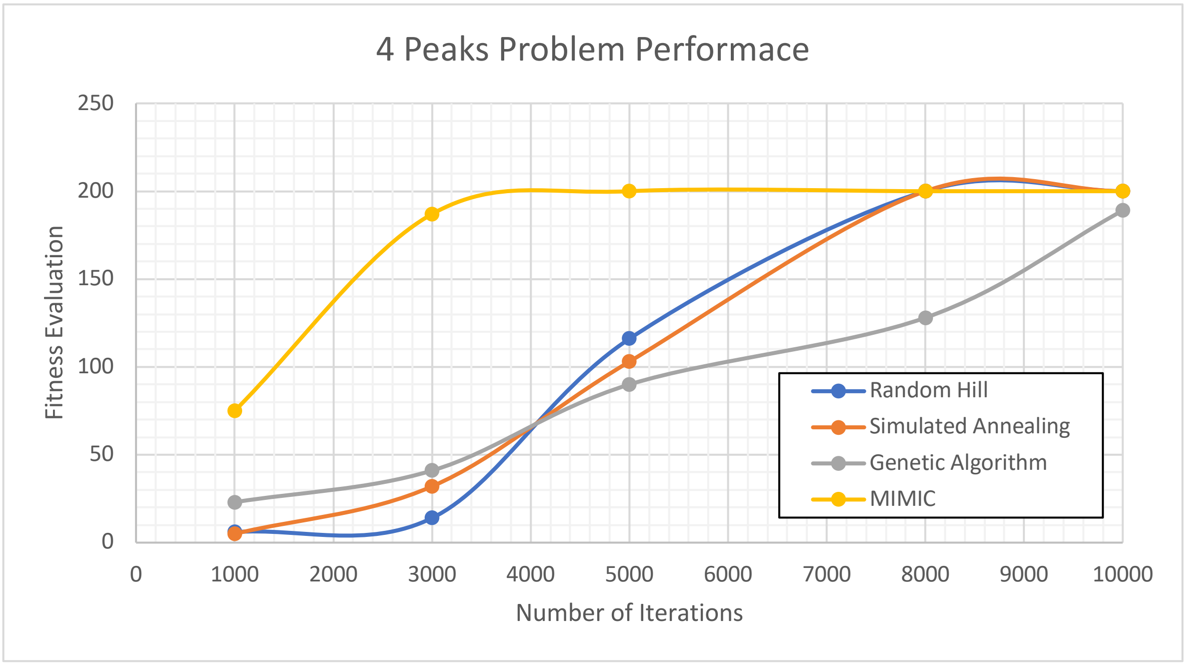

Figure 8. Performance of the 4 algorithms on the 4 peaks problem