Randomized Optimization

Sowmya Yellapragada (syellapragada3)

March 12, 2018

1 Abstract

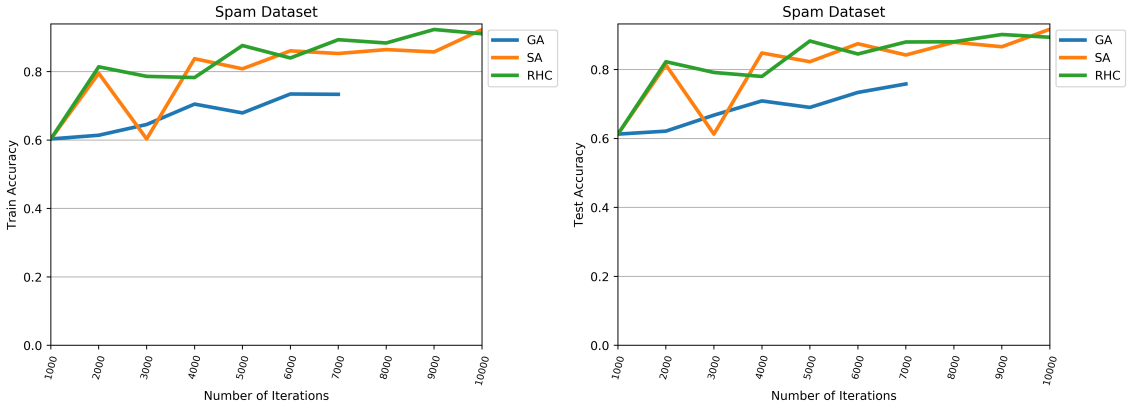
This paper explores the performance of different optimization algorithms, namely - randomized hill climbing (RHC), simulated annealing (SA), genetic algorithms (GA) and MIMIC. In the first section, the first three algorithms are applied to determine the weights in a neural network on spam data set. We compare the algorithm that produces best accuracy against back propagation weight calculation for the same neural network. Further in the next section, we apply all the four optimization algorithms to different optimization problems, and try to identify the algorithm that is a best fit for each of the problems individually. The analysis is done using JAVA ABAGAIL library provided with the assignment

2 Neural Networks

The weight(w) values of the neural network are described as its internal learn-able parameters which are used in computing the output values and are learned and updated in the direction of optimal solution i.e minimizing the loss by the network's training process and also play a major role in the training process of the neural network Model. The internal parameters of a Model play a very important role in efficiently and effectively training a Model and produce accurate results. This is why we use various optimization strategies and algorithms to update and calculate appropriate and optimum values of such model's parameters which influence our model's learning process and its output.

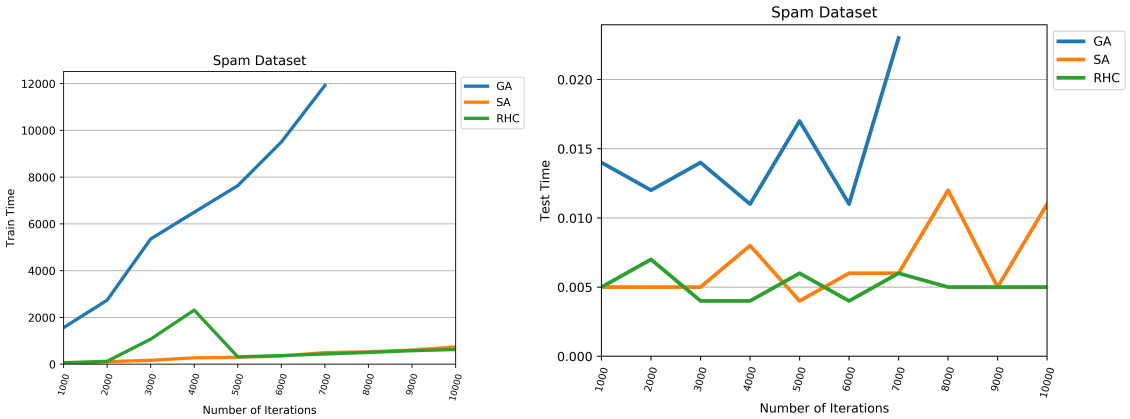
In this section, we apply three different randomized optimization algorithms: randomized hill climbing, simulated annealing and genetic algorithms to find the optimal weights of the neural networks for the Spam email database. We compare their performances based on the accuracy levels they produce and choose the algorithm that gives the best accuracy rate. We further compare the accuracy rate produced using back propagation methods, using scipy's Multi-layer perceptron classifier(MLP-Classifier).

Spam E-mail Database: Spam email is a major concern for most people using an email service. Important emails are often lost among the huge dump of spam emails. Simple classification based on what the user had marked spam is no longer sufficient. Machine learning algorithms have become common place for identifying trends among data that at the first sight looks random or un-suspicious. In this report we explore classification of email into spam (1) or not spam (0). Most of the 58 attributes in the data set indicate whether a particular word or character was frequently occurring in the e-mail. The dataset containing 4601 instances was classified into testing and training split containing 33%, 67% of the instances respectively. The neural network used in this application uses one hidden layer with 15 nodes. The cross validation accuracy and test accuracy with back propagation implementation are 0.91 and computation time is 2 secs



(a) Train accuracy over different iteration sizes (b) Train accuracy over different iteration sizes

Figure 1: Algorithm accuracy comparison: Spam Dataset

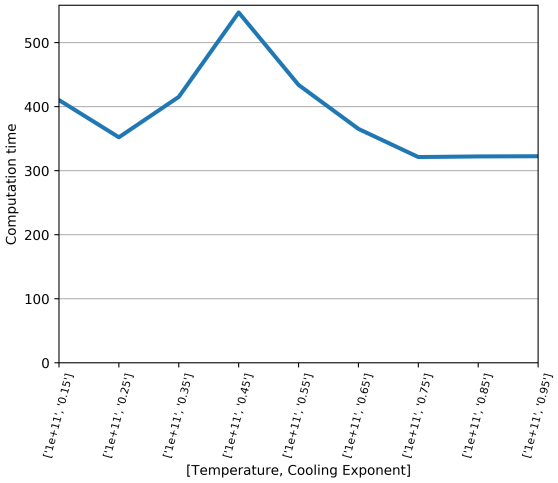


(a) Train time over different iteration sizes (b) Test accuracy over different iteration sizes

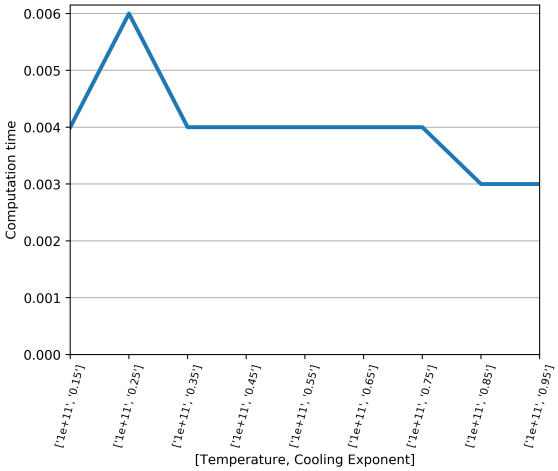
Figure 2: Algorithm computation time comparison: TSP

2.1 Randomized Hill Climbing, Simulated Annealing

From the graphs above, we can observe that randomized hill climbing and simulated annealing perform extremely well for the spam dataset. Both these algorithms rely on the inherent randomness in the dataset. They are more suitable when the optimal point is random over a wide range of points. This seems to be significant. It is interesting to note that the training and testing accuracies produced by both these algorithms are closely coupled. Their training times seem to be equivalent as well. Significant difference is observed in the computation time during testing.
It can be observed for fewer number of iterations, the accuracy levels produced by both these algorithms is fairly low, and increases steadily as the iteration count increases. At iteration counts close to 10000, they seem to achieve and accuracy level that is close to 0.9, which is close to the accuracy of (0.91) obtained through back propagation on this same neural network.

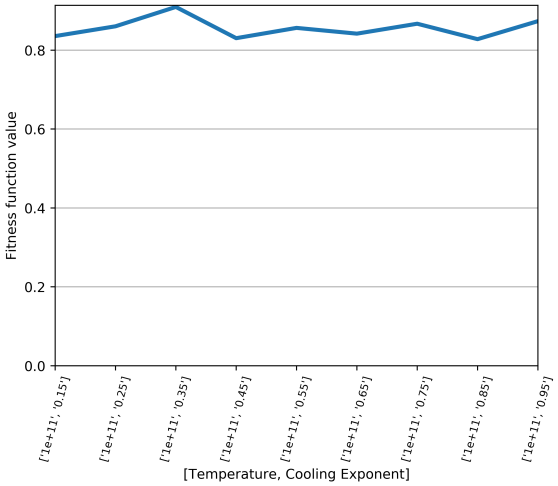


(a) Training computation time over different input parameters for SA

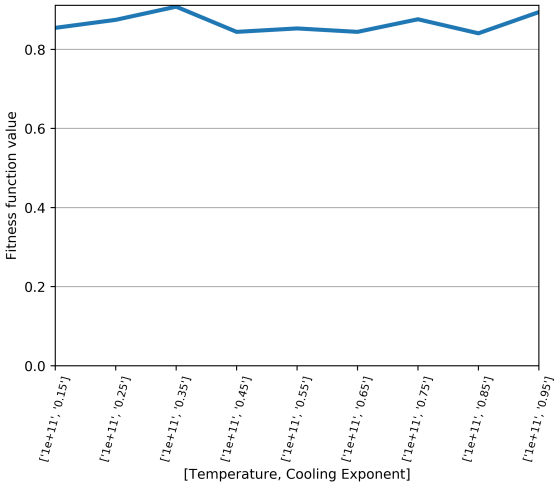


(b) Testing computation time over different input parameters for SA

Figure 3: Computation time comparison for Simulated Annealing



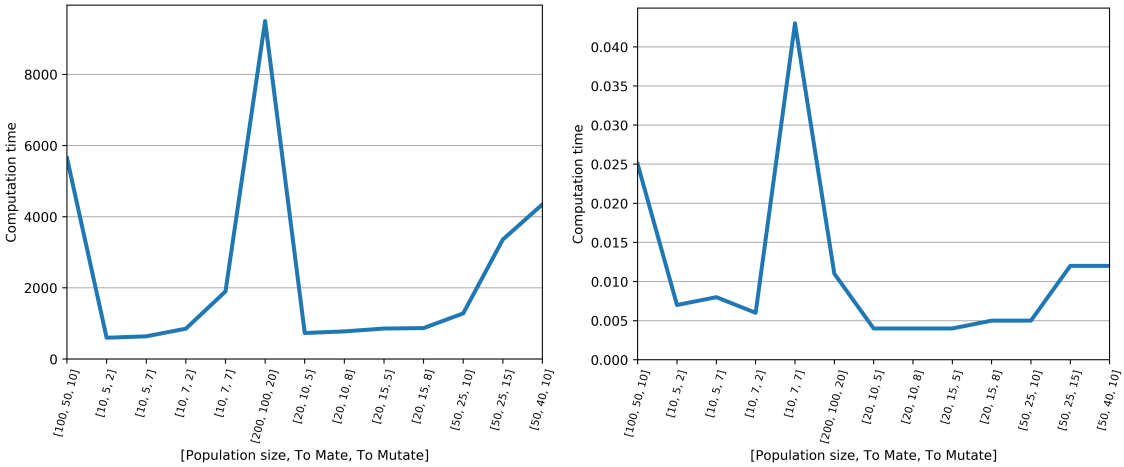
(a) Training accuracy over input parameters for SA



(b) Testing accuracy over input parameters for SA

Figure 4: Accuracy comparison for Simulated Annealing

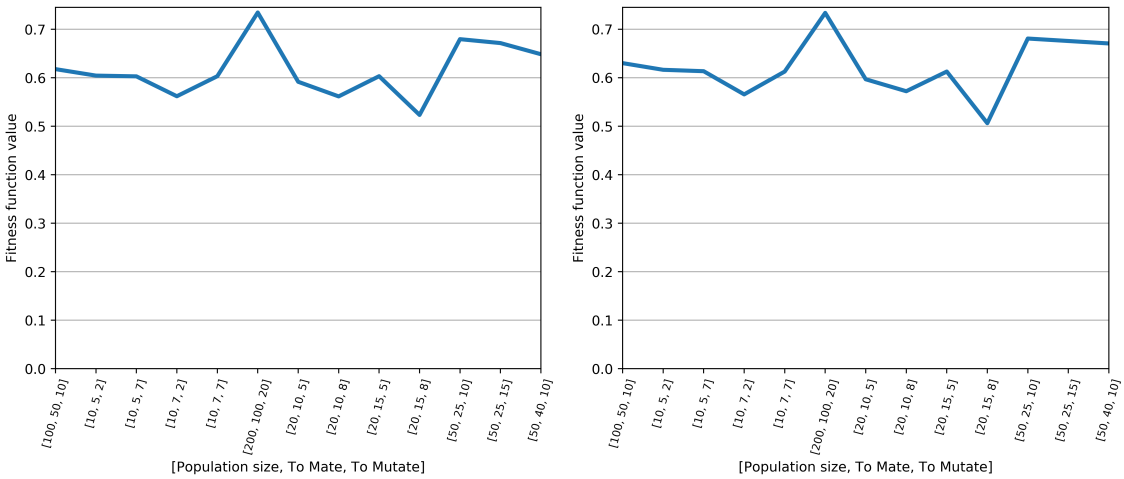
2.2 Genetic Algorithm



(a) Training computation time over input parameters for GA

(b) Testing computation time over different input parameters for GA

Figure 5: Computation time comparison for Genetic algorithms



(a) Training accuracy over input parameters for GA

(b) Testing accuracy over input parameters for GA

Figure 6: Accuracy comparison for Genetic algorithms

The above graphs compare the test and train computation accuracies and time for genetic algorithm implemented on the neural network structure. As we can see the performance of GA is fairly mediocre in terms of accuracy and extremely slow in terms of time, in comparison to randomized hill climbing and simulated annealing.

2.3 Summary and Conclusion

Simulated Annealing and Randomized hill climbing provide good accuracy levels at large iteration counts, as we increase the iteration counts, it is possible for us to achieve the accuracy produced back propagation on neural networks, but at the expense of large computation times. Back propagation hence fares better in comparison to the randomized optimization algorithms

3 Optimization Problems

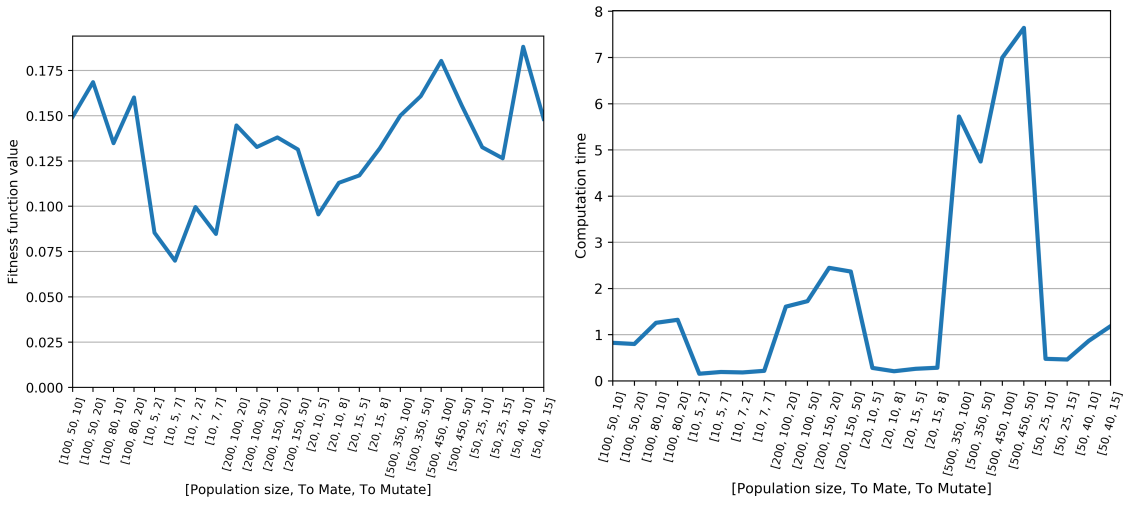
In this section we apply randomized hill climbing, simulated annealing, genetic algorithms and MIMIC to different optimization problems. We explore the performance to each of these problems.

Methodology:
Each of the optimization problems have been implemented using ABAGAIL library, with varying iteration counts as specified for each problem. Input parameters for each of the algorithm (except RHC, since it doesn't require any inputs) have been varied for every iteration count. The best input parameter for each algorithm was observed. We then compare each of the algorithms (with the above input parameters) and plot the graphs to compare their relative performance.

3.1 Traveling Salesman Problem

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point. This is a classic NP-hard problem, and it is particularly interesting since it has many applications ranging from finding optimized routes in the transportation industry to optimal computer wiring

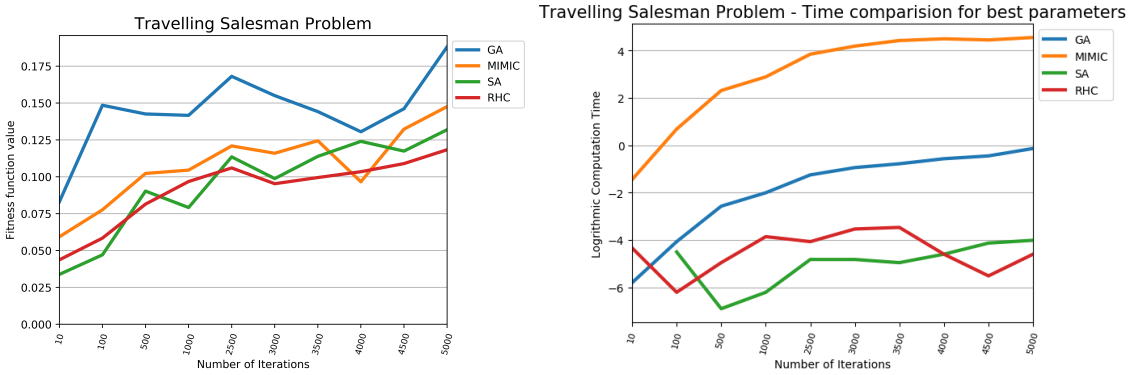
Inputs:
The number of cities that need to be visited, has been set to 50 and the distance between the cities is a random double value, between 0.0 and 1.0. Iteration counts were varied as follows - 10, 100, 500, 1000, 2500, 3000, 3500, 4000, 4500 and 5000.
The **best input parameter** observed for the algorithms are as follows-
GA (Population size, to mate, to mutate): 50, 40, 10
MIMIC (Samples to generate, samples to keep): 200, 20
SA (Initial temperature, cooling exponent): $e^{12}, 0.25$



(a) Fitness function value vs input parameter for GA

(b) Computation time vs input parameter for GA

Figure 7: Performance of best optimization algorithm (GA) for TSP, across different input parameters



(a) Fitness function value vs iteration count

(b) Logarithmic computation time vs iteration count

Figure 8: Algorithm comparison: TSP

It is easy to notice that genetic algorithms (GA) outperforms all the other algorithm in determining the most optimum fitness function value. Further from graphs plotted for the GA algorithm for different input parameters, it is evident that the fitness function value peaks at multiple points, and the global peak is at (50, 40, 10), also the computation time for this input is quite low. Note that these values correspond to the 5000th iteration for each input value. Since no convergence was directly evident, we chose to restrict our experiment to 5000 iterations.

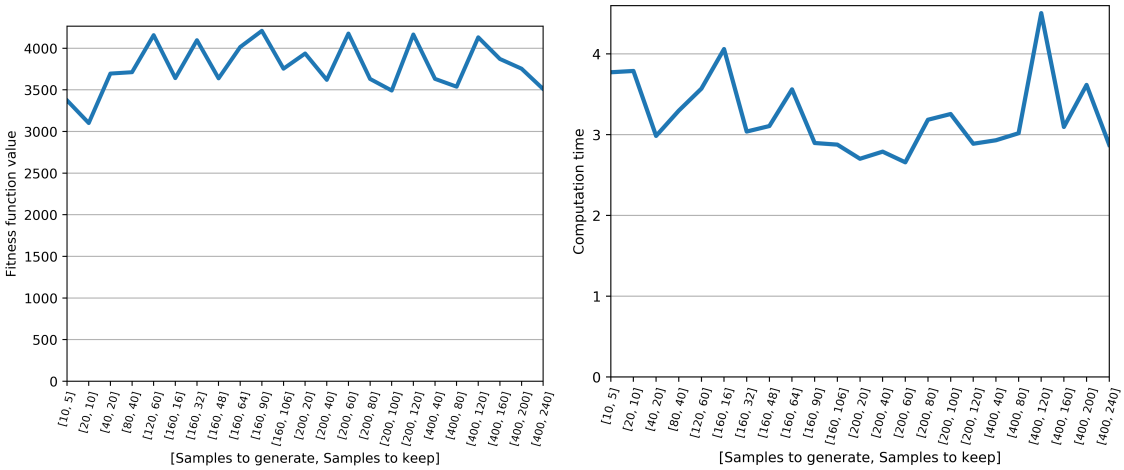
3.2 Knapsack

Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. Knapsack problem has been studied for over a century and has many application. It often arises in resource allocation where there are financial constraints or time constraints.

Inputs:
We provide an input of 40 unique items, with 4 copies of each, a maximum weight and volume per item is 50. The maximum volume in the sack is restricted to 3200. We implement each of the four optimization algorithms by varying the iteration count of 10, 100, 500, 1000, 2500, 3000, 3500, 4000, 4500, 5000 and 6000.
The **best input parameter** observed for the algorithms are as follows-
GA (Population size, to mate, to mutate): 500, 450, 100
MIMIC (Samples to generate, samples to keep): 160, 90
SA (Initial temperature, cooling exponent): $100000, 0.95$

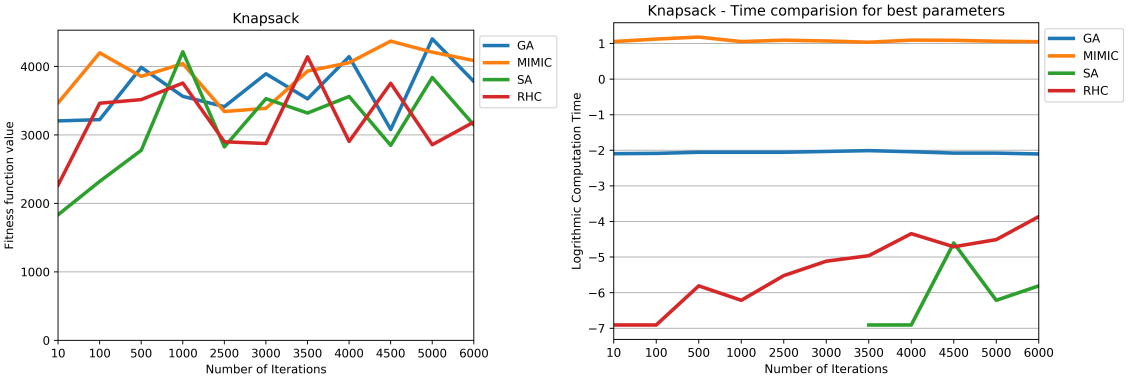
Figure 9 shows the plot of different input parameters for MIMIC against fitness function value and time. Notice the many peaks achieve the maximum fitness function. We choose the input parameter that helps us achieve the peak (maximum fitness function value) with least computation time possible. This was observed at the input parameter - 160, 90. Similar analysis has been performed to obtain the best input parameters in other algorithms as well. Note that these values correspond to the 5000th iteration for each input value. Since no convergence was directly evident, we chose to restrict our experiment to 5000 iterations.

From figure 10, It is easy to notice that genetic algorithms (MIMIC) outperforms all the other algorithm in determining the most optimum fitness function value.



(a) Fitness function value vs input parameter for MIMIC (b) Computation time vs input parameter for MIMIC

Figure 9: Performance of best optimization algorithm (MIMIC) for Knapsack, across different input parameters



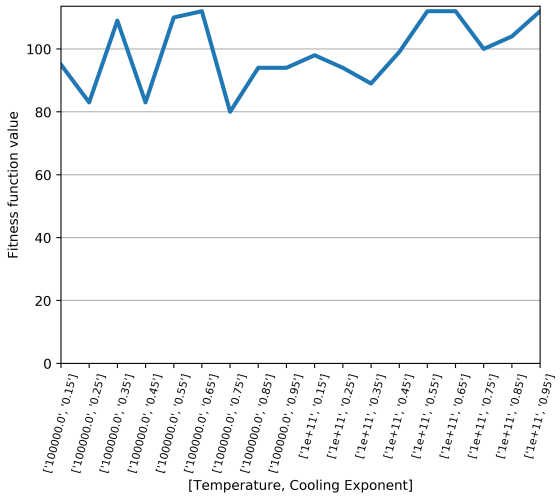
(a) Fitness function value vs iteration count (b) Logarithmic computation time vs iteration count

Figure 10: Algorithm comparison: Knapsack

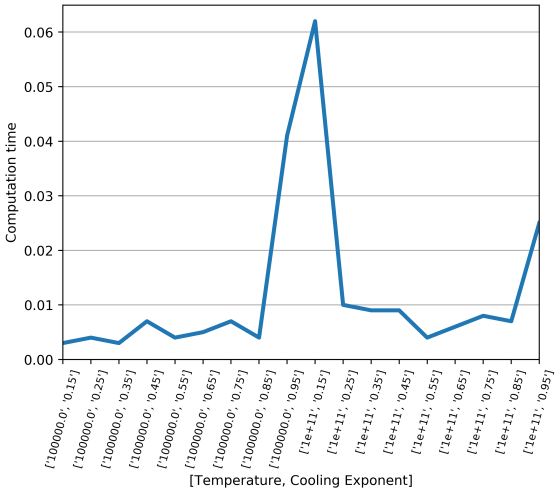
3.3 Continuous Peaks

In this problem, we are trying to determine the maximum number of continuous 1s or 0s in a binary bit string. If this maximum value is grater than a given threshold T, then we return the number maximum value determined + size of the bit string, otherwise, we return the maximum value itself. Applications of this problem might not be directly evident, however, it is a very interesting problem to tackle.

Inputs:
We start with an input bit string length on N=60. The threshold value T is set to N/60 = 10. We implement each of the four optimization algorithms by varying the iteration count of 10, 100, 500, 1000, 2500, 3000, 3500, 4000, 4500, 5000, 6000, 7000, 8000, 9000, 10000.
The **best input parameter** observed for the algorithms are as follows-
GA (Population size, to mate, to mutate): 200, 150, 10
MIMIC (Samples to generate, samples to keep): 40, 20
SA (Initial temperature, cooling exponent): $e^{11}, 0.55$

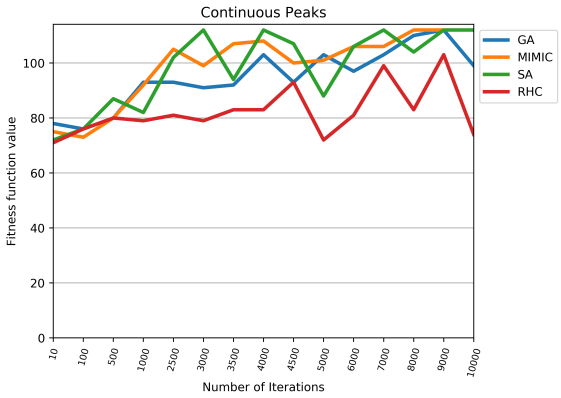


(a) Fitness function value vs input parameter for SA

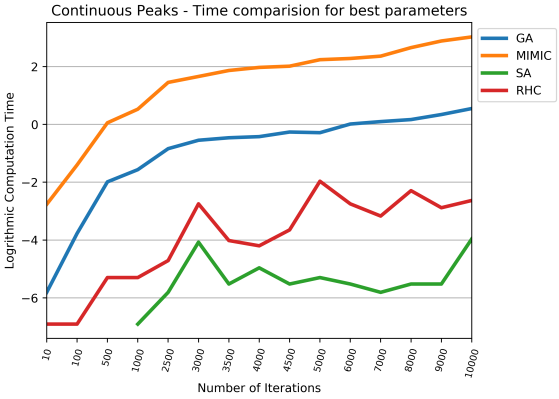


(b) Computation time vs input parameter for SA

Figure 11: Performance of best optimization algorithm (SA) for Continuous Peaks, across different input parameters



(a) Fitness function value vs iteration count



(b) Logarithmic computation time vs iteration count

Figure 12: Algorithm comparison: Continuous Peaks

It is interesting to notice from Figure 12 (a) that both SA and MIMIC are converging to a maximum value at 9000 iterations. However, from Figure 12 (b) it is clearly evident that the computational time for SA is far less than MIMIC, hence we can say that SA is the most suitable optimization algorithm for continuous peaks. Figure 6 indicates the fitness function value and computation time against different input parameters for the SA algorithm.

3.4 Summary and Conclusion

From the analysis performed on each of the optimization problems, it is clearly evident that no optimization algorithm is best suited for all. The performance of each optimization algorithm is dependent on the nature of the problem. MIMIC and Genetic algorithms both operate well in problems where retaining history is useful, and building up on historic computation enhances the solution, as indicated in Knapsack and traveling salesman problem. Simulated Annealing and random hill climbing are more suitable when the optimal point is random over a wide range of points, as indicated in continuous peaks analysis and spam dataset's neural network analysis in previous section. It is interesting to note that simulated annealing performs better than randomized hill climbing, as it first starts with a random walk but then with each iteration moves toward the optimal solution, enabling the exploration of more of the domain space