Randomized Optimization

Georgia Institute of Technology CS7641 Machine Learning Assignment 2

Yan Cai

GTID: ycai87

Abstract

This paper explores different randomized optimization algorithms. The algorithms are random hill climbing, simulated annealing, genetic algorithms and MIMIC. In part one, the first three optimization algorithms are applied to determine the weights in a neural network on phishing website dataset. In part two, three different optimization problems are analyzed to evaluate all four optimization algorithms. The whole analysis is done in JAVA ABAJAIL library.

Optimization Algorithms

Randomized optimization algorithms are often used to get the global optimum value for different functions that are usually not continuous or differentiable. We will go over all four different randomized optimization algorithms.

Randomized Hill Climbing (RHC)

Randomized Hill Climbing searches for local optima by moving towards neighbors that have more optimum values until it reaches a peak. RHC randomly starts its position in order to avoid getting stuck at local optimum. The random start also increases the chance to bring closer to the global optimum. RHC is run using ABAGAIL library.

Simulated Annealing (SA)

Simulated Annealing comes from metallurgy where ductility in metals can be improved by being heated to high temperature and then slowly being cooled to keep its structure. The algorithm evaluates whether the new neighbor point is better or worse than the current point in fitness function values. If the new point is better, it becomes the next point. However, if the current point is better, the algorithm would calculate the probability to move to the new point based on the acceptance function. The algorithm accepts a worse point based on an acceptance function, which is a function of initial temperature and cooling rate. SA accepts worse solutions with a slow decrease in probability as it explores the neighbor spaces. By repeating the process, SA is able to reach the global optima slowly but more extensively. SA is run using ABAJAIL library.

Genetic Algorithms (GA)

Genetic algorithms tries to find optimum solutions based on based on a natural selection process that mimics biological evolution. The algorithm repeatedly modifies the population of solutions. At each step, GA randomly selects individual solutions from the population and make them as parents to produce the children for the next

iteration of solutions. Over the iterations, the population would evolve to the optimal solution. The children can be determined by mutating and mating different parts of population. One of the drawbacks for GA is that it does not scale with problem complexity. When the number of elements becomes larger, the search space increases exponentially. GA is run using ABAGAIL library.

MIMIC

Mutual Information Maximizing Input Clustering (MIMIC) algorithm finds the optimum value by evaluating the probability density functions. MIMIC uses the probability densities to build the structure of the solution space at every iteration. Over the iterations, it would generate a better solution structure and sample the points from the solution space in order to find optimal values. MIMIC is run using AABAGAIL library.

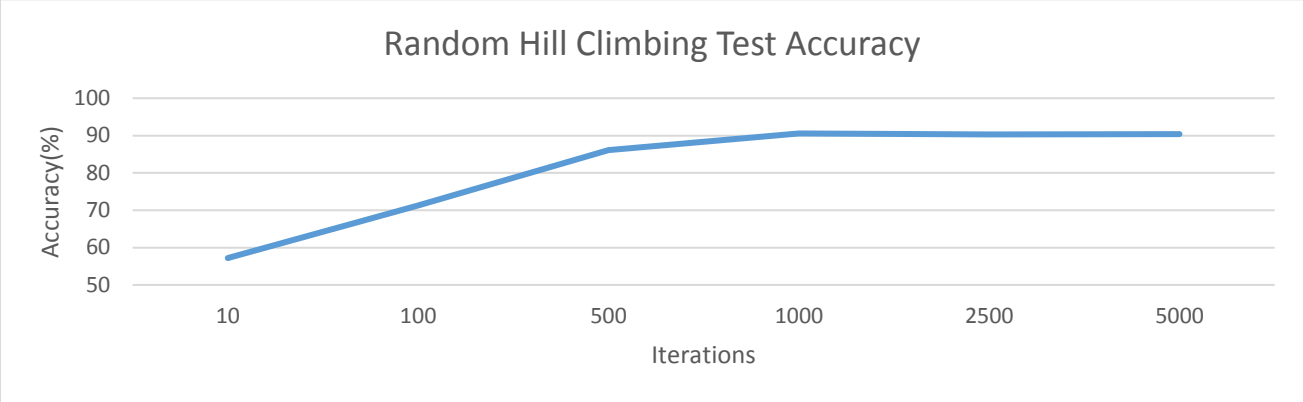
Part 1: Neural Networks with Random Optimization

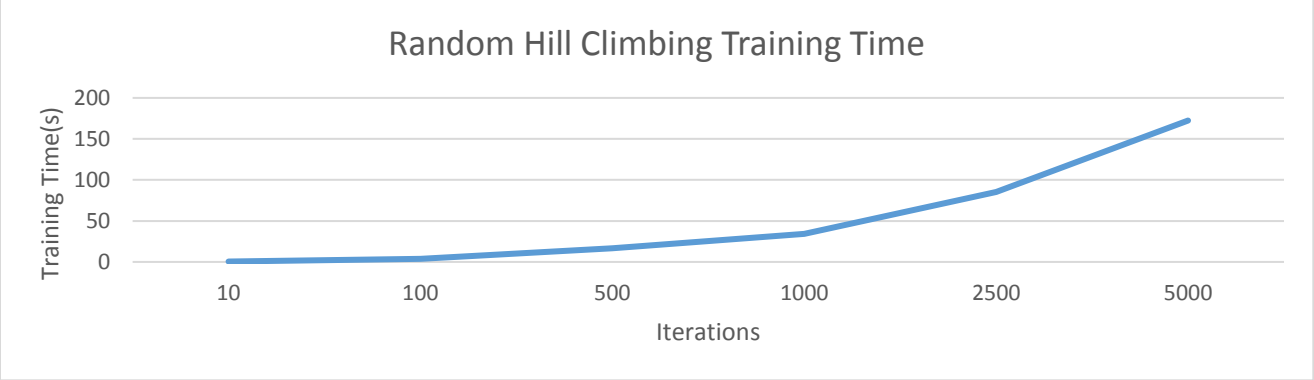
In this section, we apply three different randomized optimization algorithms: randomized hill climbing, simulated annealing and genetic algorithms to find the optimal weights of the neural networks for phishing website dataset.

Dataset and Methodology

The phishing website dataset has 11055 instances, 30 attributes and 2 classes. The classification is binary, indicating whether the instance is phishing website or not. The dataset is split into 70-30. 70% of the dataset is training set and the remaining 30 % of the dataset is testing set. All randomized algorithms are run with one hidden layer with 16 perceptron units. All tests are run in ABAGAIL. We tune the parameters for each algorithm with iterations = 10, 100, 500, 1000, 2500, 5000 and in the end we also compare all three algorithms together as well as results using Weka to do gradient decent back propagation neural networks.

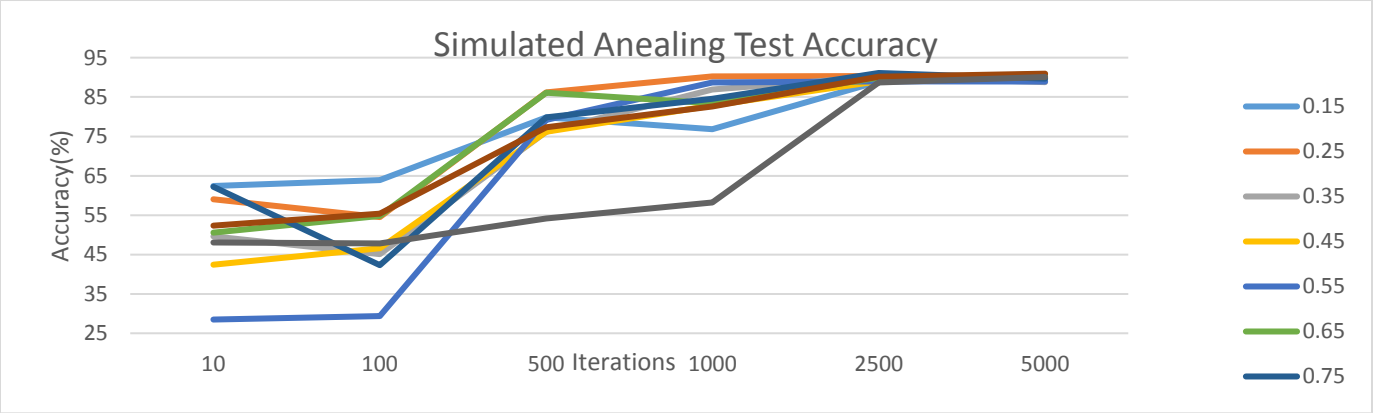
Randomized Hill Climbing

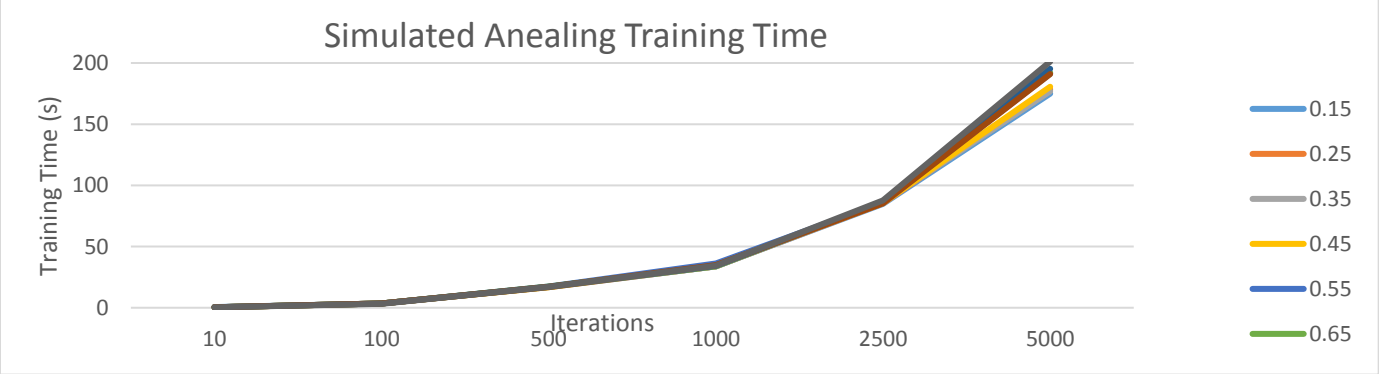




From the results, RHC converges as the iteration increases, but still takes more time with more iterations. The accuracy stays at around 90.3%.

Simulated Annealing



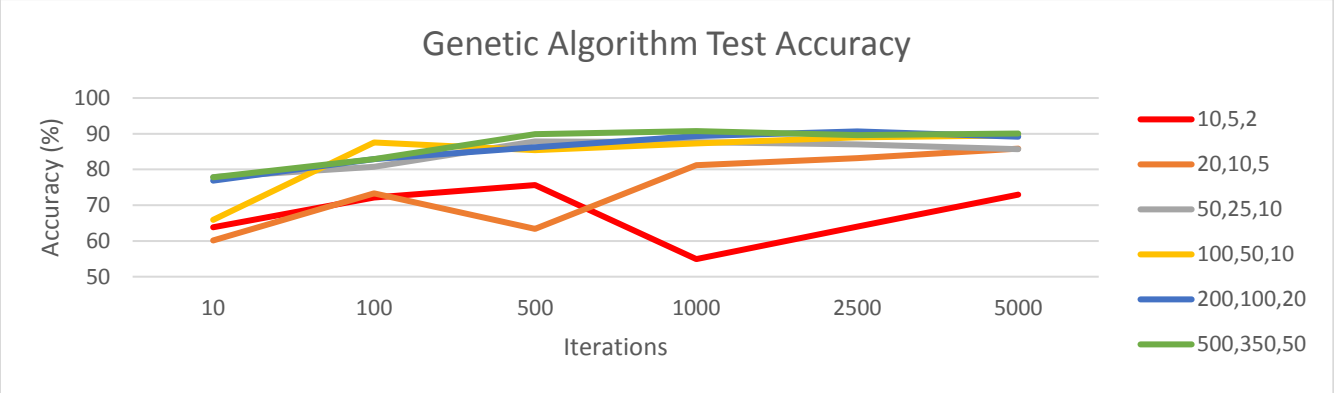


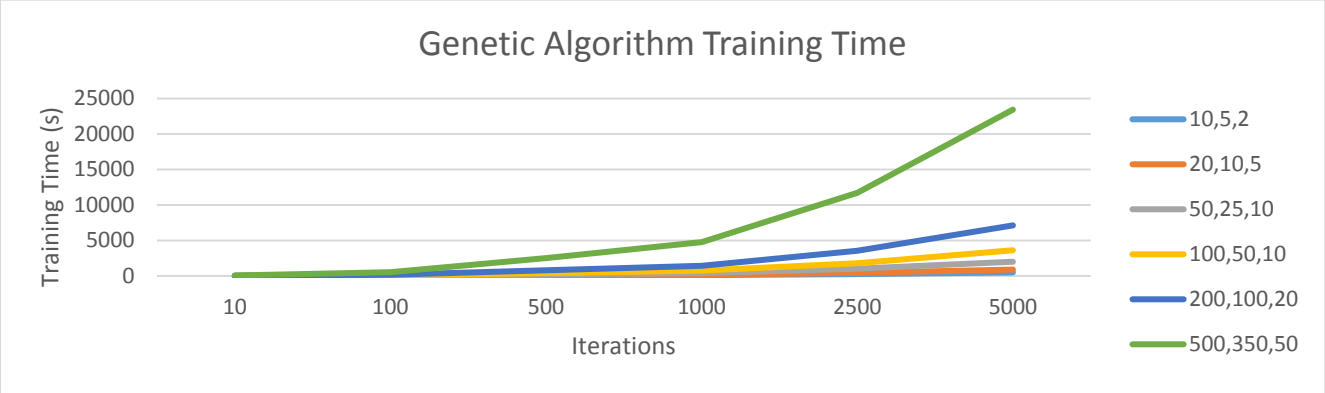
For simulated annealing, the performance is relatively low at fewer iterations and the accuracy starts to increase as the iterations increase. The tables shows the statistics at iteration = 5000. The performance is very close and training time increases as iterations increase.

Cooling Exponent	Test Accuracy	Training Time	Test Time
0.15	89.961	174.994	0.011
0.25	89.901	178.39	0.011
0.35	88.785	176.898	0.011

0.45	89.599	180.587	0.011
0.55	88.906	191.357	0.012
0.65	90.745	192.141	0.013
0.75	89.659	195.283	0.016
0.85	90.926	190.956	0.013
0.95	90.081	201.172	0.014

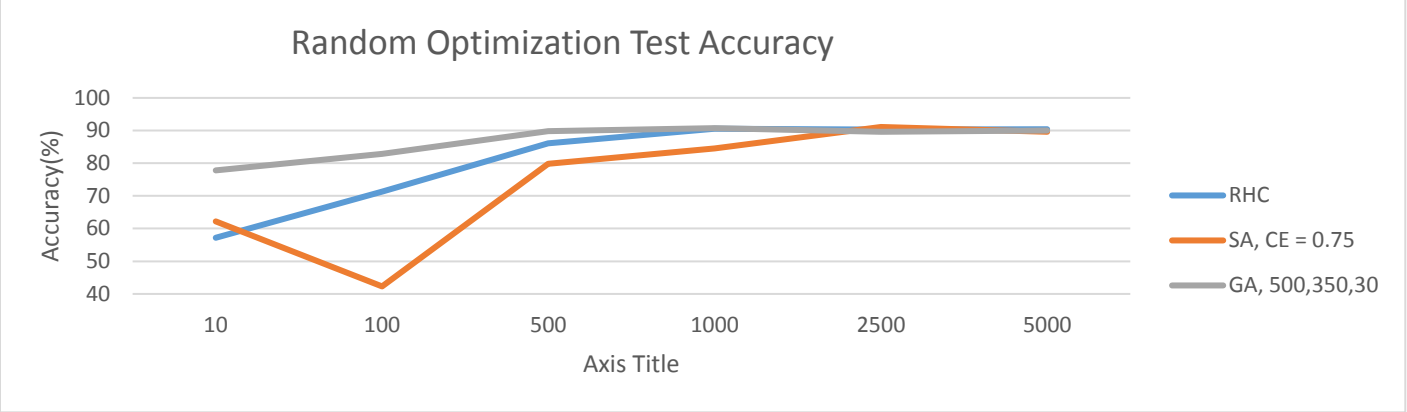
Genetic Algorithms

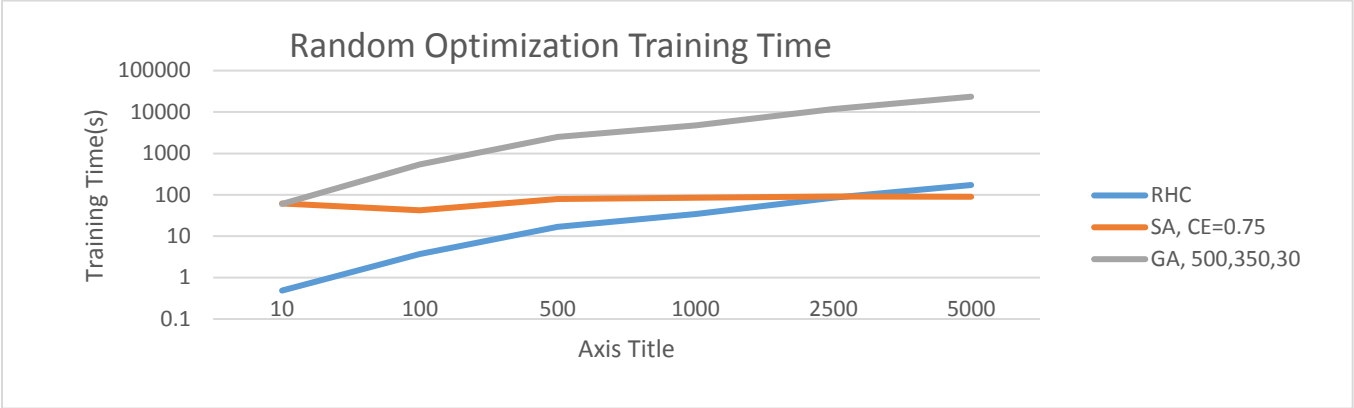




From the figures, we can see larger population size tends to have better accuracy. The accuracy converges to around 90% at iteration 5000. However, more population time would take a significantly more time while only achieving same performance.

Algorithms Comparison





Comparing all three algorithms, RHC achieves the best performance at iteration = 5000, test accuracy = 90.383. Furthermore, RHC takes much less training time. It is computationally prohibitive to run GA at large population size, in some times, it takes many hours.

With back propagation neural networks in Weka, we actually achieve 96.1724% test accuracy and the training time is 96.1724s. Back propagation beats the computation beats all three randomized optimization algorithms in both accuracy and time.

Part 2: Optimization Problems Analysis

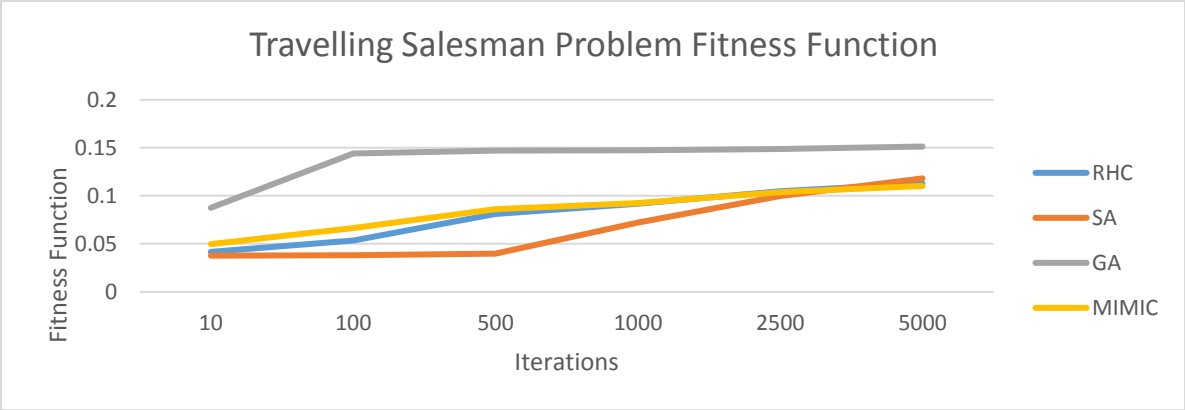
There optimization problems are analyzed to evaluate four different optimization algorithms mentioned above. The problems are traveling salesman, continuous peaks and knapsack.

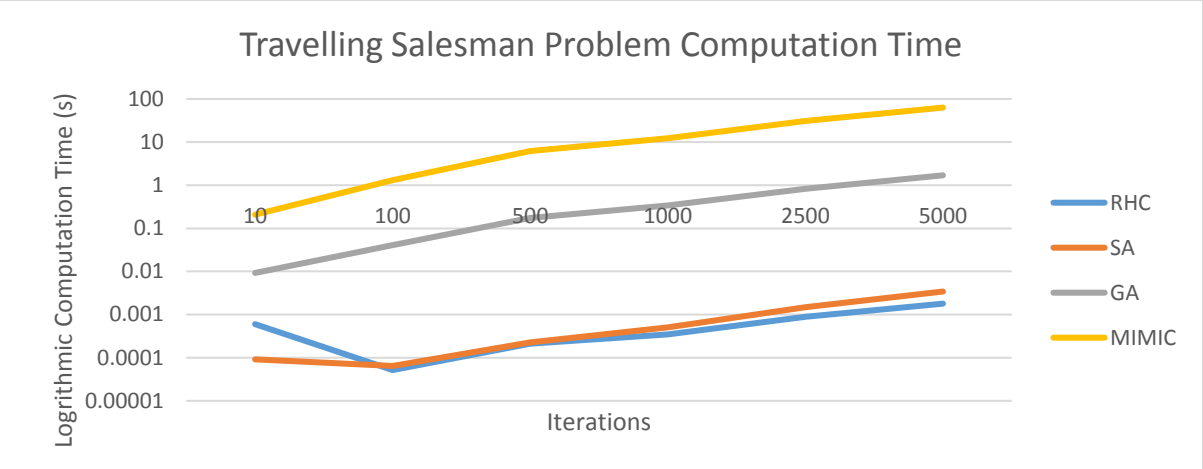
Methodology

Each optimization problem is run with four different optimization algorithms by an iteration count of 10, 100, 500, 1000, 2500 and 5000. In addition, each algorithm is run 10 times in each iteration. Fitness function outputs are compared among different algorithms as well as computation time. Both fitness function and computation time is averaged over 10 runs. We then pick best algorithm for each optimization problem based on fitness function output and computation time.

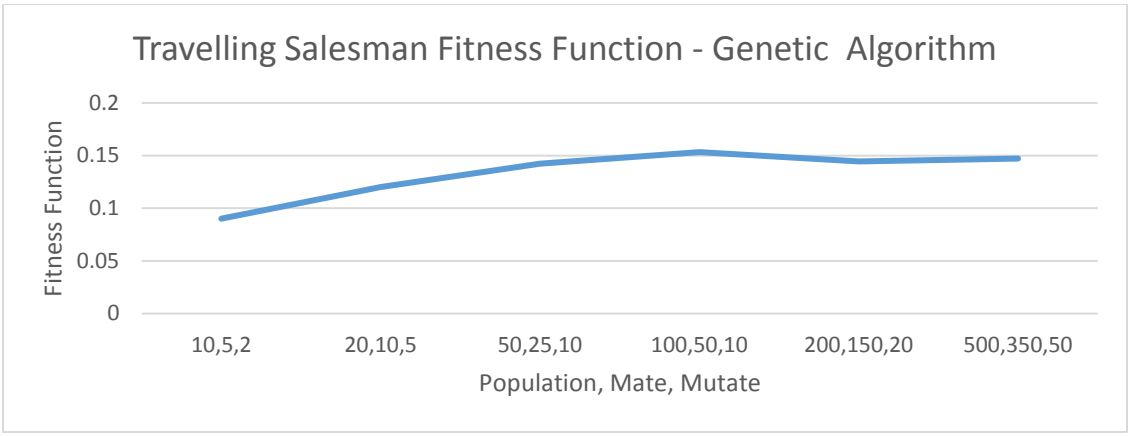
Travelling Salesman Problem

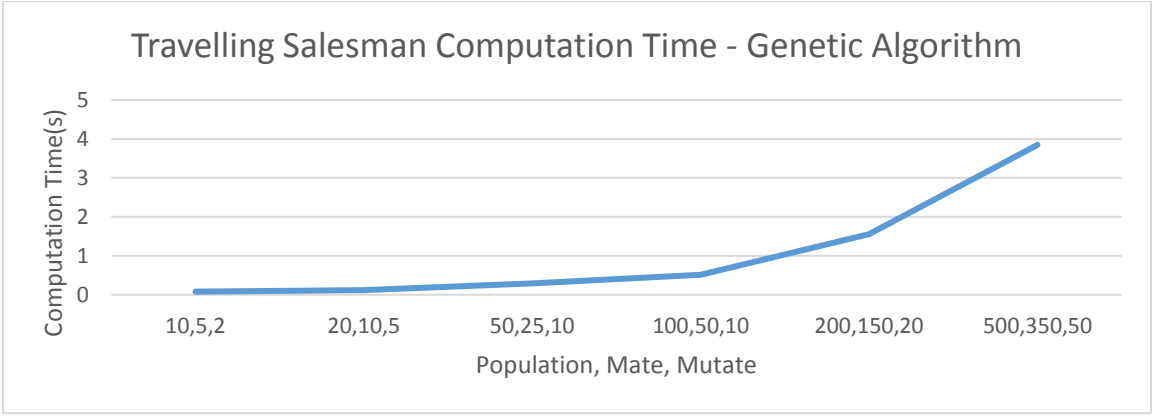
Travelling salesman problem is a classic NP-hard problem. The objective is to find the shortest round trip between different points while visiting each point only once. It is a problem with practical applications in transportation and travel planning. We set N = 50 in ABAGAIL.





From the figures above, we can see that **genetic algorithm** is the best performing algorithm for the problem in terms of fitness function values. Computation time show in the figure shows that genetic algorithm takes more time than simulated annealing and random hill climbing, but significantly less time than MIMIC. We also examine the performance of genetic algorithms by comparing the different parameters. The population size is the size of population survives from the iteration. Mate is the crossover number representing the mating between the individuals. Mutate is the mutation that causes random modifications.

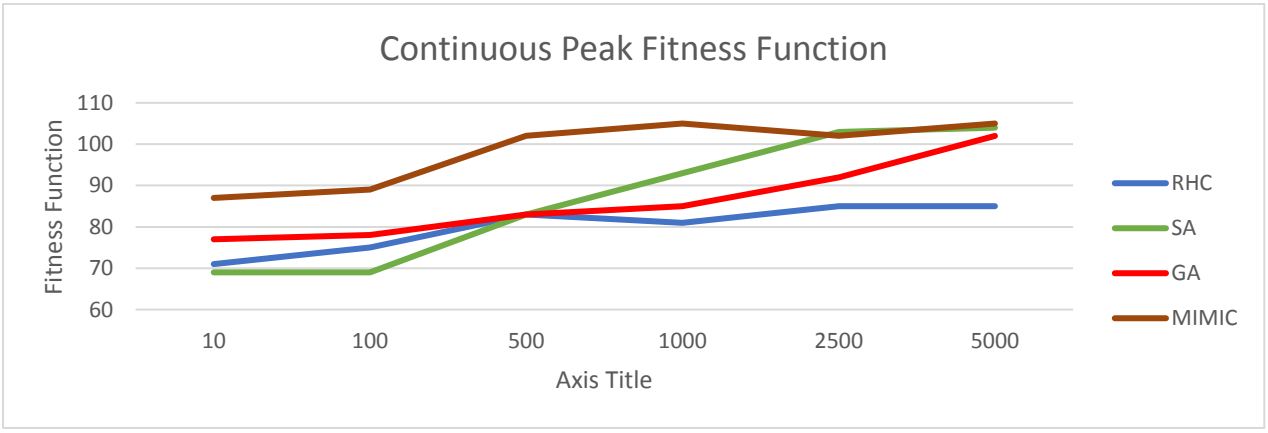


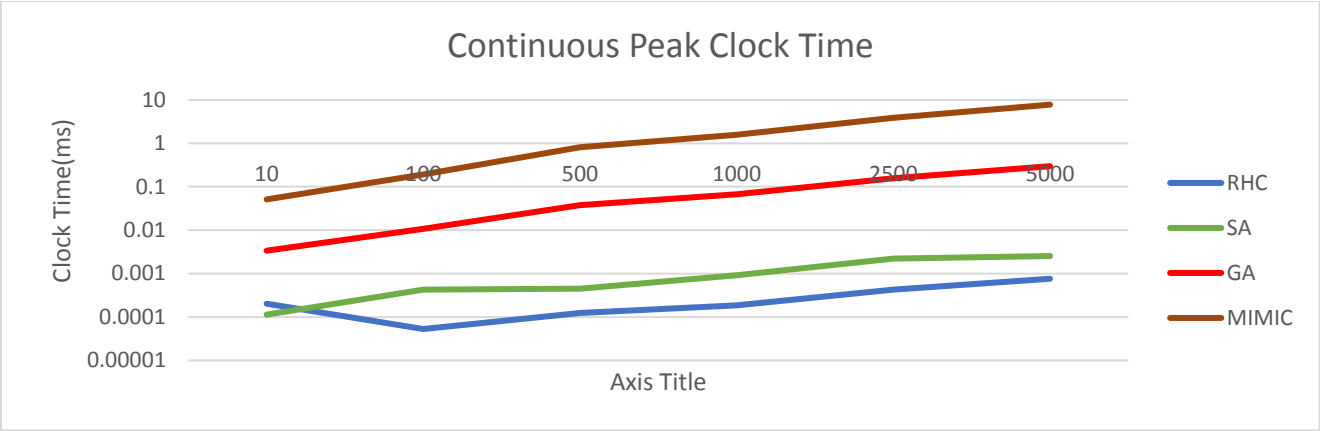


The figure shows that the best configuration is when population size is 100, mate is 150 and mutate is 10. The computation time increases as the population size increases.

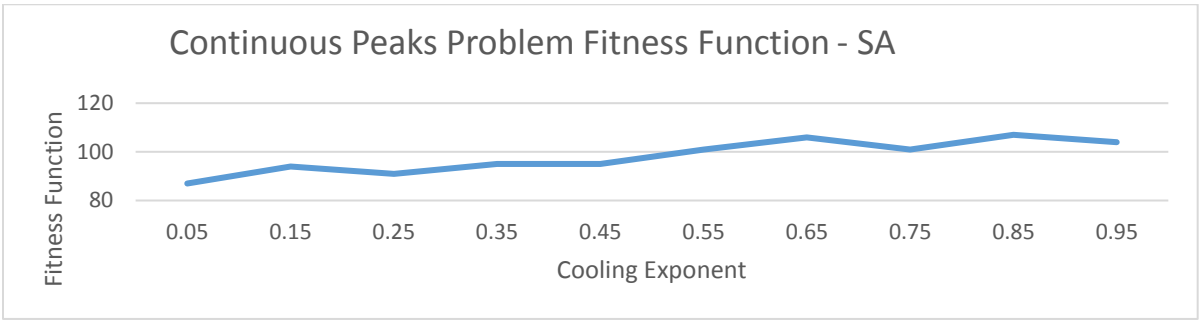
Continuous Peaks Problem

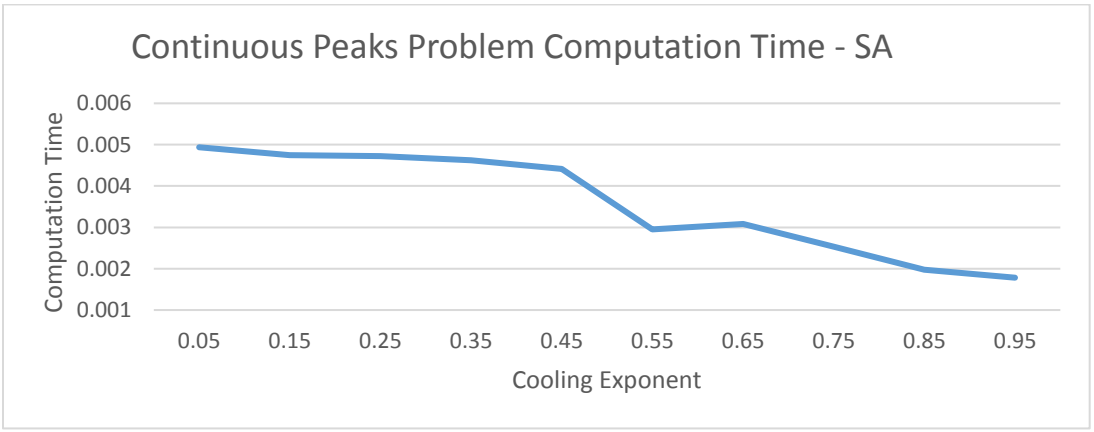
Continuous peaks problem is an optimization problem that tries to find out the highest peak (global optima) with respect to other peaks (local optima). In ABAGAIL, we set N = 60 and T = 6.





As we can see from the figures, MIMIC and simulated annealing has close performance at iteration 2500 and 5000. However, simulated annealing takes far less time than MIMIC and still have comparable performance. Therefore, **simulated annealing** is the best algorithm in this problem. We further examine simulated annealing by comparing the cooling exponent at iteration 5000 with initial temperature = 1E11.

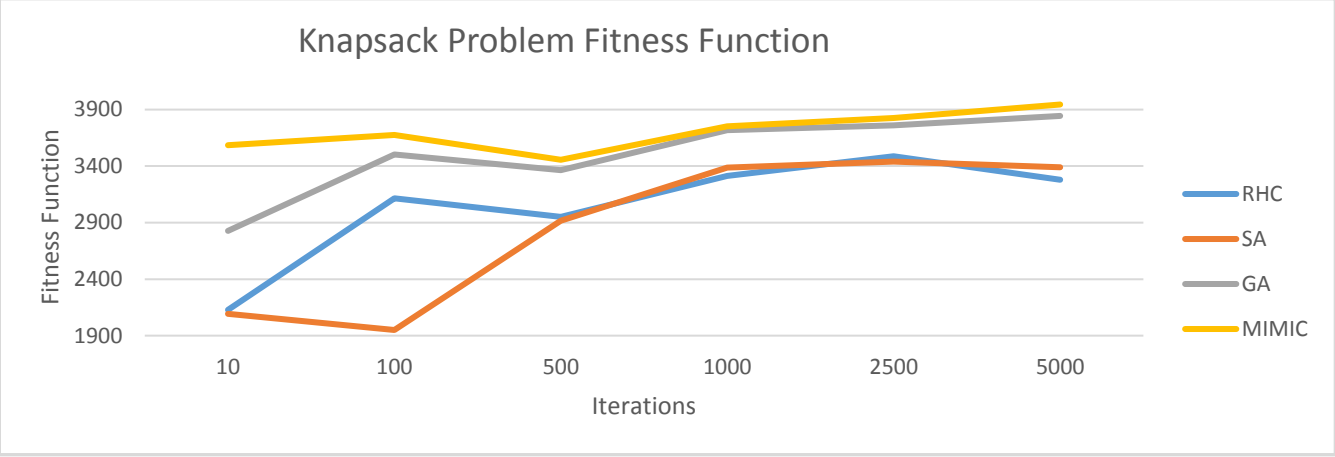


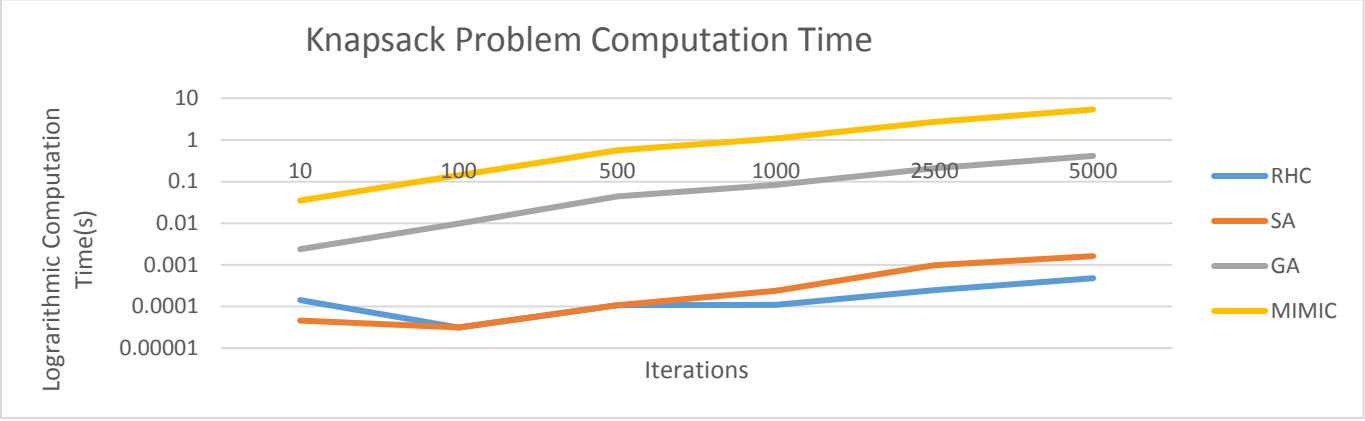


From the figures, we can see when cooling exponent = 0.85, SA performs the best with highest fitness function value. Meanwhile, we also find out the computation time decreases as cooling exponent increases. This could be because larger cooling exponent means the temperature drops more slowly, which makes the acceptance probability decrease more slowly, therefore, it is more likely to accept worse solutions instead of aggressively moving towards other points and less likely to stuck in the local maxima.

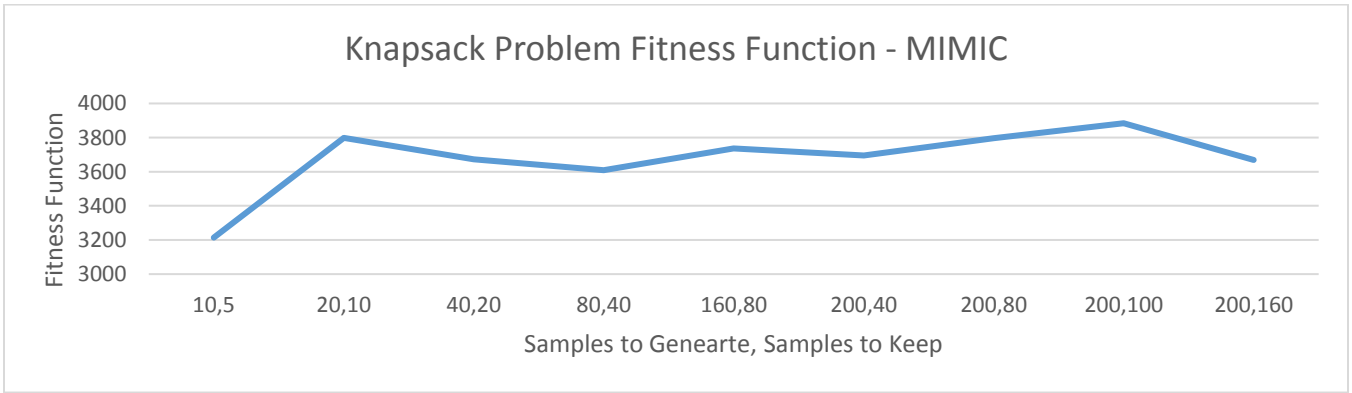
Knapsack Problem

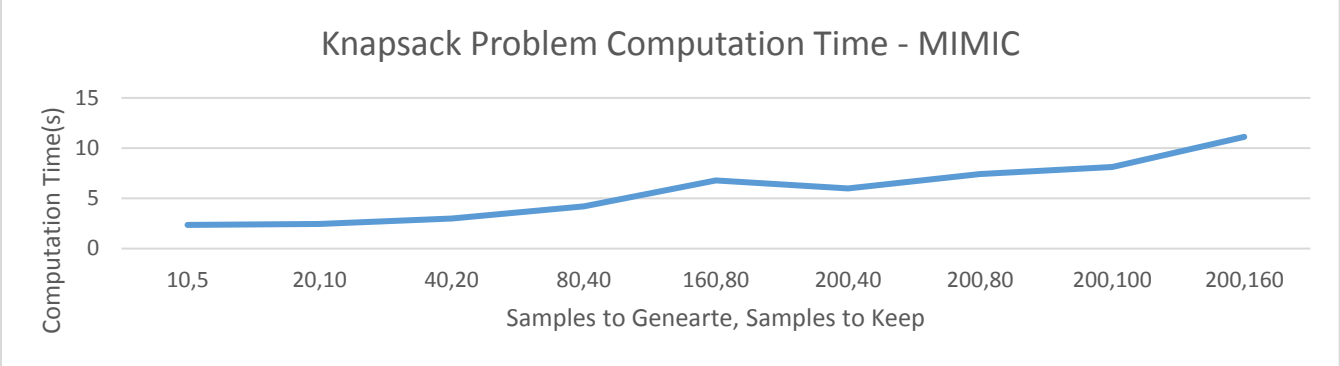
Knapsack is an optimization problem. Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.
We use the optimization algorithm to figure out the optimum strategy. The problem is run in ABAGAIL. We have 40 unique items and 4 copies of each and the maximum volume of the sack is 3200. Maximum weight is 50 and maximum volume is 50 per item.





As we can see, **MIMIC** outperforms other algorithms for all iterations. MIMC takes a lot more time than other algorithms. Furthermore, we tune different parameters for MIMIC. All experiments are run in the iteration of 5000.





From the figures we can see MIMIC generally performs better and takes more time with larger samples, the fitness function reaches peak when samples to generate is 200 and samples to keep is 100. Noted that when samples to generate is 200, the performance for MIMIC changes at samples to keep. If samples to keep is small or large, MIMIC does not perform well. MIMIC reaches the optimum value when to keep is about half, at 100. This suggests while samples to generate can help increase performance, usually the larger the better, we also need to pick appropriate samples to keep values, neither too small nor too large to have the optimal performance.

Conclusion

It has been shown that different optimization problems can served best with different optimization algorithms. In neural networks, however, back propagation using gradient decent still outperforms any optimization algorithms. Different algorithms can be tuned by parameters to have better performance.

Algorithm	Parameters	Computation Time	Comments
Random Hill Climbing	0	Fast	Good for time sensitive, simple structure, low cost problems
Simulated Annealing	2	Fast	Good for time sensitive, simple structure low cost problems
Genetic Algorithms	3	Medium	Good for complex structure problems
MIMIC	2	Slow	Good for complex structure problems

Eventually, it depends on the problem to choose different algorithms. If computation time is important, RHC and SA is preferable, the problem relies on the structure and evaluation functions are expensive to compute, GA and MIMIC can be options.

References

1. Phishing website datasets. https://archive.ics.uci.edu/ml/datasets/Phishing+Websites
2. ABAGAIL. https://github.com/pushkar/ABAGAIL
3. Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.