Randomized Optimization

Xujian Liang GTID:903421366

Abstract:

In this paper, I explore the different randomized optimization algorithms including 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 the part two, three different optimization problems are analyzed to evaluate all four optimization algorithms.

Optimization Algorithms

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

Randomized Hill Climbing

It searches for optima by moving towards neighbors that have more optimum values until it reaches the global optima. Randomly starting its position in order to avoid getting stuck at local optimum. The random start can help us to have another chance to get the global optimum when we get stuck.

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, 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.

Genetic Algorithms(GA)

Genetic algorithm tries to find optimum solutions 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 genetic algorithm is that it does not scale with problem complexity. When the number of elements becomes larger, the search space increases exponentially.

MIMIC

Mutual Information Maximizing Input Clustering 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.

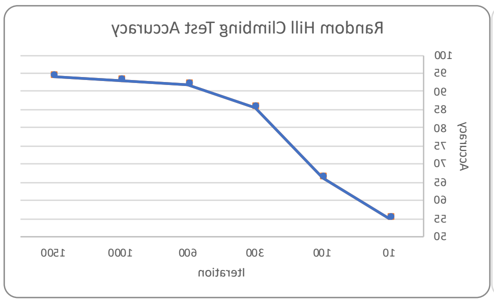
PART1: Random Optimization for Neural Network.

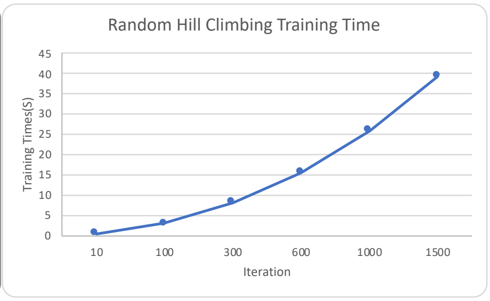
In this section, I applied three different random optimization algorithms: randomized hill climbing, simulated annealing and genetic algorithms to find the optimal weights of the neural networks for phishing website dataset.

Dataset and Methodology

The phishing website dataset has around 10,000 instances, 30 attributes and 2 classes. The dataset will be split into 70-30 as usual. 70 is for training set and the remaining is for testing set. All randomized algorithms are run with one hidden layer with 16 perceptron unites. All tests are run in ABAGAIL. We tune the parameters for each algorithm with iterations = 10, 100, 300, 600, 1000, 1500 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.

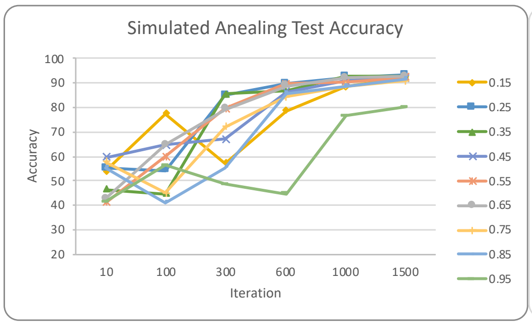
Randomized Hill Climbing

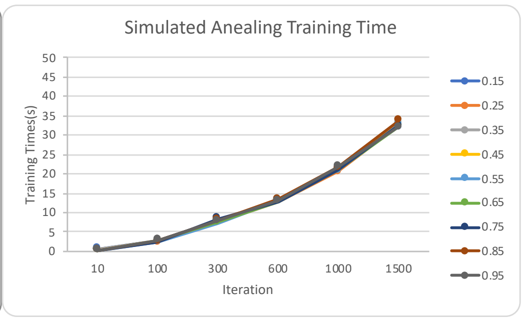




From the results, RHC converges as the iteration increases, but still make more time with more iterations. The accuracy stays at around 93%.

Simulated Annealing

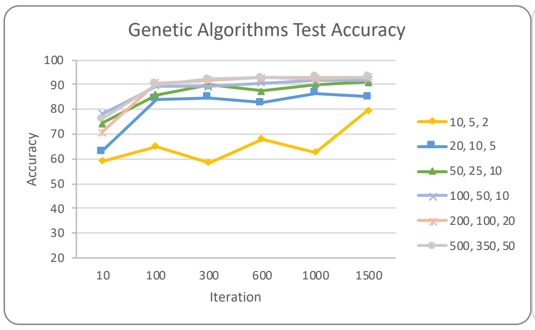


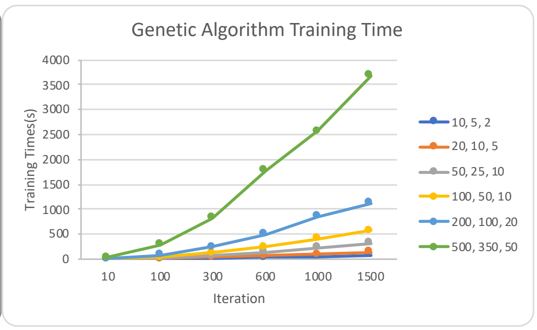


In simulated annealing, I got low accuracy where I use few iterations. The accuracy increases with the number of iteration. It also shows that at iteration 1500, different cooling exponent performances are close except 0.95, but it took lots of time to train.

Cooling Exponent	Test Accuracy	Training Time	Test Time
0.15	92.318	32.012	0.006
0.25	93.368	32.369	0.007
0.35	92.515	33.129	0.008
0.45	91.924	32.390	0.008
0.55	92.383	32.647	0.010
0.65	92.515	32.144	0.007
0.75	90.808	32.327	0.006
0.85	91.858	33.684	0.007
0.95	80.302	32.107	0.012

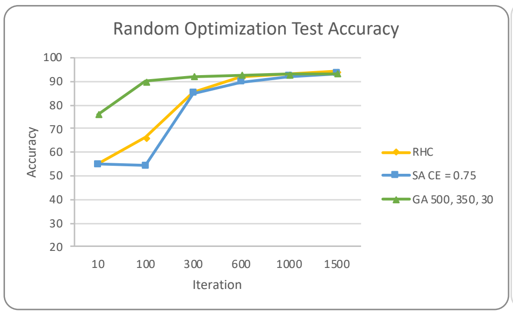
Genetic Algorithms

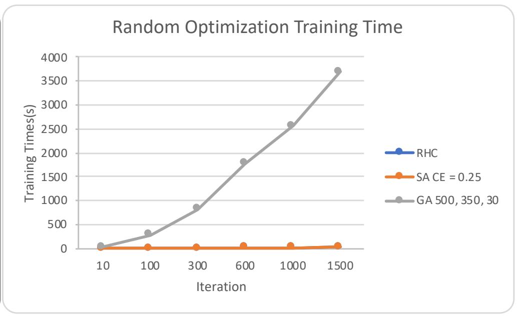


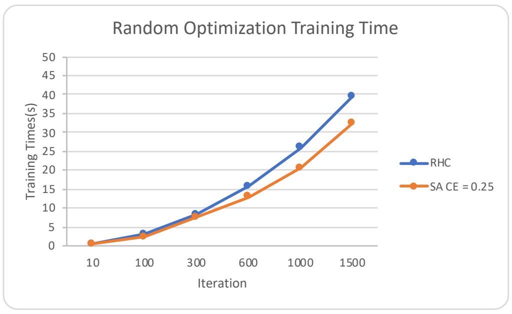


From the figures, we can see larger population size tends to have better accuracy but it will take lots of time. When we get 300 iterations, the accuracy has pass 90% with over 50 population.

Algorithms Comparison







Comparing these three algorithms, RHC has achieved best performance when iterations go to 1500 with 94.091% accuracy. Furthermore, RHC takes much less training time. It is computationally prohibitive to run GA at large population size, it took me couple of hours to run it.

I also run backprop for the neural network in weka, it tooks about 30.892 seconds to build the model and accuracy is 95.231.

PART2: Optimization Problems Analysis.

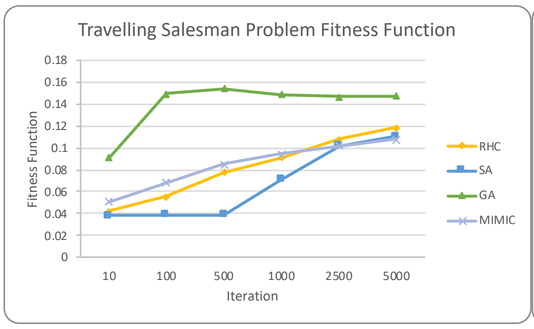
In this part, three optimization problems are analyzed four different algorithms mentioned above. The problems are traveling salesman, continuous peaks and knapsack.

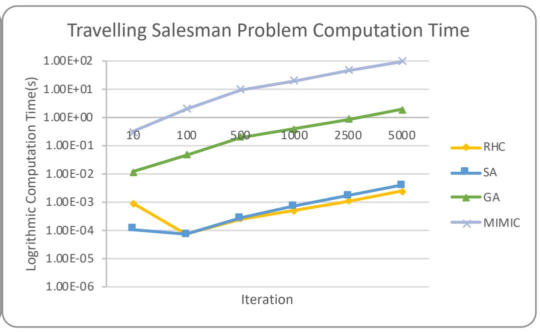
Methodology

Each optimization problem is run with four different optimization algorithms by an iteration count of 10, 100, 500, 1000, 2500, 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. Then pick the best algorithm for each problem based on fitness function output and computation time.

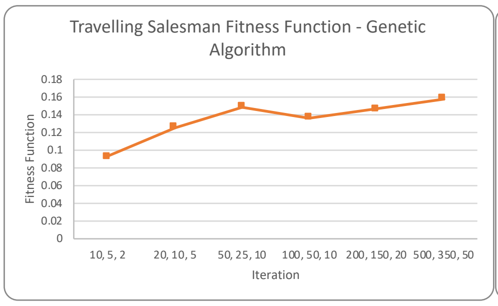
Travelling Salesman Problem

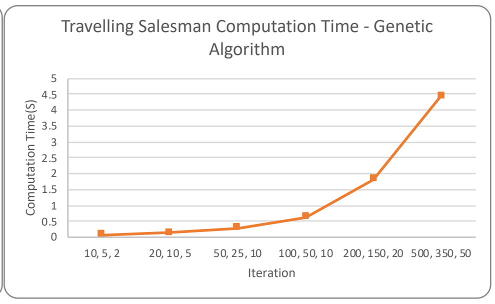
Travelling salesman problem is a classic NP-hard problem. The objective is to find the shortest round trip between different points while visiting each point only once. It is a problem with practical applications in transportation and travel planning. We set N = 50.





From the above figures, 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. I also examine the performance of genetic algorithms by comparing the different parameters. The population size is the size of population survive from the iteration. Mate is the crossover number representing the mating between the individuals. Mutate is the mutation that causes random modifications.

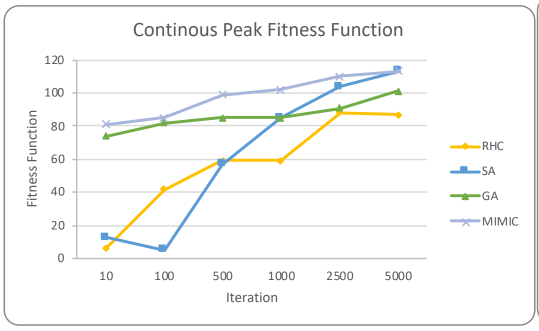


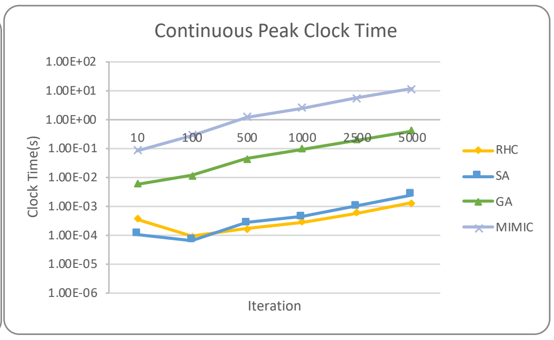


The figure shows that the best configuration is when population size is 100, mate is 150 and mutate is 10. The computation time increases as the population size increases.

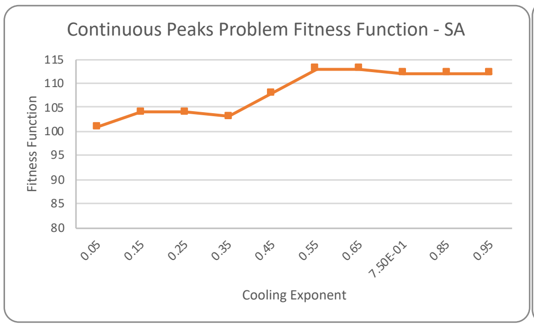
Continuous Peaks Problem

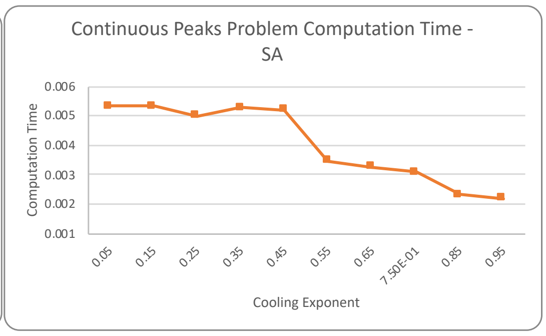
Continuous peaks problem is an optimization problem that tries to find out the highest peak. In ABAGAIL, we set N = 60 and T = 6.





As we can see from above figures, MIMIC and simulated annealing has close performance at 2500 iterations and 5000. But, 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.



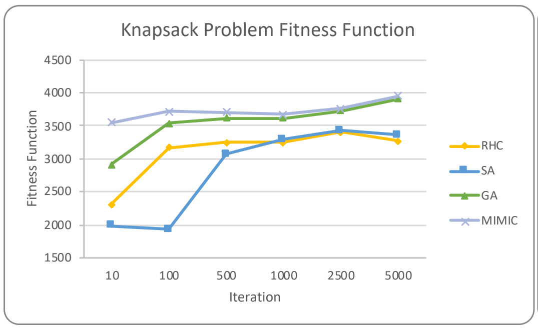


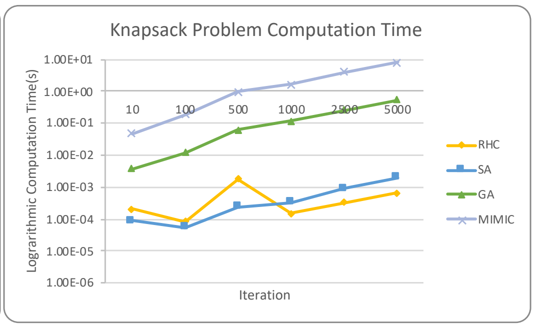
From the figures, we can see when cooling exponent = 0.55, SA performs the best with highest fitness function value. Meanwhile, we also find out the computation time decreases as cooling exponent increases. It may happen when larger coding exponent means the temperature drops more slowly, which makes 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

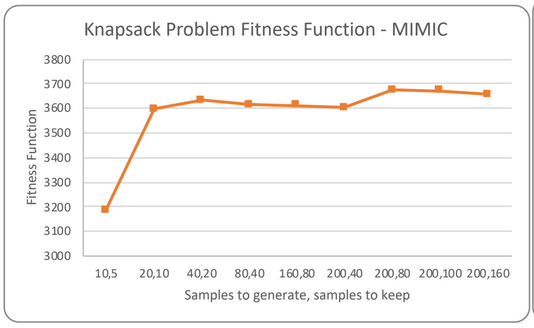
Knapsack is an optimization problem. Give a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the value need to get as large as possible.

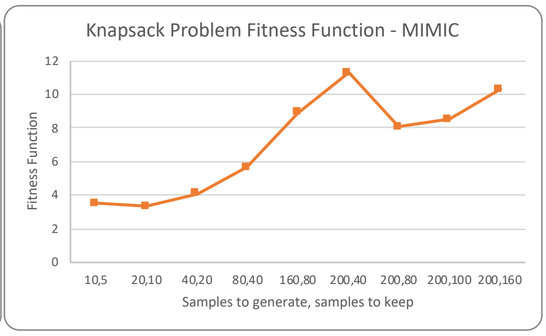
We use the optimization algorithm to figure out the optima. The problem is run in ABAGAIL. WE have 40 unique items and 4 copies of each and the maximum volume of the sack is 3200. Maximum weight is 50 and maximum volume is 50 per item.





As we can see, MIMIC outperform other algorithms for all iterations. MIMIC 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 that 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 80. Noted that when samples to generate is 200, the performance for mimic do not change much at samples to keep. MIMIC reaches the optimum value when to keep is about half at 80 or 1000. 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.

Conclusion

Based on above three problems, we can find out that different optimization problems can served best with different optimization algorithms. In neural networks, back propagagtion using gradient decent still outperforms any optimization algorithms.

Algorithm	Parameters	Computation Time	Comments
RHC	0	Fast	Good for time sensitive, simple structure, low cost problems
SA	2	Fast	Good for time sensitive, simple structure, low cost problems
GA	3	Medium	Good for complex structure

			problems
MIMIC	2	Slow	Good for complex problems

RHC and SA can perform very well when computation time is very important. The problem relies on the structure and evaluation functions are expensive to compute, GA and MIMIC can be options.

Reference

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.