Dantong Zhu

CS 7641

Project 2

Randomized Optimization Writeup

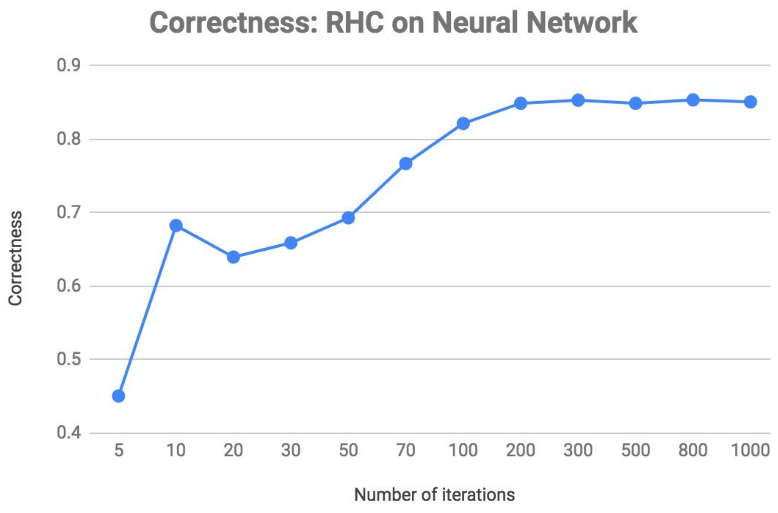
Section 1: Three randomized algorithms on neural network

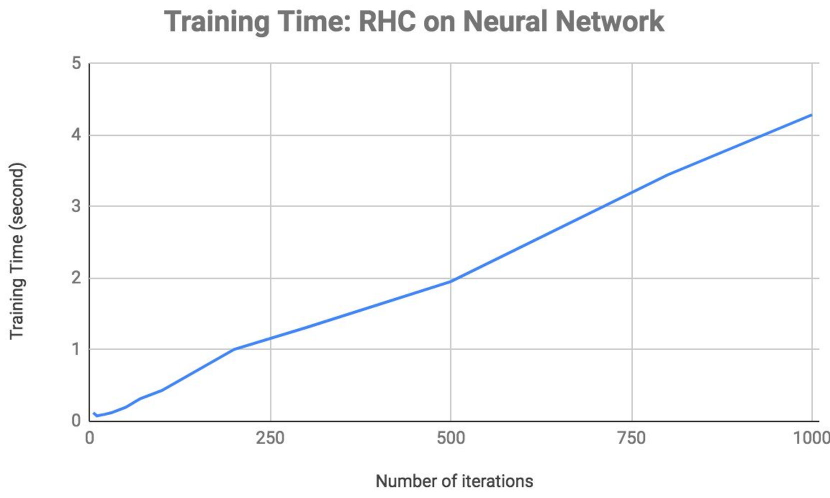
Recap and introduction to the problem:

Our dataset is the same from Assignment 1 about forest fires in the northeast region of Portugal. There are 517 data points total. There are 12 attributes, including location, time, weather, and etc (see README.txt for detail). The original label for each data point is the buring area of each fire, or non-fire in the case buring area = 0. The goal is to predict the burning area of a some fire given the 12 attributes. In our classification problem, we label a data point as 0 if the burning area is strictly smaller than 5, and as 1 otherwise. In Assignment 1, we used backpropagation to determine that the performance of a neural network on our dataset starts to converge when there are 10 hidden layers.

In the experiments this time, we set the neural network for our data to have 10 hidden layers. We are going use Randomized Hill Climbing (RHC), Simulated Annealing (SA), and Genetic Algorithm (GA) to find good weights. All algorithms we used in the experiments are modified from the ABAGAIL package. For every experiment, we run 5 trials for the same parameters and took the average for the performance and running time. We first run each one of the three randomized optimization algorithms with comparison with respect to varying parameters. We then pick the best parameters for each algorithm and compare the three algorithms.

1.1 Randomized Hill Climbing (RHC)

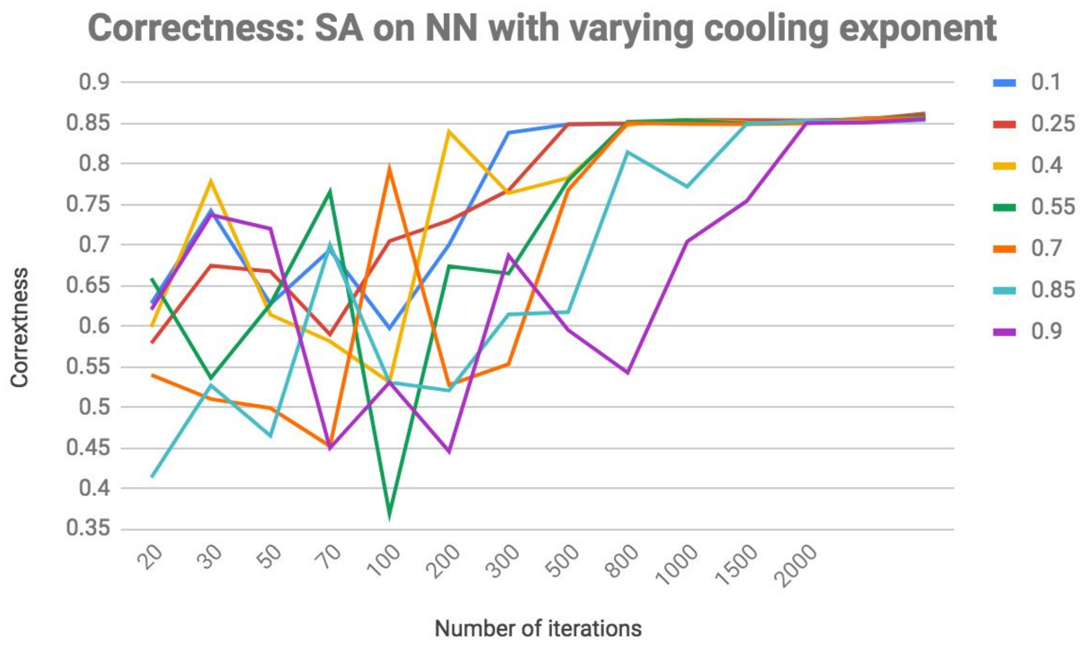




We started with the Randomized Hill Climbing algorithm. We increase the number iterations running and recorded the correctness of classification. The observation and analysis of this experiment are as follows:

(1) As the number of iterations increase, the correctness of the algorithm increases in general, except at the point when there are 10 iterations. This agrees with the nature of RHC - that is, when the number of iterations is small, the algorithm gets stuck in local maxima more easily; whereas when the algorithm iterates for more times, there is a higher chance that the randomly generated starting point in an iteration falls near a better local maximum, which eventually gives a better classification of the data.
(2) When the number of iterations reaches around 100, the correctness of the algorithm starts to converge around 0.83 with the high at 0.853 when running 300 iterations. This shows that to get the best performance with RHC, it suffices to stop when around 100-300 iterations.
(3) The (local) peak at 10 iterations is because there is a higher probability that RHC gets stuck in bad local maxima when there are smaller number of iterations; whereas when the number of iterations is big enough, we obtain a smoother curve and a more stabilized performance, because the chance of getting stucking in a bad local maxima is low.
(4) The running time grows in a approximately linear speed as the number of iterations grows. This again agrees with the nature of RHC, because the time spent on each iteration is more or less the same.

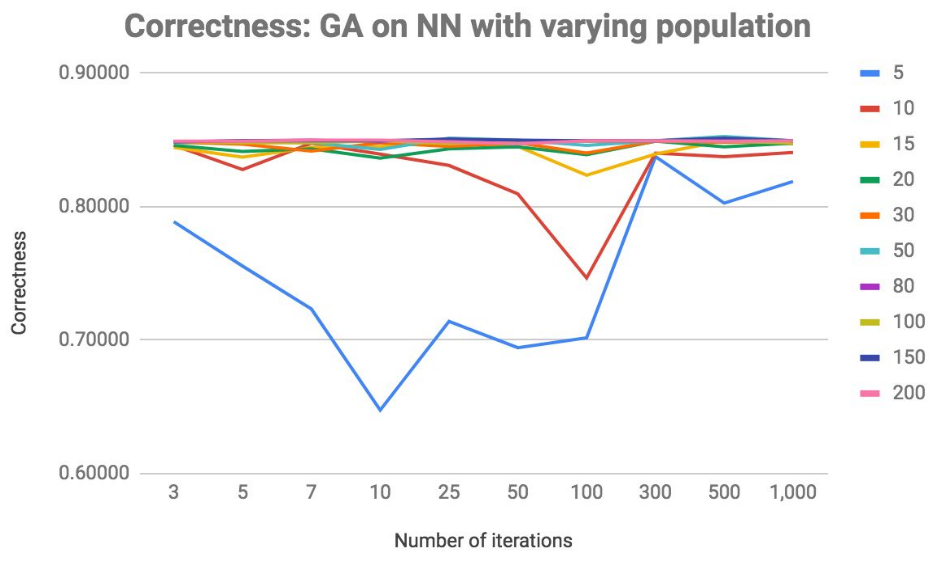
1.2 Simulated Annealing (SA)



We set the starting temperature to be 10^11 fixed and varied the cooling exponent from 0.1 to 0.9. The bigger the cooling exponent, the slower the temperature is cooled down. We obtain the following observation and analysis:

(1) For every cooling exponent, i.e. each curve in the graph, the correctness eventually converges around 0.85, agreeing with the convergent point in RHC.
(2) Before every curve starts to converge, the performance (correctness) shows a good amount of up and down, as shown on the left half of the graph. This agrees with the set-up of the Boltzmann function: when the temperature is higher, the algorithm treats higher and lower values of the performance function more similarly.
(3) The convergent point gets delayed as the cooling exponent increases, as we can see the two curves representing cooling exponents 0.85 and 0.9 start to converge around 1500 iterations, whereas the blue curve for cooling exponent 0.1 starts to converge around 300 iterations. This again agrees with the set-up of the Boltzmann function: as the cooling exponent is small, the temperature gets cooled down faster, meaning that big values and small values of the performance function get distinguished faster, and therefore the algorithm starts to converge earlier.
(4) Recall that if the temperature cools down too fast, we might get stuck in local maxima more easily. And for our dataset, the fact that the algorithm can converge with a relatively low cooling exponent 0.1 also implies that the values of local maxima are not too far away from the global maximum.

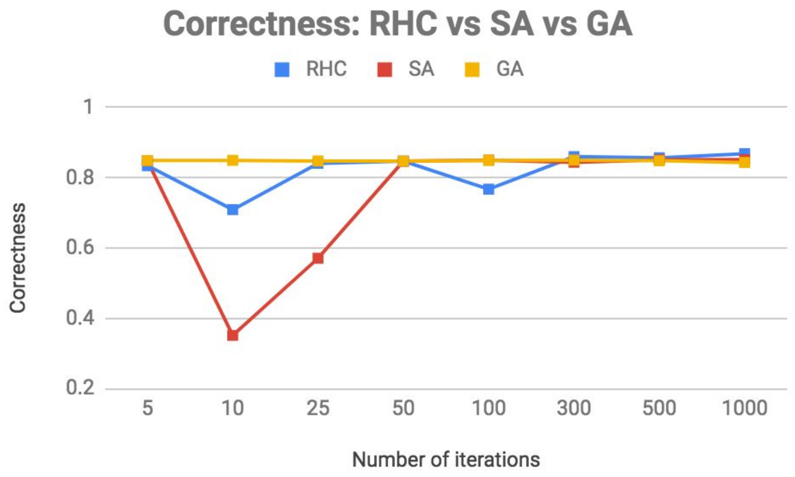
1.3 Genetic Algorithm (GA)

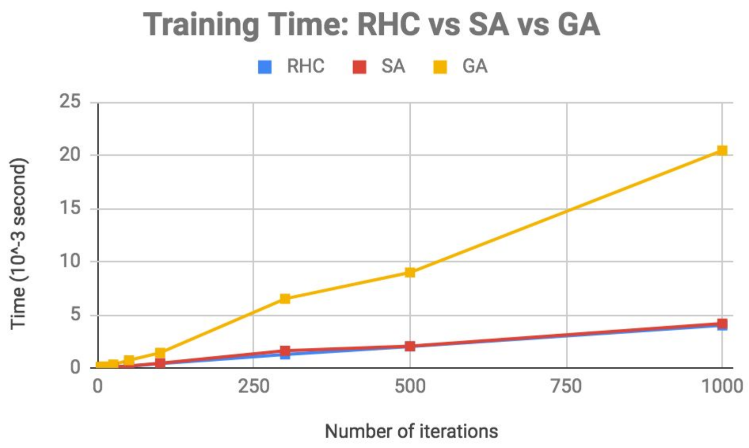


In the first part of the experiment for Genetic Algorithm, we vary the population from 5 to 200 and keep the change to make and mutate as ⅕. From result plotted in the graph above, we have the following observation and analysis:

(1) when the population is on the small side, the algorithm in general performs poorly. This is because we miss the potential good “parents” to generate good children. However, as we set the population to 20 or even higher, the correctness performance can stay high around 0.85, agree with results from RHC and SA above.
(2) When the population is not too small, the correctness converges at a pretty early stage, in terms of number of iterations. This is probably implying that the structure of our dataset is in general pretty nice that we do not really need to combine “good” parameters for too too many times in order to obtain the optimal solution.

1.4 Comparison among the 3 algorithms



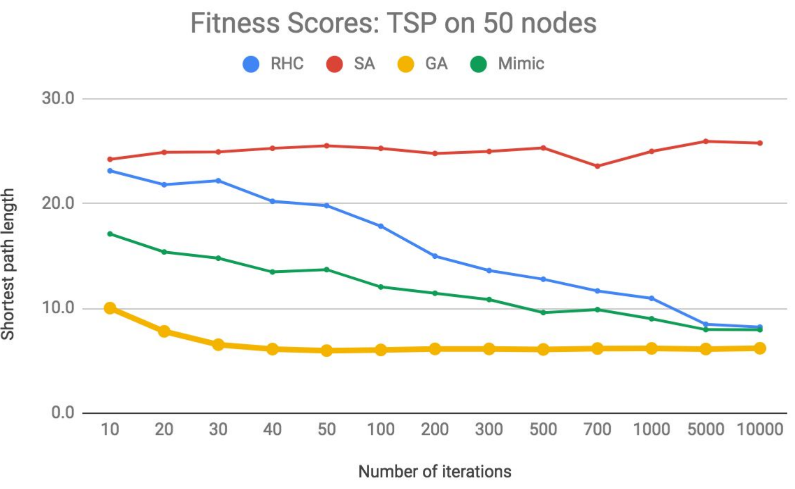


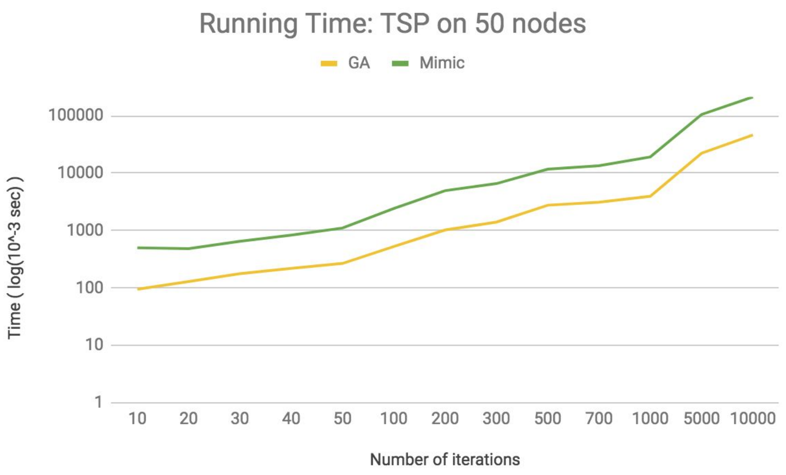
Now, we picked the best parameters for each one of the three algorithms from the previous results. Precisely, for SA we choose starting temperature to be 10^11 with cooling exponent 0.1; and for GA we choose population size 50 with chance to mate and mutate both ⅕. In terms of correctness performance, both RHC and GA are better than SA, and both RHC and GA converge around 300 iterations. And in terms of training time, each algorithm’s training time increases linearly as the number of iterations increase. However, the linear factor for GA is way bigger than for RHC and SA. Therefore, we conclude that for our dataset, RHC is the best algorithm out of the three for finding weights for the neural network.

Section 2: Three discrete optimization problems

2.1 Traveling Salesman Problem (TSP)

We use the Traveling Salesman Problem (TSP) to show advantages of Genetic Algorithm. TSP is the very classic NP-hard problem that, given a weighted graph, finds the shortest path that visits every node and comes back to the origin eventually.





In the experiment for TSP, we start with a graph on 50 nodes with randomly generated weights on the edges. For each graph generated, we run all four algorithms. We plotted the results in the graphs above and present the following observation and analysis:

(1) First note TSP is a minimization problem and therefore we are looking for curve/points with lower y-values.

(2) Among the four algorithms, GA gives the best performance, and furthermore that it converges the earliest around only 30 interactions. This is probably due to that mate and mutation captures great combinatorial properties of graphs. For example, two strings of parameters to mate probably represents well how two paths can be combined in the graph to obtain a better path.

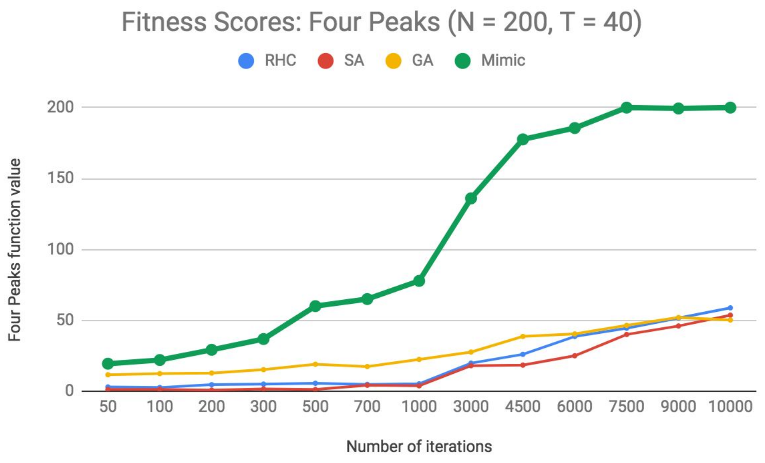
(3) In terms of running time, both RHC and SA can finish 10000 iterations in less than 0.006 seconds. GA spent 0.22 seconds to finish 40 iterations when it first starts to converge whereas MIMIC spent 105 seconds to finish 5000 iterations. Thus even with longer running time, given the performance of GA and its early convergence point, we conclude GA is the best algorithm our of the four for TSP.

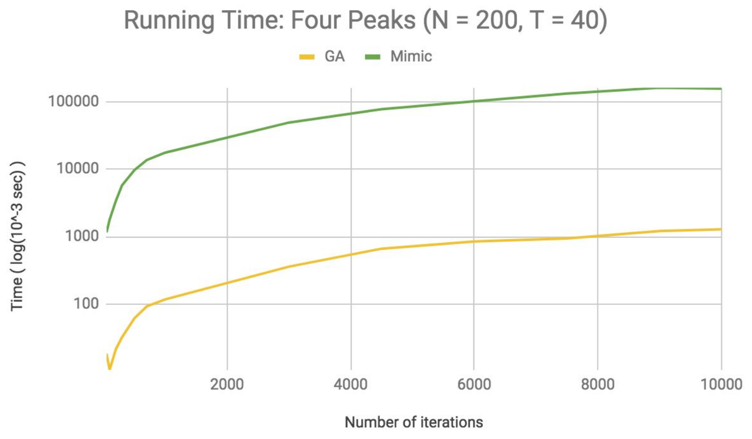
2.2 Four Peaks

We use the Four Peaks problem to show advantages of MIMIC. Given a 0-1 vector/string *X* of length *N* and an integer *T<=N*, the Four Peaks function *f* of *X* is defined as follows:

1. Let *max0* be the number of consecutive 0's at the end of *X*
2. Let *max1* be the number of consecutive 1's at the beginning of *X*
3. Let *r = N* is both *max0* and *max1* are strictly greater than *T* and otherwise *r = 0*
4. *f(X) = max{max0, max1} + r*

In our experiment, we set N = 200 and T = 40. Note there are two global maxima with values 2*N - T - 1 = 359 when the string starts with 41 ones followed by 159 zeros or starts with 159 ones followed by 41 zeros, and also that the function has a huge number of local maxima with value 200 when X has at least 41 starting ones and at least 41 ending zeros. To be exactly there 2^119 points in the parameter space are loca maxima with the total size of the parameter space being 2^200.





We run the four randomized approximation algorithms on the Four Peaks function. We plotted the result in the graphs above and present the following observation and analysis:

(1) MIMIC is the only algorithm out of the four that converges to some local maxima, and it converges when the number of iterations is around 7500. This shows that MIMIC is good at solving optimization problems on a discrete space where there are a huge number of local maxima.

(2) Recall the in the MIMIC algorithm, we optimize the dependency tree in which we assume every bit is dependent on exactly one other bit. Because of this

assumption, MIMIC theoretically should perform better when each parameter is dependent on only a few other parameters. Here in the Four Peaks problem, since we only care about consecutive zeros and ones, each bit in the string is very dependent on the neighboring two or one bits and pretty independent from all other bits. As the nature of the Four Peaks problem matches the assumption in MIMIC, we empirically proved that MIMIC does work the best on the Four Peaks problem.

(3) Even though MIMIC performs the best in terms of number of iterations, it is also the most slow algorithm among the four. RHC and SA need at most 0.005 second to finish 10000 iterations, whereas the GA needs 1.28 second and MIMIC needs needs 156 seconds. As shown in the graph on the right, running time for both GA and MIMIC grows exponentially as the number of iterations increases.

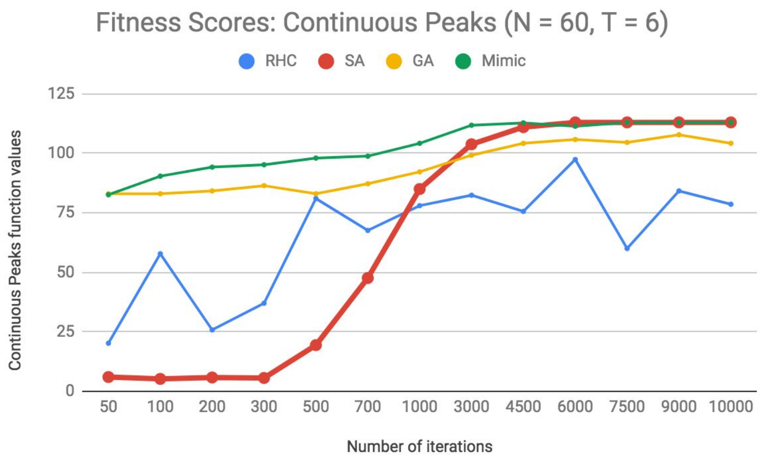
(4) Unfortunately, none of the four algorithms is able to find a global maximum or any value greater than 200 within 10000 iterations.

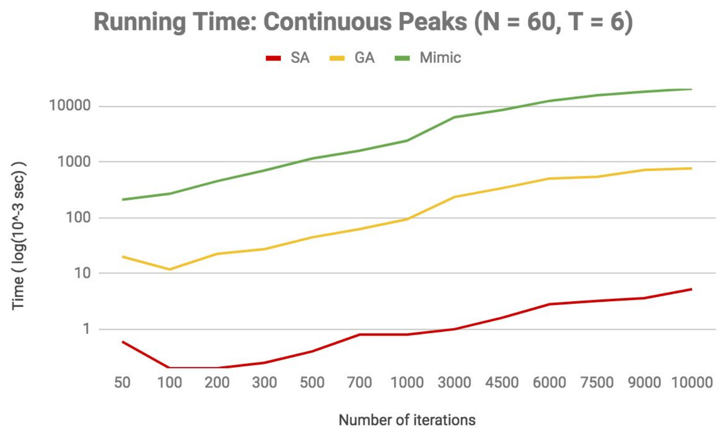
2.3 Continuous Peaks

We use the Continuous Peaks problem to show advantage of Simulated Annealing (SA). The Continuous Peaks function is similar to the Four Peaks function. It is defined precisely as follows. Given a 0-1 vector/string *X* of length *N* and an integer *T<=N*,

1. Let *max0* be length of the longest all-zero substring of X
2. Let *max1* be length of the longest all-one substring of X
3. Let *r* = *N* is both *max0* and *max1* are strictly greater than *T* and otherwise *r = 0*
4. *f(X) = max{max0, max1} + r*

In the Continuous Peaks function, similarly to Four Peaks there are exactly two global maxima with value 2*N-T-1 when the string starts with *(T+1)* ones followed by *(N-T-1)* zeros or starts with *(N-T-1)* ones followed by *(T+1)* zeros. In the meanwhile, we are getting lot more points with value at least *N+T+1* now, which happens as long as there exist both an all-one substring of length at least *T+1* and an all-zero substring of length at least *T+1*.





In the experiment for Continuous Peaks, we set N = 60 and T = 6. Given this information, our global maxima have value 113. We plotted the result in the graphs above and present observations and analysis in the following:

(1) So much different from the results for Four Peaks, both SA and MIMIC converge to global maximum at 113 within 10000 iterations! In addition, GA achieves its highest at 108 which is also pretty close to the global maximum.

(2) Similarly to Four Peaks, MIMIC converges to a global maximum and it actually converges at a pretty early stage. The reason why MIMIC works well is similar to what was presented in the analysis for Four Peaks: that is, every bit in the string in Continuous Peaks is still very dependent on its only two or one neighboring bits and independent from all other bits, which matches the assumption of MIMIC.

(3) However, SA starts with the lowest fitness score when the number of iterations is smaller than 700 and reaches a global maximum around 4500 iterations. This is because, in the parameter space, points having both consecutive all-ones and consecutive all-zeros have values a lot more continuous than the case in Four Peaks. This continuity allows Simulated Annealing to be able to always have a good chance to find points of higher value and improve the fitness score.

(4) In terms of running time, SA finishes running 6000 iterations in 0.0028 second when it first reaches a global maximum, whereas MIMIC finishes running 3000 iterations in 6.315 seconds when it first reaches a global maximum. With the same best performance but much less time, we conclude that SA is better for Continuous Peaks than MIMIC.