**1. Different optimization methods for ANN weights:**

I chose to use the titanic data set for this optimization test. That’s because the small size and simplicity of this data set, so I can test on very large neural networks and more iterations. In order to compare the performance of different optimization methods, I use the same 3:1 split of training and testing set for all methods. Also based past knowledge of this data set, the ‘optimal’ structure of hidden layer is one layer with 5 units, and activation function is logistic sigmoid. These were kept the same for all the tests.

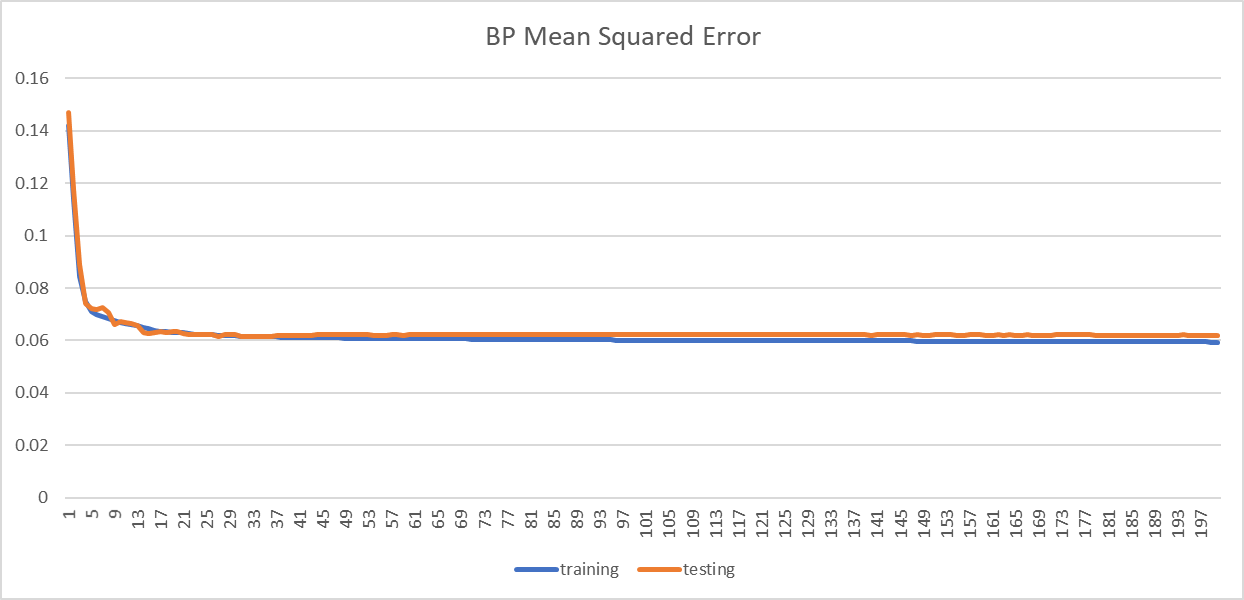
First, I implemented the **back propagation** as a base line method. It obvious that back propagation converges after several hundred iterations. The following results are after 2000 iterations, the MSE decreased rather smoothly for every run. But the result prediction accuracy differs a lot in different runs, below are two final accuracy outputs of two runs:

iteration MSE\_trg MSE\_tst acc\_trg acc\_tst elapsed

1990 0.05938 0.06198 0.82489 0.8375 2.329107063

1990 0.07120 0.10210 0.78659 0.6062 2.407784919

Here is a MSE plot for the best performing run:

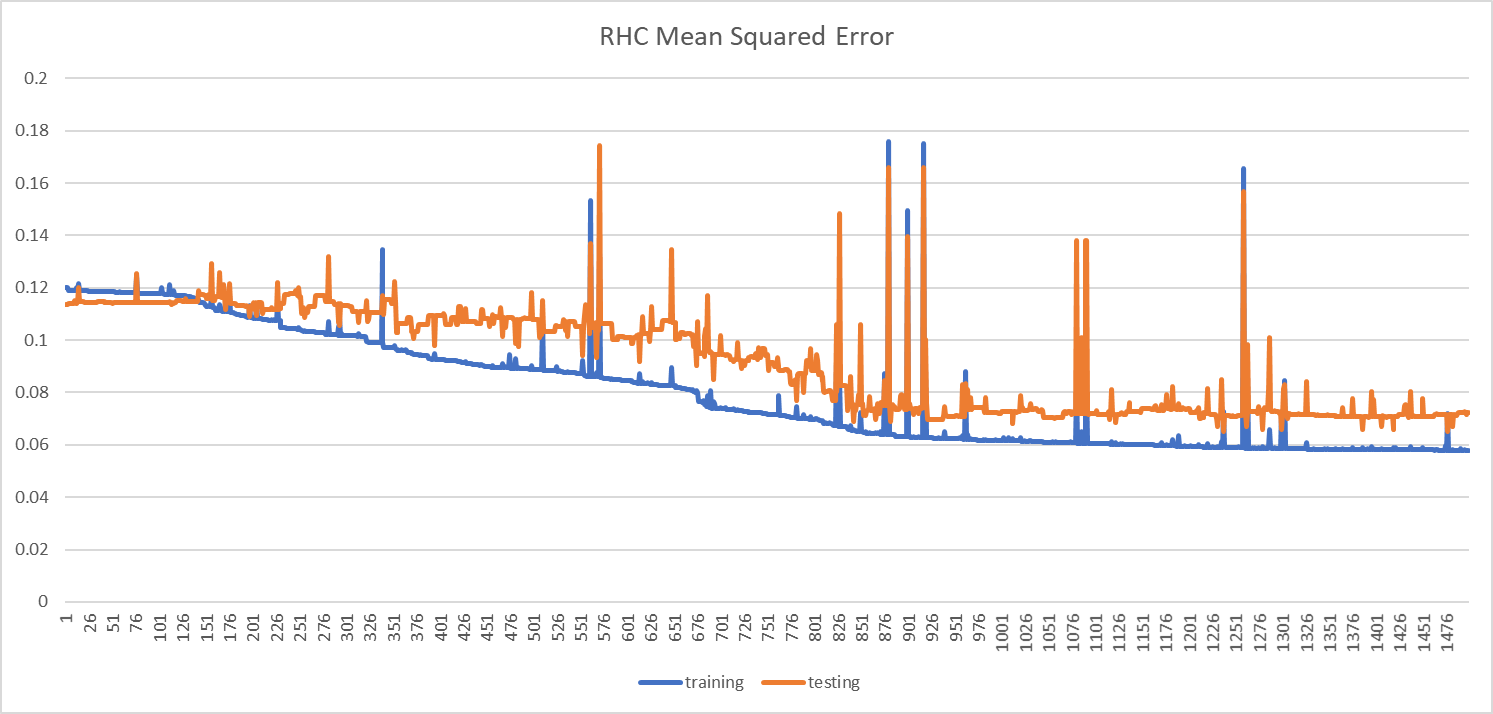


The huge difference of testing accuracy should be a result of different starting weights. Because those weights were randomly selected in back propagation and don’t reset through out the process. The converged testing accuracy varies a lot, which means back propagation can’t provide a reliable output after one run only. But the optimal accuracy of 0.83 gave me a performance standard.

For **randomized hill climbing**, the same NN structure were used. But RHC converged much slower and need more than 10000 iterations to converge. There are some obvious peaks in the MSE plot, which seems to be a strange thing in optimization plots. At first, I thought those were caused by random restart of hill climbing, but I still can’t understand why a higher MSE solution was reported as an optimal during the process. It’s also very interesting that quickly after each of those peaks, the overall fitness return to previous levels. That seems to show most random restarts won’t find a better solution, but instead will return to previous optimal result. After all, the overall trend of error decreasing and accuracy improving is correct. The final output is:

iteration MSE\_trg MSE\_tst acc\_trg acc\_tst elapsed

14990 0.05796 0.07226 0.84541 0.8 12.57300482



We can see that randomized hill climb didn’t give a better testing accuracy. But one big advantage of RHC is that it doesn’t rely on initial random weights that much. I tried running 15000 iterations for several times. All result accuracies were around 0.8.

For **simulated annealing**, I set an initial temperature of 1e10 to see the effects of cooling factor first. After trying different CE of 0.15, 0.35, 0.55, 0.7 and 0.95. A CE of 0.95 can converge within tens of thousands of iterations and the final testing accuracy is the highest. The last iteration outputs are:

CE MSE\_trg MSE\_tst acc\_trg acc\_tst elapsed

0.15 0.05870 0.05939 0.85088 0.83125 7.13342

0.35 0.06654 0.09766 0.81121 0.75 8.12596

0.55 0.06465 0.07450 0.83857 0.8125 7.21438

0.7 0.06885 0.09274 0.82763 0.75 7.43623

0.95 0.06194 0.06667 0.84952 0.84375 36.0838

This can be understood as the temperature cools too fast, the searching of weights was constrained within a local optimal. As a result, no matter how many iterations passed, the testing accuracy can’t improve given too low a temperature. Then I compared different starting temperature of 1e8, 1e10 and 1e12. The 3 final outputs are:

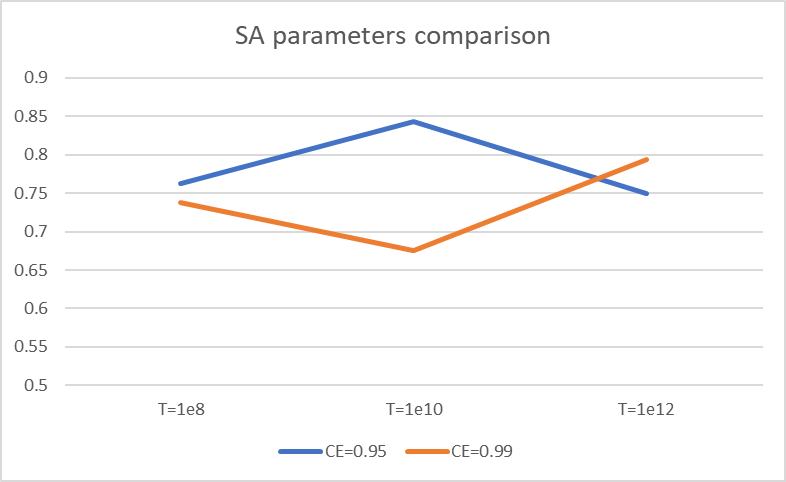
T MSE\_trg MSE\_tst acc\_trg acc\_tst elapsed

1e8 0.06677 0.09016 0.84268 0.7625 36.4665

1e10 0.06194 0.06667 0.84952 0.84375 36.0838

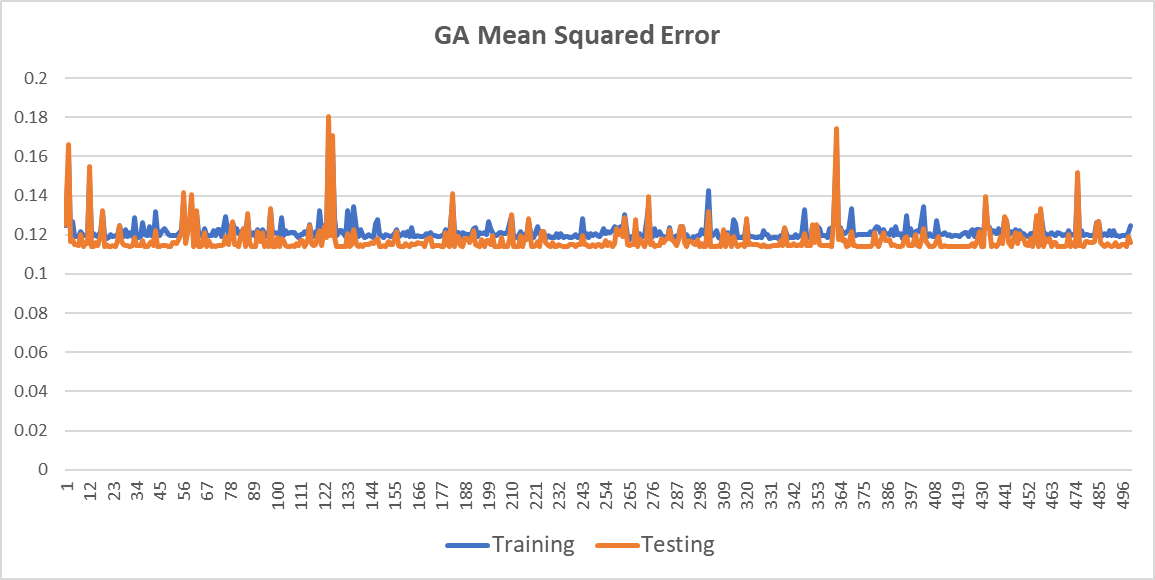
1e12 0.07544 0.08600 0.80574 0.75 44.9766

If I compare the different combination of T and CE. The testing accuracy plot looks like:



The initial temperature of 1e10 provides a much higher accuracy score. A higher temperature of 1e12 combine with larger CE of 0.99 can improve on accuracy. But the time it takes to converge is much longer as maybe several minutes. And it’s not practical to apply a cooling factor of 0.99 for larger data set trainings. In all, the current combination of T=1e10, CE=0.95 seems to have an optimal performance.

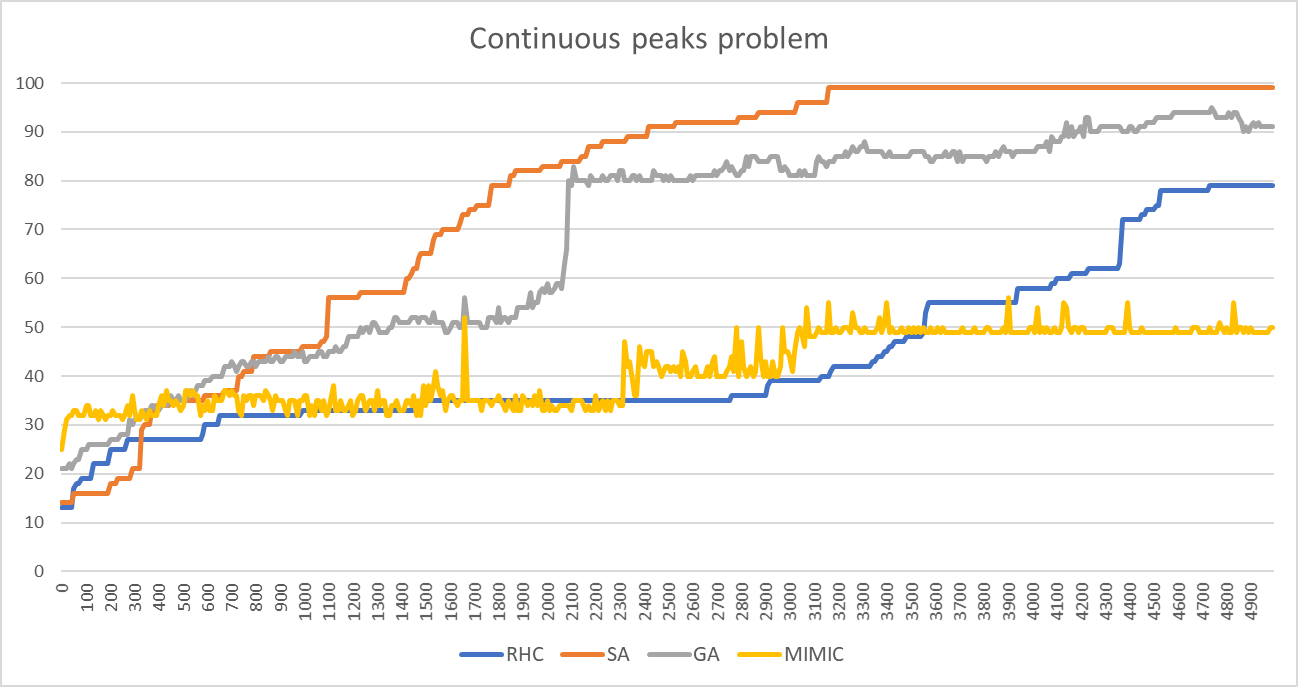
For **genetic algorithm**, one notable fact is that GA is much slower in each iteration of crossover. The overall run time of a 5000 iteration run can take 107 seconds, while the previous algorithms can finish similar iterations in several seconds. This may be due to the parameters of mating and mutation frequency, but even on my smallest setting of 20 mating and 20 mutations, it still takes more than 61 seconds. When finished several runs with different [population, mate, mutation] settings, the result testing accuracy never improved during the process. A MSE plots example:



This is true even when I did 50000 iterations. So I thought there must be something wrong in the mating selection and mutation process. After checking the StandardGeneticAlgorithm function in ABAGAIL, there seems nothing more I can change except the above three parameters.

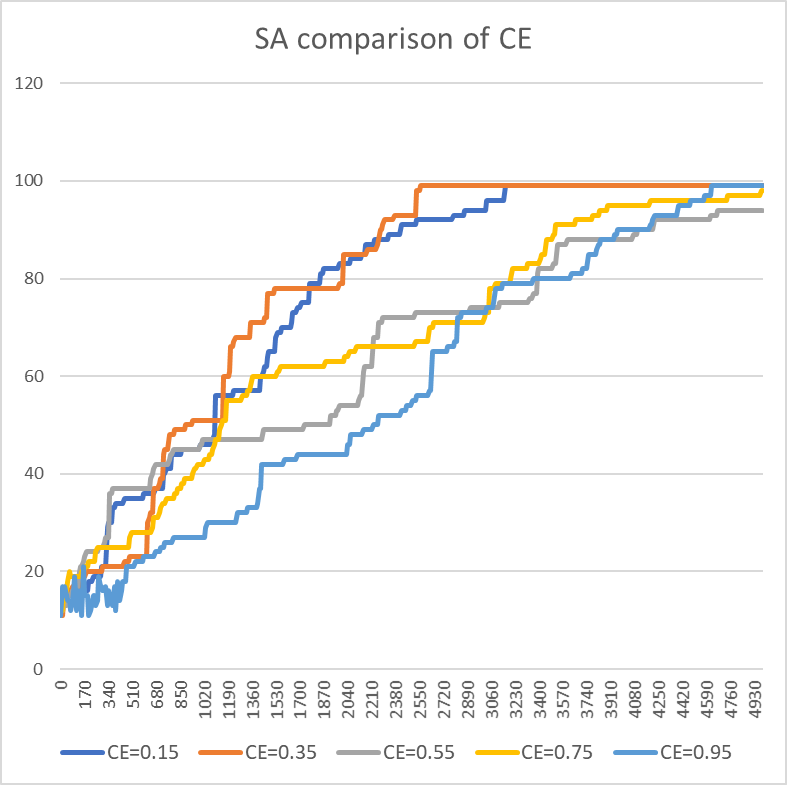
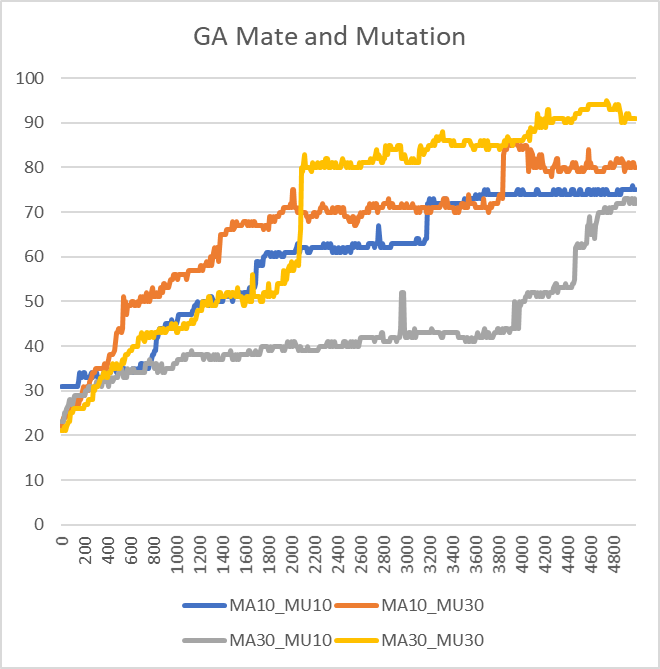
**2. Continuous Peaks (SA)**

This problem contains many local optima in a one-dimension number array. It’s one of the simplest optimization algorithms, but it can highlight some important aspects of different randomized processes. I used the continuous peak evaluation function in AGABAIL and copied a similar problem set up as in the jython sample script of four peaks problem. The maximum peak height was set to 100 and 5000 iterations were done. I did parameter search for SA, GA and MIMIC. If I just pick the best performing result of the four methods, the result is:



If we compare the performance of these four algorithms, simulated annealing clearly outperforms the three other methods. No matter what cooling factors I chose, SA can always converge before 5000 iterations done, while other algorithms all can’t find the global optimal within 5000 iterations. The reason SA have much better result is that it combines random search and local optimal naturally, this process can work best for 1D array searches. And other algorithms can have better implications for more complex model searches. The curve shape also shows that RHC improves score rather slowly comparing to GA and SA. So I can conclude from here and previous test that we need to use much more iterations in RHC run. This seems to be okay for me since we don’t need to tweak parameters for RHC.

Here are the detailed comparison of different parameters of SA and GA:

For simulated annealing, the initial temperature was the same of 1e11. But different cooling temperatures have a huge impact on the performance of this algorithm. A CE of 0.35 result in the fastest convergence at around 2500 iterations. Comparing with the previous analysis, a 0.35 CE is very small. This seems to show that my set up of continuous peaks is a very simple one. The algorithms don’t need to randomly explore the number array very much. Some straightforward local optimization is enough.

For genetic algorithm. I tested different mate and mutation settings. The larger parameters of 30 mates and 30 mutations seems to work best. This result is expected as larger mates and mutations will require more computation. But one interesting fact is that reducing mutation to 10 will have a larger effect on performance than reducing mates to 10. My understanding is mating in this 1D array search only serves to test the mean of the two parents, this doesn’t provide much insights. But mutation works like a random search along the array and will more likely to provide improvements.

**3. Traveling sales man (GA)**

This is a classic graph theory problem to find an optimal traversal route with minimum weights sum. We all know this problem to be NP hard and usually not practical to find the real global optimal. But some complex randomized searching methods should be able to get significantly better local optima for this problem. Here is the overall comparison of the 4 methods:

We can see that though all algorithms can’t converge within 2000 iterations. Genetic algorithm found some traveling routes that’s much better than other methods. That’s because the cost of a traveling route is related to the order of places it visited. The mating and mutation of GA can help optimize many small local routes, thus it’s more like to get the global optimal result. If we compare the clock time of different methods, the plot is like:

The overall cost of running MIMIC is several thousand times higher than other algorithms. That’s because MIMIC uses joint probabilities to develop an understanding of dependencies within the data, but the exact dependency between alternative routes can’t be driven easily due the large search space of directed graphs. It’s also quite interesting that changing the parameters of GA didn’t affect its performance very much. It’s really the underlying mating and mutation process that’s significantly more suitable for this problem.

**4. Knapsack Problem (MIMIC)**

This a combination problem that don’t care about the order of items added to the backpack. So in theory the search space of this problem is much smaller, but one important constraint is the limit of item weights, that will make this problem also a NP hard one. This problem is also a very useful test set for all resource limited optimization problems and can have many implications in business and other real-world problems. The comparison of 5000 iterations is:

MIMIC performed exceptionally very comparing to others. It can converge after several hundred iterations and always find a better fitted solution. Comparing to the TSP problem, the searching space of this problem is smaller, and it should be easier to find the dependency relation of each item to the overall fitness. Thus, it’s easier to estimate probability distribution using the dependency tree. Also because of the complex tree structure, MIMIC always have hundred times larger clock time for each, the trade-off between time and accuracy should also be considered when apply MIMIC on real data.

I also did a comparison of different small m numbers used to create dependency tree:

It’s obvious that a larger m will provide better overall fitness. This preference of initial value of dependency seems to show that most of the items in KS problem have a very high dependency to each other. And by initialize a larger m value, MIMIC can estimate the probability distribution more accurately.