Randomized Optimization Analysis

Randomized Hill Climbing | Simulated Annealing | Genetic Algorithm | MIMIN

1. Introduction:

1.1 Algorithm explanation

- Randomized Hill Climbing:
  An optimal search algorithm from family of local search. Starting from a point and keep moving until it cannot find a higher position by gradient descent method. Then start over again and redo this process. That's how it avoids local maximum.
- Simulated Annealing:
  An optimal search algorithm from family of local search. This algorithm will act like Hill-Climbing but it picks a random more. If the new move improves the situation, then the new move will be accepted.
- Genetic Algorithm
  This is a algorithm based on the idea of natural selection where the better qualities will be passed to the next generation. Parents generate children who has their characteristics. Ideally, the children should have better characteristics than their parents'.
- MIMIC
  Stands for Mutual Information Maximizing Input Clustering. The basic idea is to figure out the structure of the data and then find the global maximum.

1.2 Dataset

- Description: The dataset I used is also from Kaggle, https://www.kaggle.com/drgilermo/nba-players-stats. It's called "NBA Players stats since 1950". The dataset contains aggregate individual stats for 67 seasons. There are in total ore than 15000 tuples storing each season's information. Attributes are Pos, Age, G, GS, MP, PER, TS%, 3PAr, FTr, AST%, etc.
- Preprocessing:
  Randomly shuffled the whole CSV file so that the distribution will be more even. There are 5000 out of 15000+ tuples will be used in this assignment. Training set: 3000, Testing set: 1500, validation set: 500. I will remove year and age because they are irrelevant to the result. Then I will replace the output with 0 or 1 in order to make binary classification. The output will be 1 if the athlete plays, otherwise the output will be 0.

2. Implementation:

2.1 Supportive resources:

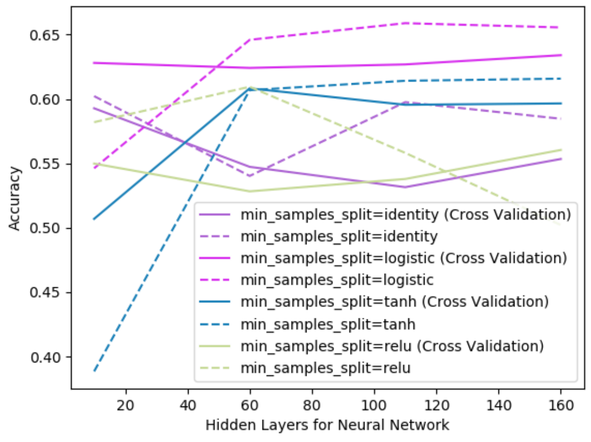
- ABAGAIL:
  A library contains java packages that implement machine learning algorithms. This library contains Randomized hill climbing, simulated annealing, genetic algorithm, and discrete dependency tree MIMIC. The algorithms that provided by this library could optimize the weights of neural networks and allow users to make crossover functions, mutation functions, and probability distribution.
- Jython:
  Jython is an implementation of Python written in Java. Jython programs use Java classes instead of Python modules. Jython is applied in this project because this project is implemented by Python but the ABAGAIL library is written in Java.
- Other toolkits: numpy, matplotlib, Pandas, sci-kit learn.

2.2 Parameter Tuning:

In this project, the goal is to figure out one of hyperparameters for neural network by running random research algorithm. The basic idea is for any distribution over a sample space with a global maximum, the maximum of 60 random observations lies within the top 5% of the true maximum, with 95% probability (Bergstra, J., & Bengio, Y. 2012). I would not provide the whole proving process in here but you are free to check to research paper from the citation source.

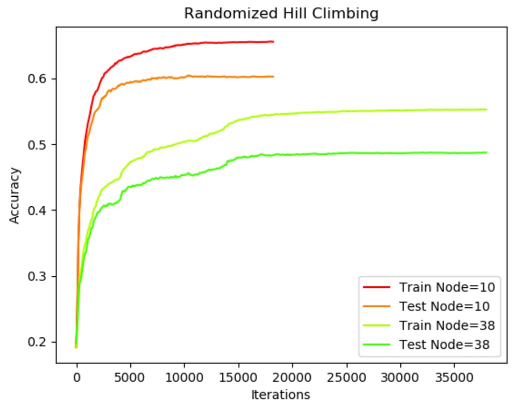
2.3 Reprocess Neural Network in assignment1:

Since I choose a different dataset, I would re-run the dataset and figure out the hyperparameter. Based on the graph, nodes more than 60 in the hidden layer would be too much because the curves are going straight which means there exists overfitting. However, if there are too less nodes say 20 nodes, the accuracy is also quite low. In this way, I would choose number around 40 as my number of nodes in the hidden layer.



3. Algorithm Analysis:

3.1 Randomized Hill Climbing:



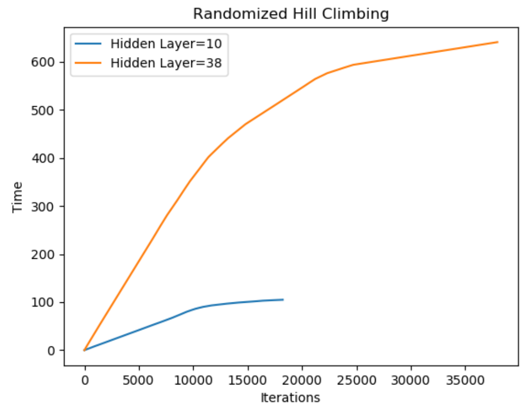


Figure 1.

Randomized

Hill Climbing

In Figure 1, the left graphs showing the performance given different hidden layers. The right graphs showing the running time given different hidden layers. In general, the neural network with the hidden layer of 10 perceptron has better performance than the neural network with the hidden layer of 38 perceptron. This result clearly shows that the larger hidden layer leads to the overfitting. Also, we need to think of the size of dataset. Since I only use a size of 5000 dataset for this project, I will not say that the model could be trained very good. I picked a dataset with many features so that ideally the size of training set and testing set will much larger than what I provide. However, with the consideration of implementation efficiency and my PC calculation ability, that’s what I compromise.

For randomized hill climbing, the algorithm is still easily satisfied with the local maximum while the situation of extremely unevenly distribution happening. And this will lead to too many times restart so that it takes very long time.

3.2 Simulated Annealing:

For simulated annealing, it will consider the next state while the algorithm is still stay in current state. Then it will calculate the probability of bettering off the situation by moving. What I mean is it will try to figure out how much it can better off the current situation by make that move. Generally, this algorithm could eventually lead the system to reach the low energy position. One thing that the simulated annealing works better than randomized hill climbing is that the simulated hill climbing would not use restart to avoid local optima. Before I run any code, at least I expect that the simulated annealing algorithm will have a faster speed then Randomized Hill Climbing. The calculation of probability of going downward is $p(x,xt,T)=1$ *if* $f(xt)\geq f(x)$, *or* $e^{\wedge}{}_{(f(xt)-f(x))/T}$ *otherwise*. In this way, simulated annealing will have a very low probability to get stuck at local optima.

The graphs are below:

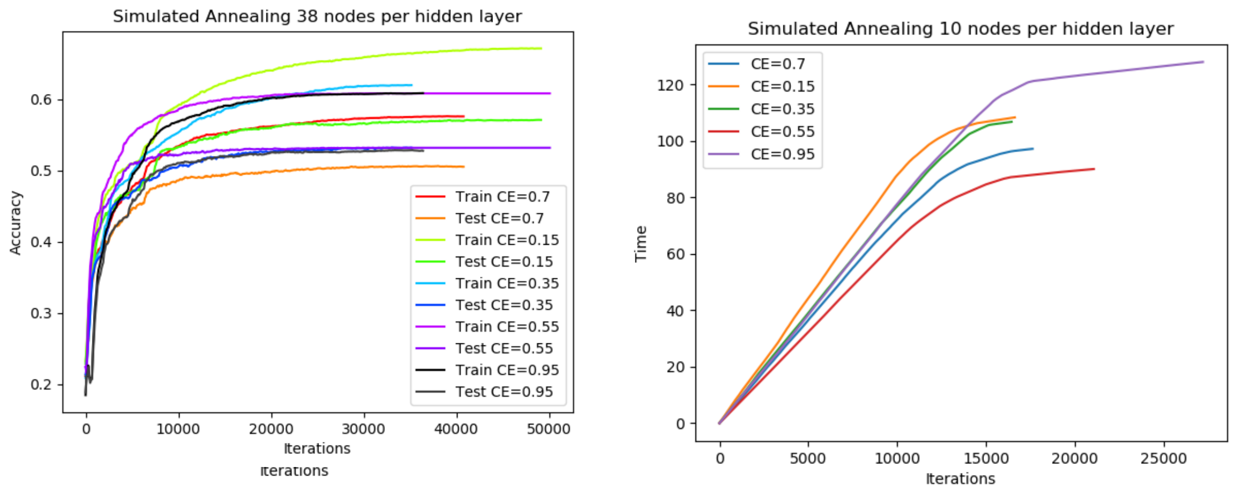


Figure 2. Simulated Annealing Algorithm

For the algorithm implementation, I give the algorithm a quite large starting temperature. CE stands for the cooling exponent. Based on the graphs of accuracy, the varying of numbers of nodes in the hidden layer does influence the accuracy. Basically, the hidden layer with 10 nodes has a better performance. However, if we compare the graphs of running time, we will easily notice that the large number of nodes in the hidden layer will cost very large running time.

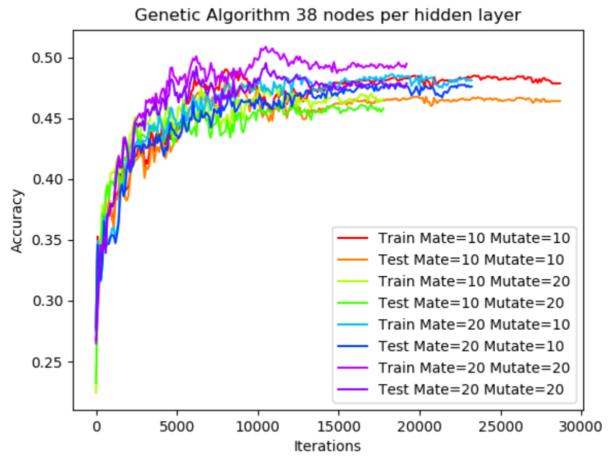
Using simulated annealing, the core idea is gradually cooling until the situation is stable and the temperature is low by Sexton, R. S., Dorsey, R. E., & Johnson, J. D. (1999). To the graph with 10 nodes per hidden layer, the cooling rate doesn't influence the accuracy to much. The reason could be the model has been already trained well and the cooling rate only would make running time different. Also, after 10000 iterations, the accuracy tends to converge. All those situations don't happen on the "38 nodes per hidden layer" could be because the model is overfitting.

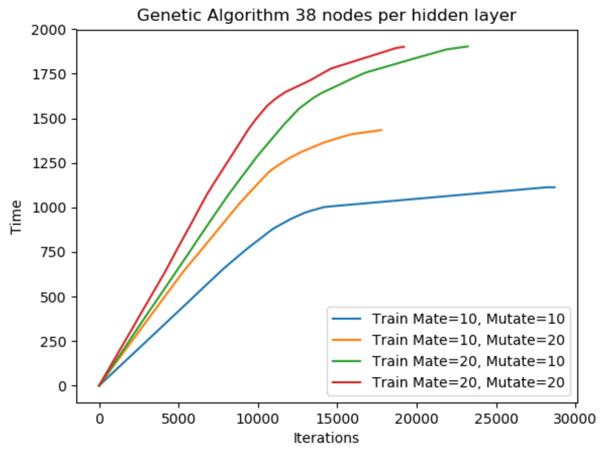
Either ways, the optimal cooling rate cannot be determined. It could be because my dataset is originally well-organized or the size of dataset isn't enough for training.

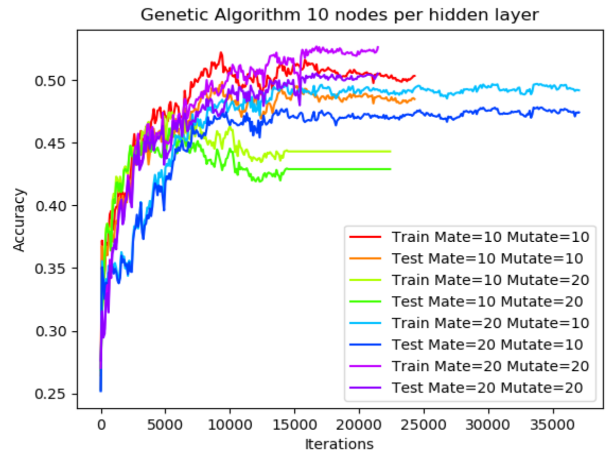
3.3 Genetic Algorithm:

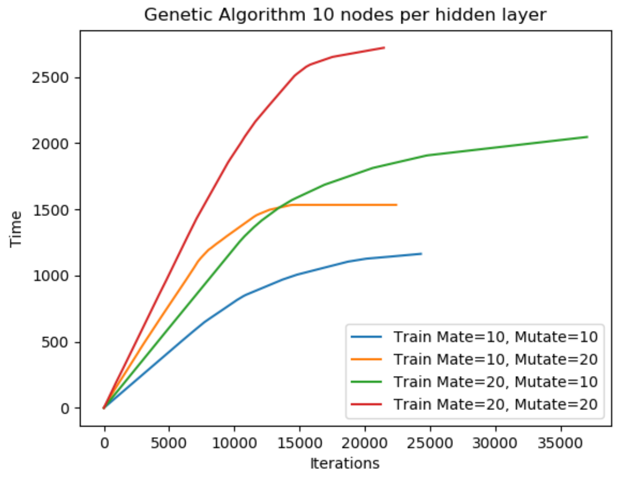
For this algorithm, it creates the population in random. During each iteration, the current population will create the next generation. As human beings, we know that it's easy to identify the genetic connection between ours and our biological parents, but it gets harder for people and their ancestors because there are too much combinations and possibilities. In this way, for genetic algorithm, it will take very long time to run if the population Due to the large dataset space, it's kind hard to consistently find the small portion of population which will improve the model. In this way, my expectation to the accuracy for this algorithm will be lower than others.

The running result is below:





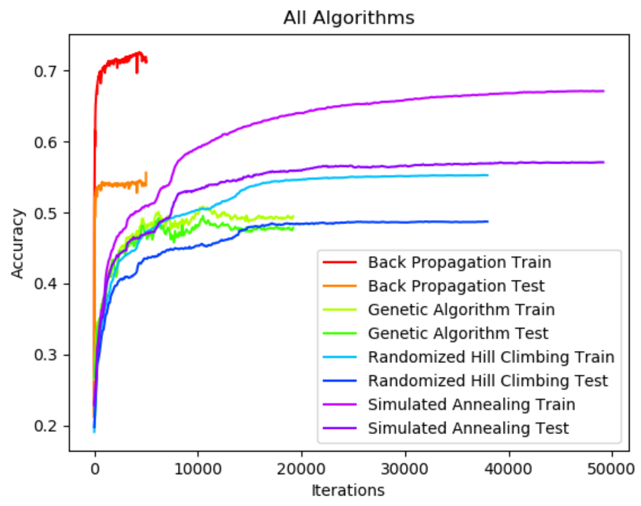




First thing I would like to mention is that I spend nearly two eight hours running this algorithm. For each iteration, it almost takes tens of minutes to run. In this algorithm, it will generate random variable for each bit in the sequence so that implementing the mutation. In this way, the mutation is like doing exploring because it tries to find something different. And the crossover is just do some recombination. Based on the graph, the larger mate and mutate will lead to a higher accuracy. Since the dataset is relatively large while comparing with the dataset only has a few hundred tuples, more mate and more mutate mean that higher probability to fit the global optima. But the it also means there will be a higher time-consuming.

3.4 Algorithm Comparison:

Up until now, we've tried Randomized Hill Climbing, Simulated Annealing, Genetic Algorithm. The purpose is to figure out if these algorithms could do better than back propagation in Neural Network. In this way, I need to run a comparison test to figure out their accuracy individually. First let's go over the training accuracy of Back Propagation. It has the highest accuracy, However, the testing accuracy of Back propagation is extremely low comparing with training. There is one possibility regrading this situation: Overfitting. Think of what the Back Propagation do in Neural Network algorithm, it just keeps changing weights based on the given feedback. This process will have a very good learning effect if the data is completely linearly separable say figure out y = 2 x, then the back propagation is very useful. However, for dataset with many features and not linear separable, back propagation is easily to be overfitted. In average, simulated annealing has the best performance since we've already said that back propagation is overfitting. However, Genetic Algorithm has the least difference between training set and testing set.

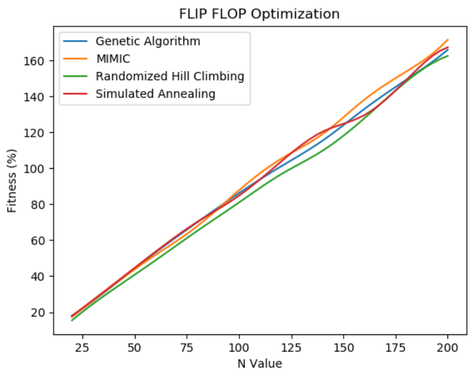


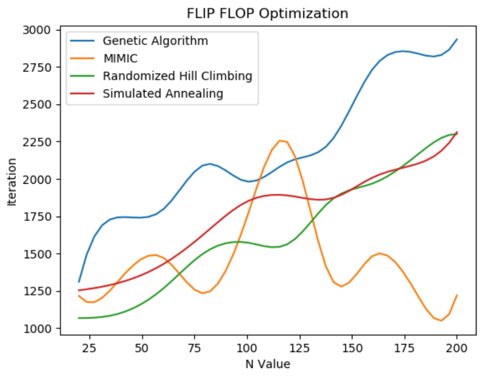
One thing to notice is that the network weights are continuous values instead of discrete values, based on calculus, we know for a continuous space, there are infinite possibilities. Now I will group randomized hill climbing (RHC) and simulated annealing (SA) together and compare them with genetic algorithm. Since RHC and SA has less random choices than genetic algorithm. RHC only start over once it reaches the local optima, and SA only randomly research the next state. However, genetic algorithm has mutation and crossover so that it has more probabilities so that in a infinity space it will has the best performance if we consider overfitting as a bad training effect.

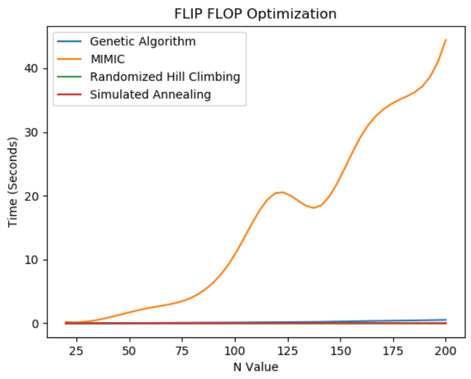
4. Self-made problems:

4.1 Flip-Flop:

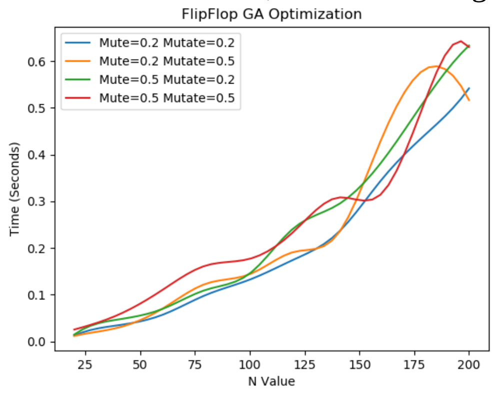
This is the example problem from ABAGAIL library. Running four algorithms on this problem and figure out the fitness and running time in order to analyze which algorithm has the best performance.

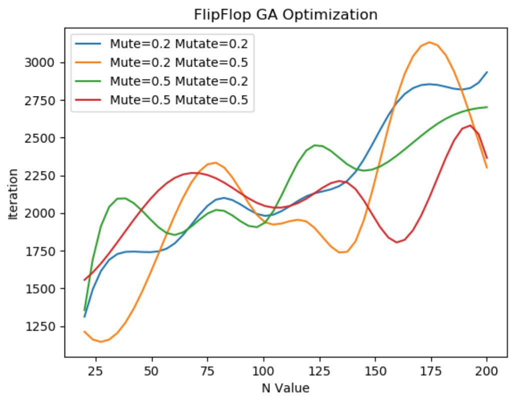


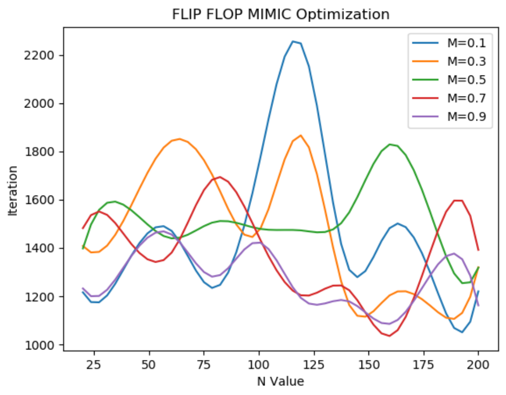


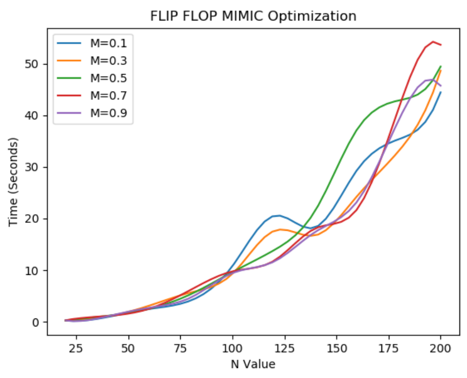


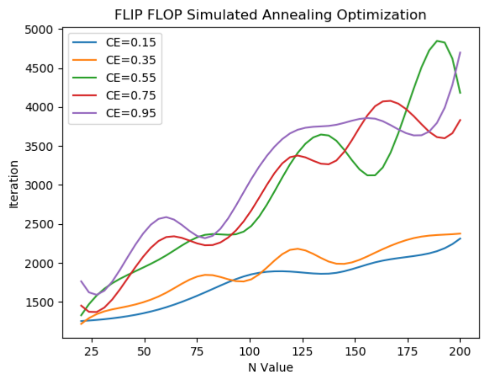
For the N-value, it just for different size of uniform distribution. For example, the distribution could be [1,1,1,1,1] and [1,1,1,1,1,1,1,1,1,1] for different size of N. The reason I am doing that is because I want these local search algorithms to search is a range of area. Ideally, the smaller the N is, the less iteration will be required in order to finish to training process. At this point, the default values is following: mate = 0.2 mutate = 0.2 for Genetic Algorithm, CE = 0.1 for simulated annealing, and M = 0.1 for MIMIC. In above graphs, MIMIC has the weirdest behavior so that we need to give some analysis on that. The graph shape of MIMIN in the middle picture is very similar with normal distribution graph while they are not technically same. Think of how MIMIC works. MIMIC is the algorithm which is trying to figure out the structure for the entire dataset. Since the training samples of Flip-Flop is limited, for a large range of distribution, MIMIC almost always could do better than other local search algorithms because it's very easy for them to be truck in local optima. However, all four algorithms reach the same level of fitness after they figure out the optima. And the picture one actually shows the linear relationship. The reason that MIMIC takes very long time to running shown as picture 3 is because it need to figure out the structure and the larger the range of distribution is, the harder to figure out the structure. Then I will run three local search algorithms:

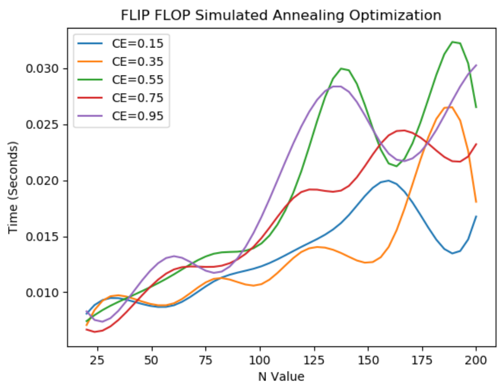








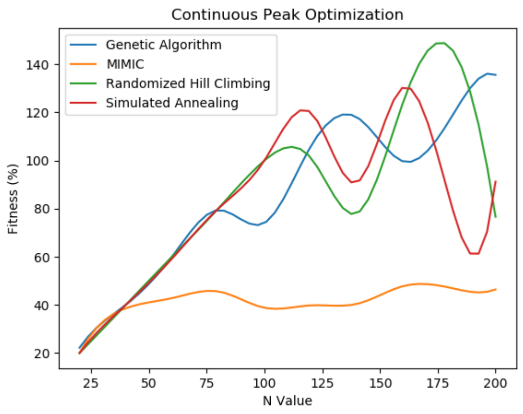


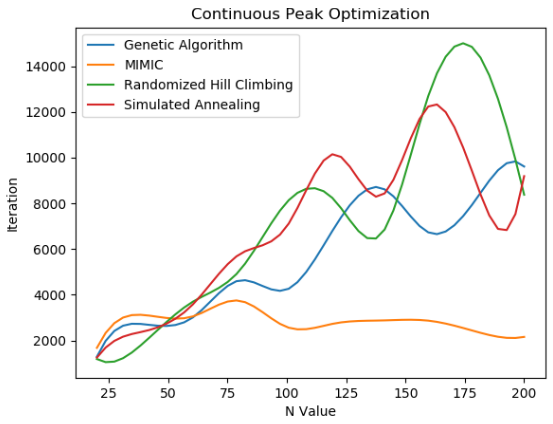


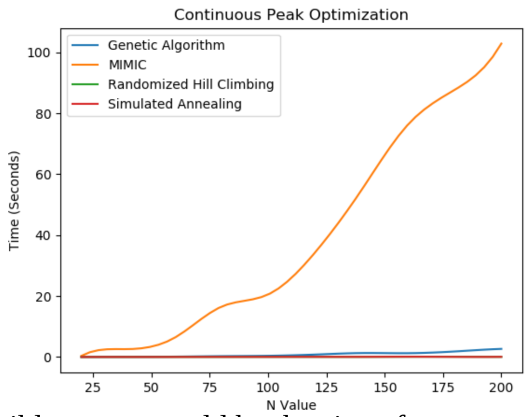
Overall, Simulated Annealing has the lowest variance and shortest running time so that simulated annealing has the best performance on this problem. Since Flip-Flop is a quite simple problem unlike analyzing the stock prices, simulated annealing is very fit in this circumstance. Simulated annealing would pick the next state randomly and evaluate this state. For simple question like that, it doesn't have many too much local optima and the distribution is quite even so that simulated annealing has better performance.

4.2 Continuous Peaks

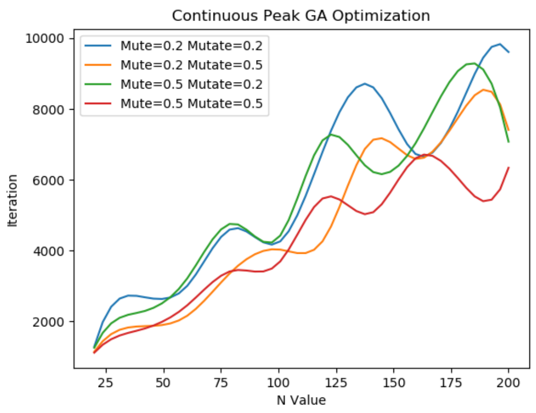
This is from the Abagail example problem. Basically, this problem would assign the fitness value with the unbroken 0s and 1s sequence. This question is that kind of question which has a lot of local optima and the global optima is hard to find. In this time, the N is increment from 25 to 200 with 25 incrementation per iteration. The overall performance is below:

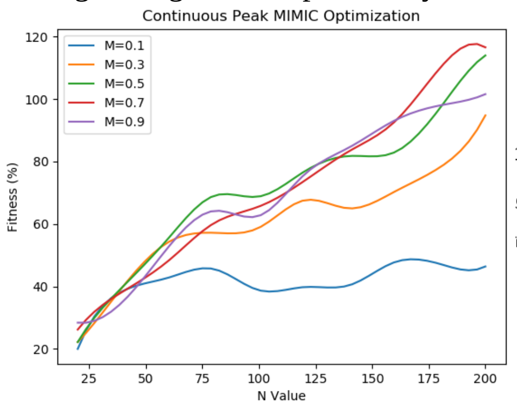


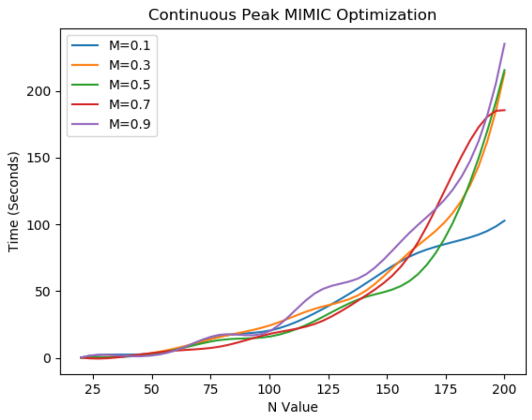




This time, MIMIC has the worst fitness, and performance. One possible reason could be the size of the problem is too large to allow MIMIC figure out the structure. Now, notice the peak of randomized hill climbing in second picture. The randomized hill climbing get peak at N = 175, meaning that it stuck in the local optima. However, the iterations go down later, meaning that randomized hill climbing finally figures a way the get rid of local optima. Knowing that hill climbing only restart when it reach the local optima, so here are some interesting things worth to be mentioned. For the distribution which has many local optima but they are very easy to approach, then the randomized hill climbing wouldn't take too much time before it finds the global optima. Now, running the algorithms specifically:



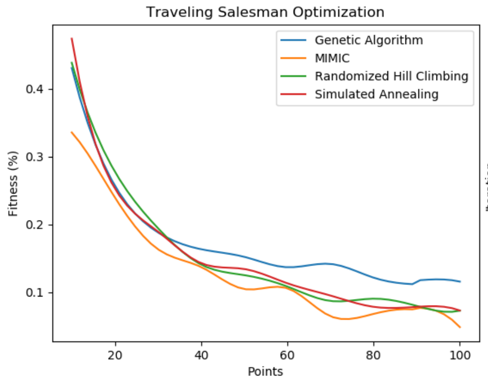


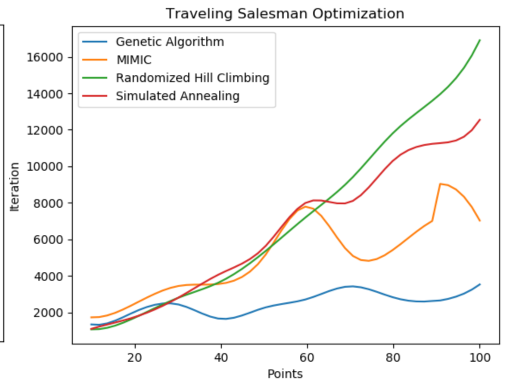


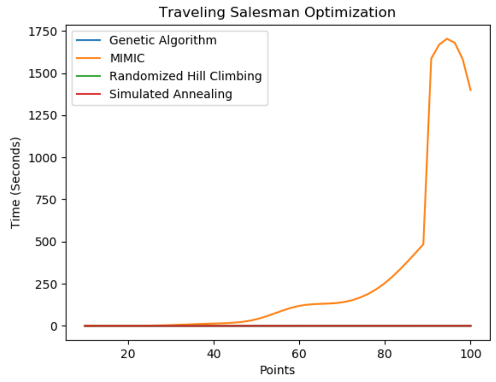
Since I've already figure out simulated annealing, so that I would like focus on genetic algorithm and MIMIC. So the genetic algorithm will take less iteration when it has the highest mute and mutate values. This result just shows that the more variety genetic algorithm has, the faster it will be to find the global optima. For MIMIC, it has lower fitness value when the M value gets bigger. Basically, genetic algorithm has the least variance, seeing from continuous peaks optimization (fitness and N value), and has generally least iteration increment. In this way, genetic algorithm has the best performance on this problem.

4.3 Traveling Salesman

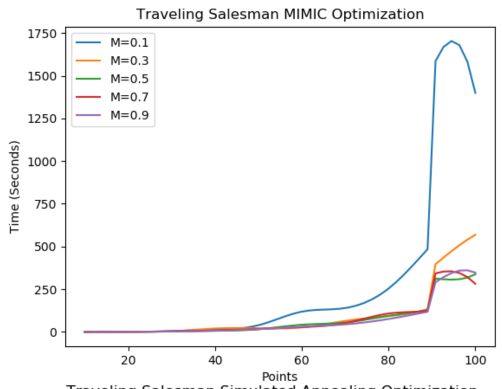
The traveling salesman problem is about finding the shortest possible route that visits each city and returns to the origin city. Just think of this problem as apply Eulerian cycle to a graph problem and we need to find the shortest Eulerian cycle. Here, we use "points" stand for cities. What probably will happen is that the increment of points will lead the fitness goes down because the whole situation goes to more complex.

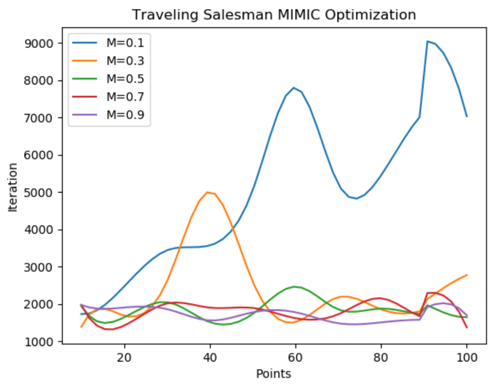


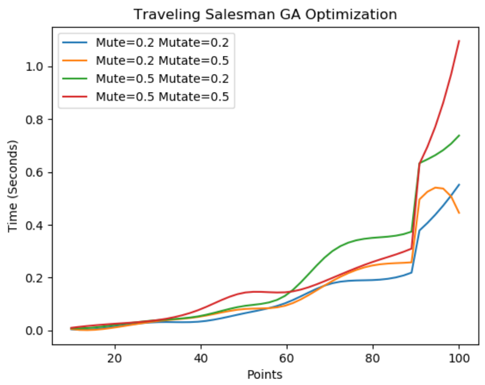


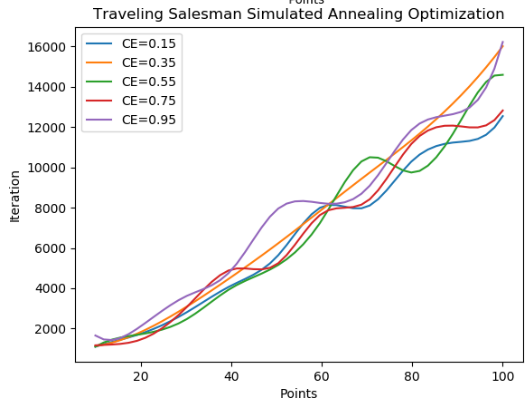


Genetic algorithm takes least iteration and time. Since genetic algorithm uses the crossover function instead of randomly choose so that this is might be the reason why it has the better performance. Think of how the salesman problem work: when the salesman picks a city, his current choice will directly affect his next move. Based on this property, crossover function is just doing the same thing cause there are parents and children. Interestingly, MIMIC always takes very long time to run once the structure get complicated. For simulated annealing and randomized hill climbing, technically there are very similar and they also have the close performance in this problem.









With the increasing of points, the iteration of MIMIC doesn’t change a lot except for M = 0.1. What we need to notice is that MIMIC only cost a large time when M = 0.1 and MIMIC’s time and iteration aren’t influenced too much by incrementing of points. I will try to analysis MIMIC and genetic algorithm together.

By observing the graph which contain four algorithms’ performance, we find that MIMIC has the second least iteration but its fitness is worst. For these two algorithms, they both try to solve problem without using random search or random taking. MIMIC tries to find the structure and genetic algorithm focus on improving by generations. If any problems that MIMIC would have a better performance, I think it would be that kind of problem which is very structural and has unevenly distribution.

5. Citation:

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*(Feb), 281-305.

Sexton, R. S., Dorsey, R. E., & Johnson, J. D. (1999). Beyond backpropagation: using simulated annealing for training neural networks. *Journal of Organizational and End User Computing (JOEUC)*, *11*(3), 3-10.

Idrissi, M. A. J., Ramchoun, H., Ghanou, Y., & Ettaouil, M. (2016, May). Genetic algorithm for neural network architecture optimization. In *2016 3rd International Conference on Logistics Operations Management (GOL)* (pp. 1-4). IEEE.