CS 7641 Project 2: Randomized Optimization

Yaling Wu (ywu342)

Part 1 – Optimization Algorithms for Neural Network

Dataset – Car data: https://archive.ics.uci.edu/ml/datasets/Car+Evaluation

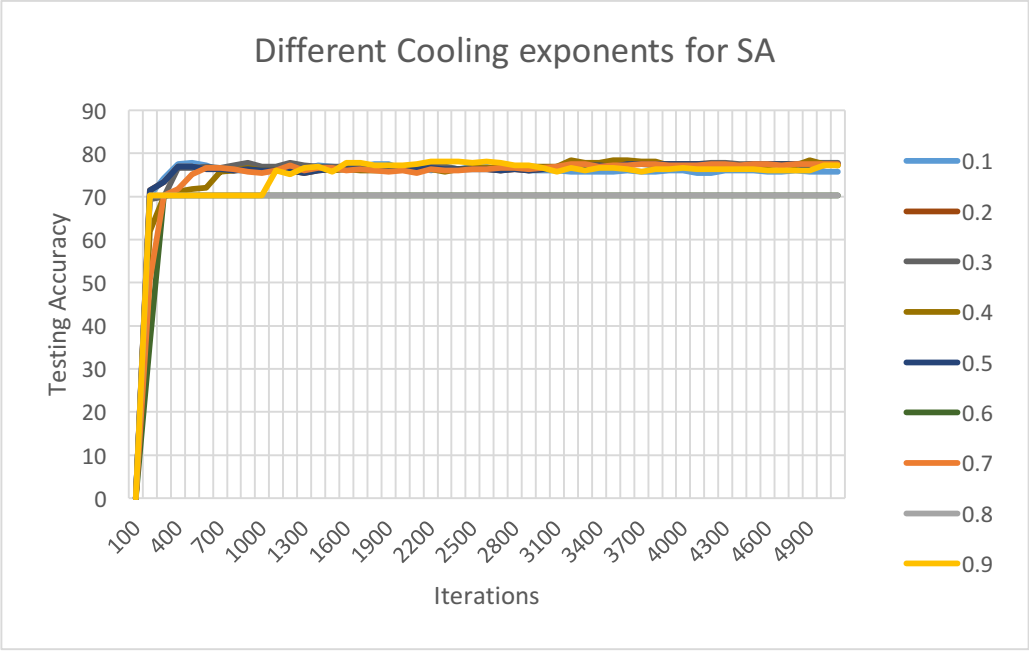
I used Car dataset from last assignment. This data was derived from a hierarchical decision model. It was used for the evaluation of Hierarchical Induction Tool, which was proven to be able to reconstruct the original hierarchical model. So it is particularly useful for testing constructive induction and structure discovery methods. It includes 6 attributes like buying, maintenance prices, doors, persons, size of luggage boot, and safety. The output classes indicate acceptability of cars: unacc (70%), acc (22%), good (4%), vgood (3.8%), where the percentages are representatives of the data distribution. There are 1728 instances in total.
In earlier experiments, neural network using backpropagation did fairly well on this dataset. It reached a testing accuracy of 84.78% with parameters of 200 iterations, 1 hidden layer and activation function relu.

Optimization experiments on Neural Network

In this project, I replaced the propagation logic with three optimization algorithms in NN such that I could compare the effects of all four weight optimization techniques on NN. The parameters of NN are fixed to be 6 nodes in input layer, 4 nodes in output layer, and 1 hidden layer (same as last assignment). Results of three optimization problems are displayed below. Similar to assignment one, I picked some parameters to tune a bit and compared how different parameter values affect the performance over a range of iterations from 100 to 5000.

Simulated Annealing (SA)

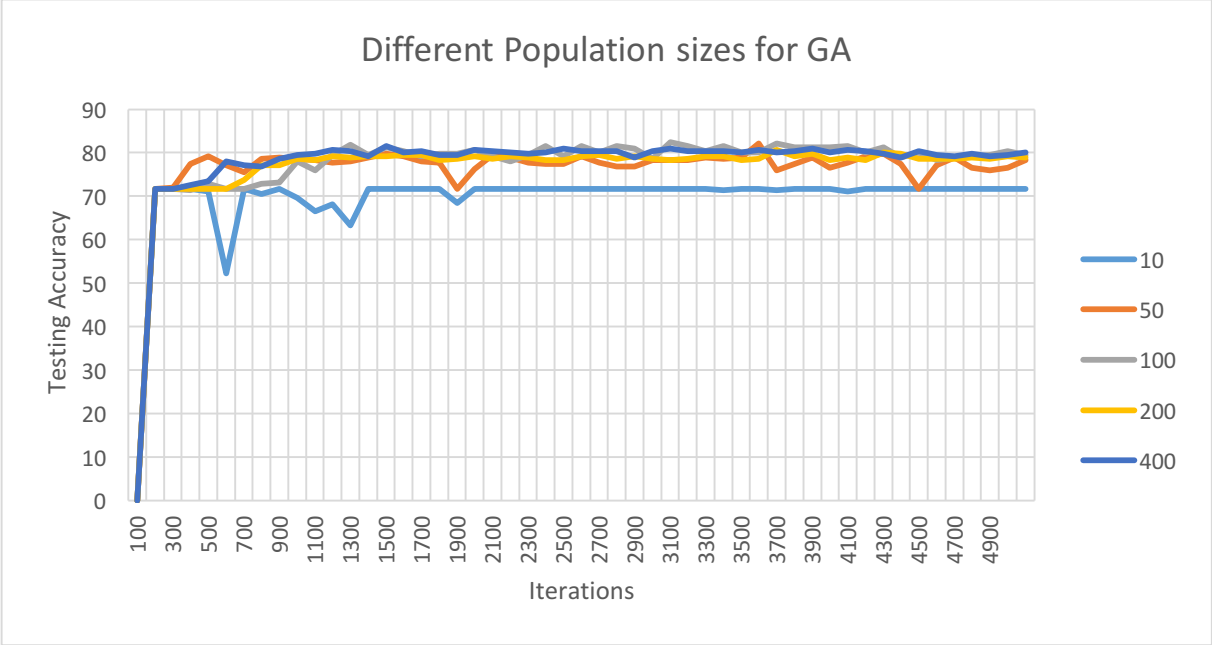
For SA, only cooling exponent was the parameter of interest. Cooling exponent indicates the rate of temperature decrease. The higher it is, the more slowly temperature decreases. Starting temperature was a fixed value of 1e11.



From the figure, we could see that all cooling exponents led to converging testing accuracy at around 80%, except that 0.8 performed a little below others. As cooling exponent increases, SA gets slower at finding a local optimum, i.e. reaching convergence. For instance, at the cooling exponent of 0.9, it took almost 1300 iterations to reach to a higher testing accuracy level. This does not contradict with the properties of SA itself. If it slowly decreases temperature, or the exploration metric, it will spend more time in exploring new possibilities and wandering around. But the fact that all of the cooling exponents converged to the same level towards the end tells us that SA is most likely to be destined to fall in the same local optimum unless we increase the exploration chance more dramatically (i.e., via other parameters of SA).

Genetic Algorithm (GA)

For GA, the controlled parameter was population size. It is the number of points we look at in each iteration. The larger this population size is, the more time we will spend on each iteration. But meanwhile, we explore more points in each iteration. The corresponding toMate number was always half of the population size.

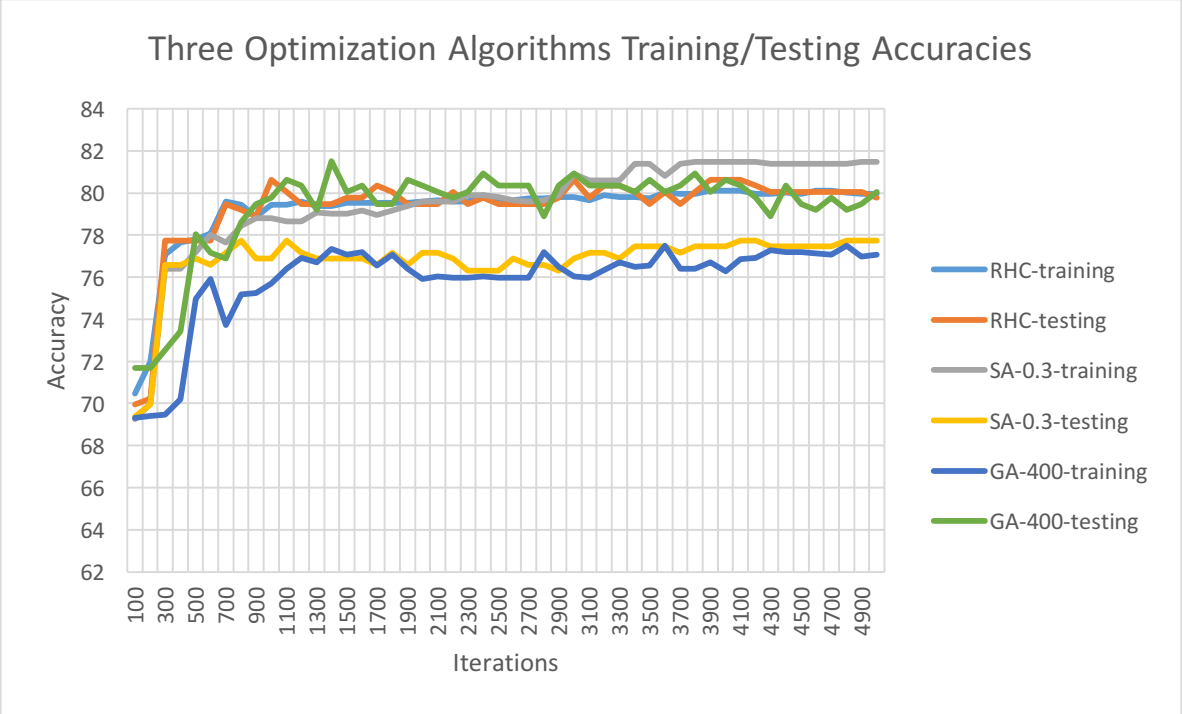


From above figure, we could see that a population size of 10 was not enough to lead GA to a good local hill on the fitness function, compared to other population sizes. All other sizes eventually caused a convergence at around the same level. So they performed almost equally well once size got over 10. Still, the larger the size is, the better the optimal NN weights are. Like SA, GA also took over 1000 iterations to settle down on an optimal set of weights. It seemed GA may has also taken on the same local maximum as SA in terms of fitness.

Comparisons

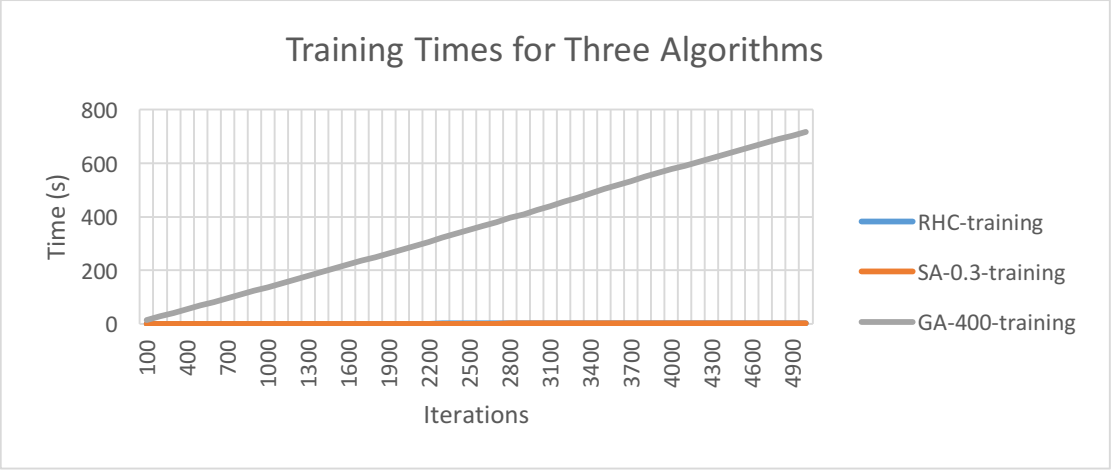
Optimization Algorithm	# of Iterations	Algorithm Parameters	Testing Accuracy	Training Time
Backpropagation	200	-	84.78%	1.786 s
RHC	3900	-	80.636%	2.549 s
SA	4100	Cooling exp: 0.3	77.746%	2.362 s
GA	1500	Population size: 400	81.503%	208.4 s

Above table shows stats resulting from the Neural Network classification with four different weight optimization algorithms. The parameters of RHC, SA and GA were selected using their maximum performance in terms of testing accuracy. Overall, none of the new algorithms beat backpropagation when it comes to classification but the three cut it pretty close.
Backpropagation won the game also with the shortest training time. RHC, SA and GA algorithms needed more time/iterations to find a relatively good weight set like backpropagation. This could be due to the fact that the domain of fitness function (in this case, NN weights) consists of real values. Weight values are continuous. The neighbors of a weight value can be infinite and hard to determine. RHC, SA and GA needed more time to find informative neighbors to decide on the direction go on the fitness function, while gradient descent can quickly tell which direction potentially lead to a local hill. Therefore, backpropagation is the best optimization technique to use for this dataset.



Above figure shows a more detailed comparisons of accuracies among three algorithms. For SA and GA, I selected the parameters causing maximum average accuracies across the iterations range. Mostly training accuracy is lower than testing accuracy. They all increase as the number of iteration gets larger but they also converged at the end. So increasing number of iterations might not help too much to improve the performance. All three algorithms did not perform very differently.
SA performed the best with a high number of iterations, in terms of training accuracy, which implies that it found the best local optimum with training data. This result tells us that SA's dynamic exploration of less fit neighbors at the beginning (when the temperature is high) could be very helpful for this dataset because it may lead SA to explore a nearby higher local optimum. On the contrary, RHC and Backpropagation just heads to the direction of exploitation (i.e., the direction of fitter neighbors or gradient descent). GA did not perform too well in

finding local optima compared to the other two, because it gave the lowest training accuracy. It makes sense as the weight parameters take in effect as a whole, not individually. There are no such things as good input parameters which are beneficial to pass on.
However, it is not impossible there still exists another global maximal hill somewhere further down. It could be useful to strengthen the initial exploration in that case, by increasing initial Temperature and making cooling exponent bigger for SA, or by increasing population size, toMutate and toMute for GA.



As shown in above graph, even though GA turned out to be the best of the three (in terms of classification performance), it took significantly longer time to train than the other two. Since GA needs to evaluate hundreds of data points Time it takes increases even more significantly as population size becomes bigger.

Part 2 – Optimization Problems

Experiments of the four algorithms: RHC, GA, SA, and MIMIC were performed on three optimization problems so we can see a deeper understanding of them. Two metrics to evaluate how each algorithm performs were the time it uses to solve the problem and the fitness of final optimum it gives. For each optimization problem, I used fixed algorithm parameter values, which are indicated besides algorithm names in the legends of figures.

Four Peaks Problem

The Four Peaks problem is defined as follows: Given an N-dimensional input vector $\vec{X}$

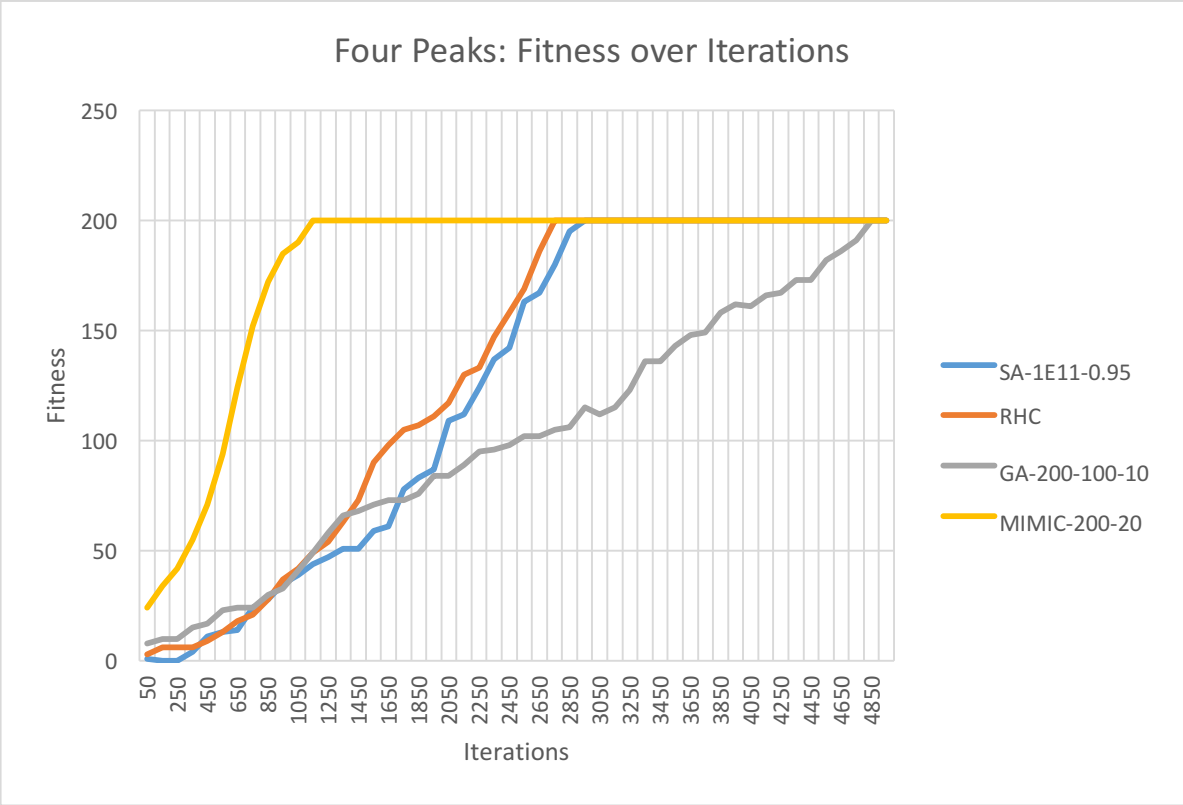
$$f(\vec{X},T) = \max\left[tail(0,\vec{X}), head(1,\vec{X})\right] + R(\vec{X},T)$$

where

$$tail(0,\vec{X}) = \ number\ of\ trailing\ \ 0\text{s in}\ \vec{X}$$
$$head(1,\vec{X}) = \ number\ of\ leading\ 1s\ in\ \vec{X}$$
$$R(\vec{X},T) = \begin{cases} N\ if\ tail(0,\vec{X}) > T\ and\ head(1,\vec{X}) > T \\ 0\ otherwise \end{cases}$$

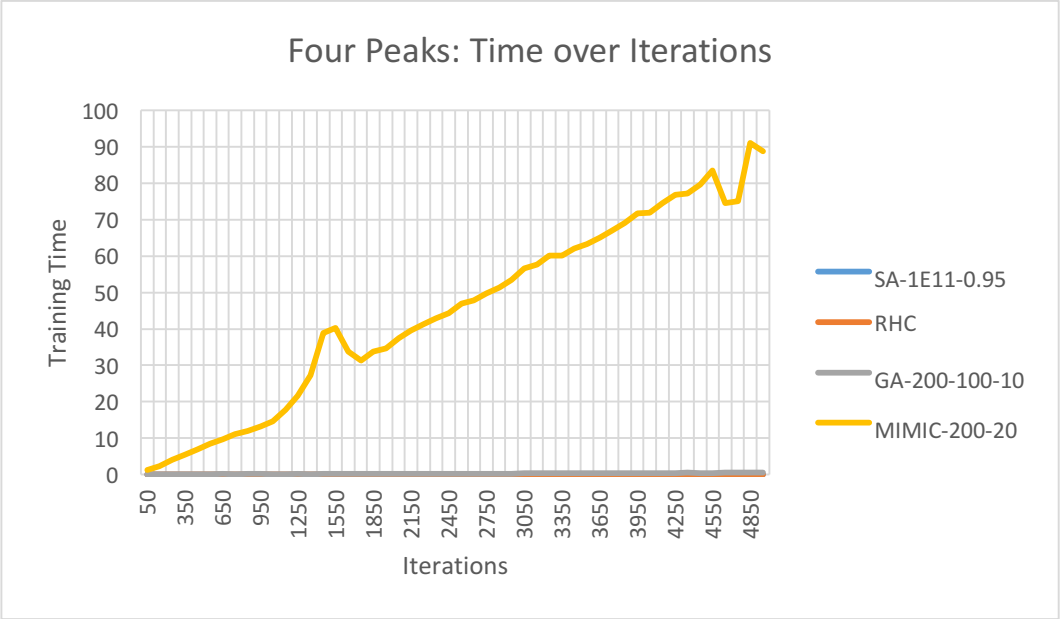
Two global maxima are available for this problem. They are either T+1 number of 1s followed by all 0s or T+1 number of trailing 0s preceded by 1s. Two other local maxima are all 1s or all 0s. Since this is a discrete problem, the fitness function is non-smooth.

This problem is interesting in that there are some structural dependencies among the input parameters (i.e. the first T bits should be 1s) that some optimization algorithms may exploit. The N I used was 200 and T was 20. So the optimal maxima should be 200+79=279. And suboptimal local maxima should be 200.



All the algorithms converged eventually in this problem (i.e., they all found a suboptimal optimum). That implies 5000 iterations were sufficient to explore the basin of attraction for the optimal maxima. Among them, MIMIC performed the best because it had the highest fitness values all along and it converged within fewest number of iterations. As I have mentioned in the problem definition, there are dependencies among input parameters. MIMIC dedicates to estimate this dependency tree from samples at each iteration. Thus it is a perfect problem for MIMIC to exploit. As to how to make MIMIC converge even faster, we can increase the number of samples generated at each iteration and decrease the number of samples to keep. The performance of RHC and SA stayed pretty closely as they behaved similarly in this problem. For them, they had no idea of the structure in the function. Instead, they randomly explored around to try to reach an optimum by looking for neighbors at each iteration. To improve SA on this problem, it is probably helpful to increase its initial temperature and cooling exponent such that it has a larger range of exploration, which causes a higher likelihood of reaching global optima faster. Since they only look at one point and change a bit to get neighbors at a time, it would take them longer to find an optimum. Even if GA tries to preserve good groups of parameters over generations but since its mutations and crossovers are random, it is very easy for it to lose this preservation in the offspring generation. Especially when the group of parameters is large, it is more likely for break the group at next generation. We can try

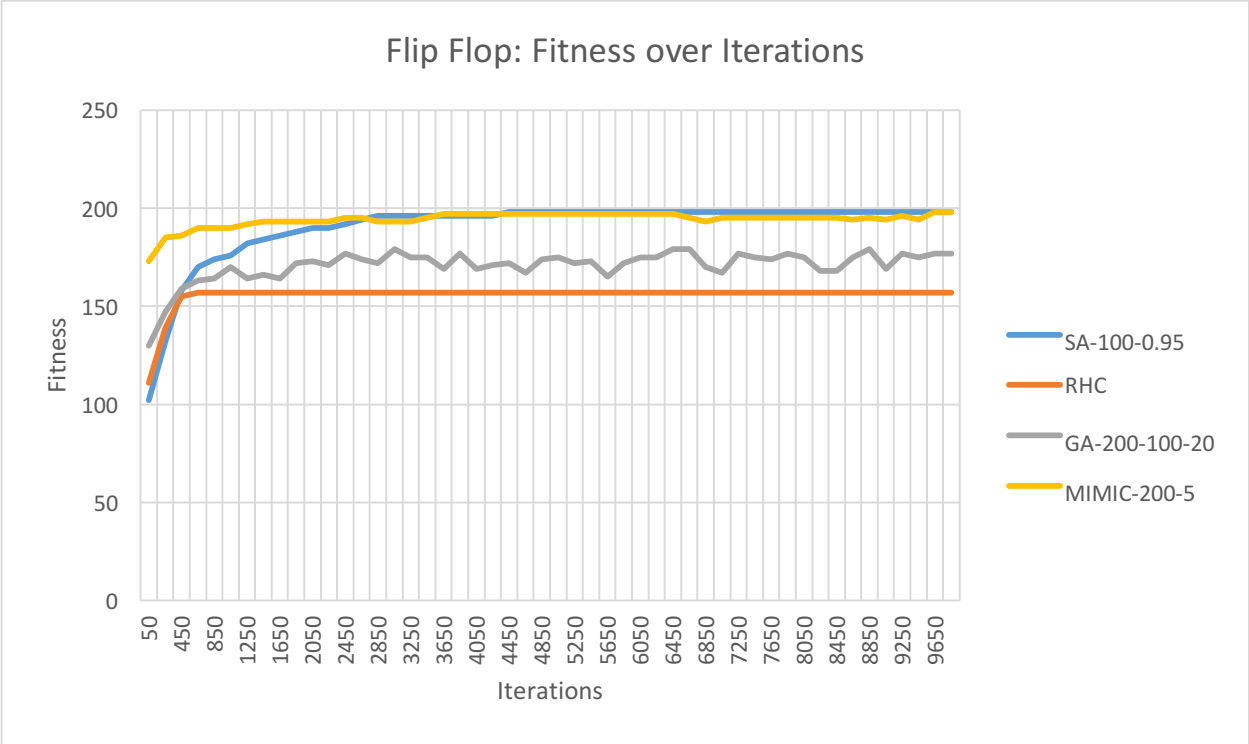
increasing the number of mating or decreasing number of mutations to decrease the likelihood of aforementioned situation happening.

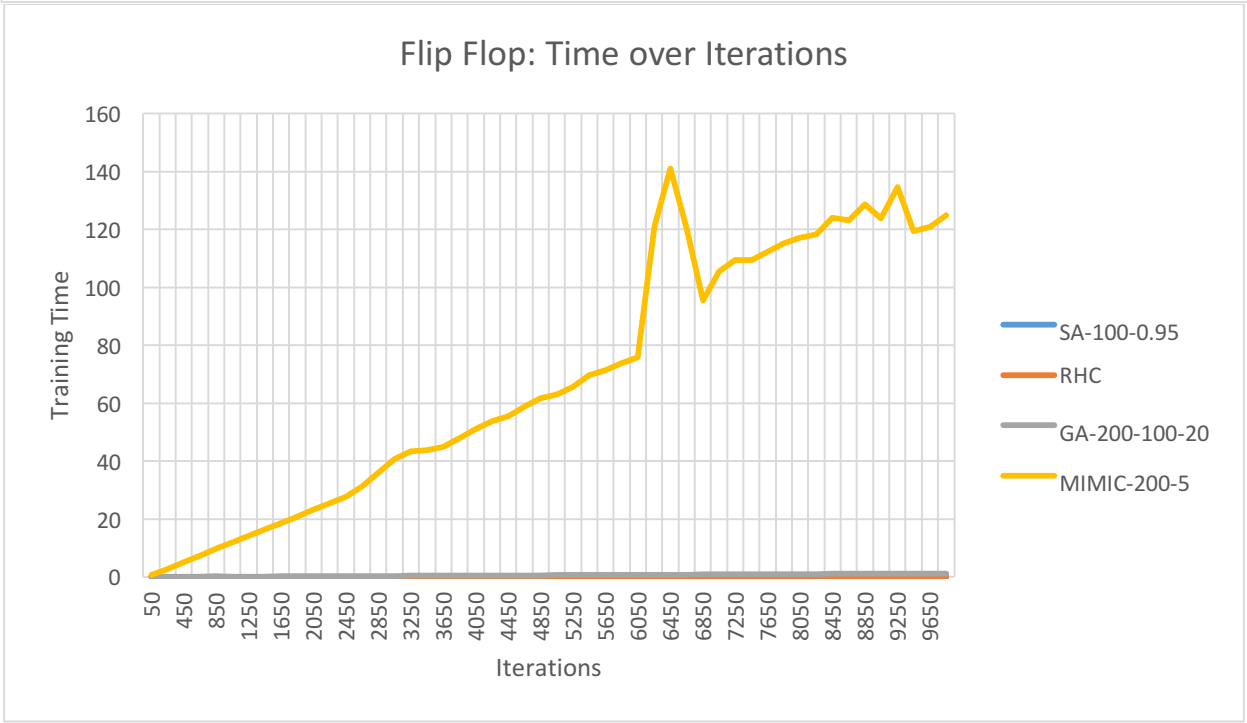


As shown in above figure, the time it takes for MIMIC to train exceeds other algorithms by a lot. From last figure, we learned MIMIC converged at around 1100 iterations, which was much quicker than other algorithms but it still consumed much more time. That shows each iteration of MIMIC takes very long compared to others. This was because MIMIC needed to do computationally expensive tasks (i.e. computing mutual information pairs and maximum spanning tree) at each iteration. And among other three algorithms, GA was the slowest due to the number of samples it needed to compute at each iteration.

Flip Flop Problem

This problem aims to bit strings consisting of alternative bits. Therefore, there are two optimal maxima: 1010101010... and 0101010101…. And there are a lot of suboptimal local maxima in a pattern like 101010|010101. The longer the input vector is, the more local maxima there are. And a lot of them are quite close to the global optima. The fitness function should be non-smooth. Again, N was fixed to be 200 in my experiments. So the global optima should be 199.



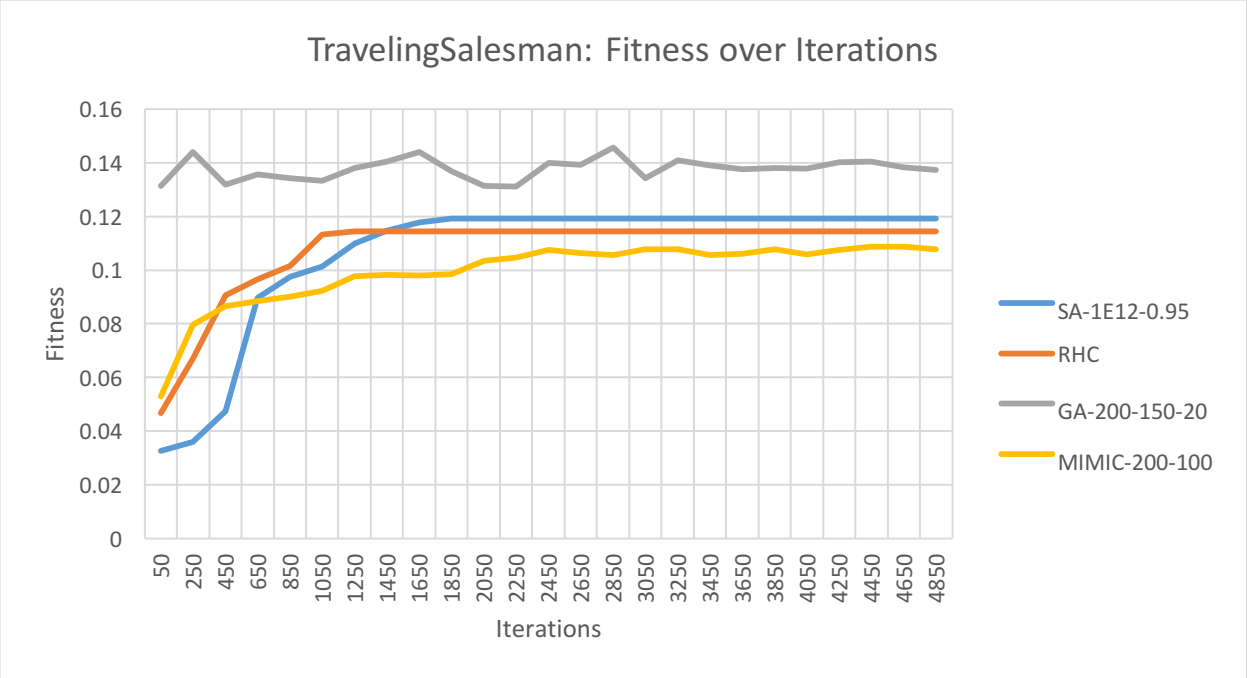


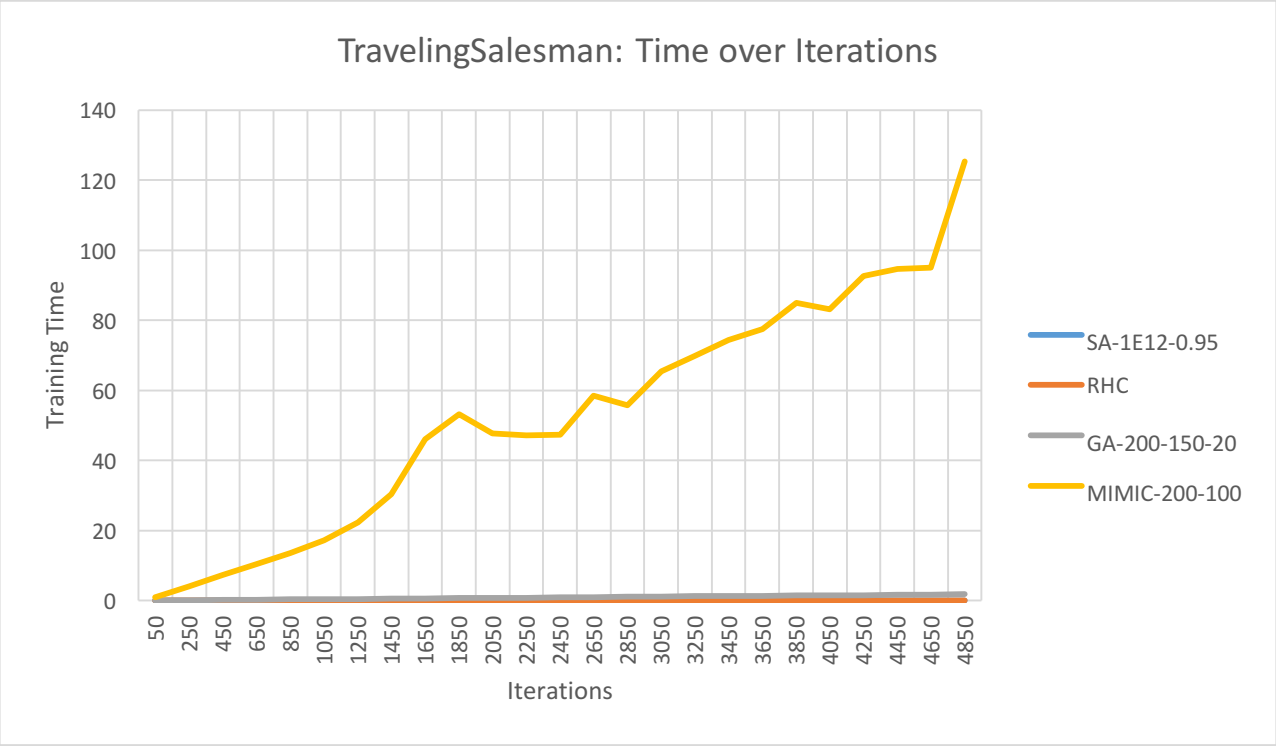
From above graph, we can tell that SA outperformed other algorithms. Here MIMIC performed a little worse than SA because it was not as stable as the latter and most of the time the latter maximized the fitness values. MIMIC did not reach to around the highest value until around 4000 and then later it fell down a bit. SA converged at around 4400 iterations. It found the global optima successfully at the point of converging. This performance is not surprising because SA is good at handling various local optima due to its frantic exploring behavior in the

beginning. It can climb across multiple small hills or even higher level hills if the temperature is high enough. That makes it very likely to fall in the basin of attraction for global optima if they are nearby. The only drawback here is its converging point is quite far down the road. To speed up the convergence, one way is to decrease the cooling exponent but it will also increase the likelihood of missing out global optima because of decreased exploration time. RHC converged to a local maximum more quickly but it did not get to the optimal solution. This was caused by its random restarts that led to a non-optimal basin. GA, in this case, was trapped with a group of samples that have similar suboptimal fitness values even after recombination. With a slim chance, it is possible to crossover at the right spot to go uphill. To increase this chance, it might be helpful to increase GA's toMate parameter. MIMIC did very well for this problem, even though it was not as good as SA. This is because there exist inherent structural relationships among input parameters that it can exploit. Each input bit depends on its previous bit. But there is also a big downside of MIMIC that cannot be avoided. That is the time it takes to compute, as shown in Time vs Iterations graph. When both MIMIC and SA got to the same level at around 4000 iterations, MIMIC's training time was already more than 10000 times as long as SA's.

Traveling Salesman Problem

TSP is an NP-hard problem, so there is still no polynomial-time solution to it. It is a problem that describes a salesman who must travel between N cities. He needs to visit each city once during his trip and finishes at where he started. His goal is to find the shortest path resulting from the whole trip. The number of possible combinations of the order in which he visits the cities is N!. There can be more than one optimal solution and many suboptimal solutions. The N I used here was 50.





GA is obviously the best approach to take when it comes to solve TSP. It got the best performance over all iterations and as usual, it was the second more time consuming algorithm, but it was not bad (around 1.808 at 4850 iterations). GA did well on TSP because there are subsets of the city order list that are considered as good genes. We would like to keep these subset city orders as is in the upcoming exploration. As we all know, GA is good at passing on good genes to the next generation. SA and RHC were not able to capture this advantage. What they did was to aimlessly trying out all the swapped neighbors and went to the good ones. But good ones may not lead in the right direction. Increasing initial temperature for SA may help to extend the exploration period. MIMIC performed the worst here because there are no dominating structural dependencies among input vector parameters. There is not too much of a structure for it to exploit here. And as always, it takes a lot more time during each iteration than other algorithms. It is much more expensive to increase number of iterations for MIMIC. But with more iterations, fitness values might be able to slowly increase.

Conclusion

All the four algorithms have their own advantages and disadvantages. Depending on the nature of each optimization problem, some algorithms perform better than the others. I will use a table to summarize advantages and disadvantages of all four algorithms:

Optimization Algorithm	Pros	Cons	Computation Time
RHC	Very fast. Can quickly find an optimum if the basin of attractions for global optima is huge. Very effective if there are only a few optima hills	Relies too much on chance to approach the global optima. Can be easily stuck at a local optima.	Fast
SA	Good at deal with problems with a lot of small nearby local optima, especially if they can lead it to the direction of global optima. Better than RHC in that it is smarter at exploring neighbors and does not rely on exploitation too much.	Not too effective if basin of attraction for global optima is far away and small. Sometimes it needs more iterations to converge because of the cooling effect.	Fast
GA	Tries to preserve and propagate groups of parameters that might be partially responsible for good fitness values.	Based on chance, good genes may be destroyed over the generations. Can get stuck in local minima if crossovers and mutations generate similarly fitted offspring.	Medium
MIMIC	Typically requires orders of magnitude of less iterations. Tries to exploit structural dependencies as much as possible at each iteration. Very useful when parameters are closely related. Good for problems which have an expensive fitness function.	Each iteration takes long. It only assumes pairwise relationships so more complex relationships are omitted.	Slow