Bhavani Jaladanki

GTID: 903009659

CS 7641

**Randomized Optimization**

*Introduction:*

In this assignment, I will be running different optimization methods: Randomized Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithms (GA), and Mutual-Information-Maximizing Input Clustering (MIMIC). First, instead of backpropogating (which uses gradient descent), I use the first three algorithms to find better weights for the neural network that I used for the Adult dataset in the previous assignment. Adult is data from the 1994 census which shows whether a person’s income is greater than 50k per year, or not, based on attributes/factors such as age, education, occupation, native-country, marital-status, etc. This dataset contains 14 such attributes, and 17,908 instances, which are classified as yes or no for whether the person earns > 50k or not. Second, I applied all four algorithms to the three different optimization problems: Continuous Peaks, Flip Flop, and Travelling Salesman Problem.

Hill climbing is an algorithm that tries to find an optimal solution by using local search. It picks a random starting point, sees if either neighbor has a better value, and moves in that direction. It keeps moving until it cannot move in a better direction, which means it has hit some local optima. This optima might not be a global optima though. To make sure hill climbing doesn't get stuck, there are random restarts. Once some optima is found, it restarts the search in a random place (further from current) in hopes of finding the global optima. This might take long depending on the distribution of the data. If most of the data lies in a big basin of attraction for the local optima, and the basin of attraction for global is small, even with random restarts, we might still fall in the local optima basin. The advantage to this algorithm is that it is simple, and restarting multiples times is a constant factor, so it is not much more expensive than normal hill climbing. If we keep track of which points we already searched upon, we won't repeat a point if we randomly fall on it again (Tabu search). That way, we won't do worse than just searching all the points to find the global optima.

Simulated Annealing is more robust than hill climbing. SA believes that we can't just always go in the improving (exploit) direction. Sometimes we must search (explore) in negative directions to find a better position. Exploiting gets you stuck because you are almost overfitting since you believe your data so much that given a sample, you always go in one direction. With exploring, you don't believe your data at all, so we want a tradeoff between the two. For a given number of iterations, we sample a point, and jump to that point with some probability. This probability is defined by the temperature (T) and cooling factor. We start our temperature high. When it is high (like random walk), we are likely to accept a neighbor even if it has a bad fitness function with high probability, but as temperature decreases, we accept bad neighbors less (like hill climbing). The cooling factor is how fast we decrease T. When T is high, we don't get stuck at local optima since we jump around big valleys, and even if it goes in a bad direction, it will randomly go back into a better place. Therefore, we want to decrease slowly so that we have a chance to explore. SA usually converges to a global optimum if we start with a high T.

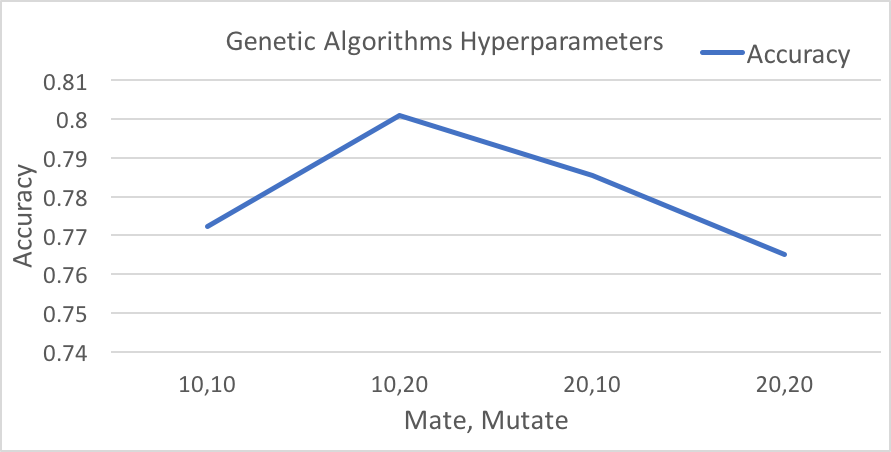
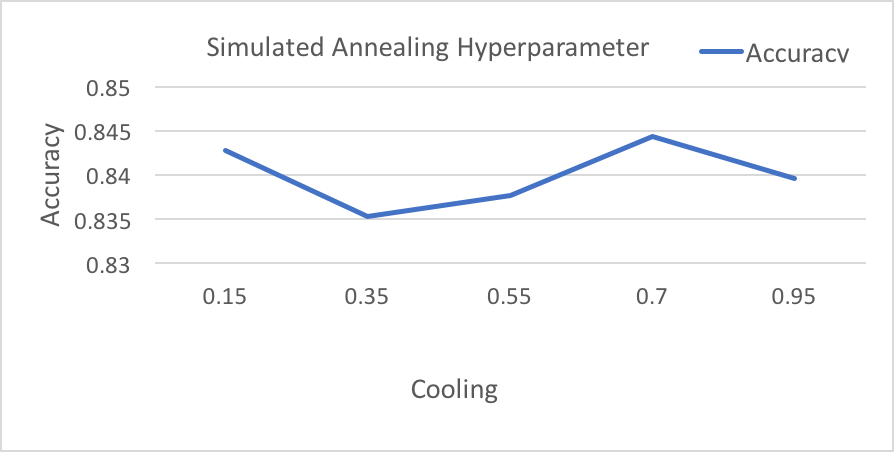
Genetic algorithms take an initial population, finds the most fit (highest fitness scores) individuals, and performs crossover to take the best attributes of both parents together to create a stronger child. Another way GA replaces the least fit is by mutating some of the offspring so that there is diversity and reduces chance of being stuck at some local optima.

The above algorithms don't take their distribution into account as much. SA somewhat does, but MIMIC tries to come up with a better solution where you use the probability distribution of the data to find structure and use that to pick better points. Given a uniform distribution, we generate sample from the population and create a new distribution with the nth percentile, which is most likely to have the global optima. Each time, based on the previous iteration's result, a new distribution is estimated consistent with the best points. Dependency tress help estimate this new distribution by finding relationships between the points so that you can eventually understand the optima. It's somewhat like the relationship for crossover in GA since crossover is trying to represent some structure based on locality. MIMIC can take fewer iterations, but more time for each iteration. The more samples, the more time, since it keeps drawing from those samples to find a best fir for the new distributions. SA takes much many more iterations to converge, but less time for each iteration. Although, MIMIC gives you more information per iteration at the cost of building the distribution, so when the data is complex, MIMIC is better even though there is a trade off with the time.

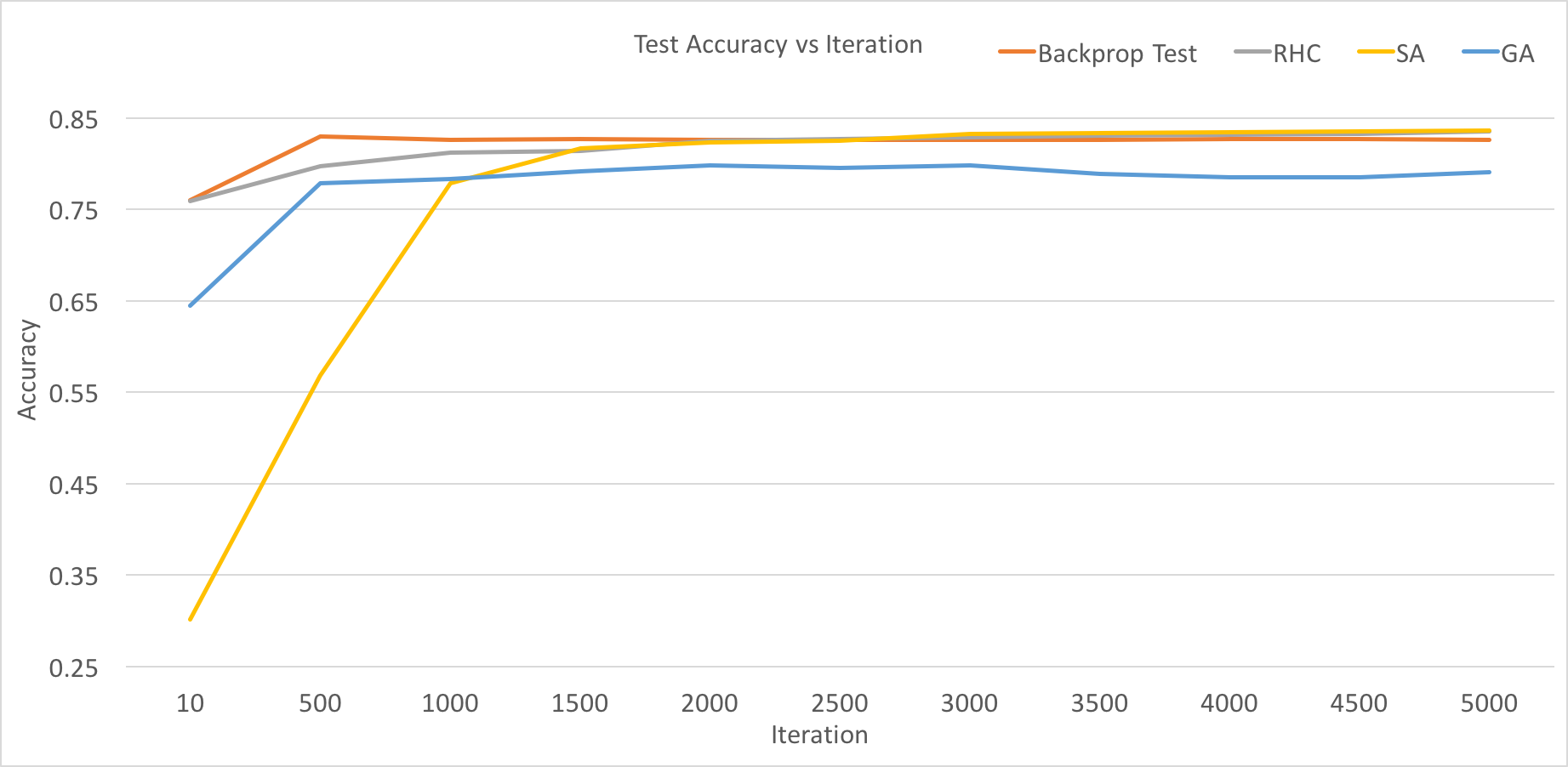
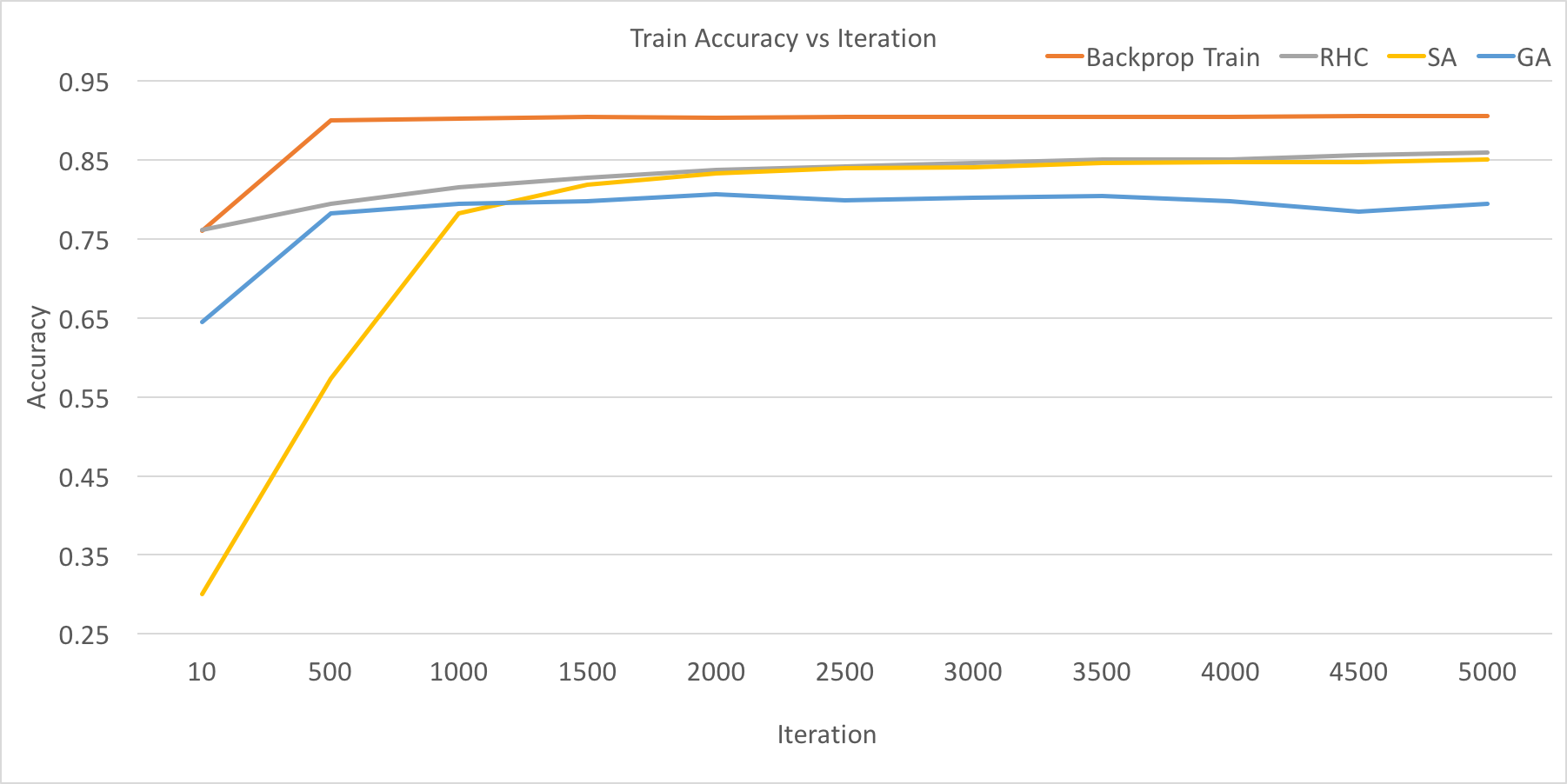
*Part 1 - Finding weights using optimization:*

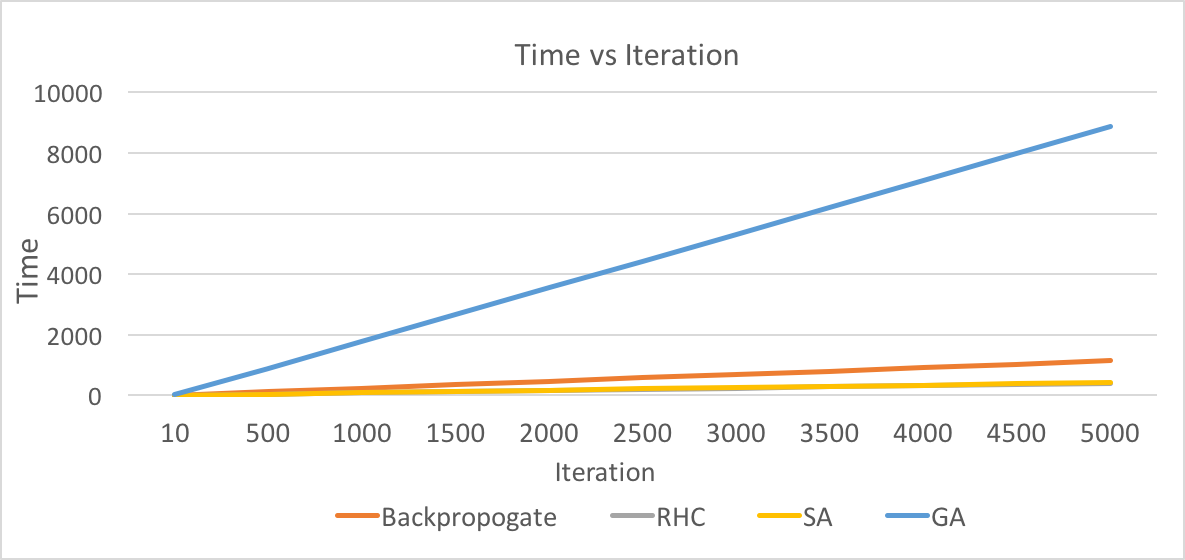
In supervised learning the hidden layer sizes for my optimal neural network for 28 for three layers, and the input layer size was 56, and the accuracy was around 80 percent. I passed the same parameters into RHC, SA, GA, and compared the performance with that of backpropogation. For SA, I kept my T at a high value for all runs, since it makes sense to start with a high T (explained above), given any cooling factor. T was set to 1E10, and I tuned cooling with values: 0.15, 0.35, 0.55, 0.7, 0.95. The number of iterations I ran all algorithms for was 5000. Below is the graph showing how well each cooling factor did. We can see that all the cooling factors performed similar - the difference is miniscule. With a fast decrease in temperature at 0.7, it did just slightly better than a low value of 0.15. It seems like the basin of attraction was not that small for the global optima, that we needed to explore a lot, so decreasing T at a fast rate and exploring less was just as good as decreasing T at a slow rate and exploring less.

For GA, the hyper parameters that I tuned were mate and mutate. Given a constant population sample of 100, I tested how values of mating and mutating performed. Here, mating 10 from each parent and mutating 20 was clearly the best. Too much crossover seemed to take too much randomness into account, and not focus on the best fit individuals, especially since we are again mutating for more randomness. Taking fewer most fit, and mutating a few to generate randomness was the best option.



These parameters were tuned with cross validation, and after fitting the model with the best parameters, the below graphs show the train and test accuracy per iteration, along with time for each iteration. I used cross validation instead of validation because I did not want to waste any data, and cross validation is helpful for reducing overfitting on training data and fitting a better model. We can see that backpropogation performed the best on the training set, but on testing set, both RHC and backpropogation perform similar, which shows that there was a lot of overfitting (almost 10% decrease in test accuracy) on the training data with backpropagation, while the rest of the algorithms had little overfitting. RHC is the winner here. Not only does it take the least amount of time (can barely see the time since it so little compared to the others), but it also starts to converge to the highest accuracy at a smaller iteration than SA, at around 1000 iterations, while SA took 1500. So even if SA eventually performed the same, the amount of extra time it took is important to take into account. Although backpropogation converges faster around 500 iterations, it is better to pick a model that over fits less. Since RHC performs well at around 84% accuracy, this means that we don’t have many local optima to get stuck at, and are able to find the global optimas with decent accuracy and time compared to the others. With almost the same number of iterations both backpropogation and RHC converge to the same accuracy, showing that the basin of attraction for global optima was large enough to find it with a few iterations/restarts. The randomness also helped to restart and find a better location to climb and find global optima. Timewise, GA takes much longer, due to the huge size of the dataset, which means crossing and mutating and can take a while. Therefore, an optimization algorithm found weights for the network in a better manner.



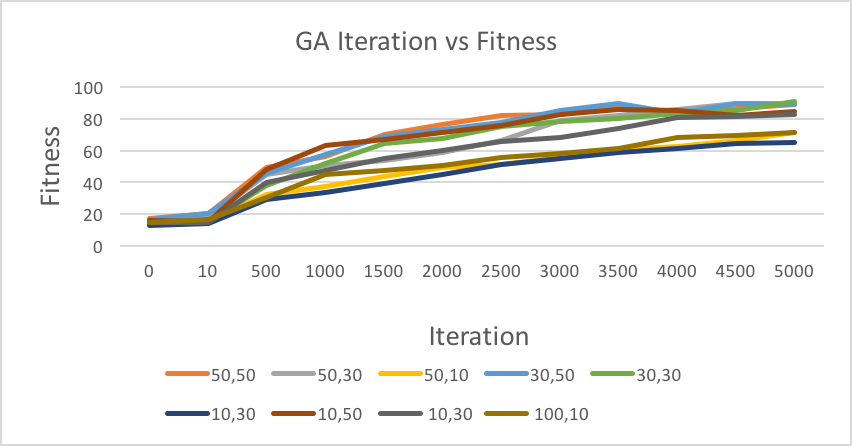
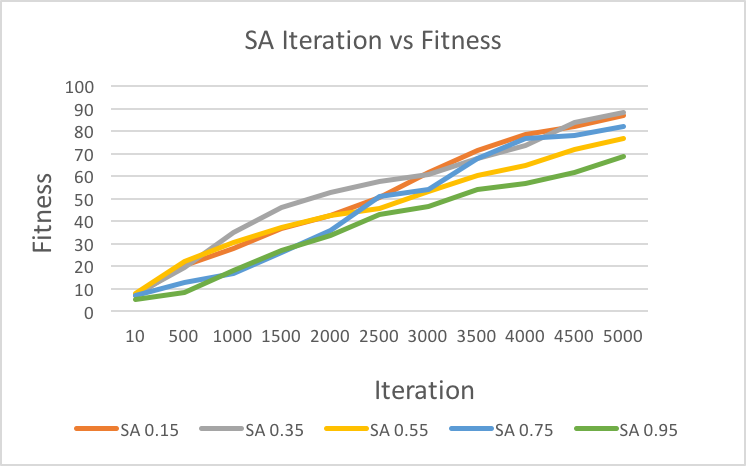


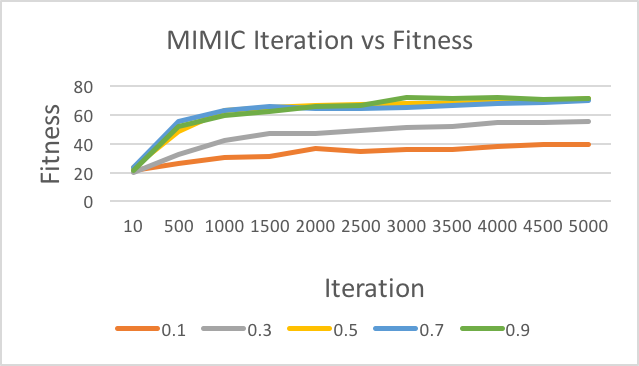
*Part 2 - Optimization Problems with Four Search Techniques*

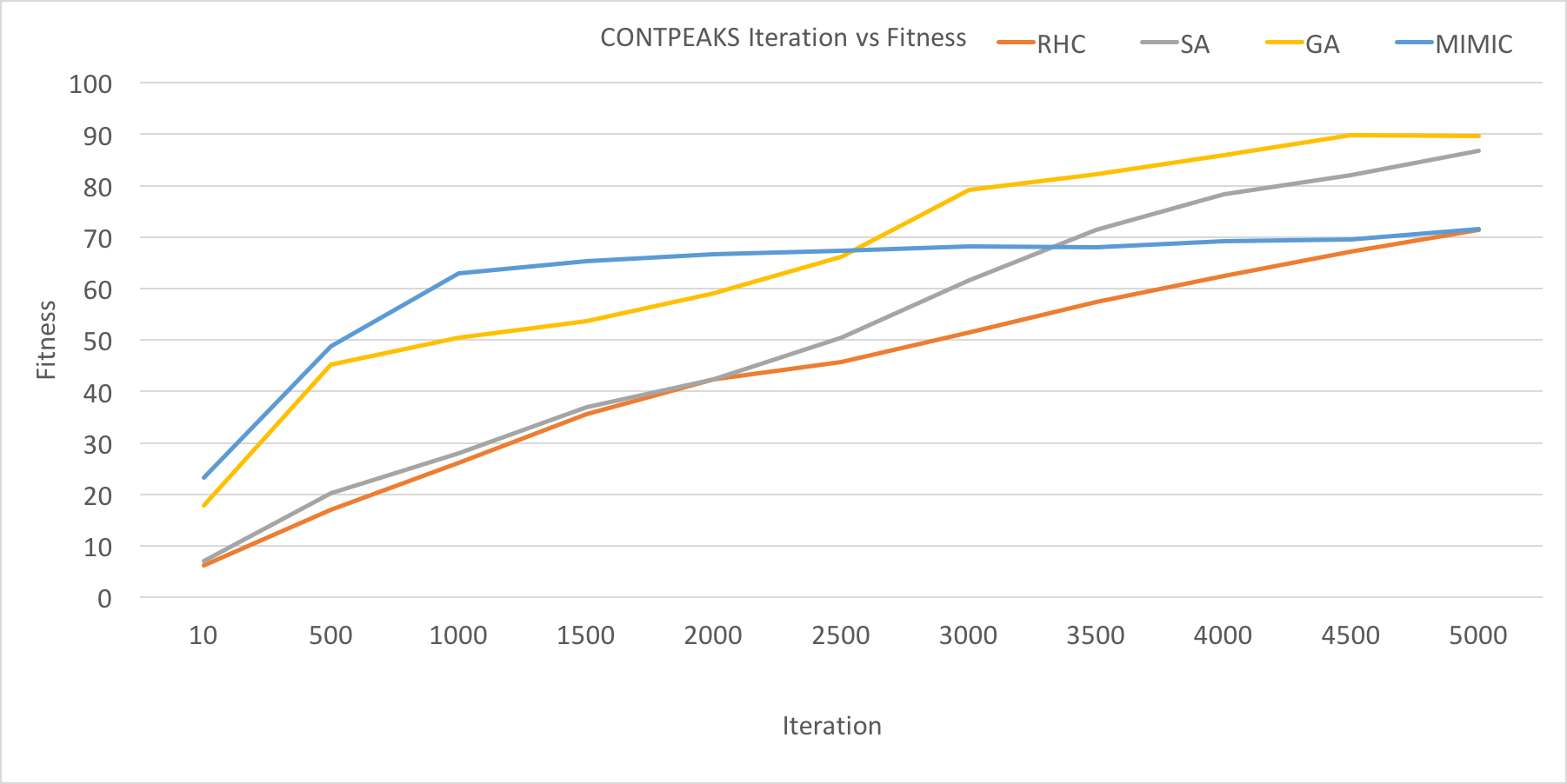
*Continuous Peaks:*

This problem is related to the four peaks problem. This problem tries to achieve consecutive 0s at the beginning, and consecutive 1s at the end of the string, and these sums account for the fitness function. But, if the number of 0s and the number of 1s are above some threshold, then there is a bonus added to the fitness score. So two peaks are when we have lots of 0s, or lots of 1s that result up to a threshold, and these are the smaller peaks, while the two other, and bigger peaks, are the bonus where both 0s and 1s have to pass the threshold. I used a bit string of size 100, and my threshold, T, is 49. I used the same hyper parameters as before for SA, but I introduced some more combinations of hyper parameters for GA to explore and see if it helps for tuning purposes. For MIMIC, I always took 100 samples from the uniform, and kept 50 of those best fit samples, and tuned the Bayesian estimate parameter, with values: 0.1, 0.3, 0.5, 0.7, 0.9. This parameter is used when constructing the dependence tree. It tells me the likelihood of a node, by fitting a correlation with this parameter and the sample’s probability/distribution. All algorithms were run for 5000 iterations.

For SA, the best cooling factor was 0.35. You can see that .15 was not far behind, but increasing the rate more definitely decreased the performance, meaning more exploiting was needed in this case to find global optima’s basin of attraction. For GA, 30 mate and 30 mutate did the best. These values show that mating 50 at a time took in too much randomness, and mating 10 took just did not get enough of the best fit, even after mutating. The best combination was picking 30 and mutating 30. As for MIMIC, both 0.5 and 0.9 for epsilon performed the best. Low likelihood for picking a node of exploration doesn’t help us choose the global optima, so in this case, we had to set it a little higher, and went ahead with picking 0.5 in this case. RHC has no parameters to tune, and has not been shown in any of the problems.



The iteration vs fitness graphs show each algorithm’s performance with the best parameters found for each. You can see that MIMIC and SA did the best for fitness, but for this problem, I am choosing SA as the best optimization problem. GA and RHC perform the worst, although GA performed slightly better because it converges to the best fitness around 1000 iterations, while it took RHC 5000 iterations to do random restarts to be able to find the same optima. Performance wise, GA was better than RHC, but it still could not reach as high a score as SA and MIMIC, since it got stuck at a local optima, and takes a while to find a new string to escape, since it does not understand structure of the problem. RHC and SA both start of similar, but around 2000 iterations, SA starts increasing in fitness at a higher rate. This is because it had a high enough T and small enough cooling factor to explore the whole space and not get stuck when it saw the local optima. Now we compare SA and MIMIC and see why SA is better even though they eventually result in the same fitness score, using Iteration vs Time. MIMIC by far takes more time per iteration, while SA’s is linear. As we said before, this is because we redo the distribution each time before exploring. In this case, SA performs just the same, with the same number of iterations, so we don’t need the time complexity of MIMIC. Even though we would think MIMIC would perform the best by understanding the structure and getting information about all four maxima to pick the best, the time complexity, the fact that SA does use a type of probability distribution, and simplicity of SA given it still resulted in same fitness as MIMIC, made SA the clear winner.

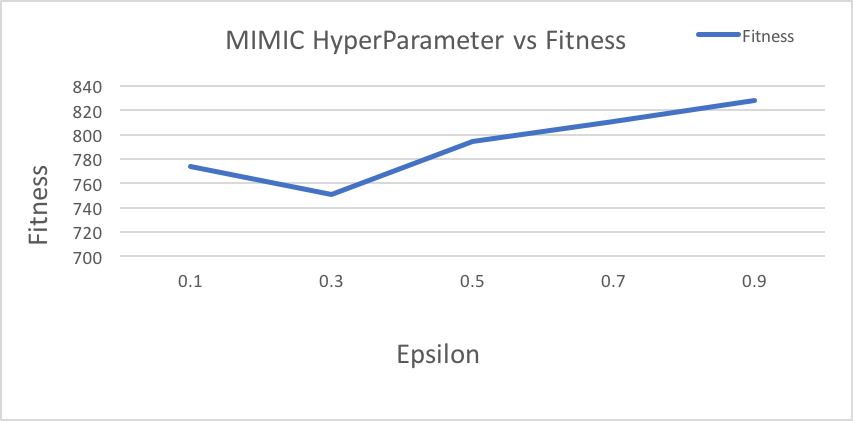
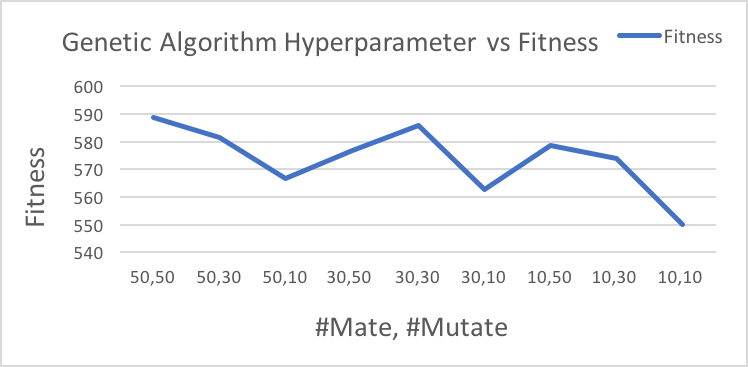
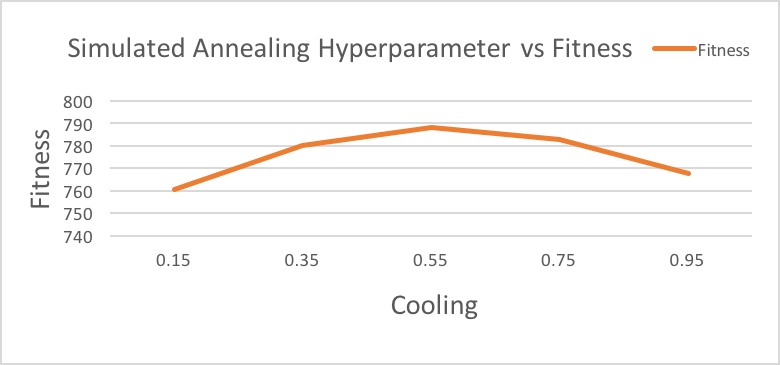




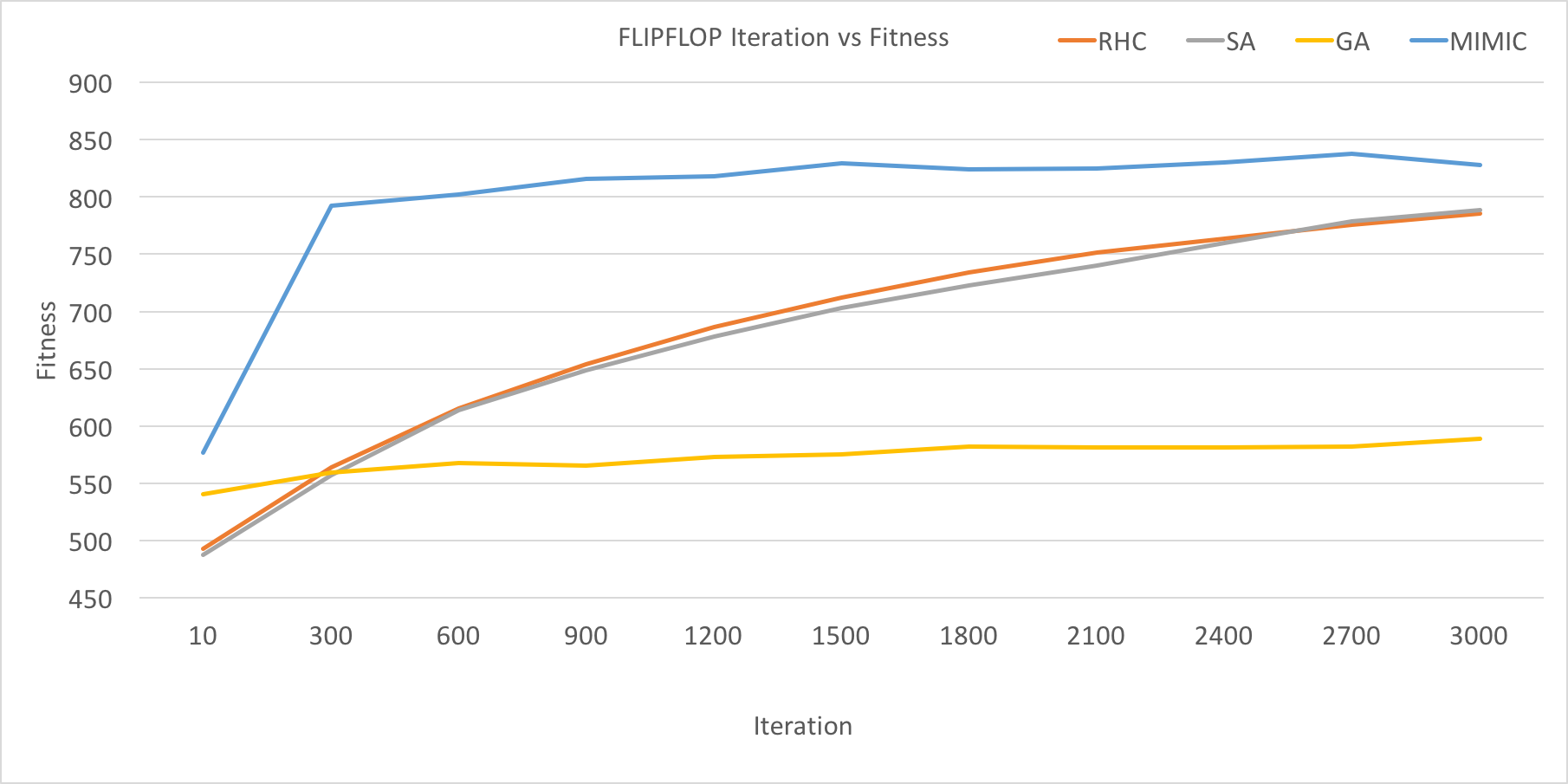
*Flip Flop:*

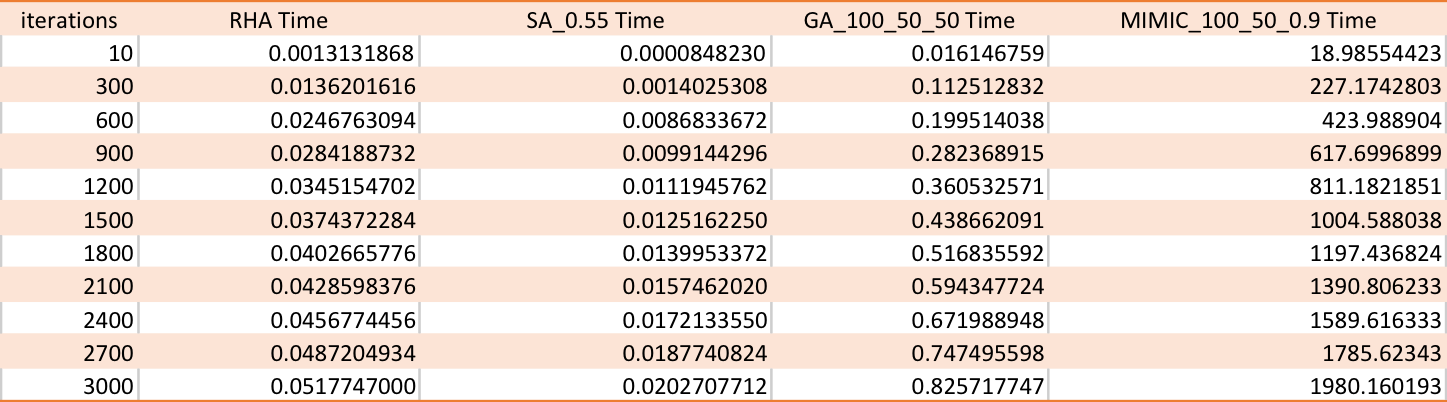
This problem is another bit string problem that uses a fitness function which counts how many times bits are alternating between 0 and 1. Therefore, we might have many local optima in this problem, since the bit string length is huge at 1000, which makes it harder to find the global optima. This problem was run for 3000 iterations given the magnitude of the problem, and the same hyper parameters as continuous peaks were used. For this problem, MIMIC performs the best.

For SA, the best cooling parameter was 0.55. Decreasing the rate of T quickly decreased performance. We are not giving enough exploration times to jump over the local optima valleys, especially when we might have many in such a problem. The algorithm tends to exploit more and believe the data too much that it just keep going in one direction with respect to a sample. Decreasing T slowly also performed bad. This means that the algorithm is exploring too much, and not believing the data enough to give it a chance to slowly climb and converge to optima when it is near it. So the middle of these was perfect. For GA, the highest number of mating and highest number of mutating worked the best, which makes sense considering the size of the bit string. And for MIMIC, and epsilon of 0.9 for the Bayesian parameter when creating the dependency tree, gave the highest fitness score by far. Increasing the likelihood of a node helped find the global optima better.



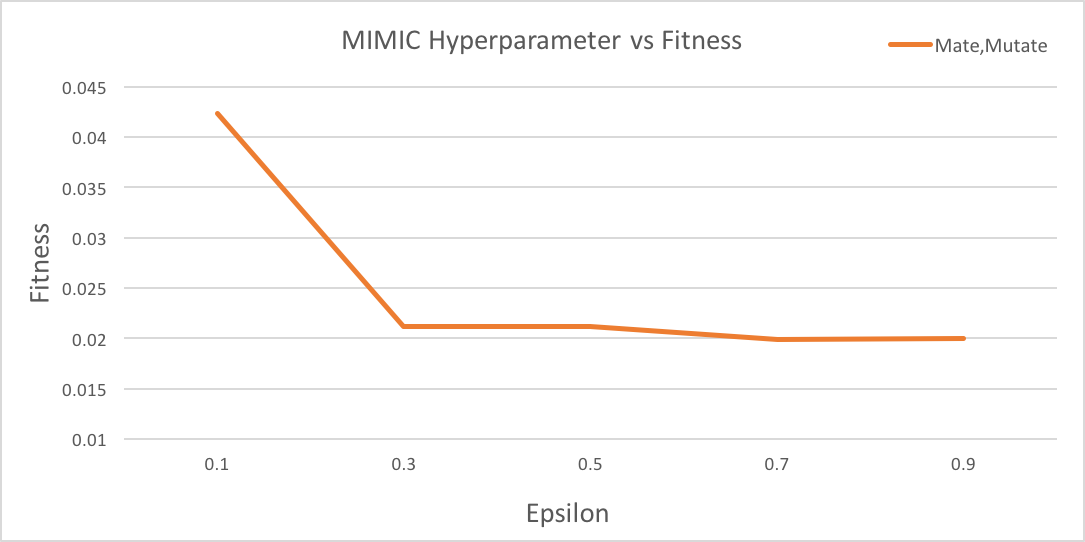
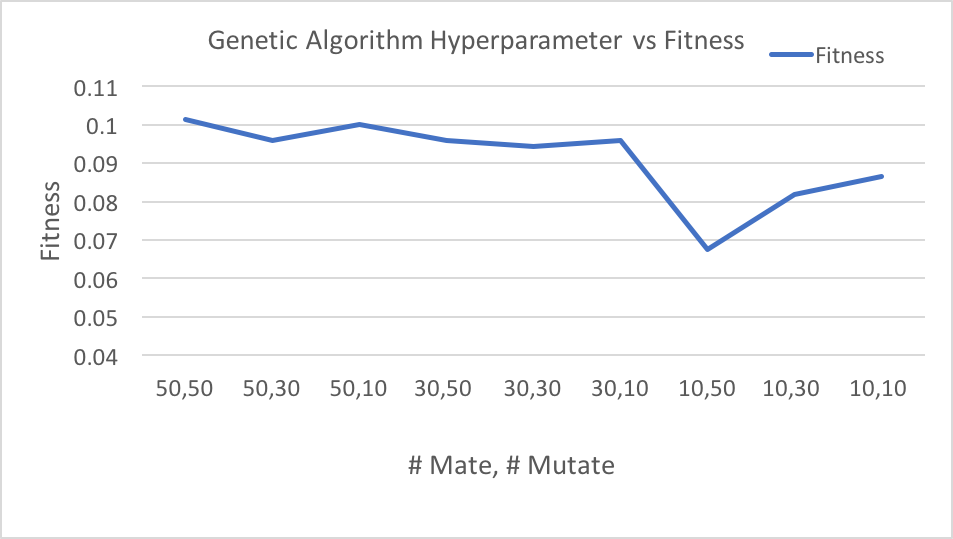
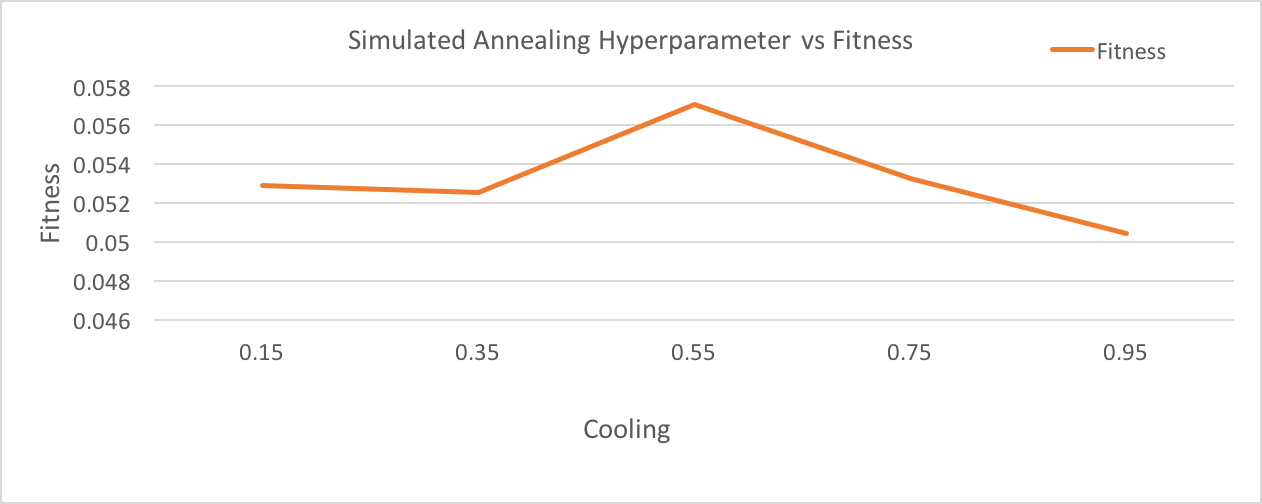
Looking at the iteration vs fitness graph, MIMIC results in the best fitness score around 825, while RHC, and SA result in the same around 790. GA does not do well at all with a fitness score of about 600. With GA, even one mutation can throw the whole problem off. We are dealing with alternating bits, and if we mutate a bit in a way that breaks a string of alternating bits, we don’t necessarily find the best pairing. Even with the fact that there are so many local optima, GA can get stuck with one and not understand that there is a bigger alternating string out there. RHC and SA are perform much better, due to random restarts and exploration, but notice how SA didn’t do any better than RHC even though SA usually tends to preforms slightly better. Due to many local optima, even SA did not find the global optima, and maybe it could do better with more tuning of the hyper parameters. MIMIC performs the best, but it’s worth noticing that it hits a fitness score of 800 around the 300th iteration, and then slowly increases to the highest of about 840 at the 2700th iteration. To get to the same fitness of 800, MIMIC took 1/10 the number of iterations it took SA AND RHC, showing how powerful MIMIC was for this problem. Although MIMIC also takes almost 100k times longer, it gave a similar answer in the same amount of time, showing the tradeoff between time and iteration. MIMIC understands the underlying structure of the problem, which was more necessary when given a, simple to understand, yet complex problem due to the number of local minima.





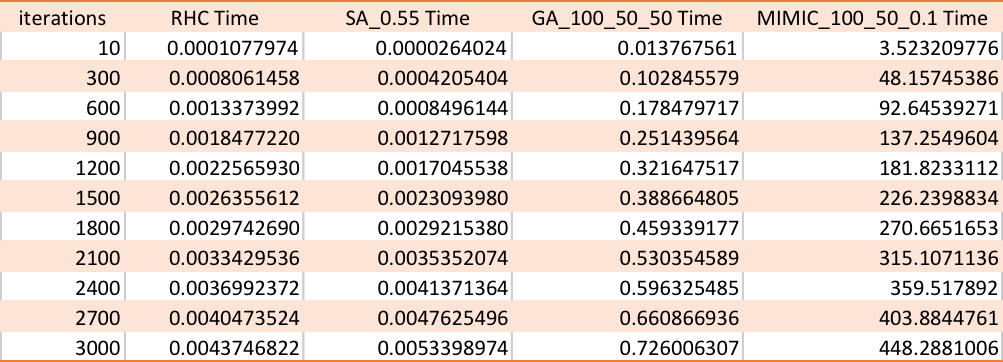
*Travelling Salesman Problem:*

This problem tries to find the shortest path that starts and ends at the same city, and covers all cities once. For this problem, I set the number of cities to 100, and ran the problem for 3000 iterations for each algorithm. The hyper parameters stay the same for each algorithm. In this case, GA performs the best. The best hyper parameters were 50 mate and 50 mutate for GA, showing that larger crossover and mutation values worked better. Although, the fitness is similar up until 30 mate and 10 mutate. Mating just 10 seemed to leave out some of the most fit candidates, and that is plausible given the size of the input problem. Mutation of any value does not make much of a difference here, meaning that adding more or less diversity or noise to the problem does not make a difference as long as we selected the most fit individuals. But, we always need mutation to actually converge in GA, because mutation is like the explore part of SA, for GA. For SA, the best cooling factor is at 0.55. Again, decreasing T fast did not allow enough exploration, and decreasing T too slow did not exploit and believe the data enough. As for MIMIC, a value of 0.1 for epsilon worked the best by far, which is interesting because in the previous problems, a higher value for epsilon worked better. If decreasing the likelihood parameter helped the most, that means that MIMIC is not a great performer for this type of problem, which means that finding the underlying probability distribution is not the most important aspect for this problem.



SA and RCA performed similar, although they gave just a slightly better accuracy than MIMIC. The crossover and mutation parts are important for this problem. Since no city can be visited more than once, crossover make sure that after copying from one parent, the values you are mating with the second parent are not the same. Mutation has to make sure not to mutate a city if it duplicates with another city, so mutation might not be as valued as crossover here. This was shown above with the hyper parameters, where any value of mutation performed similar for the given crossover. If we are determining the best unique combination about a distribution, GA handles that the best since it already combines the population to produce more fit results, but this time in a way without duplicates. GA does take much longer than RHC and SA, but nowhere near the amount of time as MIMIC. For the difference in fitness score compared to RHC and SA, the time complexity is worth it to find the global optima with GA.





*Conclusion:*

In part 1, we looked to see if optimization problems can find better weights for a neural network. Most of the algorithms performed well, although not as great as backpropogation. Although the values are continuous in neural networks, the optimization algorithms were able to work in that space since they essentially just need a sense of which neighbors are better and which are worse. Backpropogation uses gradient information to improve weights, so there is more information about the space than the optimization algorithms. Although, it was interesting to see how RHC performed almost the same as backpropogation, given that it usually performs worse than say, SA. This shows that the dataset we deal with at any point is very important. Given RHC’s performance on this dataset, although the size was huge, local optima were less, and global optima had a huge basin that RHC was able to perform well, relatively fast. This leads into how different optimization algorithms behaved based on the given problem in part 2.

In general, RHC is an inexpensive algorithm for simple problems, but might require more random restarts to not get stuck at the local optima and discover the global optima, so more iterations are preferred. SA incorporates RHC a little, but performs better and usually converges to a global optima. It is also not as expensive, similar runtime to RHC and especially shows good performance where there are a lot of local optima and it can get easy to get stuck at one. This makes it a very optimal algorithm. Using temperature and cooling, SA explores and exploits (Hill Climbing) until it finds the global optima. GA usually takes longer than SA and RHC, but it depends on the complexity of the problem, and GA can also be parallelized. GA are best used when you want to find the best combination for a set of data such as shortest route, which is exactly why GA performed the best for Travelling Salesman. As for MIMIC, it works best for complex problems that need structure in order to better find the global optima. Time complexity is a trade off with MIMIC due to how it resamples the distribution, because MIMIC takes exponentially longer, so it is better preferred for smaller problems.

Given the analysis so far, it makes sense why an algorithm worked best for each problem. For Continuous Peaks, we wanted to essentially find the largest basin of attraction where 0s and 1s pass the threshold. SA was perfect for that because we have many local optima, and need to converge to the global optima. Using MIMIC for a less complex problem was not needed given its runtime, and RHC gets stuck at local optima, while GA performs the least because we are not trying to calculate fitness of each individual and mate/mutate to find a better child. We don’t want the best combination of our data, we just want to see if a certain space of data is large enough, so SA makes sense since it can find the optima with the largest basin, and it did perform the best.

For Flip Flop, MIMIC worked the best. This makes sense because this was a bit more complex problem and we definitely needed structure since we wanted to optimize the number of alternating 0s and 1s. With so many local maxima, MIMIC was needed to get an idea of the underlying distribution in order to better find the global optima. Even though the bit string was long at 1000 bits, the time trade off was worth the fitness.

Finally, for Travelling Salesman, GA is the best algorithm. The time taken was long as expected, but the number of cities was 100, so it makes sense. Given how the problem matches the scope of GA perfectly, it did perform the best. We wanted to find the shortest path between all our data without repeating. Crossover and mutation makes sure to come up with the best combination/child while making sure the same cities do not mate or mutate into each other, resulting in the most fit order of visiting all cities.

REFERENCES

*A. For Dataset:*

UCI Machine Learning Repository: Flags Data Set, archive.ics.uci.edu/ml/datasets/Adult.

*B. For randomized optimization code:*

JonathanTay. JonathanTay/CS-7641-Assignment-1. GitHub, 23 Oct. 2017, github.com/JonathanTay/CS-7641- assignment-2.