Comparison of Randomized Optimization Algorithms

Introduction

This paper explores the randomized optimization techniques including Randomized Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA) and Mutual Information Maximizing Input Cluster (MIMIC). In the first part of the paper, I analyzed the performance of RHC, SA and GA in choosing weights for a Neural Network on the Wisconsin Diagnostic Breast Cancer (WDBC) dataset. Subsequently, I compared the performance of RHC, SA, GA, and MIMIC on the Traveling Salesman problem, Four Peaks problem and Continuous Peaks problem. All algorithms were tested using the ABAGAIL library in Java.

I. Wisconsin Diagnostic Breast Cancer (WDBC)

The WDBC dataset extracts features from digitized image of a fine needle aspirate of a breast mass to describe the characteristics of the cell nuclei present in the image. This is a binary classification problem where each instance is classified into either benign or malignant. In Assignment 1, I built a feed-forward Neural Network to model the WDBC dataset. Particularly, I used backpropagation to update the weights of the perceptron in the Neural Network. It was found that the optimal number of hidden layers for WDBC is 6 with 16 neurons in each layer, yielding 90% test accuracy. Figure 1 indicates the learning curve obtained with backpropagation.

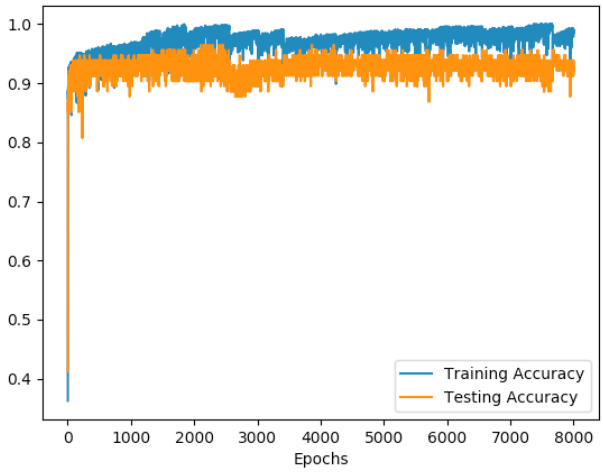

Fig 1: NN with Backpropagation (6 hidden layers)

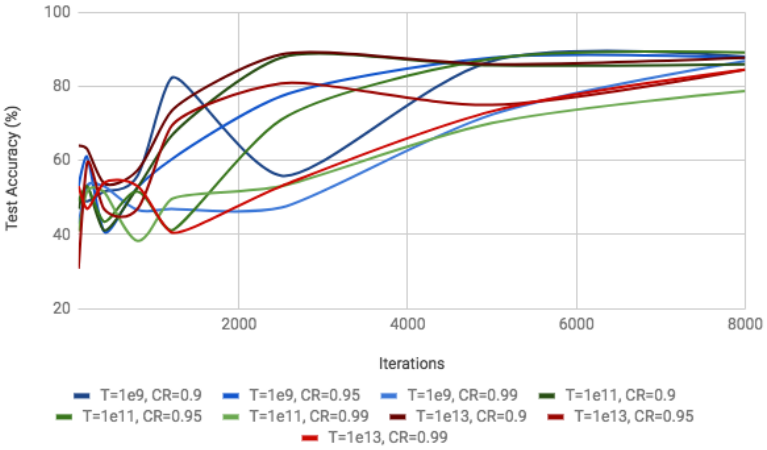

Fig 2: Simulated Annealing with varying T and CR

Simulated Annealing

SA is a stochastic variant of local search that it permits moving downhill in order to allow for a more extensive search for the global optimal solution and prevent from being prematurely trapped in local optima. The extent to which it moves downhill depends on the hyper-parameters initial temperature (T) and cooling rate (CR) of SA. CR defines the rate at which the temperature decreases in each iteration. Lower temperatures influence SA to move towards points closer to the goal; which in turn increases the chances of getting stuck in a local optima. I analyzed the impact of varying T from 1e09 to 1e13 and CR from 0.9 to 0.99 on average test accuracy on 3 trials each of iterations ranging from 200 to 8000. As shown in Figure 2, the chosen T-CR pairs show good differences in performance especially when run for fewer iterations. This is specifically true for T as its impact diminishes over time due to CR. However, choosing an optimal T can potentially help converge to an optimal solution much quicker. In case of WDBC, it was found that lower T generally performed better. However, CR was a noticeably pronounced impact on performance. As shown in Figure 2, lower CR for each set T consistently performs better than higher values of CR. This is because a lower CR helps allows for a more gradual and extensive search of the global optima. The examples shown on the figure demonstrate the randomness of SA especially on fewer iterations. For instance, (T=1e9, CR=0.9) is able to obtain test accuracy of 82% at iterations but declines to lower accuracies on further iterations, eventually leading to a test accuracy higher than the initial 82%. This shows SA's tendency to move towards worser solutions in order to explore the solution space more extensively. At iterations of about 1200 or less, we notice that the test accuracy tends to oscillate but eventually all converging towards accuracies close to 80% at 8000 iterations. Therefore, careful choice of T-CR pair is critical in obtaining good performance at fewer iterations. Although most of the T-CR combinations chosen eventually reach similar levels of accuracy after many iterations, we are interested in a pair that can perform well in relatively smaller number of iterations; therefore, we choose a T = 1e9 and CR

= 0.95.

Genetic Algorithm

Genetic algorithm is a randomized search technique inspired by natural selection. It stems from the idea of mutation and crossover/mating of population samples (in this case, weights of the NN) forming generations to evolve to an optimal state. Each generation is evaluated on its fitness. In case of WDBC, we use test accuracy as fitness. A key benefit of GA is that it can easily move out of local optima due to the nature of genetic diversity as a result of mutation and crossover. However, this can potentially be a disadvantage as GA may converge to arbitrary points rather than the global optimum. There are 3 hyper-parameters that are require tuning: population size (P), crossover (C) and mutation (M). I chose to test P of size 100 and 200, C of size 60% and 90% of P and M of size 1.5%, 5% and 10% of P. As GA takes significantly longer time relative to other algorithms, I ran it for iterations ranging from 200 to 800, each with 3 trials. The results obtained are shown in figure 3 below. There is clear distinction in that larger values of P outperform smaller P by a large margin in case of WDBC; a difference of almost 20-30% in test accuracy at 400 iterations. We also infer that a larger P converges to the optimal solution much faster at 400 iterations. This is intuitive as searching over a larger sample space in each population allows enables GA to explore a larger search space in each iteration. There is another clear winner in that smaller ratios of C generally do better as shown by the darker shades of blue and red in figure 3. Similarly, as seen by the 2 dark red curves (P=200, C=120), it is evident that higher values of M perform better leading to a difference in test accuracy of 5% at 800 iterations between M=4 and M=20. Therefore, mating smaller number of instances and mutating/combining higher number of instances post-mating leads to better test accuracy. Therefore, WDBC tends to perform well with less aggressive mating and more aggressive mutations. The results of my experiment also demonstrates GA's ability to get out of a local minima as seen in case of (P=200, C=120, M=4) and (P=200, C=120, M=20); the test accuracy quickly rises from about 40-45% to 90% when going from 200 to 400 iterations. Similarly, the examples I have chosen to share also demonstrate the flipside; i.e. the potential of leaving a good local optima to worse points as demonstrated by (P=100, C=60, M=2) going from 82% test accuracy at 200 iterations to 57% test accuracy at 400 iterations. However, (P=100, C=60, M=2) is able to reach 83% accuracy at 800 iterations. This demonstrates that GA generally tends to result in oscillating test accuracies at fewer iterations. As it runs more iterations, test accuracy tend to become higher; thus, demonstrating the randomness of the algorithm. In case of (P=100, C=60, M=2), it may have been so that it got lucky at relatively fewer iterations. Since we are interested in achieving high test accuracy at relatively fewer iterations, we choose P=200, C=10, M=20 as the optimal hyper-parameters.

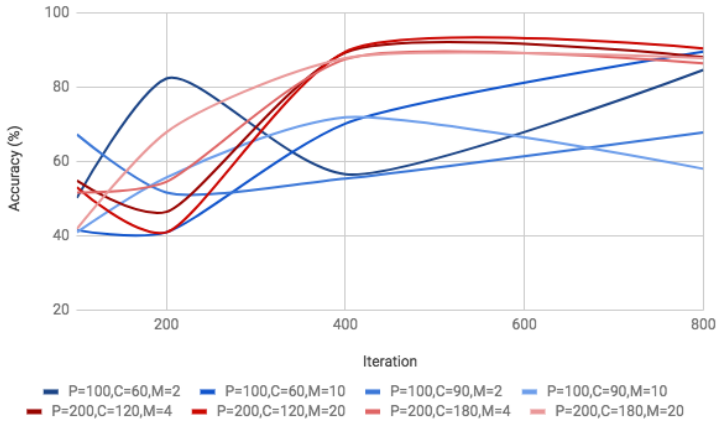

Fig 3: Genetic Algorithm on WDBC with various values of P, C, M.

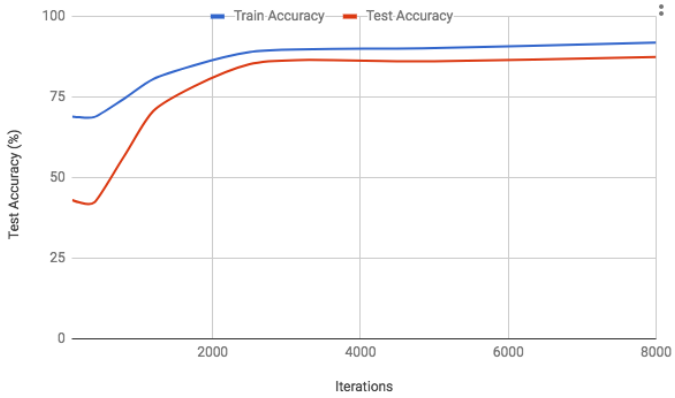

Fig 4: Learning Curve of Randomized Hill Climbing on WDBC

Randomized Hill Climbing

The last algorithm run on WDBC is randomized hill-climbing (RHC). RHC conducts a series of hill climbing searches from randomly generated initial states, running each until it halts or makes no discernible progress (Russell & Norvig, 2003). RHC starts by evaluating a random point and tends to move towards better neighboring points until it reaches a point such that none of its neighbors are better. One of the risks of using RHC is that it often tends to get stuck at local optima and is not suitable for problems that may have a lot of local optima. In order to prevent getting stuck at such non-optimal states, we use random restarts to ensure that we extensively search for the global optima from several random starting points. Therefore, a one key hyper-parameter in RHC is the number of iterations, which defines the total number of restarts. As shown in Figure 4, I ran RHC from 100 to 8000 iterations with 3 trials for each . We notice that it converges to a test accuracy of about 85% at 2500 iterations. However, the test accuracy never seems to be as high as training accuracy,

which goes up to about 90%. This is due to the random nature of RHC. It solely relies on starting at good random points to obtain the global optimum.

Having chosen the optimal hyper-parameters for each of the algorithms, we can now compare the performance of each with respect to WDBC. As shown in the log-scale graph learning curve in Figure 5, it is clearly evident that GA is the winner in achieves a test accuracy of 90.5% in just 400 iterations while SA achieves an accuracy of 89% in 8000 iterations and RHC achieves 87% accuracy in 8000 iterations. GA is clearly able to reach very high accuracies on WDBC in a very short amount of time. Surprisingly, RHC performs better than SA in fewer iterations although SA outperforms RHC at 8000 iterations. It is likely that with a high number of iterations, RHC is able to search the solution space quite exhaustively. In case of SA, we notice the wavy characteristic of its learning curve especially until about 1000 iterations; thus, it demonstrates that SA needs to be iterated on longer in comparison to RHC and GA to find optimal solutions. However, over time, SA can obtain accuracies that are satisfactory. However, it is clear that using backpropagation in Neural Networks can lead to optimal solutions very quickly. As shown in Figure 1, it is able to converge to an accuracy of 90% in just about 15 iterations, which is very impressive. Although SA, GA and RHC are able to reach accuracies close to 90%, it requires significantly more iterations.

Fig 5: Test Accuracy of SA, GA and RHC on WDBC.

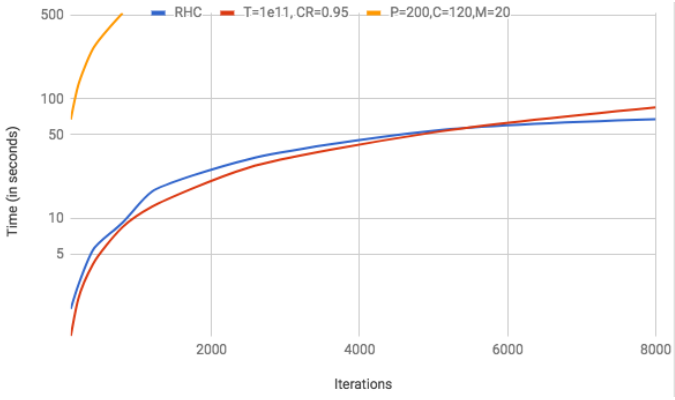

Fig 6: Training Time of SA, GA and RHC on WDBC

Although we have established that GA is certainly better than SA and RHC while SA is better than RHC in the long run, it is important to compare the wall clock times of each. As shown on the log scale time graph of each algorithm in Figure 5, we observe that GA takes exponentially longer than RHC and SA. While GA is able to achieve 90.5% accuracy in about 500s, RHC is able to achieve 87% accuracy in 82.4s and achieves 89% accuracy in 85s. Although GA achieves an overall accuracy much higher than SA and RHC, it lacks in efficiency. However, it's important to note that GA is faster with a smaller population size, crossover and mutation. However, the accuracy tends to suffer with subpar hyper-parameter values. Therefore, time to convergence is exceptionally high in case of GA with the optimal hyper-parameters. GA is particularly time-consuming for high-dimensional problems with large number of instances. As WDBC contains 30 features, GA performs quite poorly. All things considered, we come to the conclusion that SA performs reasonably well on high-dimensional datasets like WDBC with a relatively good time to convergence.

II. Traveling Salesman Problem

TSP is an NP hard optimization problem in which the key objective is to determine the shortest possible route to all cities on a scattered plot in a coordinate plane. This is an NP hard problem, in which given N cities, there are (N-1)! possible routes. This is particularly interesting problem as small changes in location of the points on the scattered plot can lead to vastly different optimal routes. The fitness of each algorithm is determined by 1/distance traveled; therefore, a higher fitness implies shorter distance traveled. I first compared the performance of RHC, SA, GA and MIMIC on TSP with N = 50.

As discussed earlier, the key hyper-parameter in RHC is the number of restarts performed in the algorithm while hyper-parameters for SA are cooling rate (CR) and initial temperature (T) and those of GA are crossover (C) and mutation (M). In this section, we will analyze the strengths and weakness of MIMIC as well. MIMIC is unique from the rest in that it explicitly passes on information about fitness and mistakes as it progresses to higher iterations. There are 2 key hyper-parameters that need to be tuned for MIMIC: first is sample (S), which denotes the number of samples to generate from the distribution each iteration; second is to_keep (K), which denotes the number of samples to keep after each iteration. As MIMIC passes on information about fitness to higher iterations, it is computationally more expensive and takes up more time. Consequently, I ran the MIMIC algorithm on TSP with S values varying from 100 to 250 and K varying from 10 to 90. I ran the algorithm for iterations between 100 and 2500 to study its effectiveness. The results obtained are as shown in Figure 7 below. It is clearly evident that all instances with low K values obtained higher fitness than with higher K values. Although it has a relatively subtle impact, higher S values generally perform better on TSP. This is intuitive as higher the

number of sample, MIMIC has more instances to generate from the distribution in each iteration. The fact that lower K values perform better further indicates higher randomness in each iteration. Another key observation is that MIMIC converges to the optimal fitness of about 0.18 (in case of S=250) in relatively fewer iterations at 1000 iterations. Thus, we come to the conclusion that for number of cities set to 50, the optimal hyper-parameters for MIMIC is S=250 and K=10 in TSP.

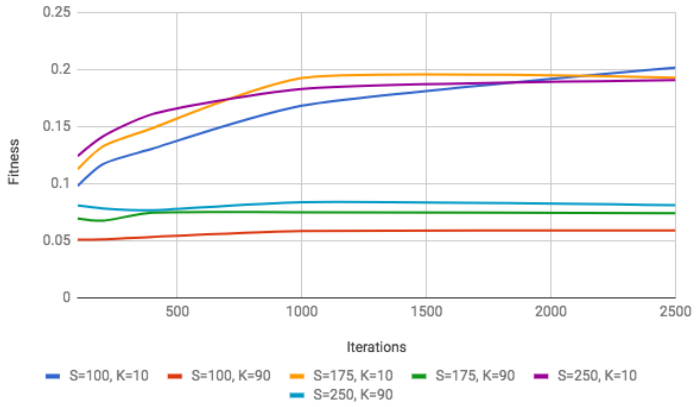

Figure 7: Fitness of MIMIC with varying S, to_keep

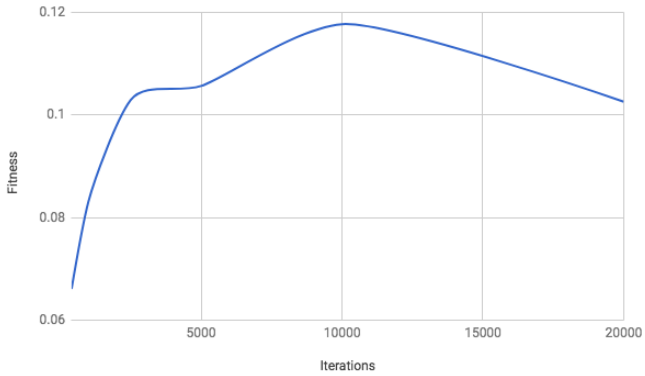

Figure 8: Learning curve of RHC

Consequently, I ran the RHC algorithm on TSP with iterations ranging from 100 to 20,000 with 3 trials for each. Since we are analyzing randomized algorithms, it is critical that we run each setting with multiple trials and take the overall average for each iteration. This will attribute for lucky/lucky situations. The fitness of RHC on TSP is obtained as shown in Figure 8. We observe that although the fitness reaches its peak of 0.119 at 10,000 iterations while decreasing to 0.103 at 20,000 iterations. This particularly highlights the randomness of the RHC algorithm. It could perhaps mean that inspite of a higher number of iterations (20,000), the RHC algorithm didn't encounter the potentially better point starting point that it did in case of 10,000 iterations. This highlights the critical issue with greedy algorithms such as RHC; that is, even with a relatively high number of iterations, RHC is not guaranteed to converge on certain problems.

As described earlier in the paper, I followed a similar approach to obtaining the optimal hyper-parameters for SA and GA for the TSP problem by running each algorithm between 100 and 20,000 iterations with 3 trials for each. I tested values of initial temperature (T) ranging from 1e8 to 1e13 and value of cooling rate (CR) ranging from 0.8 to 0.99. It was found that T=1e13 and CR=0.8 were the optimal values of hyper-parameters on TSP. Consistent with the previous findings of the paper, T has a somewhat minimal impact on fitness while lower values of CR converge to the optimal solution much faster. Furthermore, I took a similar approach to obtaining the optimal hyper-parameters for GA. I tested GA with crossover (C) values ranging from 100 to 180 and mutation (M) ranging from 3 to 15 on a population size of 200. The results obtained are as shown on a log-scale graph in Figure 9. We notice that the fitness for all C-M pairs oscillates between +/- 5% fitness. Take for instance C=100, M=3 on the plot, we notice that it obtained a relatively high fitness of 0.155 on 100 iterations and declined on progressive iterations to 0.140. This demonstrates both a relative strength of GA in that it can easily move to a global optima due to the nature of genetic diversity; however, at the same time, it move to worse points due to the same phenomenon. Based on the graph in Figure 9, crossover, C=180 and mutation, M=3 were the optimal values of hyper-parameters on TSP.

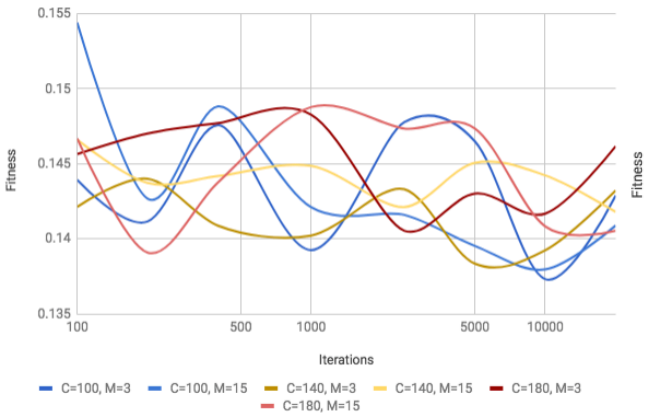

Figure 9: Fitness of GA with varying crossover and mutation

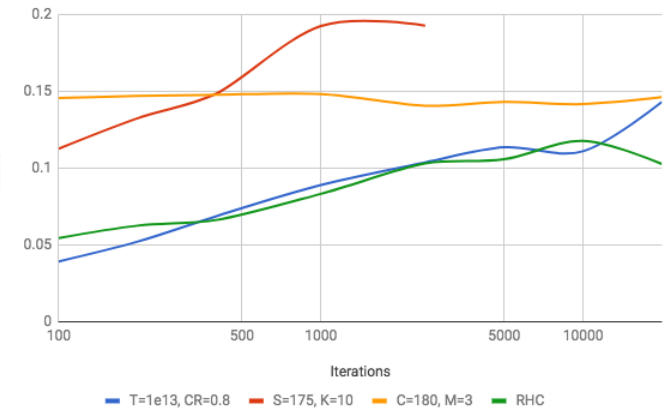

Figure 10: Comparison of all 4 algorithms

Based on the optimal hyper-parameters obtained, we now compare the performance of RHC, SA, GA and MIMIC as shown on a log-scale graph in Figure 10. It is evident that MIMIC outperforms all other algorithms as it reaches a fitness of about 0.195 at just 1000 iterations. It reaches the global optima much sooner than all the other algorithms. However, its fitness tends to drop at 2500 iterations. This is because it is hard to model the structure of TSP since routes can vastly differ based on slight changes to the positions of the cities. It is interesting to note that the performance of GA remains the same throughout; that is, a higher number of iterations don't improve its fitness. However, it is impressive that GA maintains a fitness score at a very low iteration of 100 in TSP such that it scores higher than MIMIC on smaller iterations. On the other hand, we observe that the SA and RHC perform the worst on TSP. However, both of these algorithms exhibit a relatively linear relationship between fitness and the number of iterations. This is because greedy algorithms like RHC and SA have low induction bias and will end up sampling a larger solution space give a large number of iterations. This explains the jump in fitness of SA at 10,000 iterations. Therefore, they require a large number of iterations to do well and are better at problems with smaller solution spaces.

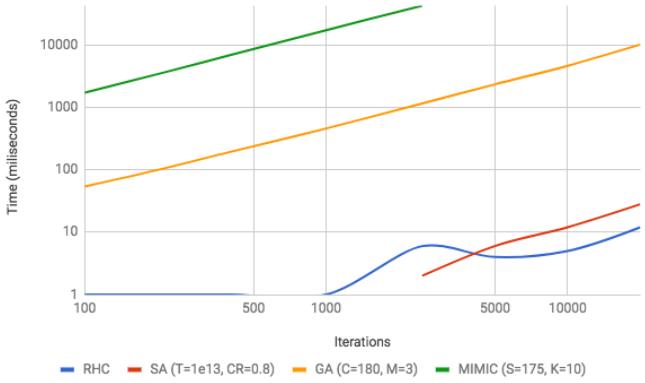

Figure 11: Runtime of all Algorithms

The log-scale time-graph of 4 algorithms on the TSP problem with N=50 is shown in figure 11. It is clearly evident that the time taken by MIMIC is few orders of magnitude higher than RHC and SA. This is understandable due to the very design of MIMIC in that it passes on information about fitness in each stage, which is what makes it more accurate. We also take note that GA takes up a considerable amount of time in comparison to SA and RHC. Considering the relatively high fitness of 0.15 obtained with SA on 20,000 iterations in just 30ms, it certainly performs better than GA in this problem, which takes about 100ms to reach the same level of fitness at 100 iterations.

However, SA is unlikely to perform well on a large N as discussed earlier. We take note that GA has a relatively high fitness throughout, especially at low iterations. This makes GA unique and highlights the fact that it is superior in comparison to random sampling. It's relatively good performance is because it models relationships between the attributes to arrive at better solutions.

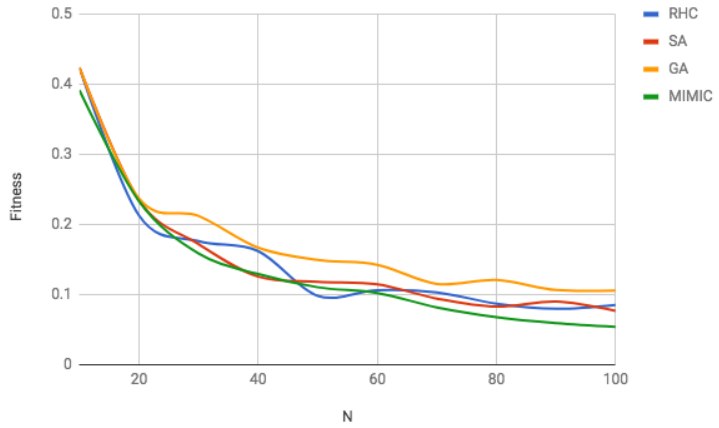

Figure 12: *Fitness of all Algorithms with varying N*

A key factor that determines GA's utility is the crossover function. In this case, the function asserts that the any sub-path in a shortest path is also the shortest. Crossovers in GA also help model the structure of the problem from previous sampling. This, I believe, gives it better performance over the other algorithms. However, it's also important to consider the performance of each of these algorithms when the number of cities (N) is varied. I ran all 4 algorithms on TSP with N ranging from 10 to 100 as shown in Figure 12.

Here, we confirm our earlier assertion that GA tends to perform better as N becomes larger as it tries to model the underlying relationship between the attributes. We also notice that SA suffers as N grows as shown in case of N = 100. This is due to the random nature of RHC and SA; each of these are greedy algorithms that randomly search over the entire. As a result, RHC and SA reach optimal solutions at a much later point as the solution space grows.

III. Four Peaks Problem

The next problem I will analyze is the Four Peaks problem, which is an optimization problem that contains 4 peaks, of which 2 are global maxima and 2 are local maxima. Given a string, the global maxima is achieved when there are T+1 leading 1's followed by all 0's or when there are T+1 trailing 0's preceded by all 1's. In the event that a string contains all

1's or all 0's, we reach the suboptimal local maxima. This is a difficult problem for large T values, as the basin of attraction for the inferior local maxim become larger.

I ran all 4 algorithms on Four Peaks with total inputs N=200 and a T value of 40 for iterations between 100 and 20,000 in order to obtain the optimal hyper-parameters for each. For SA, I obtained optimal initial temperature, T = 1e11 and cooling rate (CR) = 0.8. For GA, I obtained optimal crossover, C = 180 and mutation, M = 15 while for MIMIC, I obtained optimal sample size, S = 250 and keep, K = 20. Figure 13 below is a log-scale representation of the number of iterations against the fitness of each algorithm.

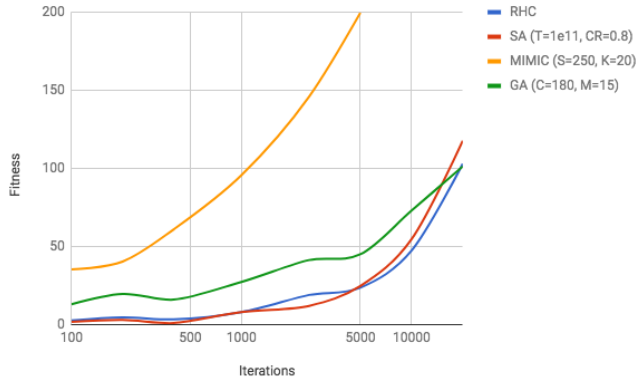

Figure 13: *Performance of all algorithms on Four Peaks*

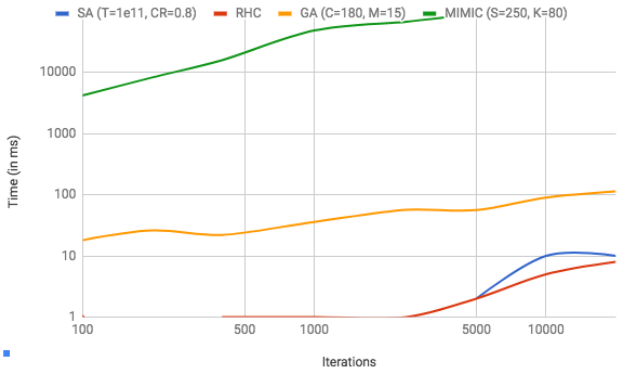

Figure 14: Runtime on Four Peaks

In Four Peaks, MIMIC overwhelmingly outperforms all other algorithms as it achieves a fitness score of 100 in just about 1000 iterations in comparison to SA, RHC and GA all of which only reach the same score at 20,000 iterations. Therefore, MIMIC is able to achieve the same level of performance in just $1/20^{th}$ of the iterations. Furthermore, the rate at which MIMIC's fitness score improves is exponential in comparison to the other 3 algorithms which improve rather linearly on a log scale. However, MIMIC can possibly take a lot of time to run. In Figure 14, we show the time it takes to run each algorithm on the Four Peaks problem. It is evident that RHC and SA take the least amount of time at about 10ms even for 20,000 iterations. GA comes next at about 100ms for 20,000 iterations while MIMIC takes the most amount of time at above 10,000ms to converge to the optimal solution. This is due to the complex operation passing information about fitness in MIMIC. However, it easily outperforms the rest due to the exponential rate at which it gains fitness. The success of MIMIC lies in that searching over a uniform distribution of random variables allows it to narrow down the solution space to the desired peaks. Unlike TSP, there are only 4 optimal solutions out of the total 2^N possible bitstrings.

Next, I will compare the performance of each of these algorithms on inputs, N = {25, 50, 100, 200} on iterations 100 to 20,000 with 3 trials for each iterations. The findings are presented in Figure 15 below on a log-scale graph.

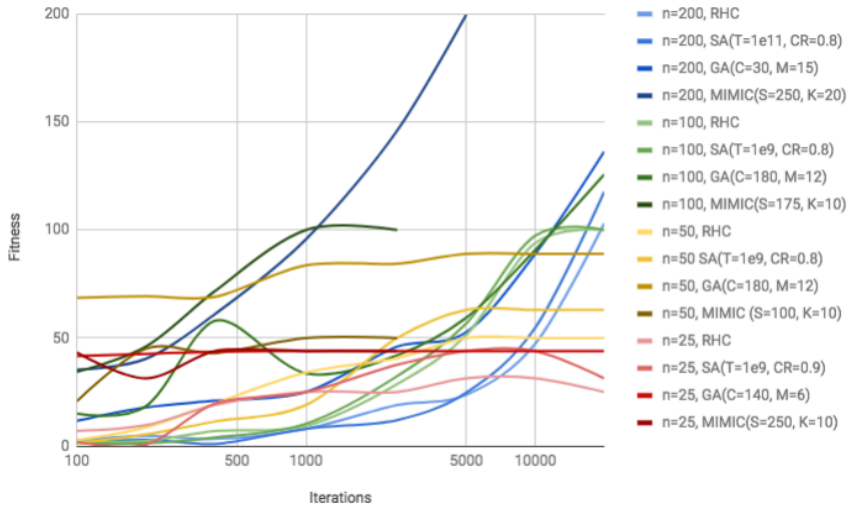

Figure 15: Learning Curve of all Algorithms with varying N

Here, each color denotes an input size (such as blue for n=200) and shade of darkness represents the algorithm type. We observe that MIMIC clearly outperforms all other algorithms at all input sizes, especially when N is large. Therefore, MIMIC performs at an exponentially better rate than the other algorithms as we increase the input size. This reconfirms my intuition that MIMIC works the best on the Four Peaks problem as it consistently performs well when input N >= 50. Furthermore, an interesting observation is that for N <= 100, the performance of RHC and SA (two lighter shades of all colors) are very similar.

However, SA starts to perform a little better than RHC when the input size is increased. We observe that the fitness of MIMIC at N=25 (red curves) is not much higher than the other algorithms. In fact, GA comes quite close to MIMIC at such small inputs. We notice that this difference widens at N=50 (yellow curves) and it widens even more at N=100 (green curves). This gives us the intuition that MIMIC is certainly the most suitable algorithm for problems with large input sizes due to its ability to pass information about fitness. This conclusion does have its limitation as MIMIC takes up much exponentially longer time on large input sizes. However, the Four Peaks problem is one that becomes very difficult to solve with large input sizes as we can see from the dismal performance of RHC, SA and GA on input size=200 to the extent that these 3 algorithms may never maximize the 4 peaks until very high iterations. Therefore, MIMIC is suitable to such complex problems with large input sizes as it is able to structure the problem based on a uniform distribution of random variables. However, it is noteworthy that GA is likely to do sufficiently as well as MIMIC in a shorter amount of time on smaller input sizes. Therefore, GA comes in second in the Four Peaks problem.

IV. Continuous Peaks Problem

The Continuous Peaks problem is a variation of the Four Peaks problem. Instead of constraining the 1's and 0's of the bit string to be at the ends of the solution string, they can be present anywhere in the string. Therefore, we reach a solution when there are more than T contiguous bits set to 0 and 1.

I ran all 4 algorithms on Continuous Peaks with total input bit string N=60 and a T value of 15 for iterations between 100 and 20,000 over 3 trials for each iteration in order to obtain the optimal hyper-parameters for each. For SA, I obtained optimal initial temperature, T = 1e13 and cooling rate (CR) = 0.8. For GA, I obtained optimal crossover, C = 40 and mutation, M = 15 while for MIMIC, I obtained optimal sample size, S = 150 and keep, K = 20. Figure 16 below is a log-scale representation of the number of iterations against the fitness of each algorithm.

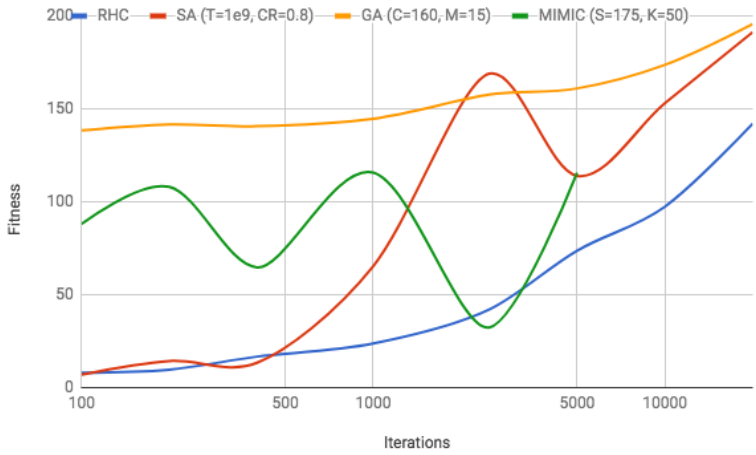

Figure 16 Fitness of all Algorithms on Continuous Peaks

Unlike the TSP and Four Peaks problem, MIMIC surprisingly underperformed on the Continuous Peaks problem. It is evident that the MIMIC gets stuck on local maxima on complicated problems such as the Continuous Peaks problem, as seen from the repeated oscillated in its fitness as it progresses with higher iterations. This shows that is repeatedly oscillating between the similar sub-optimal solutions at each iteration. As expected, the performance of RHC is also rather dismal on this problem. This is mainly due to the large solution space of the problem.

On the other hand, we notice that SA gains in fitness score quite quickly as it rises from near-0 fitness at 100 iterations to a score of 170 at 2500 iterations. However, we notice that the fitness score declines at 5000 iterations and goes back up to about 190 on the 20,000th iteration. This indicates the cooling mechanism of SA which enables it to evaluate worse points so it can exhaustively search the solution space.

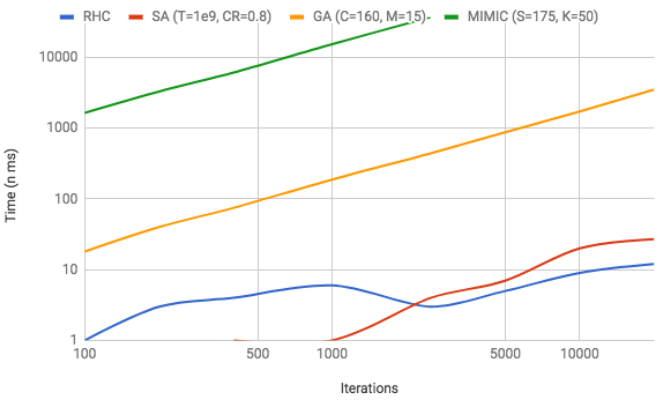

Figure 17: Runtime of all Algorithms on Continuous Peaks

This is both a key benefit and disadvantage of SA. The fact that it is able to exhaustively search the solution space means that it will eventually find the optimal solution but with higher number of iterations when the solution space is very large. GA, on the hand, performs very well on this problem. We observe that it starts out with a very high fitness score of 140 at 100 iterations and gradually rises to a fitness score of 195 at 20,0000 iterations. This shines light on the benefit of the mating-mutation mechanism in GA. This proves that GA tends to particularly perform well when there are multiple local optima in the solution space and particularly performs well on large input sizes (in this case, the input string was set to length 120).

Although GA is not as computationally expensive as MIMIC, it does have a significantly higher runtime than SA and RHC as shown on the runtime graph of all 4 algorithms for 100 to 20,000 iterations in Figure 17. As expected RHC and SA take minimal time at about 40ms to reach optimal fitness. Given that MIMIC has performed the worst on this problem and it's excessively high run time as shown on the graph, MIMIC clearly loses this one. GA's runtime goes up from 20ms for 100 iterations to 5,000ms for 20,000 iterations. SA reaches a similar fitness score in just 40ms. That is a difference of 125x; thus, making SA a lot faster than GA. Although greedy randomized algorithms such as SA usually performs poorly on large solution spaces as discussed previously in the paper, it has performed reasonably well on Continuous Peaks with N=120, which represents quite a large input space.

Based on this finding, SA seems to be a perfect candidate to be the optimal algorithm for Continuous Peaks although GA is a close second. In order to validate this idea, I ran all 4 algorithms on the Continuous Peaks problem with varying values of N ranging from 10 to 140 as shown in Figure 18.

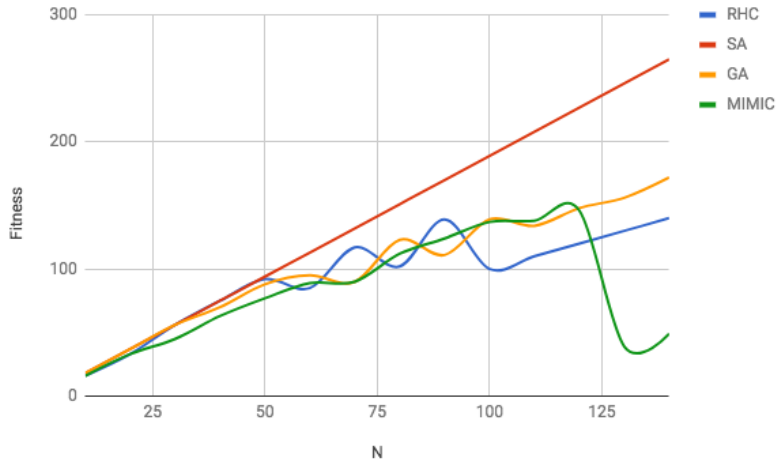

Figure 18: *Performance of all Algorithms on Continuous Peaks with varying N*

Not surprisingly, MIMIC performs quite poorly on Continuous Peaks. This is perhaps because, unlike Four Peaks where there are 4 optima, Continuous Peaks may have more than 4 optima. Therefore, MIMIC is unable to accurately model the solution space. It is clearly evident that SA emerges as a winner in the Continuous Peaks problem. It consistently does better than all other algorithms. What is perhaps very surprising about this is that the fitness levels are quite similar for all algorithms until about N = 40 but with larger N values, SA outperforms the others by a larger margin. Figure 19 shows the runtime for each algorithm with varying values of N. As expected, MIMIC is the slowest out of all algorithms while RHC is the fastest due to its purely random nature. It is, however, surprising to find that the runtime of GA is essentially lower than SA for N <= 100. Therefore, although SA performed well in this problem for the chosen size of N, it is worth considering GA as well as it's crossover-mutation strategy is generally much better at finding the optimal solution than a random approach. However, it is also the case that SA is able to obtain optimal fitness much faster than GA in this experiment.

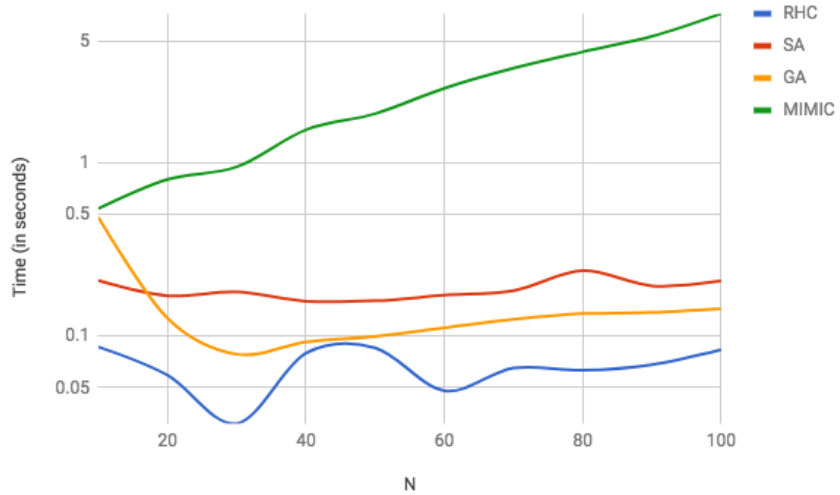

Figure 19: *Runtime of all on Continuous Peaks Algorithms with varying N*

Conclusion

All 4 randomized optimization algorithms analyzed in this paper have unique strengths and comes with its weaknesses. RHC is perhaps the simplest algorithm out of all of them. It has low induction bias which makes it suitable for problems that have no inherent structure. This is particularly evident from RHC's failure to model a structured problem like TSP. Just as RHC, SA is also a greedy randomized algorithm. It was found that RHC and SA share similarities in that both tend to do better in problems, which have a small solution space. It is only in larger problems that RHC and SA suffer as we have learnt from the exceptionally dismal performance of RHC and SA on the Neural Network problem to model the WDBC data. This is because RHC and SA tend to exhaustively search the solution space and is not as clever at modeling the data as GA and MIMIC; therefore, they tend to be extremely slow at converging to optimal solutions in large solutions spaces (No Free Lunch). Similarly, both SA and RHC requires large iterations when the number of dimensions is large such as observed in TSP.

MIMIC is perhaps most suitable for problems that have an expensive cost function. This is because it may as well learn as much information as possible to model the data since it's paying a high cost. One of the biggest advantages of MIMIC is that it is able to generate optimal solutions in very few iterations as seen in case of the Four Peeks problem. It also tends to do well on problems which have a structure. However, MIMIC is very time/memory intensive which makes in unfeasible to apply on large problems, which is ironic as it performs well in such problems.

Apart from MIMIC, GA was another very strong algorithm that performs well in large solution spaces. As seen from the Continuous Peeks problem, it is able to converge to optimal solutions in very few iterations. It is generally good at solving problems that have a structure.

References

1. Russell, S. *Artificial Intelligence: A Modern Approach*
2. ABAGAIL: https://github.com/pushkar/ABAGAIL