CS 7641 Machine Learning

Assignment 2 – Randomized Optimization

Liyue Hu (lhu81)

Abstract

This paper looks to compare and contrast various methods of randomized optimization. The methods explored include Randomized Hill Climb (RHC), Simulated Annealing (SA), Genetic Algorithm (GA) and MIMIC. Three of the four methods (RHC, SA and GA) were first applied on optimizing the weights for a Neural Network (NN) for the problem of predicting credit card default. Their performance was benchmarked against the results from backpropagation. Then, a closer look at three distinct problems were implemented to demonstrate the relative strengths and weaknesses of RHC, SA and MIMIC respectively. The three problems implemented were the Travelling Salesman Problem (TSP), Continuous Peaks and Flip Flop.

1. Neural Network Weights Optimization

For this assignment, I have selected one of the problems I explored for Assignment 1 to compare the ability for the various randomized optimization methods to optimize the weights to NN. The problem I explored is predicating default on credit card payment. The dataset had 23 features, which translated into the 23 input parameters for the network. This is followed by two layers of hidden layer each with 32 units, and an output layer with 1 unit. Jython was used for the implementation of the project. I chose 5000 iterations as the graph had little changes after that.

Performance Analysis

Overall performance across all algorithms can be seen in Table 1 and Figure 1 below as the test F1 Score over 5000 iterations. Both RHC and SA achieved comparable results that were better than backprop towards higher iterations. GA, interestingly had fluctuating accuracy performance as iterations increased. Towards the lower end of the iterations, RHC had significantly better performance than SA, though SA caught up when it got closer to 3000 iterations. Not surprisingly backprop took the fewest iterations to reach a fairly high accuracy level (770 iterations for best) and stayed around that level throughout the later iterations.

Algorithm	Parameters	Best Fitness	Best Iterations	Best Time
Backprop	23, 32, 32, 1	0.45	770	162.35
GA	Population Size: 50 Mate Size: 10 Mutate Size: 2	0.46	1100	1911.93
RHC		0.48	4530	341.47
SA	Cooling Effect: 0.9	0.50	3530	286.70

Table 1: Performance Comparison of Different Algorithms for NN Weights Optimization

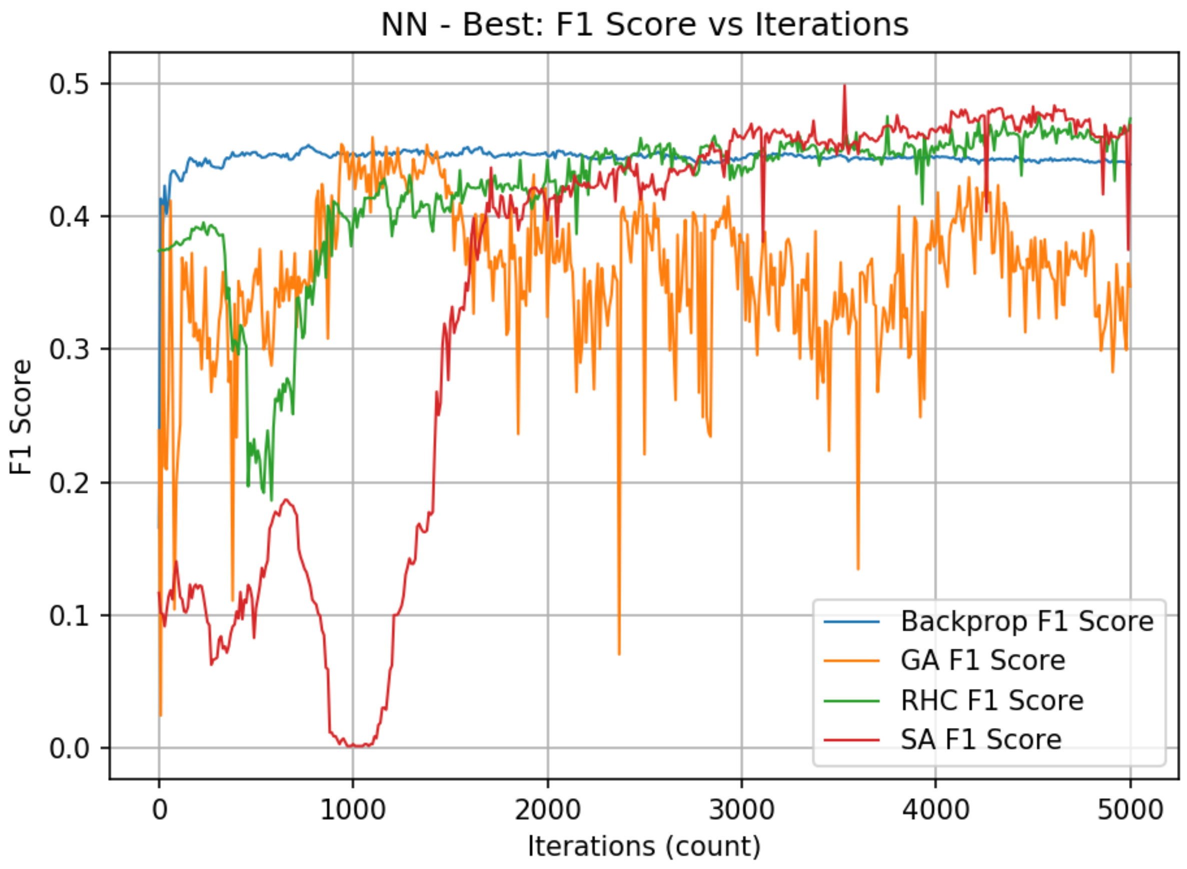

Figure 1: Accuracy Comparison of Different Algorithms for NN Weights Optimization

<u>Randomized Hill Climb:</u> In this case, RHC achieved best fitness of 0.48 at iteration 4530. Out of all the algorithms examined, RHC showed the slowest conversion towards optimal point.

<u>Simulated Annealing:</u> SA achieved the best performance for this problem over with less iterations than RHC. This is a good demonstration of the power of the "cooling effect". Different CE values were tested (0.15, 0.35, 0.55, 070 and 0.95) and the best value was 0.9. SA also required a lot less time than both GA and (to a lesser degree) RHC.

<u>Genetic Algorithm:</u> GA had the second fastest conversion at iteration 1100. Various parameters were testing including population size, mate size and mutation size. The optimal parameters were 50 for population size, 10 for mate size and 20 for mutation size.

2. Randomized Optimization Problems

2.1. Travelling Salesman – Genetic Algorithm:

Problem Description:

To demonstrate the advantage of GA, I chose to implement the TSP. This classic combinatorial optimization problem asks the following question: "Given a list of cities and the distance between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" The problem is known to be NP-hard and cannot be solved in polynomial time. The function that was evaluated was defined as 1/total distance over all routes, I looked to maximize this function in order to minimize the distance. The encoding used is a permutation of [0, .., n] where there are n+1 cities. For the implementation, I used n = 100 over 3000 iterations.

Performance Analysis:

As shown in Figure 2 below, GA achieved the best performance of 0.11, as defined by fitness function compared to all other algorithms evaluated. Both RHC and SA achieved similar level of performance of around 0.05 and MIMIC had the worst performance of all, at 0.04.

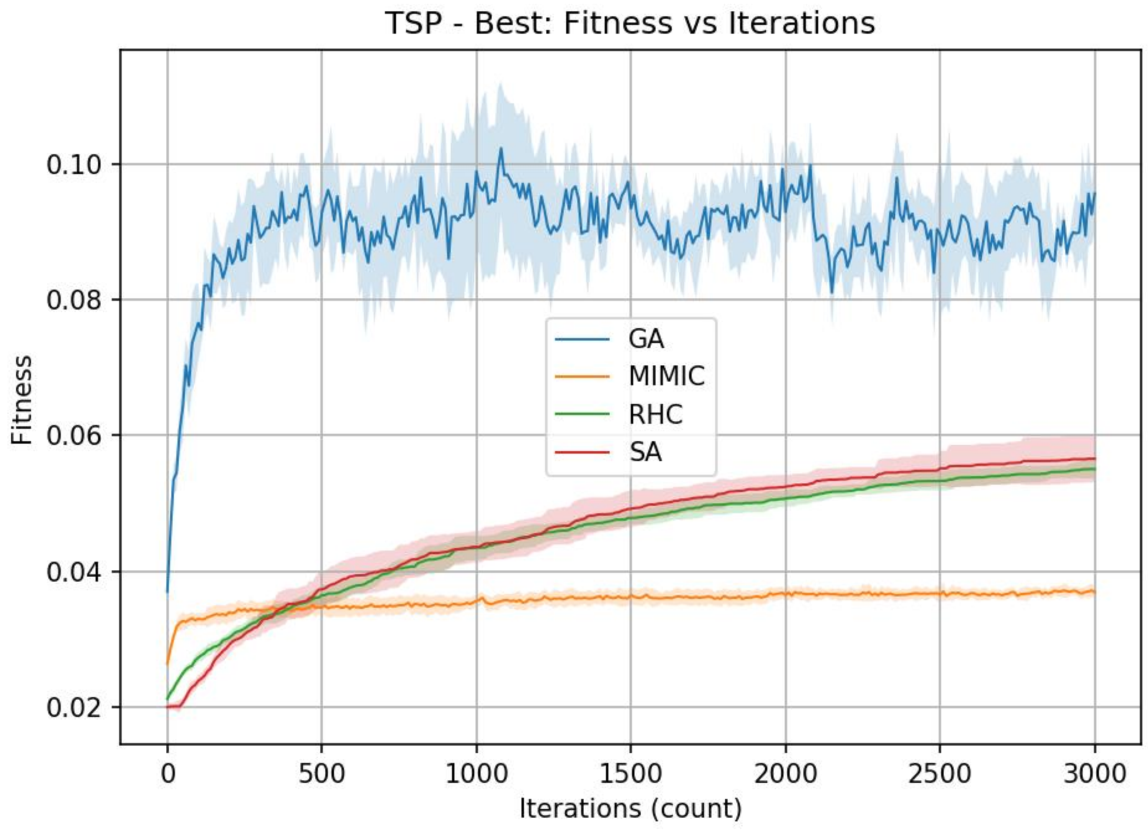

Figure 2: Performance Comparison of Different Algorithms for TSP

I also compared the running time across all the algorithm. As sown below in Figure 3, RHC and SA were the fastest, GA takes about 100x more time than the first two and MIMIC takes a considerable larger amount of time than everything else. However, if we look more closely back at Figure 2, MIMIC requires the least amount of iterations to converge to the optimum.

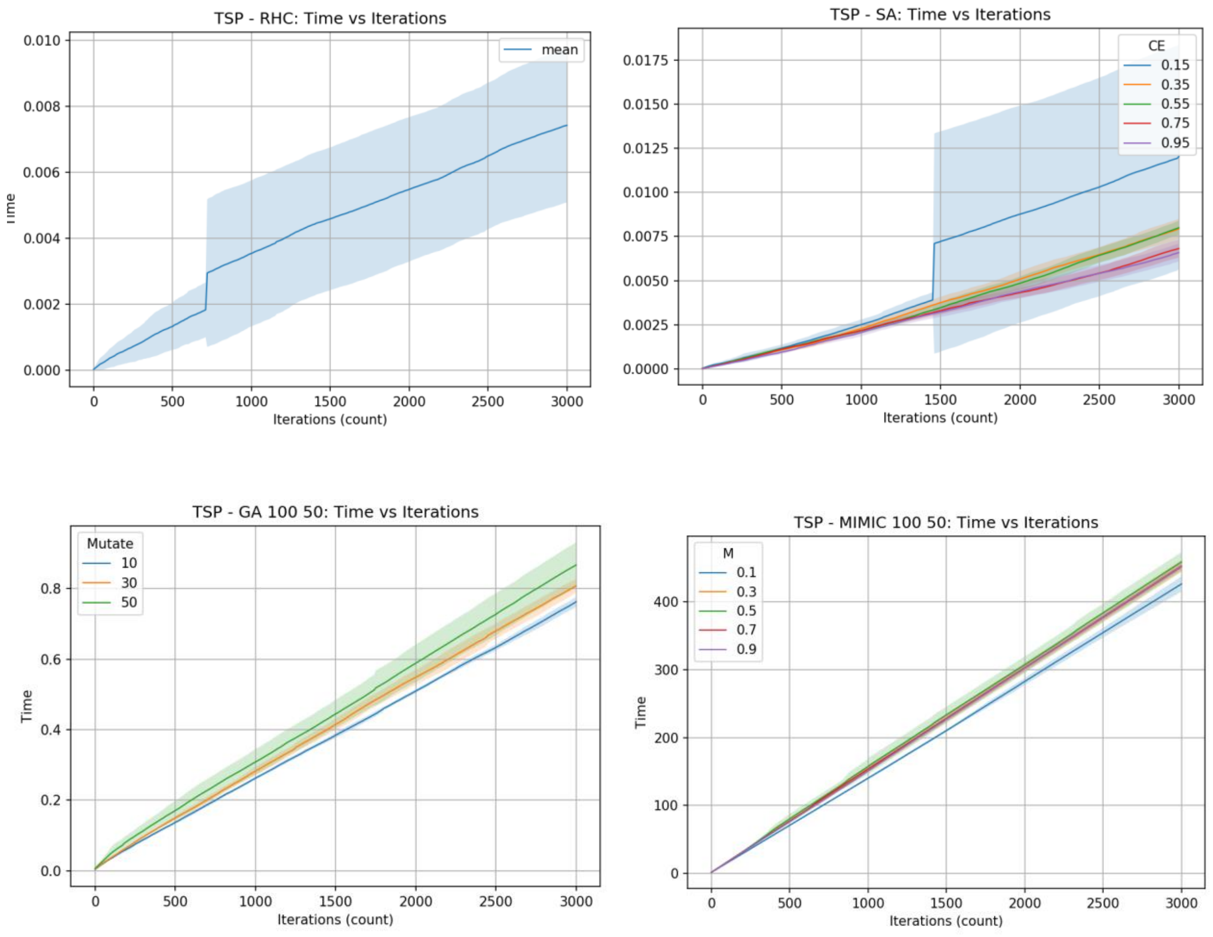

Figure 3: Time Comparison of Different Algorithms for TSP

GA performed best for this problem because GAs is especially suited to tasks in which hypotheses are complex, such as the sets of routes considered for the TSP. GA is able to consider a large set of hypothesis space and not get stuck in the many local optimum in an efficient manner through crossovers and mutations to reach the optimal solution. It shows that, in this case, the cross-over of two short routes could result in shorter route offspring. As for RHC and SA, they both seem to have gotten stuck in a local optimum. This happened because there are so many local optimum and hence a lot of different "hills" to climb/select from, as a result it takes a larger iteration to reach global optimum. MIMIC performed especially poor for this problem, also due to the large amount of local optimum. Even though MIMIC evaluated the most amount of functions out of all the algorithms, in this case, the structure of solutions really doesn't matter, which is what MIMIC is good for.

2.2. Continuous Peaks – Simulated Annealing:

Problem Description:

To demonstrate the advantage of SA, I chose to implement the continuous peaks problem. This is an expansion of the Four Peaks and Six Peaks problem discussed in the class reading. The problem seeks to find the highest peak out of all the potential peaks (local optimum) out there. The main challenge with this problem comes with the fact that there are many local optimum and we want to see which algorithm is best at finding the global optimum. For the experiment, I used N (number of integers) of 100, T of 29 (basin of attraction). The experiment was conducted over 5000 iterations over 5 tries.

Performance Analysis:

As shown in Figure 4 below, the best performing algorithm was SA. It reached the highest fitness level at the end of the 5000 iterations. MIMIC performed the worst after 5000 iterations, and RHC and GA had comparable performance.

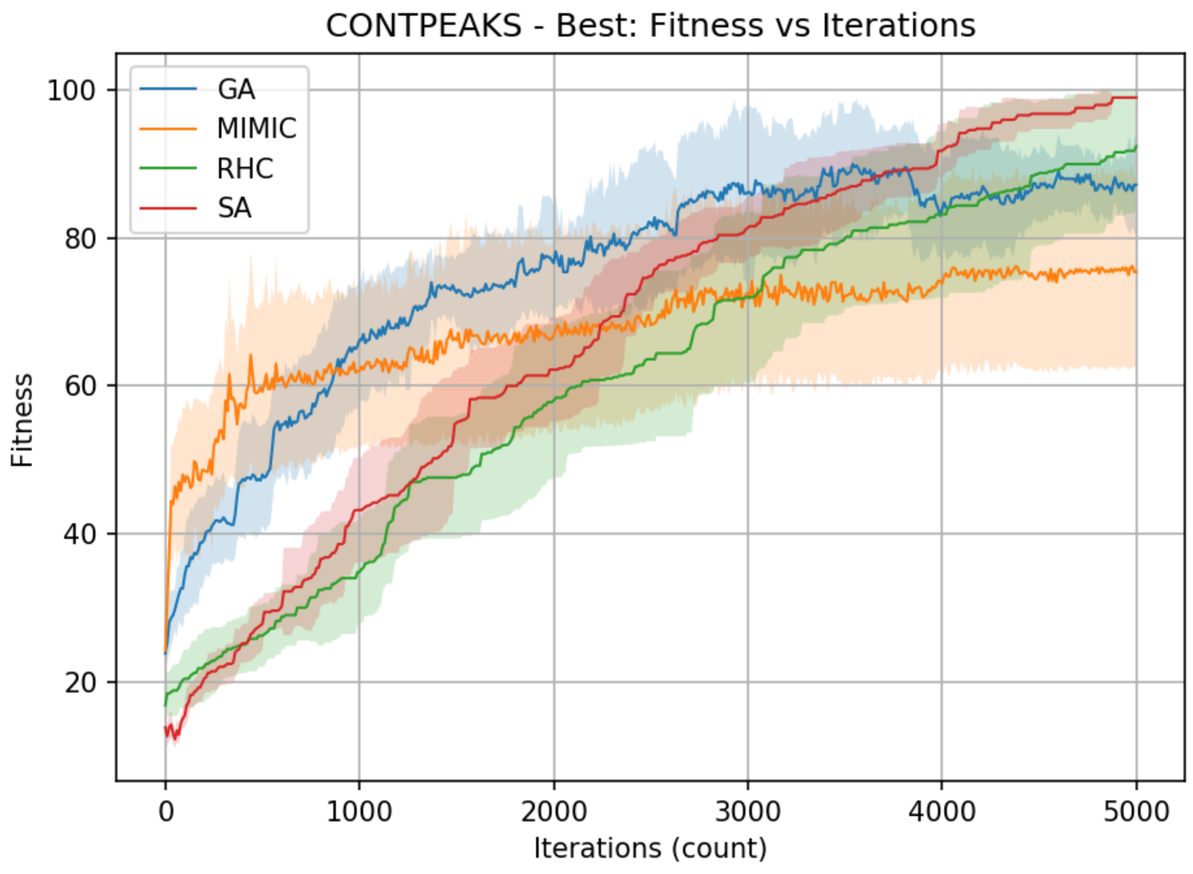

Figure 4: Fitness Performance of Different Algorithm for Continuous Peaks

This is the case because SA is specifically designed for problems of many local optimum in a large search space. It is also suited for discrete space search that characterizes this problem. The theory behind SA is that by incorporating both "heating" and "cooling" (especially cooling), the algorithm allows for a more extensive search of the hypothesis space by accepting worse solutions. This improves the likelihood of approximating the global optimum. Specifically, if we look at its performance as compared to the RHC, which is a similar algorithm without the cooling step, we can see that it surpasses the performance of RHC at around 500 iterations and then stays at a fairly constant rate above the performance of RHC.

GA performed fairly well on this problem as well. Both GA and MIMIC had better performance towards the lower end of the iterations and improved more slowly with additional iterations.

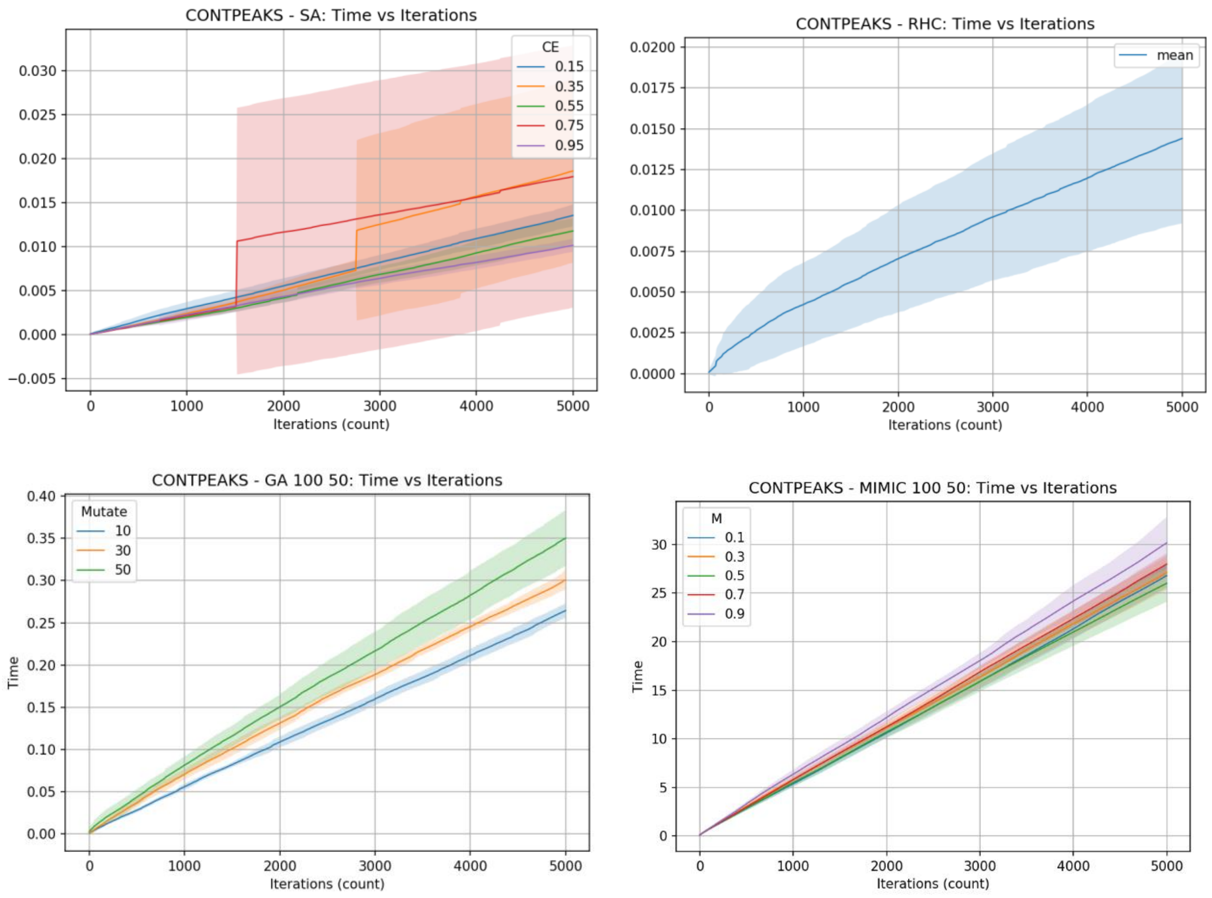

Figure 5: Time Comparison for Different Algorithms for Continuous Peaks

The benefit of SA becomes more apparent over GA and MIMIC as we look at the time of the various algorithms, as shown in Figure 5 above. SA took only about 1/10th the time of GA. I also looked at the effect of different cooling effect (CE) values and their effect on fitness, the best CE value is 0.75 though there is not too much of a different between the various values in terms of both time and fitness over iterations. See Figure 6 below for the various CE values tested over iteration. The continuous peaks problem shows that SA is a good algorithm for discrete valued hypothesis space where there are many local optimums, and one of its main advantage over GA and MIMIC is its run time and its advantage over RHC is its ability to improve fitness through the use of “cooling”.

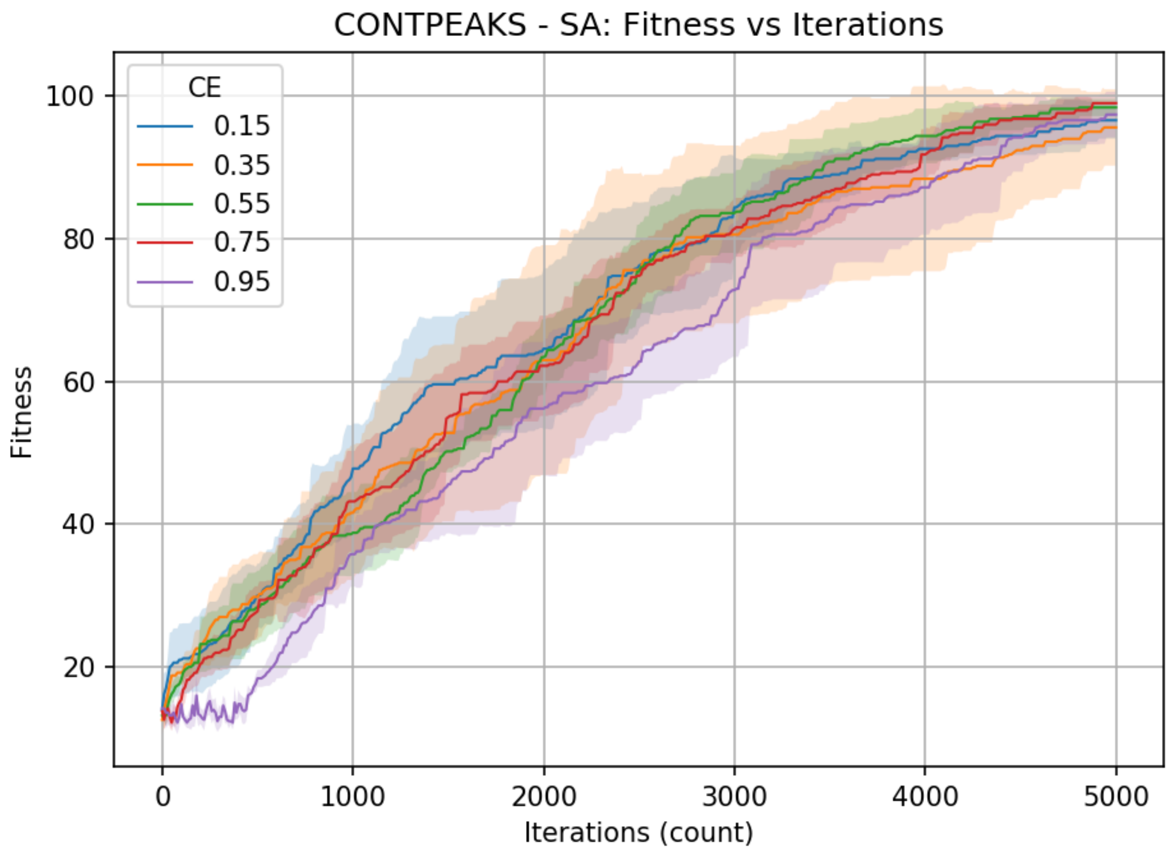

Figure 6: Performance Comparison with Varying CE for SA for Continuous Peaks

2.3. Flip Flop – MIMIC

Problem Description:

To demonstrate the strengths of MIMIC, I implemented the classic Flip flop problem. Flip flop is a simple problem that seeks to maximize the amount of times a bit string alternates between 0 and 1. The optimal solution would alternative between 0 and 1 continuously. The fitness function that is maximized is the number of uninterrupted 0 and 1 flip-flops. For the experiment, I set N (length of bit strings) to 1000, and ran for 3000 iterations.

Performance Analysis:

As shown in Figure 7 below, MIMIC had the best performance for the problem. It also reached a high level of fitness a lot faster than the other algorithms. SA and RHA achieved comparable performances with gradual increase in fitness over iterations. GA performed poorly on this problem, only reaching a fitness level of around 600.

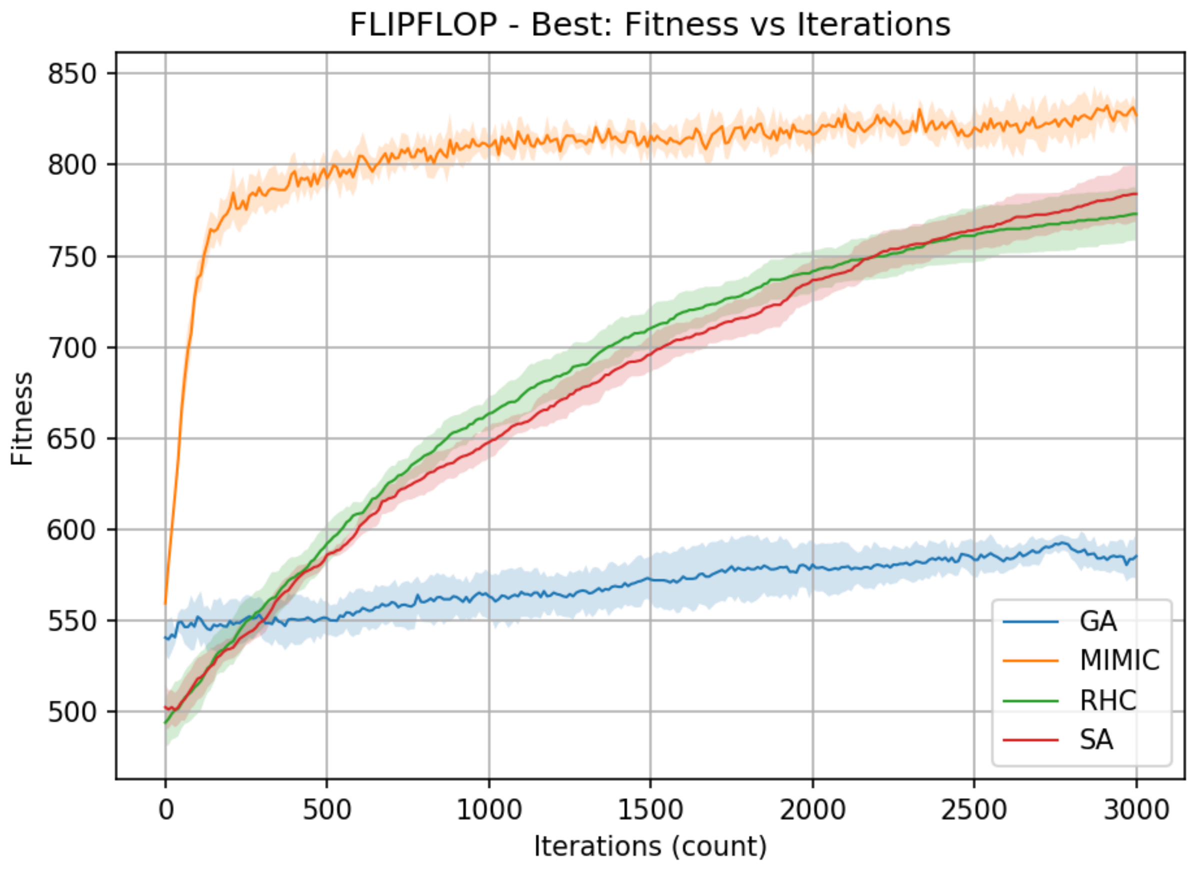

Figure 7: Performance Comparison of Different Algorithms for Flip Flop

However, we should note that none of the algorithms (not even MIMIC) achieved the optimal solution of 1000 for this problem. The best result achieved by MIMIC is 843, however, it looks like the performance has plateaued towards the latter iterations. For SA and RHC however, the performance seems to be continuously improving even towards the latter iterations, with the hoping of reaching the optimal solution of 1000 with more iterations. The best result achieved by SA was 814.

MIMIC performed well on this experiment since it is specifically designed for problems that explore relationships between input parameters, such as certain parameters are closely related and therefore should not be explored independently (connected bits). The initial advantage of understanding the structure of the distribution however waned off as the algorithm starts to fall into the many local optimums. This is characterized by the fact that in later iterations SA started to improve consistently iteration after iteration whereas MIMIC was stuck around the local optimums.

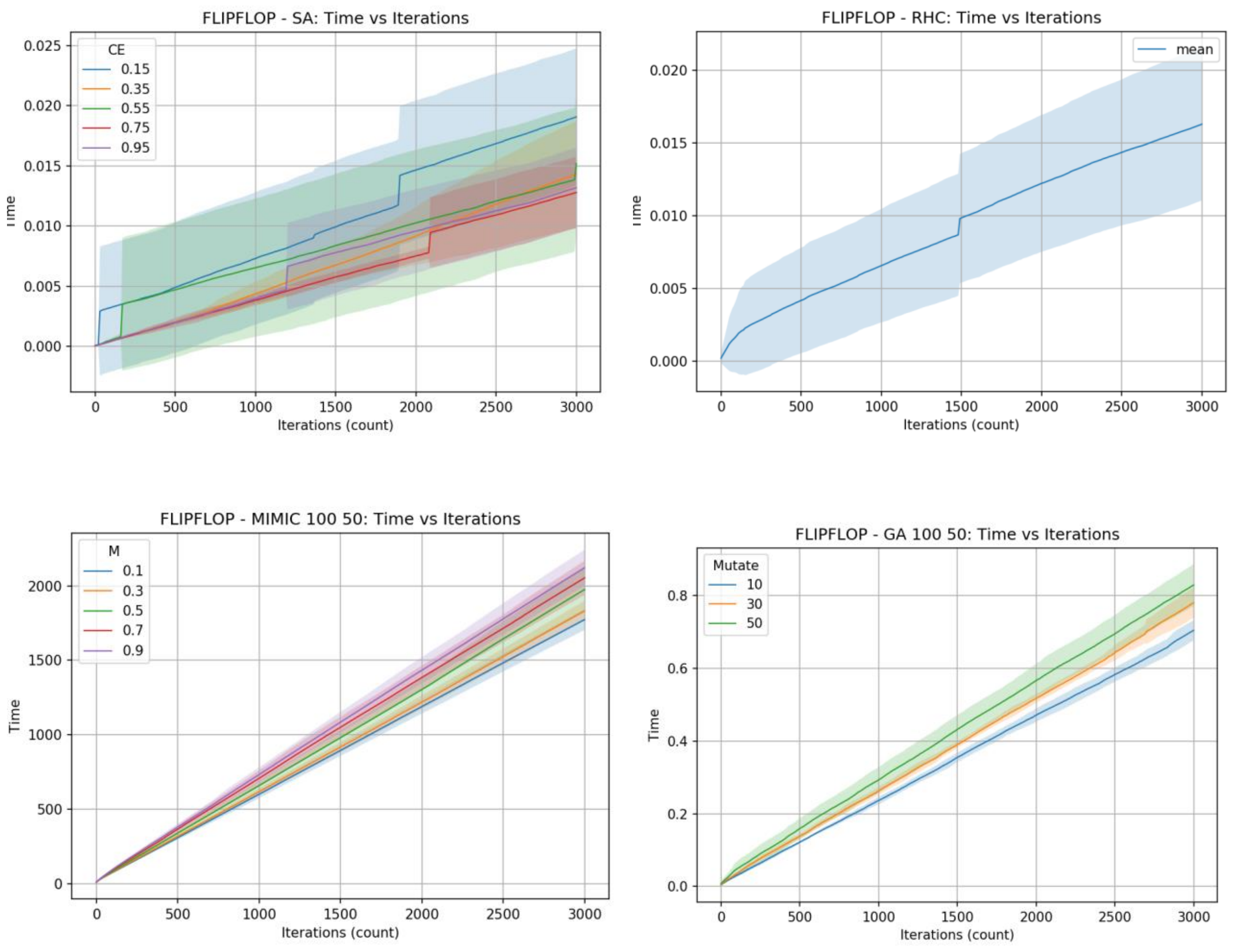

Figure 8: Time Comparison for Different Algorithms for Flip Flop

If we look at the time for the different algorithms as shown in Figure X above, we can see clearly that the initial information advantage of MIMIC came at a steep cost of time. Figure X-1 shows that MIMIC takes orders of magnitude fewer iterations than SA/RHC but still takes more time. This is shown clearly in Figure 8 above.

Although the flipflop problem showcases the advantages of MIMIC in its ability to compute dependency trees to gain information advantage with very few iterations. It is also clear that the choice between MIMIC and SA is not entirely clear as the time complexity required for the two can sometimes more than offset MIMIC's advantage.

Conclusion

As shown in this paper, the performance of the various randomized optimization algorithms can vary quite significantly depending on the problem at hand. In practice, it is extremely important to understand characteristics of the underlying problem (e.g. continuous v.s. discrete) and the constraints (e.g time complexity) to apply the most appropriate algorithm. A few remarks on the various learners include:

- Randomized Hill Climb (RHC): One of the more time efficient algorithms, and can usually achieve comparable results to SA. However, it is susceptible to the issue of local optimum.
- Simulated Annealing (SA): Requires fairly low time to run (similar to RHC), with better results for problems with a lot of local optimums, because of its inclusion of “cooling effect”, accepting worse solutions sometimes, it improves the algorithm’s likelihood to find the global optimum.
- Genetic Algorithm (GA): Requires a lot more computational time, however, can achieve significantly superior performance for certain problems that involve complex and large parameter space (such as TSP).
- MIMIC: The most time intensive algorithm to implement, but it requires significantly less iterations to converge. However, the benefit from iterations can’t outweigh the time complexity.

Reference:

https://en.wikipedia.org/wiki/Travelling_salesman_problem

https://iccl.inf.tu-dresden.de/w/images/b/b7/GA_for_TSP.pdf

https://github.com/cmaron/CS-7641-assignments/tree/master/assignment2

Code: https://github.com/huhu42/assignment2