Jiaxin Liu
GTID: jliu727

Assignment 2: Randomized Optimization

Algorithms Summary:

Randomized Hill Climbing:
Randomized hill climbing starts out from a random point and move towards more optimal neighbor until it reaches a local optimal point. This process is repeated several times starting from other random points to avoid falling into local optimal point that is not global optimum.

Simulated Annealing:
For a finite number of iterations, simulated annealing starts out from a random point and jump to a new sample with probability depending on the optimal level of neighbor point and the temperature T at that time. In the beginning, when T is very large, it acts like random walk. Towards the end, when T is small, it acts like hill climbing. The random walk part allows it to explore more places to avoid falling into local optimum that is not global.

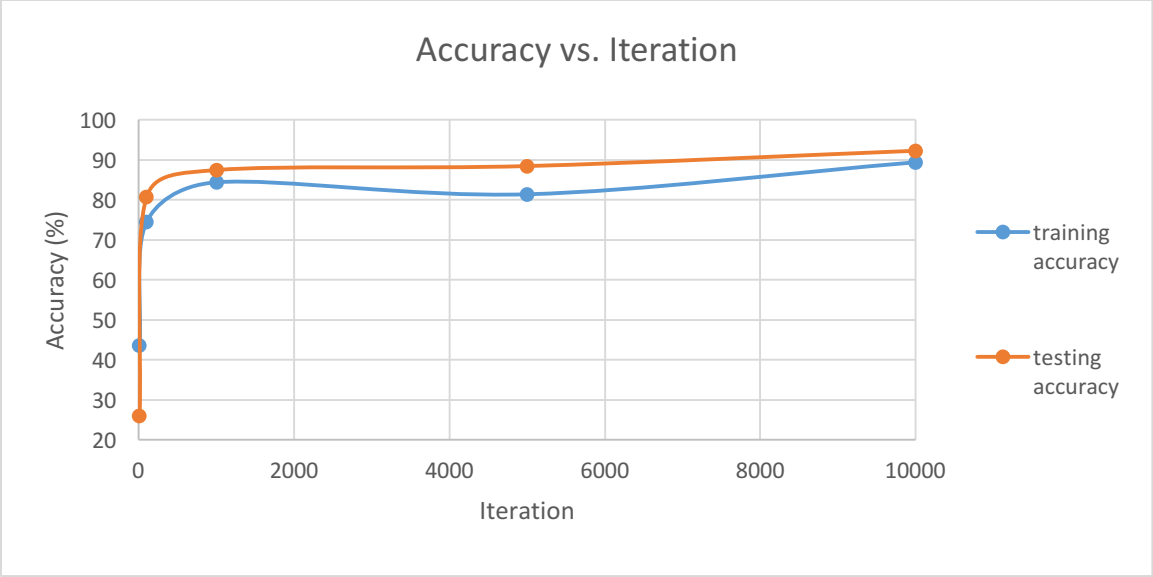
Genetic Algorithm:
Genetic algorithm would compute fitness of all qualifying x at time t and select the most fit individuals above certain threshold (e.g. top half), then pair them up and replace the least fit individual via crossover/mutation. It would continue this process until it converges.

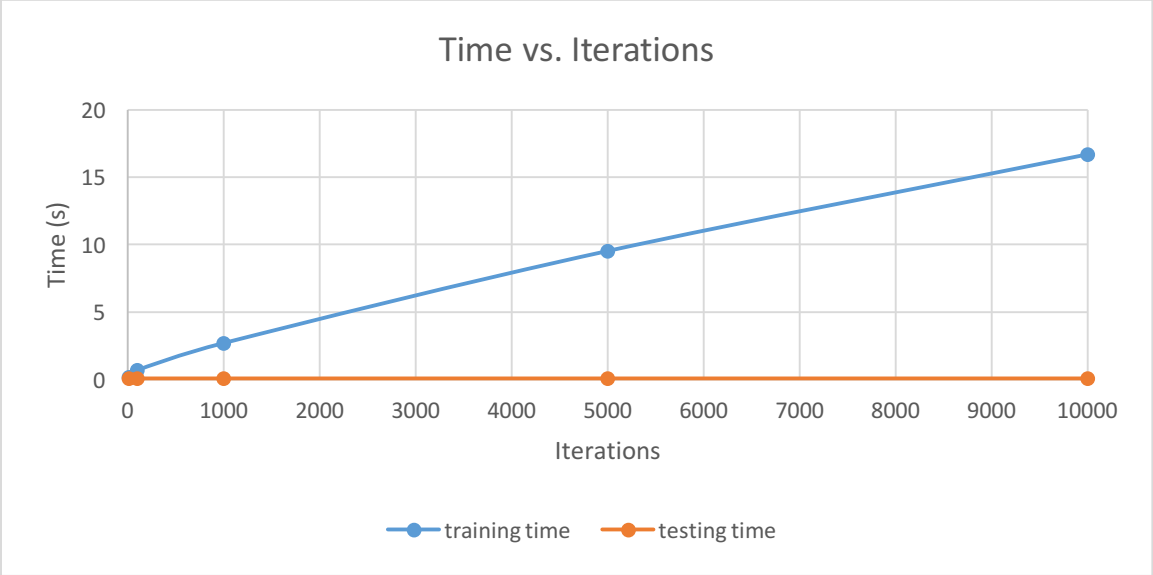
MIMIC:
All above algorithms only cares about points but not structure or history. MIMIC directly model the distribution and refine the model while conveying the structure. It samples uniformly from all points above certain threshold and estimate new probability distribution and repeat the process until it converges to the optimum.

Part I:

I am using the university ranking data from assignment 1. The dataset has 7 attributes related to the school properties and output an overall score (0-100) for the university. In this assignment, I am treating is as a binary classification problem where score < 46 counts as 0, otherwise, it counts as 1. The data has 2200 instances and I am splitting it 50%/50% for the training and testing dataset. The 3 randomized optimization used in this part are Randomized Hill Climbing (RHC), Simulated Annealing (SA) and Genetic Algorithm (GA). I am taking implementation for all 3 algorithms directly from ABAGAIL. I am using a neural network of with an input layer of 7 units, a hidden layer of 3 units and an output layer of 1 unit. Then, I look at all 3 algorithms with iterations = 10, 100, 1000, 5000, 10000

Randomized Hill Climbing:

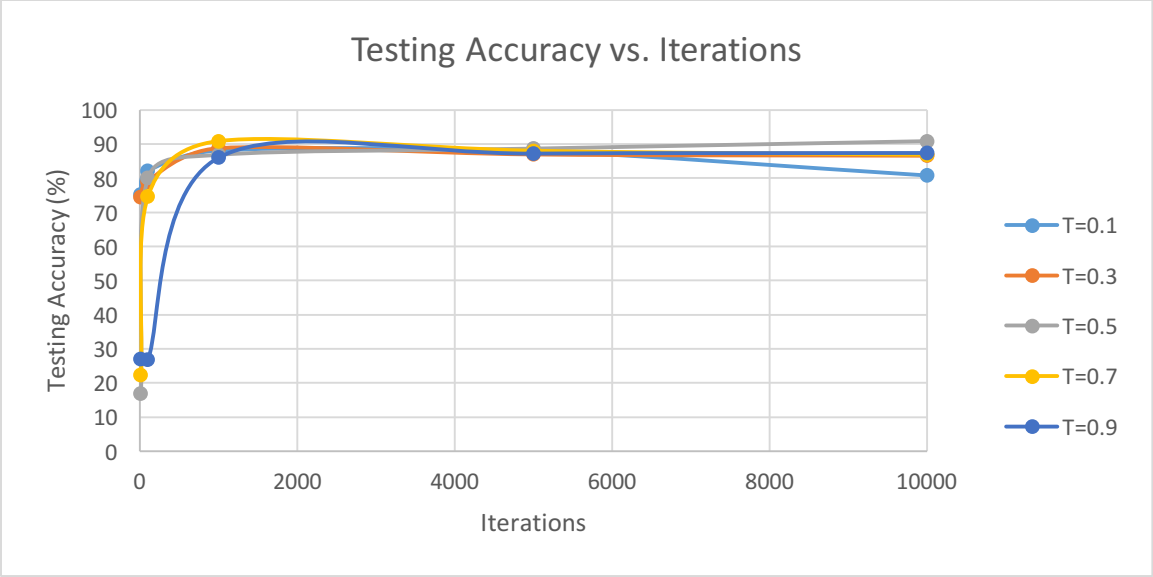




Looking from the accuracy chart, both training and testing accuracy increases with the increase in number of iterations. We can see there is a huge bump in accuracy ~ 100 iteration and the speed in accuracy increase become slow. However, more iterations allow more starts from different random points increases the probability of finding good start points, making the confidence of getting the classification right higher.

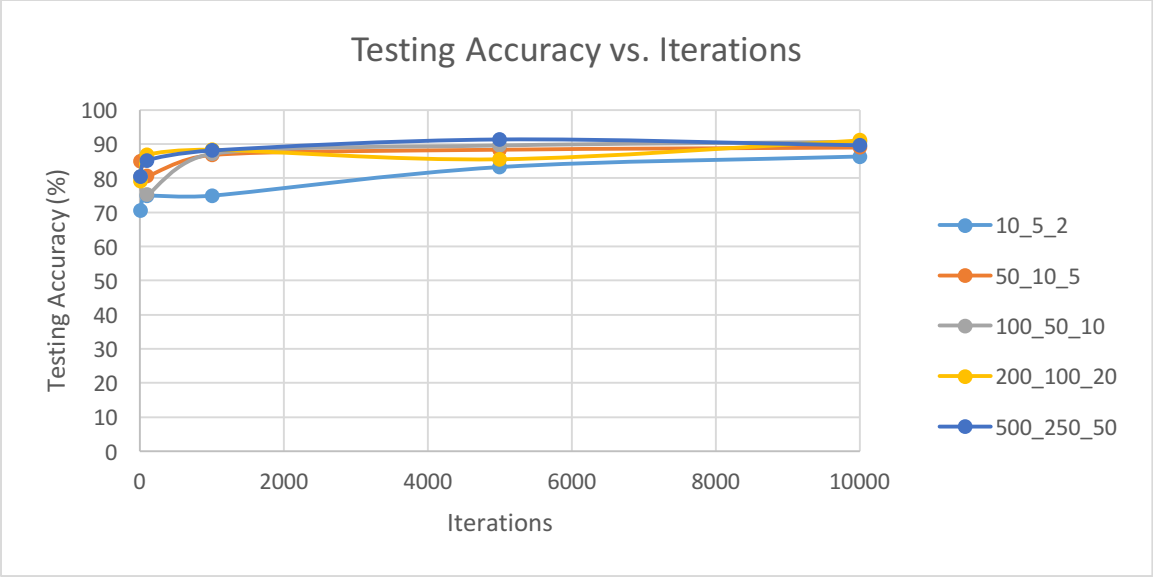
As for training time, we ca see that it is linear to the iterations we take which makes sense since the cost to do one hill climbing is mostly constant, the only factor would be the number of iterations we make.

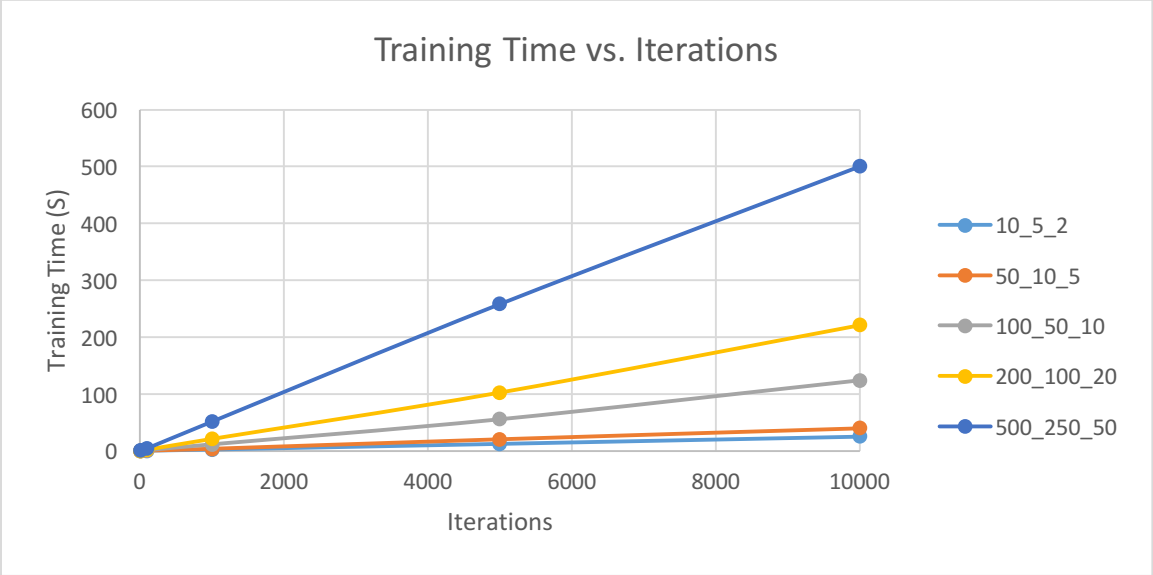
Simulated Annealing:



Looking from the accuracy chart, we can see that testing accuracy increases with the increase in iteration number. Similarly, there is a large bump ~ iteration 100 We can see that performance of most T are almost identical except when T is very low. T = 0.9 and T = 0.5 performs better. It makes sense that T = 0.9 performs better as there is more time for SA to explore the entire space so that is does not get stuck in a local optimum. I think the reason all the other ones are also performing good is because the dataset I am using is relatively simple without a lot of local optimum, therefore, even lower T can avoid falling into local optimum towards the end. We can see that training time increases proportionally with the increase in iteration number.

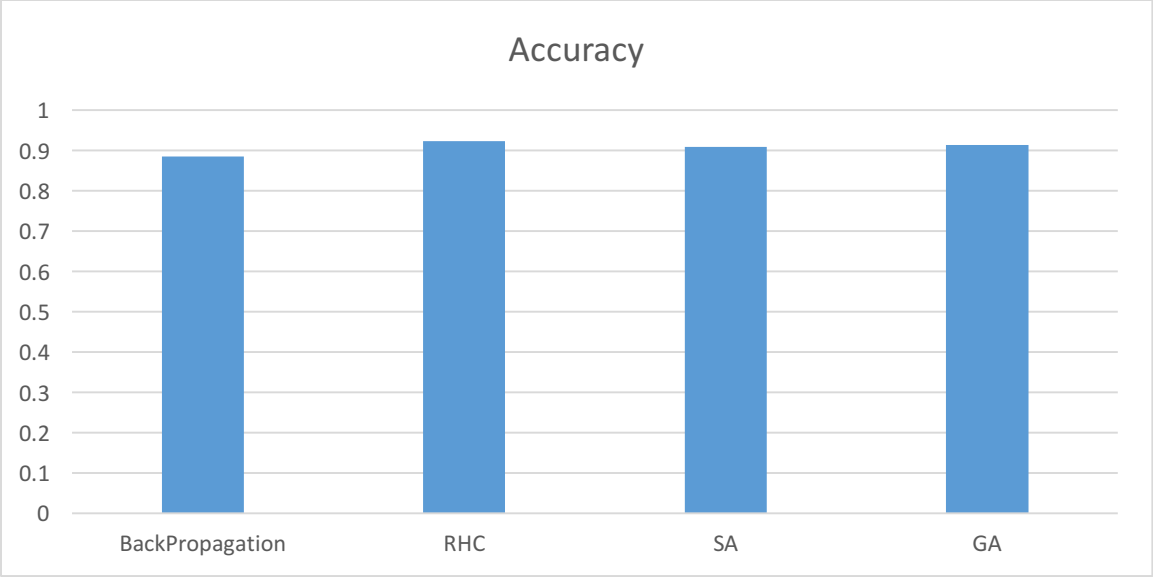
Genetic Algorithm:





Looking from the accuracy chart, we can see that GA performs better with the increase in iteration number. Similarly, there is a bump ~ iteration 100. Overall 500_250_50 performs the best and 10_5_2 performs the worst. We can see the time is still proportional to iteration for different parameter settings but the steepness is different for different parameters. This makes sense, as we increase the population, it takes longer for the algorithm to clear the whole space, hence time for a single iteration also increases.

Comparison with Back Propagation:

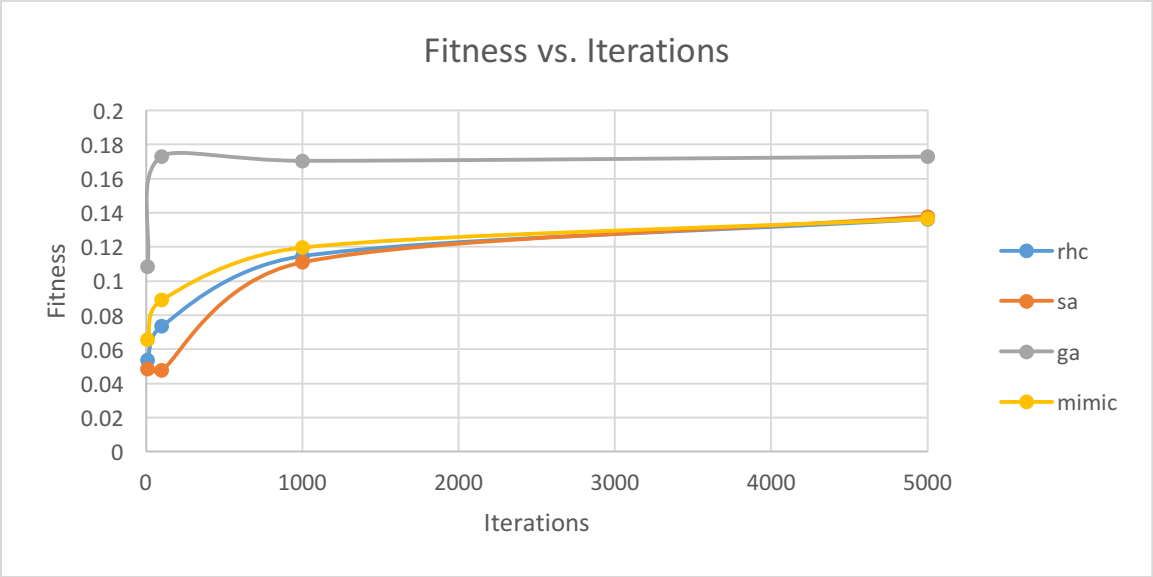


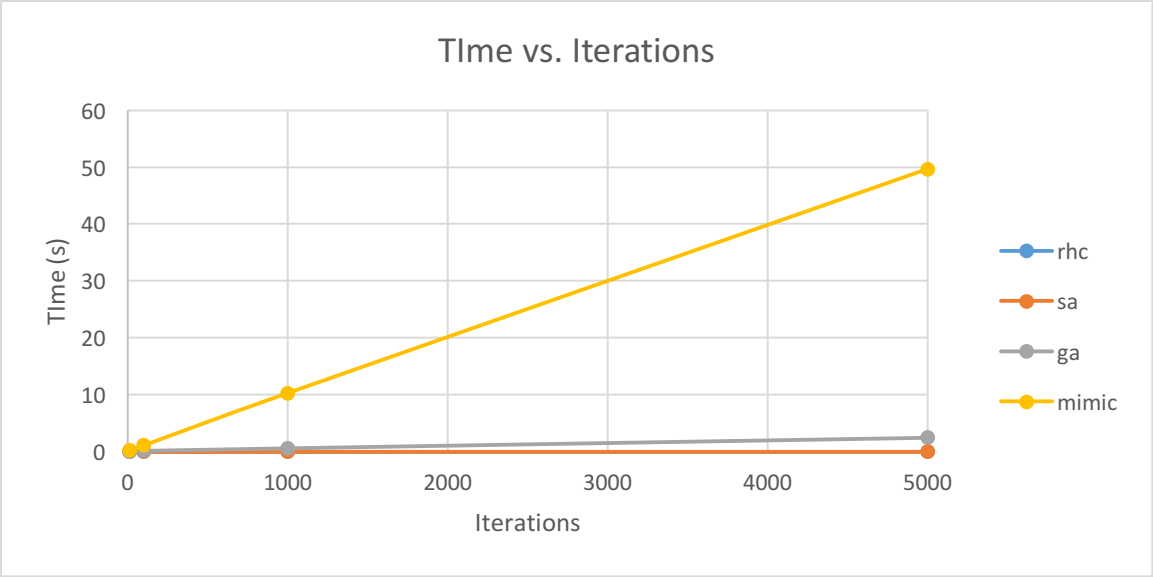
We can see that the performance of these randomized optimization algorithm is comparable to back propagation, even slightly higher. It may be because randomized optimization does not have to deal with the over fitting problem, resulting in better and better optimization with increase in iterations. Although neural network weights are continuous, we can still use randomized optimization as we search the hypothesis space by applying a small discrete change every time, which may be slower than back propagation that makes small changes to all weights every time, but still performs well.

Part II:

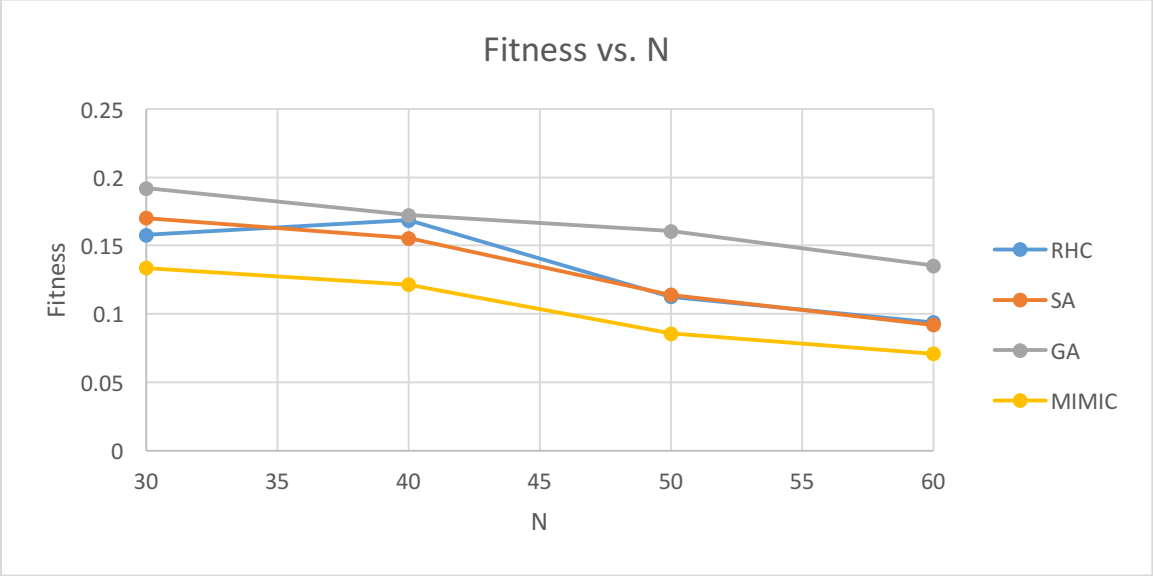
Traveling Salesman (GA):

Traveling Salesman problem requires the salesman to find the best route (shortest distance) to visit all cities. Here we are looking at 1/d so the higher, the better. We set the number of cities(N) to 40.

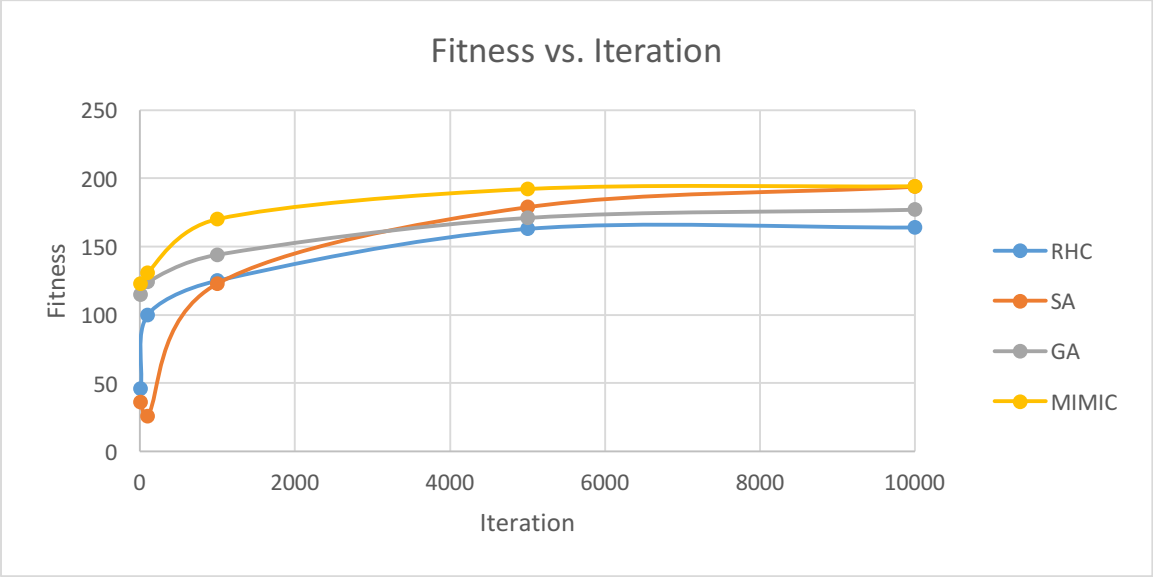


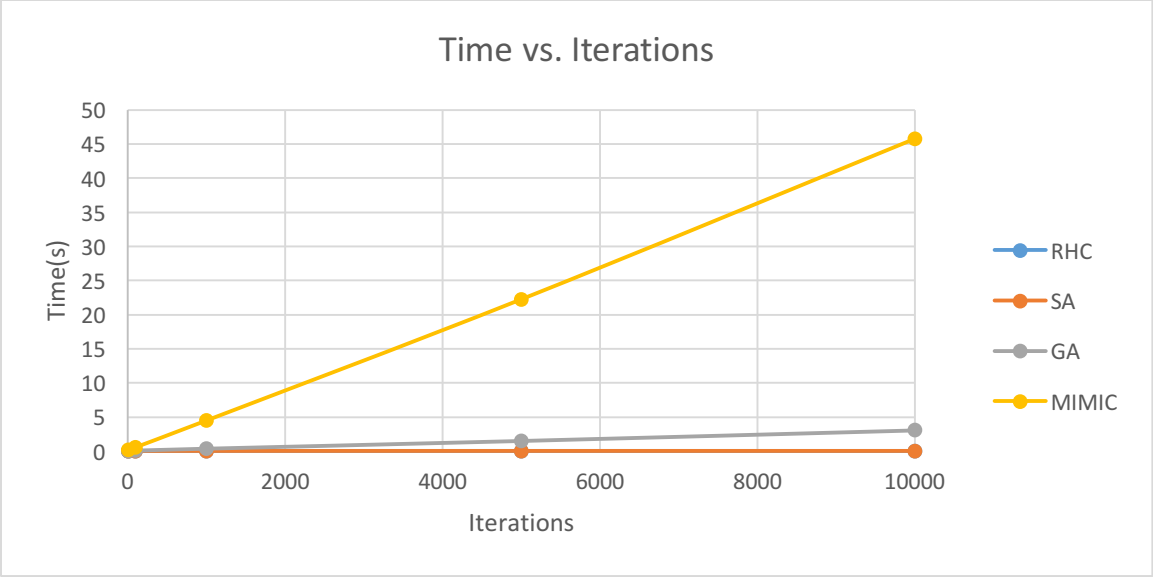


We can see for the fitness performance; GA is doing much better compared to the rest of the algorithms. Regarding to time spent, it is also tolerable. The performance of the other three algorithms are pretty much the same. It makes sense that GA performs best in this case as GA get to better path by combining and mutating two already good path, that almost certainly guarantees an as-good or better path intuitively. As for other algorithms, randomly searching or changing the path at each step does not find good path as efficiently as GA does.
We can see that it is consistent with the change in number of city N (This is done by directly changing N number in TravelingSalesmanTest.java and changing iteration for MIMIC to 500 because it is running too slow with large N) We can clearly see that GA outperforms all the time



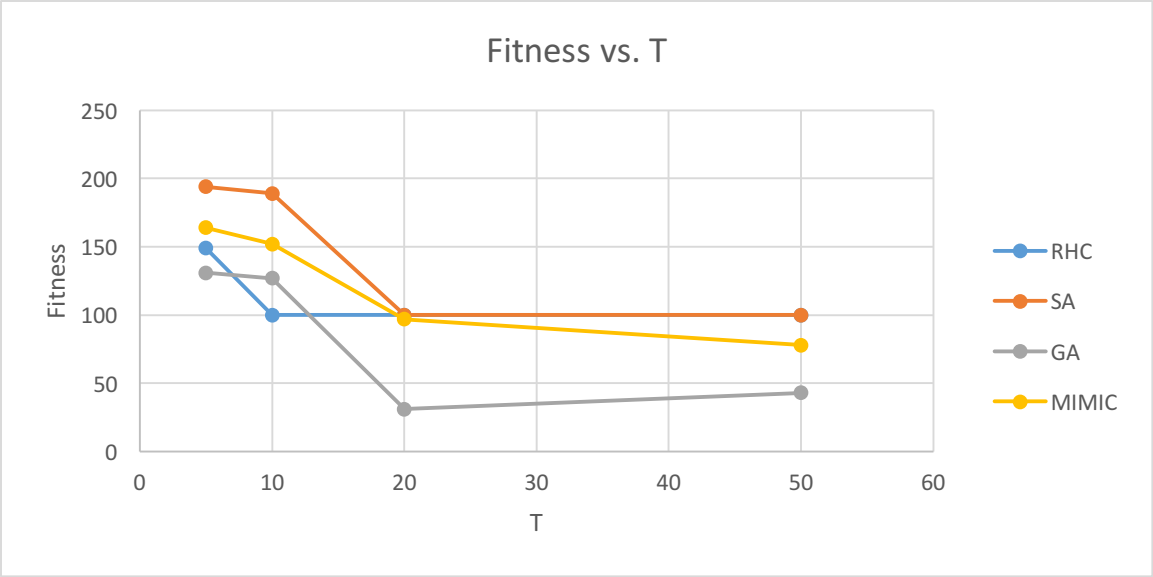
Continuous Peaks (SA):
Continuous Peaks Problem requires us to find the global maximum peak, which is essentially the largest y value on a x-y plane. The valleys and peaks on the plane are defined by the N and T (basin of attraction) value. Here we are using N = 100 and T = 5.



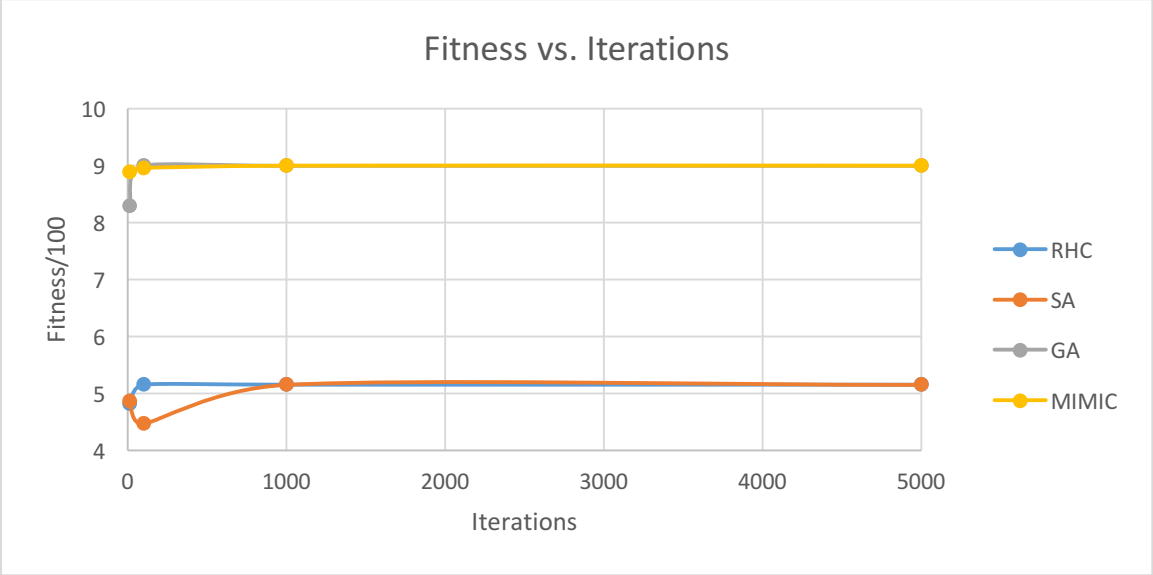


We can see that for fitness performance, both MIMIC and SA are doing pretty well at large iterations. However, we can see that MIMIC seems to be entering a plateau with large iterations while SA still has the tendency to find more optimal number. Moreover, if we look at the time spent to run each algorithm, we can see that MIMIC is spending significantly large amount of time in large iterations compared to SA. Therefore, although MIMIC was performing the best with small iteration, considering the two factors above, I think SA would be the best algorithm in this case. It makes sense theoretically as well since SA is more similar to RHC which is more designed to find the optimum point on hill.

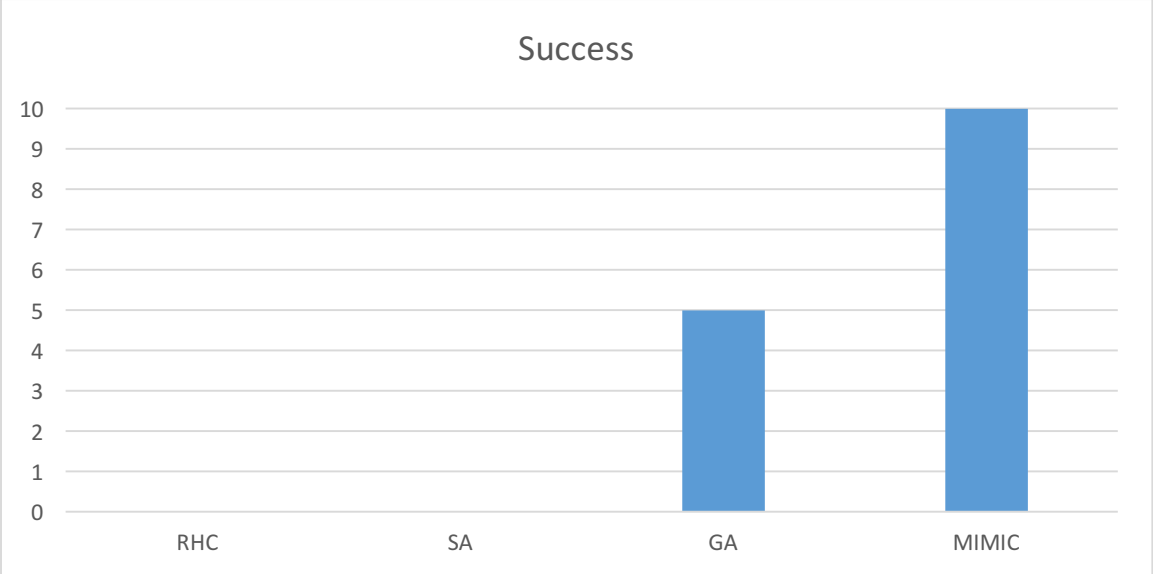
We can see that it is consistent when we change the input T (This is done by directly changing T number in ContinuousPeaksTest.java). Apparently, SA performs better than other 3 algorithms.



MaxKColoring (MIMIC):
MaxKColoring Problem asks us to find a color pattern where for each node in the graph, there is no adjacent node with the same color. We set N = 100 (number of vertices), L = 5 (adjacent nodes per vertex), K= 10 (number of possible colors).

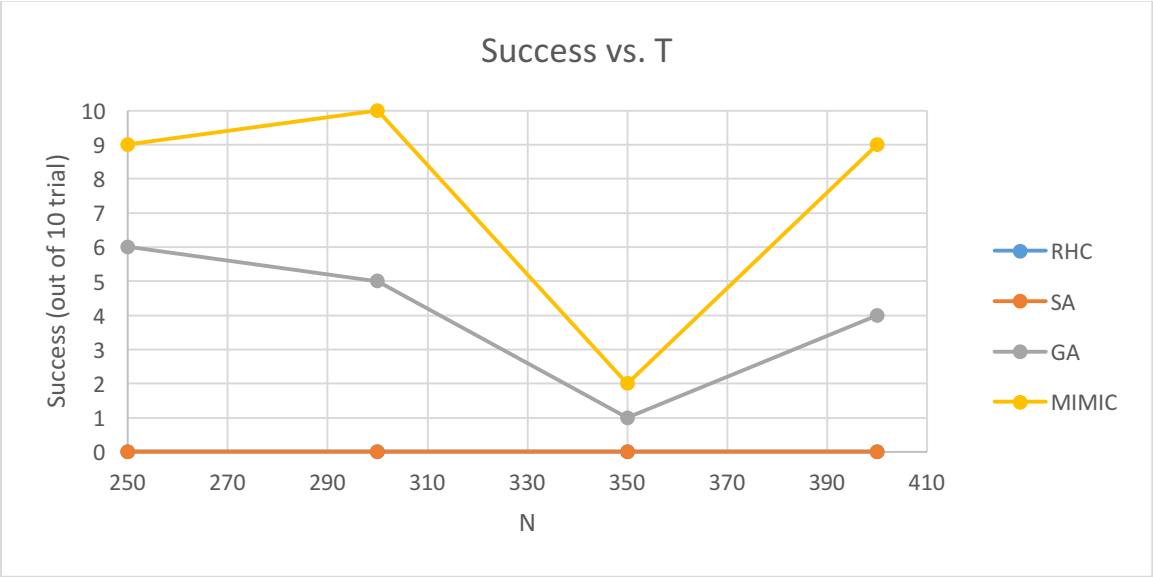


We can see from the above fitness graph that MIMIC and GA performs much better than SA and RHC. On the other hand, MIMIC is more efficient than GA time-wise. The fitness function value here does not make a lot of sense regarding to what we are looking for so we directly look at the percentage of successfully finding the Max-K Color Combination. To better separate accuracy between MIMIC and GA, we set higher N, L, K. N = 300, L = 10, K = 15 and iteration of 100. (This is directly using the TravelingSalesmanTest.java with change in N, L, K)



We can see out of 10 times, MIMIC succeeded every time! GA succeeded half of the time and RHC and SA was not able to find the pattern. This makes sense as this problem is asking us to determine the color of nodes where MIMIC is a perfect fit. RHC and SA does not work well in this case as it is hard to make small improvement in this case as a small change on one node could change the whole pattern, therefore, it is hard to determine the 'next' best step as every node is deeply related to many other nodes. In real practice, MIMIC would even outperform in time perspective as it only requires a small amount of iteration for MIMIC to find the right pattern.

We can also see it is consistent when changing the number of nodes (This is done by directly changing N number in MaxKColoringTest.java) MIMIC outperforms in different N values.



Summary:

We can see from the above analysis that each algorithm has its own advantages and disadvantages and it is important to choose the right algorithm when solving problems. RHC works well for problems such climbing hills with few optimum, where we alter value slightly every step. It is generally very fast fitness function is easy to evaluate. SA is good with solving problems where small alterations are needed while also avoiding local minimums. It takes a little longer time but still pretty fast. GA works well with problem where mutation and crossover help improve the whole population, such as the graph problem we mentioned above. It take relatively long time compared to SA and RHC especially when the population is large but it can solve more complex problems. MIMIC is good when we care about the state and interactions between each node but it usually takes the longest to run one iteration. However, it can be very efficient when applying to the right problem because it would only require very few iterations.

Reference: ABAGAIL Package: https://github.com/pushkar/ABAGAIL
https://www.kaggle.com/mylesoneill/world-university-rankings