CS 7641 – Randomized Optimization (Assignment 2)

Introduction

This paper is split into two parts that will implement four local random search algorithms – randomized hill climbing, simulated annealing, genetic algorithms, and MIMIC.

Part one will use the first three algorithms to find good weights in the Neural Network for the Phishing dataset from assignment one. Part two will use all four search techniques on three different toy optimization problems. The ABAGAIL library, a collection of Java packages that implement machine learning and artificial intelligence algorithms, will be used.

Description of Algorithms Used:

Randomized Hill Climbing (RHC):

Randomized hill climbing seeks to make local improvements that add up to global improvements. The algorithm will guess a point with a particular value and move around the neighborhood. If there is a neighbor with a value greater than the current point, it will move to the new point and repeat. Otherwise, it will stop since it has reached a local optimum and start again from a random point. The advantages of RHC is that there are multiple tries for a good starting point and it's not expensive – there's a constant factor for restarts. The disadvantage is that it can get stuck at a local optimum. RHC has no parameters in ABAGAIL.

Simulated Annealing (SA):

Simulated Annealing comes from metallurgy, where repeated heating and cooling strengthens the blade, giving molecules an opportunity to realign themselves to find the optimal solution, or blade strength. SA is a tradeoff between improving (exploiting) and searching (exploring). In each iteration, SA works by sampling a point in the space and jumps to the new sample with a probability dependent on the old point, new point, and temperature. If the fitness at the new point is greater, it always hill-climbs. If not, it depends on the fitness difference and temperature. We start off at a high temperature and decrease it by a cooling rate. As temperature decreases, the probability of going to a lower fitness decreases. When fitnesses are really close, we are likely to make a move, otherwise not. Simulated Annealing may be preferred to alternatives such as gradient descent when finding a global optimum is more important than finding a precise local optimum in a fixed amount of time. There are two parameters in ABAGAIL, the starting temperature and the cooling rate.

Genetic Algorithm (GA):

Genetic Algorithm is based on natural selection where two parents produce a "better" offspring. Genetic Algorithm has three main steps - selection, crossover, and mutation. During selection, the fitness of all individuals in the population are computed and the "most fit" individuals are selected. The individuals produce offspring via crossover and their children inherit mutations, or random changes, from the parents. These offspring will replace the least fit individuals in the population. With each iteration, this process will improve the "fitness" of the population and maybe converge to an optimal solution. There are three parameters in ABAGAIL, the population size, the number to mate each time, and the number to mutate each step.

Mutual Information Maximizing Input Clustering (MIMIC):

MIMIC finds optima by estimating probability densities. MIMIC generates samples from a probability distribution whose fitness value is greater than a value theta. From that sample it takes a set of points higher than theta and uses that to estimate a new distribution. With each iteration, over time it will converge to a theta max, which is an optimal solution. The main benefit of MIMIC is that it's able to directly model distribution, which leads to conveying structure. There two main parameters in ABAGAIL - the number of samples to take each iteration and how many samples to keep.

Part 1: Randomized Optimization for Neural Network

Description of dataset

The Phishing Websites dataset classifies 11,055 instances with 30 attributes as a binary response of -1 (not phishing) or 1 (phishing). All data is nominal, either -1, 0, or 1.

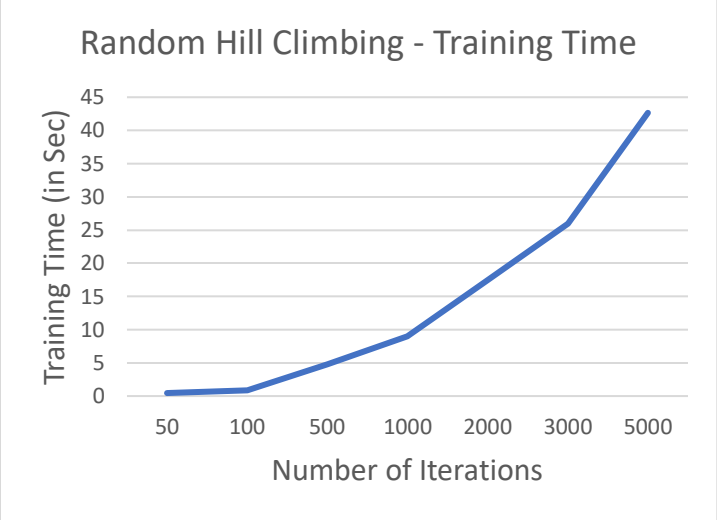
Analysis

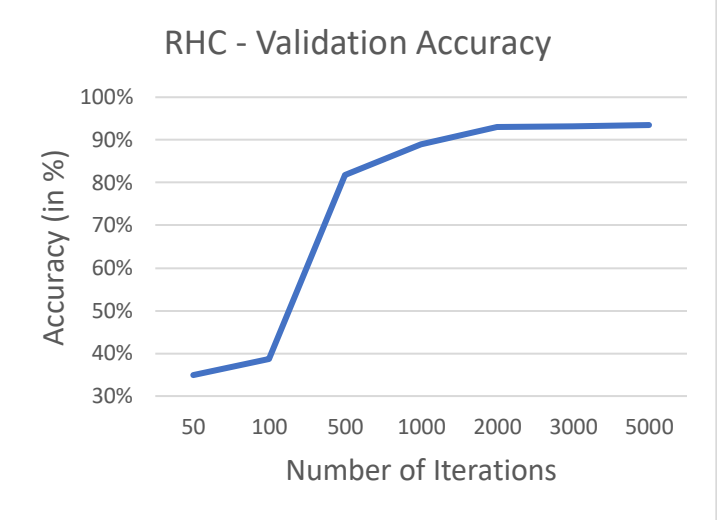
Data is split 60-20-20, 60% training, 20% validation, 20% test data. The hyperparameters for the neural network have already been set, so only the algorithm hyperparameters will be tuned. The validation set is used to tune the hyperparameters and pick an algorithm and the held-out test set to estimate the final performance of the model and compare it to backpropagation. The objective function is optimized, which means finding the best weights for our neural network using randomized optimization.

There are two graphs for each algorithm:

1. Training Time vs Iterations
    a. To see performance of hyperparameters over time
2. Accuracy vs Iterations
    a. To see performance in time/iterations - where results converge and hyperparameter comparison

Randomized Hill Climbing





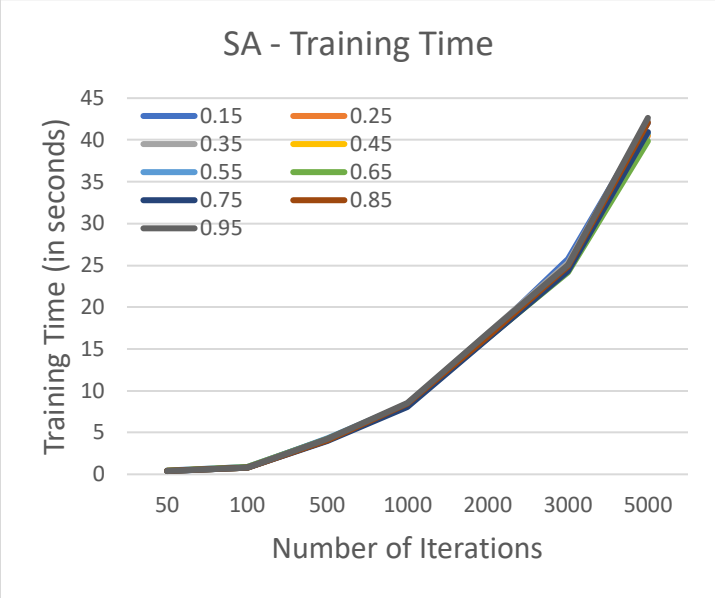
Iterations	Validation	Test	Time
3000	93.17%	94.26%	25.95
5000	93.49%	94.44%	42.65

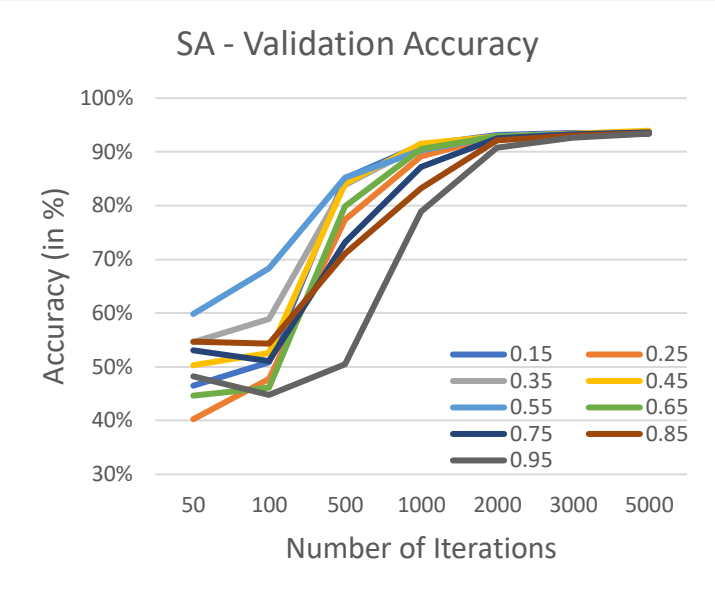
RHC has no hyperparameters to tune. Performance is dependent on iterations. We achieve a pretty good validation accuracy of 93.17% at 3,000 iterations and it seems to converge around that spot as shown by the accuracy curve.

CS 7641 – Randomized Optimization (Assignment 2)

Simulated Annealing

Cooling rates of 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, and 0.95 were used for tuning.





The training times for Simulated Annealing are very similar since it's hard to distinguish between the cooling rates in the training time curve. A cooling rate of 0.55 looks to be the best, as it performs relatively well with low and high iterations. A medium cooling rate makes sense as the algorithm doesn't want to cool too fast or slow and miss local optima.

The optimal solutions of the weights for the neural network seem to converge around 3,000 iterations as shown in the validation accuracy graph. I can see from the table that the validation accuracy for 3,000 iterations is almost similar to 5,000 iterations. It's probably not necessary to spend almost double the training time to achieve similar results.

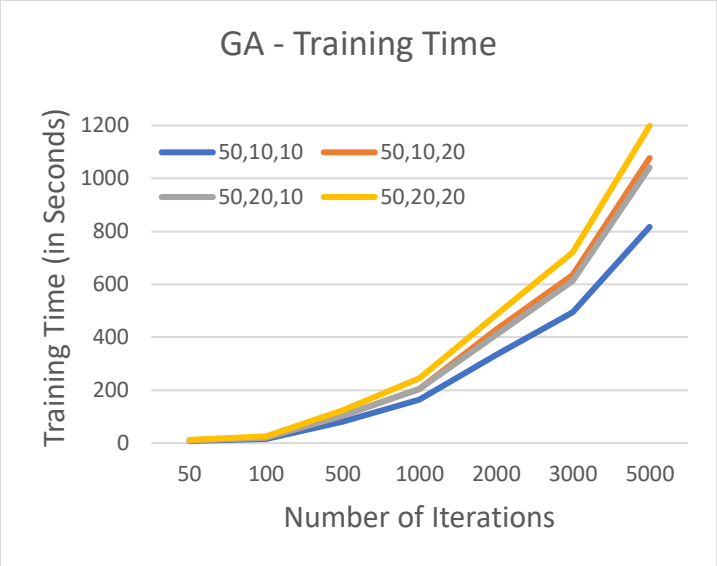
The results compared to the RHC algorithm are almost identical in terms of validation accuracy and training time!

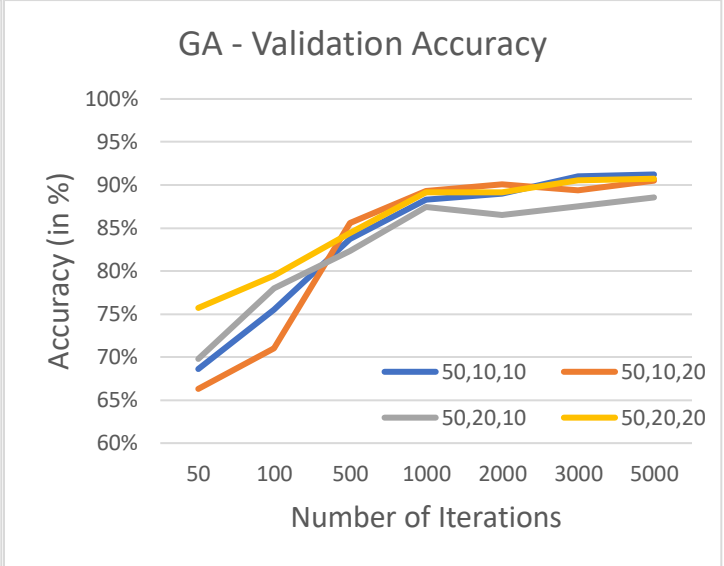
Cooling Rate (CR) = 0.55

Iterations	Validation	Test	Time
3000	93.31%	93.53%	24.44
5000	93.58%	93.94%	40.78

Genetic Algorithm

Again, the parameters in order are: the population size, the number to mate each time, and the number to mutate each step. I ran four different parameter sets (50,10,10), (50,10,20), (50,20,10) and (50,20,20).





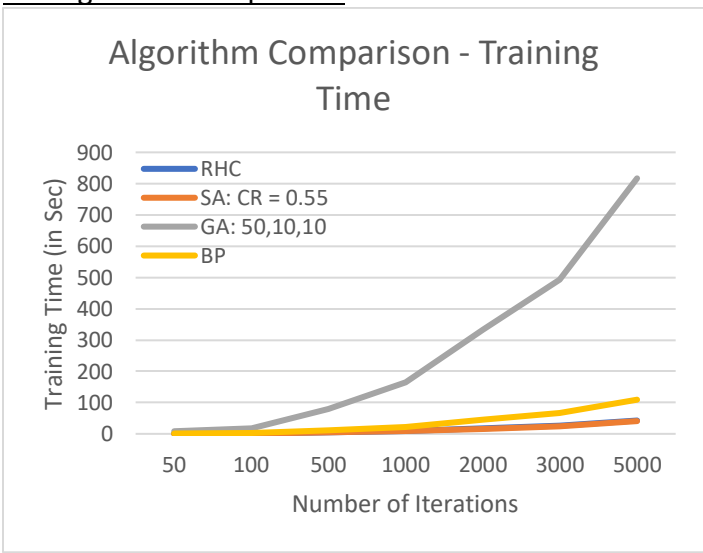
Results here also seem to converge around 3000 iterations, as shown in the accuracy graph. Hyperparameters (50, 10, 10) do well with iterations above 500. (50, 20, 20) performs the best over most iterations, however, it has a higher training time. This because more mating and mutation results in more training time. The tradeoff between accuracy vs time isn't worth it, so (50, 10, 10) is the optimal hyperparameter that will be used for algorithm comparison.

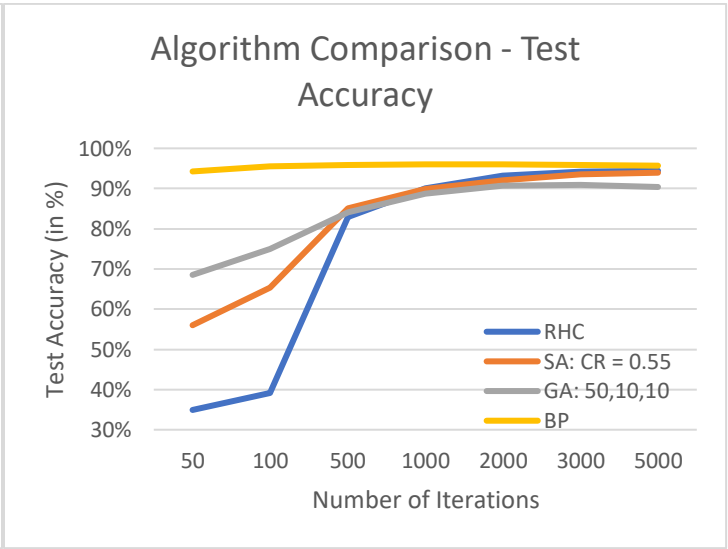
The accuracy results aren't as good as SA and RHC and training time is much higher.

For (50, 10, 10):

Iterations	Validation	Test	Time
3000	91.00%	90.91%	492.94
5000	91.23%	90.32%	816.86

NN Algorithm Comparison





Test Accuracy

Iterations	**RHC**	SA	GA	BP
3000	**94.26%**	93.53%	90.91%	95.93%
5000	**94.44%**	93.94%	90.32%	95.61%

After 500 iterations, RHC and SA perform very similarly. At 3,000 iterations, their training time is around 25 seconds and at 5,000 iterations, 42 seconds. However, the test set shows that RHC has slightly better accuracy. I would not use GA for to find weights for this dataset, because RHC and SA do a much better job in much less time.

How does it compare with Backpropagation (BP)?

Backpropagation is still more accurate by over one percent. Although, it's a bit slower at 109 seconds for 5,000 iterations. A big advantage with Backpropagation is that it doesn't need that many iterations to achieve a high accuracy. Using backpropagation, 50 iterations already achieves over 93% accuracy. Backpropagation uses far less iterations 50 vs 3,000 (~40x) to achieve similar accuracy of ~93% as RHC and SA. The training time for Backpropagation is only 1.16 seconds at 50 iterations. Relative to accuracy, it's almost 25x faster than RHC and SA, 1.16 secs vs 25 secs.

Part 2: Randomized Optimization for Toy Datasets

The following toy dataset optimization problems are used with each one highlighting a specific algorithm:

1. Continuous Peaks – Simulated Annealing
2. Flip Flop – MIMIC
3. Traveling Salesperson – Genetic Algorithm

Hyperparameter selection for each algorithm was performed in iPython notebooks for each of the problems in the Part 2 folder. Hyperparameters were selected based on best performance in accuracy and time vs. iterations.

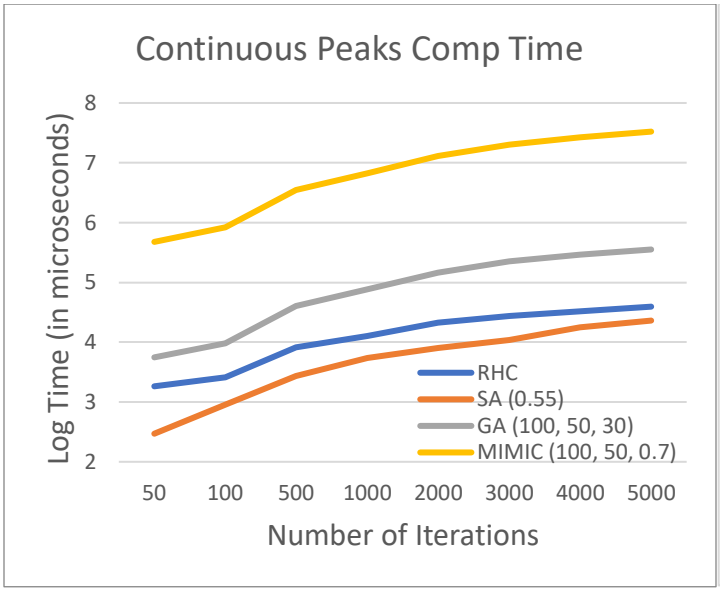
The following hyper parameters were tested:

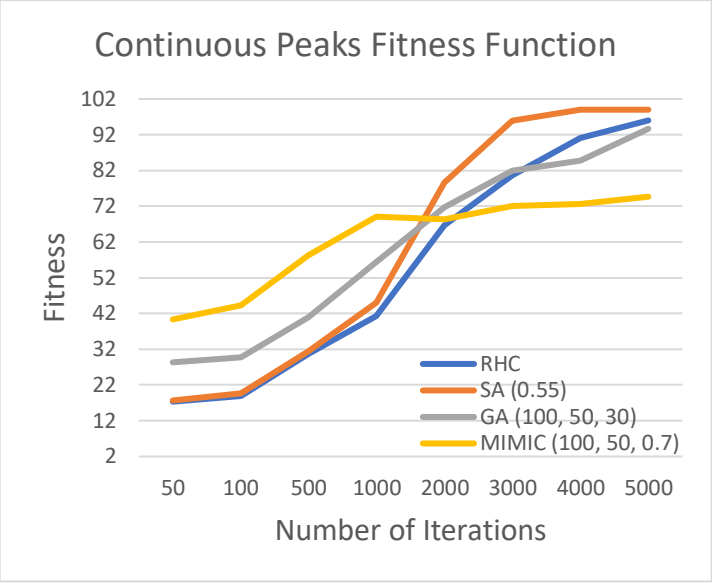
- Randomized Hill Climbing: none
- Simulated Annealing: Cooling rates [0.15, 0.35, 0.55, 0.75. 0.95]
- Genetic Algorithms: [100], [50,30,10], [50,30,10]
    - Parameters in order are: the population size, the number to mate each time, and the number to mutate each step
- MIMIC (100, 50, [0.1, 0.3, 0.5, 0.7, 0.9]]
    - 100 samples to generate each iteration and 50 samples to keep
    - The third parameter m, is a small value that updates the probability matrix of the dependency tree nodes, and makes a new discrete dependency tree distribution.

Continuous Peaks

The continuous peaks problem is similar to the four peaks problem. In this problem, we want to find the global optimal solution with respect to the local optimal solutions.

The number of N = 100 and T = 49 are set for ABAGAIL. 5,000 iterations were run and 3 trials were performed and combined into an average to account for varying difficulties.





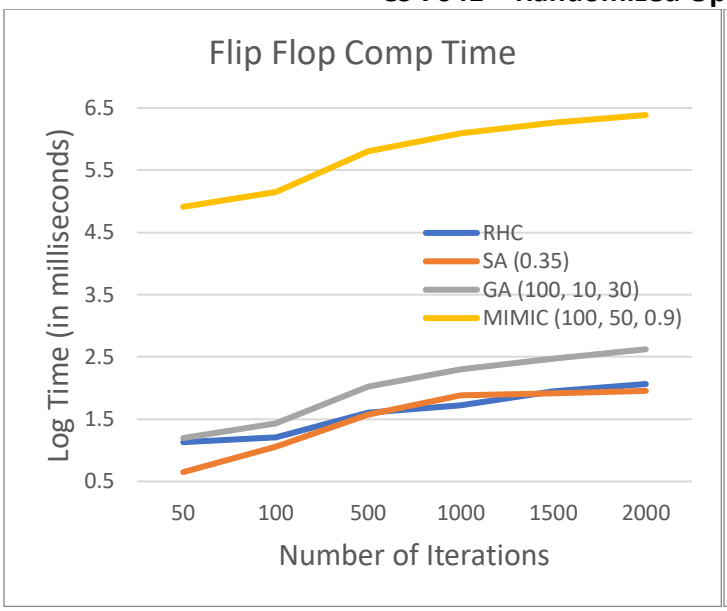
	Fitness	Time (µs)
RHC	96.00	4.59
SA	**99.00**	**4.36**
GA	93.67	5.55
MIMIC	74.67	7.52

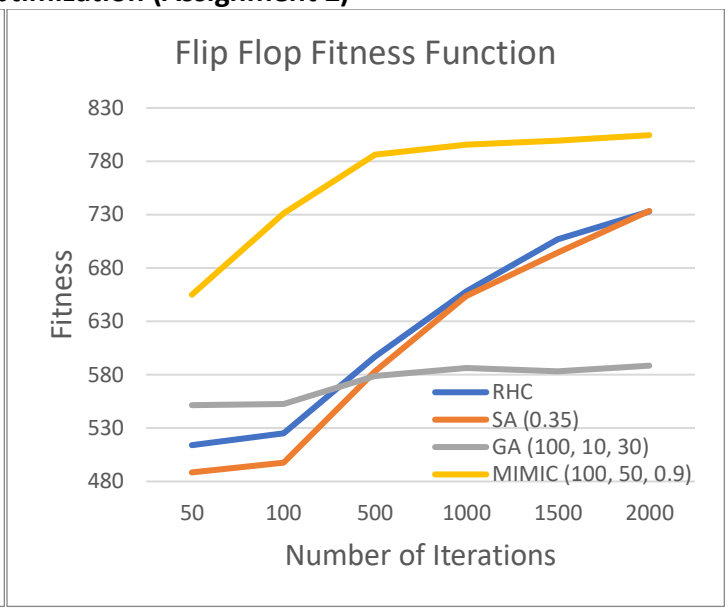
Simulated Annealing performs well with time and converges to a fitness value of 99 early on at around 3,000 iterations. It's a particularly fast algorithm. It does a good job of searching a large space by trading off between exploration and exploitation. Since there are many local optima, it searches the space well, in order to find the global optima. The cooling rate of 0.55 allows the Temperature to not cool too fast and miss local optima. Simulated Annealing is instance based, which results in very fast computational times. GA and MIMIC may have a hard time modeling a continuous random distribution like continuous peaks.

Flip Flop

Flip flop is an interesting problem of bits. It's basically a string of beads with length N where each bead is either 0 or 1. The optimal solution is where every bead is opposite or flipped. When adjacent beads are different it's optimal. When all beads are opposite, it's optimal, anything else is locally optimal or not optimal.

The number of N = 1000. 2,000 iterations were run and 2 trials were performed and combined into an average to account for varying difficulties.





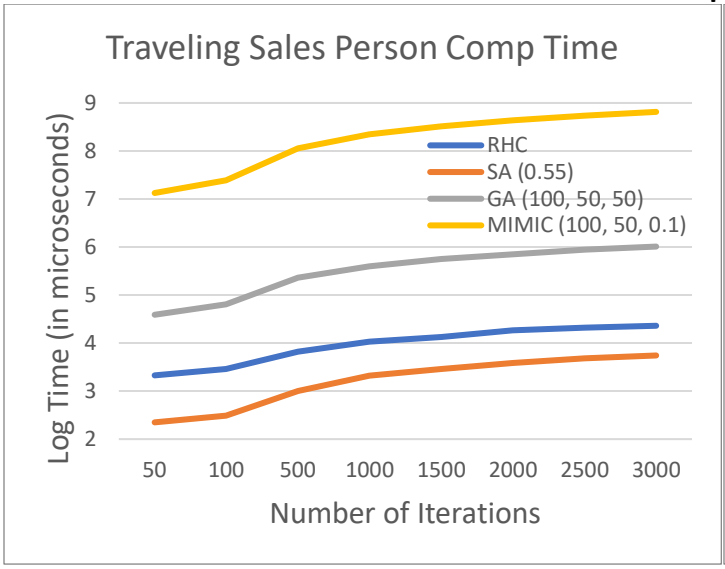
	Fitness	Time (ms)
RHC	733.00	2.07
SA	733.50	1.96
GA	588.50	2.62
MIMIC	**804.50**	**6.39**

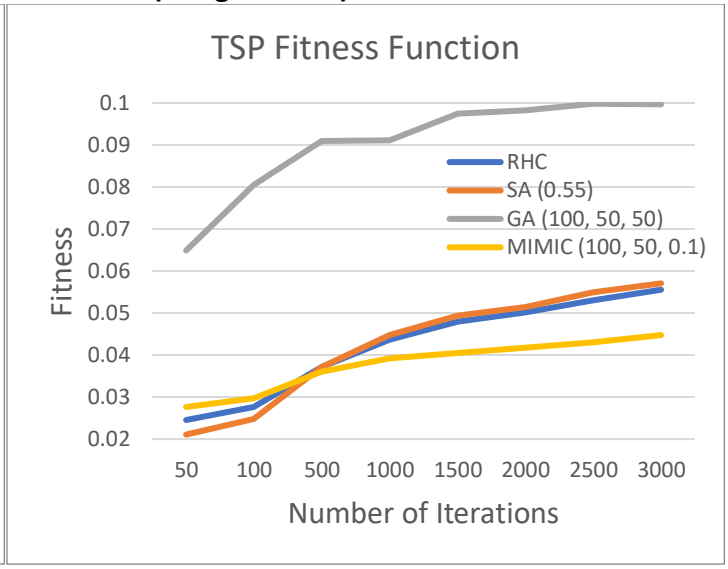
At 2,000 iterations, the function curve looks like it's starting to converge. The fitness function evaluation for MIMIC is much better than the other algorithms. The log time for MIMIC is much higher, but the tradeoff for fitness is worth it as no other algorithms really come close. MIMIC performs well because it's able to estimate the probability density of this distribution and convey some structure. It's able to learn well from sampling previous distributions and improve overall fitness. It achieves a good fitness early on with 500 iterations. RHC and SA are fast, but could be stuck at local optima in this problem more so compared to continuous peaks.

Traveling Salesperson

The traveling salesman problem is a classic optimization problem, which describes a salesman who must travel between an N number of cities, visiting each city once, and end up at the starting point. The objective is to minimize cost or distance traveled between the cities. The problem is interesting because it's considered a hard problem and has real-life applications, such as finding optimal routes for salesmen or airplanes.

The number of cities is set to N = 100. 3,000 iterations were run and 5 trials were performed and combined into an average to account for varying difficulties. The best performing hyperparameters were selected and compared here:





	Fitness	Time (µs)
RHC	0.0556	4.36
SA	0.0571	3.74
GA	**0.0997**	**6.01**
MIMIC	0.0448	8.81

Genetic Algorithm performed by far the best with a fitness value of 0.0997 at 3,000 iterations. At 2,500 iterations, the function looks like it was already starting to converge. Genetic Algorithm performed the second worst in time, but the tradeoff between computation and fitness is definitely worth it. The Genetic Algorithm is able to model the distribution of the traveling salesperson well. It improves a lot from 100 to 500 iterations, so the GA already has a pretty good fitness population by 500 iterations. MIMIC is the worst performing algorithm since it has a high computation time as well as the worst fitness over most iterations.

<u>Conclusion</u>

Algorithm	Type	Problem	Computation Time
RHC	Instance Based		Fast
SA	Instance Based	CP	Fast
GA	Distribution Based	TSP	Medium
MIMIC	Distribution Based	Flip Flop	Slow

For Simulated Annealing there is a recurring trend of a good cooling rate of 0.55. We want to decrease the temperature slowly, because it gives the system an opportunity to explore before it cools down, otherwise we might get stuck at points that aren't optimal.

Overall, RHC and SA produce similar results in terms of accuracy and computation since SA uses hill climbing.

Randomized hill climbing and Simulated Annealing have very fast computation times because they're instance based. Genetic Algorithms and MIMIC are distribution based and have slower times, with MIMIC being much slower. The main benefit of MIMIC is that it's able to directly model distribution and convey structure.