Anmol Chhabria – achhabria7
Assignment #2
CS 4641

Randomized Optimization Assignment

Dataset
For this assignment, I decided to use the *EEG MOOC* dataset. This dataset is from a study conducted on 10 college students while they watched MOOC videos. This dataset analyzes the electroencephalogram (EEG) signals in a student's brain while the student watches videos. The data has approximately 12,000 instances. The dataset contains a total of 13 attributes. The output classifier is a self-reported response of whether a person was confused while watching a video or not.

Background
This project uses randomized hill climbing, simulated annealing, and genetic algorithms to find the optimal weights neural network instead of backpropagation. Further, the randomized hill climbing, simulated annealing, genetic algorithms, and MIMIC search techniques will each be applied to 3 different optimization problems.

The hill climbing algorithm starts with a random point and using its neighbors, it finds a better fitness function. It continues going uphill until it reaches a "peak" where no neighbor has a higher value. Hill climbing has the potential to get stuck in a local optimum. Random restarts improve the algorithm by making sure that every new point is farther from the previous one. This approach is more advantageous if the global optima is in the basin of attraction. However, if the global optima are in a "smaller basin", it is basically a needle in a haystack.

Simulated annealing takes hill climbing a step further. An accurate definition of simulated annealing is the balance between exploitation – finding the global optima faster and in less time- and exploration – searching the entire state space. The hyperparameters for simulated annealing are temperature and cooling. The temperature regulates the amount of space explored, whereas the cooling regulates the exploitation. A large temperature implies that the algorithm will randomly search more of the state space, and a large cooling temperature means that it is converging to the current value. The advantages of simulated annealing are that it searches the space for the higher fitness values with the temperature parameter.

Genetic algorithms take advantage of the better performing 'individuals' of a 'population'. After selecting the most fit 'individuals' it applies crossover to select the best features of each 'individual'. The next step is mutation, which is a relatively small change to the output. For genetic algorithms, the hyperparameters are the population size, the crossover value, and the mutation value. The crossover and mutation values are ratios of the population.

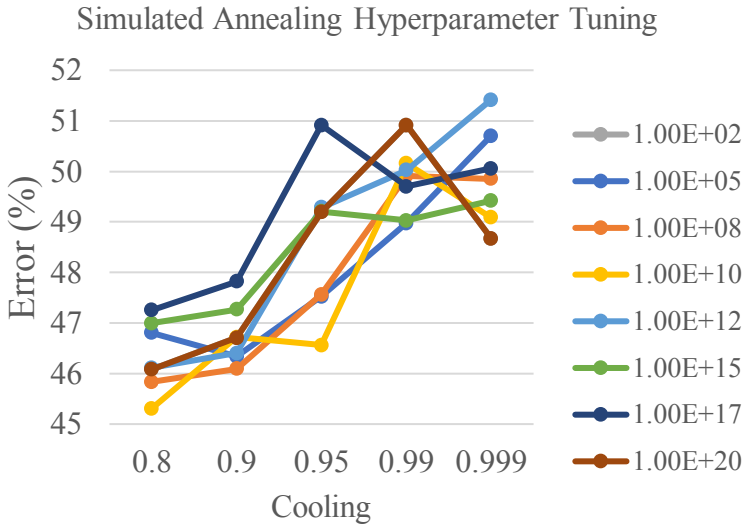
Mutual-Information-Maximizing Input Clustering (MIMIC) adds structure to these random search algorithms. MIMIC is an algorithm that uses the samples of region space that is most likely to contain the global optimum for the fitness function. The other component of MIMIC is a density estimator that is used to understand the structure of the input space. In other words, MIMIC uses previous iterations to understand 'what worked before' and 'what did not' as well as computes the probability density of a value in the sample space. These two additional components differentiate MIMIC from the previously described search algorithms.

Neural Network: Optimal Weights
In the Supervised Learning Assignment, I analyzed the performance of Neural Networks with backpropagation on the *EEG MOOC* data. The structure of the neural network was the following: 13 nodes in input layer, 7 nodes in hidden layer, and 1 output layer. In the neural networks run for Assignment 1, the lowest error I obtained was 28.5%, and the number of nodes in the hidden layer was a ((# of attributes + #of classes)/2).

The objective of this problem is to find the optimal weights of a neural network using simulated annealing, genetic algorithms, and randomized hill climbing. The usual process of finding the optimal weights for a neural network is backpropagation, an algorithm that uses gradient descent to do so. It used so that a neural network can eventually learn any input and output mappings. The training and testing data was 70/30, respectively, like Assignment #1.

Optimizing Hyperparameters

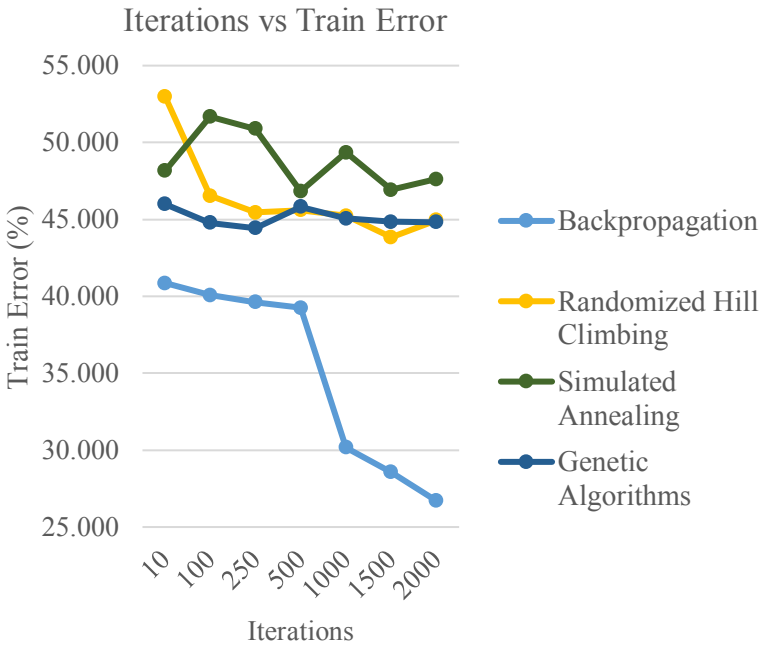
A 10-fold cross-validation was used to determine the hyperparameters. For simulated annealing, 35 different combinations of the temperature and cooling rate were tested over 1000 iterations. The best performing temperature and cooling were 1.00e10 and 0.8 respectively. As the graph shows, this temperature was in the mid-range and the cooling was the lowest. The temperature performed the best, since it gave the optimal balance for exploration. If more cooling rates were tested, it is possible that they could have given a better result due to more even exploitation.

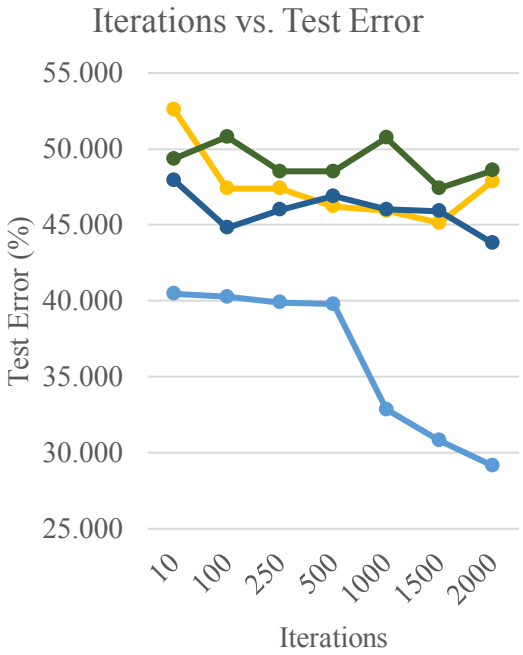


The genetic algorithm hyperparameters were also run over 1000 iterations. The optimal genetic algorithm hyperparameters determined were 0.25 population size ration, 0.02 crossover ratio, 0.02 mutation ratio.

Running Random Search Algorithms

After finding the hyperparameters for simulated annealing and genetic algorithms, all three search algorithms were run over 10, 100, 250, 500, 1000,1500, and 2000 iterations. The results for the training and testing error are indicated below. These were compared to the original backpropagation results from Assignment 1.

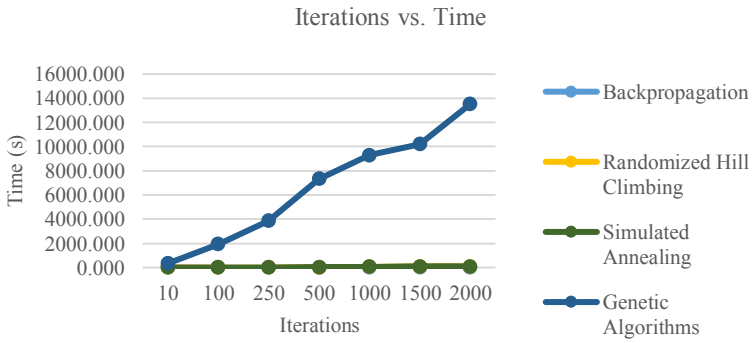




Backpropagation performs the best in finding the optimal weights for the neural network. The lowest error is 29% test error and 26% train error at 2000 epochs. Among the random search algorithms, genetic algorithms generally have the lowest test error throughout, performing best at 1500 iterations. Genetic algorithms have a test error of 45% and train error of 44%. Randomized hill climbing has train and test errors that approximate that of simulated annealing at 44% and 45%, respectively. The learning curve shows some overfitting after 2000 iterations of training. Therefore, the optimal result can be obtained at 1500 iterations. Given that randomized hill climbing performs better than expected, it can be concluded that the data has very few local maxima values. Simulated annealing shows a behavior of high exploration, since the graph for train and test error do not show a convergence. More iterations could have potentially yielded better results for simulated annealing. The lowest test and train error are 47% and 46% for simulated annealing.

Considering error and overfitting, we can conclude that randomized hill climbing yields the lowest test error and genetic algorithms converge to a lower error more quickly. Because of the convergence, I would select genetic algorithms for this problem.

Time(s)				
Iterations	**BP**	**RHC**	**SA**	**GA**
10	0.230	0.673	0.673	339.908
100	2.150	5.196	5.196	1932.556
250	5.430	12.640	12.640	3864.675
500	0.110	25.172	25.172	7327.178
1000	21.480	50.302	50.302	9297.865
1500	19.000	75.638	75.638	10216.094
2000	19.000	101.496	101.496	13515.863



As the table and the graph above show, the training time for genetic algorithms are exponentially increasing with the number of iterations. Simulated annealing and random hill climbing also show exponentially increasing times, but their runtime is only about 101 seconds for 2000 iterations. On the other hand, genetic algorithms have a runtime of 13000 seconds (about 3 hours and 45 minutes). Backpropagation is the best performing with lowest error and runtime.

Optimization Problems

Approach

The three different optimization problems I chose to analyze were the Continuous Peaks Problem, Knapsack Problem, and the Traveling Salesman Problem. For each optimization problem, 4 random search algorithms were implemented. The parameters for each search algorithm were tuned for each problem over 1000 iterations. For randomized hill-climbing, there were not any hyperparameters to be tuned. For simulated annealing, a total of 40 combinations were tested for the temperature and cooling hyperparameters. For genetic algorithms, a total of 309 combinations were tested for population size, crossover, and mutation hyperparameters. For MIMIC, a total of 27 combinations were tested. Once the optimal hyperparameters for each problem were determined, different problem sizes and iterations were tested. The problem sizes ranged from 10 to 200, while the iterations ranged from 50 to 12,500. These varied for every problem.

Continuous Peaks Problem

The continuous peaks problem belongs to the peaks set of problems. The original problem in this family of optimization problems is the four-peaks problem. The four-peaks problem uses a bitstring composed of N binary elements and a trigger value T. The problem maximizes fitness if the string can result in up to T 1's or 0's and get the opposite value for the rest of the string. In addition, the function must obtain the 100 reward to maximize fitness, which occurs if the number of 0's is greater than T and the number of 1's in the string is greater than T.

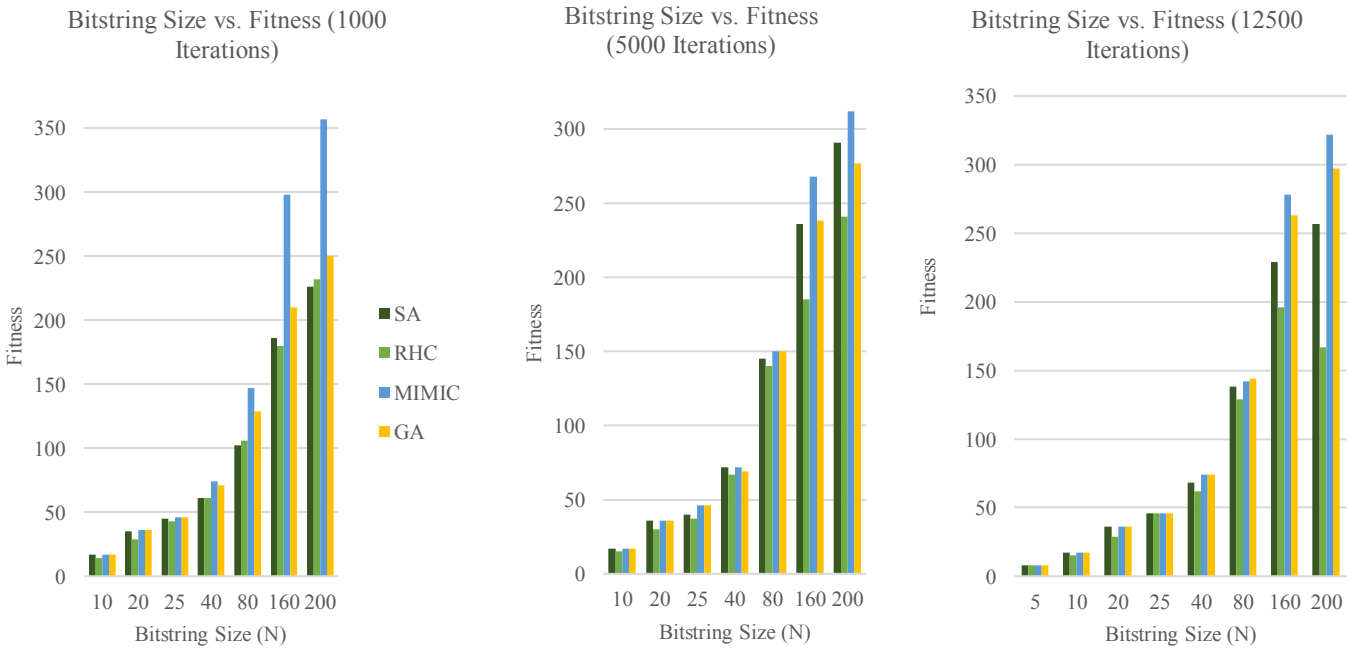
Now, we can go back to our initial problem – continuous peaks problem. Instead of forcing the bit values (0 and 1) to be at opposite ends of the bitstring, the continuous peaks fitness value is maximized where there are greater than T contiguous 0's as well as greater than T contiguous 1's. The optimal parameters for the continuous peaks problem are recorded in the table below. These were evaluated over 1000 iterations where N = 60 and T = 6 for the continuous peaks problem. Following the tuning of parameters, the problem was run for N with values of 10, 20, 25, 40, 80, 160, 200. For each run, the value for T was equivalent to N/10.

Algorithm	Hyperparameter Values	Fitness
Randomized Hill Climbing	N/A	87
Simulated Annealing	Temperature: 1.00E15, Cooling: 0.9	112
Genetic Algorithms	Population: 2500, Crossover: 150, Mutation: 150	112
MIMIC	Samples: 5000, To Keep: 100	109

Advantages of Simulated Annealing

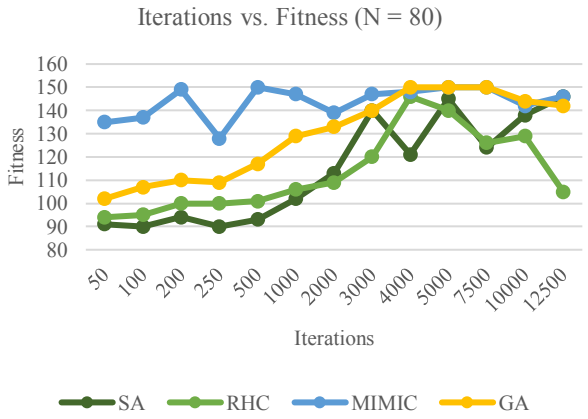
For this problem, I chose to illustrate the advantages of simulated annealing. In the Continuous Peaks problem, the fitness of a bitstring is dependent on the bit values, so MIMIC would have advantages to the problem. However, I will illustrate the advantages that would result from selecting simulated annealing over MIMIC. With the heating and cooling parameters simulated annealing has a lower probability of staying at a local maxima, and it explores more of the region space.

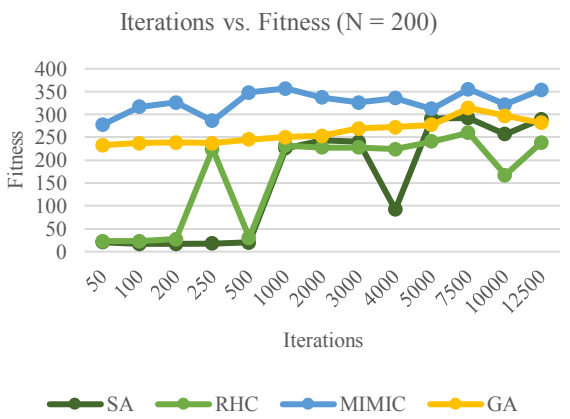
Randomized hill climbing performs the worst over 1000, 5000, and 12500 iterations. It starts out performing like simulated annealing, but as the bitstring size increases, simulated annealing performs better than randomized hill climbing. With the temperature parameter, simulated annealing explores more of the region space, and it exploits the optimal function. As the bitstring size increases, the performance of MIMIC improves exponentially. For 1000 iterations, MIMIC outperforms the other algorithms by a lot. This is a consistent trend for 5000 and 12500 iterations as well, so varying the number of iterations does not change the result. However, we notice that as the number of iterations increases from 1000 to 5000, simulated annealing has an improving performance. This is also reflected in the data for 12500 iterations.



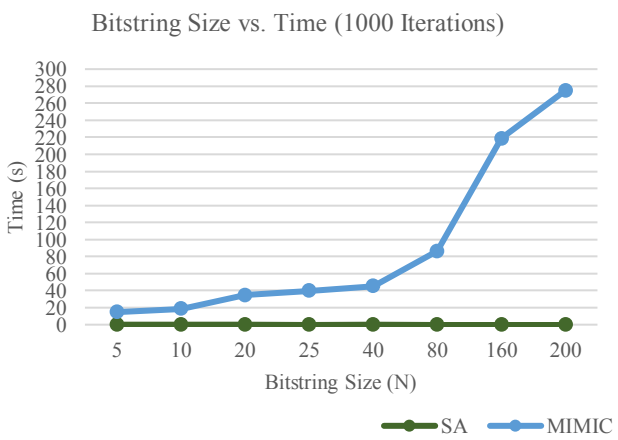
Further, the fitness compared to different iterations over constant sizes of the bitstring are analyzed. For N = 80, simulated annealing converges to the optimal fitness value of MIMIC at 12500 iterations. For N = 160, simulated

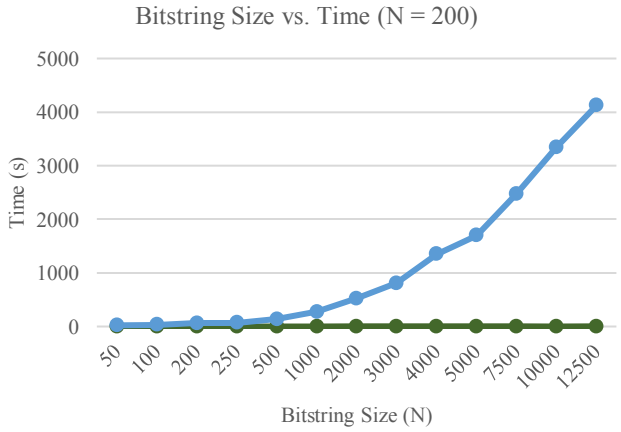
annealing does not converge to the optimal fitness value of MIMIC even at 12500 iterations. From the bar graphs and the plots, as the bitstring size increases, it takes more iterations for simulated annealing to achieve the highest fitness value, which is that of MIMIC.





With high performance comes high computation time, as far as MIMIC is concerned. The graph below shows time taken in seconds for all algorithms over a constant 1000 iterations. After analyzing the increasing bitstring size over 1000 iterations, MIMIC's run time increases exponentially. Conversely, after analyzing increasing iterations over size of bitstring at 200, MIMIC shows an increasing runtime while simulated annealing preserves its linear runtime. Since more iterations of simulated annealing tend to converge to the similar fitness value as MIMIC with less total runtime, for large bitstring sizes, simulated annealing can yields similar fitness results in more iterations and less total runtime. This allows simulated annealing to solve large problems in less time.





Knapsack Problem

The knapsack problem is a typical optimization problem that assigns each of N items in a bag a weight and a value. The knapsack has a maximum weight, and the fitness function is to maximize the total value of the knapsack. The objective of each random search is to maximize the fitness function. The optimal values for the Knapsack Problem are shown below. These were obtained after testing over 1000 iterations and N = 40.

Algorithm	Hyperparameter Values	Fitness
Randomized Hill Climbing	N/A	3564.564
Simulated Annealing	Temperature: 100, Cooling: 0.99	3970.330
Genetic Algorithms	Population: 1500, Crossover: 120, Mutation: 15	4252.393
MIMIC	Samples: 2000, To Keep: 500	4272.365

Advantages of MIMIC

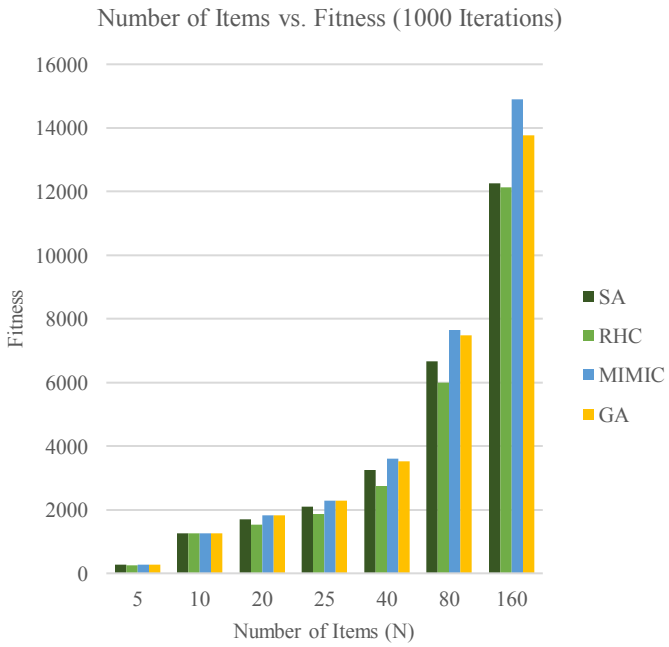
When selecting the parameters for MIMIC, it is important to highlight that higher values for the samples in combination with low values for number of samples to keep perform well.

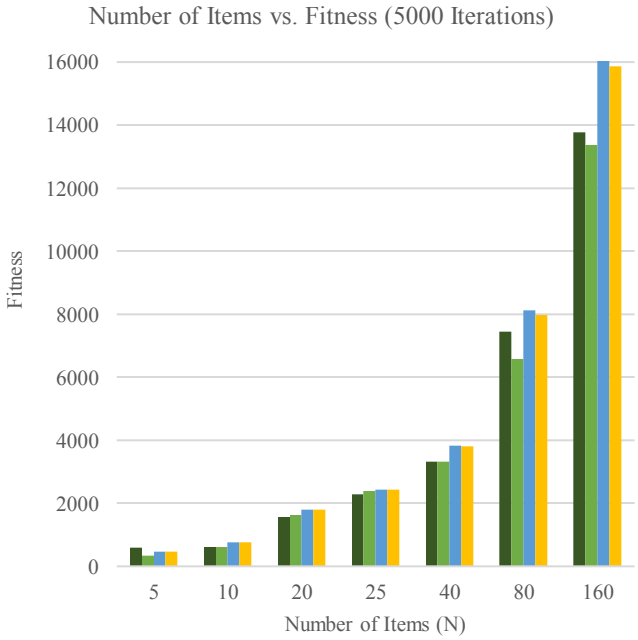
Increasing the number of samples to keep for a constant sample size reduced the value. For every iteration run, if the number of samples to keep is large, then the algorithm learns less about the 'most fit' in region space as opposed to when the number of samples to keep is small. However, there is a balance that needs to be struck to in choosing the number of samples to keep. With 2000 samples, reducing the To Keep number to 100 or 200, reduces the fitness value. The same happens if the number of samples to keep increases to 1000.

MIMIC is an algorithm that finds the optimum by creating and sampling different probability distributions of the candidate solutions. MIMIC also uses structure from the previous iterations to maximize a fitness function for a problem and combines information from all the maxima. In the knapsack problem, the total number of items is constrained by the maximum weight allowed on the knapsack as well as the values of other items in the knapsack. In other words, the number of items is depended on each other. In the knapsack problem, the most important feature is the dependency tree structure that MIMIC provides to the solution.

Different values for the number of items in the knapsack problem were graphed against their respective fitness values. These were first analyzed for 1000 iterations. Randomized hill climbing performs worse as the number of items increases. As the region space increases, the algorithm is not able to find the global optimum. Simulated annealing follows a similar trend. This is since the temperature is 100, and as the number of items increases, it does not explore as many solutions.

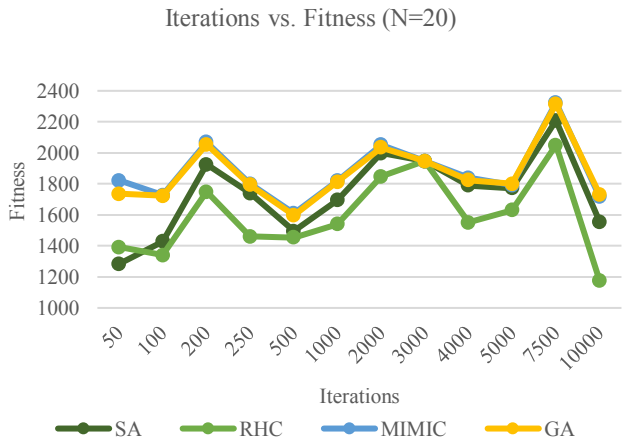
For 1000 iterations, MIMIC performed better than the rest of the algorithms. Genetic algorithms obtained similar fitness values to MIMIC, but with less iterations (1000 compared to 5000) the difference was larger. The same process was repeated for 5000 iterations. Over 5000 iterations, genetic algorithms perform almost as well as MIMIC. The commonality in MIMIC and genetic algorithms is that they can choose the most fit elements in a space, whether it is the 'individuals and their offspring' in a 'population', or the best 'samples' in a 'sample space'. To further compare their (MIMIC and genetic algorithms) performances in detail, I analyzed the data for a constant number of items over varied iterations.

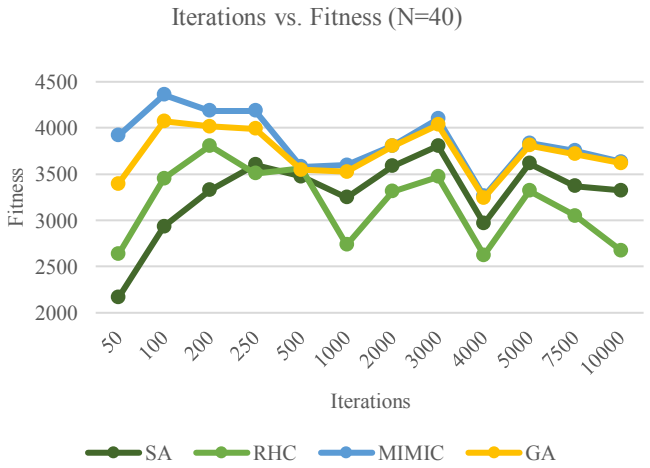


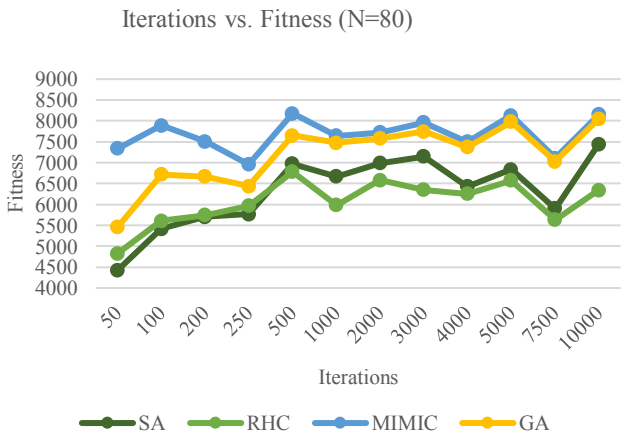


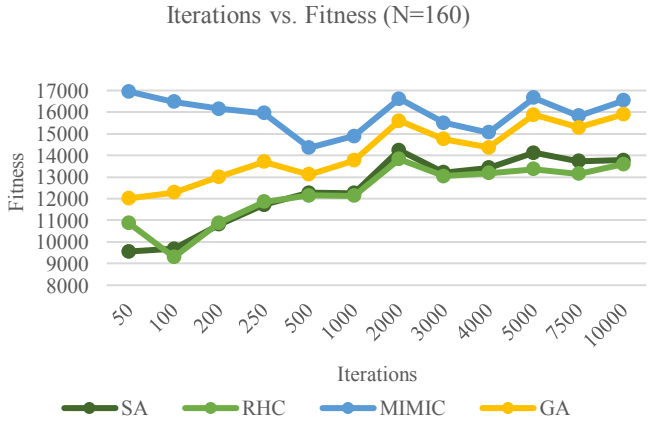
The fitness for 20, 40, 80, and 160 items were analyzed over multiple iterations ranging from 50 to 10000. For 20 items, MIMIC and genetic algorithms have very similar fitness values. For 40 items, MIMIC outperforms genetic algorithms up until 250 iterations, after which they output basically the same fitness values. For 80 items, MIMIC outperforms genetic algorithms up until 1000 iterations, after which they both output similar values. For

160 items, it is important to highlight that genetic algorithms do not converge to the fitness values achieved by MIMIC. Therefore, as the number of items increases, it takes genetic algorithms a higher number of iterations to converge to the optimal fitness value. MIMIC can find the optimal fitness value in the least number of iterations, regardless of the number of items. This is expected given that MIMIC takes into consideration probability distributions from the dependency trees of items.









In terms of fitness, MIMIC is the better performer. This leads us to compare runtimes for the algorithms. The table below shows the run time for all 4 algorithms over 1000 iterations. MIMIC's run time is exponentially increasing. This runtime is worth the tradeoff, especially as the number of items increases. If the number of items is low, then genetic algorithms would converge to the similar fitness with a higher number of iterations, which would still account for less time than MIMIC.

N	RHC Time(s)	SA Time(s)	GA Time(s)	MIMIC Time(s)
5	0.000182394	0.000338091	0.705000000	5.110000000
10	0.000109316	0.000231197	0.815000000	9.180000000
20	0.000138252	0.000267197	0.939000000	19.400000000
25	0.000176487	0.000279345	1.070000000	27.200000000
40	0.000206544	0.000307292	1.370000000	48.300000000
80	0.000349046	0.000387728	1.630000000	140.000000000
160	0.000659180	0.000566940	3.200000000	433.000000000

Traveling Salesman Problem

The traveling salesman problem finds the shortest route for N cities, given the distances between each pair, such that each city is visited once and the route starts and ends at the same location. The fitness function thus would be the inverse of the total distance, since in this case, the objective is to maximize the fitness function. The parameters were tuned over 1000 iterations for N = 50. The optimal values for the Traveling Salesman Problem are shown below.

Algorithm	Hyperparameter Values	Fitness
Randomized Hill Climbing	N/A	0.093806
Simulated Annealing	Temperature: 1.00E+6, Cooling: 0.8	0.096225
Genetic Algorithms	Population: 1500, Crossover: 90, Mutation: 90	0.169399
MIMIC	Samples: 5000, To Keep: 500	0.122073

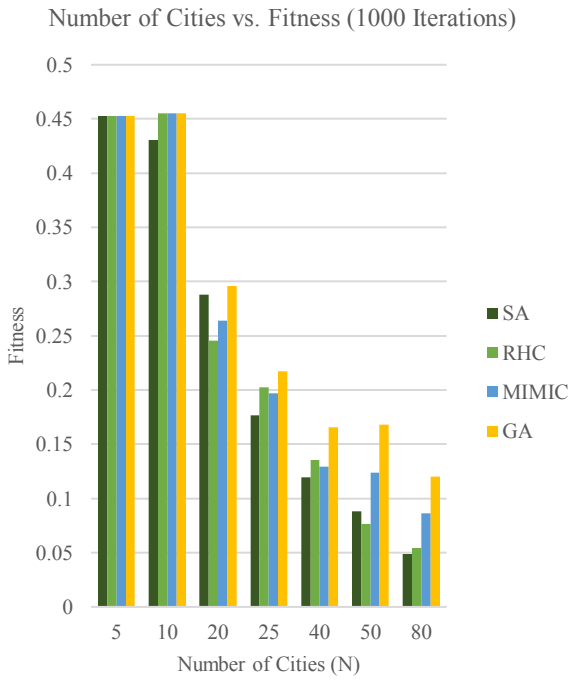
Advantages of Genetic Algorithms

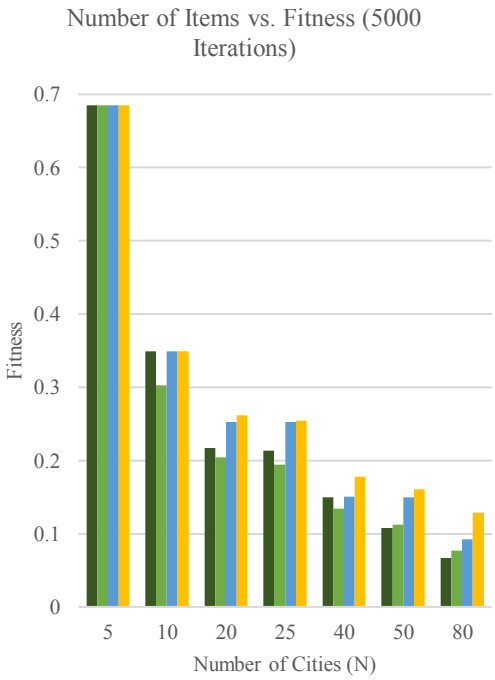
When selecting the genetic algorithm parameters for the traveling salesman problem, I noticed that mid-size sample sizes with low crossover and low mutation performed better. The top fitness values obtained for the genetic algorithms over 1000 iterations, had a crossover and mutation that was 5% or less than 5% of the population size. This makes sense given that you want to pick the best individuals to crossover, and keeping the crossover number small makes sure you can pick the best individuals. These best individuals then replace the 'least fit' individuals in the original population. The purpose of mutation is to add diversity to the new population and inhibit premature convergence. In this case, the mutation value is equal to crossover, indicating that mutation improved the fitness of the children.

I chose to analyze genetic algorithms for this problem because the solution tour can be derived from a combination of subtours. Genetic algorithms function in a similar way, since they use the 'most fit' individuals to create better solutions. In addition, many of the best solutions found for the traveling salesman problem have been found by a genetic algorithm method.

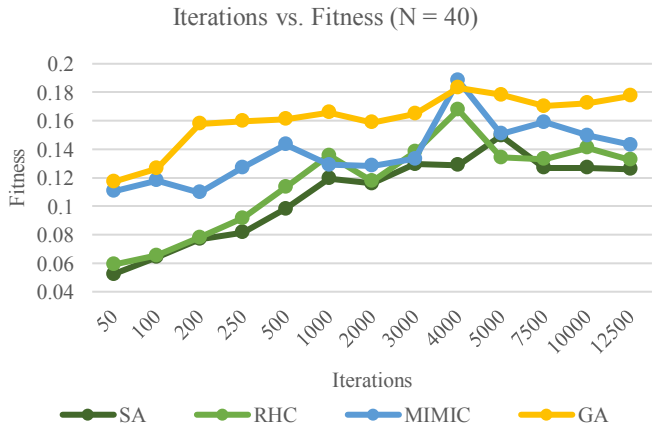
From both the graphs, we can observe that for 5 cities all 4 algorithms output the same fitness value. As the number of iterations increases, fitness value for N (number of cities) improves. For 1000 and 5000 iterations, regardless of problem size, genetic algorithms outperform the other algorithms. In comparison to the continuous peaks problem and the knapsack problem, the Number of Cities vs. Fitness graph shows a decreasing trend. As previously mentioned, this is because the shortest route is that with the shortest distance. To make sure this is problem is maximizing fitness, the fitness value is the inverse (1/shortest distance). As problem size increases, the number of possible solutions increases, so performance is exponentially decreasing.

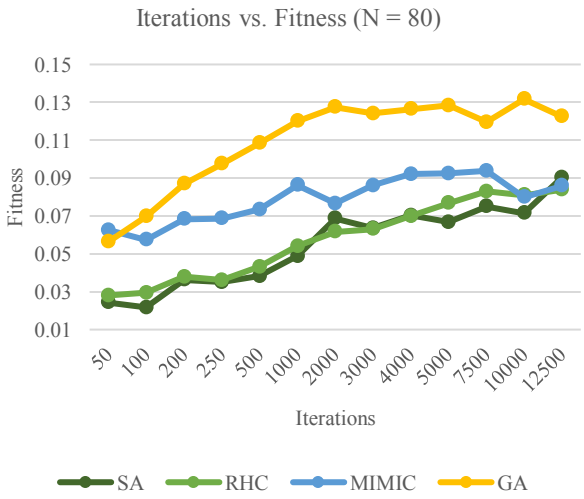
Simulated annealing generally performs the worst when the number of cities is 25 and higher at 1000 iterations and 50 cities and higher at 5000 iterations. This could be a result of a relatively low temperature of 1.00E+6. Increasing problem sizes, would do better with a higher exploration rate and a lower cooling rate, allowing it to search for more solutions and reducing exploitation at local optimums. This concept is illustrated by the fact that randomized hill climbing outperforms simulated annealing, because randomized hill climbing is exploring more of the state space with random restarts. Surprisingly, MIMIC is not the overall top performer for this algorithm, indicating that structure is not a benefit to the traveling salesman problem. The most beneficial features to the traveling salesman problem are the selection and crossover aspects of genetic algorithms.





Moreover, the fitness value for a constant number of cities over multiple iterations were analyzed. As the number of iterations increases, simulated annealing and randomized hill climbing show increasing performance. With more iterations, randomized hill climbing can explore more of the region space. For both graphs of Iterations vs. Fitness, genetic algorithms converge to a fitness value before randomized hill climbing and simulated annealing. For 40 cities, MIMIC converges at 5000 iterations while genetic algorithms converge at 4000 iterations. For 80 cities, MIMIC converges at 3000 iterations, while genetic algorithms converge at 2000 iterations.





In terms of fitness, genetic algorithms outperform the other algorithms. Genetic algorithms also have a much lower run time than MIMIC. The table below shows run time for each algorithm over 10,000 iterations for different numbers of cities. Genetic algorithms, on average, take about 500 times more than simulated annealing and about 5000 times more than random hill climbing. However, it is important to highlight that on average genetic algorithm took 11.5 seconds for 10,000 iterations, which is amazing run time considering the number of iterations. On the other hand, MIMIC has run time that is about 130 times more than genetic algorithms.

Overall, genetic algorithms are the winning approach here. This agrees with other research in the field, since genetic algorithms are used for scheduling and large scale optimization problems.

Number of Cities	RHC Time(s)	SA Time(s)	GA Time(s)	MIMIC Time(s)
5	0.001812163	0.0247	10.3	270
10	0.002325476	0.0184	8.5	316
20	0.009096145	0.0181	9.8	456
25	0.004722493	0.0134	5.6	371
40	0.019300000	0.0230	10.4	1260
50	0.007624115	0.0195	17.3	2250
80	0.024200000	0.0333	18.6	5350

Conclusion

In evaluating each algorithm in detail, I realized that these randomized optimization approaches have their benefits and tradeoffs. MIMIC has a high runtime, but it incorporates structure and dependency. If working on a problem where candidate solutions show dependency and runtime is not a concern or problem size is small, MIMIC is ideal. Genetic algorithms outperform MIMIC in scheduling and routing optimization problems, since it adds the feature of crossover. For backpropagation, genetic algorithms had a very high runtime. For the optimization problems, smaller problems, it was in the order of seconds (as opposed to hours for MIMIC sometimes). Simulated Annealing is useful when you expect to have multiple local maxima in the dataset, whereas random hill climbing can be useful in the converse. Given that runtime for both is similar, simulated annealing with the correct hyperparameter configurations can be more effective.

References

De Bonet, Jeremy S., et al. “MIMIC: Finding Optima by Estimating Probability Densities .” Accessed 11 Mar. 2017.

“EEG Brain Wave for Confusion | Kaggle.” EEG Brain Wave for Confusion | Kaggle, www.kaggle.com/wanghaohan/eeg-brain-wave-for-confusion. Accessed 12 Mar. 2017.

Gao, Shang. “Estimation of Distribution Algorithms for Knapsack Problems.” 6th Annual International Conference on Computer Science Education: Innovation & Technology (CSEIT 2015), May 2014, doi:10.5176/2251-2195_cseit15.30.

“Genetic Algorithms.” Introduction to Genetic Algorithms, www.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/hmw/article1.html. Accessed 12 Mar. 2017.

“Heuristic and Metaheuristic Algorithms for the Traveling Salesman Problem.” *SpringerReference*, doi:10.1007/springerreference_72340.

Pushkar. “Pushkar/ABAGAIL.” *GitHub*, 17 Feb. 2017, github.com/pushkar/ABAGAIL. Accessed 12 Mar. 2017.

“Randomized Localized Search as Successive Estimation of Probability Densities.” *Stochastic Processes*, 2004, pp. 93–166., doi:10.1016/b978-190399655-3/50011-6.