CS 7641: Machine Learning

Assignment 2: Randomized Optimization

Ravi Prakash Singh

February 21st, 2019

Introduction

In this assignment, we will explore different randomized search optimization processes. We will be implementing different randomized optimization algorithms and use couple of optimization examples to discuss strength and weakness of each algorithms. Specifically, we will be implementing following randomized algorithms

- Randomized Hill Climbing *(RHC)*
- Simulated Annealing *(SA)*
- Genetic Algorithm *(GA)*
- Mutual Information Maximizing Input Cluster *(MIMIC)*

The assignment is done in two parts. In first part, we will use first three algorithms to find good weights for a neural network. We will be using *Bank Marketing* data, used in assignment 1 to implement first 3 search algorithms. In second part, we will pick three different optimization problems and apply all 4 search techniques to each of them.

Part 1: Randomized Optimization & Neural Network weights

In the previous assignment, we used Bank Marketing Data to explore various Supervised Learning classification techniques.

Data

To recap, the data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed. The data has approximately 45,000 instances of different customers with information on their demographics and campaign related information such as number of contacts, duration of call etc. The classification goal is to predict if the client will subscribe a term deposit.

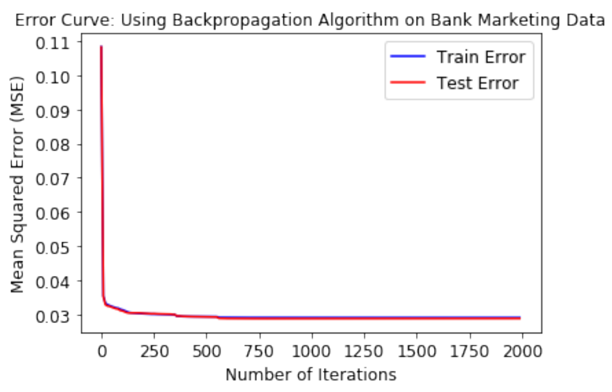
To maintain consistency in our results so that it is comparable with previous assignment, we have used the same split of Train and Test in 80:20 ratio. We have used the same train and test dataset that was used to build neural network in previous assignment. To go one step further, we have directly used normalized train and test dataset used in previous assignment.

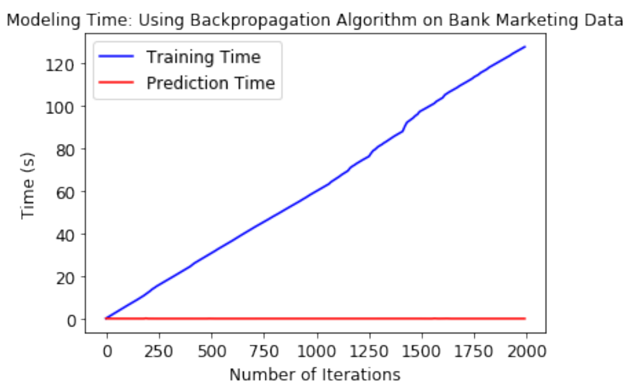
Neural Network: Backpropagation

One of the techniques explored in previous assignment was Neural Networks. In particular, we have used backpropagation technique to find optimal weights for the network. For purpose of this assignment, we will directly use the optimal set of hyperparameters found in previous assignment and we will not be tuning hyperparameters of neural network in this assignment. Following were the optimal network hyperparameters.

- *Number of hidden layers: 4*
- *Size of each hidden layer: (2,2,2,2)*
- *Learning Rate: 1e-4*
- *Activation Function: Hyperbolic Tangent Sigmoid Function*

In previous assignments, we tuned our hyperparameters and evaluated our model using F1-score as our evaluation metric. However, we didn't observe how our Mean Squared Error (MSE) improved as number of iterations for both training and test sets. Therefore, as a first step, we re-run our backpropagation algorithm and find how the MSE improves over number of iterations.



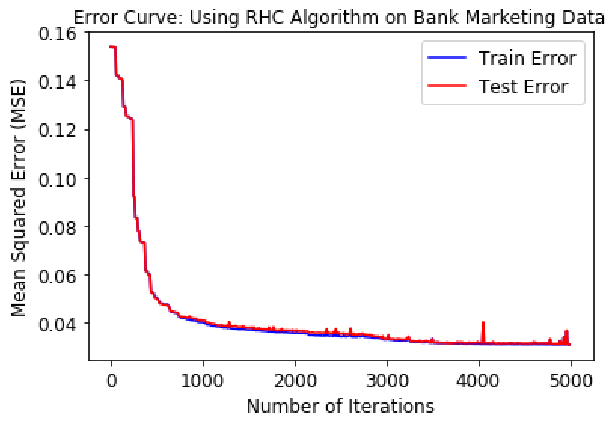


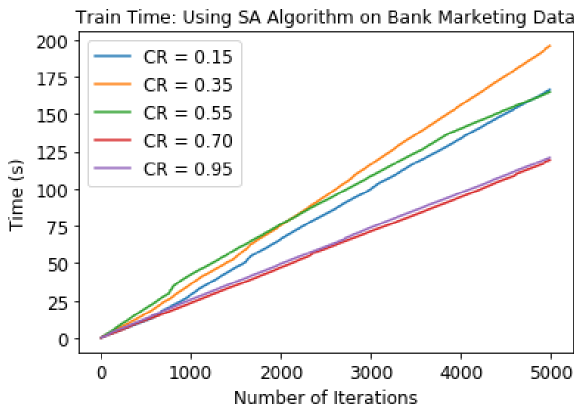
From the above plots we can easily observe that backpropagation algorithm converges very quickly in around 600 iterations. The training time as usual increases with number of iterations.

Now, we move on to other optimization algorithms. Please note that all the algorithm implementations are done using ABAGAIL package. In addition, there is no cross-validation data available; although excellent for analyzing overfitting and generalizing algorithms to data we are yet to see, ABAGAIL simply doesn't have the functionality available.

Randomized Hill Climbing

Random Hill Climbing is a straightforward brute force algorithm to obtain a local optima. The procedure is: Sample p points randomly in the neighborhood of the currently best solution; determine the best solution of the n sampled points. If it is better than the current solution, make it the new current solution and continue the search; otherwise, terminate returning the current solution. RHC randomly starts it's position in order to avoid getting stuck at local optimum. The random start also increases chance to bring the solution closer to the global optimum.





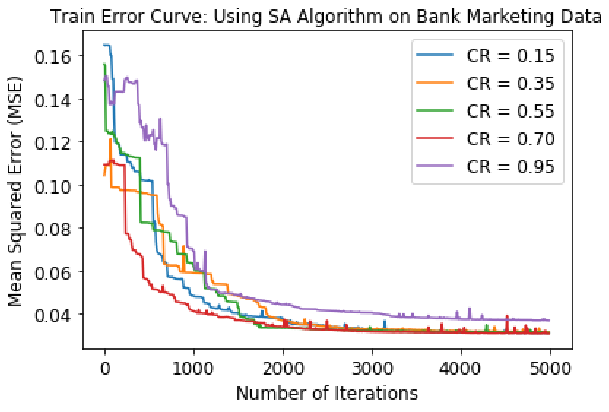
We observe that Randomized hill climbing algorithm takes many number of iterations to converge. As compared to backpropagation algorithm, which only takes around 600 iterations, RHC takes around 4,000 iterations to converge. It might be because RHC algorithm must have chosen a bad initial point to start with. To improve it's performance we can re-run the algorithm at different starting point. In general, we can have many restarts, say 10, and try to pick the best results between the various times the algorithm is run.

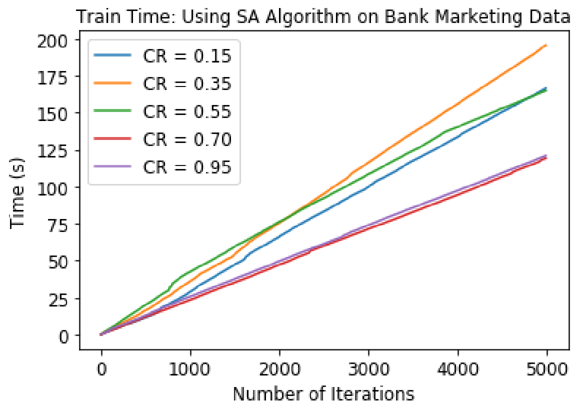
In terms of running time, RHC is much faster than backpropagation algorithm. Running time for 2,000 iterations of backpropagation took around 120s, whereas same number of iterations for RHC took 55s. Therefore, RHC is around 2 times more faster than backpropagation. However, overall it took much longer time for RHC to converge.

Simulated Annealing

Simulated annealing (SA) is a method for solving unconstrained and bound-constrained optimization problems. At each iteration of the simulated annealing algorithm, a new point is randomly generated. The distance of the new point from the current point, or the extent of the search, is based on a probability distribution with a scale proportional to the temperature. The algorithm accepts all new points that lower the objective, but also, with a certain probability, points that raise the objective. By accepting points that raise the objective, the algorithm avoids being trapped in local minima in early iterations and is able to explore globally for better solutions.

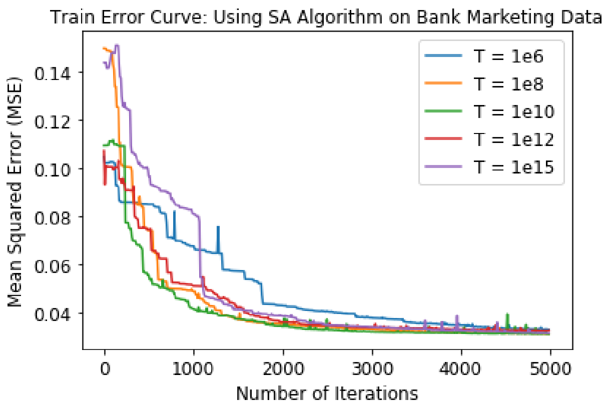
There are two hyperparameters that we can tune in Simulated Annealing algorithm. One is cooling rate (CR) and other one is starting Temperature (T). To observe effect of each hyperparameter, we will vary one of them, keeping other one constant. First, we study effect of Cooling Rate (CR) on the optimization process. For this study, we keep starting temperature fixed to 1e10. Please note that the cooling rate is a constant, between 0 to 1, defining how quickly the temperature is decreasing with each iteration. The decrease in temperature corresponds to a decrease in how much the algorithm can jump between points i.e. lower the temperature at any iteration, the more the algorithm is inclined to move towards points that move us closer towards the goal, increasing the chances of getting stuck in a local optima. The general idea of the algorithm then, is that we sample many random points in the beginning, in an attempt to determine a best optimum to move towards, slowly accelerating towards this optimum, and localizing the space of available points as the algorithm continues.

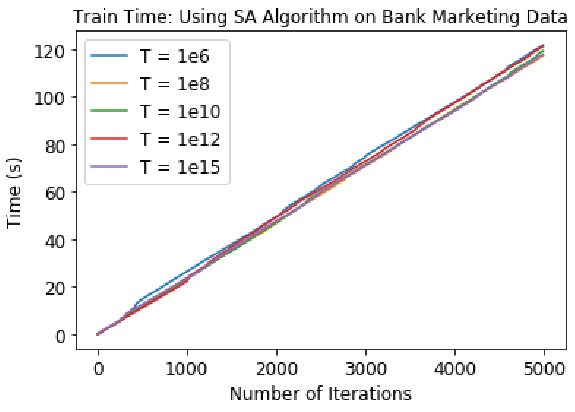




Our analysis on Bank Marketing Dataset shows that, no matter what cooling rate we choose, almost all the algorithms converge towards same optimal mean squared error value. Interestingly, we found that when the cooling rate is too high i.e. 0.95 algorithm converged to relatively higher value of mean squared error. It reflects that the algorithm has stuck in local optimum. We also observe that when the cooling rate is too low i.e. 0.15 or even too high i.e. 0.95, the algorithm takes more number of iterations too converge. It reflects that niether very high nor very low value of cooling rate is good. In addition, training time graph suggests that generally, as the cooling rate increases the training time also increases. Looking at the results we chose 0.70 as the optimal cooling rate, since in this case the convergence is fast (in terms of both iterations and training time) as well as the error is consistently lower than all other cooling rates.

Next we study effect of starting temperature (T). Intuitively, the effect of starting temperature is more important for first few iterations and then its effect becoming substantially less pronounced as more iterations occur; a result of the cooling rate. We fix our cooling rate at 0.70 and study the effect of initial temperature on convergence.





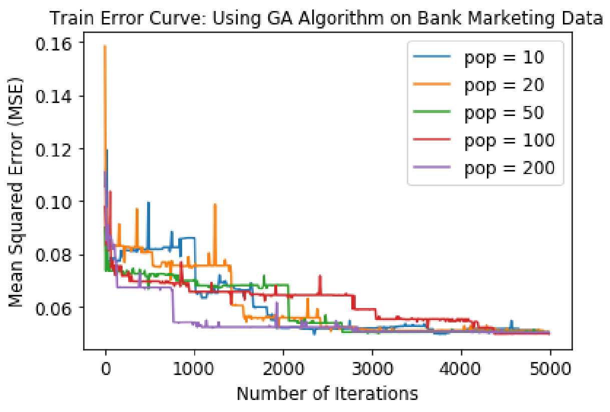
For the given cooling rate of 0.7, we find that our algorithm does converge to same optimal mean squared error, no matter what the initial temperature is. In addition, it was observed that when the initial temperature is very low, for example 1e6, the algorithm took most number of iterations to converge. Even at very high initial temperatures the number of iterations required to converge is more. Initial temperature of 1e10, niether too high nor too low, was found to converge faster and performed consistently better than other initial temperatures. In terms of training time, initial temperature seems to have negligible effect.

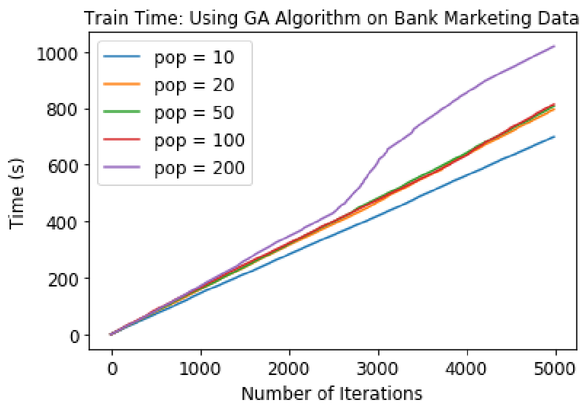
To find better solution than we already have, we can study effect of combination of Temperature and Cooling Rate on MSE rather than studying effect of one keeing other constant. It might be quite possible that certain other combination of T & CE is optimal which could not be explored via discussed methodology above.

Genetic Algorithm

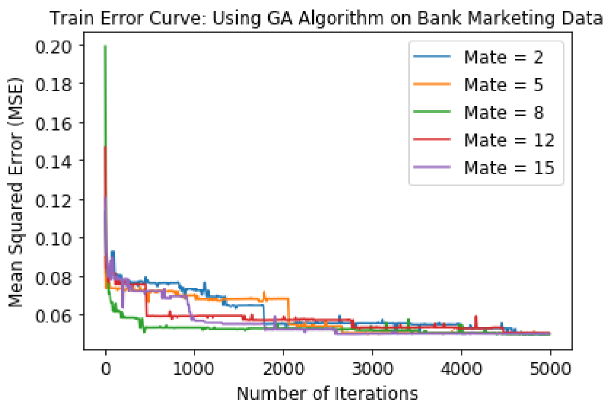
A genetic algorithm (GA) is a method for solving randomized optimization problems based on a natural selection process that mimics biological evolution. The algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm randomly selects individuals from the current population and uses them as parents to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution.

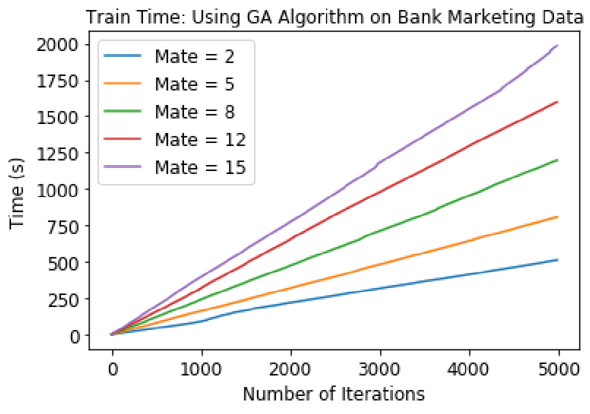
We have three hyperparameters to tune in Genetic Algorithm. First one is, Population size (pop) i.e. number of chromosomes in population (in one generation). Second one is, Mate i.e. number of cross-over to be performed. Last one is, Mutate i.e. number of chromosomes to be mutated. We will study effect of each of the parameters by keeping other parameters fixed. First, we study effect of Population Size (pop). For purpose of this study, we have fixed Mate = 5 and Mutate = 2.



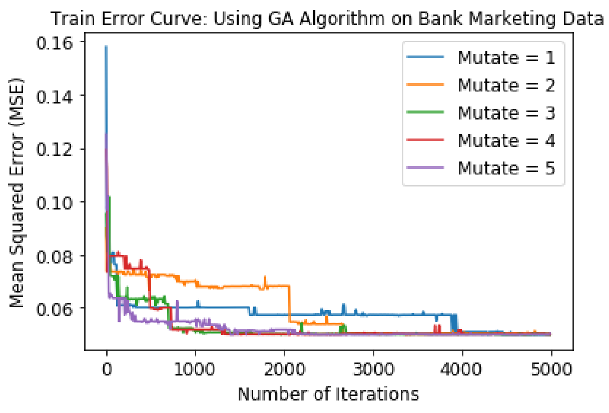


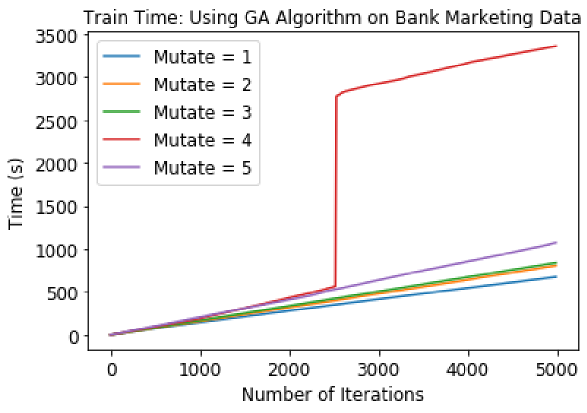
The above analysis shows that, no matter what population size we choose, all the algorithms converge to same optimal Mean Squared Error. However, number of iterations needed to converge vary quite significantly. In general, higher the population size, more iterations the algorithm required to converge. Also we observe that when population size is very high, i.e. 200, the algorithm coverges fast in the beginning and then slowes down very soon. It seems like the algorithm is caught up in a local maxima. The train time graph intuitively suggests that as the population increases, the training time also increases reflected by sudden jump in train time when population is 200. We choose pop size of 50 over others because it converges fastest, consistently performing better than others. Also, it takes reasonable time for convergence. Next, we look at hyperparameter Mate keeping population size (pop) = 50 and Mutate = 2.





We find that for all the values of Mate our algorithm converges to same optimal Mean Squared Error. For lowest values of Mate i.e. 2, algorithm took maximum iterations to converge. And the largest value of Mate i.e. 15 took least number of iterations to converge. However, training time with such a high number of Mate is very high. In general, training time has increased with increase in number of Mate. Therefore, we chose Mate = 5 because it was second best in number of iterations required to converge and also the time required to converge was relatively small. Next, we look at Mutate hyperparameter. To study this parameter, we fix our population size (pop) to 50 and Mate = 5, as found optimal in previous analyses.





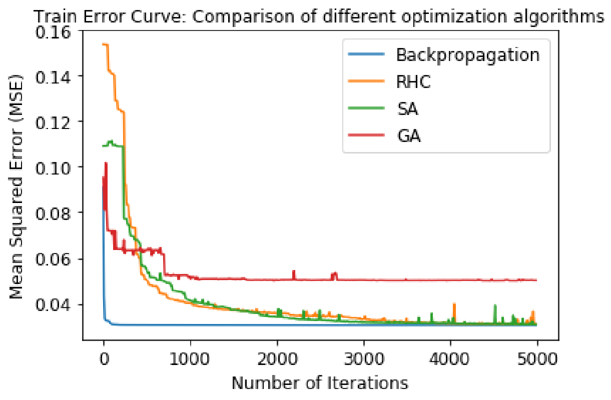
Above analysis shows that extreme values, either high or low, of mutate, takes more number of iterations to converge. We find that middle value of Mutate i.e. 3 took least number of iterations to converge and Mutate = 1 took maximum number of iterations to converge. Also, the training time increases as we increase value of Mutate. Therefore, we chose Mutate = 3 as our optimal value of mutate. We also observe an unusual increase in training time for mutate = 4. Probably, the algorithm got stuck. Hence, the overall optimal set of hyperparameters for GA algorithm is pop = 50; mate = 5; mutate = 3. However, we should vary combination of population size, mate and mutate parameters to identify if we can get a better solution rather than varying one and keeping others constant.

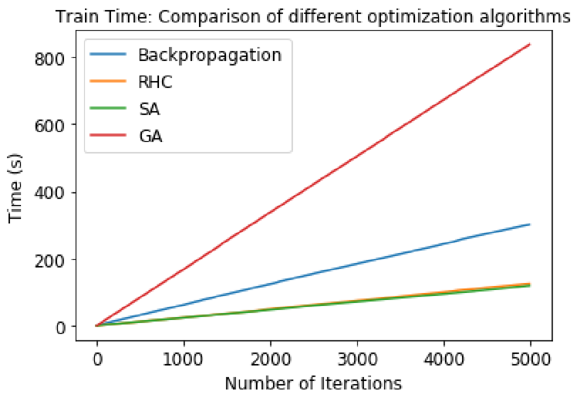
Comparison of all the 4 optimization algorithms

Using the optimal set of hyperparameters obtained for all the above discussed algorithms, we compare the performance of each of them on our Bank Marketing dataset. Here, is the optimal set of hyperparameters for each of the algorithm.

S. No.	Optimization Algorithm	Hyperparameters
1	Backpropagation	-
2	Randomized Hill Climbing (RHC)	*Number of Restarts* = 1
3	Simulated Annealing (SA)	*Cooling Rate* = 0.7; *Temperature* = 1e10
4	Genetic Algorithm (GA)	*Population Size* = 50; *Mate* = 5; *Mutate* = 3;

We compare error curve and train time of each of the algorithms.





Looking at above results, our initial Backpropagation method seems to be a clear winner in terms of number of iterations required to converge to optimal Mean Squared Error. It reaches the same level of accuracy as the other algorithms but much more rapidly and consistently. The simple reasoning is that backpropagation makes use of gradient descent, allowing it to determine with near-certainty the direction in which to head to find a very good optimum, allowing it to accelerate towards potential optima, instead of randomly sampling points or batch of points like the other algorithms.

Second thing that we may observe is that all the algorithms have converged to same Mean Squared Error value except Genetic Algorithm. We find that Genetic algorithm suffers from problem of pre-mature convergence i.e the population of optimization problem has converged too early, giving suboptimal results. It might be due to poor tuning of hyperparameters. We can try on increasing population size at cost of training time, improve mating and cross-over strategy. We can also do segmentation of individuals of similar fitness, also known as fitness sharing.

Thirdly, in terms of train time Genetic algorithm is found to be most computationally expensive. Whereas, other two randomized search algorithm i.e. Randomized Hill Climbing and Simulated Annealing take lowest amount of time. However, their convergence rate is slow, compared to backpropagation, as evident from error curve. Overall, among all the three randomized optimization algorithms we claim that simulated annealing has performed the best in terms of number of iterations taken to converge and train time.

Part 2: Three Interesting Optimization Problems

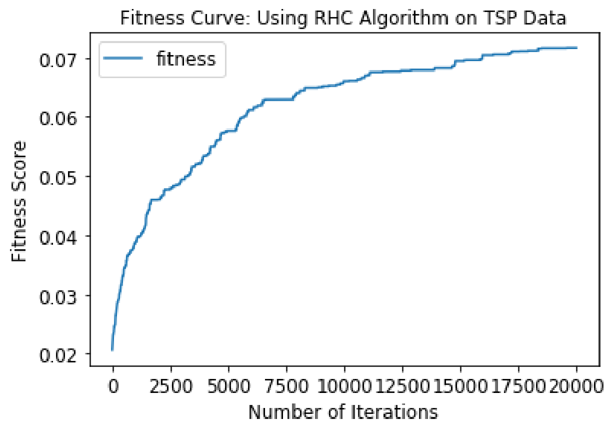
In this section, we will analyze the four randomized optimization algorithms against three interesting optimization problems. The fourth randomized optimization algorithm is Mutual Information Maximizing Input Cluster (MIMIC). We will be analyzing the performance of the algorithms on the traveling salesman problem, a simple flip flop bits problem, and finally the max K-coloring problem from graph theory. The first will highlight ABAGAIL's genetic algorithm, the second will highlight simulated annealing, and the third and final problem will highlight the MIMIC algorithm.

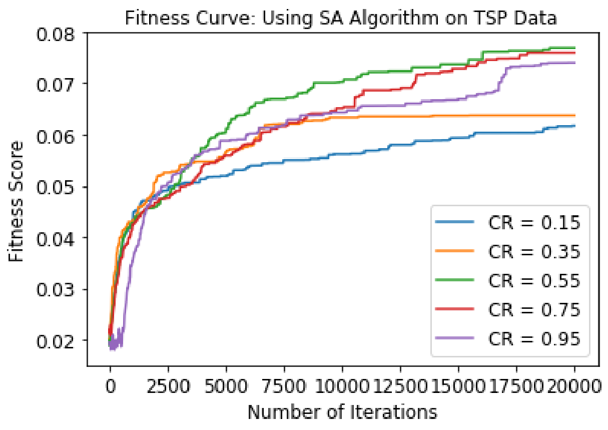
Traveling Salesman Problem *(Genetic Algorithm)*

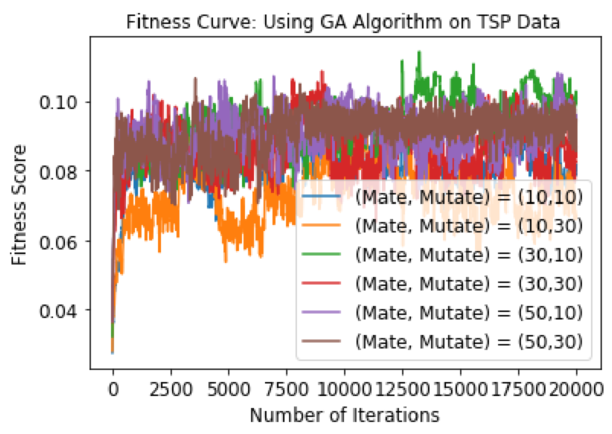
TSP is also one of the most classic NP hard optimization problem. In short, a salesman who needs to travel a few cities (n), represented by scattered points in a coordinate plane. The problem is to determine the least costly round-trip route for this salesman in terms of travel time, ticket price and distance. In other word, given a list of vertices and edges, the goal of the problem is to figure out the shortest possible route that visit each vertex once and return to the starting point.

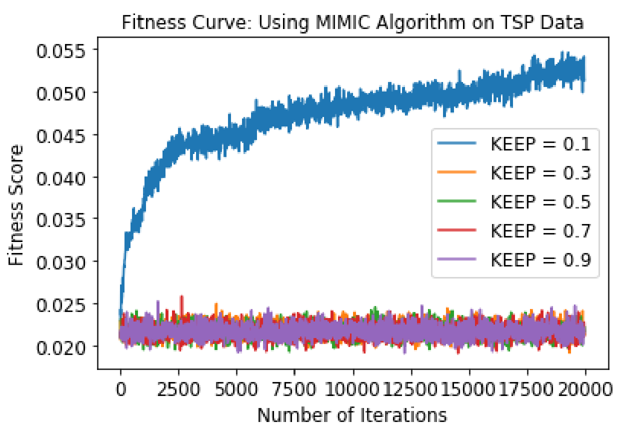
This problem is considered NP hard since for N number of cities, there are (n-1)! possible paths. Finding a universal function that address the general condition of this problem is impossible since small changes in location of the vertices will give completely different routes. Due to this nature of TSP, it is hard for greedy algorithms like SA and RHC to be able to accurately determine the best route in general.

For the purpose our analysis, we have set number of cities, N = 100 and then tried different hyperparameters to tune each of the randomized optimization algorithms. For Simulated Annealing, we fixed initial temperature to 1e10 and ran the algorithm for different values of Cooling Rates. For Genetic Algorithm, we fixed value of population to 100 and ran the algorithm for various combination of Mate and Mutate. For MIMIC, we fixed population size to 100 and number of samples to generate at 50 and ran the algorithm for various values for number of samples to keep. In the problem, we will try to optimize fitness score. The fitness score is evaluated by 1/distance traveled and thus higher fitness score means shorter travel distance.





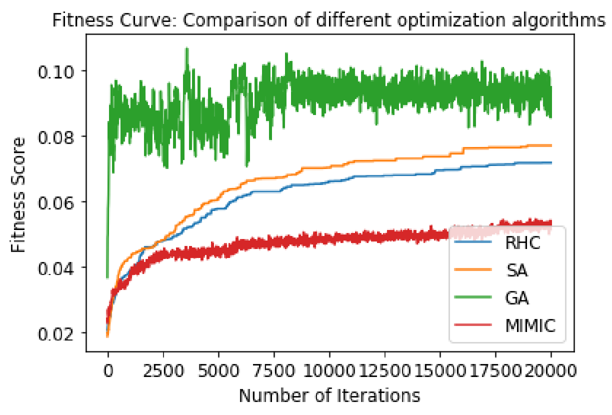


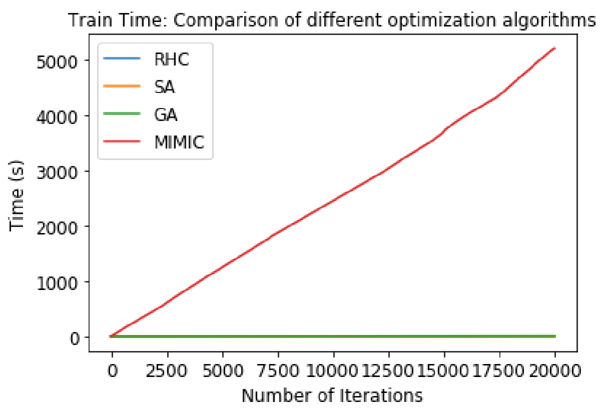


Based on above analysis, we chose our optimal set of hyperparameters for each algorithm as follows.

S. No.	Optimization Algorithm	Hyperparameters
1	Randomized Hill Climbing (RHC)	*Number of Restarts* = 1
2	Simulated Annealing (SA)	*Cooling Rate* = 0.55; *Temperature* = 1e10
3	Genetic Algorithm (GA)	*Population Size* = 100; *Mate* = 50; *Mutate* = 30;
4	MIMIC	*Population Size* = 100; *Generate* = 50; *Keep* = 0.1;

Using above set of optimal hyperparameters for each algorithm, we compare their convergence and training time.





The graph above clearly indicates, GA performed the best of the four algorithms, resulting in the best fitness overall. It also consistently reaches higher fitness with fewer iterations, while SA and RHC are still learning. This makes sense as TSP tends to lend well with slight mutations. In other words, finding a series of good paths (a population), and mutating them to create a better overall series of paths (a better population), and repeating, is a good way to solve the problem. Intuitively speaking, when a person solves problem, a series of routes is usually suggested, and combinations/changes to these routes to get better routes would yield a good result (mimicking GA), as opposed to just picking one route at random and just making a change once as you go (mimicking RHC and SA).

MIMIC does converge pretty quickly but it seems to have stuck in local optima. Also in terms of iterations MIMIC took substantially longer to run than any other optimization algorithm over shadowing the fact that even GA took relatively large amount of time to converge compared to SA or RHC.

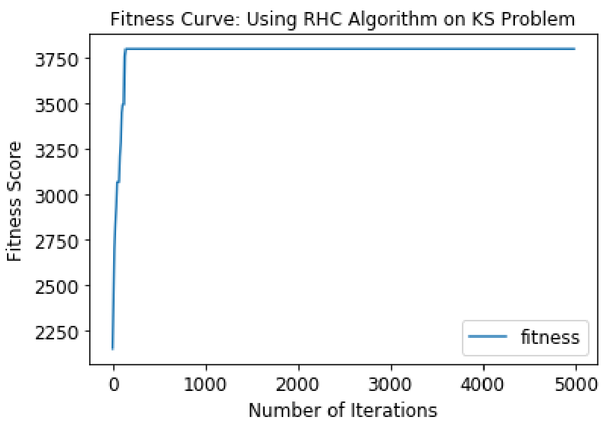
<u>Knapsack Problem (MIMIC)</u>

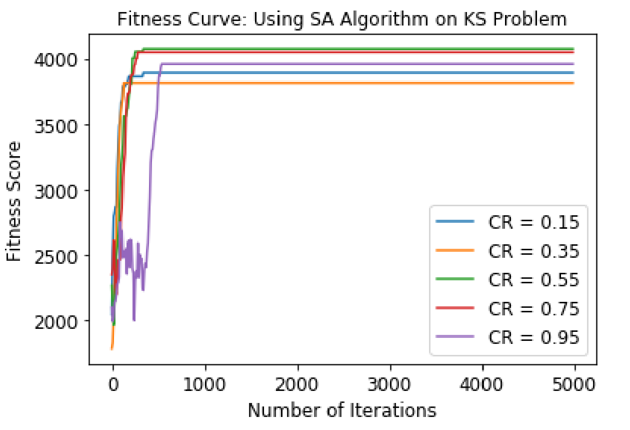
The knapsack problem is a constrained optimization problem: You are given a container with limited weight capacity and some items where each item has a weight and a value. We need to choose items to include in this container such that the weight limit is not exceeded and total value of the items included is maximized. Problem represents a lot of resource allocation where the resource is limited by financial constraints while the planning needs to achieve optimal efficacy. KP is a NP-hard problem and there is no known polynomial algorithm that describes the solution. For the purpose our analysis, we have set following parameters of the problem.

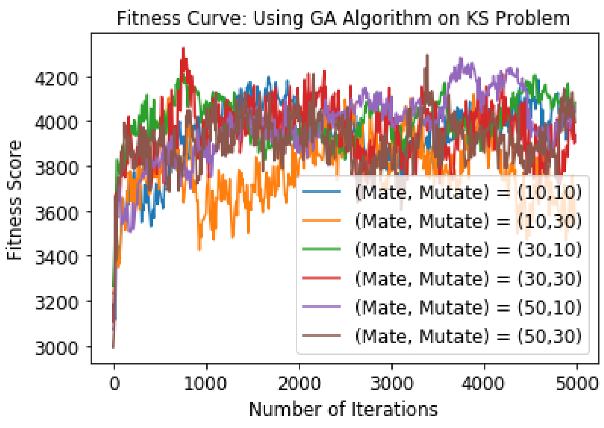
- Number of items, NUM_ITEMS = 40.
- Number of copies of each item, COPIES_EACH = 4.

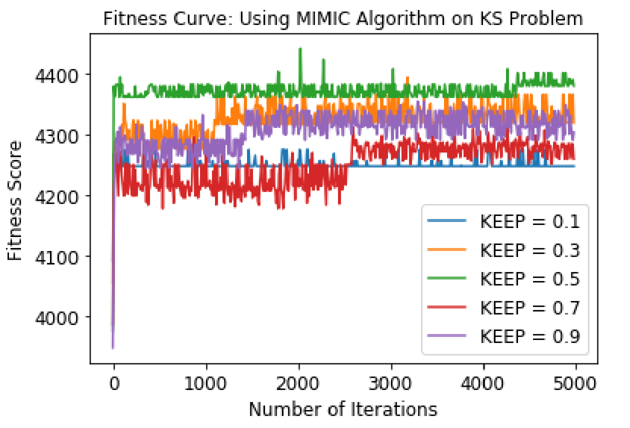
- Maximum weight of one item, MAX_WEIGHT = 50.
- Maximum volume of one item, MAX_VOLUME = 50

We tried different hyperparameters to tune each of the randomized optimization algorithms. For Simulated Annealing, we fixed initial temperature to 1e10 and ran the algorithm for different values of Cooling Rates. For Genetic Algorithm, we fixed value of population to 100 and ran the algorithm for various combination of Mate and Mutate. For MIMIC, we fixed population size to 100 and number of samples to generate at 50 and ran the algorithm for various values for number of samples to keep. In this problem, we will try to maximize fitness score. The fitness score is evaluated by maximum weight (or value) of items present in Knapsack and thus higher fitness score means more value and better results.





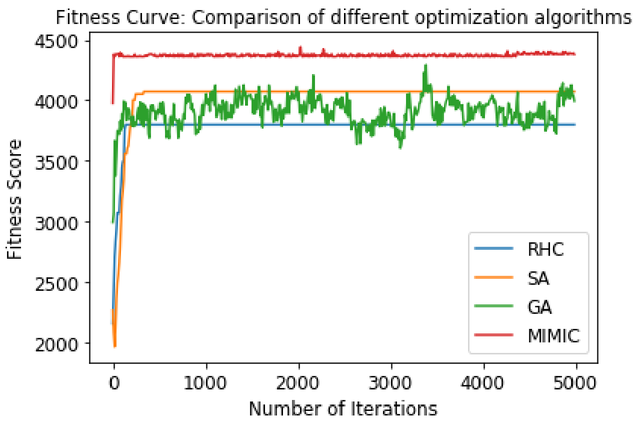


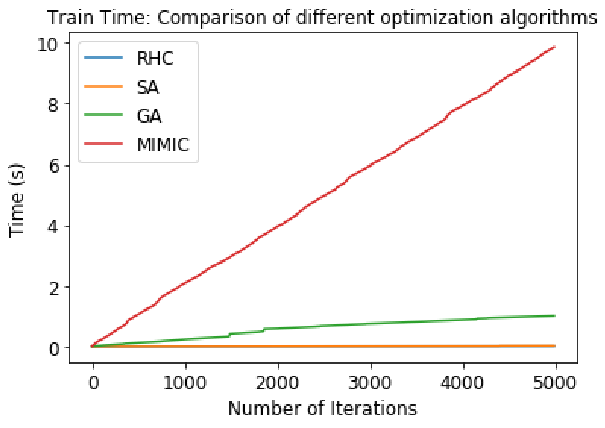


Based on above analysis, we chose our optimal set of hyperparameters for each algorithm as follows.

S. No.	Optimization Algorithm	Hyperparameters
1	Randomized Hill Climbing (RHC)	*Number of Restarts* = 1
2	Simulated Annealing (SA)	*Cooling Rate* = 0.55; *Temperature* = 1e10
3	Genetic Algorithm (GA)	*Population Size* = 100; *Mate* = 50; *Mutate* = 30;
4	MIMIC	*Population Size* = 100; *Generate* = 50; *Keep* = 0.5;

Using above set of optimal hyperparameters for each algorithm, we compare their convergence and training time.





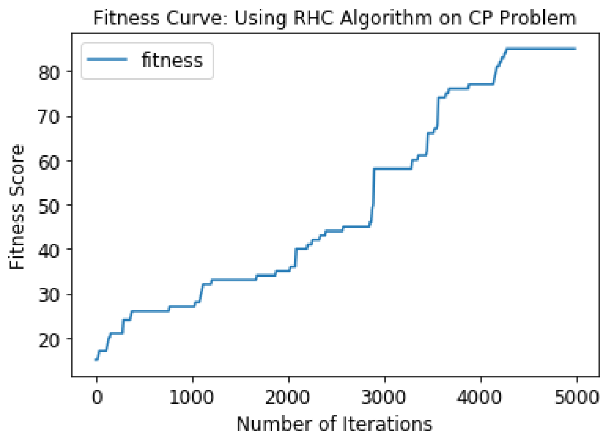
The graph above clearly indicates, MIMIC performed the best of the four algorithms, resulting in the best fitness overall. Not only it converges to a better fitness score but also it takes least number of iterations to converge to an optimal fitness score value. This may be because the MIMIC algorithm retains information about the problem space and is able to iteratively estimate better distributions based on this knowledge. Due to the advantage of MIMIC in extracting as much information as possible from each iteration, MIMIC has very high efficiency even within few iterations. The fitness score from MIMIC is being quite stable as the number of iterations increases. Thus, if cost evaluation function is very time consuming, MIMIC will be the best candidate since it takes much less iteration to converge to a solution. However, in terms of training time MIMIC takes longest time among all the algorithms.

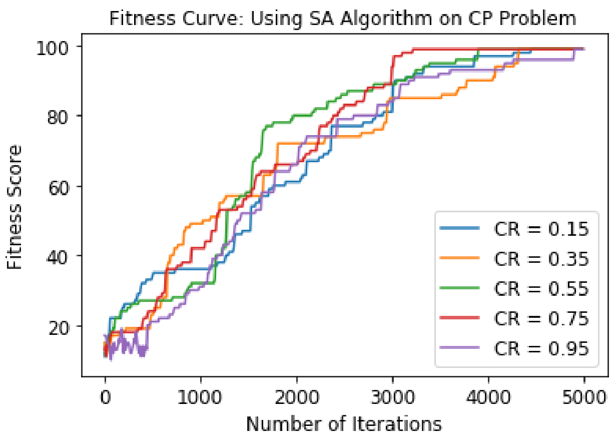
We also observe that SA and RHC converge much earlier but it converges to suboptimal score. This is because a slight change in item picked could nullify all the gains made. Whereas, GA didn't even converge in 5,000 iterations. It might be because the problem itself does not lend itself well to mutation and creating multiple populations is not that advantageous. We can either increase number of iterations or increas population size to see if the Genetic Algorithm converges to an optimal vale of fitness score.

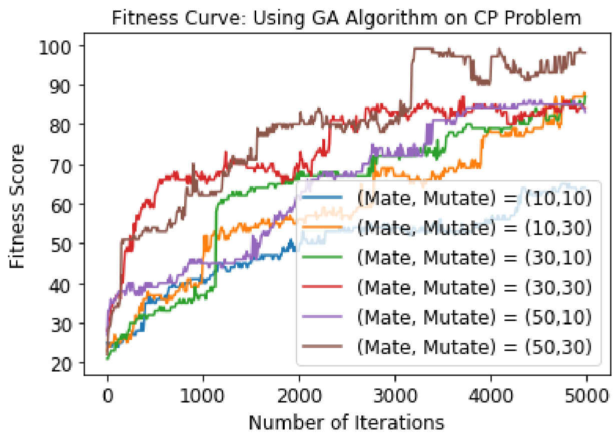
<u>Contiuous Peaks Problem (Simulated Annealing)</u>

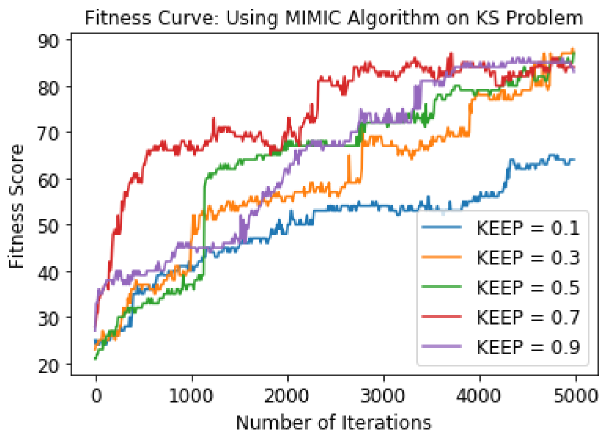
The Continuous Peaks Problem assigns higher fitness values for sequences of unbroken 0s and 1s in the bit string and assigns a bonus when the max sequence of both 0s and 1s is greater than a preassigned value of T. The global optimum is 2N - T - 1 but for higher values of T this can be a difficult optimum to find and the algorithms tend to settle in local optima. We have fixed number of integers, N = 100 and the basin of attraction, T = 49 for analysing different optimization algorithms.

We tried different hyperparameters to tune each of the randomized optimization algorithms. For Simulated Annealing, initial temperature was fixed to 1e10 and algorithm was run for different values of Cooling Rates. For Genetic Algorithm, we fixed value of population to 100 and ran the algorithm for various combination of Mate and Mutate. For MIMIC, we fixed population size to 100 and number of samples to generate at 50 and ran the algorithm for various values for number of samples to keep. In this problem, we will try to maximize fitness score. The fitness score is evaluated based on our discussion in above paragraph and higher fitness score represents better results.





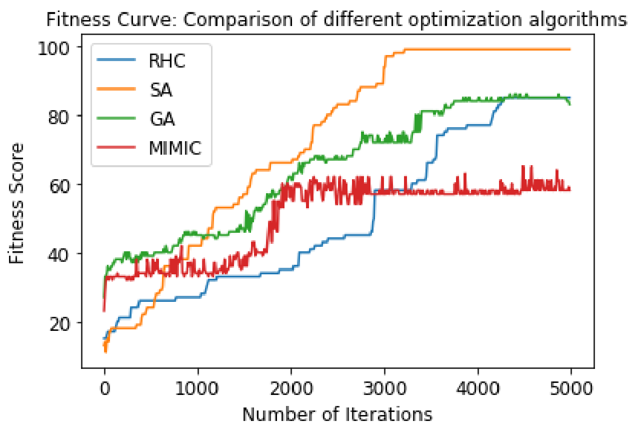


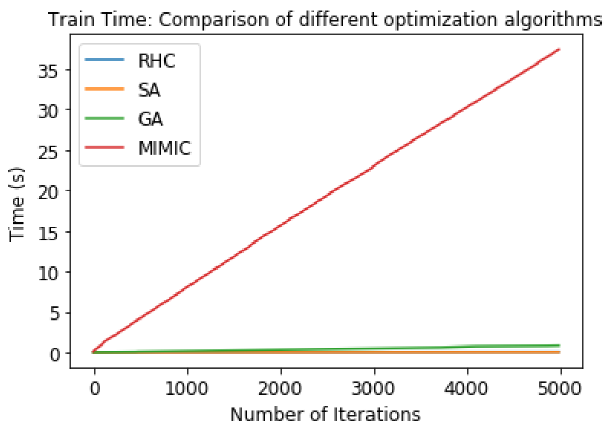


Based on above analysis, we chose our optimal set of hyperparameters for each algorithm as follows.

S. No.	Optimization Algorithm	Hyperparameters
1	Randomized Hill Climbing (RHC)	*Number of Restarts* = 1
2	Simulated Annealing (SA)	*Cooling Rate* = 0.75; *Temperature* = 1e10
3	Genetic Algorithm (GA)	*Population Size* = 100; *Mate* = 50; *Mutate* = 10;
4	MIMIC	*Population Size* = 100; *Generate* = 50; *Keep* = 0.9;

Using above set of optimal hyperparameters for each algorithm, we compare their convergence and training time.





The graph above indicates that Simulated Annealing achieves highest fitness score compared to any other randomized optimization algorithm. This makes intuitive sense because Simulated Annealing is made to climb hills, which is what finding the highest Y value on an XY graph is. Also, at given high value of T, such as 49, used in our problem there could be many local optimum in the basin of attraction. Given simulated annealing is better in avoiding local optima, it's performance is expected to be better than algorithms like Randomized Hill Climbing.

Continuous Peaks, on the other hand, does not work well with Genetic Algorithms, as the problem itself does not lend itself well to mutation and creating multiple populations is not that advantageous. In other words, combining two high values (i.e. mutation/crossover) would not necessarily lead to a good new point. For example, if two points near two local optima are crossed over, the child point could be in a valley, and thus a low value is introduced into the population).

Similarly, time for the results shows that Random Hill Climbing is quickest, that is because the algorithm is geared towards climbing hills (which is what Continuous Peaks is doing), but is much quicker because it may get stuck with a local optimum and quit early. Simulated Annealing is quick, but the exploration aspect of SA takes a bit longer. The outlier is MIMIC, which is extremely slow. MIMIC works well when the evaluation function takes a long time, but the number of iterations is not large. However, Continuous peaks requires lots of iterations, and MIMIC's construction of trees slows it down considerably.

Conclusion

It is clear that certain algorithms work well for certain problems. All algorithms have strengths and weaknesses as witnessed with the 3 problems above. Randomized Hill Climbing works well for problems with incremental improvements and lots of iterations, but do not have local optimum, and there is one optimum (e.g. one maximum value like in Continuous Peaks). Simulated Annealing takes slightly longer, but solves the local optima problem. Genetic Algorithms work well when mutation is in play with an entire population (e.g. a graph), and MIMIC works well when state is a consideration (e.g. the neighboring nodes' color).