Brad Ware
Randomized Optimization
CS 4641 Project 2

Neural Network Wine Quality Problem

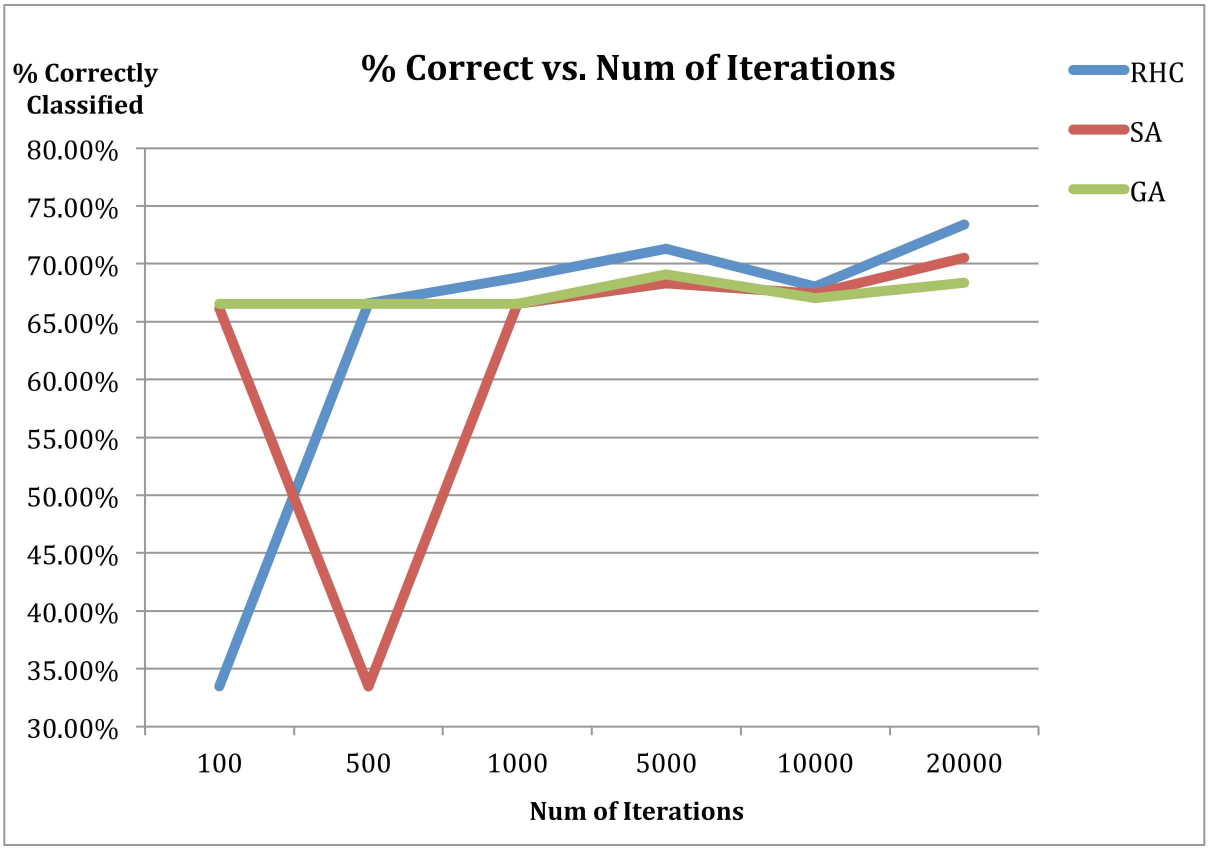
In project 1, one of my classification problems was to identify the quality of an instance of *white* wine. Some of the attributes included pH, alcohol content, and levels of acidity. It used backpropagation and gradient descent to determine the best weights for the neural network. The task this time was to use optimization algorithms that included Randomized Hill Climbing, Simulated Annealing, and Genetic Algorithms to find optimal weights for the neural network, replacing backpropagation. I used the ABAGAIL library to implement a neural network with my white-wine dataset varying the amount of iterations for all three algorithms. The results will be analyzed further in the next section.

Wine Quality - Further Note

All three of these algorithms will vary in performance based on the assumptions and tactics they take to find the optimal value. Randomized Hill Climbing (RHC) will always find the optimal value as long as it doesn't get stuck in a local maximum. This can be avoided with random restarts but there is still a threat of never getting out of a local optimum value. I therefore predict that RHC will perform more stable when there are more iterations, as it will have more time to restart in different areas of the sample space. Simulated Annealing (SA) counteracts local optima by accepting lesser values then the current based on the evolving temperature (T) equation. However, if there are less iterations then T will still be "hot" and thus more likely to accept values that are below the current. This is scary because SA could end up with very suboptimal weights, which will affect the neural networks classification capability. So similar to RHC, I also think that SA will act more consistently as the training iterations increases. Lastly, Genetic Algorithm (GA) has a very different approach than the first couple, and relies heavily on the accuracy and relevancy of the crossover and fitness function. Thus, because the crossover function is combining two data instances to make a child, the attributes of each instance may not be related at all. Because two instances of wine are completely independent of each other, they may not give optimal values when combined in the crossover function for reproduction. I don't think that the number of iterations will have a drastic effect on the ability for GA to find optimal weights.

Wine Quality - Correctness Results

So now we can analyze the results. This first figure is a graph showing the percentage of accurate classifications vs. the number of iterations. The graph breaks it down for each algorithm: Genetic Algorithm, Randomized Hill Climbing, and Simulated Annealing.

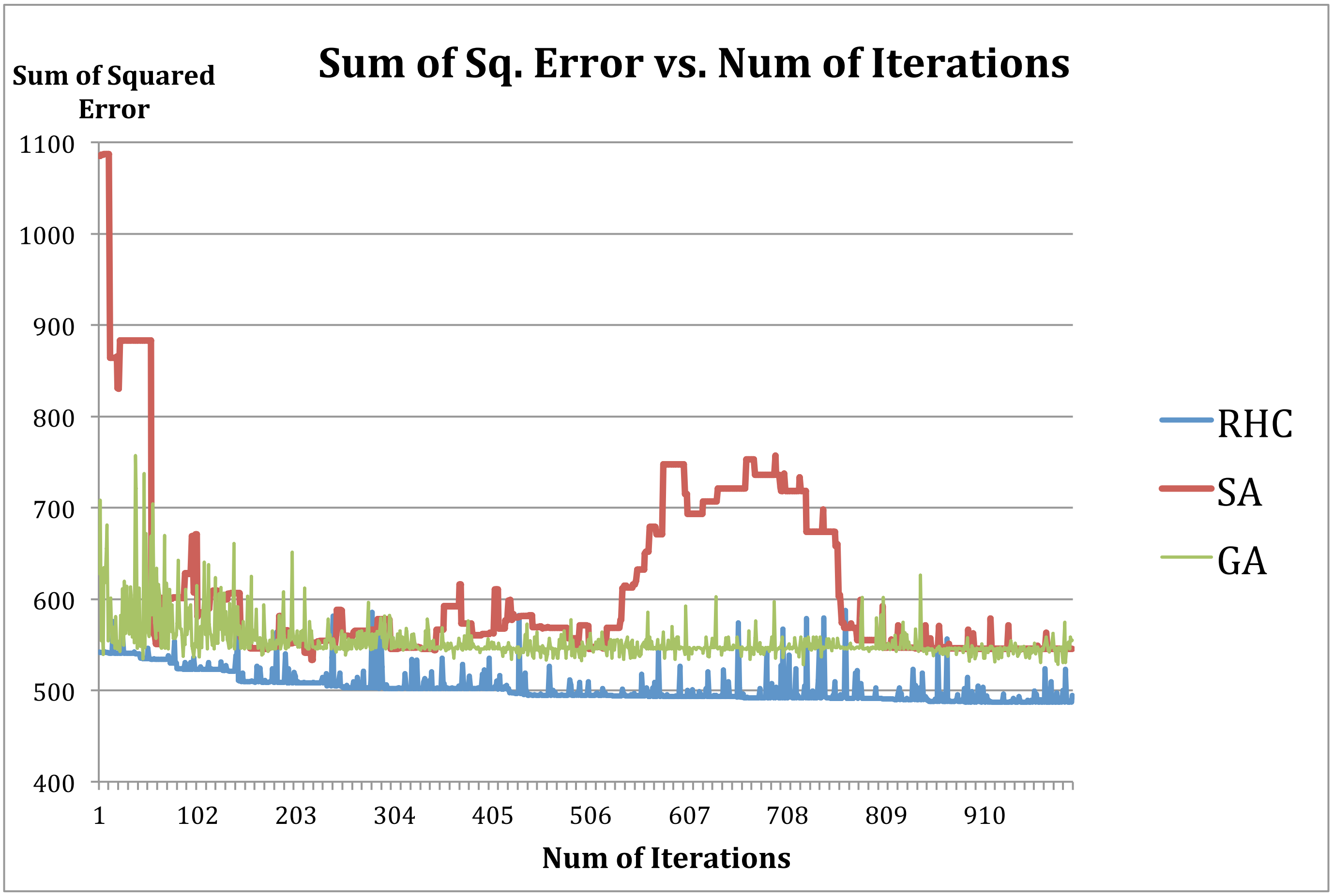


As I thought, RHC and SA stabilized in their classifications as the number of iterations increased. The algorithms traded places until 1000 iterations were reached, then they stabilized to run around the same. My inherent guess that SA ran so poorly at 500 iterations (33%) was due to the fact that it accepted sub-optimal values that were significantly lower because T was still very hot. SA spent more time hopping around rather than "climbing the hill "to the max. However, this stopped occurring when the number of iterations increased to 1000 and beyond (see table below).

Training Iterations	RHC	SA	GA	RHC Seconds	SA Seconds	GA Seconds
100	33.48%	66.13%	66.52%	1.593	1.327	26.764
500	66.58%	33.48%	66.52%	6.451	5.787	128.839
1000	68.80%	66.52%	66.52%	12.339	12.474	258.309
5000	71.29%	68.33%	69.07%	58.48	60.133	1301.419
10000	68.05%	67.44%	67.03%	117.997	130.929	2496.861
20000	**73.42%**	70.52%	68.35%	229.493	221.682	5027.429

**yellow highlight is min, green is max, blue is for emphasis

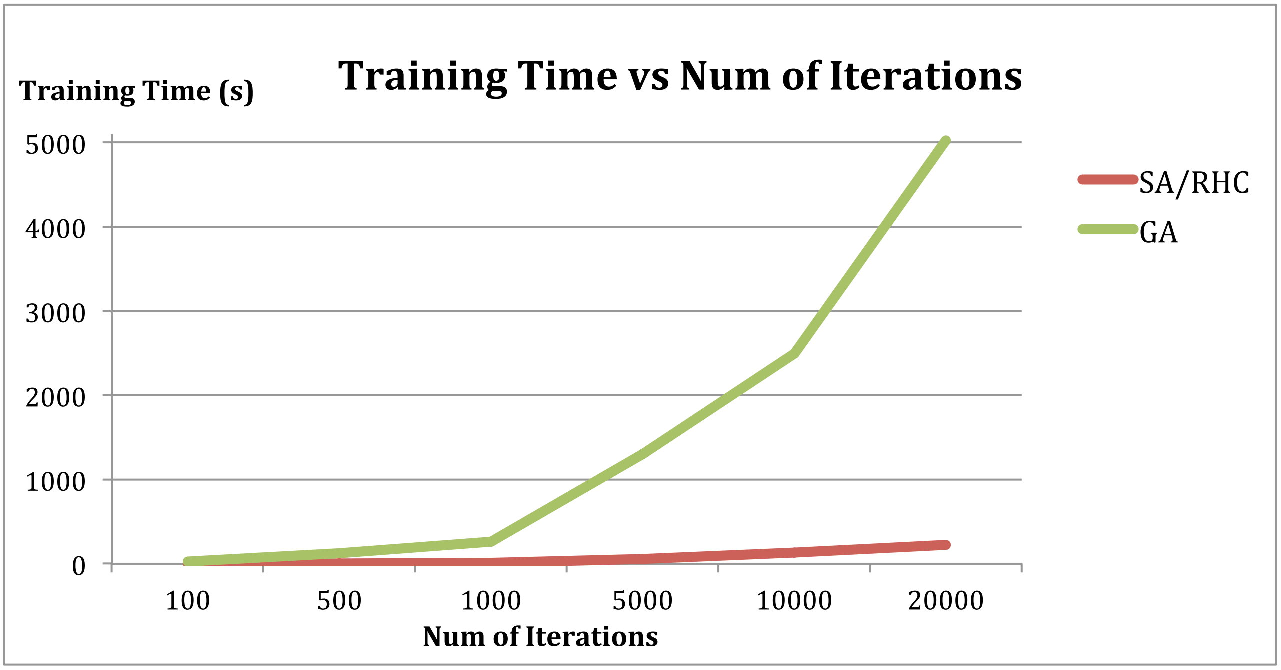
Also as predicted GA was not influenced really at all by the number of iterations. This seems logical as the crossover function just produces new children through reproduction and doesn't depend on acceptance criteria. Overall RHC performed the most optimal when used as weights for the neural network. This is surprising to me as I thought SA would find the ultimate max, but it seems that RHC never became stuck in a local maxima. It could be that if few local maxima existed over the sample space then SA's approach to accepting lesser values actually inhibited it from finding the true max.



I also want to analyze the difference in sum of squared error. In this graph each algorithm's sum of squared error decreases over time. As expected, RHC sum of squared error was the most consistent, slowly decreasing over time. This aligns with the nature of the algorithm, never taking risks and jumping to values that are only greater than the current. What's interesting is how high SA error was during the low number of iterations, as the other two were more consistent. This is definitely due to the fact that T was hot and SA was accepting lower values. As predicted, GA was not really affected at all by the number of iterations, even though the randomness of the crossover function caused a few outlying errors. At around 750 iterations SA and GA sum of squared error became almost equivalent, and it remained that way through 1,000 iterations.

Wine Quality - Time Results

In this graph we analyze the running time of each algorithm as the number of iterations increases. RHC and SA were combined due to the fact that there running times were so similar. As you can easily tell from this graph, the GA is much less efficient than RHC and SA. This honestly is to be expected, as the fitness and crossover function can be computationally very expensive. It's interesting to see just at 1000 iterations that RHC and SA are significantly more efficient, and the gap only increases. When looking at this further it seems that this could possibly be attributed to the fitness function, which is user defined. If there is minimal knowledge about the problem domain then the inductive bias could impact the fitness function in a negative way.



The crossover function could also be suboptimal in mating, by producing an offspring that is less than the parents. When GA has almost no information about the dataset it becomes hard for the algorithm to perform optimally.

Wine Quality - Conclusion

RHC was the most optimal algorithm for my white wine dataset in efficiency and accuracy. I have a couple of theories for this. One could be that it simply got lucky and never chose to originally start or restart in a space that leads to a local max. In this case it could only be a matter of time of re-running the data that SA might match or even beat RHC. However, it may just be the case that there was few local maxima in the data, resulting in a small chance that RHC would become stuck on one. If that is the scenario then it negatively impacted SA to accept sub-optimal values, as it wouldn't help find a true max in the sample space. GA definitely performed the worst in terms of efficiency, and even slightly in accuracy. My theory for that was explained above, but as a recap I think it is because the fitness and crossover functions don't account for dependences among different data instances. This will effect reproduction and produce offspring that are less optimal then their parents, preventing it from finding the optimal weights for the neural network. When dealing with classifying the quality of wine, using a simple RHC optimization algorithm not only finds the best weights, but also runs extremely efficient.

Optimization Problems

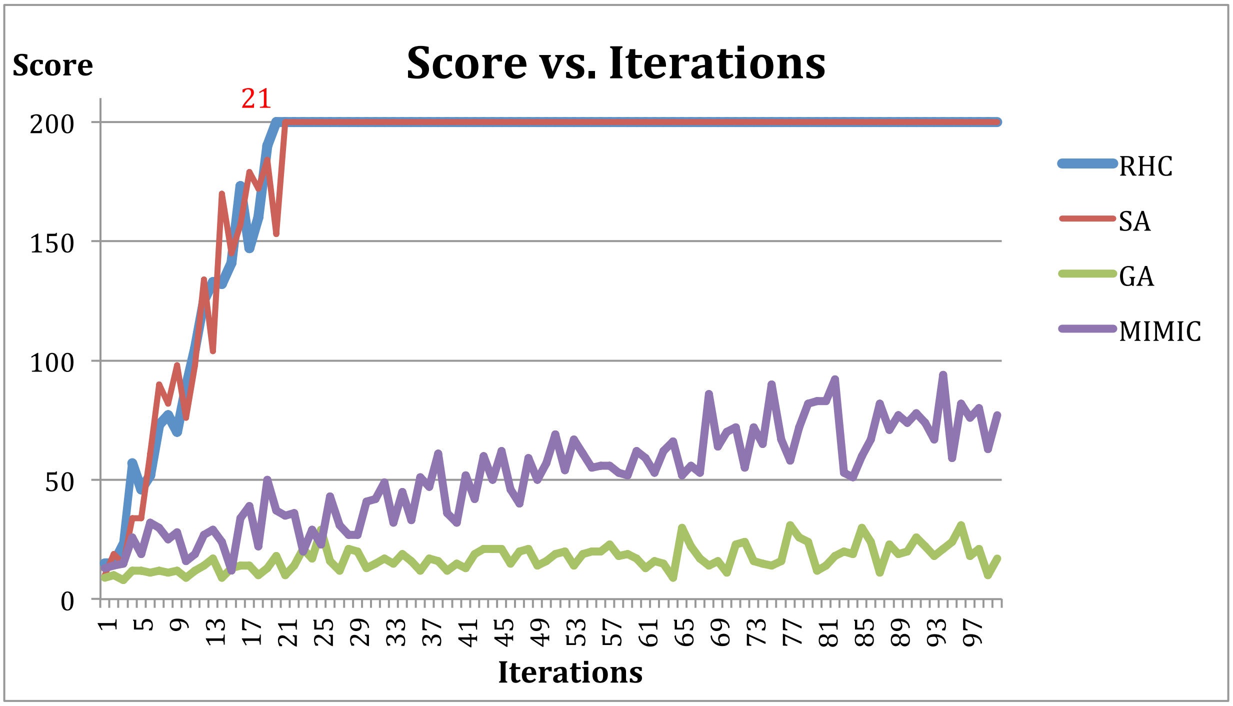
For the second part of the assignment, we were asked to choose three optimization problems to maximize the fitness function. The three problems I chose to analyze are Four Peaks, Traveling Salesman, and the Knapsack problem. Each of the applications are over discrete value parameter spaces, which gives optimization problems an advantage. If these problems were in continuous domains then it would be simple enough to take the derivative of the fitness function we are trying to maximize, just as the gradient descent algorithm would do. However, these are discrete problems and therefore each was picked to highlight Simulated Annealing (SA), Genetic Algorithm (GA) and MIMIC. I will now dive further into each of the problems and highlight all of the algorithms advantage per problem.

Four Peaks – Introduction

Four Peaks is a function with two global maxima and two local maxima. The global maxes are based off of a T value of leading 1's and trailing 0's. The amount required for a maximum changes as T is manipulated. Based on what I know about the problem, I predict that RHC and SA will outperform GA and MIMIC.

Four Peaks – Results

As you can see, SA was barely outperformed by RHC on average, and this was probably due to the fact that SA took more chances because of the nature of the algorithm. Because it accepts sub-optimal points sometimes, it makes sense that it will have a risk of running poorly, where as RHC only will increase it's current situation.

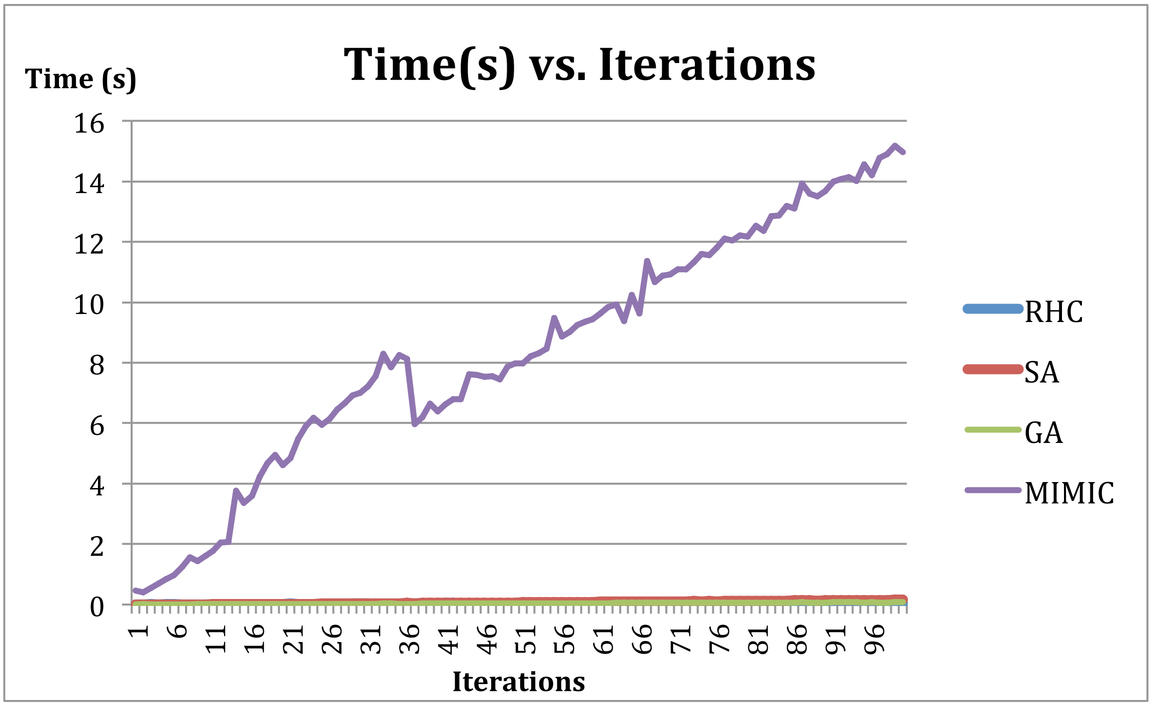


*** For each iteration, multiply by 2,000 to get the number of iterations that the Fixed Iteration Trainer used.

The interesting aspect of this graph is how quickly SA and RHC found the max solution, and they never ventured away after that. It only took them 21 iterations to reach 200 (optimal score) where the Fixed Iteration Trainer was given 40,000 iterations. Even though SA ran on 200,000 iterations in the last instance, it only needed 40,000 to find the max in this Four Peaks problem. MIMIC increased it's performance when more iterations were added for training, however GA did not depend at all on the amount of training iterations. GA just simply performed awful for the Four Peaks problem. Below are more detailed results about the rest of the report.

Algorithm	Max	Average	Min	T Max (s)	T Avg (s)	T Min (s)
RHC	200	180.23	15	0.074	0.032	0.004
SA	200	180.15	10	0.163	0.074	0.001
GA	31	17.07	8	0.079	0.03959	0.005
MIMIC	94	51.51	12	15.174	8.37	0.404

Now we can finally assess the efficiency of each of the algorithms.



*** For each iteration, multiply by 2,000 to get the number of iterations that the Fixed Iteration Trainer used.

A clear point is that SA outperformed MIMIC and GA easily in accuracy, but also is a much better choice than MIMIC in terms of efficiency. MIMIC proves again to be the most inefficient algorithm, whereas the other three have minimalistic differences in runtime.

Four Peaks – Conclusion

It's clear that out of SA, GA, and MIMIC, SA is by far the optimal choice for the Four Peaks problem. SA was able to avoid both of the local maxima's in the Four Peak's problem as well as find both of the true maxes. Even though RHC performed just as well as SA, it could have easily gotten stuck in either of the local maximum if the T variable was increased a substantial amount. Four Peaks is a great example to show the strengths that SA provides not only in efficiency, but also accuracy.

Traveling Salesman Problem – Introduction

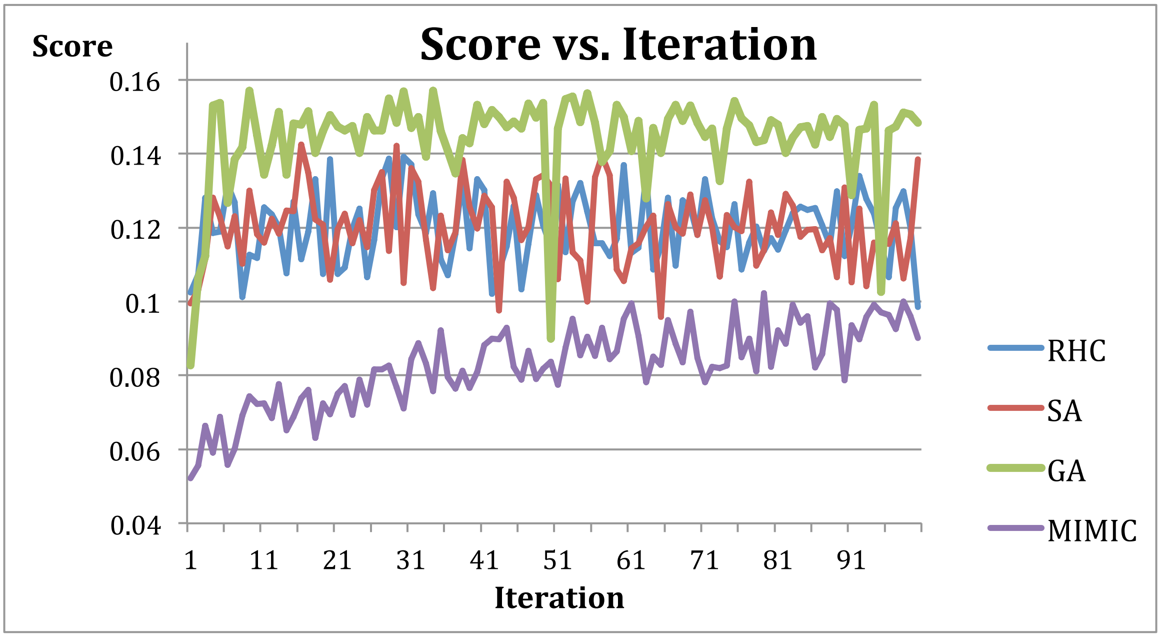
Traveling Salesman is a classic NP-hard combinatorial problem where you are given a list of cities and distances between each and it tries to visits all the cities traveling the minimal total distance. In this implementation, ABAGAIL takes 1/distance as the output, so the algorithm with the highest score had the lowest distance (converted from minimum to maximum optimization problem). My prediction is that GA will have the highest performance but may be more costly in time compared to RHC and SA. Using a crossover function of two cities and maximizing the child of that relationship is crucial in determining the optimal path, specifically what order the cities should be visited. But let's dive deeper into the results.

Traveling Salesman Problem – Results

Here is a table summary of the results from running Traveling Salesman on RHC, SA, GA, and MIMIC.

Algorithm	Max	Average	Min	T Max (s)	T Avg (s)	T Min (s)
RHC	0.139	0.12	0.099	0.085	0.04	0.003
SA	0.142	0.12	0.10	0.44	0.14	0.002
GA	0.157	0.144	0.082	0.40	0.21	0.03
MIMIC	0.102	0.083	0.052	16.76	8.04	0.27

This was extremely close, as the range for all of the algorithms differed by six tenths of a point. GA was able to perform above MIMIC easily, and also be significantly better than SA and RHC. In terms of computational performance, GA was able to perform efficiently. I have not included a visual to exclude redundancy, but by the table it is easy to see that MIMIC by far performed the worse again, while the other three were similar. RHC did extraordinarily well surprisingly, and I am also surprised that GA did better than SA. It seems that implementing the crossover function with two cities was an efficient way to reproduce. To analyze the score further, I have included a graph below detailing score vs. the number of iterations.



*** For each iteration, multiply by 2,000 to get the number of iterations that the Fixed Iteration Trainer used.

What's interesting here is that GA took some pretty big hits in score with reproduction (at iterations 1, 50, 65, and 95). It's amazing how at iteration 50 GA put up a score of 0.088, but then it jumped up so quickly to 0.156 at iteration 55. The crossover function with reproduction is an amazing tool for finding substantial success in few iterations.

Traveling Salesman– Conclusion

For the Traveling Salesman problem, GA simply outperformed the rest, and even ran the most efficient (except for RHC). As I predicted, the crossover functionality with two cities was able to minimize the distance quickly and more combinations of paths than RHC, SA, and MIMIC. The GA is a powerful tool that allows our function to creatively try visiting different orders of each city.

Knapsack Problem – Introduction

The knapsack problem is a historical optimization problem where you are given a set of items that has a value and mass associated, and the goal is to determine the number of each item to add to a collection. Overall, the total weights of every item has to be less than or equal to a given limit, and obviously you are trying to maximize the weights without exceeding the limit. This is a classic problem of determining the cost associated with each item and how valuable it is to your knapsack. After reading online, it seems historically that the GA has proven to be superior for the knapsack problem. This aligns logically as RHC will only jump to knapsacks that are greater than the currently found one, which is an easy way to become stuck on a local max if suboptimal items are chosen to start. SA will not be ideal either as it will accept knapsacks that are of lesser value. This becomes an issue with

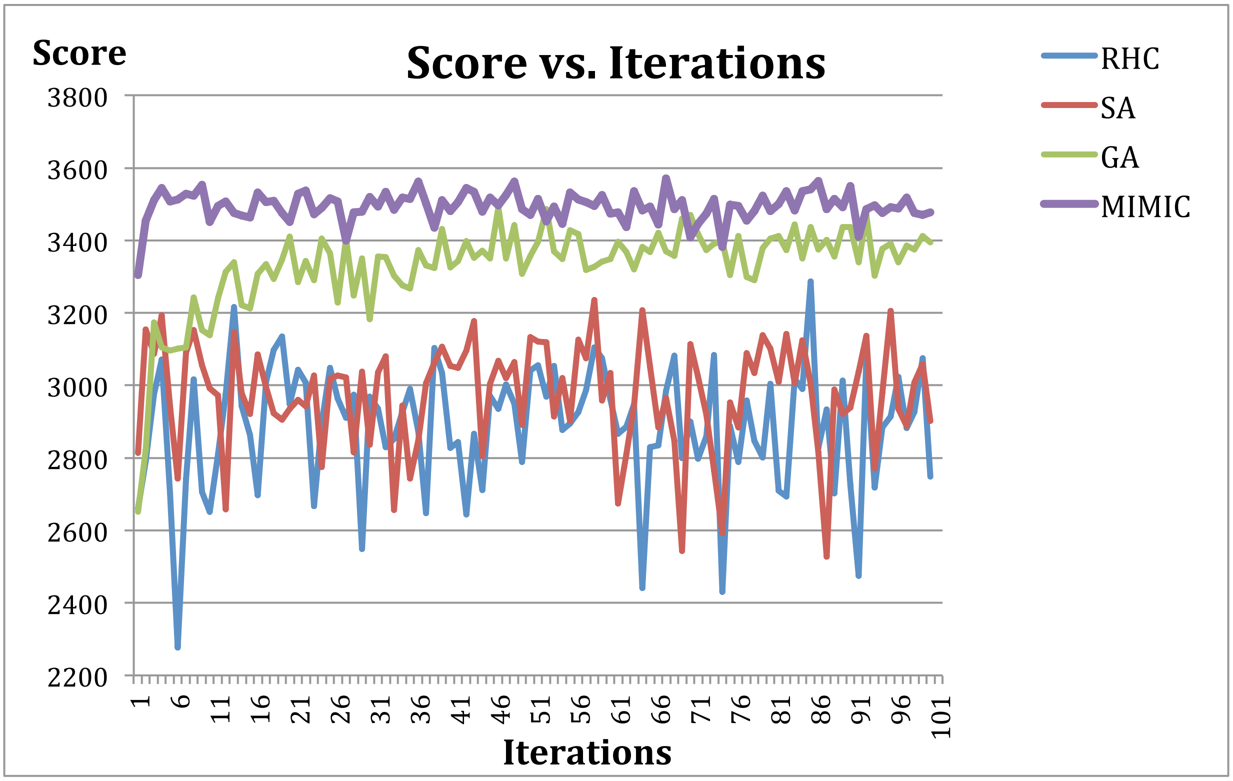
Dynamic Programming problems (like knapsack) where solutions are iteratively built on one another and rely on maximizing each step in the process. Therefore I predict that GA will outperform the rest of the algorithms given that it takes a different approach with reproduction.

Knapsack Problem – Results

Here are the results for the knapsack problem. Each algorithm was run through 100

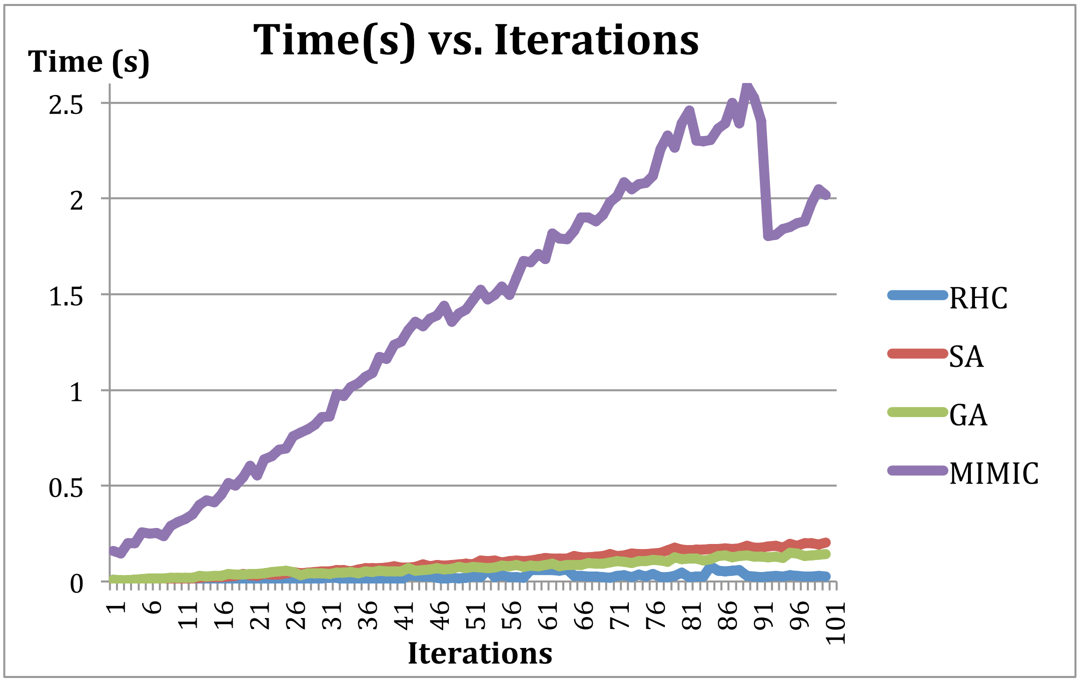
Algorithm	Max	Average	Min	T Max (s)	T Avg (s)	T Min (s)
RHC	3286	2886	2277	0.079	0.023	0.003
SA	3235	2975	2527	0.202	0.10	0.002
GA	3491	3330	2651	0.151	0.074	0.007
MIMIC	3570	3494	3304	2.59	1.376	0.15

iterations and then the max, min, and averages were calculated. Our knapsack problem used 40 items where there were 4 copies of each item. The max weight and volume of each item is 50. The max volume of the knapsack was 3200. The goal was to maximize the weight in the knapsack not exceeding the volume limit. However the surprise of these results was what MIMIC. MIMIC on average scored higher than GA by more than 150 and outscore RHC and SA on average by more than 500.



*** For each iteration, multiply by 2,000 to get the number of iterations that the Fixed Iteration Trainer used.

It’s very clear that while GA, SA, and RHC are relatively equal in terms of efficiency, MIMIC was definitely worse. Not only was it’s average running time almost 20 times worse than the other algorithms, it grew at a much greater rate with each iteration, while the others did not.



Knapsack Problem – Conclusion

In wrapping up the knapsack problem, it is very clear that surprisingly MIMIC outperforms the rest of the algorithms. As predicted, GA is a better choice and more stable than RHC and SA. However, if having to choose an algorithm with limited resources in time, it seems that GA is still the best choice. MIMIC’s run-time is extremely suboptimal compared to the other algorithms while it performs only slightly better than GA.