Assignment 2: CS7641 - Machine Learning

Jeff Shi

October 13, 2016

1. Introduction

In this assignment, I explored several randomized optimization algorithms. First, I took the first three algorithms to discover good weights for a neural network for the Voice Gender problem I used in Assignment 1. Then, I experimented on four different optimization algorithms included in ABAGAIL[1], which is a library of interconnect Java packages that implement machine learning algorithms. The algorithms were: Randomized Hill Climbing (RHC), Simulated Annealing, Genetic Algorithm, and MIMIC (Mutual-Information-Maximizing-Input Clustering). Finally, I analyzed the four optimization algorithms by applying them to three different optimization problems.

2. Optimization Algorithms

2.1. Randomized Hill Climbing

Hill Climbing, is an iterative searching algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution through incremental changes. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found and an optimum is reached. Randomized Hill Climbing utilizes random restarts, in which the algorithm randomizes its original starting solution to mitigate the risk of finding only a local optimum.

2.2. Simulated Annealing

Similar to the concept of annealing in metallurgy, Simulated Annealing involves setting a very high "temperature" variable and then allowing that variable to gradually "cool". In each iteration, the algorithm takes the current state and a neighboring state and probabilistically decides on whether to move the system to the new state or stay in the current state. Then, at the end of each iteration, the temperature variable is decreased. The iterations are repeated until the system reaches an optimum solution, or until a set number of iterations have been complete.

2.3. Genetic Algorithm

As implied by their name, Genetic Algorithms solve optimization problems by continuously evolving a set of candidate solutions into better solutions. Through this evolution, non-adequate parts of the population are eliminated and the subset of the population that have the best "traits" are retained. The evolution process, which is an iterative process, starts from a set of randomly generated candidates. In each iteration, the algorithm evaluates the "fitness" of every candidate. The "fitness" is the metric that measures how well the optimization problem is being solved. The more "fit" candidates are selected from the population, and each candidate's "genes" are recombined to form a new generation. This process repeats until the algorithm terminates when either a set number of generations has been produced, or a satisfactory population fitness level (optimum solution) has been reached.

2.4. MIMIC

MIMIC finds optima by estimating probability densities. It is one of several estimation of distribution algorithms in existence. These estimation of distribution algorithms seek to search for the optimum by building and sampling probabilistic models of promising candidate solutions. The algorithm incrementally updates the probabilistic model until it reaches a model that generates only the global optima. MIMIC specifically uses a chain distribution to model interactions between variables[2].

---

[1] ABAGAIL can be found at: https://abagail.readthedocs.io

[2] https://econ.ubbcluj.ro/~rodica.lung/taco/literatura/EDA_review.pdf

3. Finding Weights for a Neural Network

3.1. Purpose

In this part of the assignment, I analyzed the performance and results of three optimization algorithms when used to find the optimal weights for a neural network for the Voice Gender data set from Assignment 1. The three optimization algorithms are: Randomized Hill Climbing, Simulated Annealing, and Genetic Algorithm.

3.2. The Voice Gender Data

The second classification problem has 20 voice attributes as inputs and labels each instance as a male or female voice. A copy of the data set can be found here: https://www.kaggle.com/primaryobjects/voicegender. I picked this data set because there's a potential to use this data to artificially create male or female voices for artificial intelligence, robotics, and in media such as video games.

Like in Assignment 1, the Voice Gender data set was split in several ways using Weka Explorer. First, I split the dataset 80/20, with the 20% being the final validation set. Then I split the 80% 70/30, with the 70% data set being the training set and the 30% being the cross-validation set. In the end, from a full data set of 3168 instances, we had 1774 instances for training and 760 instances for cross-validating.

3.3 ABAGAIL Implementation

3.3.1. Java Code

Per the Piazza discussion topic #281[3], I modified the example code from AbaloneTest.java using the Eclipse Integrated Development Environment (IDE) to fit the Voice Gender data set. Using tips from Piazza and open-source research, I modified the code (renamed as VoiceGenderTest.java) that implemented the following changes in order to run the optimization algorithms for this assignment:

- The original AbaloneTest.java only loaded one training set, used it to train, and then also used it to determine the misclassification error. Obviously that's what we've been told to avoid when determining true error rate. Therefore, I created a separate "initializeInstances" method (called "initializeInstancesTest") that loaded the testing set, and modified the testing procedure to determine the misclassification error using the testing set instead of the training set. These necessary changes allowed the model to predict the values of the testing set and report real accuracy.
- A bug in the StandardGeneticAlgorithm class was taking instances from the old population and using them directly in the new population. However, since they were chosen at random with replacement, there is a chance that the same instance from the old population would have appeared twice in the new population. If that happened and that instance is then mutated in the same iteration, I could have ended up with the wrong result from calling "getOptimal"[4]. I modified this class to only use a copy of the instance for the new population. That way, only that copy would be mutated.
- I modified several variables in VoiceGenderTest.java in order to tweak it to my data set. Specifically, I modified the number of input layers, the number of hidden layers, and change the size of the "attributes" array to fit my data set.

3.3.2. Neural Network Configuration

I configured VoiceGenderTest.java to use the same hidden layer configuration that Weka generated from Assignment 1. I set the number of Input Layers to 20, the number of Hidden Layers to 12, and number of Output Layers as 1. All experiments in this assignment are based on this 20-12-1 network. All algorithms will be run over 1000, 2000, 3000, 4000, and 5000 iterations. This is so I can compare training times of each algorithm over the

[3] https://piazza.com/class/is4sqcnt24q3zn?cid=281
[4] https://piazza.com/class/is4sqcnt24q3zn?cid=296

number of iterations. All other analysis was done using the output generate from running the algorithms at 5000 iterations.

3.3.3. Algorithm Analysis Methodology

I evaluated and compared each of the three algorithms based on their Sum of Squared Error results over number of iterations, the percentage of incorrectly classified labels over the number of iterations, and the computational time over the number of iterations. The training time data was collected for 1000, 2000, 3000, 4000, and 5000 iteration mark.

3.4 Results and Analysis

First, I examined the performance of each algorithm based on how many instances of the test set were mis-classified over the number of iterations. See **Figure 1.**

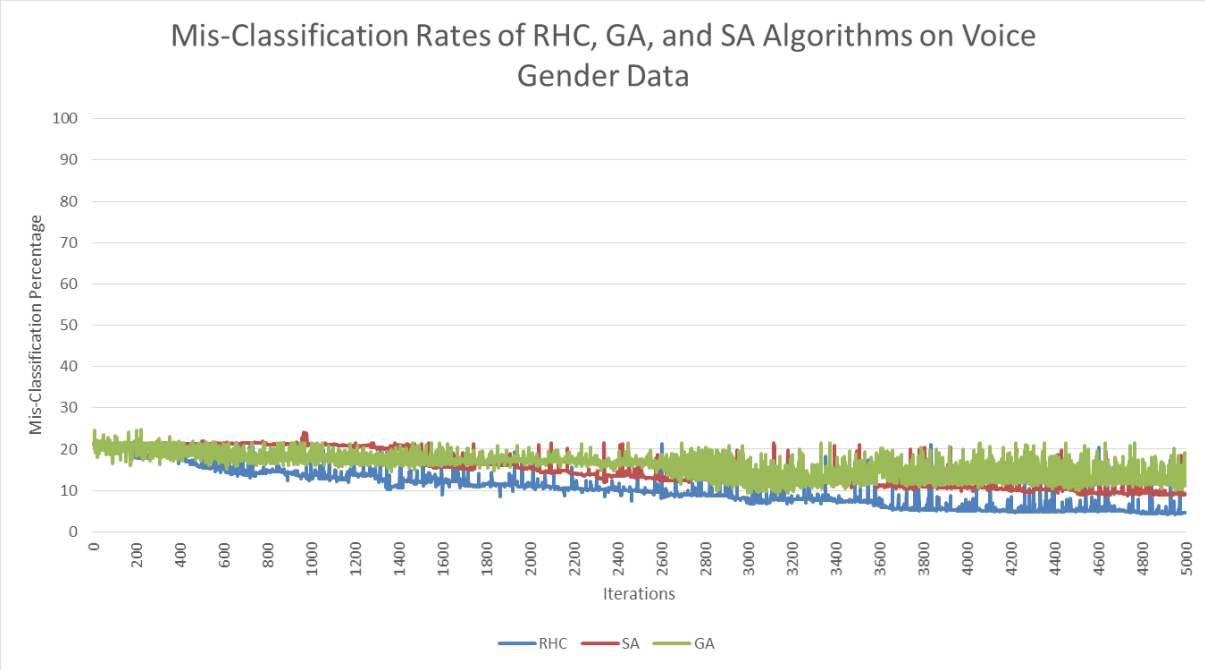

Figure 1: Mis-Classification Rates of RHC, GA, and SA Algorithms on Voice Gender Data

From my experimentation, the RHC algorithm produced the highest accuracy, with SA following suit and GA being the least accurate. From the raw data analyzed in Excel, the RHC algorithm produced the lowest mis-classification percentage (4.287%) at the 4919th iteration. The SA algorithm produced the lowest mis-classification percentage (8.742%) at the 4735th iteration. The GA algorithm produced the lowest mis-classification percentage (9.419%) at the 3137th iteration. Similarly, in **Figure 2**, I examined the Sum of Squared Errors (SSE) for each algorithm over the number of iterations.

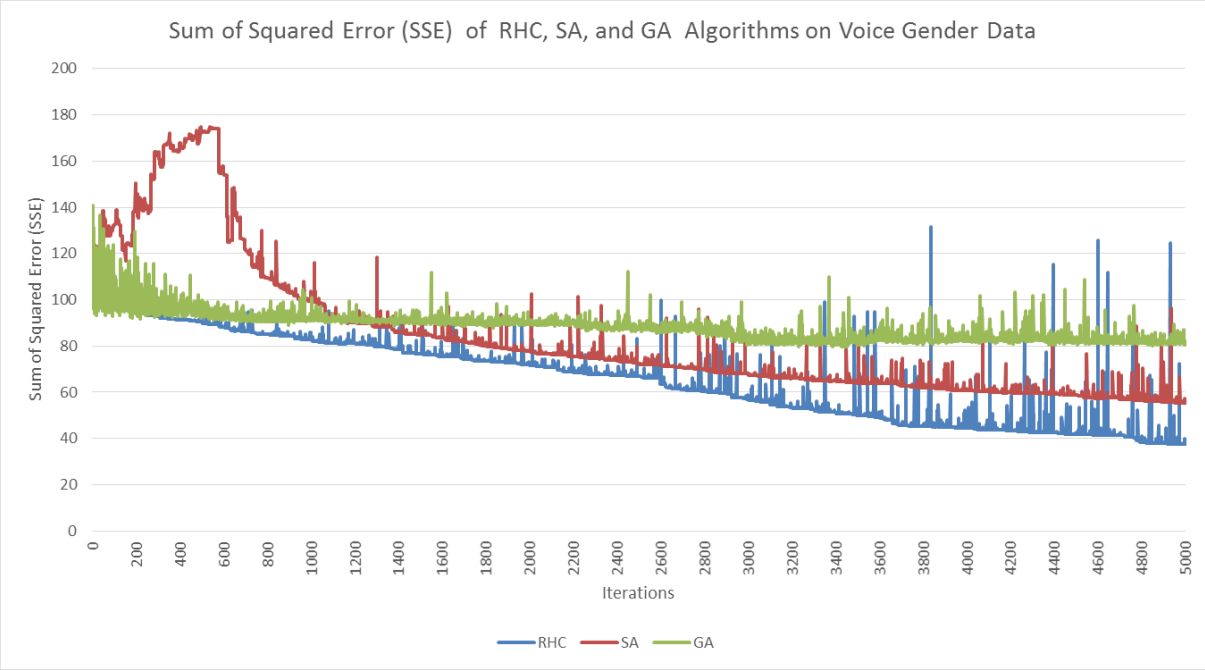

Figure 2: SSE of RHC, SA, and GA Algorithms on Voice Gender Data

Similar to the performance results shown in **Figure 1**, RHC demonstrated the lowest SSE, followed by SA and GA. Interestingly enough, between 1 and 1200 iterations, SA yielded much higher SSE than the other two algorithms, with a generally increasing SSE until around the 600th iteration, and then improving for the rest of the iterations. This is expected behavior from the SA algorithm, as the earlier iterations were a period of "high temperature" and the SSE decreased as the temperature variable "cooled down". Finally, I examined the training time data for each of the three algorithms, which is reflected in **Figure 3**.

Iterations	RHC (s)	GA (s)	SA (s)
1000	4.976	188.006	4.429
2000	9.402	370.931	8.838
3000	13.926	556.023	13.495
4000	18.106	743.841	17.725
5000	22.859	931.101	22.227

Figure 3: Training Time Performance of RHC, SA, and GA Algorithms on Voice Gender Data

My experimentation showed that RHC and GA performed the fastest at training (22.859 seconds and 22.227 seconds at 5000 iterations, respectively), while GA performed considerably worst (a whopping 931.101 seconds for 5000 iterations). SA performed slightly faster than RHC for all iterations.

Therefore, RHC was the best performing optimization algorithm of the three in terms of classification accuracy, while SA was the best performing algorithm in terms of training time.

Finally, I compared the performance of the three algorithms with ANN Back-Propagation (BPP) from WEKA in Assignment 1. I ran the MultiLayerPerceptron classifier with the 20-12-1 structure over 5000 iterations (all other options default) and documented the results. **Figure 4** compares the mis-classification rate between BPP, RHC, SA, and GA, at 5000 iterations, and **Figure 5** compares the training times between BPP, RHC, SA, and GA, for 5000 iterations.

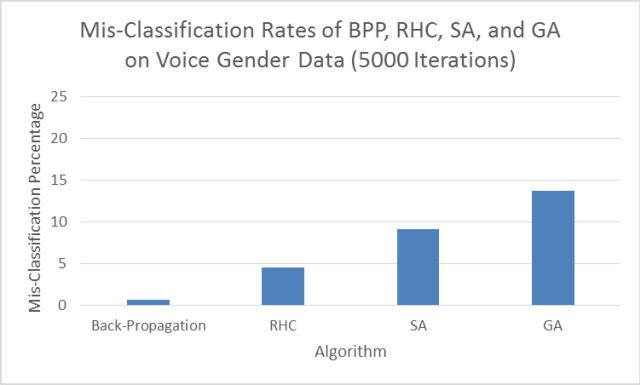

Figure 4: Mis-Classification Rates of BPP, RHC, SA, and GA on Voice Gender Data

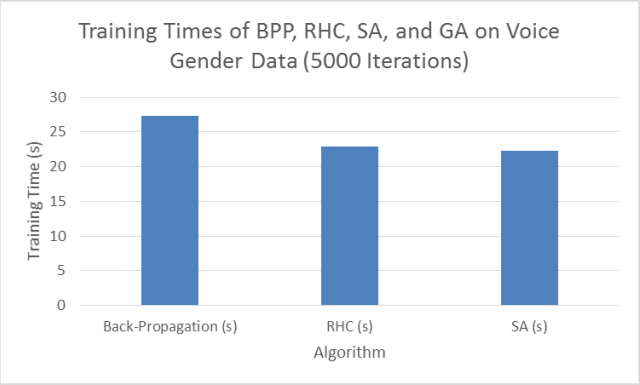

Figure 5: Training Times of BPP, RHC, SA, and GA on Voice Gender Data

Based on these results, I concluded that BPP had the best performance in terms of accuracy on the test set. However, it did not perform as well as RHC and SA in terms of training time.

3.5 Other Considerations

I also experimented with changing certain algorithm parameters for SA and GA to see if the changes affected the SSE and training times. For SA, I modified the Cooling Exponent parameter. For GA, I experimented with several population size, mate, and mutate parameter combinations. For the purpose of controlling the appearance of volatility and to more clearly demonstrate trends, I took a 5-iteration moving average for the SSE. My results are shown in Figure 6, Figure 7, Figure 8, and Figure 9.

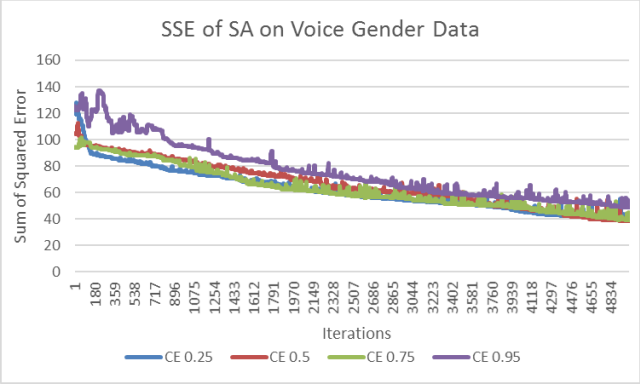

Figure 6: SSE of SA on Voice Gender Data

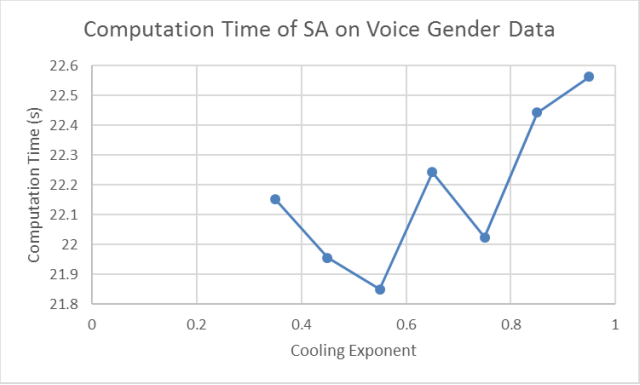

Figure 7: Computation Time of SA on Voice Gender Data

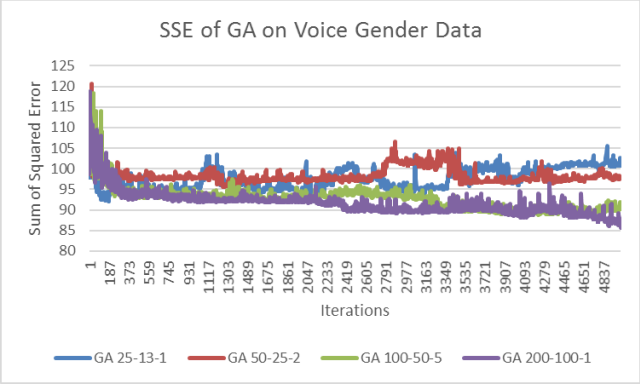

Figure 8: SSE of GA on Voice Gender Data

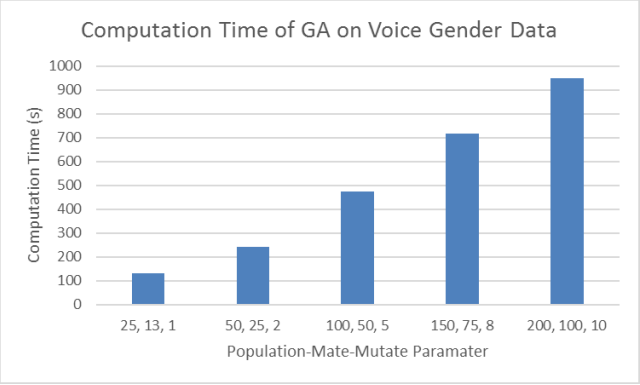

Figure 9: Computation Time of GA on Voice Gender Data

From these results, I concluded that both the CE and the Population-Mate-Mutate parameters have a direct effect on the SSE and the computation time. Both a higher CE for SA and higher Population-Mate-Mutate parameters for GA led to longer computation time. Lower parameters for each algorithm also led to lower SSE.

4. Optimization Problems

In this section of the assignment, I used three different optimization problems to analyze and assess the performance, strengths, and weaknesses of the four randomized optimization algorithms (RHC, SA, GA, MIMIC). I applied all four search techniques to these three optimization problems. The first problem highlighted the advantages of GA, the second of SA, and the third of MIMIC. For each problem, I ran experiments at 1, 500, 1000, 2000, 3000, 4000, and 5000 iterations, and documented the fitness function values and computation times for comparison. Other testing parameters were left to their default states but tweaked later on for further analysis.

4.1 Travelling Salesman Problem

The Travelling Salesman Problem "describes a salesman who must travel between N cities. The order in which he does so is something he does not care about, as long as he visits each one during his trip, and finishes where he was at first. Each city is connected to other close by cities, or nodes, by airplanes, or by road or railway. Each of those links between the cities has one or more weights (or the cost) attached. The cost describes how "difficult" it is to traverse this edge on the graph, and may be given, for example, by the cost of an airplane ticket or train ticket, or perhaps by the length of the edge, or time required to complete the traversal. The salesman wants to keep both the travel costs, as well as the distance he travels as low as possible"[5]. I implemented this problem in ABAGAIL using the TravelingSalesmanTest class. Modifications were made to the code in order to provide better metrics for analysis. The results of my testing are found in Figure 7 and Figure 8.

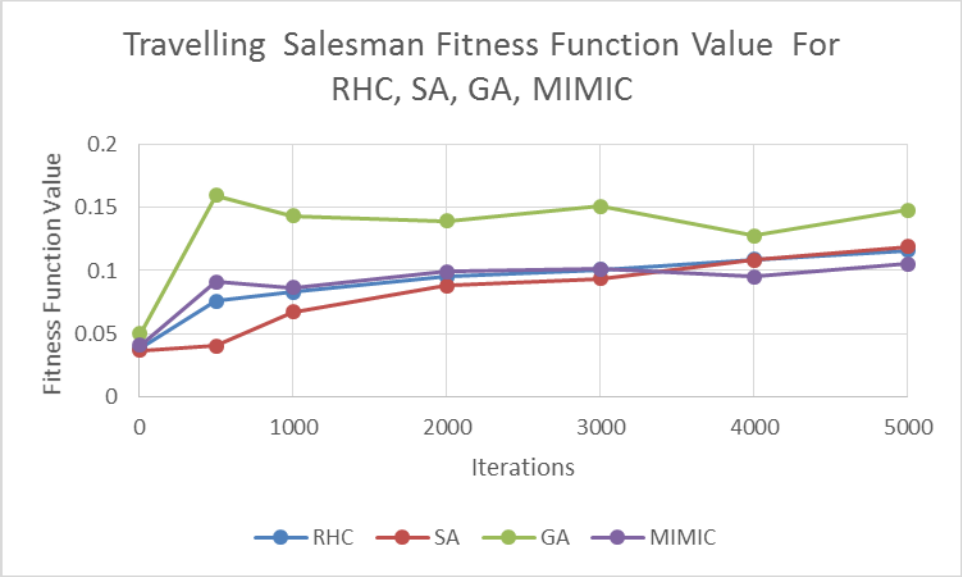

Figure 10: Travelling Salesman Fitness Function Value of RHC, SA, GA, MIMIC

[5] https://simple.wikipedia.org/wiki/Travelling_salesman_problem

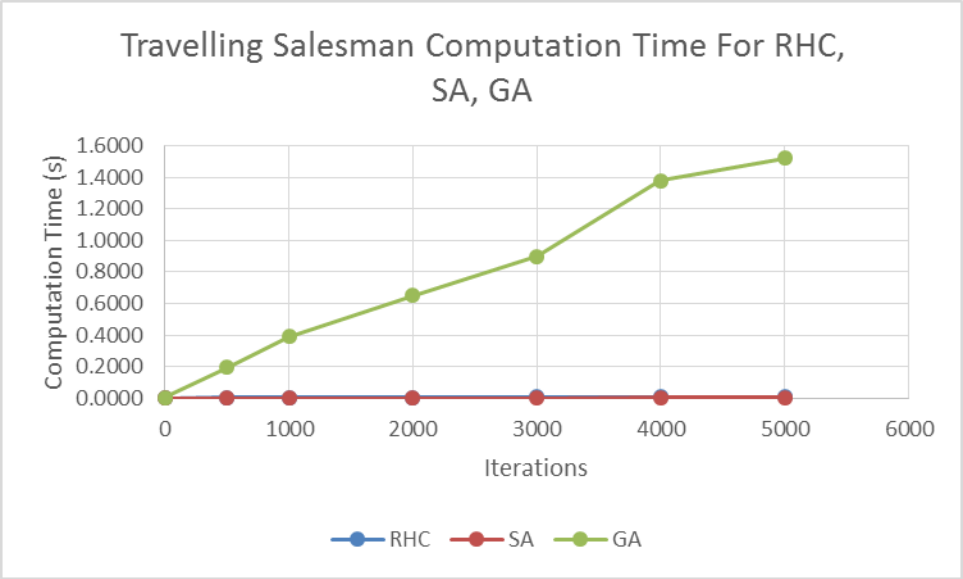

Figure 11: Travelling Salesman Computation Time For RHC, SA, GA

Based on these results, it was evident that GA outperformed the other algorithms in terms of fitness function value. However, it was the most exhaustive in terms of computation time. The reason GA performed the best was because the algorithm's concept is suited for timetabling and scheduling problems. I later tried to modify the population, mate, and mutate parameters to see if they had any effect on performance. See Figure 12 and Figure 13.

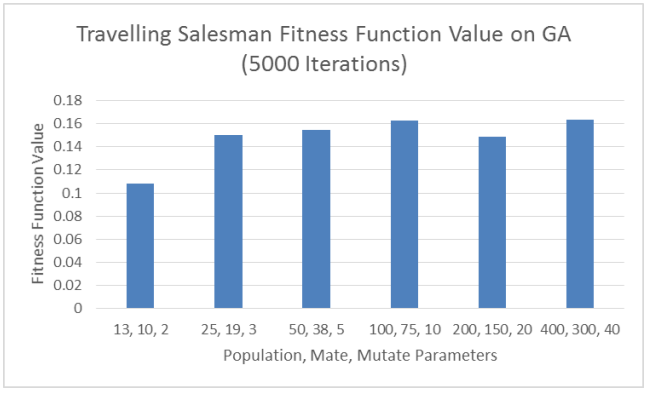

Figure 12: Travelling Salesman Fitness Function Value for GA

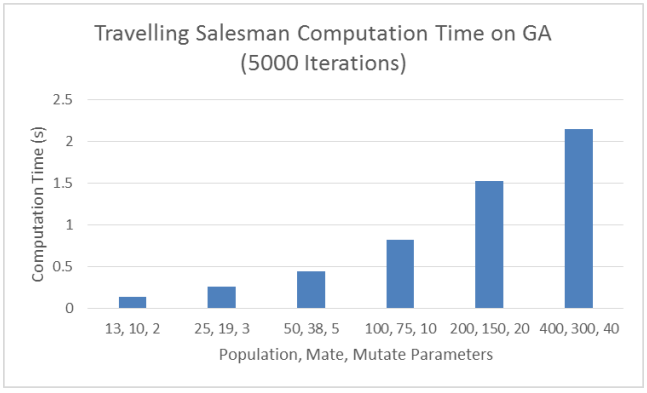

Figure 13: Travelling Salesman Computation Time on GA

From these results, it appeared that both the 100-75-10 and 400-300-40 combination produced the best fitness function value. However, it was also evident that larger parameters meant longer computation time. Therefore, the 100-75-10 configuration was optimal for GA.

4.2 Continuous Peaks Problem

The Continuous Peaks Problem is a study that exemplifies the struggle between finding the global optimum, or highest peak, as opposed to local optimum, or subsidiary peaks. I compared the performance of RHC, GA, SA, and MIMIC in finding the fitness function values and the computation time required. This problem was implemented in ABAGAIL using the ContinuousPeaksTest class. Some modifications were made to the class in order to provide better metrics for analysis. The results of my testing are found in **Figure 14** and **Figure 15**.

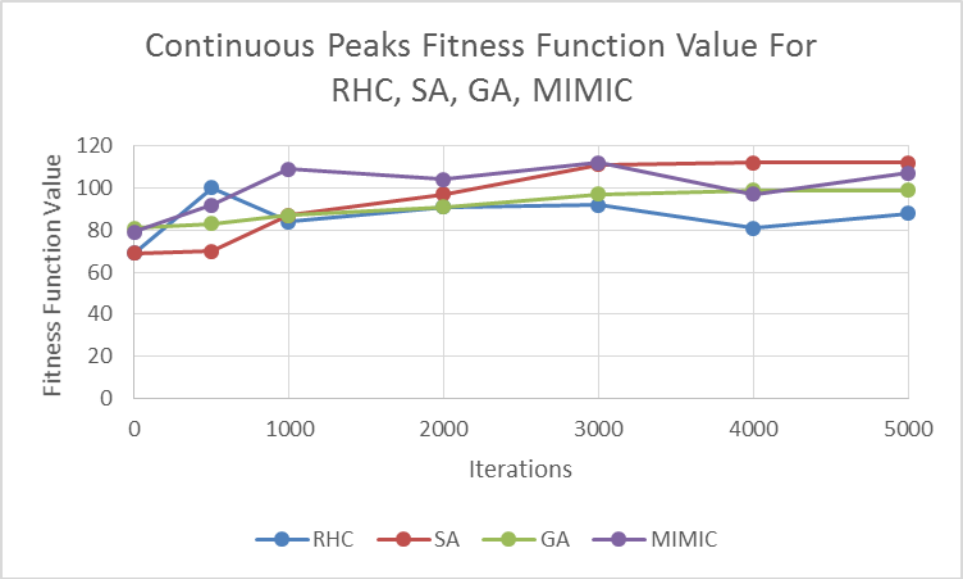

Figure 14: Continuous Peaks Fitness Function Value for RHC, SA, GA, MIMIC

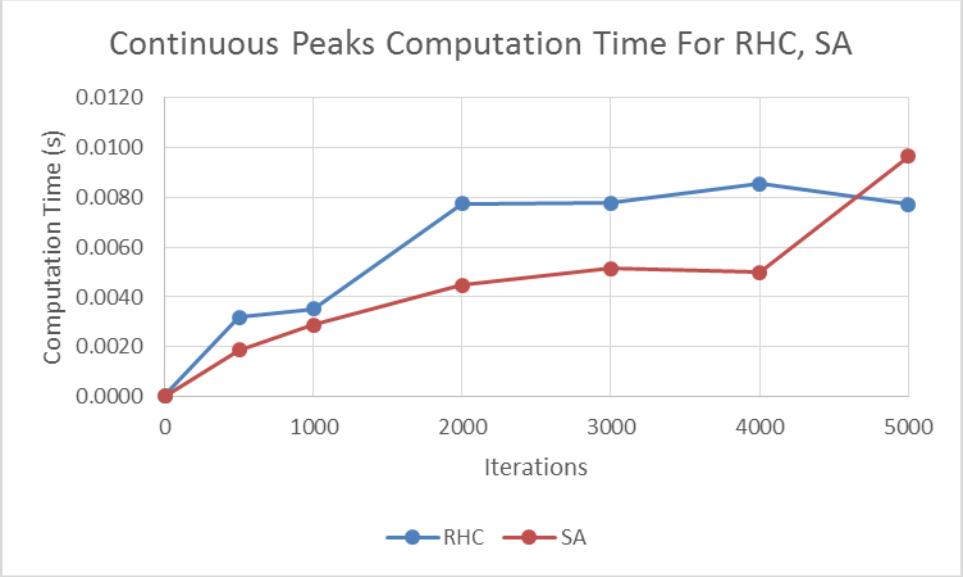

Figure 15: Continuous Peaks Computation Time for RHC, SA

From these results, it appeared that SA outperformed the other algorithms for larger iterations, even though it did not outperform for smaller iterations. This is due to the fact that SA has to gradually "cool" in order to find a global optimum. Once the state has sufficiently cooled, it has found a neighborhood where the global optimum lies. This is why SA is particularly suited for the Continuous Peaks problem, as being stuck on a local optimum is a big concern. From my experimentation, both RHC and SA had the lowest computation time (the other algorithms were orders of magnitude higher). When I compared RHC computation time with SA, SA had a lower computation time except for 5000 iterations. I experimented with the SA Cooling Exponent (CE) parameter to see if it affected performance. My results are shown in **Figure 16** and **Figure 17**.

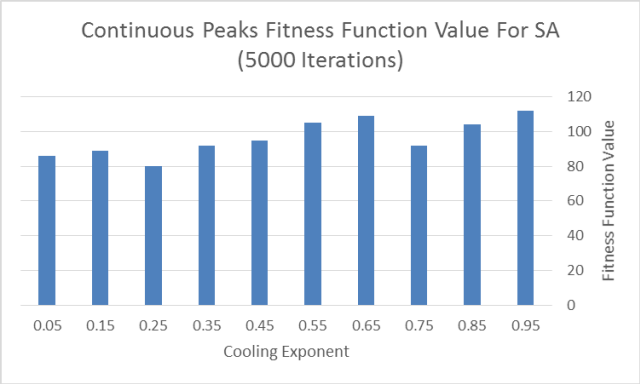

Figure 16: Continuous Peaks Fitness Function Value for SA

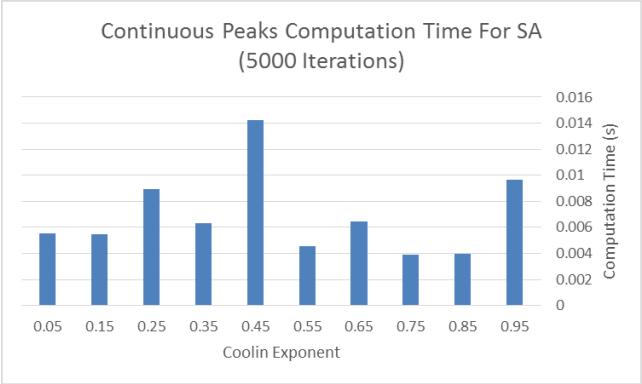

Figure 17: Continuous Peaks Computation Time For SA

From these results, I concluded that 0.95 CE value produced the highest fitness function value. However, it was also the second slowest.

4.3 Knapsack Problem

The Knapsack Problem is "a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items."[6] This problem was implemented in ABAGAIL using the KnapsackTest class. Again, I made some modifications to the code in order to provide better metrics for analysis. The results of my testing are shown in **Figure 18** and **Figure 19**.

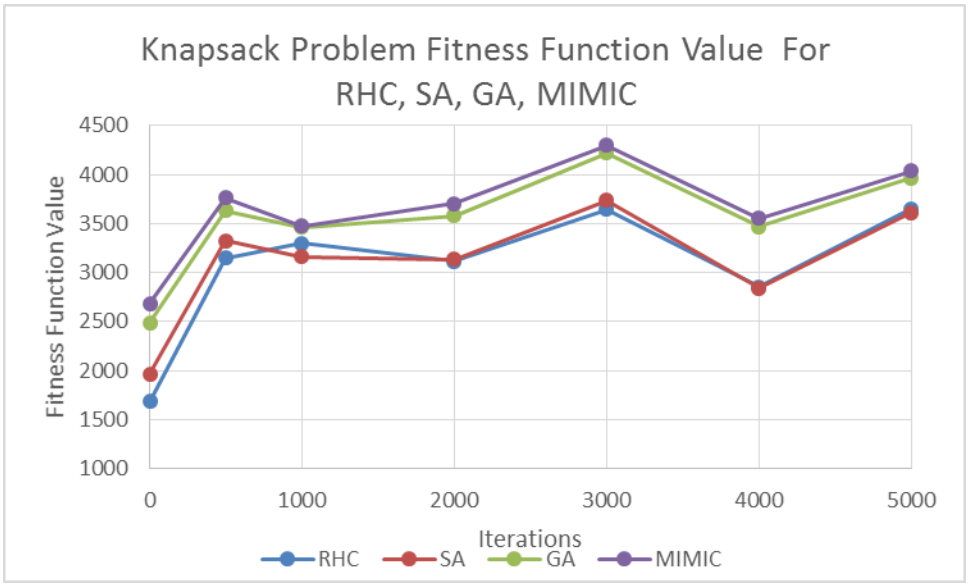

Figure 18: Knapsack Problem Fitness Function Value for RHC, SA, GA, MIMIC

[6] https://en.wikipedia.org/wiki/Knapsack_problem

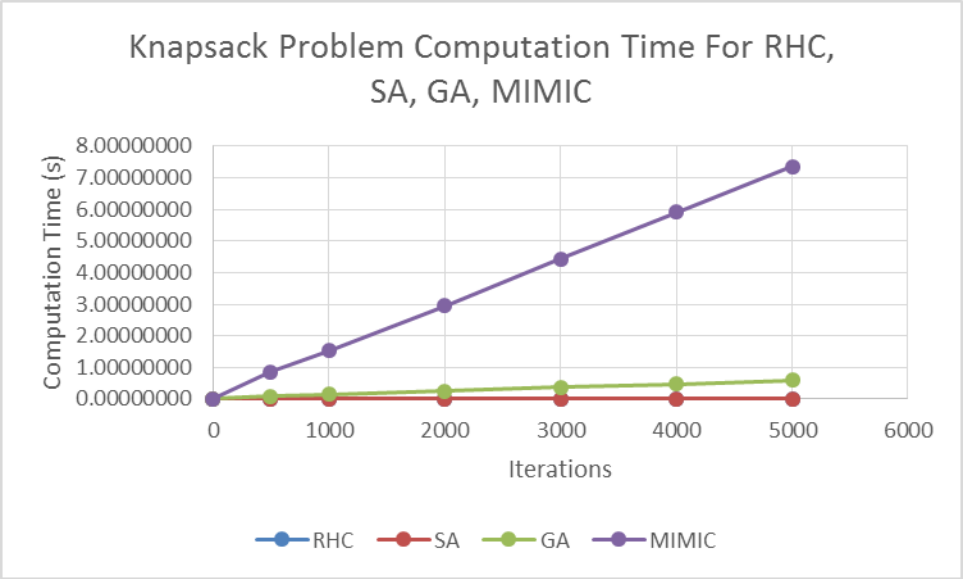

Figure 19: Knapsack Problem Computation Time for RHC, SA, GA, MIMIC

Based on these results, MIMIC outperformed the other algorithms in terms of the fitness function value. However, as evidenced in the other optimization problems in this assignment, it performed the slowest, by orders of magnitude. I modified the population and sample parameters to see if it affected MIMIC performance. My results are shown in Figure 20 and Figure 21.

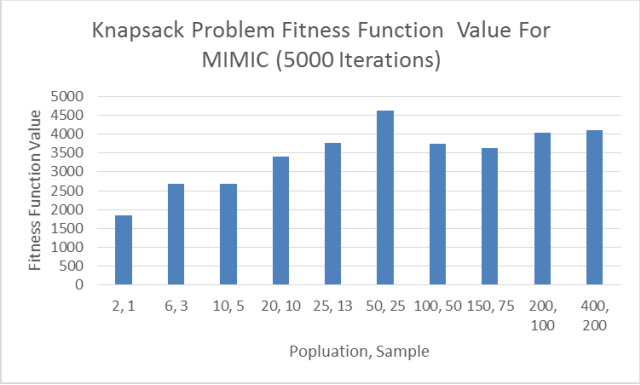

Figure 20: Knapsack Problem Fitness Function Value for MIMIC

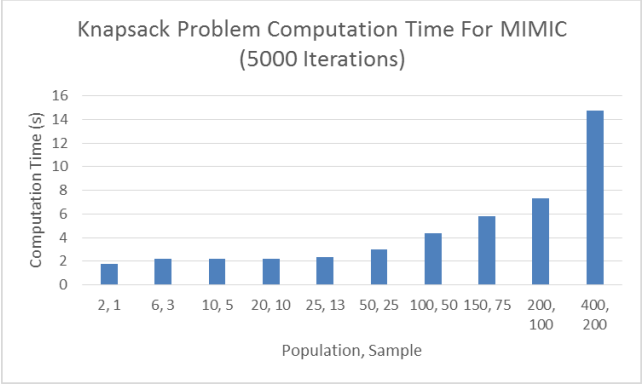

Figure 21: Knapsack Problem Computation Time for MIMIC

From these results, the 50-25 parameter configuration produced the highest fitness function value. In addition, the higher the population-sample parameters, the higher the computation time. Therefore, I concluded that the 50-25 parameter was the optimal parameter for MIMIC for this problem.

5. Conclusion and Other Thoughts

In this assignment, I explored randomized optimization algorithms in two parts. First, I applied RHC, GA, and SA to the Voice Gender data set from Assignment 1 and compared their performances against each other and ANN BPP. Second, I explored RHC, GA, SA, and MIMIC performance for three optimization problems, highlighting the specific algorithms that each problem was suited for.

For future consideration, I would have used Design of Experiments to setup a factorial design in order model the performance impact of MIMIC, SA, and GA based on a more procedural modification of their algorithm parameters. I did not have the proper software tools to conduct such an experiment, so I manually picked some parameter configurations.