CS7641 - Random Optimization Report

Shawn R. Mailo

October 15, 2018

1 Introduction

This report explores different types of optimization algorithms to learn how they work, as well as their advantages and disadvantages. The report contains two parts in which in the first part the report explores how different optimization algorithms (Randomized Hill Climbing, Simulated Annealing, Genetic Algorithm, MIMIC) can help with finding weights for a neural network to solve the TELCO Customer Churn Dataset. The second part explores 3 optimization problems that are included in the ABAGAIL framework in order to highlight the advantages of the simulated annealing, MIMIC, and genetic algorithms.

2 Optimization Algorithms

2.1 Randomized Hill Climbing

Randomized Hill Climbing is a simple optimization algorithm that uses iterative searching. It begins at a random solution to the problem and iteratively searches for a better solution. It does this by searching the neighbors of the original solution and changes to a new solution if it provides a better outcome. It does this repeatedly until no further improvements can be found. To avoid local optima, Randomized Hill climbing implements random restarts where it randomizes the original stating location.

2.2 Simulated Annealing

Simulated Annealing is a probabilistic algorithm that is often used for large and discrete search spaces when finding a global optima is preferable than just finding a local optima. This algorithm is based off of the metallurgy concept of annealing in which it uses the notion of slow-cooling to decrease the probability of accepting worse searches while the search space is explored. Towards the end, as temperature minimizes, the algorithm only focuses on the solutions that provide better outcomes.

2.3 Genetic Algorithm

The Genetic Algorithm is an optimization algorithm that is inspired by the biological process of natural selection. Specifically, this algorithm uses mutations, crossovers, and selections to generate high quality solutions to optimization problems. It starts out with a population of randomly generated solutions to the problem. In each iteration the selected population is evaluated based

on a fitness function. The most "fit" (better solutions) are selected, and each of the fit candidates "genes" are subjected to mutations and crossed overs with others' genes to create a new population of candidates. This process repeats until a specified number of generations or until it reaches a specified satisfactory fitness level.

2.4 MIMIC

MIMIC is a novel estimation algorithm that uses probability densities to find optima. The algorithm starts off approximating the conditional distribution of the inputs based on a bounded cost function. During this process the optima of the cost function becomes more likely. In addition MIMIC discovers and retains information about the structure of the optima through second order statistics. It is known to need less iterations to reach converge. However the computation time per iteration is costly.

3 Part 1 - The TELCO Customer Churn Problem

3.1 Purpose

In this section, 3 optimimzation algorithms (Randomized Hill Climbing, Simulated Annealing, Genetic Algorithm) are applied the TELCO Customer Churn problem and their results are compared to each other as well as to the results obtained in Assignment 1 where back propagation was used for optimization.

3.2 The Churn Dataset

This data set comes from IBM's Sample Data sets and is used to predict the customer behavior in order to develop focused customer retention programs. This data set has 7043 rows of data. Every row in the data set represents a customer while the columns contains the 21 attributes of the customer. All attributes are categorical except for 3 which are numerical. The target attribute is Churn which specifies if the customer discontinues their subscription to the service. Since Churn, the target attribute, is only either 'Yes' or 'No', all models that will be created will be considered binomial classification models. The data set is skewed with 73% of customers having 'No' for their churn attribute. This data is interesting because of the skewness, majority of the attributes are categorical, and the data has contains noise. All the data was preprocessed and split into training, cross validation, and testing sets. These preprocessing steps can be found in the Assignment 1 report.

3.3 Methodology

3.3.1 ABAGAIL Framework and Jython

In order to train the neural networks with customized optimization functions, the ABAGAIL framework was used. This framework comes with a library of functions to train, test, and implement machine learning algorithms. Since this frame work uses JAVA, JYTHON was installed/used in order to write python code which can utilize the JAVA libraries. The AbaloneTest.JAVA script

was edited and modified to accept an outside dataset and to output a csv with the following output information (Iteration Number, Training Mean Square Error, Validation Mean Square Error, Testing Mean Square Error, Training Accuracy, Validation Accuracy, Testing Accuracy, Elapsed Time)

3.3.2 Neural Network Learner

To replicate the neural network used in Assignment 1, the neural network parameters where set as follows: input layer with 41 inputs, 1 hidden layer with 5 neurons, output layer with 1 output. During training, these neural networks were ran over 5000 iterations to obtain optimized weights. Once trained, each neural network was tested against a test dataset to analyze and compare their results. Specifically, this report investigates learning curves(accuracy and computation time) and complexity curves(accuracy and computation time) to understand the differences between the algorithms and the effects of hyperparameter tuning.

3.4 Results and Analysis

3.4.1 Learning Curve

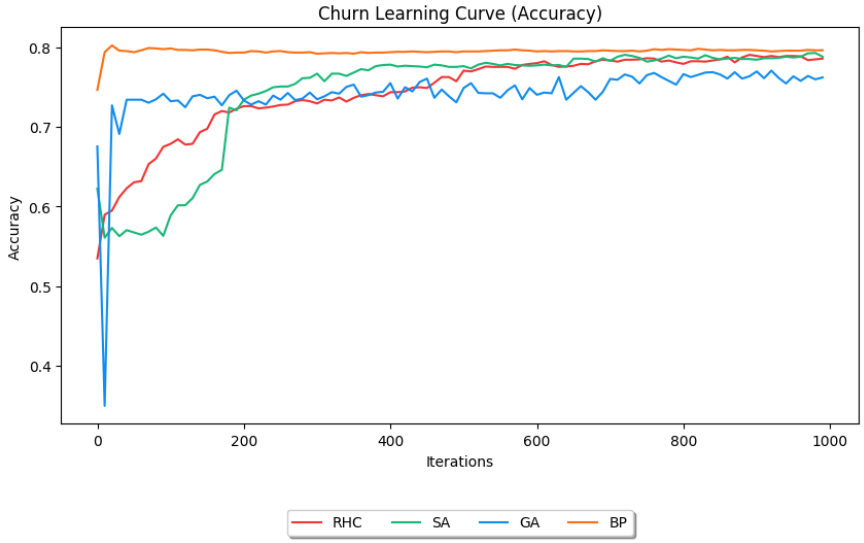

Figure 1: Accuracy Vs Iterations

Table 1: Churn Data Set - Computation Times and Test Accuracy Score

Optimization Algorithm	Computation Time (s)	Test Accuracy Score
Backpropagation	22.45	79.7%
Randomized Hill Climbing	8.75	80.2%
Simulated Annealing	9.97	79.6%
Genetic Algorithm	629.41	76.7%

After training the neural networks over 3000 iterations the top performer with the highest classification accuracy was the neural network with the randomized hill climbing optimization with

a 80.2% accuracy rate. Very close behind was backpropagation with 79.7% and simulated annealing with 79.6%. The genetic algorithm performed significantly worse with a 76.7% test accuracy. One thing to notice on the Learning Curve figure, is that backpropagation plateaued at its maximum much faster than the other algorithms (approx 20 iterations). The winner, randomized hill climbing, didn't reach close to its maximum until after 1500 iterations.
Looking at the table, Randomized hill climbing also had the fastest computation time with 8.75 seconds to complete 3000 iterations. Backpropagation scored significantly less at 22.45 seconds (however keep in mind, it had been plateauing for a longtime before it reached 3000 iterations). And Genetic Algorithm scored the least at 630 seconds.

3.4.2 Complexity Curve

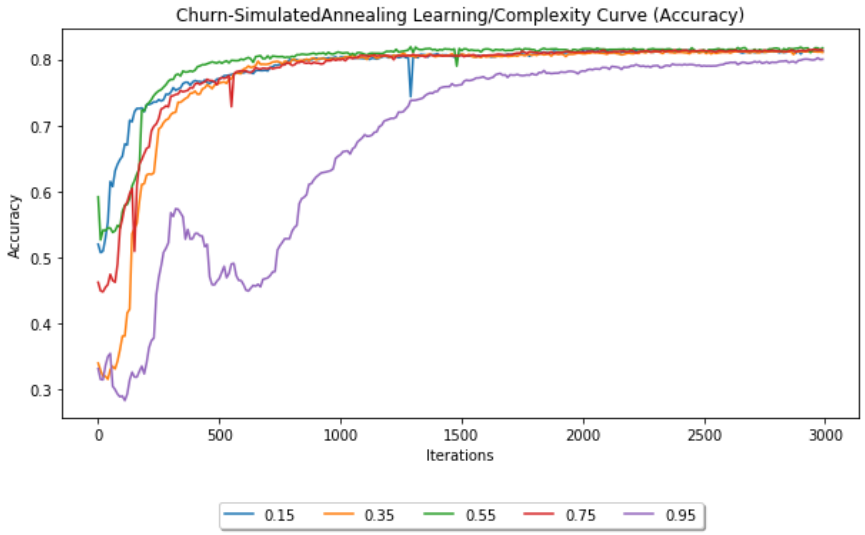

Figure 2: Accuracy Vs Iterations

Table 2: Hyperparameter Tuning for Simulated Annealing Algorithm - Computation Time and Test Accuracy Results

Cool-Down Variable	Computation Time (s)	Test Accuracy Score
0.15	8.49	79.8%
0.35	9.32	79.7%
0.55	9.97	79.6%
0.75	10.23	79.8%
0.95	9.14	78%

To understand simulated annealing better, tuning of its hyperparameter (the Cool-Down Variable) was analyzed. This variable controls the "temperature" of the state of the algorithm. It determines how slowly it takes the algorithm to stop exploring and only exploit the search space. From the graph and the table it seems that after 3000 iterations all versions of the algorithm reached the same optima. When the Cool-down variable is equal to 0.95 the results were about 2% less than the others. This most likely because even at 3000 iterations when the cool-down variable is equal to 0.95 the algorithm is still taking its time to explore worse solutions. This is very evident on the graph as its line stays significantly lower than the others. From looking at the computation

times, there seems to be a positive correlation between the computation time and the cool-down variable. However all versions executed quickly ranging from 8.49 to 10.23 seconds.

4 Part 2 - Optimization Problems

4.1 Traveling Salesman

This optimization problem was formulated in the 1930s, has been studied intensely, and is used as a benchmark for optimization methods. The traveling salesman problem asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?". With many permutations of different routes this problem becomes very complex as the number of cities increases. Because the evaluation function looks for global maximas, 1/distance was used in the fitness function. Using ABAGAIL, the TravelingSalesmanEvaluationFunction.java script was used and modified to obtain results for this report.

4.1.1 Traveling Salesman - Learning Curve

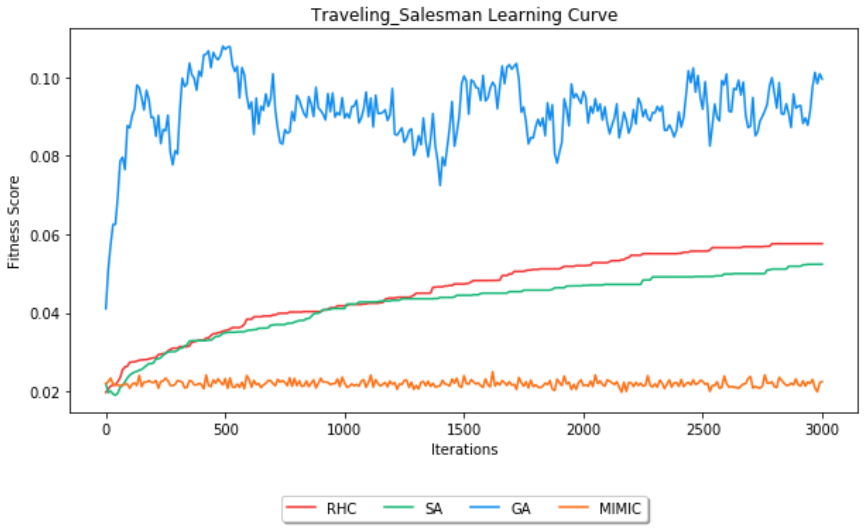

Figure 3: Fitness Vs Iterations

Table 3: Traveling Salesman - Computation Times and Test Fitness Score

Optimization Algorithm	Computation Time (s)	Fitness Score
Randomized Hill Climbing	0.03	0.058
Simulated Annealing	0.006	0.052
Genetic Algorithm	0.873	0.1
MIMIC	669.065	0.022

Based on the graph and table, it is evident that the genetic algorithm performed significantly better than all other optimization algorithms. It plateaued (with noise) much early than the

other algorithms at around 250 iterations. Naturally Genetic algorithms have more noise due to its nature of mutations and crossovers. In terms of computation time The genetic algorithm performed significantly worse than the other algorithms at about 670 seconds to complete 3000 iterations. Having a fitness score more than double of the others, makes it easier to overlook the longer computation times. One reason why the Genetic Algorithm performed best for the traveling salesman problem is because it is known to do well with timetabling and scheduling problems.

4.1.2 Traveling Salesman - Complexity Curve

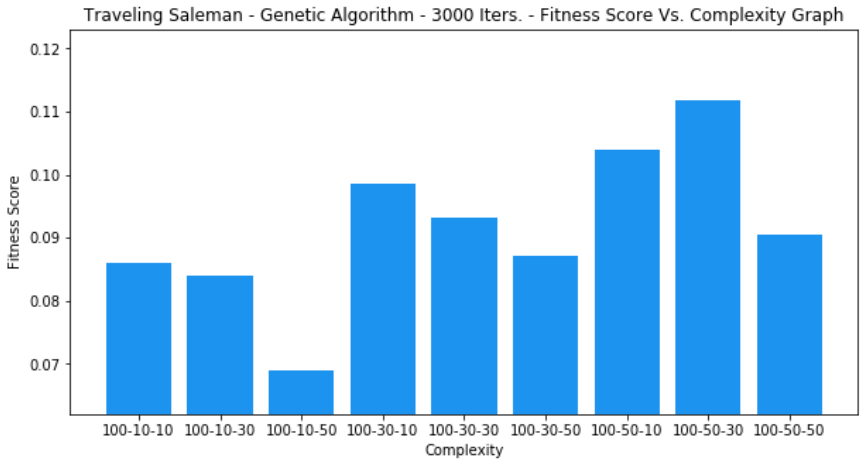

Figure 4: Fitness Score Vs Complexity

Table 4: Hyperparameter Tuning for Genetic Algorithm - Computation Time and Test Fitness Results

Pop-Mate-Mutate	Computation Time (s)	Test Fitness Score
100-10-10	0.263	0.086
100-10-30	0.33	0.084
100-10-50	0.363	0.069
100-30-10	0.581	0.099
100-30-30	0.644	0.093
100-30-50	0.693	0.087
100-50-10	0.877	0.104
100-50-30	0.899	0.112
100-50-50	1.085	0.091

To understand Genetic Algorithms more indepth, an analysis of its hyperparameters were performed. Tuning was done to both the mutation and the mate variables. From the graph and the table it is quite evident that higher values for the mate variable improved the fitness score. In addition, decreasing the mutate variable seemed to get better results except for one example (100-50-30). When looking at computation times, when increased, both parameters lead to higher computation times. The optimal solution was found with the parameters set to 100-50-30.

4.2 Continuous Peaks

The Continuous Peaks problem is defined as the following: With an input vector of N binary elements, a reward is given when there are a greater number of contiguous bits set to 0, and a greater number of contiguous bits set to 1. This problem has many different local optima making this problem tricky and a good way to compare optimization techniques. Using ABAGAIL, the ContinuousPeaksEvalulationFunction.java script was used to obtain results for this report.

4.2.1 Continuous Peaks - Learning Curve

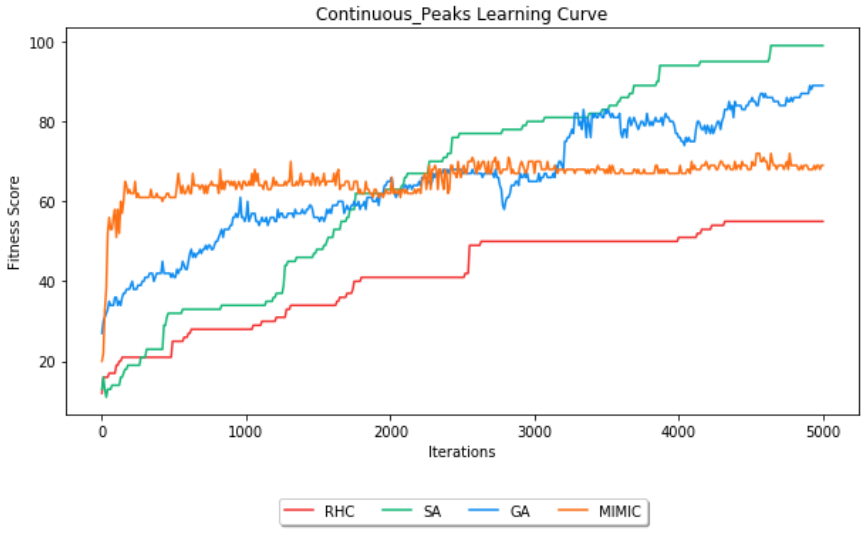

Figure 5: Fitness Vs Iterations

Table 5: Continuous Peaks - Computation Times and Test Fitness Score

Optimization Algorithm	Computation Time (s)	Fitness Score
Randomized Hill Climbing	0.045	55
Simulated Annealing	0.034	99
Genetic Algorithm	0.399	89
MIMIC	28.317	69

The continuous peaks problem showed that simulated annealing was the best performing optimization algorithm with a fitness score of 99. Second place was genetic algorithm with 89. In the lower iterations, MIMIC jumped up in fitness score way quicker than the others but appeared to get stuck on a local optima, apparent by the plateauing score. Simulated annealing also performed the quickest at 0.034 seconds. MIMIC and the genetic algorithm scored significantly worse with a magnitude and 3 magnitude difference. Simulated annealing worked better for this problem most likely because it's good at avoiding local optima where other algorithms may get stuck.

4.2.2 Continuous Peaks - Complexity Curve

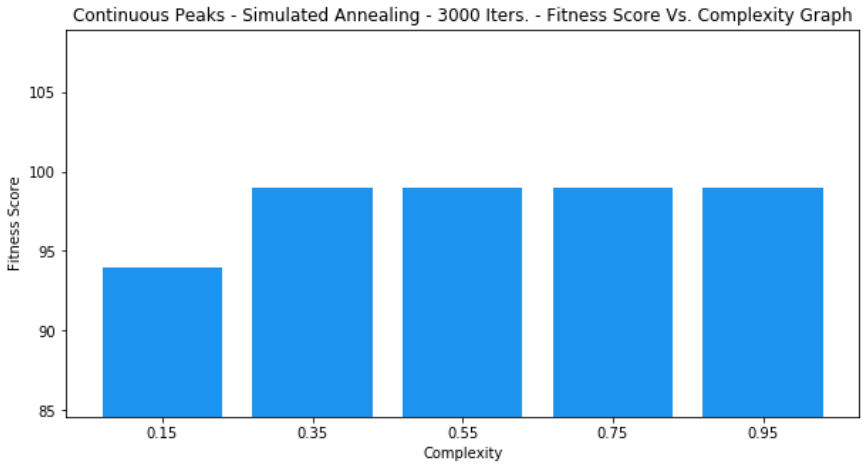

Figure 6: Fitness Score Vs Complexity

Table 6: Hyperparameter Tuning for Simulated Annealing Algorithm - Computation Time and Test Fitness Results

Cool-Down Variable	Computation Time (s)	Test Fitness Score
0.15	0.01	94
0.35	0.049	99
0.55	0.034	99
0.75	0.008	99
0.95	0.034	99

Similar to part 1 simulated annealing was further studied by tuning its cool-down parameter. These results show that for all values of the cool-down variable, except for 0.15, the versions of the algorithm reached the same optima of 99. A possible reason why 0.15 received a worse score is because the low cool-down value limits the exploration of the algorithm. It forces the "temperature" to cool down faster and focus more on exploiting than exploring the search space. Most likely the 0.15 version got stuck on a local optima. All versions had quick execution times which is a characteristic of simulated annealing. Following the nature of the algorithm the results show no correlation between the cool down variable and the computation time.

4.3 FlipFlop

The FlipFlop problem is defined as the following: With an input vector of N binary elements, a reward is given for each bit that is different from both of its closest neighbors. A possible optima would look like the following: "01010101". This problem has many different local optima making this problem tricky and a good way to compare optimization techniques. Using ABAGAIL, the FlipFlopEvalulationFunction.java script was used to obtain results for this report.

4.3.1 FlipFlop - Learning Curve

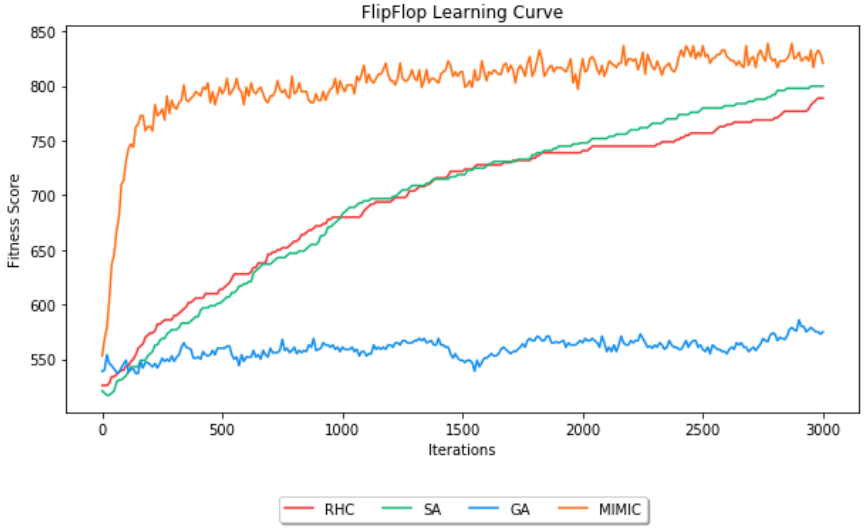

Figure 7: Fitness Vs Iterations

Table 7: FlipFlop - Computation Times and Test Fitness Score

Optimization Algorithm	Computation Time (s)	Fitness Score
Randomized Hill Climbing	0.047	789
Simulated Annealing	0.038	800
Genetic Algorithm	0.678	575
MIMIC	2508.77	821

The FlipFlop problem evidently shows that MIMIC outperformed the other optimization algorithms. Not only did it take less iterations to reach a high fitness score but it also had the highest fitness score out of all 4 algorithms at 821. At 3000 iterations, simulated annealing and RHC came in second and third place with a fitness score of 800 and 789. Due to not have reached convergence, it seems that with more iterations both SA and RHC could be able to reach or surpass the results from MIMIC. The Genetic algorithm performed extremely poorly, ending with a fitness score of 575. In terms of computation time, however, MIMIC scored significantly worse (more than 4 magnitudes) than the others at 2508 seconds. MIMIC is know to be computationally heavy but usually needs significantly less iterations get to an optima. The graph and tables supports this theory.

4.3.2 FlipFlop - Complexity Curve

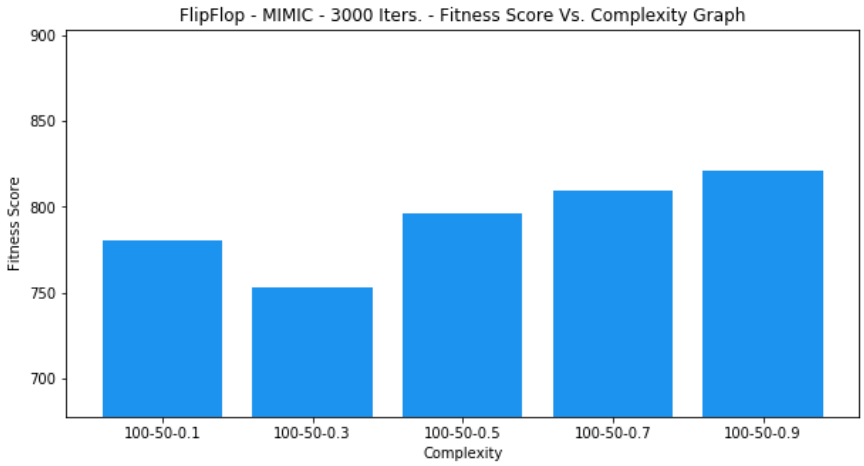

Figure 8: Fitness Score Vs Complexity

Table 8: Hyperparameter Tuning for MIMIC Algorithm - Computation Time and Test Fitness Results

Samples-Keep-M	Computation time (s)	Test Fitness Score
100-50-0.1	2568	780
100-50-0.3	2593	753
100-50-0.5	2502	796
100-50-0.7	2350	809
100-50-0.9	2508	821

To further understand how MIMIC works, a complexity analysis was performed on the algorithm. The hyperparameters of the MIMIC algorithm include population-tokeep-m – m being the Bayesian estimate. A grid search of values (0.1, 0.3, 0.5, 0.7, 0.9) for the Bayesian estimate was used in the FlipFlop problem. Excluding the 0.1 version, the algorithm performed better with the increasing value of the Bayesian estimate. The optimal solution was found at the highest value of the Bayesian estimate, which was 0.9. In terms of computation time, there was no evident correlation with the Bayesian estimate. The computation time ranged from 2350 - 2598 seconds.