Randomized Optimization

Brittany N. Tan
Georgia Institute of Technology, brittanytan@gatech.edu

***Abstract* - This project includes analyses of four different randomized optimization algorithms on the spam e-mail dataset from the UCI Machine Learning Repository. The algorithms used include randomized hill climbing, simulated annealing, a genetic algorithm, and MIMIC, and are implemented by the open source artificial intelligence library ABAGAIL. First, the algorithms are analyzed with respect to backpropagation. Then they are applied to three optimization problems, where the applications highlight the advantages of each of the algorithms.**

Randomized Optimization and Neural Networks

Based on experiments run for neural networks from the Supervised Learning project, the data showed that 6 hidden layers was optimal for the Spambase dataset. For all of the randomized optimization algorithms analyzed in this project, the dataset will be run with this optimal value while tuning the respective hyperparameters. Due to the randomization of the algorithms, averaging the accuracies over several trials allows for better generalization as opposed to analyzing a single trial that may have coincidental outcomes or outliers. Results are also compared to a backpropagation network as a baseline.

Randomized Hill Climbing

For the randomized hill climbing algorithm, the Spambase dataset was split into a training set and a testing set, with the training set being 70% of the total data and the testing being 30%. Five different trials were run, each with 6 hidden layers and 5,000 iterations, and the results were averaged. A plot of the average accuracies over 5,000 iterations are shown in Figure 1.

The training set experienced a steady increase as the number of iterations was increased and reached an average training accuracy of 92.93% at 5,000 iterations. The average testing accuracies, however, tend to remain more constant. One notable observation is that at the iterations where the training accuracies experience a significant increase and then plateau, the testing accuracies experience an opposite change. This happens between iterations 2,900 and 3,200 and also between iterations 4,000 and 5,000. This is likely because the neighborhood of the current "best" point at the time is too small; therefore, the algorithm will slowly converge around this point [1]. Since the algorithm is greedy and makes assumptions based on the neighborhood of the current "best" solution, it will also remain stuck in this hill before recovering to its prior pattern. This displays a false positive in training, which is exhibited by the opposite response from the testing set.

While the training and testing accuracies continue to increase and approach the accuracy achieved by backpropagation, it requires much more time since the number of iterations needs to increase significantly, and even at 5,000 iterations, it has still not reached that accuracy of 94.95%. Additionally, the rate of increasing continuously slows down as the number of iterations increases. Therefore, while backpropagation demonstrates signs of overfitting, it takes significantly longer to reach a similar accuracy using randomized hill climbing.

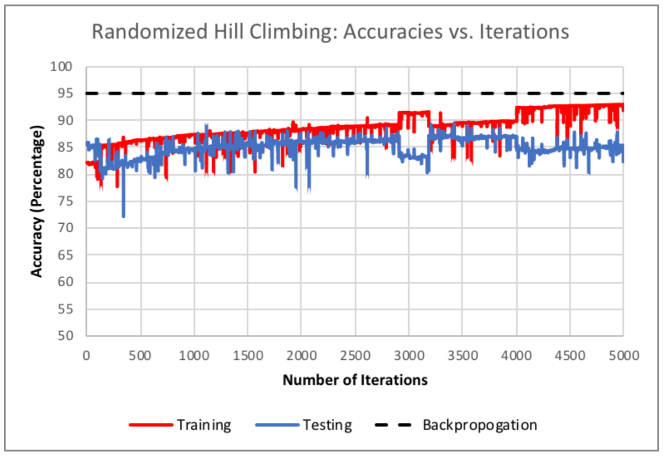

FIGURE 1
TESTING AND TRAINING ACCURACIES BASED ON NUMBER OF ITERATIONS

Simulated Annealing

For the simulated annealing algorithm, the hyperparameters adjusted were temperature and cooling rate, each adjusted separately in order to find the optimal. Five different trials of 5,000 iterations each were run, and results were averaged to generate training and testing curves. While running simulated annealing on different starting temperatures, cooling rate was held constant at a rate of 0.75. The temperatures run were 1×10^{6}, 1×10^{8}, 1×10^{10}, 1×10^{12}, and 1×10^{14}. The training curves and testing curves for the varying temperatures are shown in Figure 2 and Figure 3, respectively.

Both the training and testing data experience a continuous increase in accuracy as more iterations are run, regardless of temperature. The testing data shows greater variance than the training data in the beginning, which indicates variance in the dataset overall. Since the dataset has 58 attributes, this is likely to happen because some attributes may not be directly relevant to classifying the data [2]. Regardless, while there exists more noise in the earlier iterations, the accuracies begin to converge towards similar values by the time it reaches the 5,000th iteration. Overall, the temperature of 1×10^6 demonstrates the best performance by a marginal amount, but since all the values become more similar as the number of iterations increases, there may not exist an actual optimum in temperature. Regardless, 1×10^6 was used to run experiments varying the cooling factor.

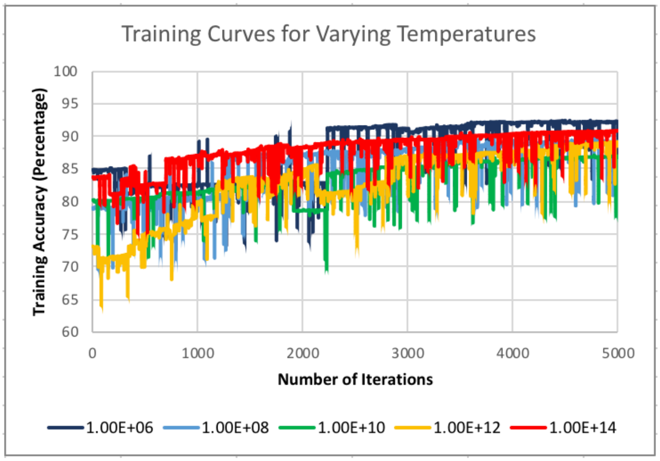

FIGURE 2
TRAINING CURVES BASED ON DIFFERENT TEMPERATURES

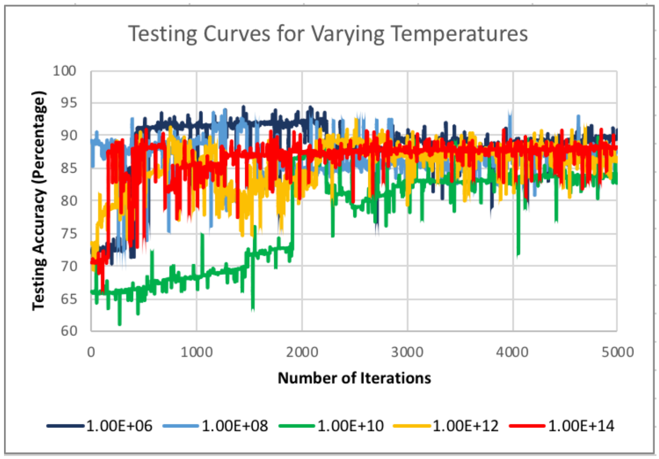

FIGURE 3
TESTING CURVES BASED ON DIFFERENT TEMPERATURES

Figure 3 and Figure 4 display the training and testing curves for varying cooling factors based on the constant temperature of 1×10^6. For the most part, the training curves experience a continuous increase as the amount of iterations increases. Similarly to the testing curves from varying temperatures, these testing curves appear to be more sporadic in earlier iterations, but later begin to converge towards similar values. The changes are more drastic in the beginning iterations because a higher cooling rate is more likely to explore while a lower cooling rate is more likely to exploit.

All of the simulated annealing curves produced more noise than the curves generated by randomized hill climbing, but also maintained their overall trends rather than getting stuck on small hills and then recovering. While five trials are run and then averaged to cut down on the noise, this is bound to happen because simulated annealing explores a the entire search space than that of randomized hill climbing, which only explores certain neighborhoods [3]. However, since there is a greater search space, simulated annealing is less likely to remain stuck on a local "best" point.

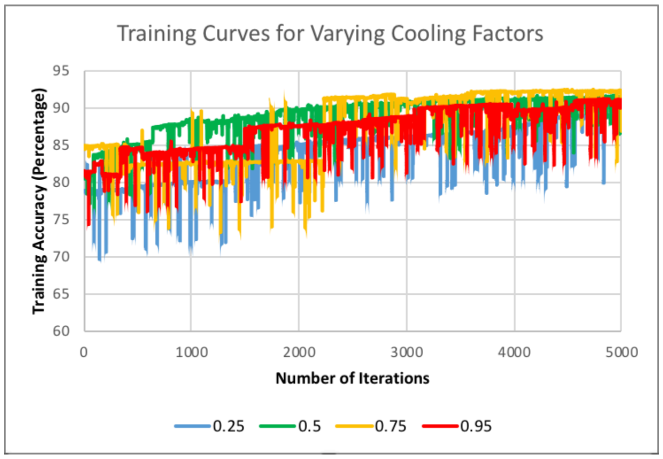

FIGURE 4
TRAINING CURVES BASED ON DIFFERENT COOLING FACTORS

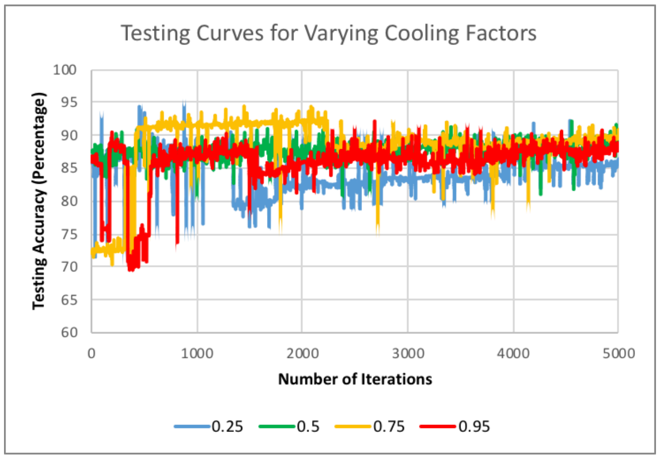

FIGURE 5
TESTING CURVES BASED ON DIFFERENT COOLING FACTORS

GENETIC ALGORITHM

This algorithm has three hyperparameters that can be tuned: initial population size, amount to mate per iteration, and amount to mutate per generation. The first hyperparameter to be tuned was the initial population size. For each population size, three trials of 1,500 iterations were run, and the results were averaged. Three different population sizes were tested, 100, 200, and 300. While altering this parameter, the mating rate and mutation rate were held constant, both at 50. Of the three population values tested, 200 produced the highest testing accuracies. For genetic algorithms, too large of a population makes it difficult to phase out unwanted characteristics because there may exist a larger sample and wider variety of genes. Conversely, if the population is too small, then suboptimal characteristics may be eradicated much more quickly, which may rapidly lead to the population to become homogenous and plateau too soon [4].

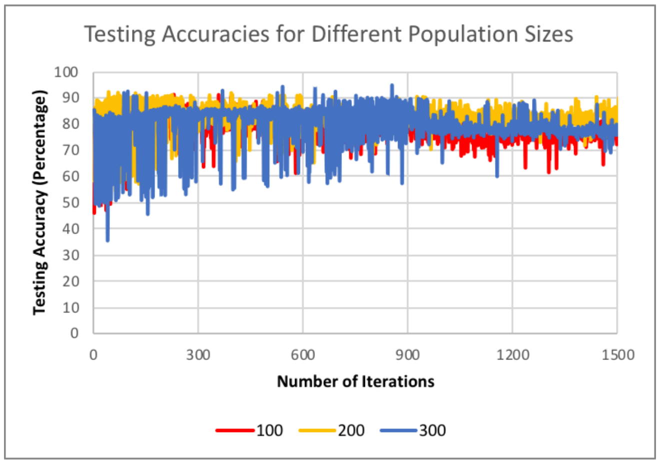

FIGURE 6
TESTING CURVES BASED ON DIFFERENT POPULATION SIZES

The amount to mate per iteration was then tweaked using the initial population of 200 and still holding the amount to mutate per generation at 50. The algorithm was run on three different trials of 1,500 iterations each and then averaged. The experiment tested 25, 50, 75, and 100 mates, and the results are displayed in Figure 7. While the differences may not be considerable, the data is able to display a clear pattern in this hyperparameter. Given fewer maters, there will be less variety. Since this dataset is a binary classification set, then having less variety will allow for a greater likelihood that instances will be classified correctly [5].

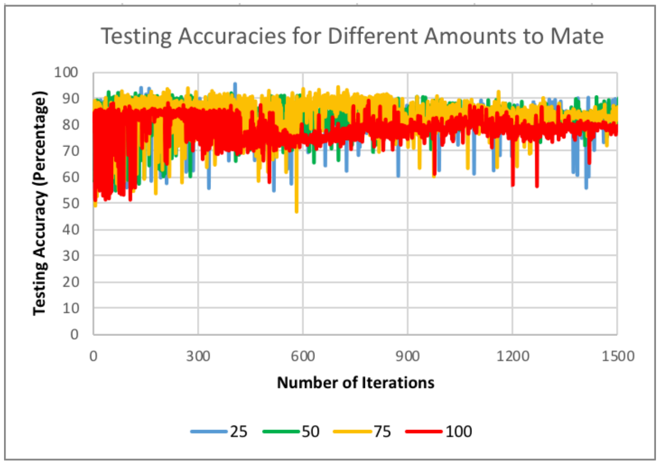

FIGURE 7
TESTING CURVES BASED ON DIFFERENT AMOUNTS TO MATE

The value of 25 mates was kept for running experiments with 25, 50, 75, and 100 amounts to mutate per generation. The same amount of trials and iterations were performed, and results were graphed as shown in Figure 8. All the curves converge near the end of the iterations, but there does not exist a pattern among the varying values of mutations. This is because the mutations for this algorithm are randomly generated, and therefore the accuracies will be swayed by the mutations themselves rather than the amounts of mutations [6].

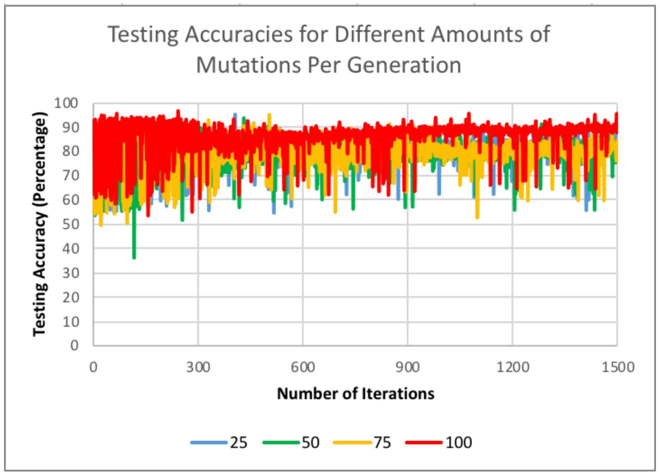

FIGURE 8
TESTING CURVES BASED ON DIFFERENT AMOUNTS OF MUTATIONS PER GENERATION

Of all the randomized optimization algorithms analyzed in this project, this genetic algorithm produces results closest to that of backpropagation on a multilayer perceptron. The genetic algorithm reached the highest accuracies within the fewest amount of iterations; however, it was also the longest to run of the three. This is because the algorithm is recursive in nature in order to preserve the optimal genes within a population. Likewise, backpropagation preserves a propagation of errors that distribute backwards among the network of layers [7].

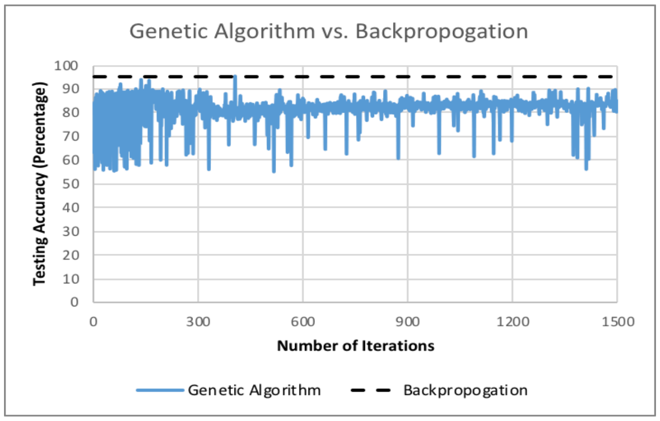

FIGURE 9
ACCURACIES OF GENETIC ALGORITHM VS. ACCURACIES OF BACKPROPAGATION

INTERESTING PROBLEMS

The three randomized optimization algorithms analyzed previously in this project as well as a fourth algorithm, MIMIC, will now be applied to three interesting problems. The Traveling Salesman Problem is a famous NP-hard problem where the goal is to optimize the shortest possible path on a graph that visits all nodes and then returns to the starting node. Flip Flop is an algorithm that returns the number of times bits alternate in a bitstring. Finally, the knapsack problem is a famous combinatorial optimization problem that returns the optimal number of items to include in a collection such that the total weight of the items is as large as possible, while also less than or equal to a given limit.

TRAVELING SALESMAN PROBLEM: GENETIC ALGORITHM

The Traveling Salesman Problem was run using the four optimization algorithms on randomized graphs of size 50. With the optimal parameters found from previous analysis, the algorithms generated a fitness measure per iteration. Figure 10 displays the fitness curves over 1000 iterations, and throughout all iterations, the genetic algorithm performed the best. Since the genetic algorithm is suited for many frequently changing heuristics [8], its fitness measure rapidly increases at the beginning and remains optimal for this problem. Moreover, genetic algorithms are better suited for exploration rather than exploitation because they want to preserve diversity within the dataset. This method of preserving diversity helps to detect local optima [9].

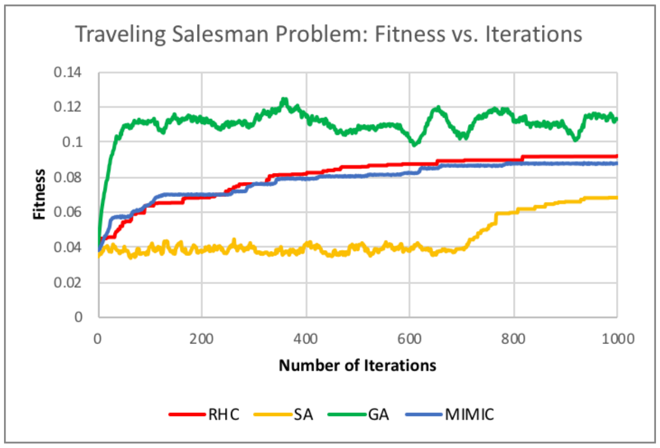

FIGURE 10
OPTIMIZATION ALGORITHMS FOR THE TRAVELING SALESMAN PROBLEM

FLIP FLOP: SIMULATED ANNEALING

The Flip Flop problem was run on randomized graphs of size 80, the preprogrammed value on ABAGAIL. Figure 11 shows the results for each algorithm run, with MIMIC producing the highest fitness measures in the beginning due to the nature of the algorithm, but as the number of iterations increases, simulated annealing's fitness measures began to exceed MIMIC's. This is because the search space for this problem is very small since the problem only relies on bits that are directly adjacent to each other. Therefore, simulated annealing is best suited for more simplistic problems that do not require look ahead due to its principle of localization.

Adversely, the genetic algorithm would be better suited for a more complex problem space. Since this problem consists of bitstrings, there exists a limited amount of permutations for bitstrings of set lengths. Because of this, the hyperparameters for genetic algorithms would not have a significant effect since there does not exist much variety in the problem set.

Running randomized hill climbing on Flip Flop would also not be as effective because once a local optimum is reached, the algorithm would be very loathe to change due to the lack of variety in the problem. The algorithm would be very susceptible to becoming stuck on hills for extended iterations for simplistic datasets.

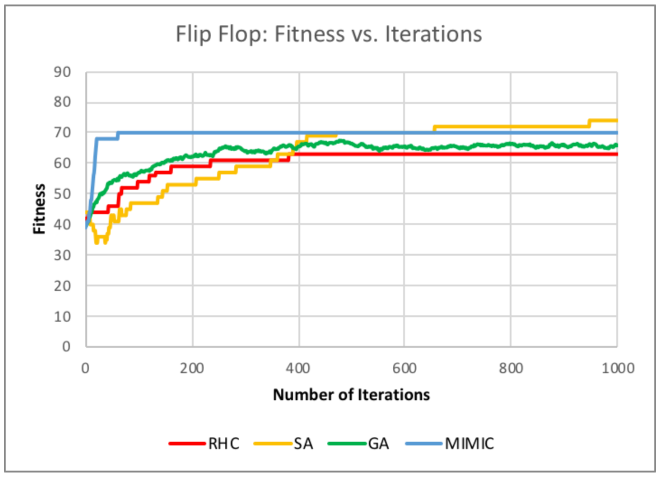

FIGURE 11
OPTIMIZATION ALGORITHMS FOR FLIP FLOP

KNAPSACK PROBLEM: MIMIC

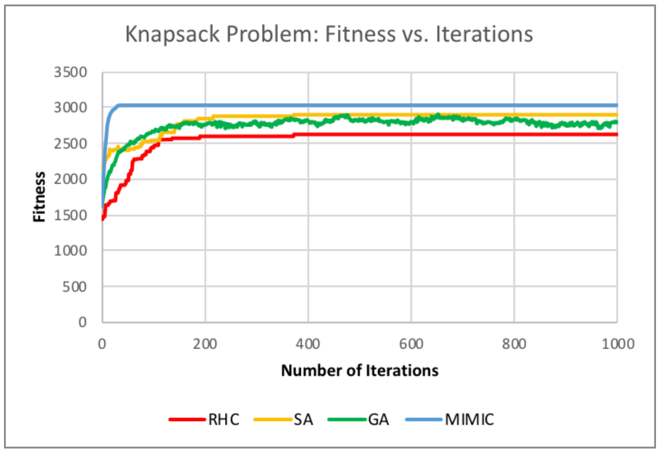

FIGURE 12
OPTIMIZATION ALGORITHMS FOR THE KNAPSACK PROBLEM

The Knapsack Problem was run with 4 copies of 40 items each with a maximum weight and maximum volume of 50 for a single element. All optimization algorithms were run, with MIMIC resulting to be the most optimal. MIMIC requires the least amount of iterations for producing good results, with the tradeoff of taking longer to run since it uses probabilistic functions to produce its results. Since this optimization problem is NP-hard, MIMIC's direct communication of the cost function among the different stages of search can more quickly optimize the number of items to include [10]. MIMIC, like genetic algorithms, is best suited for more complex algorithms that may require retaining information across the search space.

REFERENCES

[1] R. Holte. "Combinatorial Auctions, Knapsack Problems, and Hill-Climbing Search", *Advances in Artificial Intelligence*, pp. 57-66, 2001.

[2] W. Gutjahr and G. Pflug, "Simulated Annealing for noisy cost functions", *Journal of Global Optimization*, vol. 8, no. 1, pp. 3-10, 1996.

[3] R. Storn and K. Price, *Journal of Global Optimization*, vol. 11, no. 4, pp. 341-359, 1997.

[4] J. Horn, N. Nafpliotis and D. Goldberg, "A niched Pareto genetic algorithm for multiobjective optimization", *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*.

[5] G. Jones, P. Willett, R. Glen, A. Leach and R. Taylor, "Development and validation of a genetic algorithm for flexible docking 1 1Edited by F. E. Cohen", *Journal of Molecular Biology*, vol. 267, no. 3, pp. 727-748, 1997.

[6] D. Weile and E. Michielssen, "Genetic algorithm optimization applied to electromagnetics: a review", *IEEE Transactions on Antennas and Propagation*, vol. 45, no. 3, pp. 343-353, 1997.

[7] M. Buscema, "Back Propagation Neural Networks", *Substance Use & Misuse*, vol. 33, no. 2, pp. 233-270, 1998.

[8] S. Chen, Y. Wu and B. Luk, "Combined genetic algorithm optimization and regularized orthogonal least squares learning for radial basis function networks", *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1239-1243, 1999.

[9] Gupta D, Ghafir S. An overview of methods maintaining diversity in genetic algorithms. International journal of emerging technology and advanced engineering. 2012 May;2(5):56-60.

[10] De Bonet JS, Isbell Jr CL, Viola PA. MIMIC: Finding optima by estimating probability densities. InAdvances in neural information processing systems 1997 (pp. 424-430).

AUTHOR INFORMATION

Brittany N. Tan, Undergraduate Student, College of Computing, Georgia Institute of Technology.