Assignment 2 - Randomized Optimization

Brian Hayward
Computer Science
Georgia Institute of Technology
Utrecht, Netherlands
bhayward7@gatech.edu

***Abstract*—In this article, we evaluate the performance of multiple algorithms when applied to different optimization problems. In part 1 of our research, we experiment using each algorithm to solve two common optimization problems and compare and contrast performance. We observe that Simulated Annealing is most effective at solving the K-Color optimization problem and that the Genetic Algorithm is most effective at solving the Four Peaks optimization problem. In part 2 of the research, we experiment using the multiple algorithms to determine optimal weights for a neural network from assignment 1, where we observe that Gradient Descent is the most effective algorithm at optimizing weights in the neural network.**

I. Introduction

In part 1 of this research, we will be experimenting with four different fitness optimization algorithms - randomized hill climbing, simulated annealing, and the genetic algorithm - and evaluating their application for two common optimization problems: K-color and Four Peaks.

In part 2, we will use the same fitness optimization algorithms to identify optimal weights for one of the neural network models from assignment 1. In this section, we will compare and contracts the performance of each algorithm.

II. Dataset

For experimentation, we will use the **US Health Insurance** dataset from assignment 1. The insurance dataset provides an interesting case for experimentation with different optimization algorithms for multiple reasons.

The dataset includes a diverse set of a features, including both numerical features and categorical features. In addition, there are complex relationships between the various features and individual medical costs, likely nonlinear. For example, age has little impact on medical costs early in life, then increase significantly as people enter old-age. Similarly, smoking status can have a significant impact on medical costs, but because the dataset contains a boolean categorical feature, the impact to medical costs across the feature values is stepped. BMI could actually have a parabolic impact on medical costs, where very low and high BMI might correlate with increased medical costs, but average ("healthy") BMI will likely correlate with lower medical costs. The size of the dataset (1338 samples) is also small enough to be manageable, but large enough to provide meaningful results during training and testing.

III. K-Color Optimization

A. Introduction

The first optimization problem that we experiment with is the K-color optimization problem. In this optimization problem, we color vertices, connected by edges, such that no adjacent vertices share the same color. This means that no vertices connected by an edge share the same color. In this context, optimization means that we are trying to reduce the number of conflicts that occur in the graph of connected vertices, where two adjacent vertices share the same color. If we assign a new color to a vertex, this can often violate the constraint and result in a worse solution, meaning we have found a local optimum. With a large number of vertices (individuals from the dataset), the probability of finding local optima when assigning colors is very high. This presents an interesting environment to test each of the optimization algorithms.

To setup this problem, we will use k-color optimization to group individuals from the **US Health Insurance** dataset such that each group is well-distributed and does not contain individuals with similar medical costs (within $3000). In an optimized graph, each color should include minimal individuals with medical costs within the $3000 range. A practical application of this K-Color problem could include spreading costs across different risk pools such that high health insurance premium costs are not concentrated within a single patient demographic (e.g. elderly, high BMI, or smokers).

We hypothesize that **simulated annealing** will perform better that randomized hill climbing or the genetic algorithm in terms of fitness optimization. However, we predict that randomized hill climbing will perform better in terms of wall clock time because it more easily converges on a local optimum relative to simulated annealing and the genetic algorithm. We predict that the genetic algorithm will perform worst in terms of wall clock time.

B. Methodology

We configure each of the algorithms using the *mlrose* Python library. For randomized hill climbing, we configure the hyperparameters *max_iters = 500* and *random_state = 42*. For simulated annealing, we configure the hyperparameters *schedule = mlrose.ExpDecay()*, *max_attempts = 50*, *max_iters = 500*, and *random_state = 42*. For the genetic algorithm, we configure the hyperparameters *pop_size = 100*, *mutation_prob = 0.2*, *max_iters = 500*, and *random_state = 42*.

For the genetic algorithm in particular, we needed to ensure the population size was large enough to ensure genetic diversity and that mutation probability was high enough to facilitate exploration of the solution space. This is meant to help overcome the higher probability of local optima that exist in the K-Color problem.

We then run each of the algorithms using this hyperparameter configuration and report the best fitness score, number of function evaluations, and wall clock time.

C. Results

We observe the following results when testing:

Randomized Hill Climbing - K-Color Optimization	
Best Fitness	92.0
Wall Clock Time	0.02 seconds
Function Evaluations	33

Simulated Annealing - K-Color Optimization	
Best Fitness	54.0
Wall Clock Time	0.37 seconds
Function Evaluations	500

Genetic Algorithm - K-Color Optimization	
Best Fitness	76.0
Wall Clock Time	0.99 seconds
Function Evaluations	17

As we hypothesized, **simulated annealing** outperforms randomized hill climbing and the genetic algorithm in terms of fitness score. A lower fitness score is better in this context, where it indicates a reduction in violation of constraints of the K-Color problem. Randomized hill climbing achieves the fastest wall clock time, and the genetic algorithm converges with the fewest number of function evaluations.

D. Discussion

The results mirrored our predictions, where **simulated annealing** performed best in terms of fitness score, and randomized hill climbing performing best in terms of wall clock time.

Given the frequency of local optima in the K-Color problem, *randomized hill climbing* is a sub-optimal algorithm to use for optimization. Even when randomized, there is a high likelihood that we quickly reach local optima across all iterations of the algorithm, resulting in worse fitness scores. Instead, we need to be able to sometimes accept worse, "pass-through" solutions where we have violated the K-Color constraint, in order to eventually reach a more optimal solution. Knowing this, we would hypothesize that *simulated annealing* would provide a better solution to the K-Color optimization problem. *Simulated annealing* would allow us to accept solutions that violate the K-Color constraint with some probability, which could be adjusted based on the frequency of local optima.

The Genetic Algorithm could also be applied to this optimization problem, but experiences challenges when used to solve the K-Color problem. One example of this is that the *genetic algorithm* is dependent on crossover to identify new solutions, which has high probability of leading to worse solutions with so many local optima existing in the K-Color problem. It can also be difficult to maintain strong population diversity if the distance between each individual is relatively low, meaning the effectiveness of crossover and mutation are lowered and the algorithm is more likely to converge quickly.

IV. Four Peaks Optimization

A. Introduction

The second optimization problem that we experiment with is the Four Peaks optimization problem. In this problem, there exist two global optima and two local sub-optima. This environment makes it challenging for algorithms such as randomized hill climbing and simulated annealing to escape local sub-optima because of their more limited ability to explore the solution space. Oppositely, this problem helps highlight advantages of the genetic algorithm and MIMIC algorithms in exploring the solution space. This makes it an interesting problem for comparing and contrasting these multiple algorithms.

With the above in mind, we hypothesize that the **genetic algorithm** and **MIMIC** algorithm will perform better that randomized hill climbing or simulated annealing in terms of fitness score. Randomized hill climbing and simulated annealing are still expected to have a faster wall clock time.

B. Methodology

To test this, we have created a custom risk assessment score that is based off of multiple features in the **US Health Insurance** dataset: age, BMI, smoker status, and region. Later on, we also perform additional experimentation while using a 5th feature, number of children.

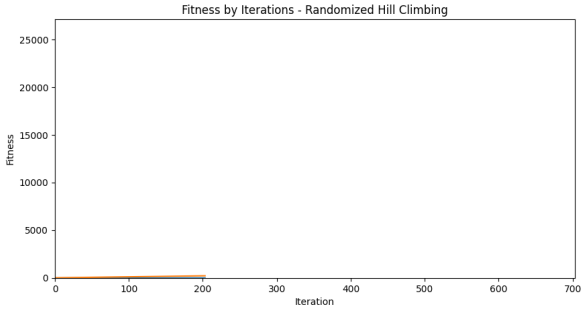
We configure each of the algorithms using the *mlrose* Python library. For randomized hill climbing, we configure the hyperparameters *max_iters = 1000*, *restarts = 10*, and *max_attempts = 100*. For simulated annealing, we configure the hyperparameters *schedule = mlrose.ExpDecay()*, *max_attempts = 100*, and *max_iters = 1000*. For the genetic algorithm, we configure the hyperparameters *pop_size = 200*, *mutation_prob = 0.1*, *max_iters = 1000*, and *max_attempts = 100*. For the MIMIC algorithm, we configure the hyperparameters *pop_size = 200*, *keep_pct = 0.2*, *max_iters = 1000*, and *max_attempts = 100*.

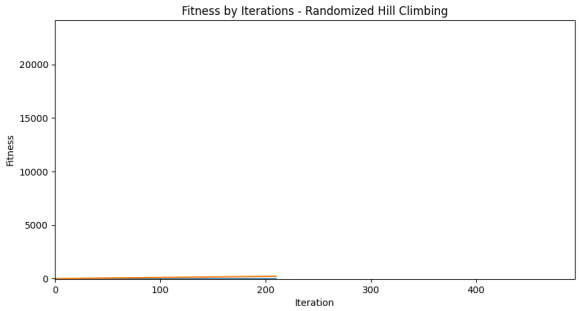
C. Results

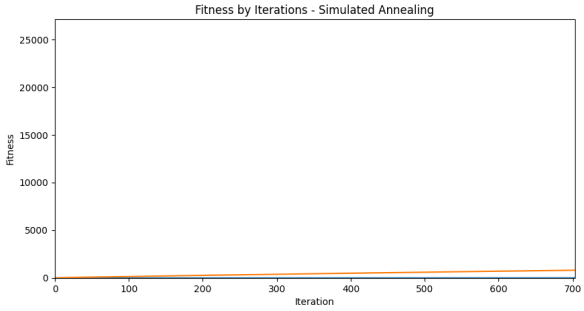
We observe the below results when testing with a problem size of 4, including the features age, BMI, smoker status, and region. We then increase the problem size to 5 by adding number of children to the variables included in the risk factor calculation.

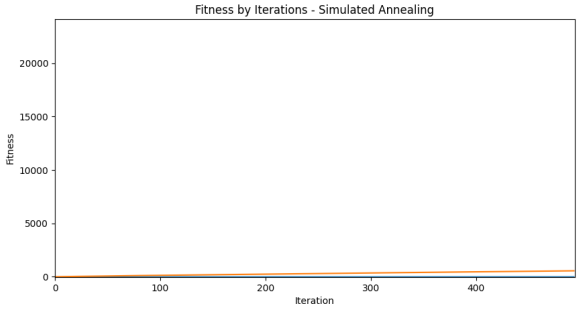
In the context of the Four Peaks problem a higher fitness score is better. As predicted, we observe that the **genetic algorithm** and **MIMIC** algorithm perform best in terms of fitness score. Simulated annealing performs significantly worse than the genetic algorithm in terms of fitness score, but still better than randomized hill climbing. Simulated annealing has

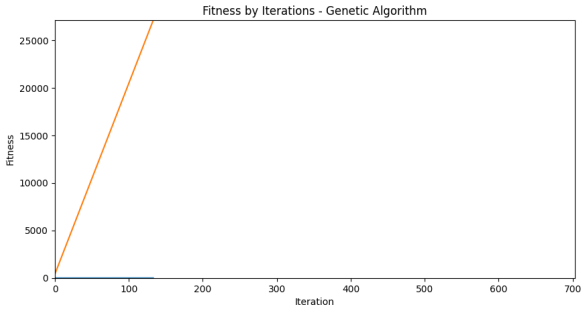
the fastest wall clock time, with randomized hill climbing in close second. MIMIC has the slowest wall clock time.

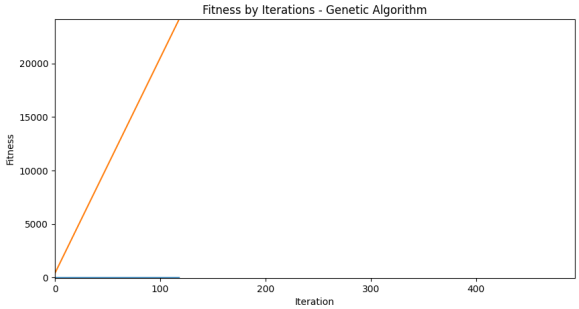


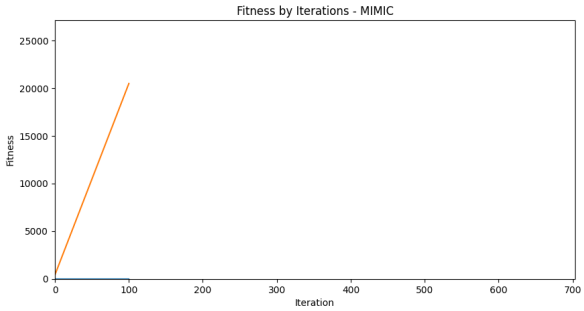


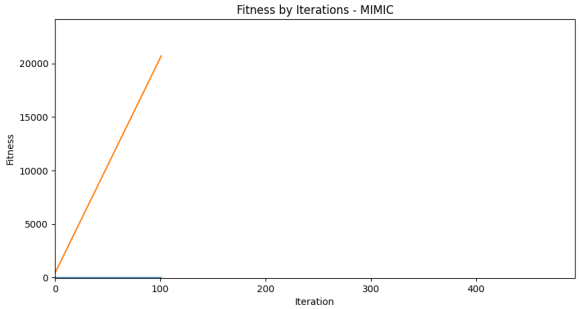












D. Discussion

If we increase the max attempts for *randomized hill climbing*, it will try to find a better neighbor at each step. This

reduces the chance that it will find a local optimum. Thus, we observe that the number of iterations of *randomized hill climbing* increases when we increase max attempts to find a better neighbor solution. However, the fitness score only increases marginally because it is still only able to continue taking small, iterative steps toward global optima. We see similar outcomes when changing the *restart* hyperparameter for randomized hill climbing. Increasing the number of random restarts does not significantly increase the likelihood that the algorithm will be able to escape local optima.

For *simulated annealing*, if we change the schedule to GeomDecay, we see no effect. If we change the schedule to ExpDecay, we also see no effect. Similar to randomized hill climbing, this is likely due to the inability of simulated annealing to explore the solution space, regardless of the temperature cooling schedule. We can try to increase the *max_attempts* or *max_iters* parameters, but these also have minimal impact on the fitness curve for simulated annealing.

For the *genetic algorithm*, if we increase the mutation probability, it results in greater variance in fitness scores by iteration between runs, but it can sometimes result in a much steeper fitness curve. If time is not a constraint, one strategy to optimize fitness could be running the genetic algorithm multiple times while using a higher mutation probability. Alternatively, if we increase population size, we consistently get an increased fitness score. This is a result of greater population diversity, where there is increased opportunity to explore the solution space. For both mutation probability and population diversity, increasing these parameters too much can result in early convergence to a local optimum. So, it is important to balance usage of these parameters to improve fitness, but not converge too early toward local optima.

V. Part 2 - Neural Network Weight Optimization

A. Introduction

In this section, we will compare and contrast the results of applying four different optimization algorithms - *gradient descent*, *randomized hill climbing*, *simulated annealing*, and the *genetic algorithm* - when trying to optimize weights used in the neural network model from assignment one.

The weights in the neural network are continuous with high-dimensionality, and there can exist many local optima. We expect this to have a tangible impact on the performance of these algorithms during optimization, especially for heuristic algorithms such as *randomized hill climbing*, *simulated annealing*, and the *genetic algorithm*. In contrast to *gradient descent*, they do not calculate an exact step in the direction of a local optimum. Instead, they use imprecise, heuristic methods to explore the solution space which often leads to worse selection of solutions and longer run times.

With this in mind, our hypothesis is that *gradient descent* will outperform the other algorithms in terms of weight optimization, error reduction, and wall clock times.

B. Methodology

In order to compare each of the algorithms fairly, we updated the neural network model used in assignment one to instead use the *mlrose* library to calculate the *mean squared error* and *wall clock time* for the gradient descent algorithm.

To evaluate the impact of each of the algorithms, we initially configured the neural network model with the following parameters:

- hidden_nodes=[50]
- activation='relu'
- algorithm='gradient_descent'
- schedule=mlrose_hiive.GeomDecay()
- max_iters=2000
- bias=True
- is_classifier=False
- learning_rate=1
- early_stopping=True
- clip_max=5
- max_attempts=10
- random_state=42

As seen above, we initially set the optimization algorithm to be *gradient_descent*. We then update this parameter for randomized hill climbing, simulated annealing, and the genetic algorithm in order to compare the results of each optimization algorithm.

C. Results

Using the above configuration, we observe the following *mean squared error* and *wall clock times*:

Backward Propagation - Result 1	
Mean Squared Error (MSE)	244177229.86
Wall Clock Time	0.06 seconds
Iterations	14

Randomized Hill Climbing - Result 1	
Mean Squared Error (MSE)	310852872.06
Wall Clock Time	0.75 seconds
Iterations	2000

Simulated Annealing - Result 1	
Mean Squared Error (MSE)	322423117.76
Wall Clock Time	0.75 seconds
Iterations	413

Genetic Algorithm - Result 1	
Mean Squared Error (MSE)	302571939.75
Wall Clock Time	60.02 seconds
Iterations	115

To improve the performance of randomized hill climbing, we attempted to run the algorithm multiple times and choose the best solution. The results of this test while running the algorithm 20 times are below:

Backward Propagation - Result 2	
Mean Squared Error (MSE)	307629718.10
Wall Clock Time	6.73 seconds
Iterations	2000

D. Discussion

Because weights in the neural network are continuous, and there exist many local optima and there are generally smoother optimization landscapes. This increases the difficulty for heuristic algorithms such as randomized hill climbing, simulated annealing, and genetic algorithm in comparison with backward propagation. Backward propagation relies on differentials, and can optimize very efficiently in this type of optimization landscape where it can be exact with adjustments to weights in order to identify better solutions. As a result, backward propagation outperforms all the other algorithms in terms of mean squared error, wall clock time, and iterations.

In our results, we see that the mean squared error of randomized hill climbing, simulated annealing, and the genetic algorithm are much larger than that of back propagation. This is not unexpected because we also know that the heuristic algorithms struggle to identify global optimum solutions. In this case, **randomized hill climbing** found a local optimum that was much higher than the optimum solution identified by back propagation. Due to the different and complex relationships between features and the target variable in the insurance dataset, there can be significant variability in fitness of solutions. This doesn't necessarily increase the risk that randomized hill climbing will find a local optimum. However, given the significant variability in fitness, it does mean there is higher risk that there is a large difference between the local optimum and the global optimum.

The wall clock time for randomized hill climbing was slightly longer than that of back propagation. We can improve the results in a couple different ways. Firstly, we can run the randomized hill climbing algorithm multiple times and compare the local optima. By running it multiple times, we can select the best optimum identified by the algorithm to try to avoid worse local optima. However, this comes at a cost, as it takes longer for RHC to complete.

When attempting this method during experimentation, we saw minimal improvement in selection of an optimal solution. This means that there are a large number of similar local optima, such that running the RHC algorithm multiple times does not find a solution that is closer to a global optimum.

There are a couple ways we could improve results. First, we could run the randomized hill climbing algorithm multiple times and compare the local optima (to then identify a more global optimum). Secondly, we could adjust the learning rate or step size. The higher the learning rate, the more that the weights are changed in each iteration.

Simulated Annealing is similar to randomized hill climbing, but provides a more sophisticated solution for preventing selection of local optima. It sometimes selects, with low probability, a *worse* neighboring solution in order to avoid optimizing toward a local optimum. The probability is determined by selecting a temperature value. Increasing the temperature value has a direct relationship with probability. If the temperature is increased, so does the probability of selecting a sub-optimal solution. If the temperature is decreased, so does the probability of selecting a sub-optimal solution. The temperature is cooled over iterations of the algorithm, decreasing the chance that a sub-optimal solution is selected.

To try to improve results for simulated annealing, we can attempt to increase the initial temperature or decrease the exponent constant in order to expand its exploration of the solution space. We can also attempt to change the temperature decay schedule. However, when optimizing these hyperparameters, the performance of the algorithm in optimizing the neural network still pales in comparison against backward propagation.

Genetic Algorithm takes the longest amount of time to run, but performs better than randomized hill climbing and simulated annealing. This is likely due to its better ability to explore the solution space and escape local optima. However, it takes a much longer amount of time to complete when compared to all the other algorithms. Increasing the mutation probability actually increases the mean squared error, indicating that the algorithm is more likely to converge on local optima.

VI. Conclusion

In this article, we evaluated the performance of multiple algorithms when applied to different optimization problems. In part 1 of our research, we experimented using each algorithm to solve two common optimization problems and compared and contrast ed performance. We observed that Simulated Annealing was most effective at solving the K-Color optimization problem and that the Genetic Algorithm was most effective at solving the Four Peaks optimization problem. In part 2 of the research, we experimented using multiple algorithms to determine optimal weights for a neural network model from assignment 1, where we observed that Gradient Descent is the most effective algorithm at optimizing weights in the neural network.

References

[1] OpenAI. (2023). *ChatGPT* (Mar 14 version) [Large language model]. https://chat.openai.com/chat

[2] Hayes, G. (2019). mlrose: Machine Learning, Randomized Optimization and SEarch package for Python. https://github.com/gkhayes/mlrose. Accessed: 24 June 2024

[3] Anirban Datta. US Health Insurance dataset. Retrieved from https://www.kaggle.com/datasets/teertha/ushealthinsurancedataset/data.

[4] Charles Isbell, Michael Littman. Recorded Machine Learning lecture videos for CS 7641. Retrieved from https://gatech.instructure.com/courses/418394/modules.