OMSCS\_CS7641 HW2

Randomized Optimization

Clement Li (cli620) Fall 2020

# Introduction:

The purpose of this project is to explore random search on discrete optimization problems. These techniques include 1) randomized hill climbing (RHC) 2) simulated annealing (SA) 3) genetic algorithms (GA) and 4) MIMIC. Optimization problems means fitness functions trying to maximize return.   
There are 2 parts to this analysis: 1) three problems will be introduced to the randomized optimization methods to distinguish the benefits/flaws of each and 2) evaluate the randomized optimization methods would be used as a backprop to a neural net. This analysis paper will leverage the mlrose library, particularly some work that ezerilli [1] and make modifications to his code and generate my own results based on different problem sets.

# Part 1: Random Search Algorithms

This section explores the pros and cons of four random search algorithms on finding the optimized solution. These are randomized hill climbing, genetic algorithm, simulated annealing, and MIMIC. The problems being introduced to these algorithms are the traveling salesman problem (TSP), four peak problem, and max k-color problem.

## Traveling Salesman Problem

### Problem Description

|  |  |
| --- | --- |
|  | Figure 1: City coordinates inputs for the Traveling Salesman problem obtained and truncated from https://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html . The coordinates respective from top to bottom are: (6.734, 1.453), (2.233, .010), (5.530, 1.424), (.401,.841), (3.082, 1.644), (7.608, 4.458), (7.573, 3.716), (7.265, 1.268), (6.898, 1.885), (1.112, 2.049), (5.468, 2.606), (5.989, 2.873) |

This problem takes the city coordinates, calculates the distances between all pairs of cities and finds the shortest possible route that visits each city exactly once and returns to the origin city.

### Results/Analysis

|  |  |
| --- | --- |
|  |  |
|  | *Figure 2-4: These plots analyzes the effects of different hyperparameters tuning available in the mlrose toolkit.*  *For GA and MIMIC, shown in Figures 2 and 3, the API allowed tuning to population size and keep percentage, for which I chose 200 to 1000 for population size and 0.1% to 0.8% for keep percentage.*  *The available parameter for SA (Figure 4) is exponential decay rate, for which we choose 0.005 to 0.105 with increments of 0.005.*  *These numbers were chosen based on having chosen a larger range and truncating it after convergence.* |

From figures 2 to 4, we analyze the optimization of parameters using GA, MIMIC, and SA. The objective here is described as the distance taken to visit all the cities exactly once. Therefore, the best algorithm is the one with the lowest cost since we want to take the least distance for the trip. There is a clear separation between the fitness of all three algorithms. This demonstrates that GA has the lowest fitness (54.5) compared with (55.2) and (55.3) for MIMIC and SA respectively. With this new information, we chose the best parameters for each algorithm to compare the objectives and run times with each other.

The results in Figure 5 and 6 shows performance of the search algorithms on the traveling salesman problem; here we observe, like figures 2-4, that GA performs with the lowest cost. MIMIC clearly ends with a higher cost than GA. SA converges with RHC after 400 iterations and the GA curve ends lower than RHC before it converges. Figure 6 shows that GA is faster than MIMIC but slower than SA and RHC to find a solution. The times are linear with clearly higher slopes as we move between each algorithm.

The rationale of the performance lies in the data that is provided. For GA, the algorithm takes 2 parent paths (ex: [**1 6** 4 2 3 5] and [**3 5** 2 6 1 4]) and generate a child path (ex: [3 5 4 2 3 5] and [1 6 2 6 1 4]) with chances of mutation of randomly changing a “chromosome” with a random city. With such this will explore combinations of cities faster. In these plots we have the GA and MIMIC algorithm set to keep 20% of the given parent. Here GA works well because there are no specific routes and swapping percentages of cities around quickly and thoroughly explores many different paths as possible. The SA algorithm occasionally randomly chose its next neighbor. This prevents the process from being stuck on a local minimum. The approach takes a city and tries to find the next city with the lowest distance, with the probability of accepting the city even if it is not the lowest distance. However, this is not as effective as GA to find the optimal solution because it is not as random. The first few accepted cities is already set, whereas in GA there is a chance that even the first couple cities will change. MIMIC takes a cost function and randomly guesses the first half of the solution, then generates more samples for the second half based off the distribution of the first half. This performs worst than GA for this problem because, like SA, it is less random for the first half of the solution. The cost of the MIMIC solution would be heavily dependent on the goodness of the first half guess.

Looking at the time analysis per algorithm, SA/RHC would perform the fastest because it is essentially breadth first search which is known to have time complexity of less than O(V+E). Where V is the number of vertices and E is the number of edges. The acceptance randomness will make it faster because there’s a chance it will not touch all the nodes in the tree. MIMIC has O(V log E) time complexity. Per iteration, MIMIC has to find the top population percent to keep, update probability estimates and generate new sample for the next child. GA randomly selects the indices of the two parents and “reproduce” a child output. This shows the number of steps MIMIC has more than GA, explaining the significantly higher MIMIC time complexity. Since the same algorithms are used throughout the paper, the time complexity explanations will be the same.

|  |  |
| --- | --- |
| *Figure 5: Final path distances versus iteration for the 4 search algorithms* | *Figure 6: Times taken to run these algorithms on the optimized parameters.* |

## Four Peak Problem

### Problem Description

|  |  |
| --- | --- |
| *Figure 7: Sample input of the problem used to generate plots.* | This problem can be thought of as a mountain range with heights as your data. Solving the problem means finding the highest point of the mountain range. Naïve methods are known to be stuck on top of a local maximum. (i.e.: being stuck on the peak at 27 range because everything to the sides of it is negative.) |

### Results/Analysis

|  |  |
| --- | --- |
|  |  |
|  | *Figure 2-4: These plots analyzes the effects of different hyperparameters tuning available in the mlrose toolkit.*  *For GA and MIMIC, shown in Figures 2 and 3, the API allowed tuning to population size and keep percentage, for which I chose 200 to 1000 for population size and 0.1% to 0.8% for keep percentage.*  *The available parameter for SA (Figure 4) is exponential decay rate, for which we choose 0.005 to 0.105 with increments of 0.005.*  *These numbers were chosen based on having chosen a larger range and truncating it after convergence.* |
| *Figure 5: Final path distances versus iteration for the 4 search algorithms* | *Figure 6: Times taken to run these algorithms on the optimized parameters.* |

## Max K-Colors

### Problem Description

|  |  |
| --- | --- |
| *Figure 7: Sample input of the problem used to generate plots.* | The input for this problem is a list of edges, representing the country borders, generated by the number labeled vertices. The problem is trying to color each of the neighboring numbered nodes a different color while minimizing the number of colors used. |

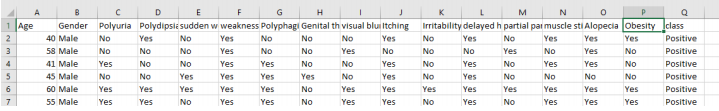
### Results/Analysis

|  |  |
| --- | --- |
|  |  |
|  | *Figure 2-4: These plots analyzes the effects of different hyperparameters tuning available in the mlrose toolkit.*  *For GA and MIMIC, shown in Figures 2 and 3, the API allowed tuning to population size and keep percentage, for which I chose 200 to 1000 for population size and 0.1% to 0.8% for keep percentage.*  *The available parameter for SA (Figure 4) is exponential decay rate, for which we choose 0.005 to 0.105 with increments of 0.005.*  *These numbers were chosen based on having chosen a larger range and truncating it after convergence.* |
|  |  |
| *Figure 5: Final path distances versus iteration for the 4 search algorithms* | *Figure 6: Times taken to run these algorithms on the optimized parameters.* |

# Part 2: Neural Net with Randomized Optimized Weights

## Diabetes Detection

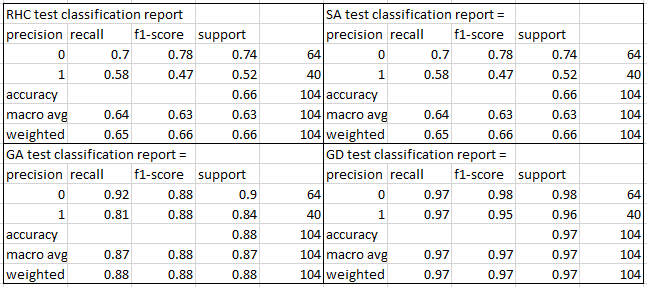
### Problem Description



This early diabetes detection dataset, pulled down from the UC Irving machine learning repository, has each row represent an individual patient. Each row describes whether they have any of the 16 characteristics of diabetes and a flag of if they were diagnosed with diabetes. Some example characteristics includes age, gender obesity, etc. There are 521 patients in this dataset. The data is mostly binary besides the age field. Our goal is to use four different neural nets with a different optimizer for its weights to train on this dataset. We will use common metrics to evaluate the machine learning algorithm techniques, such as data loss, time, and F1 Score. Here we will use the sklearn metrics toolbox’s classification\_report to generate some of these analysis.

### Results/Analysis

|  |  |
| --- | --- |
| *Figure 5: Final path distances versus iteration for the 4 search algorithms* | *Figure 6: Times taken to run these algorithms on the optimized parameters.* |



# Summary

# References:

[1] <https://github.com/ezerilli/Machine_Learning/tree/master/Randomized_Optimization>

[2] <https://www.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf>

[3] <https://stackoverflow.com/questions/9146086/time-complexity-of-genetic-algorithm#:~:text=Genetic%20Algorithms%20are%20not%20chaotic%2C%20they%20are%20stochastic.&text=Given%20the%20usual%20choices%20(point,the%20size%20of%20the%20individuals>.