October 13, 2019 (F19)
Md Khaled Hassan (ID: mhassan49)
Georgia Institute of Technology

Randomized Optimization
CS7641: Machine Learning (Assignment 02)

Abstract:

The objective of this project is to explore random search algorithms on various maximization problems and analyze the advantages of a certain algorithm over others depending on the nature of the problem. In addition, the algorithms were also applied to EEG eye state dataset that I explored in assignment 1 and carry out weight optimization and compared the accuracies to neural network analysis with back propagation that we did in assignment 1.

Introduction:

For part 1, The algorithms that we applied are as follows and their performances are analyzed in the next sections:

- Randomized Hill Climbing
- Simulated Annealing
- Genetic Algorithm
- MIMIC (Mutual Information Maximizing Input Clustering)

All these algorithms are briefly discussed below:

Randomized Hill Climbing: Randomized hill climbing is an iterative technique that instead of examining all the neighboring nodes in a local search problem, randomly selects a neighboring node based on the amount of improvement in that neighbor [1]. Depending on the amount of improvement, it may move to that neighbor or examine another [1]. The solution obtained by this algorithm may not be the global maximum [1].

Simulated Annealing: Simulated annealing is a probabilistic technique for approximating the global optimum of a given function [2]. Specifically, it is a metaheuristic to approximate global optimization in a large search space for an optimization problem [2]. The algorithm is inspired by the annealing process in

metallurgy, that involves heating and cooling to improve crystal quality and reduce defects [2].

Genetic Algorithm: Genetic algorithm is an adaptive system which is motivated by Darwin's theory of evolution [3]. Genetic algorithm can capture the structure of an optimization landscape by using an ad hoc embedding of input parameters onto chromosomes [3]. An initial population of candidate optima's are chosen randomly and allowed to evolve over several generation according to the fitness value derived from the environment [3]. Methods of evolution falls in general into selection, mutation, and crossover categories [3].

MIMIC: MIMIC is an optimization algorithm that provides faster and more reliable way to converge compared to other traditional algorithms [3]. The algorithm successively approximates the conditional distribution of the inputs given a bound of the cost function [3]. In addition, it computes second order statistics and sampling from a distribution to discover common underlying structure about optima [3].

In part 2, the dataset that I explored is available from UCI machine learning repository [4] and called *EEG (electroencephalography) eye state dataset* (Link: https://archive.ics.uci.edu/ml/datasets/EEG+Eye+State). This data set determines the eye state of a person and a binary class dataset ('1' indicates the eye being closed and a '0' indicates the eye being opened). We applied the first three algorithm (RHC, SA, and GA) on this dataset using the neural network that I optimized using this dataset in Assignment 1.

Methodology:

In this project, I have used the mlrose [5] and the Scikit -learn [6] libraries of python. 'mlrose' is a machine learning library developed in Python particularly for randomized optimization. Unless other wise specified, I have used the following parameters in the simulations carried out for this assignment: Maximum attempts=10, Population size=200, mutation probability =0.1, Maximum Iterations=1000. In addition, geometric decay function was used for all simulations of simulated annealing.

Part1: Apply Random Search Algorithms to Optimization Problems

I implemented three optimization problems: 1) Traveling with a Tesla (TWT), 2) Continuous Peak (CP), and 3) FlipFlop (FF) In order to highlight the advantages of Genetic Algorithm (GA), Simulated Annealing (SA), and MIMIC algorithms respectively. The algorithms are implemented using the mlrose python library [5].

Traveling with a Tesla (Genetic Algorithm):

This problem is inspired by the classic traveling salesman problem (TSP). In a TSP problem, for a given list of cities, the objective is to find the shortest possible route for a sales person that visits each city exactly once and returns to the origin city [7]. It is an NP hard problem and the given cities can be identified by their coordinates or distances between each pair of the cities. In the TWT problem, a tourist visiting the state of California would like to visit as many cities in CA as possible with a Tesla starting and ending at the same city. Since the tourist would like to spend as much time as possible visiting various tourist spots, he would like to make sure he travels minimum amount of distance to stop over at the Tesla superchargers to charge his electric car. Therefore, he needs to figure out the best route to cover as many cities as possible and stopping at a super charging station exactly once in each city and come back to the origin city and there by maximizing time he can spend visiting his favorite spots. I am using the list of coordinates of the Tesla supercharging stations in CA available here [8]. The number of cities used in this problem is 19. In Fig.1, I have plotted the fitness function and the computation time for various randomized algorithms against the number of iterations. From the fitness function plot, we observe that the Genetic Algorithm performs best although the computation time is also maximum. The population size was set to 1000 and the mutation probability was set to 0.1 for GA. Solution to this problem involves every possible permutation of routes that visit each city once. The genetic algorithm uses multiple local optima uses random selection rules and therefore performs very well on this type of problem.

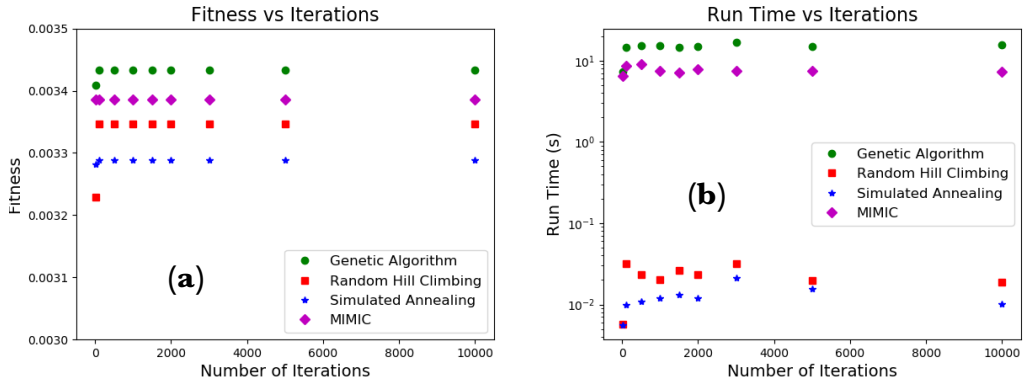


Fig.1: TWT Problem: a) Fitness vs Iterations and b) Computation time vs Iterations

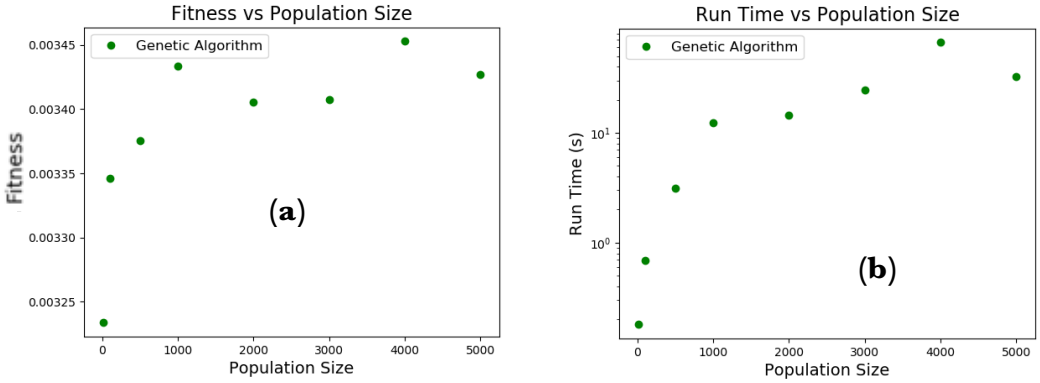


Fig.2: TWT Problem: a) GA Fitness vs Population size and b) GA Computation time vs Population Size

I further explored GA algorithm and plotted the fitness function and computation time as a function of the population size in Fig 2. We observe that, a population size of 1000 is enough for the TWT problem with 19 cities. With larger number of nodes, this algorithm may require even higher population size to reach the maximum fitness. The computation time keeps increasing with the increase in population size.

Continuous Peaks (Simulated Annealing):

The Continuous Peaks (CP) problem is the extension of the classic four peaks problems. In this optimization problem the landscape is filled with multiple peaks and the problem tries to find out the global optima (highest peak) with respect to the other peaks. In Python's mlrose library,

the threshold parameter (T) is expressed as a percentage of the state space dimension n ($T=t_{pct}\ x\ n$) [1]. For this optimization problem, we set t_{pct}=0.15 and n=5. We observe in Fig.3 that simulated annealing works best for this optimization problem iteration numbers 200 or higher. In terms of computation time, only RHC beats simulated annealing. However, RHC show worst fitness among all these four algorithms. Therefore, SA is the best performing algorithm for this problem. The SA performs well when the problem space is simple and statistically ensures finding an optimal solution. The algorithm seems to be a good fit to find solutions to this Continuous peaks optimization problem.

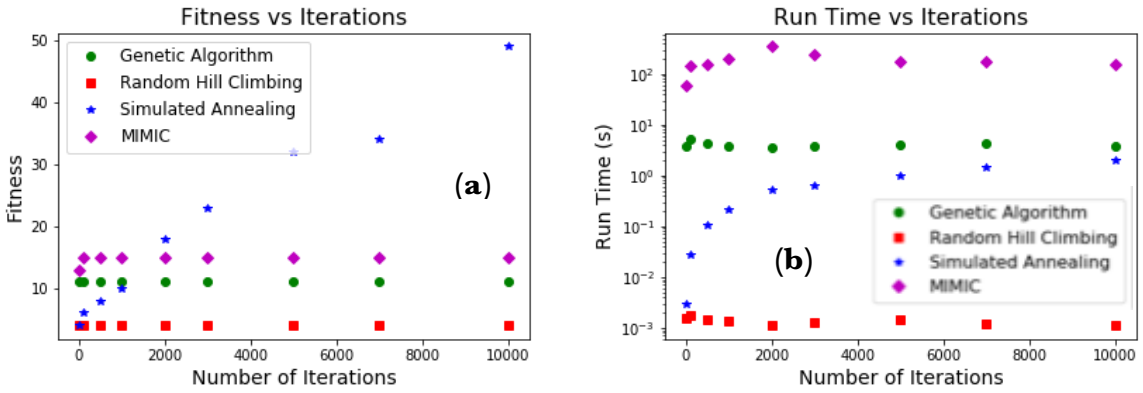


Fig.3: CP Problem: a) Fitness vs Iterations and b) Computation time vs Iterations

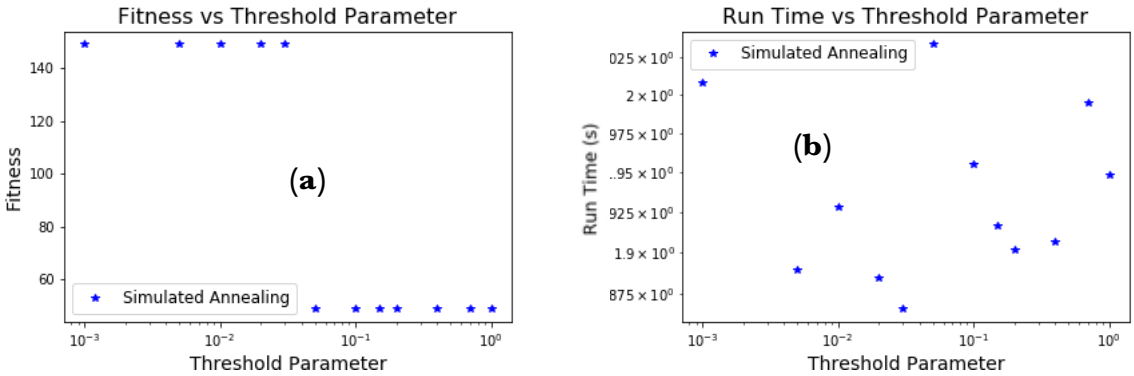


Fig.4: CP Problem: a) SA Fitness vs t_{cpt} and b) SA Computation time vs t_{cpt}

The SA algorithm is further explored for the CP problem. I plotted the fitness and run-time as a function of the threshold percentage parameter (t_{cpt}) in Fig.4. We observe that the fitness function can be improved significantly with lower percentage value (0.03 or less). I believe this is because at a very low

temperature, the algorithm behaves more like hill climbing and taking steps to improve the fitness function [9]. We do not observe any particular trend in the computation time.

Flip-Flop (MIMIC):

The Flip-Flop (FF) problem evaluates the fitness of a state vector x as the total number of pairs of consecutive elements of that state vector [5]. From the fitness function plot in Fig. 5, we observe that MIMIC does best among all the algorithms. Although the computation time is worst, MIMIC maximizes the fitness function in least number of iterations. Therefore, MIMIC is the best performing algorithm for this problem. The population size was set as 200 and the proportion of 'samples to keep' in each iteration *keep_pct* was set at 0.2. In this maximization problem, we are trying to maximize the alterations where every bit is different from its neighbor. MIMIC does best when the problem space involves this type structure and relationship between the consecutive neighbors [9]. Unlike MIMIC, the other algorithms do not learn about the optimization space itself to use that information to be more effective [9].

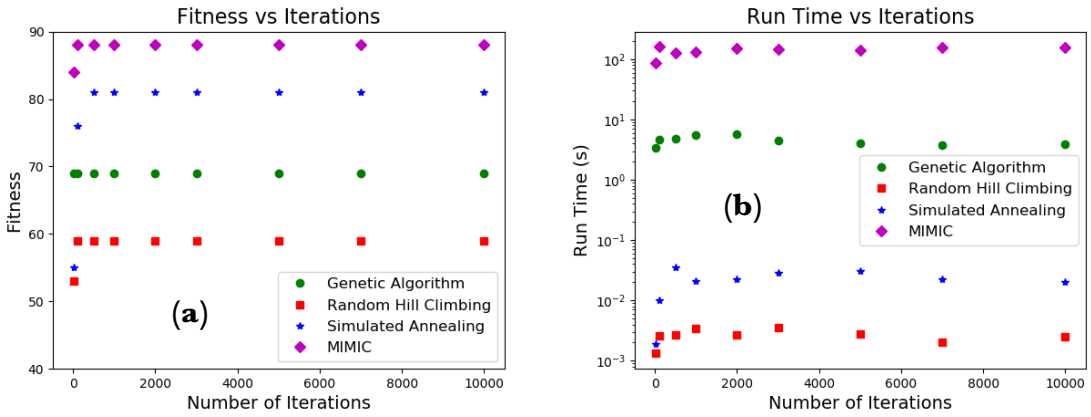


Fig5: FF Problem: a) Fitness vs Iterations and b) Computation time vs Iterations

In Fig. 6, we further explored the performance of the MIMIC algorithm on the FF problem. We observe that, increasing the keep percentage to 40% can further improve the fitness and drastically reduces when set to above 60%. The computation time shows similar trend indicating that at a very high keep percentage, the MIMIC stops looking for a better solution and therefore computation time also decreases.

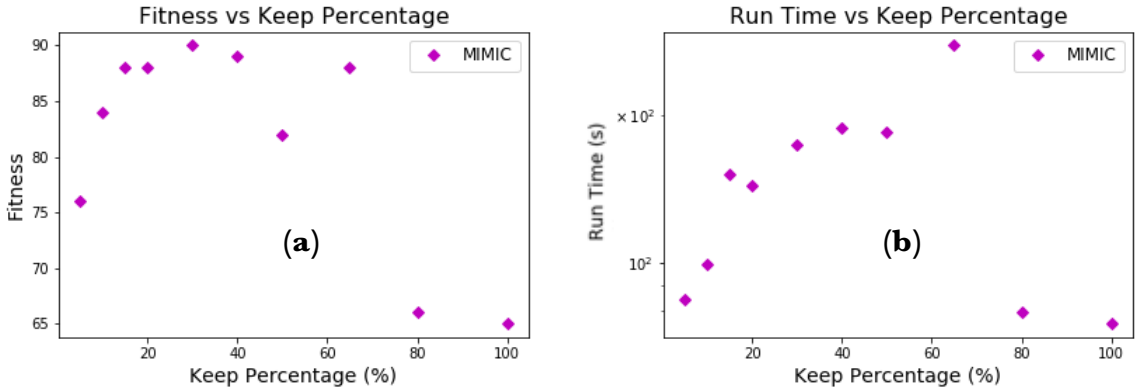


Fig.6: FF Problem: a) MIMIC Fitness vs Keep Percentage (*keep_pct*) and b) MIMIC Computation time vs Keep Percentage (*keep_pct*)

Part 2: Weight Optimization for a Neural Network

Among the two datasets I explored in assignment 1, I have picked up the EEG eye state dataset that is available from the UCI repository [4]. We have explored the random hill climbing (RHC), simulated annealing (SA), and the genetic algorithm (GA) for this part of the assignment and compared them with the Neural Network that we optimized in assignment 1. For this analysis, I have used 75% of data for training and the rest 25% for test purpose.

Weight Optimization with EEG eye state dataset:

We have used Multilayer Perceptron (MLP) classifier for neural network analysis in assignment 1 and did grid search with10 fold cross validation to optimize parameters to achieve maximum CV-10 score. The best accuracy was achieved with the 'tanh' activation function and with a hidden layer size of 250. The best test accuracy using back propagation that was obtained is ~63.5%. In this assignment was have explored optimization with RHC, SA, and GA algorithms. From. Fig.7, we observe that both test and train accuracies of these algorithms are almost insensitive to the number of iterations. Only GA shows somewhat dependency to the number of iterations when the number is less than 10. In addition, GA is most computationally expensive among all these algorithms although for high number of iterations (500 or more), MIMIC also shows similar training time. All the algorithms show worse performance compared to the neural network with back propagation.

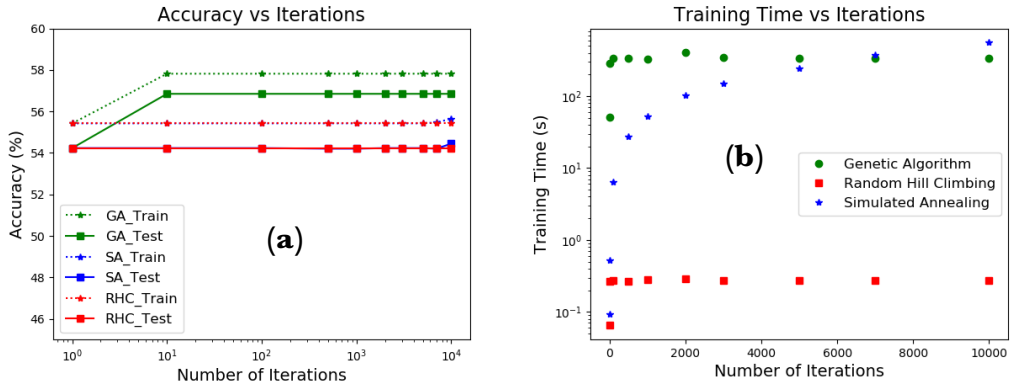


Fig7: EEG Eye state dataset a) Test and Train Accuracy vs Iterations and b)Training time vs Iterations

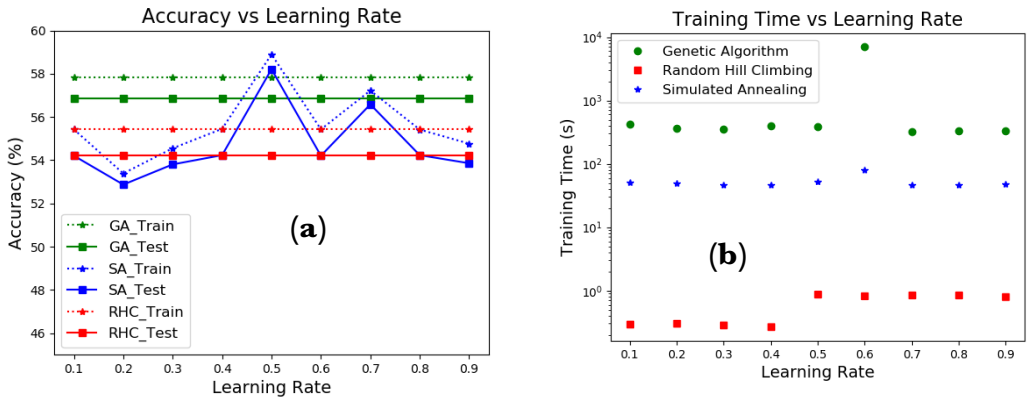


Fig.8: EEG Eye state dataset a) Test and Train Accuracy vs Learning Rate and b)Training time vs Learning Rate

To further explore the sensitivity of these algorithms, I plotted the accuracies and fitting time as a function of the learning rate. We observe from Fig. 8 that the SA algorithm shows very interesting variation is accuracies although the other algorithms are insensitive to the learning rate. The training time of these algorithms are also flat (constant) when plotted as a function of the learning rate. I am not sure how to explain the behavior of SA in this plot. However, from Fig 7, 8, we can conclude that GA performs best among these three algorithms and therefore, it can be assumed that the dataset is very noise and that could be the reason for the unexplained behavior of SA algorithm in Fig. 8.

The charts 1 and 2 shows the ranking of these algorithms compared to the Neural Network (NN) that we optimized in Assignment 1. The rankings are provided based on the maximum accuracies for each of the algorithms extracted from Fig.7 and 8, respectively and compared with the best accuracy obtained for the NN (MLP classifier) in assignment 1. In both cases (as a function of number of maximum iterations and learning rate), the NN with back propagation shows best performance. Among the randomized algorithms, SA and GA performs better than RHC. I believe, the data may have bias and as a result of that, may not be suitable for neural network analysis.

Chart 01. Algorithm Rank (Function of Maximum Iterations)

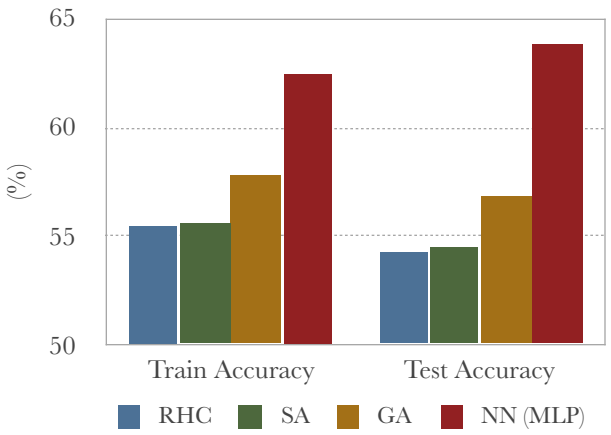
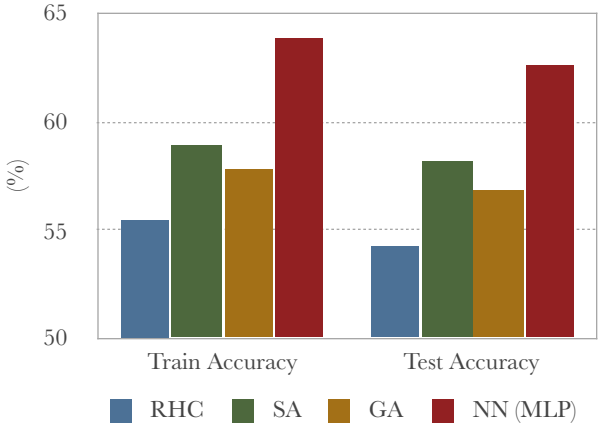


Chart 02. Algorithm Rank (Function of Learning Rate)



References:

1. https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/
2. https://en.wikipedia.org/wiki/Simulated_annealing
3. Randomized Local Search as Successive Estimation of Probability Densities, Charles Isbell, 2003 (https://www.cc.gatech.edu/~isbell/classes/2003/cs4600_fall/docs/mimic-tutorial.pdf)
4. UCI repository (https://archive.ics.uci.edu/ml/datasets.php)
5. Hayes, G. (2019). mlrose: Machine Learning, Randomized Optimization and SEarch package for Python. https://github.com/gkhayes/mlrose. Accessed: 10/2019
6. Scikitlearn (https://scikit-learn.org/stable/index.html)
7. https://en.wikipedia.org/wiki/Travelling_salesman_problem
8. https://teslamotorsclub.com/tmc/threads/a-list-of-all-supercharger-coordinates-and-a-script-to-grab-them.37948/
9. Udacity lectures of CS7641