CS 7641 Assignment 2: Randomized Optimization

Abstract

The purpose of this assignment is to explore randomized optimization. This assignment consists of applying 4 randomized optimization algorithms namely Randomized Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA) and Mutual-Information-Maximizing Input Clustering (MIMIC) to 3 discrete optimization problem domains. These algorithms (except MIMIC) are also applied to 1 continuous optimization problem domain namely finding optimum weights for a Neural Network. Results from each of these algorithms for every problem domain are compared & contrasted. Their comprehensive summary & recommendations are also presented.

Introduction

The objective of randomized optimization algorithms is to find a 'X' that approximates global optimum with respect to some fitness function 'f'. In this assignment, 4 such randomized optimization algorithms, namely Randomized Hill Climbing (RHC), Genetic Algorithm (GA), Simulated Annealing (SA) and Mutual-Information-Maximizing Input Clustering (MIMIC), are studied.

Randomized Hill Climbing (RHC)

Hill climbing is a mathematical optimization technique which belongs to the family of local search. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by making an incremental change to the solution. If the change produces a better solution, another incremental change is made to the new solution, and so on until no further improvements can be found [1]. RHC runs hill climbing multiple times, each time it starts with a random initial state and finds local optima. Best amongst these local optima is returned by RHC and approximates global optima.

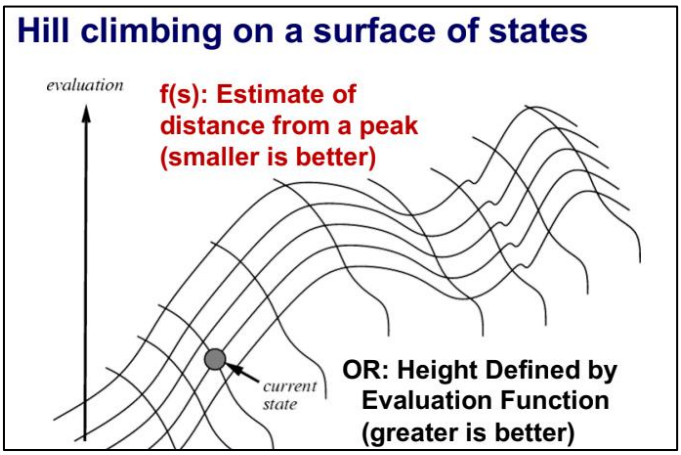

Figure 1: Hill Climbing

Genetic Algorithm (GA)

A genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection [2]. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction to produce offspring of the next generation.

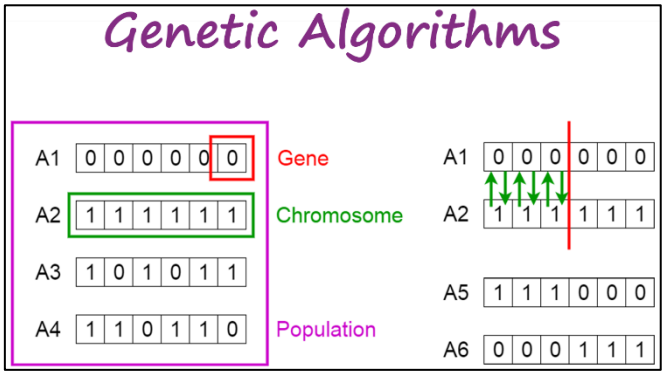

Figure 2: Genetic Algorithm

Simulated Annealing (SA)

Simulated annealing is a probabilistic technique for approximating the global optimum of a given function [3]. Annealing is the process of a metal cooling and freezing into a minimum-energy crystalline structure. SA mimics this process by occasionally accepting a decreased fitness function, so it has a chance to escape from local optima. One of its parameter "T" is the temperature. The higher the temperature, the higher probability of accepting a worse solution. Typically, T decreases as the algorithm runs longer. Theoretically this algorithm always finds the global optimum but it can run very slow for some problems and in practice it would be a problem as to how to decide the rate at which to decrease T [4].

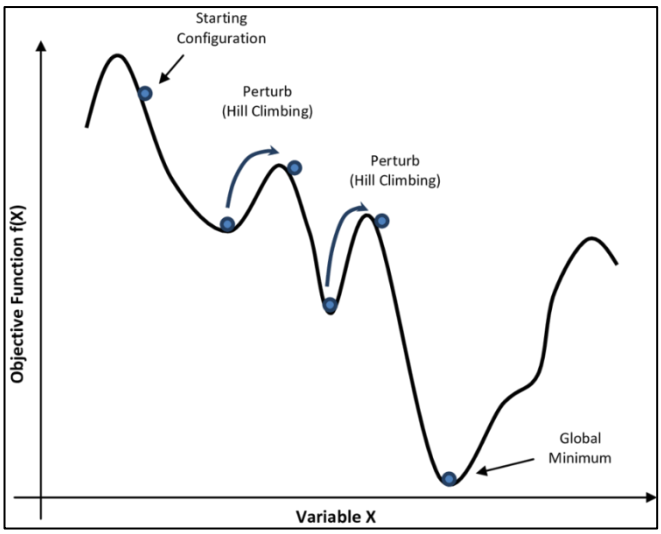

Figure 3: Simulated Annealing

Mutual-Information-Maximizing Input Clustering (MIMIC)

MIMIC is one of the estimation of distribution algorithms (EDA) that belong to the larger class of evolutionary algorithms (EA) [5]. MIMIC analyzes the global structure of the optimization landscape and use the knowledge of this structure to guide a randomized search through the solution space and, in turn, to refine the estimate of the structure [6].

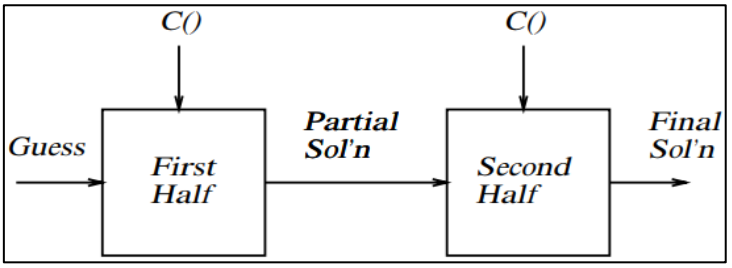

Figure 4: MIMIC

Implementation

The assignment was implemented in Python programming language and mlrose library [7] was used with a few modifications for running randomized optimization algorithms.

Results

Each algorithm under consideration is run 3 (N=3) times on a given problem for different values for corresponding input parameters. For RHC, number of restarts is used as input parameter and was set to 10 (Restarts=10). For GA, mutation probability with values 0.01, 0.05, 0.1, 0.15, and 0.2 (Mutation Prob = [0.01, 0.05, 0.1, 0.15, 0.2]) was used as a variable parameter. Similarly for SA and MIMIC, geometric decay and keep fraction with values 0.9, 0.92, 0.94, 0.96 & 0.98 (Geom Decay = [0.01, 0.05, 0.1, 0.15, and 0.2]) and 0.01, 0.05, 0.1, 0.15, & 0.2 (Keep Pct = [0.01, 0.05, 0.1, 0.15, and 0.2]), respectively, was used an input parameter. Termination of SA, GA and MIMIC is defined as inability to find better solution after 100 (M=100) attempts.

Mean and standard deviation of the fitness function values obtained from N=3 runs is plotted against number of iterations to obtain graphs analogous to learning curves in supervised learning.

Finally, for every configuration of all algorithms "best" fitness function value and execution time for that configuration is plotted to aid comprehensive comparison.

Continuous Peaks Problem (CPP)

CCP contains multiple local optima and fitness function for Continuous Peaks problem is defined as follows [7]:

$$Fitness(x, T) = \max(max\_run(0, x), max\_run(1, x)) + R(x, T)$$

where:

- x is n-dimensional state vector
- $T = t_{pct} \times n$ is threshold parameter
- t_{pct} is a real number between 0 and 1
- $max\_run(b, x)$ is the length of the maximum run of b's in x
- $R(x, T) = n$, if ($max\_run(0, x) > T$ and $max\_run(1, x) > T$); and
- $R(x, T) = 0$, otherwise

Following results are obtained from experiments where, n = 64 and t_{pct} = 0.1.

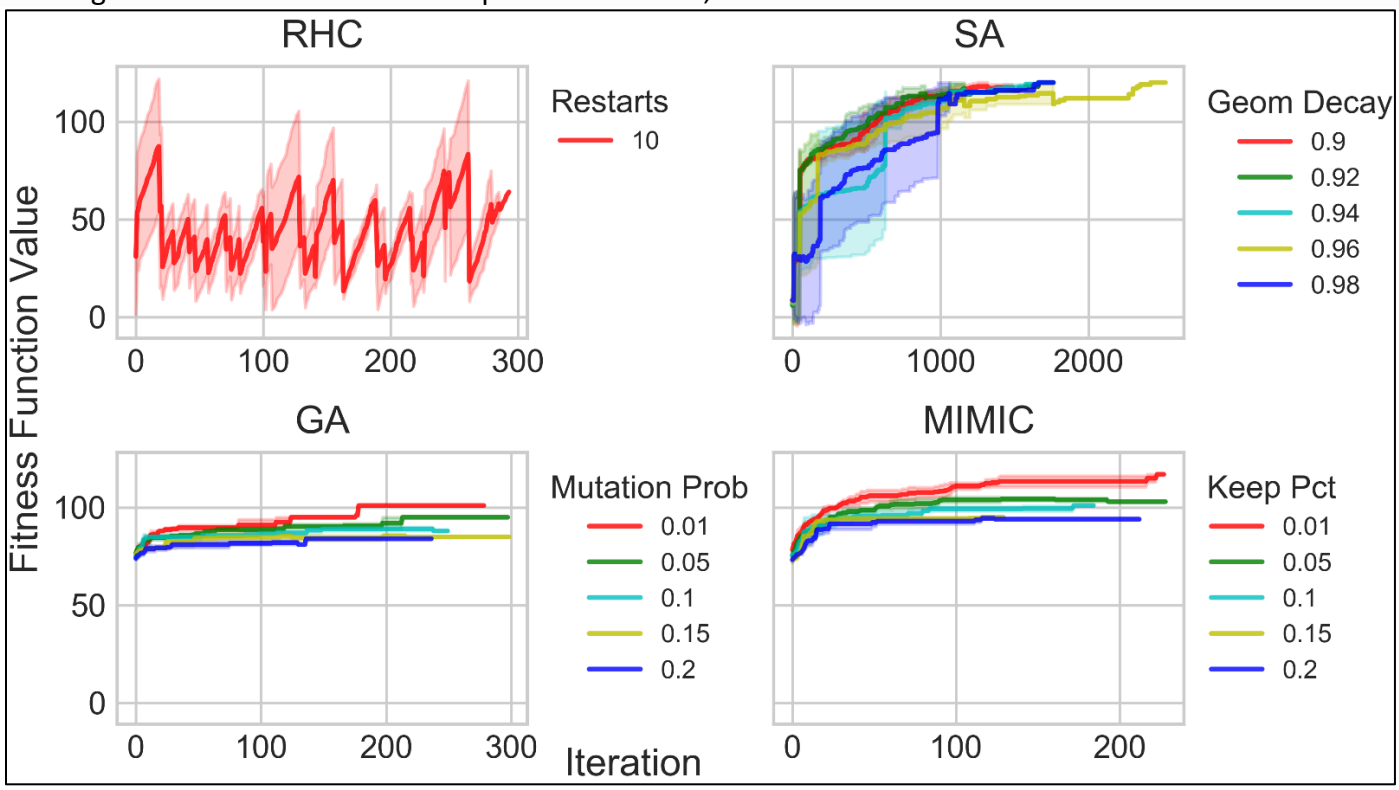

Figure 5: [CCP] Fitness function values obtained via (a) RHC algorithm with 10 restarts, (b) various mutate probabilities of GA algorithm, (c) various geometric decay of SA algorithm, and (d) various keep fractions of MIMIC algorithm. N=3, M=100.

Following graphs better compare the model performance and execution time:

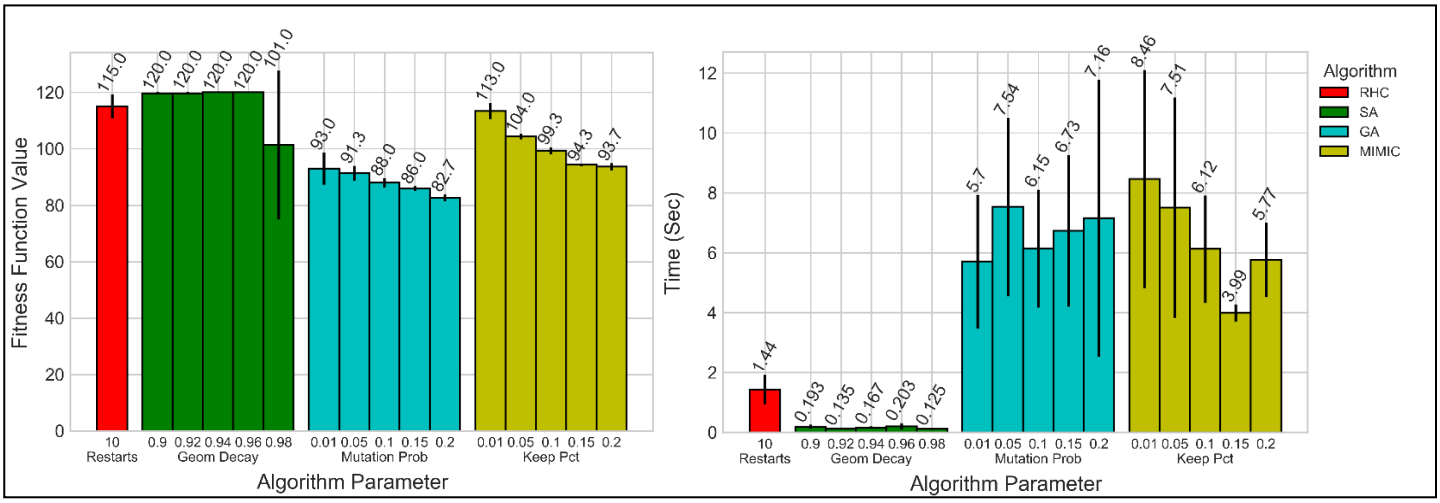

Figure 6: [CCP] (a) Fitness function values, and (b) execution time for all algorithms and corresponding parameters. N=3, M=100.

Following observations can be made from Figures 5 and 6:

- RHC has the expected zig-zag pattern. As every time it restarts at a random state, the fitness function value tends to become less optimal, it then hill climbs to find a local optima from that position. Multiple restarts help RHC better explore the fitness landscape and generate a near optimal result. Each hill climbing takes ~30 iterations on average to find the local optima. Total number of iterations for RHC is ~300. RHC is fast, but not as fast as SA because during each iteration it has to find the best local step to move uphill. Time required for RHC is directly proportional to number of available local moves from each step, i.e. length of vector (n) in this case.
- GA does not do as well as the other 3 algorithms for this optimization problem. This could be because GA has tendency to get stuck in local optima as it does not know "how" to sacrifice short-term fitness to gain longer-term fitness [2]. Best values from GA are obtained when mutation probability is set to 1%. For each configuration it run for ~300 iterations before termination. Moreover, GA takes longer to run as it has to perform a decent chunk of work in each iteration to find "fittest" parents, perform crossover & mutation. Execution time of GA is directly proportional to the size of population and the size of state vector, i.e. length of vector (n) in this case.
- As expected, SA is not monotonically increasing. Occasionally, it navigates to a state of lower fitness function values but overall it has a new upwards trend. SA does know "how" to sacrifice short-term fitness to gain longer-term fitness, especially in the beginning when the temperature is high. SA converges and gives the global optimal of 120 for this problem setup for different values (0.9, 0.92, 0.94, 0.96) of the geometric decay parameter. On average SA takes ~2000 iterations to terminate. In practice, SA is very quick, as amount of work done in every iteration is very less because it has to only look at one of the neighboring states and decide whether to take it or not. Run time of SA is dependent on the decay parameter and complexity of the fitness landscape.
- Lastly, MIMIC performs slightly better than GA but not as good as RHC or SA. This could be because there is no real physical structure that we are trying to optimize for, and given that there are multiple local optima. Best parameter for MIMIC was found to be 1% keep percentile. On average, MIMIC takes ~200 iterations to terminate. In practice MIMIC takes longer to run, as is does a lot of work in every iteration to estimate probability distribution and generate samples from that probability distribution. Execution time of MIMIC is directly proportional to the size of population an of the size of state vector, i.e. length of vector (n) in this case.

The results show that SA is best suited for Continuous Peak Problem as it finds global optima in minimum time. Closely followed by RHC as it finds near-optimal solution in slightly more time. Execution time for GA and MIMIC is many folds that of the other two, also they produce less optimal solution and hence not well suited for this problem.

K-Colorable Problem (KCP)

In graph theory, graph coloring is a special case of graph labeling; it is an assignment of labels traditionally called "colors" to elements of a graph subject to certain constraints. In its simplest form, it is a way of coloring the vertices of a graph such that no two adjacent vertices are of the same color; this is called a vertex coloring. A coloring using at most k colors is called a (proper) k-coloring [8]. K-coloring problem can be modeled as minimization problem such that the number of adjacent vertices with same color be minimized.

Following results are obtained from experiments for 2-coloring problem (2CP) from a graph with 64 (n=64) vertices connected uniformly at random with 1024 edges.

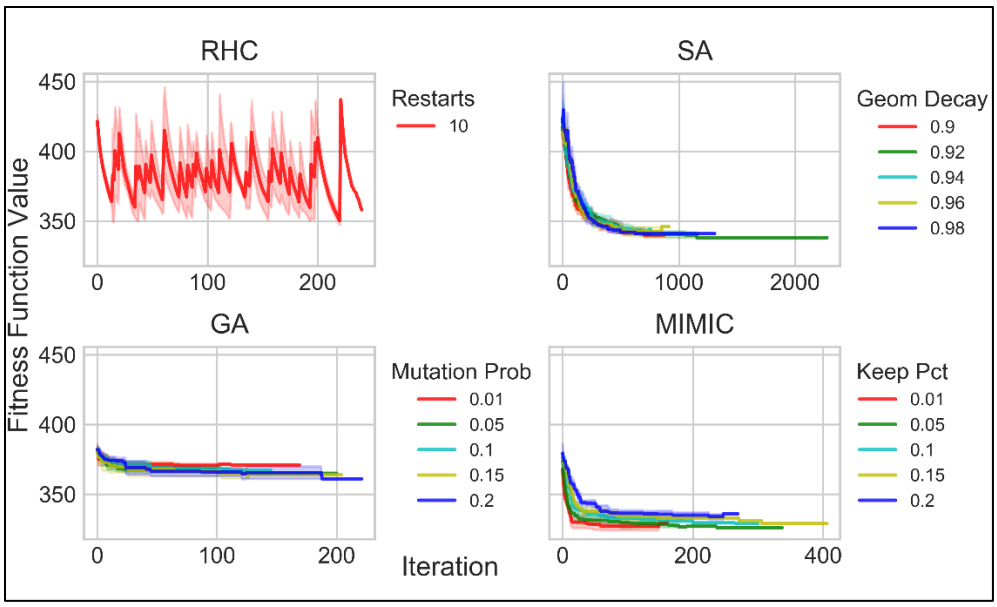

Figure 7: [2CP] Fitness function values obtained via (a) RHC algorithm with 10 restarts, (b) various mutate probabilities of GA algorithm, (c) various geometric decay of SA algorithm, and (d) various keep fractions of MIMIC algorithm. N=3, M=100.

Following graphs better compare the model performance and execution time:

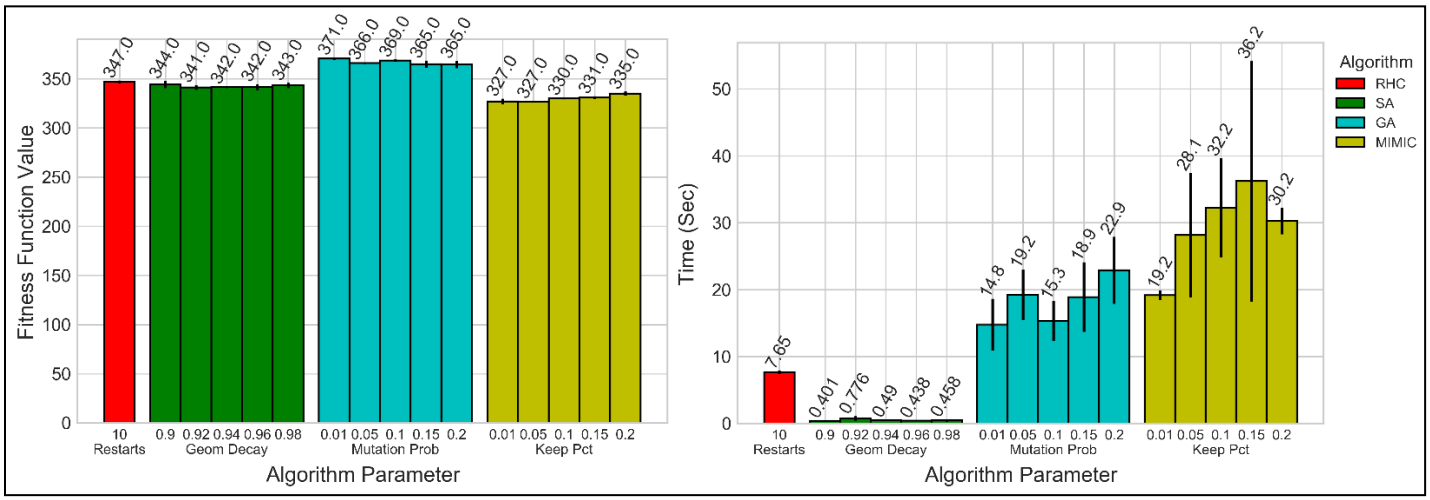

Figure 8: [2CP] (a) Fitness function values, and (b) execution time for all algorithms and corresponding parameters. N=3, M=100.

Following observations can be made from Figures 7 and 8:

- RHC has the expected zig-zag pattern. As every time it restarts at a random state, the fitness function value tends to become less optimal, it then hill climbs to find a local optima from that position. Multiple restarts help RHC better explore the fitness landscape. Each hill climbing takes ~25 iterations on average to find the local optima. Total number of iterations for RHC is ~250. RHC is fast, but not as fast as SA because during each iteration it has to find the best local step to move uphill. Time required for RHC is directly proportional to number of available local moves from each step, i.e. number of vertices (n) in this case.
- GA does not do as well as the other 3 algorithms for this optimization problem. This could be because GA has tendency to get stuck in local optima as it does not know "how" to sacrifice short-term fitness to gain longer-term fitness [2]. Best values from GA are obtained when mutation probability is set to 20%. For each configuration it run for ~200 iterations before termination. Moreover, GA takes longer to run as it has to perform a decent chunk of work in each iteration to find "fittest" parents, perform crossover & mutation. Execution time of GA is directly proportional to the size of population and the size of state vector, i.e. number of vertices (n) in this case..

- As expected, SA is not monotonically decreasing. Occasionally, it navigates to a state of lower fitness function values but overall it has a new downwards trend. SA does know "how" to sacrifice short-term fitness to gain longer-term fitness, especially in the beginning when the temperature is high. SA gives the 2-nd best solution for this problem setup for the geometric decay parameter value of 0.92. On average SA takes ~1500 iterations to terminate. In practice, SA is very quick, as amount of work done in every iteration is very less because it has to only look at one of the neighboring states and decide whether to take it or not. Run time of SA is dependent on the decay parameter and complexity of the fitness landscape.
- Lastly, MIMIC gives most optimal solution for this problem setup. This could be because there is a real physical structure (bi-partite graph) that we are trying to optimize for, and MIMIC excels at such problems. Best parameter for MIMIC was found to be 5% keep percentile. On average, MIMIC takes ~400 iterations to terminate. But in practice MIMIC takes longer to run, as is does a lot of work in every iteration to estimate probability distribution and generate samples from that probability distribution. Execution time of MIMIC is directly proportional to the size of population an of the size of state vector, i.e. length of vector (n) in this case.

The results show that MIMIC gives the best solution for the 2-Coloring Problem though it takes the maximum time to execute. If the optimal solution is crucial and there is time to trade, MIMIC is the best bet for this problem setup. SA is the 2-nd best bet as it gives near-optimal solution in a fraction of time. Solution found by RHC is not as good as MIMIC or SA and it takes more time than SA, so would not be good candidate for this setup. Execution time for GA is large, also it produces less optimal solution and hence not well suited for this problem.

Travelling Salesman Problem (TSP)

The travelling 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 exactly once and returns to the origin city?" It is an NP-hard problem in combinatorial optimization, important in theoretical computer science and operations research. TSP can be modelled as a graph problem, with the cities being the graph's vertices, paths being the graph's edges, and a path's distance being the edge's weight. It is a minimization problem starting from a specified vertex and finishing at the same vertex after visiting all other vertices exactly once [9]. Following results are obtained from experiments with 32 cities (n=32) placed uniformly at random in 100 X 100 space.

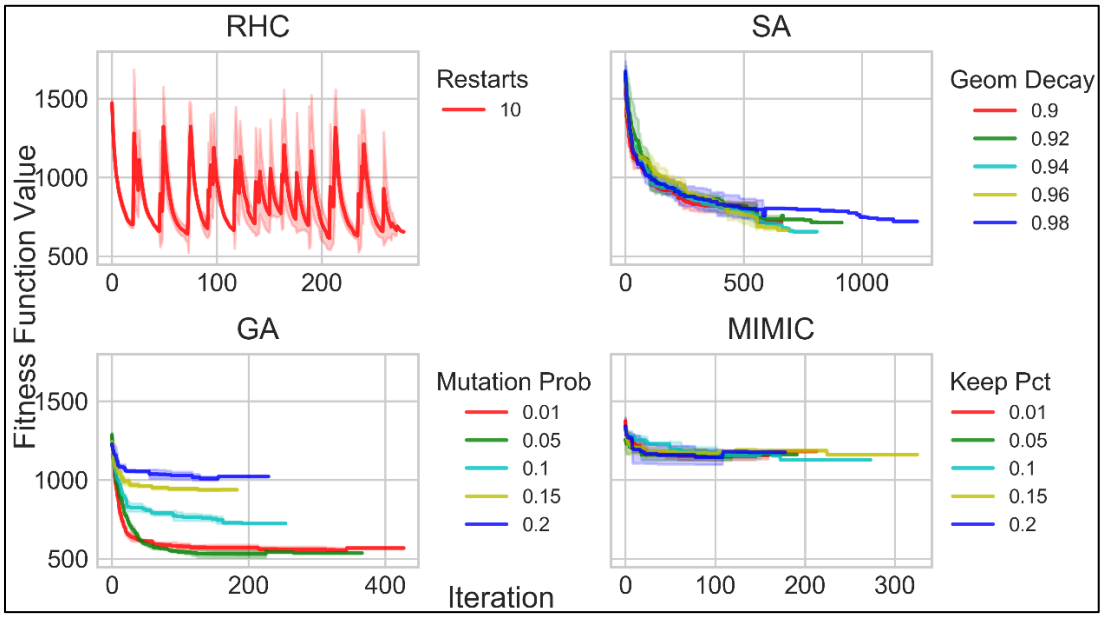

Figure 9: [TSP] Fitness function values obtained via (a) RHC algorithm with 10 restarts, (b) various mutate probabilities of GA algorithm, (c) various geometric decay of SA algorithm, and (d) various keep fractions of MIMIC algorithm. N=3, M=100.

Following graphs better compare the model performance and execution time:

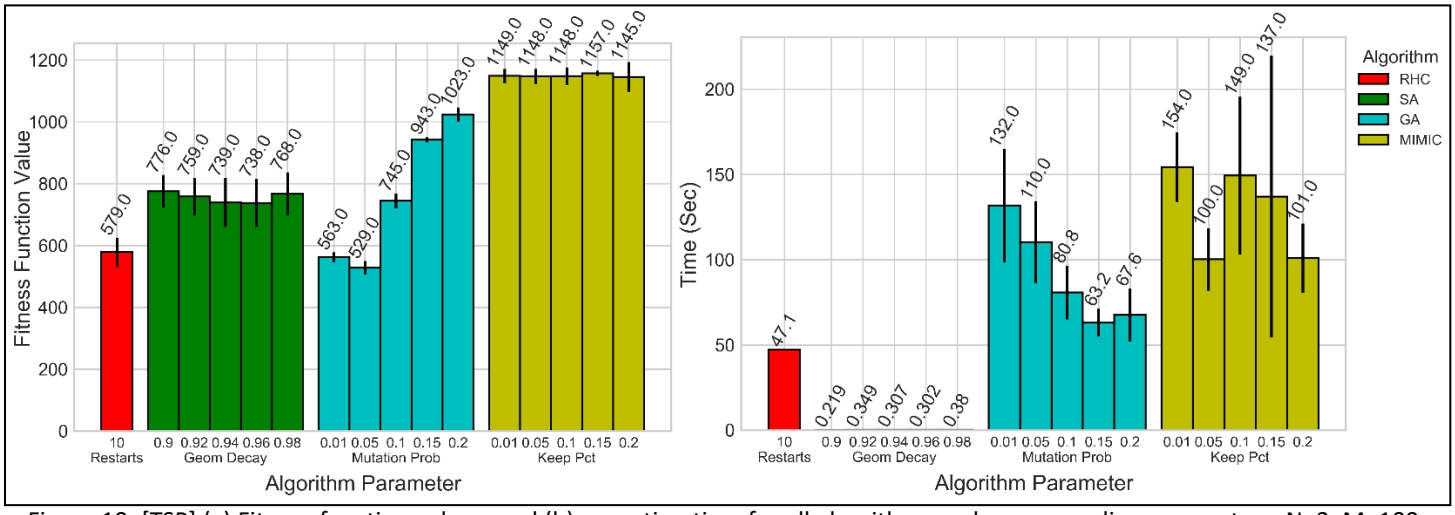

Figure 10: [TSP] (a) Fitness function values, and (b) execution time for all algorithms and corresponding parameters. N=3, M=100.

Following observations can be made from Figures 9 and 10:

- RHC has the expected zig-zag pattern. As every time it restarts at a random state, the fitness function value tends to become less optimal, it then hill climbs to find a local optima from that position. Multiple restarts help RHC better explore the fitness landscape and generate a near optimal result. Each hill climbing takes ~30 iterations on average to find the local optima. Total number of iterations for RHC is ~300. RHC is fast, but not as fast as SA because during each iteration it has to find the best local step to move uphill. Time required for RHC is directly proportional to number of available local moves from each step, i.e. number of cities (n) in this case.
- GA gives the most optimal solution for this problem setup. This is because GA is one of the best known optimization techniques for TSP [10]. Best values from GA are obtained when mutation probability is set to 5%. For each configuration it run for ~400 iterations before termination. Moreover, GA takes longer to run as it has to perform a decent chunk of work in each iteration to find "fittest" parents, perform crossover & mutation. Execution time of GA is directly proportional to the size of population and the size of state vector, i.e. number of cities (n) in this case.
- As expected, SA is not monotonically decreasing. Occasionally, it navigates to a state of lower fitness function values but overall it has a new downward trend. SA knows "how" to sacrifice short-term fitness to gain longer-term fitness, especially in the beginning when the temperature is high. SA does not give as good result as GA or RHC for this setup. Best solution from SA is found when the geometric decay parameter is set to 0.96. On average SA takes ~1000 iterations to terminate. In practice, SA is very quick, as amount of work done in every iteration is very less because it has to only look at one of the neighboring states and decide whether to take it or not. Run time of SA is dependent on the decay parameter and complexity of the fitness landscape.
- Lastly, MIMIC performs the worst both in terms of fitness values and time. This could be because there is no real physical structure that we are trying to optimize for. Best parameter for MIMIC was found to be 20% keep percentile. On average, MIMIC takes ~300 iterations to terminate. In practice MIMIC takes longer to run, as is does a lot of work in every iteration to estimate probability distribution and generate samples from that probability distribution. Execution time of MIMIC is directly proportional to the size of population an of the size of state vector, i.e. number of cities (n) in this case.

The results show that GA gives the best solution for the Travelling Salesman Problem though it takes ~twice as much time as the 2nd best algorithm, which is RHC. SA takes the least amount of time, but the output produced by it is far inferior to that by GA or RHC. MIMIC performs the worst both in terms of fitness values and time and hence not well suited for this problem.

Neural Network Weights Optimization Problem (NNWOP)

For several different machine learning models, the process of fitting the model parameters involves finding the parameter values that minimize a pre-specified loss function for a given training dataset. Neural network is one such model that typically use gradient decent (via Back Propagation) to find optimal weights of model parameters. However, the problem of fitting the parameters (or weights) of neural network can also be viewed as a continuous-state optimization problem, where the loss function takes the role of the fitness function, and the goal is to minimize this function. Moreover, this loss function is convex. By framing the problem this way, we can use any of the randomized optimization algorithms that are suited for continuous-state optimization problems to fit the model parameters.

The dataset used in this problem is binarized version of data used in a study to predict average wind speed (in knots) at one (MAL) of the 12 (RPT, VAL, ROS, KIL, SHA, BIR, DUB, CLA, MUL, CLO, BEL and MAL) synoptic meteorological stations located in Ireland, during the period 1961-78 [11], [12]. The dataset is binarized by computing the mean and classifying all instances with a lower target value as positive ('P') and all others as negative ('N') [13]. The aim of this study was to help the Irish government decide regarding the use of wind energy to meet a significant portion of Ireland's energy needs.

The dataset contains 6574 instances of which 3501 are positive and 3073 are negative. Each instance contains 14 features and corresponds to one day of data in the following format: year, month, day, average wind speed (in knots) at each of the 11 stations (RPT, VAL, ROS, KIL, SHA, BIR, DUB, CLA, MUL, CLO and BEL). The target variable is binarized version of average speed at MAL station. The data does not contain any null values making our job little easy. Figure 11 shows attribute-wise distribution on the dataset grouped by the binarized target. At first glance, it can be seen that the average speed at each of the feature station is normally distributed with 2 separate peaks, one each for positive and negative classes. Such a distribution generally results in good classification. Additionally, it can be seen from month histograms that there is seasonal variation in the wind speed with more higher-average-speeds observed in winters (Oct-Mar) than in summer (Apr-Sept).

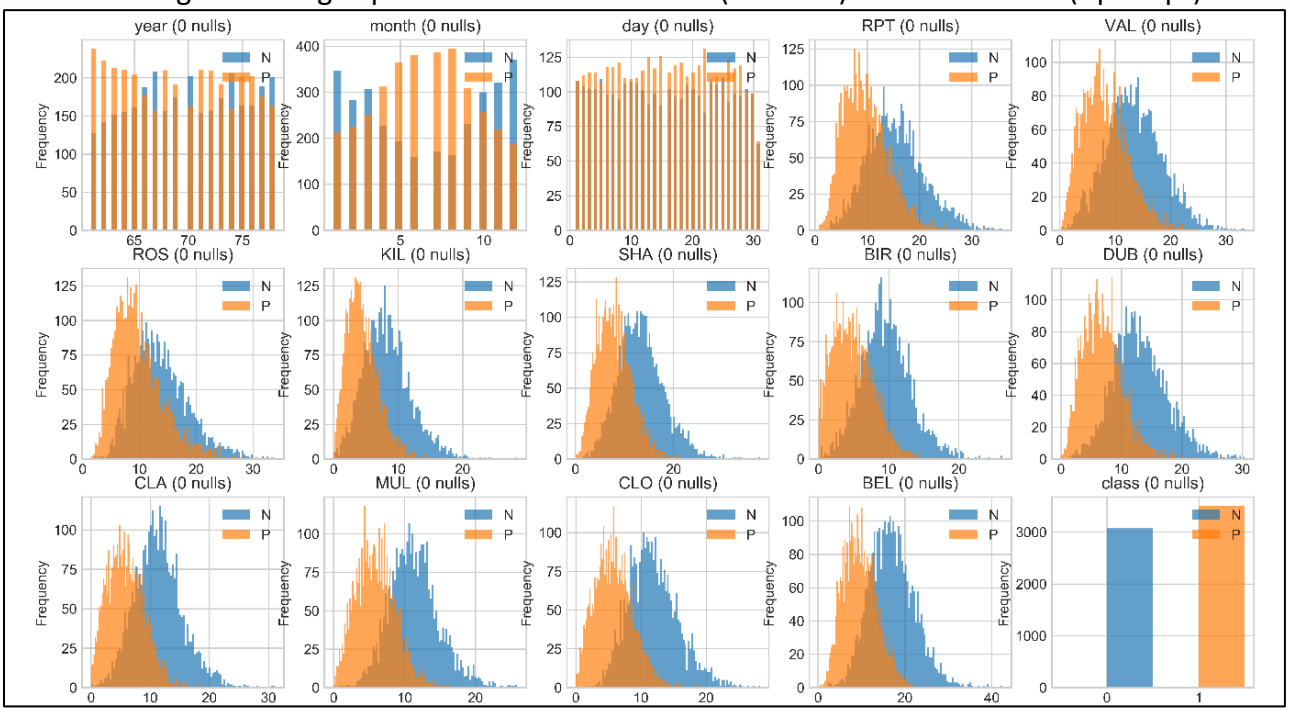

Figure 11: [NNWOP] Attribute-wise distribution of wind speed dataset [Blue : high quality], [Red : low quality]

In assignment#1, best results from neural network for this wind speed dataset was obtained with 1 hidden layer with 6 hidden units. Following results were obtained with the same setup, which gives (14+1) X 6 + (6 + 1) X 1 = 97 real valued parameters to optimize. To optimize these parameters, step size was set to 0.5.

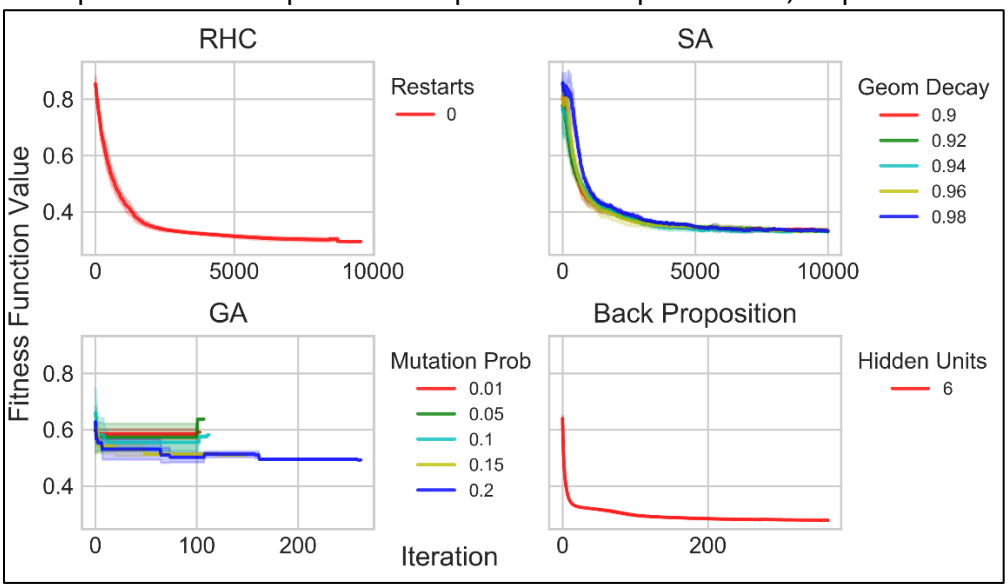

Figure 12: [NNWOP] Fitness function values obtained via (a) RHC algorithm with 10 restarts, (b) various mutate probabilities of GA algorithm, (c) various geometric decay of SA algorithm, and (d) various keep fractions of MIMIC algorithm. N=3, M=100.

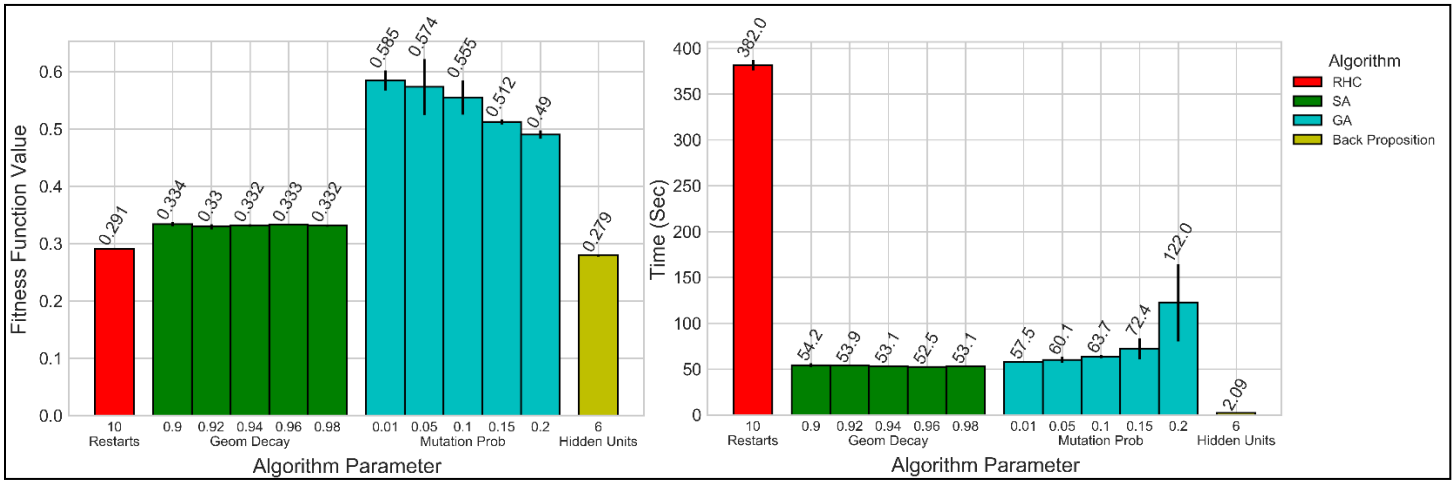

Figure 13: [NNWOP] (a) Fitness function values, and (b) execution time for all algorithms and corresponding parameters. N=3, M=100.

Following observations can be made from Figures 12 and 13:

- As the fitness function is convex, random restart in RHC would not add any value and hence was not performed. Hill climbing to find the global optima was capped at 10000 iterations and returned a near optimal value of the fitness function. RHC runs the slowest for this setup, as there are 970 neighbors and for each neighbor fitness values has to be computed, which is expensive. RHC run time is directly proportional to number of available local moves from each step.
- GA performs the worst in terms of the fitness function value and 2-nd worst with respect to the execution time. This could be because GA not very good and convex optimization problems [2]. Best values from GA are obtained when mutation probability is set to 20%. For each configuration it run for ~100 iterations before termination without learning much. Moreover, GA takes longer to run as it has to perform a decent chunk of work in each iteration to find “fittest” parents, perform crossover & mutation. Execution time of GA is directly proportional to the size of population and the size of state vector.
- Unlike for other problems, in this problem SA results in an almost monotonically decreasing curve. This could again be due to the convex nature of the optimization problem. SA was allowed to run for 10000

iterations but failed to converge by then, indicating it could have learnt more given more iterations. It gives good result but not as good as RHC or Back Propagation. Best solution from SA is found when the geometric decay parameter is set to 0.92. SA was not as quick for this problem as for the other ones, this could be because; one, it ran for 10000 iterations and two, at each iteration it has to calculate the fitness value, which is expensive. Run time of SA is dependent on the decay parameter and complexity of the fitness landscape.

- Lastly, as a benchmark Back Propagation (BP) was also used to compute weights for the neural network. Unsurprisingly, it performed the best both in term of fitness function value and execution time.

However, finding the optimal model weights is merely a means to an end. We want to find the optimal model weights so that we can use our fitted model to predict the labels of future observations as accurately as possible and not because we are actually interested in knowing the optimal weight values. Therefore, to get a true handle on the model performance various evaluation metrics are computed on unseen data for the best parameter configuration for each algorithm and displayed in Table 1.

	auc	f1	accuracy	precision	recall
RCH	0.850656	0.855934	0.850837	0.856569	0.855300
SA	0.842933	0.849854	0.843346	0.846154	0.853587
GA	0.796163	0.797896	0.795437	0.819444	0.777452
BP	0.858838	0.865455	0.859316	0.859827	0.871157

Table 1: [NNWOP] Evaluation metrics on unseen data for the best parameter configuration.

The results show that BP gives best fitness function values as well as best generalization on unseen data and hence it is perfect for optimizing weights of a neural network. Fitness function values and generalization achieved via RHC and SA is very close to the optimal values, but they take orders of magnitude longer to run. GA is not well suited for this problem as it results in worst both fitness function value and generalization.

Conclusions

In this report, we studied 4 randomized optimization algorithms. Each algorithm was found to have a niche problem domain. For example, RHC and SA excels navigating fitness landscape that has multiple local optima (CCP) and convex optimization (NNWOP). GA works well with problems that has near optimal substructure (TSP) and MIMIC is the best if we are trying to optimize for physical structure (2CP). Following table gives a comprehensive summary of the algorithms studied in this report.

Algorithm	Advantages	Disadvantages
RHC	• Suitable for problems with multiple optima • Suitable for convex optimization • Fast, if fitness function is easy to compute • Almost no parameter tuning	• Slow, if fitness function is expensive • Slow, if there are too many neighbors
GA	• Suitable for problems that has near optimal substructures	• Not suitable for convex optimization • Slow • Parameters are hard to tune • Tends to get stuck in local optima
SA	• Suitable for problems with multiple optima • Suitable for convex optimization • Fast, in most cases	• Slow, if temperature is reduced very slowly for convergence • Parameters are hard to tune
MIMIC	• Suitable for problems that has real physical structure	• Not suitable for convex optimization • Slow • Parameters are hard to tune

References

[1] "Hill climbing - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Hill_climbing. [Accessed: 09-Oct-2020].

[2] "Genetic algorithm - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Genetic_algorithm. [Accessed: 09-Oct-2020].

[3] "Simulated annealing - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Simulated_annealing. [Accessed: 09-Oct-2020].

[4] "Introduction, Implementation and Comparison of Four Randomized Optimization Algorithms | by Nini | Medium." [Online]. Available: https://medium.com/@duoduoyunnini/introduction-implementation-and-comparison-of-four-randomized-optimization-algorithms-fc4d96f9feea. [Accessed: 09-Oct-2020].

[5] "Estimation of distribution algorithm - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Estimation_of_distribution_algorithm. [Accessed: 09-Oct-2020].

[6] J. S. De Bonet, C. L. Isbell, and P. Viola, "MIMIC: Finding Optima by Estimating Probability Densities," *Advances in Neural Information Processing Systems*, p. 424, Jan. 1996.

[7] G. Hayes, "mlrose: Machine Learning, Randomized Optimization and SEarch package for Python." 2019.

[8] "Graph coloring - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Graph_coloring. [Accessed: 09-Oct-2020].

[9] "Travelling salesman problem - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Travelling_salesman_problem. [Accessed: 09-Oct-2020].

[10] J. Y. Potvin, "Genetic algorithms for the traveling salesman problem," *Annals of Operations Research*, vol. 63, no. 3, pp. 339–370, 1996.

[11] J. Haslett and A. E. Raftery, "Space-Time Modelling with Long-Memory Dependence: Assessing Ireland's Wind Power Resource," *Applied Statistics*, vol. 38, no. 1, p. 1, 1989.

[12] "OpenML wind." [Online]. Available: https://www.openml.org/d/503. [Accessed: 10-Oct-2020].

[13] "OpenML wind." [Online]. Available: https://www.openml.org/d/847. [Accessed: 10-Oct-2020].