# Machine Learning, Assignment 2 Report

Seyed Aghigh

Gatech Email: saghigh3@gatech.edu

## 1 Introduction

In this assignment, we want to explore the applications of randomized search algorithms (RSA). To this end, this algorithms are evaluated on three optimization problems and also on finding the optimum weights of neural networks implemented in assignment #1. Selected optimization problems and RSA are as follow.

(i) Max Graph k-color Optimization Problem
(ii) knapsack Optimization Problem
(iii) Flip Flop Optimization Problem

1. Randomized Hill Climbing (RHC)
2. Simulated Annealing (SA)
3. a Genetic Algorithm (GA)
4. Mutual-Information-Maximizing Input Clustering (MIMIC)

## 2 Materials and Methods

### 2.1 Code Accessibility

All implemented functions and Read-me file are available in the following link
https://drive.google.com/drive/folders/1niJxLiiKxj-hplYPpLmLKCaTWykhTS9P?usp=sharing

### 2.2 Randomized Search Algorithm

Randomized Search Algorithm (RSA) is a heuristic search. In the first step, a problem with a known objective function must be determined, called ***fitness function***. Fitness function, completely, depends on the problem at hand, the main goal is to either minimize or maximize the fitness function. The general workflow is as follow.

1. Define the problem with known fitness function.
2. Run a RSA on a problem with pre-defined range hyper-parameters.
3. Determine the iterations that have minimum/ maximum fitness score.
4. Find the optimum hyper-parameters values based on previous step.
5. Query the whole iterations that have optimum fitness score along with minimum running time.
6. Plot number of iterations versus fitness score graph to show convergence of each RSA algorithm
7. Plot and discuss the result of hyper-parameters effect on fitness score and running time
8. plot and discuss the result of all RSA methods on problem at hand.

The hyper-parameters for each RSA method is as follow.

- **number of iterations**: $2^0, \ldots, 2^{10}$
- **RHC**: Number of restart in each iteration.
- **SA**: Temperature parameter and schedule decay methods.
- **GA**: Population size and mutations rate.
- **MIMC**: Population size and percentage of keeping the sampled data points.

## 3 Result and Discussion for Max Graph k-color

Given a graph, in max graph k-color optimization problem, we want to see if nodes can be colored with k different colors such that the color of any two adjacent are distinct. The goal is to minimize the pairs of adjusent nodes that have same colors. This problem is NP-complete problem [1] meaning that so far there is no polynomial algorithm to solve this problem. Therefore, solution to this problem are approximated using

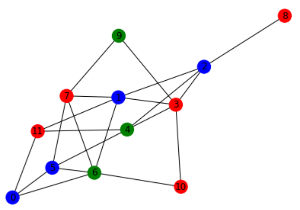
[1] https://en.wikipedia.org/wiki/NP-completeness

randomized search algorithm (RSA). The same problem of graph k-coloring is randomly generated to evaluate the four aforementioned RSA algorithm with following configuration.

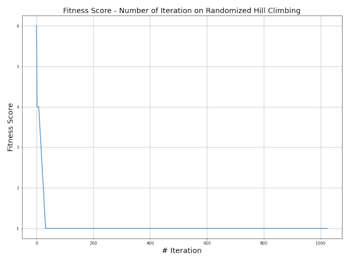
- **number of nodes in the graph**: 12
- **maximum number of links per node**: 3
- **maximum number of color (k)**: 3

## 3.1 Randomized Hill Climbing on Max Graph k-color Problem

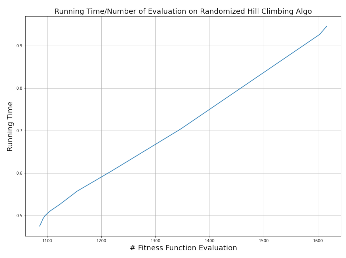
Figure 1 shows the optimum results gained by RHC. In Figure 1a,the resulting graph for RHC on graph 3-coloring problem is illustrated. As it seen, there is a pair of adjacent nodes that have the same color. This indicates the fitness function does not reach to minimum value 0 and further results also confirms this. Figure 1b shows the fitness score vs number of fitness function evaluation. As it seen, in this problem, the RHC was able to minimize the fitness score to 1, while ideally, it is wanted to be 0. Additionally, with 1123 number of fitness function evaluation for the first time, the value of fitness function is 1 which is the minimum in this case. Figure 1c shows the running time vs number of fitness function evaluation, The graph shows the running time is perfectly linear with respect to number of iterations.



**(a)** Graph 3-coloring for RHC.



**(b)** Fitness score vs # iterations in RHC on graph 3-coloring.

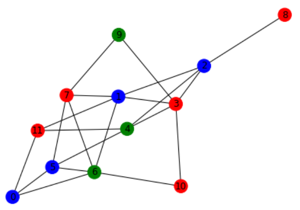


**(c)** Running Time vs # iterations in RHC on graph 3-coloring.

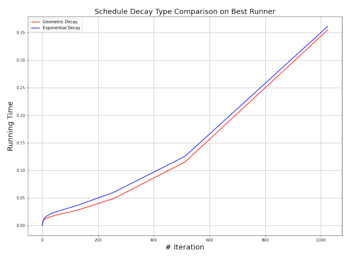
**Figure 1:** RHC result for graph 3-coloring optimization problem.

## 3.2 Simulated Annealing on Max Graph k-color Problem

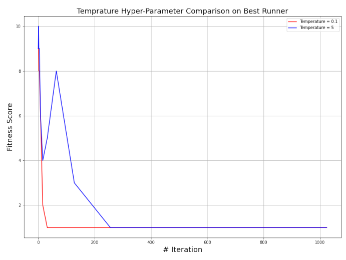
Below shows the result for SA on graph 3-coloring Figure 2. In Figure 2a shows the resulting graph. Similar to RHC, in the optimum graph coloring gained by SA also there are adjacent nodes with same color indicating that the fitness score was not able to converge the optimum 0 value. Figure 2b shows the running time with respect to the number of iterations in two different scenarios; schedule decay of geometric and exponential. As it seen from the plot, when the schedule decay is set to exponential, running time of algorithm slightly increase in compare to when decay is geometric. This could mean geometric decay is changing the rate of temperature more ideally. Moreover, Figure 2c illustrates the fitness score vs number of iterations in two scenarios; temperature equal to 0.1 (the red line) and temperature equal to 5 (the blue line). As it seen, setting temperature to smaller number (Taking smaller steps) helps the algorithm to converge is smaller number of iteration.



**(a)** Graph 3-coloring for SA.



**(b)** SA running time vs # iterations in two schedule decay scenarios; blue: exponential and red: geometric.



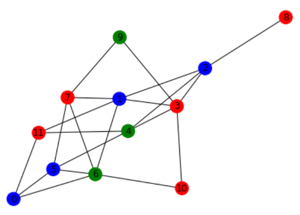
**(c)** SA fitness score vs # iterations in two different temperatures; blue: temperature is 5, and red: temperature is 0.1 .

**Figure 2:** SA result for graph 3-coloring optimization problem.

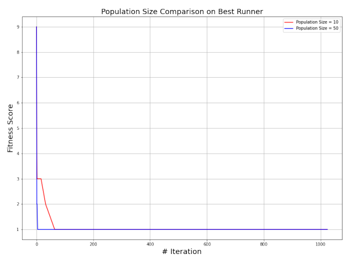
## 3.3 Genetic Algorithm on Max Graph k-color Problem

The results for GA is available in Figure 3. Figure 3a shows the optimal graph gained by GA on graph 3-coloring problem. Likewise to previous RSA, in this graph, there are adjacent nodes that have similar color which contradicts the objective of this problem. And therefore, similar to previous examples, in this problem, the fitness score also was not able to reach to minimum value 0. Figure 3b shows the fitness score vs number of iterations in two cases; red line shows when the population size is equal to 10 while the blue line is for

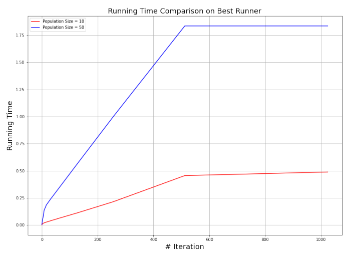
50. Comparing the result from the two cases, it is seen, starting with lager population helps the algorithm converge in lesser number of iteration versus smaller population size. Not surprisingly, Figure 3c also shows the running time of GA increased with larger population size (blue line) because it would need to compute the fitness function for every single sample. In next figure, Figure 3d, fitness score vs the number of iterations is shown when the mutation rate is 0.1 (the red line) vs the it is 0.5 (the blue line), And, as it seen, when the mutation rate is higher (i.e. 0.5) the fitness score is minimized in smaller number of iterations. This indicates the higher mutation rate has better impact on reaching the optimal answer. Figure 3e shows the running time of GA for the mutation rate hyper-parameter. On the contrary to the fitness score, in case of higher mutation rate, algorithm takes more time, this could be due to extending the exploration space by mutating at higher rate. Finally, in Figure 3f it shown that GA needs to do less number of fitness evaluation when the population size is 50 in compare to when the population size is 10, this indicates increasing the search space (samples) can help the algorithm to find the optimum solution in less number of evaluations.



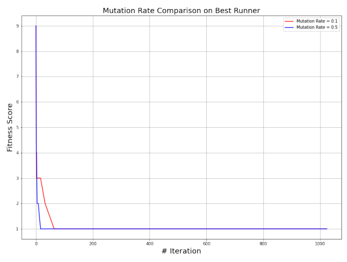
**(a)** Graph 3-coloring for GA.



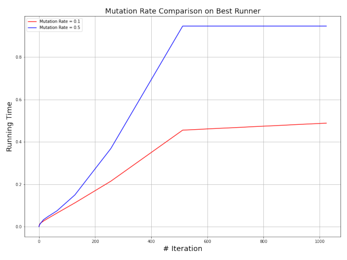
**(b)** Fitness score vs # iterations; in red populationSize is 10, in blue population size is 50



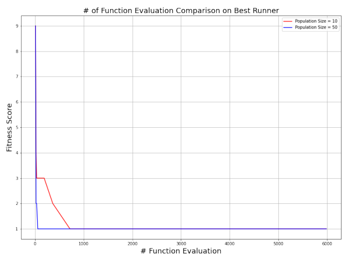
**(c)** Running time vs # iterations; in red populationSize is 10, in blue population size is 50



**(d)** Fitness score vs # iterations; red: mutation rate is 0.1 and blue: mutation rate is 0.5



**(e)** Running time vs # iterations; red: mutation rate is 0.1 and blue: mutation rate is 0.5

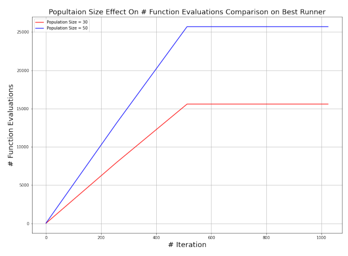


**(f)** Fitness score vs # function evaluation; red: mutation rate is 10, and blue: mutation rate is 50

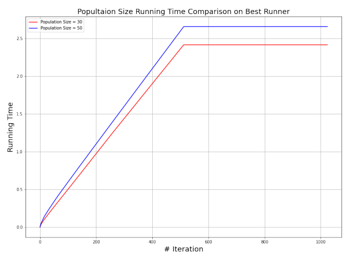
**Figure 3:** GA result for graph 3-coloring optimization problem.

### 3.4 Mutual-Information-Maximizing Input Clustering on Max Graph k-color

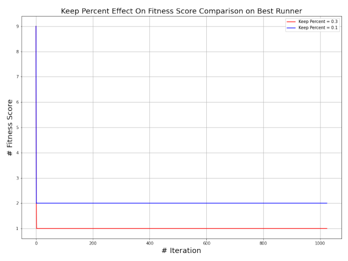
In Figure 4, the results for MIMC is illustrated. Figure 4a shows the number of function evaluations vs number of iterations in two scenarios; when the population size is 30 (the red scenario) and when the population size is 50 (the blue scenario). More evaluation is done when the population size is greater. Similarly, in Figure 4b it is illustrated that in case of higher population size (blue 50 and red 30) algorithm takes more time. Finally, as it is show in Figure 4c when keep percent is 0.3 (the red line) in compare with 0.1 (the blue line) algorithm is able to converge to the minimal value which is 1. Again, since the best gained fitness value is not 0, this means, in the final optimal answer the are at least one pair of adjacent nodes that have the same number of color.



**(a)** # function evaluation vs # iterations; red: populationSize is 30, and blue: populationSize is 50.



**(b)** Running time vs # iterations; red: populationSize is 30, and blue: populationSize is 50.



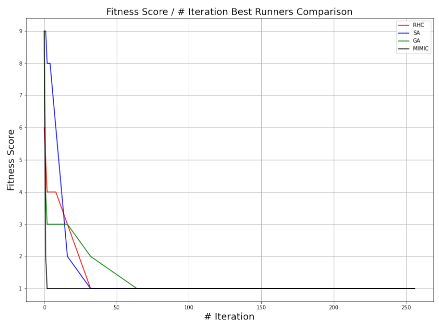
**(c)** Fitness score vs # iterations; red: keepPercent is 0.3, and blue: keepPercent is 0.1

**Figure 4:** MIMC result for graph 3-coloring optimization problem.

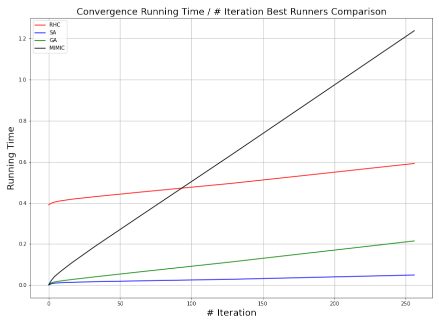
### 3.5 Conclusion for Max Graph k-color

So far, we have been discussed the optimal possible solution within each RSA domain alone. Now, we will compare the outcome of the best possible solution of each method, to agree on a single answer for this problem at hand. The two common factors within all this algorithm is the fitness score and running time. Ideally, we want to get an answer which has the optimize fitness score, (in this example, the minimum fitness score as possible) and the less running time. To this end, we compare the fitness score and running time of the best solution for each method at once. In both plots, Lines red, blue, green, and black shows the results for RHC, SA, GA, and MIMC respectively. Figure 5a shows the fitness score of the best solution of each method. As it seen, all methods at the end reach to the fitness score minimum 1. This happens in MIMC the sooner followed by SA. GA and RHC are placed in next. As it seen in Figure 5b, SA takes the least followed by GA. RHC is in the next place, and MIMC is the worse.

Comparing the two plots, and results from previous step, it can be said that SA can be selected as the best algorithm for this solution. Although it takes longer iterations to reach to the minimal fitness score in compare with MIMC, in terms of running time, it performs the best and takes the less amount of time. This amount is the highest for MIMC. Intuitively, in Max Graph k-color problem considering the underlying structure of data will help the algorithm to converge faster, among RSA algorithms, MIMIC generate and memorizes structure for given problem, however this comes at price of extensively longer running time.



**(a)** Fitness score for all the RSA methods on graph k-color.



**(b)** Running time for all the RSA methods on graph k-color

**Figure 5:** Fitness score and running time performance for RSA in graph 3-color optimization problem.

## 4 Result and Discussion for knapsack Optimization Problem

Knapsack problem is an optimization problem in which given a set of items that each has their own value ,weight and count, we want to full a knapsack with weight W of items such that the total value of the items are maximize and the total weights of the items are not exceeding W [2]. Therefore, solution to this problem can be approximated using randomized search algorithm (RSA). Single problem of knapsack is randomly generated to evaluate the four aforementioned RSA algorithm with following configuration.

- **number of items**: 8
- **max count per item**: 10
- **max weight per item**: 12
- **max value per item**: 5
- **max weight** : 0.7
- **knapsack weight**: $2 * 0.7$

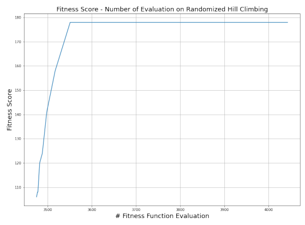
### 4.1 Randomized Hill Combining on Knapsack Problem

The results are illustrated in Figure 6. Figure 6a shows the fitness score vs the number of fitness evaluation function and as it seen, at 3551 the fitness score become 178.0. Similarly Figure 6b shows the result of the fitness score vs the number of iterations, at iteration 64 the algorithms reach the fitness score 3551. Finally, Figure 6c shows the running time vs the number of iterations which again shows the linear relation between iteration and time in knapsack problem.

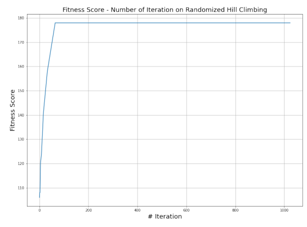
### 4.2 Simulated Annealing on Knapsack Problem

graph Figure 7 shows the result of SA on Knapsack problem. Figure 7a shows the fitness score vs number of iterations. At iteration 332 the fitness score converges to 180.0. Next, Figure 7b shows a non-linear running time of this algorithm with respect to the number function evaluations. Figure 7c and Figure 7d shows the SA in two scenarios; red line shows the exponential schedule decay with temperature equal to 8, and the blue line show the geometric schedule decay with temperature 8. As it seen, in both scenario the fitness function will be eventually 180 however geometric decay converges faster than exponential decay while, exponential decay also take more

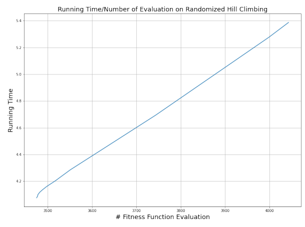
[2]https://en.wikipedia.org/wiki/Knapsack_problem



**(a)** Fitness score vs fitness evaluation function.



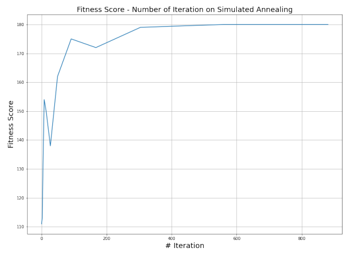
**(b)** Fitness score vs # iteration.



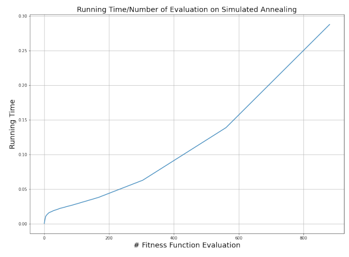
**(c)** Running time vs # iterations.

**Figure 6:** Results for RHC on knapsack optimization problem.

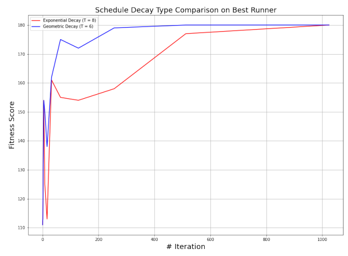
time in compare with geometric decay. Therefore, geometric is a better setting for SA in this problem. In the next step, the effect of temperature will be evaluated. Plots Figure 7e and Figure 7f illustrates the algorithm in geometric decay with two temperatures; red line is 10 and the blue line is 6. An as it seen from Figure 7e, in both the algorithms will converge to 180.0 while according to Figure 7f the algorithm with temperature 6 takes less time and number of iteration. Generally smaller temperature will encourage the exploitation which means the algorithm will spend less time exploring the input space and concentrate on reaching optimal point.



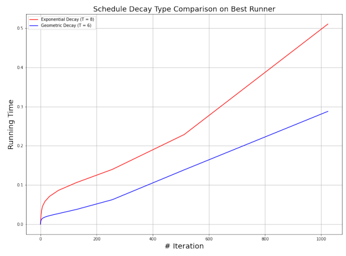
**(a)** Fitness score vs # iterations.



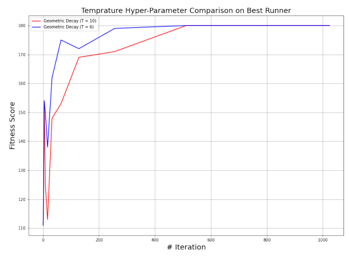
**(b)** Running time vs #3iterations.



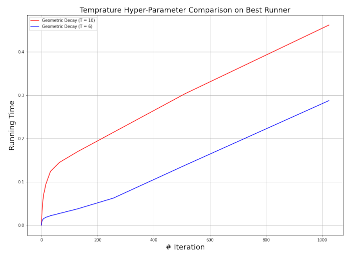
**(c)** Fitness score vs #iteration; red: exponential schedule with temperature 8, and blue: geometric schedule with temperature 8.



**(d)** Running time vs # iterations; red: exponential schedule with temperature 8, and blue: geometric schedule with temperature 8.



**(e)** Fitness score vs # iterations with geometric schedule; red: temperature is 10, and blue: temperature is 6.

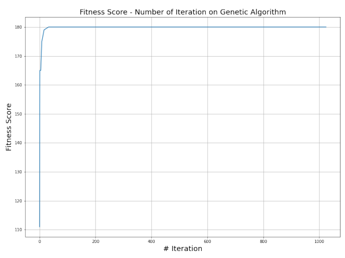


**(f)** Running time vs # iterations with geometric schedule; red: temperature is 10, and blue: temperature is 6.

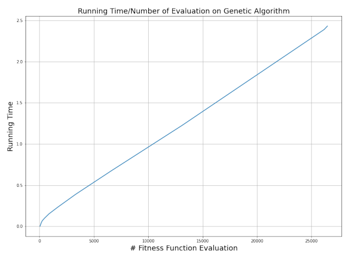
**Figure 7:** Results for SA on knapsack optimization problem.

## 4.3 Genetic Algorithms on Knapsack Problem

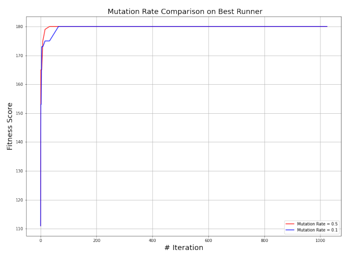
In Figure 8 results for GA on knapsack optimization problem is shown.Figure 8a shows the fitness scores vs the number of iterations. At iteration 18 the algorithms reach to 180.0. Figure 8b shows a linear running time vs the number of iterations, Figure 8c and Figure 8d shows the algorithms in two scenarios, the red line is when the mutations rate is 0.5 while the blue line indicates the 0.1 mutation rate. In Figure 8c, the algorithm in both scenarios will converge to 180, however, this convergence is a slightly faster when mutation rate is 0.5. From Figure 8d it can be inferred that the running time for mutation rate 0.1 is higher than 0.5 suggesting with higher mutation we increase the chance of reaching global optimum faster. Figure 8e and Figure 8f illustrates the algorithm when the population size is 50 (the red line) and 10 (the blue line). As it seen from Figure 8e, the algorithm with lower population size (size =10) does not achieve the optimum fitness score suggesting that, taking larger sample size will help the algorithm to explore input space more thoroughly. However, based on Figure 8f this comes at higher time expense.



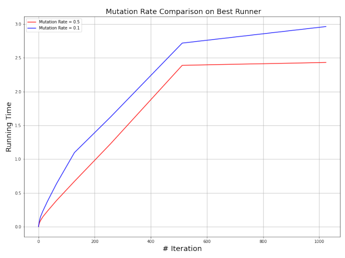
**(a)** Fitness score vs # iterations



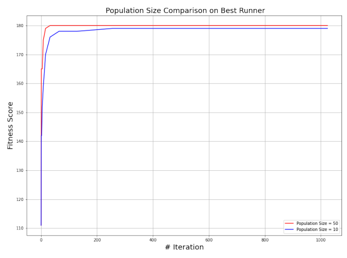
**(b)** Running time vs # function evaluations.



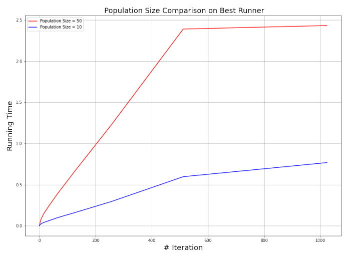
**(c)** Fitness score vs # iterations; red: mutation rate is 0.5, blue: mutation rate is 0.1.



**(d)** Running time vs # iterations; red: mutation rate is 0.5, blue: mutation rate is 0.1.



**(e)** Fitness score vs # iterations; red: population size is 50, blue: population size is 10.

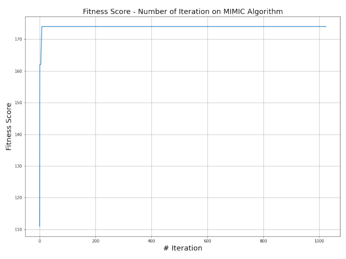


**(f)** Running time vs # iterations; red: population size is 50, blue: population size is 10.

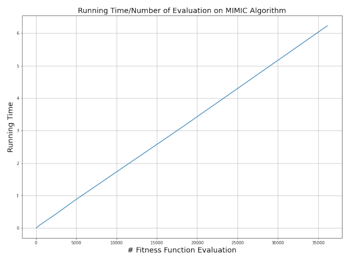
**Figure 8:** Results for GA on knapsack optimization problem.

### 4.4 Mutual-Information-Maximizing Input Clustering on Knapsack Problem

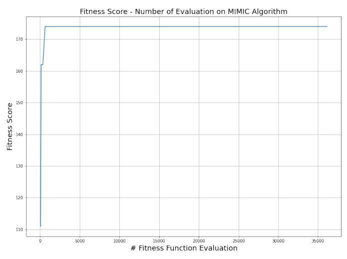
In Figure 9, the results for MIMC is illustrated. Figure 9a shows the fitness score vs the number of iterations, and this algorithms also reach the optimum value 174.0 at 8 iterations. Figure 9b also shows a linear running time with respect to number of function evaluation. Figure 9c depicts the fitness score vs number of function evaluation. In Figure 9d the keep percent hyper-parameter of MIMC algorithm is evaluated in two situations; in red and blue lines are 0.1 and 0.5 respectively. Interestingly, when the keep percent is 0.1 the algorithm converge to optimum fitness score value while with keep percent of 0.5 it does not reach opyimum value, which suggest that with lower percentage of population size algorithm tends to do a better job. The last two plots studies the effect of population size on MIMC when the population sizes are 70 for red and 100 for the blue line. In Figure 9e is shown that only in case of population size 70 the algorithm converge to 174.0 fitness score. Figure 9e shows the bigger the population size the longer is running time.



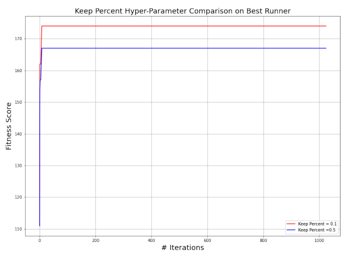
**(a)** Fitness score vs # number of iterations



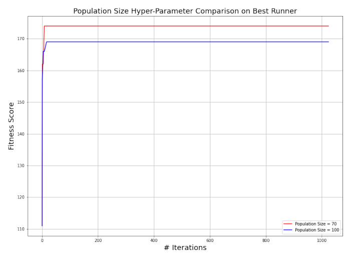
**(b)** Running time vs # function evaluations



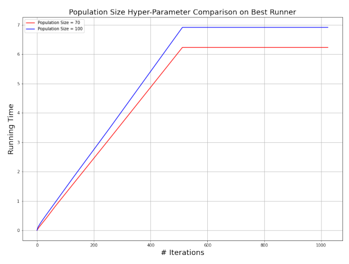
**(c)** Fitness score vs # function evaluations



**(d)** Fitness score vs number of iterations; red: keep percent is 0.1, and blue keep percent is 0.5



**(e)** Fitness score vs # iterations; red: populationSize is 70, and blue: populationSize is 100

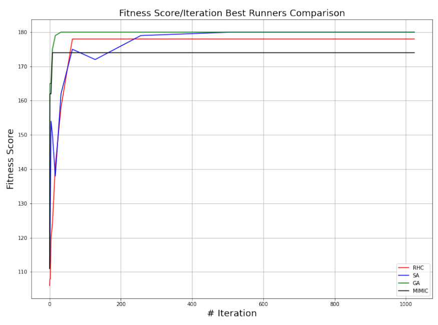


**(f)** Running time vs # iterations; red: populationSize is 70, and blue: populationSize is 100

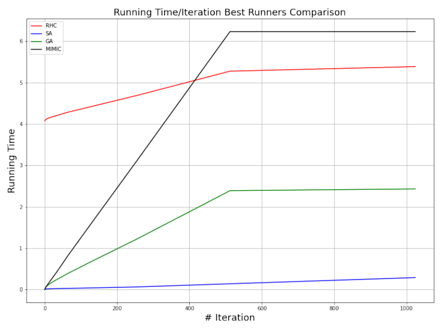
**Figure 9:** MIMC results on. knapsack optimization problem.

### 4.5 Conclusion for Knapsack Optimization Problem

Figure 10 compares the running time and fitness score efficiency for the four RSA methods on knapsack optimization problem. Lines red, blue, green, and black shows the RHC, SA, GA, and MIMC. According to Figure 10a only SA, and GA are the algorithms that reach to optimum value 180.0, and among theses two, GA takes less number of iterations in compare with SA. Based on Figure 10b, SA and GA take the least amount of time to converge, while MIMC due to its feature inter-relational computation is doing the worst and RHC is in third place. SA takes significantly lesser time than other RSA algorithms, it can be said that SA may be a better choice with GA following it in the second place. Both MIMC and RHC never meet the optimum value and consequently they are not a good choice to this problem. It is important to note that the reason RHC algorithm running time doesn't start from zero, it's because the converges does not happen in the first restart.



(a) Fitness score for all RSA methods on knapsack optimization problem.



(b) Running time for all the RSA methods on knapsack optimization problem.

**Figure 10:** Fitness score and running time performance for RSA on knapsack optimization problem.

## 5 Result and Discussion for Flip Flop Optimization Problem

Flip flop is an optimization problem in which it counts the number of times of bits alterations in a bit string. A maximum fitness bit string would be one that consists entirely of alternating digits. Therefore, solution to this problem can be approximated using randomized search algorithms (RSA). Single problem of flip flop is randomly generated to evaluate the four aforementioned RSA algorithm with following configuration.

Note that since with number of iterations up to $2^{10}$ the fitness score did not converge to optimal value, this number is increased to $2^{14}$ and it is observed that with this number of iteration, the some of the fitness score converges to it optimum value.

- **size**: 500 bits
- **number of iterations**: $2^0$, $2^1$, ... , $2^{14}$

### 5.1 Randomized Hill Combining on Flip Flop Problem

Figure 11 shows the fitness score versus number of iterations. As the number of iterations increases the fitness score converges to 413, which is the optimum value for fitness score.

### 5.2 Simulated Annealing on Flip Flop Problem

Below shows the result for SA on flip flop optimization problem. Figure 12. Figure 12a shows the fitness score vs the number iterations and its optimal fitness score of 485. In Figure 12b, illustrates a non-linear running time versus function evaluation. The next two figures shows the algorithm in two cases; red line: geometric schedule decay and blue line: exponential decay both with temperature parameter set to 5. geometric decay converges to optimal value of 485 while exponential decay failed to do so. Figure 12c shows the the number of function evaluation vs the number of iterations, and as it seen, the algorithm almost behave the same in these two cases with exponential decay edging a little bit higher. which indicates geometric decay is a better choice over exponential decay.

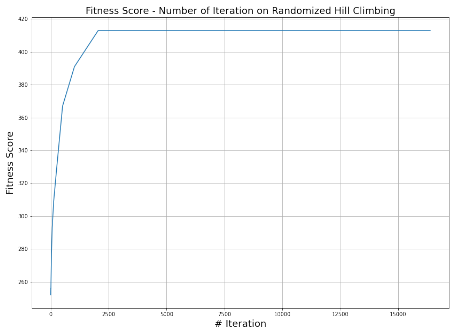
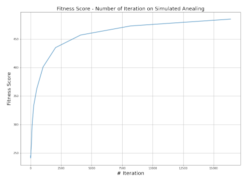
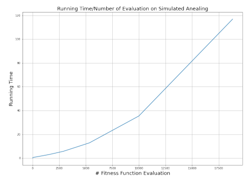
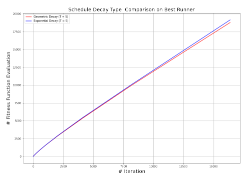


**Figure 11:** Fitness score vs # iterations for RHC on flip flop .


(a) Fitness score vs # iterations


(b) Running time vs # function evaluations


(c) # function evaluations vs # iterations; red: geometric schedule with temperature 5, and blue: exponential schedule with temperature 5

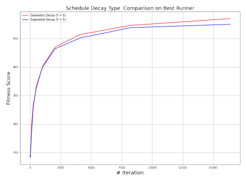
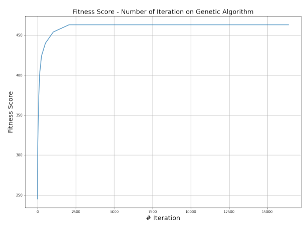
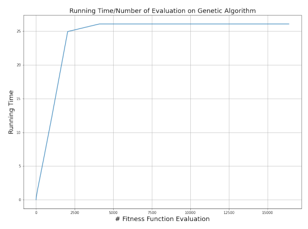

(d) Fitness score vs # iterations; red: geometric schedule with temperature 5, and blue: exponential schedule with temperature 5

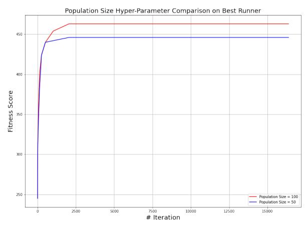
**Figure 12:** SA results on flip flop.

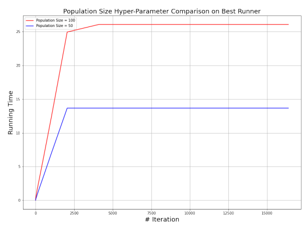
## 5.3 Genetic Algorithms on Flip Flop Problem

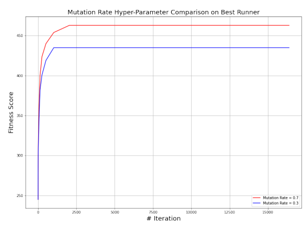
In Figure 13 results for GA on flip flop optimization problem is shown. Figure 13a and Figure 13b shows the algorithm number of iteration against fitness score and running time respectively. The optimum fitness score that it achieves is 463.0. Figure 13c and Figure 13d shows the algorithm in two scenarios; in red and blue lines population size 100 and 50 respectively. As it seen, when this hyper-parameter is 100, the algorithm achieves the highest fitness score as the input space for finding a global maxima is explored with higher population size. Additionally as expected, running time for red line is much higher than the smaller population size. The next two plots shows the algorithm when the mutation rate is 0.7 and 0.3 in the red and blue line respectively. With higher mutation rate, the algorithm could reach to higher fitness score, and also it takes more time. According to this result higher population size and mutation rate is a better choice for this algorithm. However, the running time of the algorithm in these two situation is higher.


(a) Fitness vs # iterations


(b) Running time vs # function evaluation


(c) Fitness score vs # iterations; red: populationSize is 100, and blue: populationSize is 50


(d) Running time vs # iterations; red: populationSize is 100, and blue: populationSize is 50


(e)

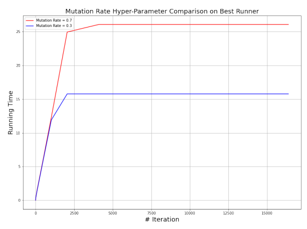
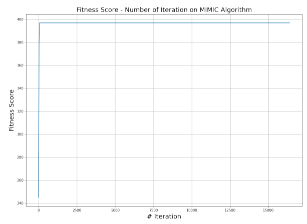

(f)

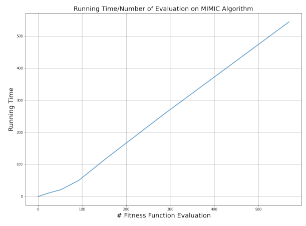
**Figure 13:** GA results on flip flop optimization problem.

## 5.4 Mutual-Information-Maximizing Input Clustering on Knapsack Problem

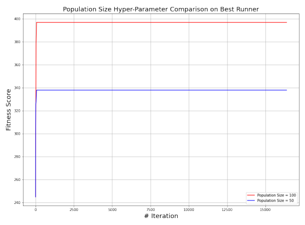
In Figure 14, the results for MIMC is illustrated. Figure 14a and Figure 14b shows the algorithm number of iterations against fitness score and running time. The algorithm achieve its optimal fitness score value 397.0 at 32 iterations. Figure 14c and Figure 14d illustrates the algorithm in two situations; the red and blue lines is the result when the population size is 100 and 50 respectively. In Figure 14c, in case of red line, the algorithm achieve a highest optimal value by far in compare with blue line, suggesting that when population size is higher, the algorithm is more successful to reach highest fitness value. Similarly, the running time for this case is also higher as all mutual information between features needs to be evaluated. The next two plots shows the algorithm in terms of the keep percent parameters which 0.5 and 1 in red and blue lines respectively. Based on Figure 14e, when the keep percent rate is smaller, the algorithm achieves a better fitness score. And, according to Figure 14f the algorithm in both scenarios, exhibits similar running time behaviour.



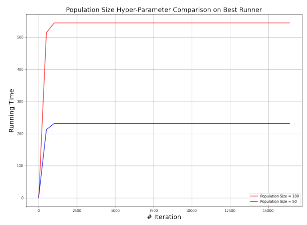
**(a)** Fitness score vs # iterations



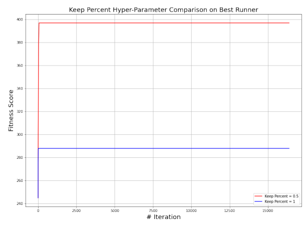
**(b)** Running time vs # function evaluations



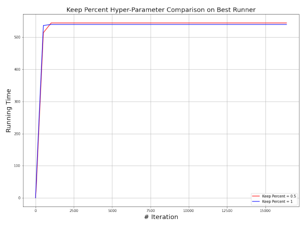
**(c)** Fitness score vs # iterations; red: populationSize is 100 and blue: populationSize is 50



**(d)** Running time vs # iterations; red: populationSize is 100 and blue: populationSize is 50



**(e)** Fitness score vs # iterations; red: mutation rate 0.7 and blue: mutation rate is 0.3

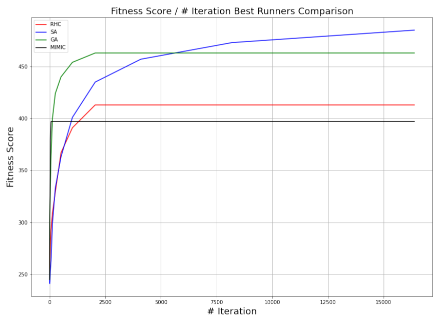


**(f)** Running time vs # iterations; red: mutation rate 0.7 and blue: mutation rate is 0.3

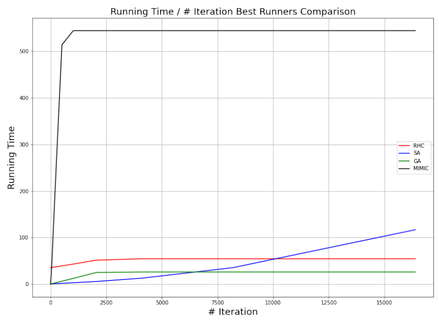
**Figure 14:** MIMC results on flip flop optimization problem.

## 5.5 Conclusion for Flip Flop Optimization Problem

Figure 15 compares the running time and fitness score efficiency for the four RSA methods on flip flop optimization problem. Red, blue, green, and black lines denote RHC, SA, GA, and MIMC methods respectively. According to Figure 15a SA is the only which achieves the best fitness score, followed by GA. MIMC and RHC are the worse. From Figure 15b it can be seen that running time of MIMC is by far worse than the rest. surprisingly, this algorithm does not perform good in this problem even though the structure of underlying problem is important. GA has the least running time, and as the number of iterations increases, the running time of SA noticeably increases. This result suggests that from fitness score perspective SA performs the best, followed by GA. Whereas from running time perspective, GA outperforms in compare to the rest of RSA.



**(a)** Fitness score and running time performance for RSA on flip flop optimization problem



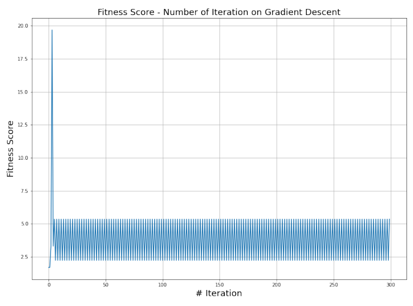
**(b)**

**Figure 15:** Fitness score and running time performance for RSA on flip flop optimization problem.

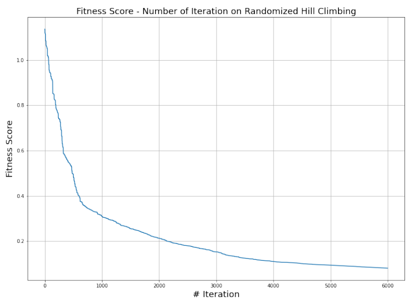
# 6 Result and Discussion for Neural Network on Bank Data

Bank dataset[3] is a supervised data which includes the ustomer demographic information, he customer's association with the bank, and a dummy variable Personal loan which indicates whether the customer accepts the loan or not. The ***objective* is to come up with a model to make prediction of the likelihood of a liability customer accepting *Personal loan***. Figure 16 shows the neural networks (NN) results for Gradient Descent and RSA optimization algorithms. Gradient Descent algorithm is used to produce a baseline comparison with other RSA methods. Figure 16a, shows Gradient Descent quickly drops from 19 to ranges between 2 and 5 and starts to oscillate in that range which could be indicative of not properly chosen learning rate. finding the optimum learning rate is skipped due to limitation of time. Figure 16b, Figure 16c, and Figure 16d illustrates the fitness score vs number of iterations on NN with RHC, SA, and GA respectively. RHC algorithm outperforms other RSA methods. Classification report for each RSA methods are included in supplementary section.

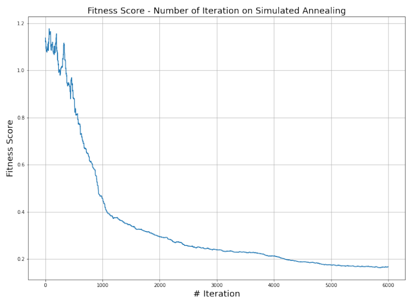
[3]urlhttps://www.kaggle.com/krantiswalke/bank-personal-loan-modelling



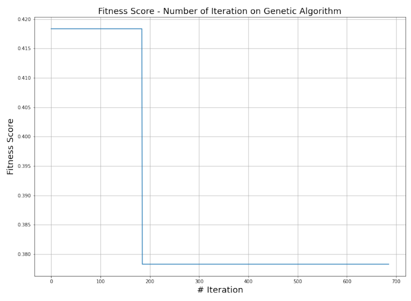
**(a)** Fitness score vs # iterations for gradient descent



**(b)** Fitness score vs # iterations for RHC.



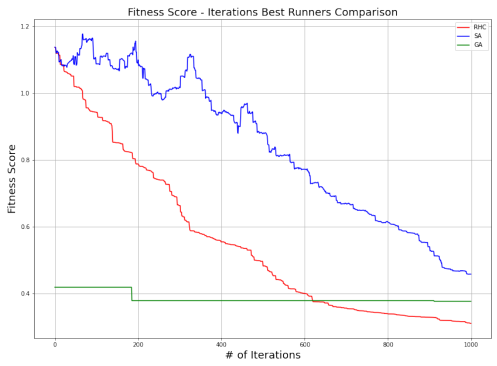
**(c)** Fitness score vs # iterations for SA



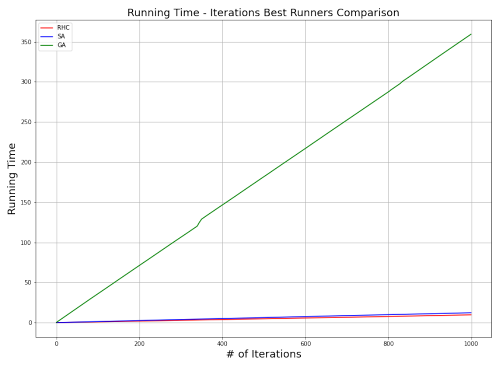
**(d)** Fitness score vs # iterations for GA

**Figure 16:** Fitness score for RSA on neural networks.

In Figure 17 all the RSA are compared in terms of fitness score and running time. Line red, blue, and green denote to RHC, SA, and GA respectively. In Figure 17a illustrate fitness score vs the number of iterations for the first 1000 iterations. Similarly, in Figure 17b the running time of the methods are compared, RHC algorithm excelled both in running time and fitness score.



**(a)** Fitness score vs # iterations



**(b)** Running time vs # iterations

**Figure 17:** Fitness score and running time for all RSA of NN.

## 7 Supplementary Documents

### 7.1 Classification Report for NN with RSA on Train and Test

Table 1 and Table 2 shows the classification report for a base line machine learning algorithm with gradient descent. Table 3 and Table 4 shows the classification report for train and test of NN with RHC. Table 5 and Table 6 shows the classification report for train and test of NN with SA. Finally, Table 7 and Table 8 shows the classification report of NN with GA on train and test.

| NN Performance on Train Data | | | |
|---|---|---|---|
| | **precision** | **recall** | **f1-score** |
| class 0 | 0.90 | 1.00 | 0.95 |
| class 1 | 1.00 | 0.00 | 0.00 |
| **micro avg** | 0.90 | 0.90 | 0.90 |
| **macro avg** | 0.95 | 0.50 | 0.47 |
| **weighted avg** | 0.91 | 0.90 | 0.86 |
| **samples avg** | 0.90 | 0.90 | 0.90 |

**Table 1:** Train NN performance on Bank data.

| NN Performance on Test Data | | | |
|---|---|---|---|
| | **precision** | **recall** | **f1-score** |
| class 0 | 0.90 | 1.00 | 0.95 |
| class 1 | 1.00 | 0.00 | 0.00 |
| **micro avg** | 0.90 | 0.90 | 0.90 |
| **macro avg** | 0.95 | 0.50 | 0.47 |
| **weighted avg** | 0.91 | 0.90 | 0.86 |
| **samples avg** | . 0.90 | 0.90 | 0.90 |

**Table 2:** Test NN performance on Bank data.

| NN Performance on with RHC Train Data | | | |
|---|---|---|---|
| | **precision** | **recall** | **f1-score** |
| class 0 | 0.98 | 1.00 | 0.99 |
| class 1 | 0.95 | 0.78 | 0.86 |
| **micro avg** | 0.97 | 0.97 | 0.97 |
| **macro avg** | 0.96 | 0.89 | 0.92 |
| **weighted avg** | 0.97 | 0.97 | 0.97 |
| **samples avg** | 0.97 | 0.97 | 0.97 |

**Table 3:** Train NN performance with RHC on Bank data.

| NN Performance on Test Data | | | |
|---|---|---|---|
| | **precision** | **recall** | **f1-score** |
| class 0 | 0.98 | 1.00 | 0.99 |
| class 1 | 0.95 | 0.79 | 0.86 |
| **micro avg** | 0.98 | 0.98 | 0.98 |
| **macro avg** | 0.96 | 0.89 | 0.93 |
| **weighted avg** | 0.93 | 0.98 | 0.98 |
| **samples avg** | 0.98 | 0.98 | 0.98 |

**Table 4:** Test NN performance with RHC on Bank data.

| NN Performance on with SA Train Data | | | |
|---|---|---|---|
| | **precision** | **recall** | **f1-score** |
| class 0 | 0.93 | 1.00 | 0.96 |
| class 1 | 0.94 | 0.33 | 0.49 |
| **micro avg** | 0.93 | 0.93 | 0.93 |
| **macro avg** | 0.94 | 0.66 | 0.73 |
| **weighted avg** | 0.93 | 0.93 | 0.92 |
| **samples avg** | 0.93 | 0.93 | 0.93 |

**Table 5:** Train NN performance on Bank data.

| NN Performance on Test Data | | | |
|---|---|---|---|
| | **precision** | **recall** | **f1-score** |
| class 0 | 0.93 | 1.00 | 0.96 |
| class 1 | 0.92 | 0.30 | 0.45 |
| **micro avg** | 0.93 | 0.93 | 0.93 |
| **macro avg** | 0.93 | 0.65 | 0.71 |
| **weighted avg** | 0.93 | 0.93 | 0.91 |
| **samples avg** | 0.93 | 0.93 | 0.93 |

**Table 6:** Test NN performance with RHC on Bank data.

| NN Performance on with GA Train Data | | | |
|---|---|---|---|
| | **precision** | **recall** | **f1-score** |
| class 0 | 0.92 | 0.96 | 0.94 |
| class 1 | 0.43 | 0.26 | 0.33 |
| **micro avg** | 0.90 | 0.90 | 0.90 |
| **macro avg** | 0.68 | 0.61 | 0.63 |
| **weighted avg** | 0.88 | 0.90 | 0.88 |
| **samples avg** | 0.90 | 0.90 | 0.90 |

**Table 7:** Train NN performance on Bank data.

| NN Performance on Test Data | | | |
|---|---|---|---|
| | **precision** | **recall** | **f1-score** |
| class 0 | 0.93 | 0.97 | 0.95 |
| class 1 | 0.49 | 0.31 | 0.38 |
| **micro avg** | 0.90 | 0.90 | 0.90 |
| **macro avg** | 0.71 | 0.64 | 0.66 |
| **weighted avg** | 0.89 | 0.90 | 0.89 |
| **samples avg** | 0.90 | 0.90 | 0.90 |

**Table 8:** Test NN performance with GA on Bank data.

Hyper-parameter tuning for algorithms on different problems

### 7.2 RHC on Max Graph k-color Problem

Randomized Hill Climbing (RHC) with following hyper-parameter setting is run the problem

- **restarting points**: 10, 20, 50

Below is the achieved final result where max_iters is the maximum number of iteration and current_restart is the number start to convergence.

- **Iteration**: 32
- **Fitness**: 1.0
- **FEvals**: 1123
- **Time**: 0.430241
- **Restarts**: 10
- **max_iters**: $2^{10}$
- **current_restart**: 2

## 7.3 SA on Max Graph k-color Problem

Simulated Annealing (SA) with following hyper-parameters is run on the problem.

- **temperature**:= $0.1, 0.5, 1, 1.5, 2, 5$
- **decay list**: Geometric & Exponential Decay

Following the procedure in subsection 2.2, below shows the configuration for best hyper-parameters. Below is the achieved final result.

- **Iteration**: 32
- **Time**: 0.016404
- **Fitness**: 1.0
- **FEvals**: 43.0
- **schedule_type**: geometric
- **schedule_init_temp**: 0.1
- **schedule_min_temp**: 0.99
- **Temperature**: 0.1
- **max_iters**: 1024

## 7.4 GA on Max Graph k-color Problem

Genetic Algorithm (GA) with following hyper-parameters is run on the problem.

- **population_sizes**: $10, 20, 50$
- **mutation_rate**: $0.1, 0.3, 0.5$

Following the procedure in subsection 2.2, below shows the configuration for best hyper-parameters.

- **Iteration**: 64
- **Fitness**: 1.0
- **FEvals**: 718
- **Time**: 0.064387
- **max_iters**: $2^{10}$
- **pop_size**: 10
- **mut_rate**: 0.1

## 7.5 MIMC on Max Graph k-color Problem

Mutual-Information-Maximizing Input Clustering (MIMC) with following hyper-parameters us run on the problem.

- **population_sizes**: 10, 30, 50
- **keep_percent_list**: 0.1, 0.3, 0.5, 1

Following the procedure in subsection 2.2, below shows the configuration for best hyper-parameters.

- **Iteration**: 2
- **Fitness**: 1.0
- **FEvals**: 94
- **Time**: 0.023918
- **pop_size**: 30
- **keep_percent**: 0.3

## 7.6 RHC on Knapsack Optimization Problem

RHC with following hyper-parameters setting is run on the problem.

- **restarting points**: 10

Below is the achieved final result where max_iters is the maximum number of iteration and current_restart is the number start to convergence.

- **Iteration**: 64
- **Fitness**: 178
- **FEvals**: 3551
- **Time**: 4.285503
- **Restarts**: 10
- **max_iters**: $2^{10}$
- **current_restart**: 6

## 7.7 SA on Knapsack Optimization Problem

Simulated Annealing (SA) with following hyper-parameters is run on the problem.

- **temperature**: 6, 7, 8, 8, 9, 10
- **decay list**: Geometric & Exponential Decay

Following the procedure in subsection 2.2, below shows the configuration for best hyper-parameters. Below is the achieved final result.

- **Iteration**: 332
- **Time**: 0.084524
- **Fitness**: 180.0
- **FEvals**: 381.0
- **schedule_type**: geometric
- **Temperature**: 6

### 7.8 GA on Knapsack Optimization Problem

Genetic Algorithm (GA) with following hyper-parameters is run on the problem.

- **population_sizes**: $10, 20, 50$
- **mutation_rate**: $0.1, 0.3, 0.5$

Following the procedure in subsection 2.2, below shows the configuration for best hyper-parameters.

- **Iteration**: 18
- **Fitness**: 180.0
- **FEvals**: 974.0
- **Time**: 0.037694
- **max_iters**: $2^{10}$
- **pop_size**: 50
- **mut_rate**: 0.5

### 7.9 MIMC on Knapsack Optimization Problem

Mutual-Information-Maximizing Input Clustering (MIMC) with following hyper-parameters us run on the problem.

- **population_sizes**: 60, 70, 80, 100
- **keep_percent_list**: 0.1, 0.3, 0.5

Following the procedure in subsection 2.2, below shows the configuration for best hyper-parameters.

- **Iteration**: 8
- **Fitness**: 174.0
- **FEvals**: 640.0
- **Time**: 0.124586
- **pop_size**: 70.0
- **keep_percent**: 0.1

### 7.10 RHC on Flip Flop Optimization Problem

RHC with following hyper-parameters setting is run on the problem

- **restarting points**: 10

Below is the achieved final result where max_iters is the maximum number of iteration and current_restart is the number start to convergence.

- **Iteration**: 1906
- **Fitness**: 413.0
- **FEvals**: 10929.0
- **Time**: 3.917301
- **Restarts**: 10
- **max_iters**: 16384
- **current_restart**: 4

### 7.11 SA on Flip Flop Optimization Problem

Simulated Annealing (SA) with following hyper-parameters is run on the problem.

- **temperature**: 0.1, 2, 5, 10
- **decay list**: Geometric & Exponential Decay

Following the procedure in subsection 2.2, below shows the configuration for best hyper-parameters. Below is the achieved final result.

- **Iteration**: 332
- **Time**: 0.084524
- **Fitness**: 180.0
- **FEvals**: 381.0
- **schedule_type**: geometric
- **Temperature**: 6

### 7.12 GA on Flip Flop Optimization Problem

Genetic Algorithm (GA) with following hyper-parameters is run on the problem.

- **population_sizes**: 50, 70, 100, 150
- **mutation_rate**: $0.3, 0.5, 0.7$

Following the procedure in subsection 2.2, below shows the configuration for best hyper-parameters.

- **Iteration**: 1633
- **Fitness**: 463.0
- **FEvals**: 1709.0
- **Time**: 1.7088
- **max_iters**: 16384
- **pop_size**: 100
- **mut_rate**: $= 0.7$

## 7.13 MIMC on Flip Flop Optimization Problem

Mutual-Information-Maximizing Input Clustering (MIMC) with following hyper-parameters is run on the problem.

- **population_sizes**: 50, 70, 100, 150
- **keep_percent_list**: 0.3, 0.5, 0.8, 1

Following the procedure in subsection 2.2, below shows the configuration for best hyper-parameters.

- **Iteration**: 510
- **Fitness**: 397.0
- **FEvals**: 538.0
- **Time**: 0.00332
- **pop_size**: 150.0
- **keep_percent**: 0.5

## 7.14 NN on Bank Data

Neural networks (NN) with following hyper-parameters is run on the banks data from Assignment1.

- **max_iters** : 1000
- **learning rate**: 0.03
- **activation** : ReLU