CS 7641 RANDOMIZED OPTIMIZATION

by
Tahsin Ekram

INTRODUCTION

The analysis explores the impact of using multiple randomized optimization problems to solve discrete optimization problem. The 3 problems used here are N queens problem, flip flop problem and continuous peak problem. The randomized algorithms are applied individually to each problem and their performances are profiled and analyzed. In addition, the optimization algorithms are applied to optimize the weights of a neural network. The final classifier resulting from each algorithm is applied to a binary classification problem.

METHODOLOY

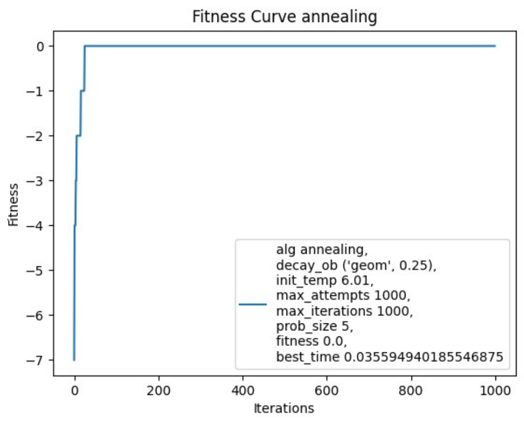
The data used for analysis was gathered by applying hyperparameter optimization to each algorithm. The algorithms and the discrete optimization problems was implemented using the mlrose python library. Hyperopt python library was used to define the parameter search space and tuning. There are 3 key parameters on which tuning was performed on for all problems across all algorithms. They are max iterations, max attempts and problem size. There are more algorithm specific parameters that were tuned but these are the common ones across all the algorithms.

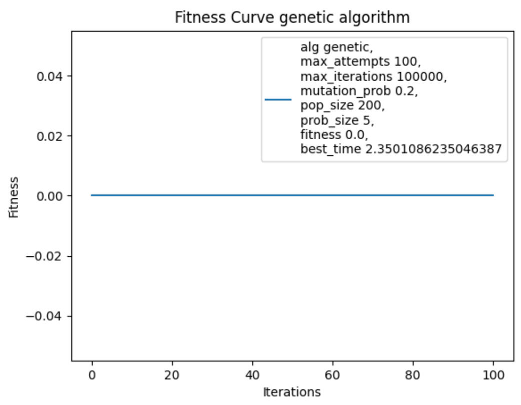
1. General parameters:
   1. max iteration: 10, 100, 1000, 10000, 100000
   2. max attempts: 10, 100, 1000, 10000
   3. problem size: 5, 25, 45, 65, 85, 105

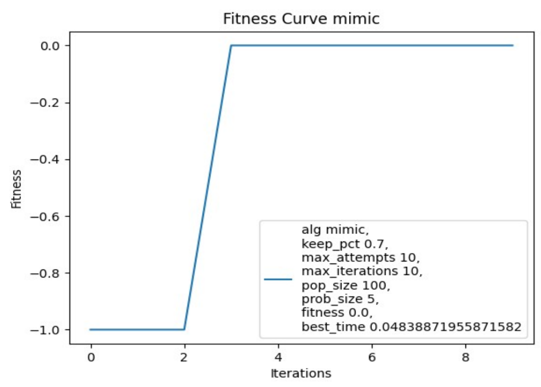
PROBLEMS

N-queens

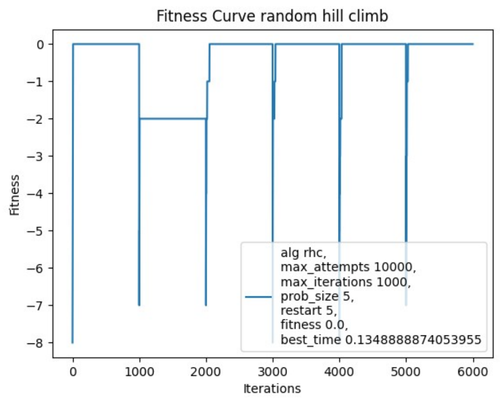
The n-queens problems tries to optimize the positioning of a certain number of queens on a chessboard such that none of them can attack each other. Thus the goal of this problem is to minimize the number of ways a certain number of queens can attack each other. As a result the a lower yield of the fitness function value represents a more optimized function.
The following graphs summarizes the results from hyperparameter tuning for each algorithm.





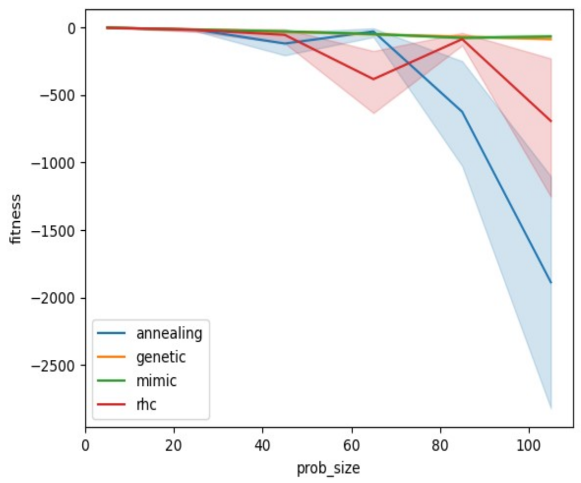


alg mimic,
keep_pct 0.7,
max_attempts 10,
max_iterations 10,
pop_size 100,
prob_size 5,
fitness 0.0,
best_time 0.04838871955871582



alg rhc,
max_attempts 10000,
max_iterations 1000,
prob_size 5,
restart 5,
fitness 0.0,
best_time 0.1348888874053955

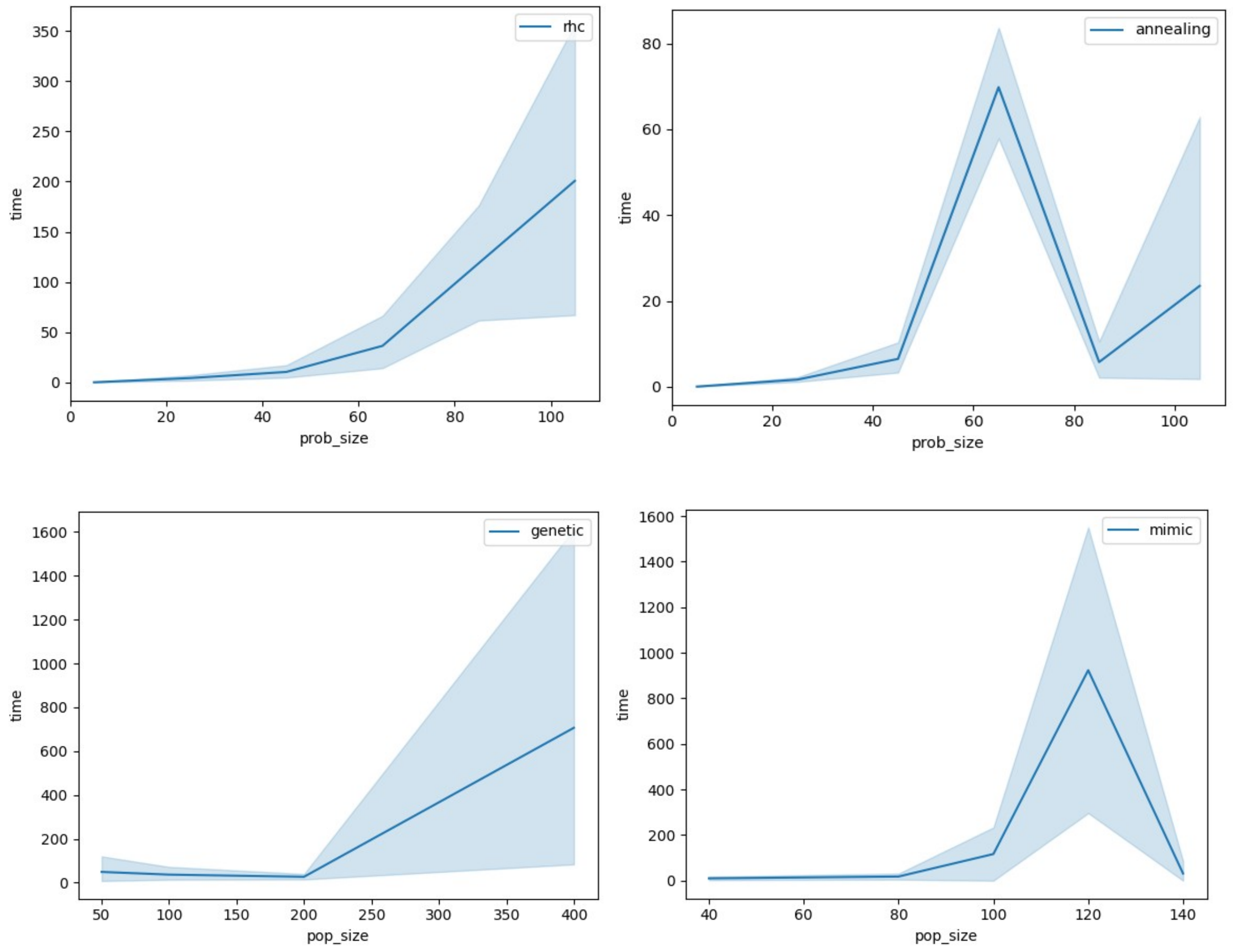
The above graphs outline the fitness curve for the best parameters from grid search for each algorithm. As seen above all these algorithms was able to achieve a fitness value of zero meaning they all found a state space where these queens don't attack one another. However, interestingly all these algorithm was able to to that for a very small problem size of 5. This is understandable as a lower number of queens allows more room for the algorithms to set it in such a way that no 2 queens attack each other. The graph below plots the relationship between the fitness and the size of problems which is the number of queens in this case.



This paints a clearer picture which we expect since increasing the number of queens will result in a more difficult problem where there may not be situation where no two queens attack each other. Furthermore as the problem size increases the overall search space goes up which makes it less likely and more difficult to find the global optima. This will be seen among all the problems discussed Another interesting point to note is the rate at which each algorithm drops in fitness with increasing problem size. Mimic and genetic algorithm as a very slight and steady reduction in fitness whereas random hill and simulated annealing shows a quite large drop.

For genetic algorithm the reason could be that it is less likely to fall into a local minima, because it moves abruptly, replacing parents with offspring that might be radically different. This allows it to find better solution however, at the cost of time which in this case is reflected by the high number of iterations. In mimic's case we see it mimic genetic algorithm. This is because of the underlying structure of mimic that makes it very similar to genetic algorithm. Mimic discards a set of observations based on certain fitness level when refining the probability function at each iteration. This is synonymous to the truncation strategy of genetic algorithm. Furthermore mimic's ability to capture structure by refining the underlying probability distribution of the problem lets it represent similar relationships to cross-over in genetic algorithms. When the problems you're trying to solve depend on relationships between features rather than specific feature values, MIMIC can handle them well; MIMIC does not care about “ties” in optima. Thus N queens seems like a great candidate where capturing structure will have a higher yield.

The following graph shows the relationship between time and problem size for each algorithm.

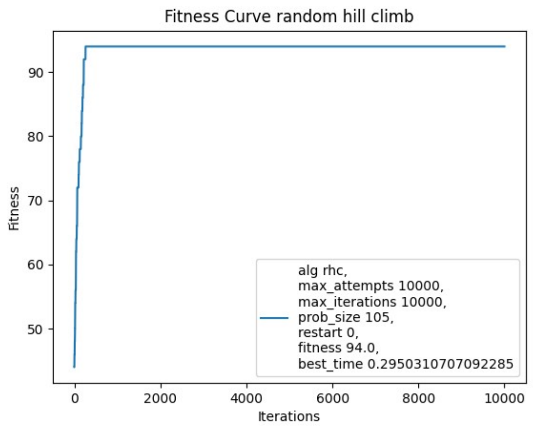


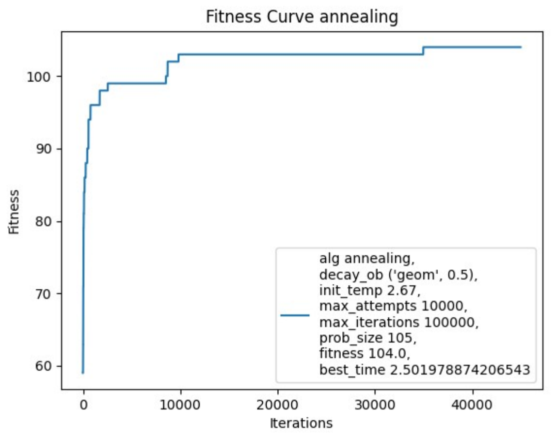
As we see above genetic algorithm and mimic in general takes more time given that both algorithms perform a type of computation on the larger population size. For annealing and random hill climb we see a lower time as they inject a certain level of randomness that allows them to capture a better optima faster. For annealing its transition from exploration to exploitation allows for that.

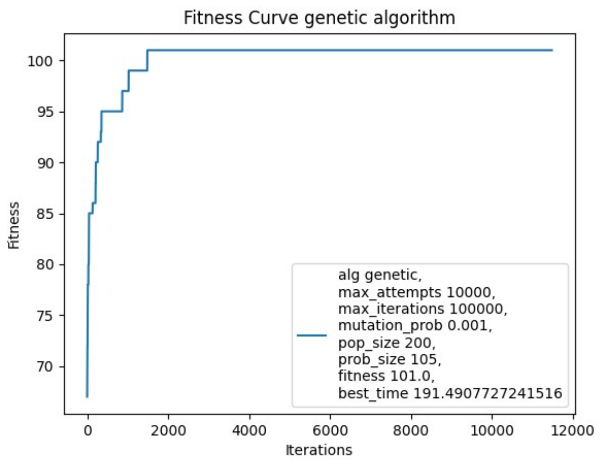
In summary genetic algorithm is more promising in handling the n queens problem. The key steps in GA is mutation and combination. With mutation GA allows the fitness function to perform a search similar to random hill climbing and yield a specific algorithm. Combination allows GA to take the best parts of these individual algorithms and combine them to come up with a better algorithm that is now able to solve multiple aspects of the problem.

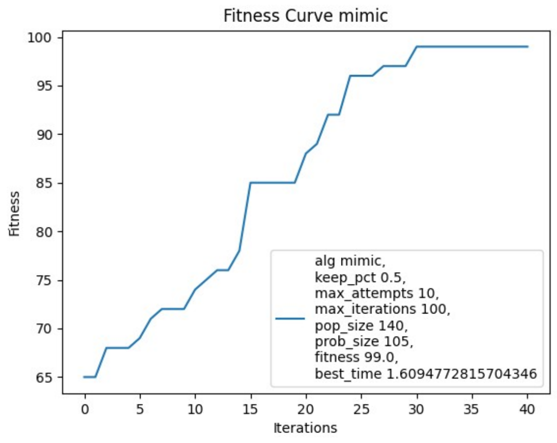
Flip flop problem

Flip-flop problem consists of consecutive bits and the fitness function is the sum of consecutive different bits. In the context of this problem maximum fitness is obtained by generating the highest number of alternating bit pattern across the entire input string. The following summarizes the results of each of the algorithms.







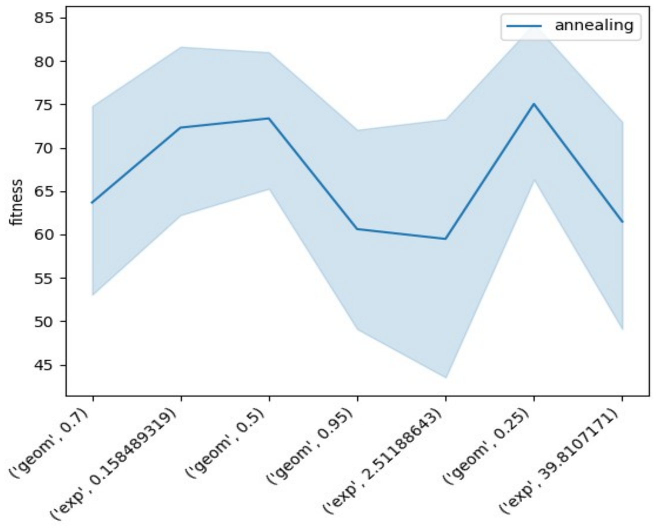


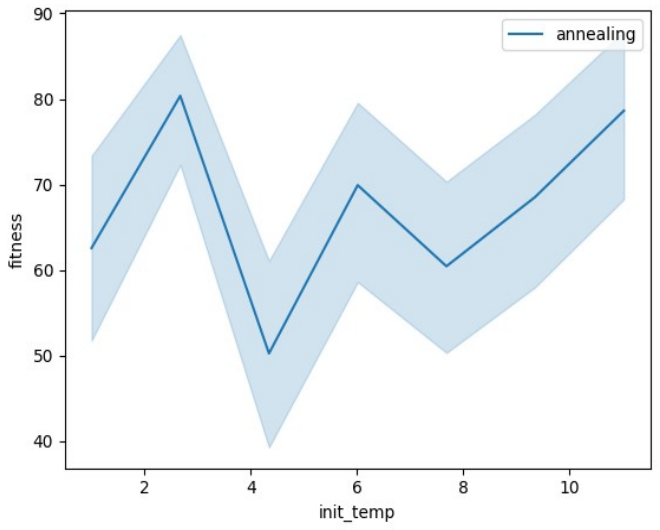
Based on the graph above the simulated annealing yields the best fitness. Given that the flip flop problem has a large number of local optimas RHC is more prone to be stuck in the basin it starts its' search thereby resulting in a lowest fitness value. Simulated annealing is able to address that by striking the balance between exploration and exploitation allowing it to explore other randomly chosen basins leading to better optimas.

Simulated annealing had the highest number of iterations. This could be due to the fact that SA starts of in the exploration phase taking most of the initial iterations aggressively search for the better optima and eventually slow down. On the other hand RHC had the lower number of iterations which further proves that it is getting stuck at the local optima.

For GA and MIMIC we see a considerably high fitness value however lower than SA. This could be due to the fact that the simplicity of the flip flop problem not being captured properly by the complex mutation and combination of GA and the structure capturing being done in mimic. Furthermore, given that there isn't much of a structure to this flip flop problem MIMIC's intensive computations to capture structure doesn't give it an edge over SA.

The following graphs outline the variation of the two hyper parameters for simulated annealing for this problem. Based on the graphs below we see that a higher initial temperature above 10 yields a higher fitness measure. This is expected as a higher initial temperature sets the algorithm to explore more initially which leads to finding the better optima.

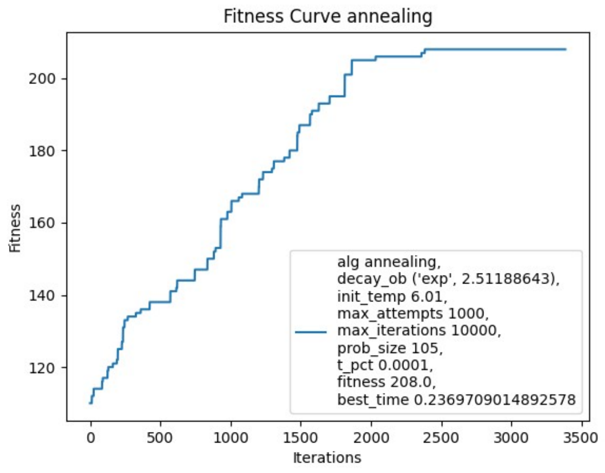




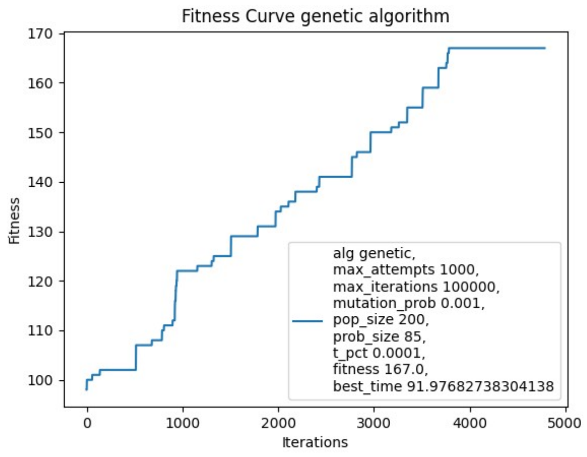
Continuous peak problems

The continuous peak problem generates higher fitness for bit strings that consist of consecutive 1s and 0s. It adds more the fitness if the maximum consecutive 1s and 0s is greater than the preassigned T value. T was one of the hyperparameter that us varied in the experiments for each algorithm. T manages the narrowness of the basin of attraction for finding the global optima.

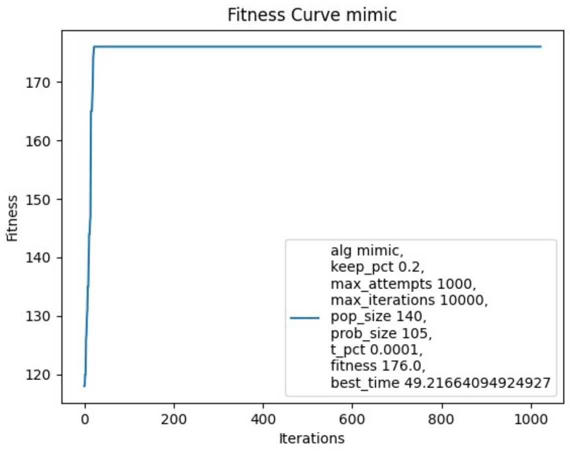
The following graphs summarizes the results for each algorithm.



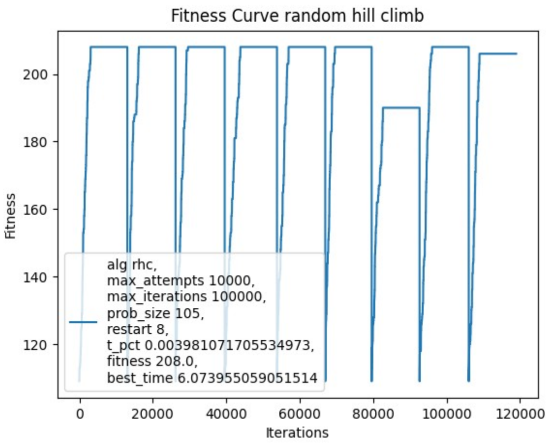
alg annealing,
decay_ob ('exp', 2.51188643),
init_temp 6.01,
max_attempts 1000,
max_iterations 10000,
prob_size 105,
t_pct 0.0001,
fitness 208.0,
best_time 0.2369709014892578



alg genetic,
max_attempts 1000,
max_iterations 100000,
mutation_prob 0.001,
pop_size 200,
prob_size 85,
t_pct 0.0001,
fitness 167.0,
best_time 91.97682738304138

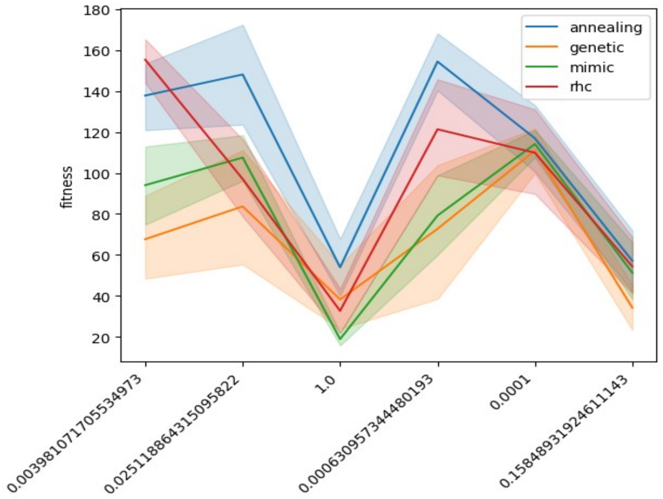


alg mimic,
keep_pct 0.2,
max_attempts 1000,
max_iterations 10000,
pop_size 140,
prob_size 105,
t_pct 0.0001,
fitness 176.0,
best_time 49.21664094924927



alg rhc,
max_attempts 10000,
max_iterations 100000,
prob_size 105,
restart 8,
t_pct 0.003981071705534973,
fitness 208.0,
best_time 6.073955059051514

According the graphs above it can be seen that all the algorithm chose a smaller T value in setting up the continuous peak problem. With a smaller T value this allow the problem to be modeled with wider basins of attraction which in turn helped in finding the global optima for which these algorithms can generate a high fitness value. The graph below highlights the relationship. As we can see all the algorithms prefers a smaller T value when designing the continuous peak problem.

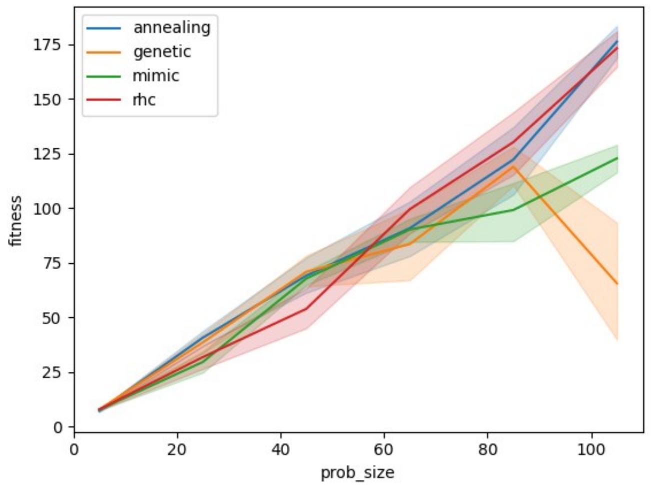


All the algorithms had the lowest yield at the highest T value as this generates very narrow basins of attraction making the task much more difficult. Among them SA still performed the best. This can be attributed to its exploration phase to not fall prey to those narrow basins.

\

In the graphs for mimic and GA above the best time parameter which represents the convergence time is the highest among all of the algorithms. This is due to the high amount of complex computation both the algorithms perform in the mutation and combination and creating the underlying distribution. Both their fitness are pretty similar but their convergence time of GA is much larger as the mechanism of combination and mutation is much more expensive.

Based on the following graph which shows the relationship between problem size and fitness it can be seen that MIMIC shows an increasing trend in fitness function similar to other algorithms for bit string size on the lower end. MIMIC’s ability to retain information about the problem space becomes the key factor in the increasing performance. On the other hand for GA we see a sudden drop beyond a certain bit string size. This was unexpected given that GA should be able to handle higher problem sizes given that its complex operations would allow it to efficiently search for a solution in a higher search space.



NEURAL NETWORKS

For this experiment SA, GA and RHC was used to optimize the weights of a neural network. It was compared with the baseline of using gradient descent. The hyperparameter search for this experiment is as follows:

1. learning rate: 1.00000000e-04, 6.30957344e-04, 3.98107171e-03, 2.51188643e-02,1.58489319e-01, 1.00000000e+00
2. hidden layers: [33],[16], [66],[33, 33][16, 16][66, 66][33, 33, 33][16, 16, 16],[66, 66, 66]

The dataset used to train and test the neural net is a classification problem called the ionoshpere dataset.

Algorithm	Learning rate	Hidden Layer	F1 score	Max iterations	Fit time
Gradient descent	1.0	(33,)	0.92	1000	0.538
Random hill climb	1.0	(33,)	0.90	10000	2.505
Genetic Algorithm	1.0	(33, 33, 33)	0.89	1000	0.438
Simulated annealing	1.0	(66,)	0.877	1000	1.145

The table above summarizes the f1 test score for these weight optimization algorithms.

GA is able to find a solution with lower iterations but it has lowest f1 score. This could be due the nature of GA in creating a complex representation of the problem which eventually overfits.

Gradient descent has performed the best out of all. This is because when it comes to weights of neural network we are working with continuous weights rather than discrete ones. This eventually makes the search space very large for these optimization algorithms which eventually results in narrower basins of attraction and thereby making it very difficult to find the global optima.

Further the f1 scores of the randomized optimizations are not too far off from gradient descent and is pretty competitive. This can be attributed to the dataset being used. In this case the ionosphere dataset contains a very small number of records. So even if the weight optimiztion problem is similar the dataset being small compensates for that by creating a somewhat smaller search space for these optimization to perform better and competitive with gradient desecent.