Assignment 2 (Randomized Optimization)

Cindy Nyoumsi
snyoumsi@gatech.edu

***Abstract*—** In this report, I will be implementing four (4) different local random search algorithms, namely randomized hill climbing, simulated annealing, a genetic algorithm and MIMIC on four optimization problems and comparing the results of each implementation on each problem.

1 OPTIMIZATION PROBLEMS DESCRIPTION

I will be exploring three optimization problems outlined below;

1.1 Knapsack Optimization Problem

For the first optimization problem, I will be looking at the Knapsack optimization problem.

This problem tries to optimize the combination of boxes of specific weights and specific values that can fit in a bag such that the total weight of all the items in the bag are not more than a constrained value X while still maximizing the combined value of all the boxes.

This problem is a really good problem to resolve using a Genetic Algorithm because we can easily represent this problem in a way that allows for us to easily leverage the main advantage of the genetic algorithm.

This problem is also a good one to highlight the strengths and weaknesses of the other random algorithms we will be using because it requires setting up different settings for the box weights and box values and thus makes it a complex problem that isn't easily solvable by simply covering all possible combinations.

1.2 N-Queens Optimization Problem

For the second optimization problem, I will be looking at the N-Queens optimization problem.

This problem tries to optimize for the best state on an arrangement of queens on a board such that no two queens are in an arrangement where they can attack each other.

This problem is a really good problem to resolve using Simulated Annealing because there are multiple states that can achieve the desired result, this means there is more than one global optima, and for an algorithm like simulated annealing that is both greedy and exploratory, this type of problem is easy to resolve.

This problem is also good to show the strengths of all the other randomized algorithms because they all share similar characteristics with Simulated Annealing exploratory and Greedy structure so I expect all of them to perform all relatively well on this problem.

1.3 Flip-Flop Optimization Problem

For the third optimization problem, I will be looking at the Flip-Flop optimization problem.

This problem tries to optimize the number of bit alternations in a string of bit such that the best state is the bit string combination where every bit is alternated.

This problem is an excellent problem to resolve using the MIMIC algorithm because we can easily build to the top from an unfit starting population of string bits to a population of string bits that contain the most alternations between bits.

This problem is also good at showing the weakness of more greedy algorithms like Random Hill Climb and I think it will be interesting to see how the other algorithms that share characteristics in between RHC and MIMIC perform in this instance.

2 ALGORITHM PERFORMANCES

For each optimization problem, I will be evaluating how well each randomized algorithm performs by comparing the following:

1. Fitness Value against Iterations
2. Max Fitness values
3. Runtimes

2.1 Knapsack Optimization Problem

I set up the Knapsack problem with a length N of 200.

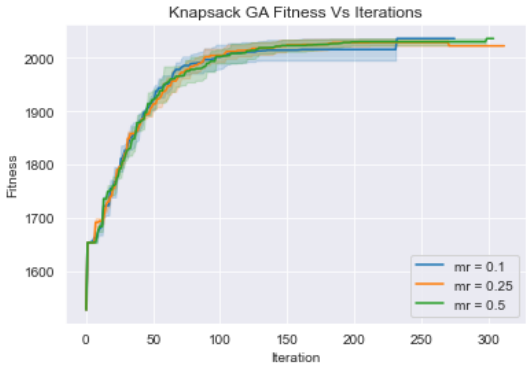
I then ran each algorithm with respect to each of their parameter variations at 10,000 max iterations each and plotted the different fitness values for each parameter set up against the number of iterations for each parameter.

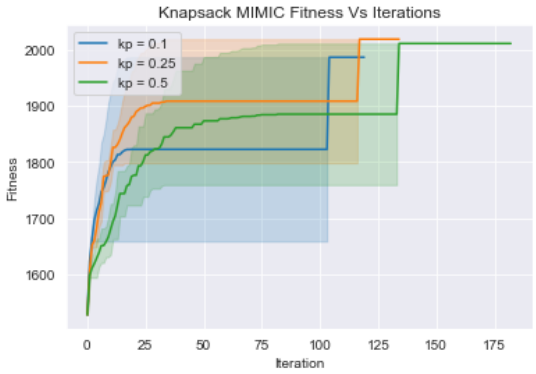
Looking at the charts below for fitness vs iterations for these algorithms, we can easily see the parameters that produce the best fitness values for each algorithm.

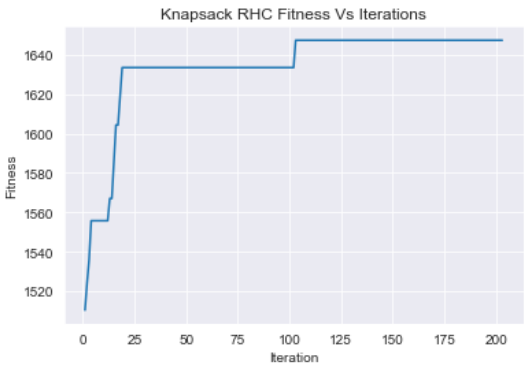
I have also put the results in the table below:

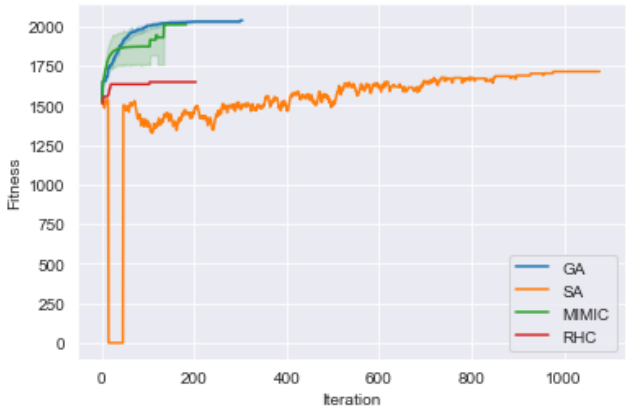
Algorithm	**Parameter**	**Best Parameter Value**	**Best Fitness Value**	**Runtime (secs)**
Genetic Algorithm	Mutation Rate, Population Size	0.5, 500	2036.04	33.7

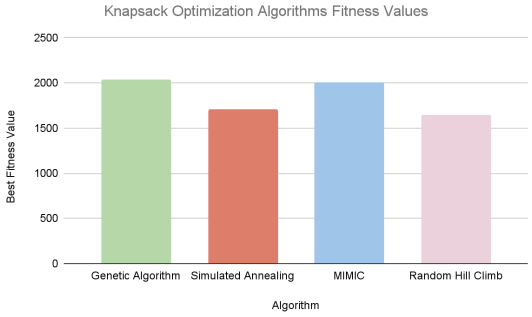
Simulated Annealing	Temperature	500	1713.61	1.84
MIMIC	Split Percentage, Population Size	0.25, 500	2004.96	243
Random Hill Climb	N/A	N/A	1647.300	51.1

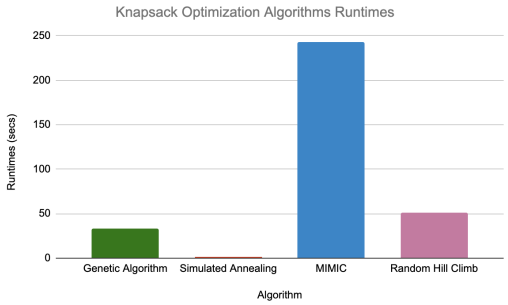












Looking at the charts and results in the table above, we can see that the Genetic Algorithm performs the best on the Knapsack problem in terms of both best highest fitness value and runtimes. Despite Simulated Annealing having the shortest runtime, it's fitness value is the poorest due to the fact that it mostly explores around and while the cooling rate helps make it more greedy in a problem like Knapsack it settles faster for a local optimal state. Despite MIMIC having fitness value as high as the genetic algorithms, it's runtime is the worst and by a very large gap at that, I think this is because the MIMIC algorithm does a lot more exploration than GA, in addition to all the calculations necessary to identify the fittest group to continue with thus it takes longer runtime wise.

Finally random Hill Climb seems to on average be the second best given it's fair balance of runtime and fitness value also given the fact that it barely necessitated any kind of parameter setup unlike genetic algorithms.

My conclusions for why the Genetic Algorithm performs the best on the Knapsack problem is because it leverages the ability to learn from the data as it explores different points while continuously improving it's chances of finding the best fitness weight and value combination for the bag through crossover.

2.2 N-Queens Optimization Problem

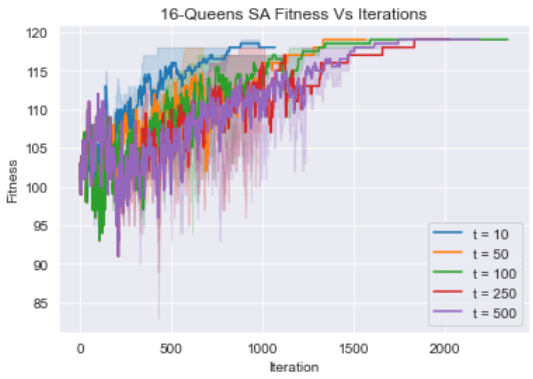
I set up the N-Queens problem to resolve a board with 16 queens.

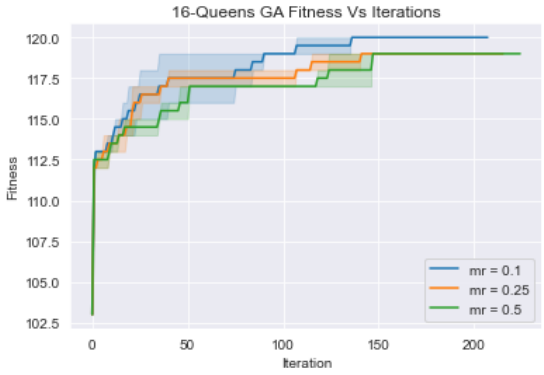
I then ran each algorithm with respect to each of their parameter variations at 10,000 max iterations each and plotted the different fitness values for each parameter set up against the number of iterations for each parameter. I used the same parameters for the optimization problem object to easily compare across the board.

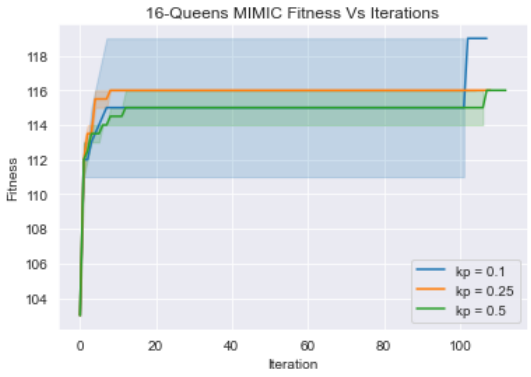
Looking at the charts below for fitness vs iterations for these algorithms, we can easily see the parameters that produce the best fitness values for each algorithm.

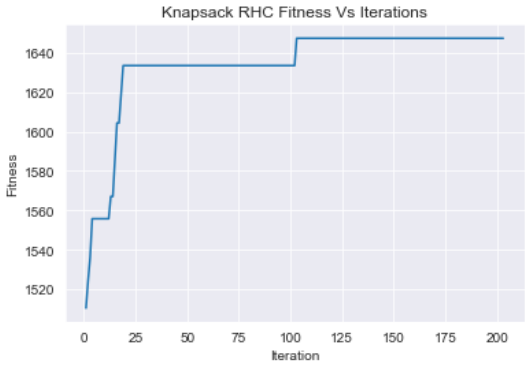
I have also put the results in the table below:

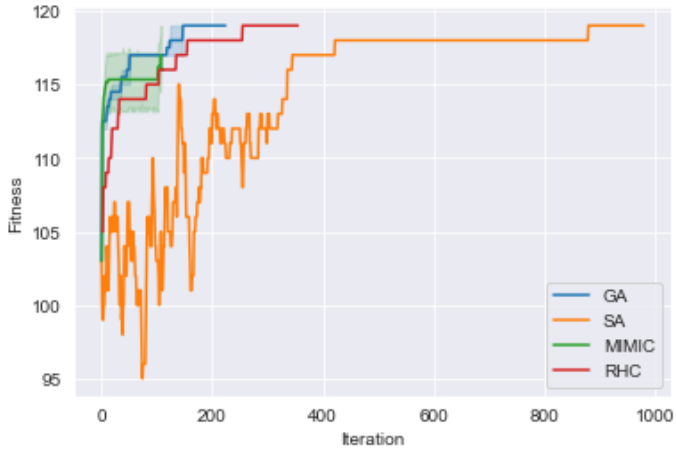
Algorithm	**Parameter**	**Best Parameter Value**	**Best Fitness Value**	**Runtime (secs)**
Simulated Annealing	Temperature	100	119.0	1.34
Genetic Algorithm	Mutation Rate, Population Size	0.1, 500	120	71
MIMIC	Split Percentage, Population Size	0.25, 500	117	57.8
Random Hill Climb	N/A	N/A	119	192

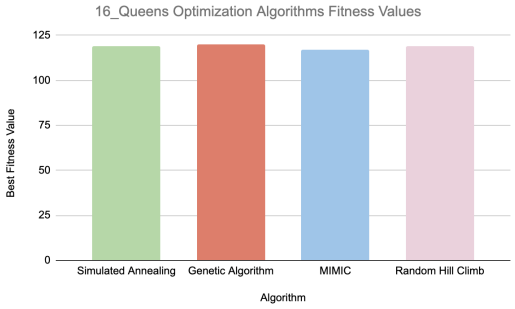


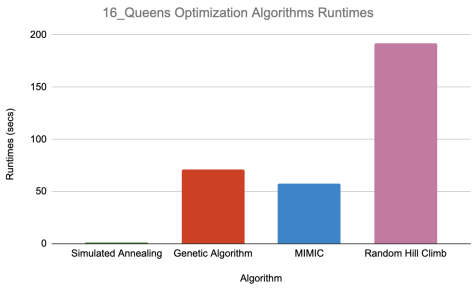












From the charts and results in the table above, we can see that the Simulated Annealing Algorithm performs the best on the N-Queens problem when looking at the combination of highest fitness value and runtimes as it has the second highest score with the smallest runtime.. A problem like N-queens is well suited for random optimization algorithms because it has multiple optimal states that maximize the fitness value and so this gives random algorithms more chances to reach such maximizing fitness value functions because they randomly search around the function.

Finally random Hill Climb seems to have the worst runtime on this problem most likely due to the fact that it is the purest random search and does not learn so it takes a while for it to reach the best fitness value for this type of constrained problem.

2.3 FlipFlop Optimization Problem

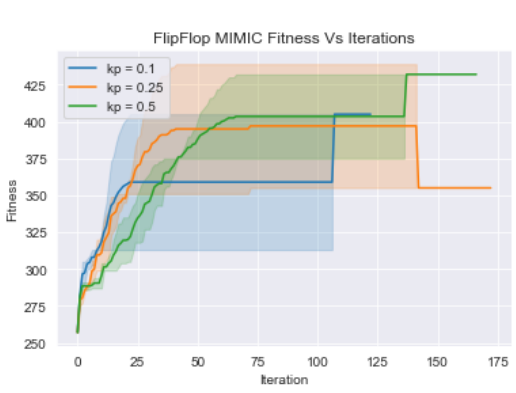
I set up the FlipFlop problem to resolve a board with 16 queens.

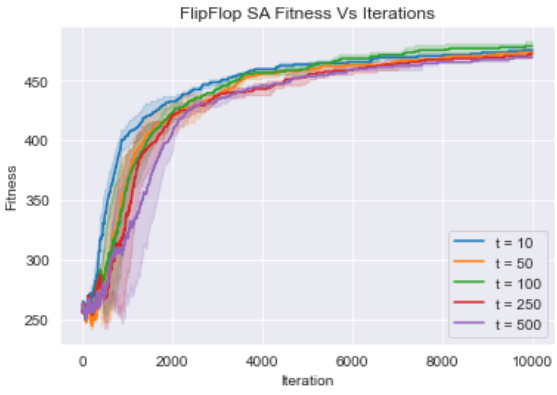
I then ran each algorithm with respect to each of their parameter variations at 10,000 max iterations each and plotted the different fitness values for each parameter set up against the number of iterations for each parameter. I used the same parameters for the optimization problem object to easily compare across the board.

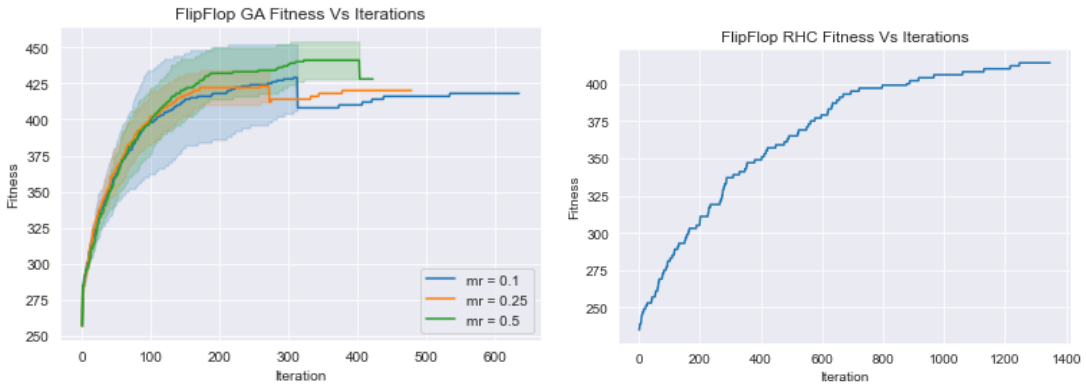
Looking at the charts below for fitness vs iterations for these algorithms, we can easily see the parameters that produce the best fitness values for each algorithm.

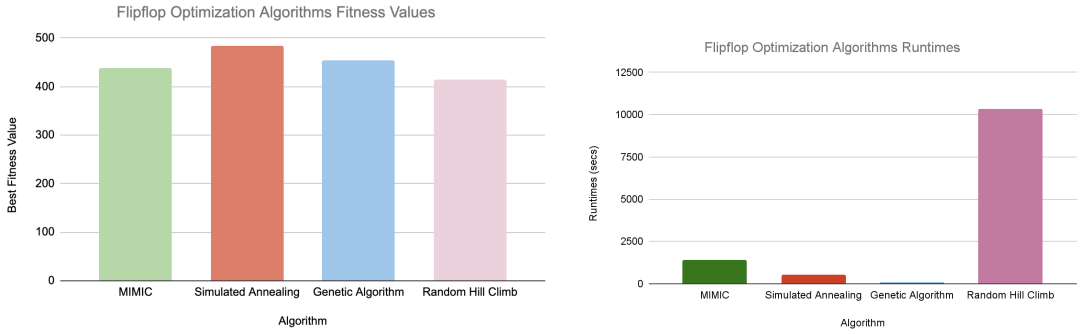
I have also put the results in the table below:

Algorithm	**Parameter**	**Best Parameter Value**	**Best Fitness Value**	**Runtime (secs)**
MIMIC	Split Percentage, Population Size	0.5, 500	439	1417
Simulated Annealing	Temperature	100	483	519
Genetic Algorithm	Mutation Rate, Population Size	0.5, 500	454	62
Random Hill Climb	N/A	N/A	414	10352









From the charts and the results in the table above, it is clear to see that once more SA is the best performing algorithm this time given it has the highest fitness value and second best runtime. I do want to note though that MIMIC performed better both in fitness value rank and runtime rank in this Flip Flop problem compared to the other two optimization problems.

I didn't expect simulated annealing to do so well in this problem but the more I think about it,I can see that the Flip Flop problem is easy enough such that there are many easy opportunities to reach the global optima.

I expected the Random Hill Climb to perform terribly on this type of problem given that such a large population will take a while to get through before reaching the optima given all the possible combinations that are possible in the Flip Flop problem, but I did not expect it to take so much longer compared to the other algorithms taking up to two hours while the other algorithms merely took no longer than 30mins. .

3 NEURAL NETWORK WEIGHT OPTIMIZATION

For the Neural Network weight optimization problem I will be trying to classify adults into risky vs non-risky lenders using demographic and payment history data.

This might be useful/interesting in a customer segmentation analysis where as a data science consultant, I want to help a loan company identify how risky their lending portfolio client base is.

I will be using the credit dataset made up of 24 attributes with 30000 instances.

6 of the attributes are demographic such as age, gender, marital status, while the remaining 18 attributes are monthly payment history.

The credit default attribute was already binary so no major pre-processing was required.

The next steps I took involved using three different randomizations algorithms to find the best weights for the neural network before making predictions.

3.1 Implementation of randomized optimization algorithms

I used RHC, SA and to find the optimal weights for the neural networking on the credit dataset and used the same parameters set up for each optimization for easier comparison.

I then compared the performance of each model by looking at their F1 score and runtime as shown in the charts below.

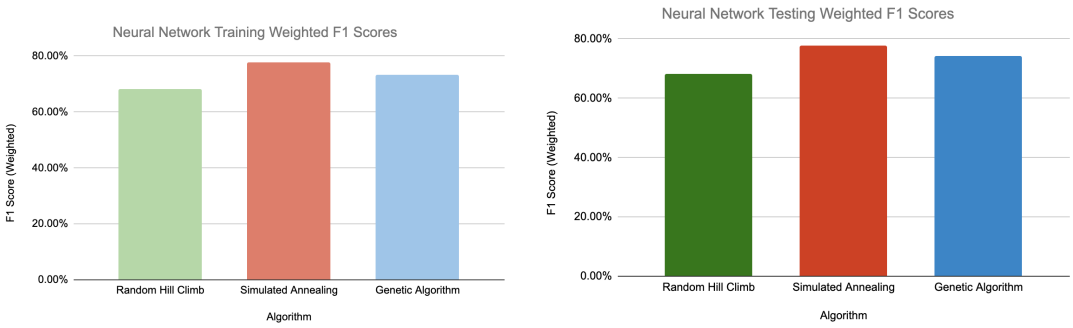
Looking at the results from the charts below, it seems like the training and testing rates are very close and even equal in fact for the RHC.

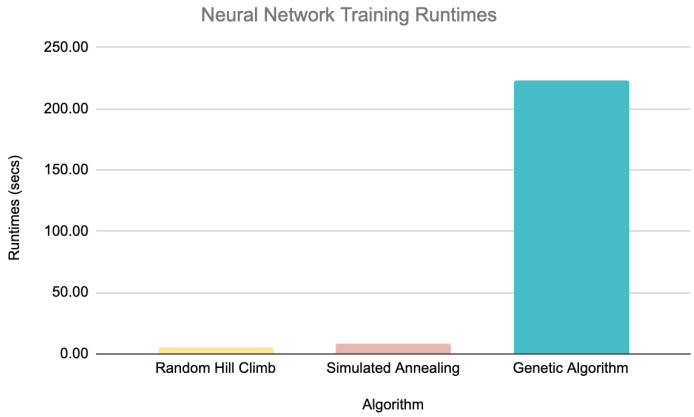
I found three things very interesting: (1) from Figure 5a, the training model and cross validation model showed similar convergence paths; (2) from Figure 5b, GA showed a larger volatility in F1 score evaluation; and (3) backpropagation, RHC and SA all converged at 3,000

iterations, but it seems GA needs more iterations to converge and backpropagation converges fastest with RHC being the second.

Additionally Simulated Annealing shows the best performance here with highest F1 score of ~78% and second highest runtime of only ~8 seconds, which is consistent with its performance on other well aligned optimization problems such as the N-queens discussed earlier.

Another thing that was interesting was seeing how much longer it took for the Genetic Algorithm to find the optimal weights, going into three minutes compared to Simulated annealing and RHC whose runtime were both less than 10 seconds. One reason for that is because finding optimal weights for a neural network isn't easily mutable and so it takes a lot longer to run.





Overall, compared to running the Neural Network without these randomized algorithms, they all performed better when looking at their respective f1 scores although they took slightly longer in runtime seconds.

7 REFERENCES

1. UCI Credit Card Data Set from Taiwan with history payments and binary: https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients
2. 8-Queens Example for Randomized Optimization in Python https://towardsdatascience.com/getting-started-with-randomized-optimization-in-python-f7df46babff0
3. Mlrose Optimization tutoring using NQueens problem https://mlrose.readthedocs.io/en/stable/source/tutorial1.html#select-and-run-a-randomized-optimization-algorithm
4. Implementation and comparison of randomized optimization algorithms https://medium.com/@duoduoyunnini/introduction-implementation-and-comparison-of-four-randomized-optimization-algorithms-fc4d96f9feea
5. Genetic Algorithms part 3 , specifically looking at the Knapsack problem https://medium.com/koderunners/genetic-algorithm-part-3-knapsack-problem-b59035ddd1d6