Assignment 2 Randomized Optimization

Lujia Zhang
CS7641
Lzhang341@gatech.edu

1. Introduction

The purpose of this assignment is to explore 3 discrete optimization problem

1. N-Queens
2. Flip-flop
3. Knapsack

implement 4 local random search algorithms

1. Randomized Hill Climbing
2. Simulated Annealing
3. A genetic Algorithm
4. MIMIC

Outside of exploring 3 discrete optimization problem, Randomized Hill Climbing, Simulated Annealing and A genetic Algorithm are used to find good weights for my assignment 1 neural network's implementation.
Implementation of the code can be found at: https://github.com/swagluke/CS7641-HW2

2. N-Queens

N-Queens is a well-known NP-complete problem of coming up with strategy to place N queen chess pieces on a n by n chessboard with the constraint of none of the 2 queens can attack each other. [1] N-Queens problem is interesting because a queen has tons of possible attack moves horizontally, vertically and diagonally on a board. It's hard for human to consider all possible attack moves even with a small amount of queens on the chess board. Adding extra queen dramatically increases the problem difficulties since it generates tons of possible attack moves. Having a queen in a different spot will also dramatically change the fitness score coming out of fitness function.

N-Queens problem is great to test out all 4 algorithms' strengths and weakness because all algorithms will try to build on current best solution or pass down the most useful information to next iterations. However, most optimal solution might look completely different from current best solution.

2. 1. Set up

I decided to test out the algorithm with 32 queens, which represents 528 possible attacks.

2.2. Fitness Function

Same as the constraint of the N-Queens problem, minimizing the number of queens attacking each other, which can also be represented as maximizing the number of queens not attacking each other. With 32 queens and 528 possible attacks, that means our max fitness value for fitness function is 528.

2.3. Randomized Hill Climbing

Randomized Hill Climbing is a randomized heuristic search algorithm that helps to find the best possible solution to the problem. [2]

It evaluates best routine out of random path from current state to improve current best solution.

For RHC parameter, I set the iteration_list as 10000, max_attempts as 100 and restart_list as 100.

This algorithm executes a total of 34425 iterations with the run time Figure 1 shows below.

```
Run time: 72.67841073799991
CPU times: user 1min 12s, sys: 369 ms, total: 1min 13s
Wall time: 1min 12s
```

Figure 1: RHC Run time- N Queens

The Average Fitness value is 488.3960396039604, the Max Fitness value is 494.0. Figure 2 shows the Learning Curve for Best Randomized Hill Climbing Random Search below.

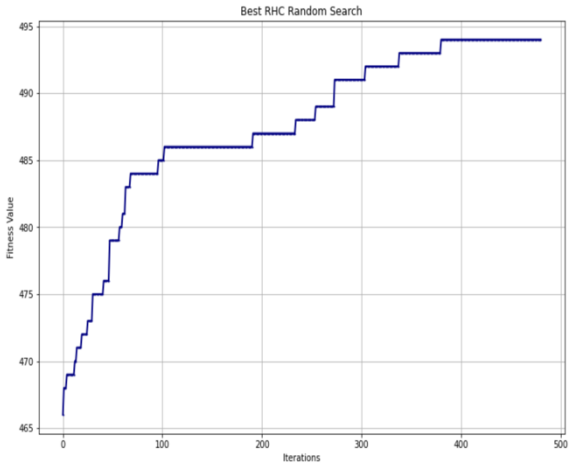


Figure 2: Best RHC Random Search Learning Curve- N Queens

It's interesting to see the learning curve somewhat looking like staircases. Due to the parameter setting, it's possible that some iterations don't present useful random paths. Therefore, at those iterations, the algorithm didn't search randomly enough, so it's sticking with the current best solution (local optimal solution).

Unfortunately, even with the best random search, we can see that the theoretical max fitness value 528 isn't achieved. Possible changes that might help with the results are increasing max attempts and restart list so RHC will search more randomly and search more times during iterations.

2.4. Simulated Annealing

Simulated Annealing is a probabilistic technique to find the optimal value of a function. It simulates the nature behavior of heating and cooling solid. [3]

For Simulated Annealing's parameter, I set the iteration_list as 10000, temperature_list with 9 different values as [1, 10, 50, 100, 250, 500, 1000, 2500, 5000], decay_list with two different functions ExpDecay, and GeomDecayand and max_attempts as 100.

This algorithm executes a total of 40418 iterations with the run time Figure 3 shows below.

```
Run time: 43.95296914700003
CPU times: user 44 s, sys: 295 ms, total: 44.3 s
Wall time: 44 s
```

Figure 3: Simulated Annealing Run time- N Queens

The Average Fitness value is 493.72222222222223, the Max Fitness value is 495.0, Figure 4 below shows the Learning Curve for Best Simulated Annealing.

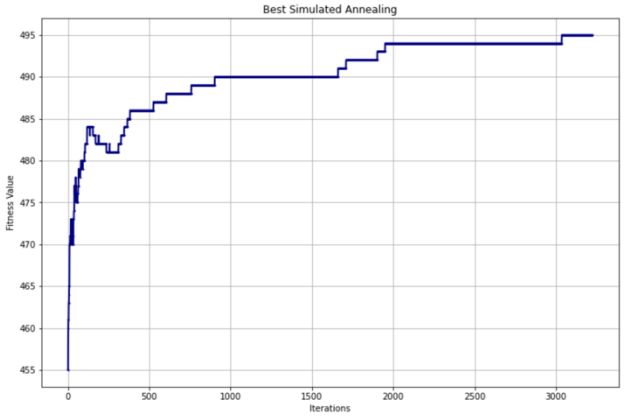


Figure 4: Best Simulated Annealing Learning Curve- N Queens

The best result comes from the ExpDecay as the decay function with init_temp as 1, exp_const as 0.05 and min_temp as 0.001.

By comparing the best RHC and best Simulated Annealing Learning Curve, you can see that Simulated Annealing runs a lot more iterations and achieved a slightly better result than best RHC. In a way, if we only look at iterations less than 500, Simulated Annealing actually produces worse result than RHC. But it's worth to mention that Simulated Annealing require way less time to run compare to RHC, save almost 40% of the time. I suspect the quick run time has something to do with the simplicity of heating and cooling instead of random search.

Possible changes that can help with the results are trying different temperature value and decay functions since it allows the algorithm to heat from a different temperature. Different decay function will also generate different results.

2.5. Genetic Algorithm

Genetic algorithm is random-based search heuristic. The whole idea came from Charles Darwin's theory of natural evolution, which is strongest survival. It believes that the strongest value will be passed down through multiple iterations in order to produce the best result.[4]

For Genetic Algorithm's parameter, I have iteration_list as 100000, max_attempts as 100, 3 different population_sizes as 50, 200, 500, and 3 different mutation_rates as 0.1, 0.25, 0.5, that makes a total of 9 combinations, which represents 9 different Genetic Algorithms.

This algorithm executes a total of 2605 iterations with the run time Figure 5 shows below.

```
Run time: 572.172726852
CPU times: user 9min 31s, sys: 170 ms, total: 9min 32s
Wall time: 9min 32s
```

Figure 5: Genetic Algorithm Run time- N Queens

For population size 50, it took 9.049084 sec to run, population size 200 took 40.778382 sec to run and population size 500 took the most 141.877683 sec to run. It doesn't surprise me that as population size increases, the run time increases due to the more complex evolution within the group.

The Average Fitness value is 493.0, the Max Fitness value is 495.0, Figure 6 below shows the Learning Curve for Best Genetic Algorithm.

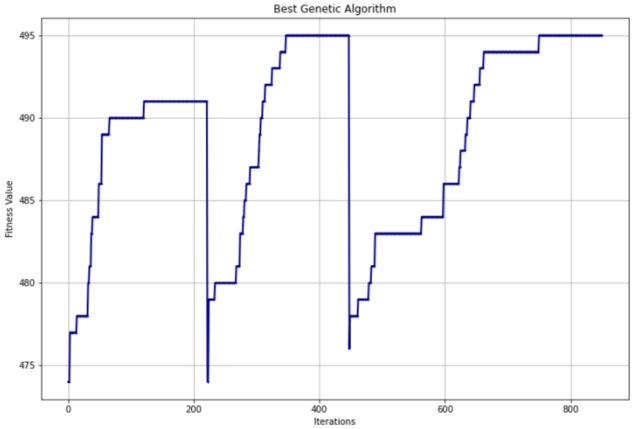


Figure 6: Best Genetic Algorithm Learning Curve- N Queens

This learning curve looks weird to me, since by theory genetic algorithm will pass down the best possible genetic value to the next iterations, however, we see big fitness value drop throughout the multiple iterations. I suspect the algorithm ran into situation where some big mutations happen during some iterations to cause fitness value drop in a short term, however, it improves the fitness values in a long term.

Compare to RHC and Simulated Annealing, Genetic Algorithm produces all most the same results but took much longer time to run.

Possible changes that can help with the results are trying different population size and mutation rate. It allows Genetic algorithm to have different evolution that generates different possible genetic information.

2.6. MIMIC

Mutual-Information-Maximizing Input Clustering (MIMIC) is an optimization algorithm that passes the most useful information about cost function between iterations. By continuing building upon those information, it's able to produce the best result for the problem.[5]

For MIMIC Algorithm's parameter, I have iteration_list as 100000, max_attempts as 100, 3 different population_sizes as 50, 200, 500, and 3 different keep_percent_list as 0.25, 0.5, 0.75, that makes a total of 9 combinations, which represents 9 different MIMIC.

This algorithm executes a total of 974 iterations with the run time Figure 7 shows below.

```
Run time: 625.5432375299999
```

Figure 7: MIMIC Run time - N Queens

For population size 50, it took 22.026153 sec to run, population size 200 took 53.626115 sec to run and population size 500 took the most 132.846246 sec to run. This follows the same logic as the Genetic algorithm run time, as complicity increases, it's normal to cost more time to run the algorithm.

The Average Fitness value is 479.4444, the Max Fitness value is 484.0, Figure 8 below shows the Learning Curve for Best MIMIC.

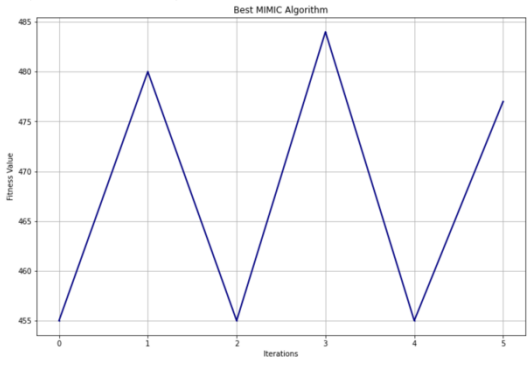


Figure 8: Best MIMIC Learning Curve – N Queens

I found MIMIC's result to be disappointing since it took the longest time to run but didn't achieve the best results as the other 3 algorithms (495 being the best fitness score those 3 algorithms produce). The learning curve follow the similar patterns as the Genetic algorithm with sudden drop in fitness score after some period of training but gain fitness score after the drop. I suspect this behavior is caused by dropping too much useful information. During some iterations, the algorithm can no longer hold any more useful information while accepting new information, therefore it has to drop some old information from previous iterations.

Possible changes to make this algorithm better will be increase the population size and keep percent, which allows MIMIC to maintain more information and keep more useful information.

2.7. N- Queens Conclusion

	Max Fitness	Mean Fitness	Run Time (sec)
RHC	494	488.3960396039604	72.58
Simulated Annealing	495.0	493.72222222222223	43.95
Genetic Algorithm	495.0	493.0	527.17
MIMIC	484.0	479.4444	625.54

Simulated Annealing is the winner algorithm here for sure since it not only produces the best results in max fitness score, mean fitness score and it's also taking the least amount of time to run. I've explained the reason behind simulated annealing's high performance in section earlier.

3. Flip-flop

Flip-flop is a circuit with two stable states 0 and 1. [6]. By flipping bit, the circuit state will be changed. Therefore, it's interesting to see how many different bits can be changed in order to get different states. This problem is able to test out the strengths and weakness of all 4 algorithms because the fitness function for each bit is related to the neighbor bit. So the neighbor bit information will play a big part in the performance.

3. 1. Set up

I decided to test out the algorithm with a circuit with 250 bits to test out the algorithms.

3.2. Fitness Function

The goal fitness function is to add up all the different consecutive bits from the circuit and represent the sum. Since I have a circuit with 250 bits, that makes our max fitness value for fitness function is 249.

3.3. Randomized Hill Climbing

For RHC parameter, I set the iteration_list as 10000, max_attempts as 100 and restart_list as 100.

This algorithm executes a total of 54553 iterations with the run time Figure 9 shows below.

```
Run time: 174.97199121300036
CPU times: user 2min 54s, sys: 987 ms, total: 2min 55s
Wall time: 2min 55s
```

Figure 9: RHC Run time – Flip flop

The Average Fitness value is 197.75247524752476, the Max Fitness value is 214.0. Figure 10 shows the Learning Curve for Best Randomized Hill Climbing Random Search below.

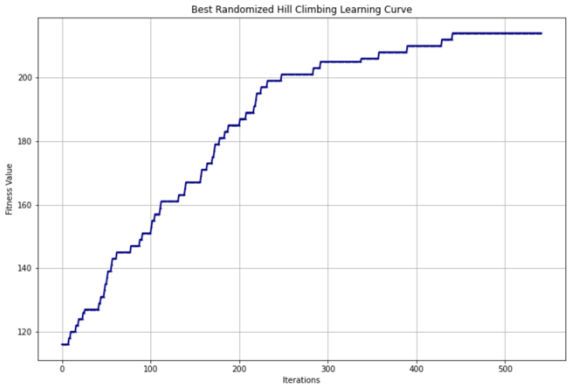
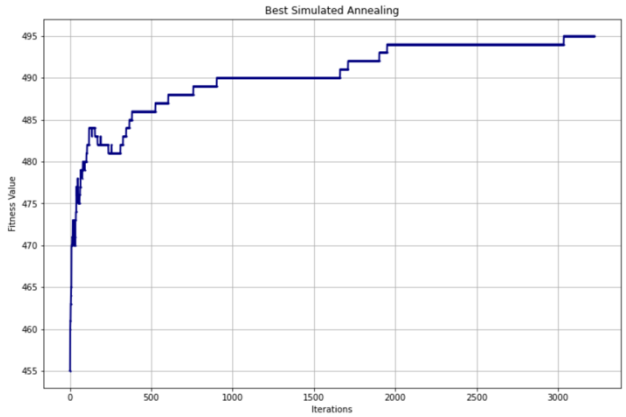


Figure 10: Best RHC Random Search Learning Curve- Flip flop

Similar Learning Curve pattern like the N-Queens problem, result is decent but not achieving the theorical max Fitness score of 249. Again, having more max attempts should help with the results since it will search more bits and try to get better results.

3.4. Simulated Annealing

For Simulated Annealing's parameter, I set the iteration_list as 10000, temperature_list with 5 different values as [1, 10, 50, 100, 250], decay_list with two different functions ExpDecay, and GeomDecayand and max_attempts as 100.



This algorithm executes a total of 153888 iterations with the run time Figure 11 shows below.

```
Run time: 236.686894157001
CPU times: user 3min 56s, sys: 1.05 s, total: 3min 57s
Wall time: 3min 56s
```

Figure 11: Simulated Annealing Run time- Flip flop

The Average Fitness value is 242.9, the Max Fitness value is 246.0, Figure 12 below shows the Learning Curve for Best Simulated Annealing.

Figure 12: Best Simulated Annealing Learning Curve- Flip flop

The best result comes from the ExpDecay as the decay function with init_temp as 50, exp_const as 0.05 and min_temp as 0.001.

Similar results like the N-Queen problem but one interesting thing here is unlike the N-Queen problem where Simulated Annealing runs much faster than RHC with similar result, with Flip flop, Simulated Annealing runs much slower than RHC. I think having a different temperature will help with results.

3.5. Genetic Algorithm

For Genetic Algorithm's parameter, I have iteration_list as 100000, max_attempts as 100, 3 different population_sizes as 20, 50, 100, and 3 different mutation_rates as 0.1, 0.25, 0.5, that makes a total of 9 combinations, which represents 9 different Genetic Algorithms.

This algorithm executes a total of 3473 iterations with the run time Figure 13 shows below.

```
Run time: 16.6583642530004
CPU times: user 16.6 s, sys: 37.9 ms, total: 16.7 s
Wall time: 16.7 s
```

Figure 13: Genetic Algorithm Run time- Flip flop

For population size 20, it took 1.431111 sec to run, population size 50 took 1.651666 sec to run and population size 100 took the most 2.435793 sec to run. It doesn't surprise me that as population size increases, the run time increases due to the more complex evolution within the group.

The Average Fitness value is 222.0, the Max Fitness value is 206.44444444444446, Figure 14 below shows the Learning Curve for Best Genetic Algorithm.

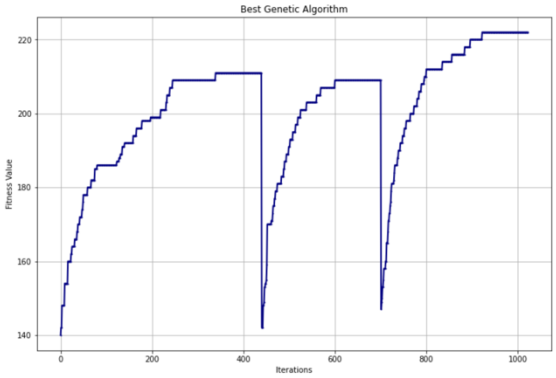


Figure 14: Best Genetic Algorithm Learning Curve- Flip flop

Not surprise to see the learning curve behaving like the one from N-Queens.

But I'm super surprised by the run time. In N-Queen, Genetic Algorithm is almost the slowest one. But here, it's the fastest among RHC, Simulated Annealing.

My guess is since circuit only has 2 states for each bit, therefore it's easy to pass down information to next generations for Genetic Algorithm. Improvement can be made by increasing population size since it helps with the Genetic algorithms' mutation and combination.

3.6. MIMIC

For MIMIC Algorithm's parameter, I have iteration_list as 100000, max_attempts as 100, 3 different population_sizes as 20, 50, 100, and 3 different keep_percent_list as 0.25, 0.5, 0.75, that makes a total of 9 combinations, which represents 9 different MIMIC.

This algorithm executes a total of 1318 iterations with the run time Figure 15 shows below.

```
Run time: 198.19088762200045
```

Figure 15: MIMIC Run time – Flip flop

For population size 20, it took 9.549652 sec to run, population size 50 took 18.125508 sec to run and population size 100 took the most 38.365220 sec to run.

The Average Fitness value is 172.44444444444446, the Max Fitness value is 198.0, Figure 16 below shows the Learning Curve for Best MIMIC.

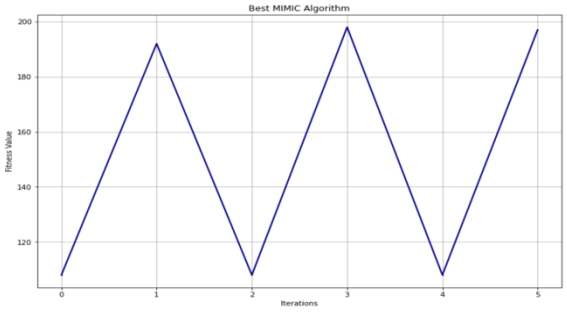


Figure 16: Best MIMIC Learning Curve – Flip flop

I found MIMIC's result to be disappointing again since it didn't produce good fitness score while costing a long time to run.

3.7. Flip-flop Conclusion

	Max Fitness	Mean Fitness	Run Time (sec)
RHC	214.0	197.75247524752476	174.9719
Simulated Annealing	246.0	242.9	236.686
Genetic Algorithm	222.0	206.44444444444446	16.7
MIMIC	198.0	172.4444	198.19

Simulated Annealing again outperformed all other 3 algorithms, however it is the most time consuming algorithm. Genetic Algorithm on the other hand generated decent result with a super-fast run time. I've stated the possible high performance reason in the Simulated Annealing and Genetic Algorithm section.

4. Knapsack

Knapsack is a well-known combinatorial optimization problem. With limited knapsack capacity, it tries to pack as many valuable items as possible into a knapsack by properly evaluate the item weight and knapsack capacity[7]

Knaspack will test the strength and expose the weakness of all 4 algorithms since it contains many possible local optimal solutions due to it's NP-Complete nature.

4. 1. Set up

I decided to test out the algorithm with 150 items and generated a list of knapsack_weights that follows normal distribution, weight ranges from 10 to 40. I also generated a list of knapsack_values that also follows normal distribution, value ranges from 20 to 30.

4.2. Fitness Function

Fitness Function for Knapsack is maximizing the amount of items in terms of value while making sure the sum of all items' weight is less than or equal to knapsack weight.

4.3. Randomized Hill Climbing

For RHC parameter, I set the iteration_list as 10000, max_attempts as 100 and restart_list as 100.

This algorithm executes a total of 11341 iterations with the run time Figure 17 shows below.

```
Run time: 8.640345281000009
CPU times: user 8.83 s, sys: 570 ms, total: 9.4 s
Wall time: 8.66 s
```

Figure 17: RHC Run time – Knapsack

The Average Fitness value is 1964.8526237082997, the Max Fitness value is 2006.2210234989425. Figure 18 shows the Learning Curve for Best Randomized Hill Climbing Random Search below.

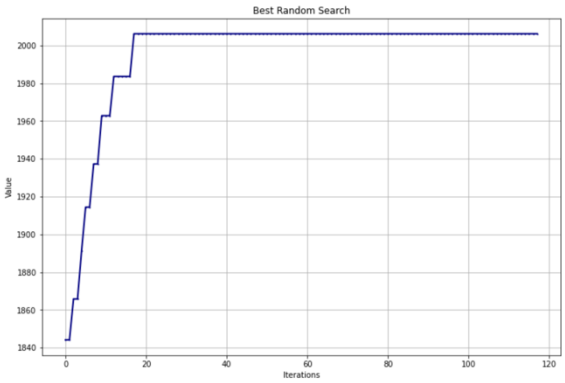


Figure 18: Best RHC Random Search Learning Curve- Knapsack

Different learning curve compare to RHC from previous two problems. The fitness score quickly reach the max and stay the same after 20 iterations. So I suspect the RHC found a max (could be local max) and didn't random search enough so it stuck there. But compare to previous two problems, RHC is running fast here with only 8 second. Having more attempts should help RHC to get out of local optimal solution, which will help with the results.

4.4. Simulated Annealing

For Simulated Annealing's parameter, I set the iteration_list as 10000, temperature_list with 9 different values as [1, 10, 50, 100, 250, 500, 1000, 2500, 5000], decay_list with two different functions ExpDecay, and GeomDecayand and max_attempts as 100.

This algorithm executes a total of 11039 iterations with the run time Figure 20 shows below.

```
Run time: 2.1033399829998416
CPU times: user 2.26 s, sys: 349 ms, total: 2.61 s
Wall time: 2.12 s
```

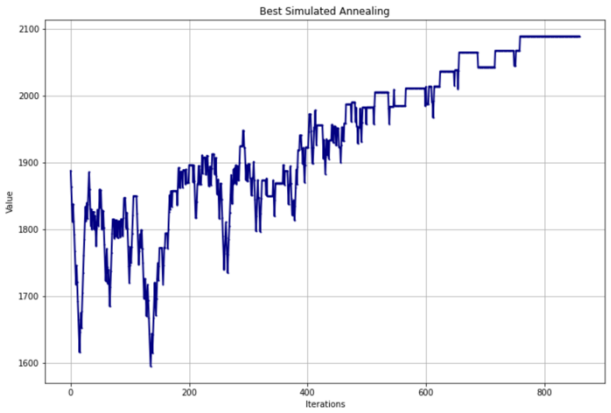
Figure 20: Simulated Annealing Run time- Knapsack

The Average Fitness value is 1981.76190311197, the Max Fitness value is 2089.243219479692, Figure 21 below shows the Learning Curve for Best Simulated Annealing.

Figure 21: Best Simulated Annealing Learning Curve- Knapsack

The best result comes from the ExpDecay as the decay function with init_temp as 250, exp_const as 0.05 and min_temp as 0.001.

This Simulated Annealing learning curve is also different from previous two questions. You can see the fitness score being volatile in the first 200 iterations and had a hard time to beat fitness score of 1900. But the



max fitness score is better than RHC and run time is shorter than RHC as well. I think this is related to Simulated Annealing fighting through all the local optimal solution to get the final global optimal solution. Having a different temperature shall help the algorithm to perform better.

4.5. Genetic Algorithm

For Genetic Algorithm's parameter, I have iteration_list as 100000, max_attempts as 100, 2 different population_sizes as 500, 1000, and 3 different mutation_rates as 0.1, 0.25, 0.5, that makes a total of 6 combinations, which represents 6 different Genetic Algorithms.

This algorithm executes a total of 1687 iterations with the run time Figure 22 shows below.

```
Run time: 155.28764001100012
CPU times: user 2min 35s, sys: 84.2 ms, total: 2min 35s
Wall time: 2min 35s
```

Figure 22: Genetic Algorithm Run time- Knapsack

For population size 500, it took 15.785109 sec to run, population size 1000 took 35.963871 sec to run. Again, it doesn't surprise me that as population size increases, the run time increases due to the more complex evolution between generations. Compare to RHC and Simulated Annealing, Genetic Algorithm again is the slowest algorithm. I suspect the slowness is related to all the best genetic information each iterations has to pass down to next, which takes a lot of time.

The Average Fitness value is 2420.5106095336205, the Max Fitness value is 2421.373122854511,

Figure 23 below shows the Learning Curve for Best Genetic Algorithm.

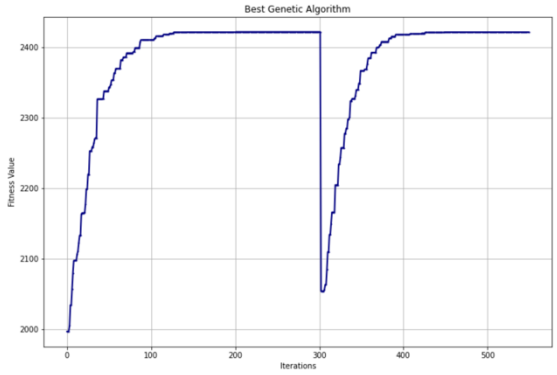


Figure 23: Best Genetic Algorithm Learning Curve- Knapsack

Genetic Algorithm is producing insanely good results compare to RHC and Simulated Annealing. By looking at the learning curve, it's obvious that even the starting point has a great fitness score of 2000, which is almost the max score for the other two algorithms. Around 300 iterations, we do see a huge fitness score drop. I suspect that's related to a new generation mutation of passing new value down to the next. Having a different mutation rate will help speed up the mutation process which might generate better results.

4.6. MIMIC

For MIMIC Algorithm's parameter, I have iteration_list as 100000, max_attempts as 100, 2 different population_sizes as 500, 1000, and 3 different mutation_rates as 0.1, 0.25, 0.5, that makes a total of 6 combinations, which represents 6 different MIMIC.

This algorithm executes a total of 874 iterations with the run time Figure 24 shows below.

```
Run time: 342.8061064220001
```

Figure 24: MIMIC Run time – Knapsack

For population size 500, it took 40.292921 sec to run, population size 1000 took 73.960037 sec to run.

The Average Fitness value is 2399.2260431527316, the Max Fitness value is 2413.4664861237916, Figure 25 below shows the Learning Curve for Best MIMIC.

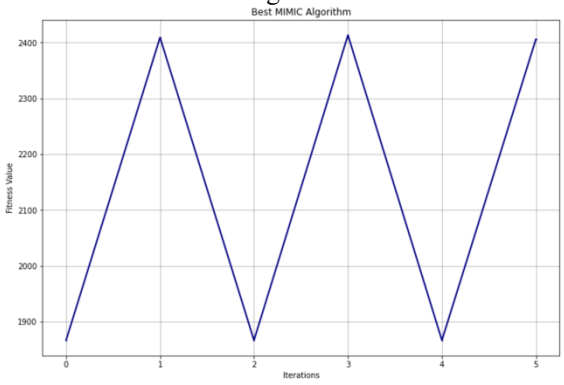


Figure 25: Best MIMIC Learning Curve – Knapsack

Even though MIMIC took the longest to run among all 4 algorithms, but it does produce the best Fitness result. I think MIMIC is able to perform so well due to the fact it's able to maintain a lot of good information (valuable items) from the density estimator. And there is trade off for the high performance for sure, hence why we see the long run time. Having a smaller iteration list might help with the run time but hurt the fitness score.

4.7. Knapsack Conclusion

	Max Fitness	Mean Fitness	Run Time (sec)
RHC	2006.22	1964.85	8.64
Simulated Annealing	2089.24.0	1981.76	2.103
Genetic Algorithm	2421.37	2420.51	155.28
MIMIC	2413.47	2399.226	342.80

Very interesting results here, both RHC and Simulated Annealing cost way less time to train compare to Genetic Algorithm and MIMIC, but don't produce great results as Genetic Algorithm and MIMIC. Like I stated in each model, in order to achieve high performance for Knapsack problem, it requires to pass down/maintain tons of information from previous iterations. And the cost of that is having a high running time, which make MIMIC and Genetic Algorithm out perform RHC and Simulated Annealing in Fitness score, but lose the battle in time complexity.

4. Assignment 2 Neural Network

I decided to use the MNIST data from assignment 1to find good weights for the neural network.

Recap on neural network(using backpropagation)'s performance on hotel booking data. It will be really interesting to see how these 3 algorithms will perform against the backpropagation.

	Neural Network (Backpropagation)
Train Score	0.99989
Test Score	0.92319
Precision	0.928
Recall	0.924
F1	0.937

4. 1. Set up

I set train, test set follows 80%, 20% rule and cross validation as 5 to test all 3 algorithms.

4.2. Randomized Hill Climbing NN

For RHC, I have set the grid_search_parameters with learning_rate with 2 different values, 0.01, 0.001, and restarts with 3 values 5, 25, 50.

RHC ran for total 3 hrs 20 mins after iterating 1000 times. Figure 26 shows the test and train results.

	precision	recall	f1-score	support
0	0.10	0.20	0.14	980
1	0.36	0.64	0.46	1135
2	0.00	0.00	0.00	1032
3	0.24	0.23	0.24	1010
4	0.29	0.01	0.02	982
5	0.06	0.01	0.02	892
6	0.30	0.44	0.36	958
7	0.19	0.59	0.28	1027
8	0.07	0.02	0.03	974
9	0.00	0.00	0.00	1009
micro avg	0.22	0.22	0.22	9999
macro avg	0.16	0.21	0.15	9999
weighted avg	0.16	0.22	0.16	9999
samples avg	0.22	0.22	0.22	9999

	precision	recall	f1-score	support
0	0.12	0.22	0.15	1962
1	0.33	0.64	0.44	2243
2	0.00	0.00	0.00	1989
3	0.22	0.20	0.21	2021
4	0.20	0.01	0.02	1924
5	0.04	0.01	0.01	1761
6	0.33	0.45	0.38	2038
7	0.20	0.63	0.31	2126
8	0.05	0.01	0.02	1912
9	0.22	0.00	0.00	2023
micro avg	0.23	0.23	0.23	19999
macro avg	0.17	0.22	0.15	19999
weighted avg	0.18	0.23	0.16	19999
samples avg	0.23	0.23	0.23	19999

Figure 26: Test, Train Metrics RHC – NN

Fairly poor results are presented. I suspect the RHC got stuck in a local optimal solution and wasn't able to search randomly enough to get out of that local optimal solution. With limited train data, it's also possible that RHC didn't have enough data to search, which cause underfitting. By increasing restart value might help with this problem and generate better results.

4.3. Simulated Annealing NN

For Simulated Annealing parameters, I have set to use ExpDecay function with 6 different temperature 1, 10, 25, 50, 100, 10000 and learning_rate with 0.0001, 0.001, 0.0025, 0.005, 0.01.

Simulated Annealing took only 1 hr 26 mins to run, which is already much faster than RHC. It turns out all the different parameters don't help the performance that much, but overall, Simulated Annealing has generated much better result than RHC, Simulated Annealing result is similar to Assignment 1 Backpropagation result, but slightly worse.

Figure 27 shows the train and test results.

	precision	recall	f1-score	support
0	0.89	0.89	0.90	980
1	0.91	0.92	0.90	1135
2	0.84	0.85	0.87	1032
3	0.93	0.91	0.91	1010
4	0.96	0.91	0.91	982
5	0.99	0.95	0.97	892
6	0.90	0.90	0.89	958
7	0.94	0.91	0.90	1027
8	0.85	0.80	0.81	974
9	0.87	0.85	0.82	1009
micro avg	0.89	0.89	0.89	9999
macro avg	0.91	0.90	0.90	9999
weighted avg	0.90	0.91	0.86	9999
samples avg	0.90	0.89	0.89	9999

	precision	recall	f1-score	support
0	0.90	0.90	0.90	1962
1	0.88	0.81	0.82	2243
2	0.86	0.86	0.88	1989
3	0.81	0.85	0.85	2021
4	0.88	0.81	0.82	1924
5	0.81	0.86	0.88	1761
6	0.82	0.80	0.80	2038
7	0.83	0.89	0.86	2126
8	0.93	0.90	0.90	1912
9	0.86	0.91	0.91	2023
micro avg	0.89	0.89	0.89	19999
macro avg	0.92	0.89	0.86	19999
weighted avg	0.90	0.89	0.86	19999
samples avg	0.89	0.89	0.89	19999

Figure 27: Test, Train Metrics Simulated Annealing – NN

Super promising results, I suspect that Simulated Annealing was able to leave local optimal solution behind the due to it's heating and cool process and find the global optimal solution. Because of that, it's not stuck with local optimal solution, therefore make the run time much shorter as well.

Trying more temperatures and different decay function might even improve the performance even more.

4.4. Genetic Algorithm NN

For Genetic Algorithm parameters, I have set to have max iteration as 1000, learning rate as 0.01 and 0.1, max attempts as 100, pop size as 100, and mutation probability as 0.1 and 0,25.

Genetic Algorithm took the longest to run for 6 hrs and 18 mins. However, the results weren't as good as I expected. My high hope came from Genetic Algorithm's mutation and combination since it will pass useful weights to next iterations. Different parameters change the performance dramatically and the best performance came from 0.01 learning rate and 0.25 mutation probability.

Figure 28 shows the Test, Train Metrics.

	precision	recall	f1-score	support
0	0.77	0.77	0.80	980
1	0.79	0.82	0.80	1135
2	0.84	0.85	0.83	1032
3	0.83	0.79	0.79	1010
4	0.76	0.79	0.79	982
5	0.79	0.75	0.77	892
6	0.80	0.80	0.77	958
7	0.84	0.79	0.80	1027
8	0.79	0.80	0.81	974
9	0.77	0.85	0.82	1009
micro avg	0.77	0.77	0.77	9999
macro avg	0.79	0.80	0.80	9999
weighted avg	0.78	0.79	0.82	9999
samples avg	0.78	0.77	0.77	9999

	precision	recall	f1-score	support
0	0.80	0.80	0.80	1962
1	0.78	0.81	0.82	2243
2	0.86	0.76	0.78	1989
3	0.81	0.85	0.85	2021
4	0.82	0.81	0.82	1924
5	0.81	0.86	0.83	1761
6	0.82	0.80	0.80	2038
7	0.83	0.77	0.86	2126
8	0.83	0.80	0.80	1912
9	0.86	0.79	0.79	2023
micro avg	0.81	0.82	0.81	19999
macro avg	0.84	0.82	0.86	19999
weighted avg	0.82	0.82	0.84	19999
samples avg	0.82	0.82	0.82	19999

Figure 28: Test, Train Metrics Genetic Algorithm – NN

As I mentioned earlier, the performance is disappointing. I think this is caused by all the complex connections between neural network layers. A small change in one of the hidden layer weight will end up changing the results by a lot. Genetic algorithm's mutation wasn't able to keep track of all those weights therefore the results aren't as good as backpropagation approach.

Possible improvement can be made by trying out more learning rate and mutation probability.

5. Neural Network Conclusion

It's obvious to see that Simulated Annealing is the clear winner here for the Neural Network problem with close to backpropagation precision, recall, f1-score. Even though it's not beating the backpropagation performance, but it came close. Different parameters don't affect RHC and Simulated Annealing's performance that much, but play a big part in Genetic Algorithm.

7. References

[1] "Eight queens puzzle," Wikipedia.[Online].Available: https://en.wikipedia.org/wiki/Eight_queens_puzzle. [Accessed: 3rd-Oct-2020].

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

[2] "Hill climbing", Wikipedia.[Online].Available: https://en.wikipedia.org/wiki/Hill_climbing. [Accessed: 3rd-Oct-2020].

[4] "Genetic algorithm", Wikipedia.[Online].Available: https://en.wikipedia.org/wiki/Genetic_algorithm. [Accessed: 3rd-Oct-2020].

[5]"Mutual-Information-Maximizing Input Clustering", [Charles L. Isbell, Jr]
Available: https://www.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf. [Accessed: 3rd-Oct-2020].

[6]"Flip-flop(electronics)"Wikipedia.[Online].Available: https://en.wikipedia.org/wiki/Flip-flop_(electronics). [Accessed: 4th-Oct-2020].

[7]"Knapsack-Problem"Wikipedia.[Online].Available: https://en.wikipedia.org/wiki/Knapsack_problem. [Accessed: 4th-Oct-2020].