CS7641 Supervised Learning HW2

Aleksandr Plokhikh (aplokhikh3@gatech.edu)

Abstract

The objective of the assignment is to learn four types of randomized search learners (randomized hill climbing, simulated annealing, genetic algorithms, and MIMIC) on various steps of standard problem sets.

Problem selection

In this assignment, we were asked to find three problems in which three different problems in which three different randomized sech algorithms are the best, If we count the fitness function as the only success criteria with marginal power time, and space complexity then it is really hard to select such problems. I analyzed N-queens, Flip-Flop, Salesman Traveler, Knapsack, Four Peaks, Six Peaks, and K-Colors problems with different problem sizes, and problems hyperparameters. I was unable to such a collection of 3 different hard problems that highlight 3 different search algorithms if we treat the fitness score as the only success criteria.

Almost always simulated annealing was the best algorithm for the task, and I saw that genetic algorithms performed well on the knapsack problem, So I included it in the shortlist, but overall genetic algorithms did not prove its name of "second best algorithm" (I think it is a joke to be honest). If we include in our success criteria the run time with high weight to the overall performance then we may ask if the sometimes randomized hill climbing is the best algorithm, and we have 3 different problems with 3 different champions (maybe it was the whole idea that selection of success criteria does matter of cause it does matter but in some cases the run time can be a bottleneck, and in other not so much. Everything depends on tasks).

The Flip-Flop problem did not come on the short list since for relatively hard problems that require long bit strings MIMIC algorithm takes too much time and is not practical to analyze and waste your time (as an assignment evaluator) to run the code, and wait for the result. The flip-flop is also not practical for randomized search algorithms overall. I also had issues with running MIMIC on the Travel salesman problem and did not include it in the shortlist. For Four peaks, and Six peaks problems even for large hard problem sets simulated annealing swiftly finds a globally optimal solution, and it is impossible to drive RHC, and MIMIC algorithms to find the global optimal solution faster. The performance graphs for tasks that did not come in the shortlist are present in the assignment folder and can be regenerated from the main HW2.py program if needed.

In the shortlist, I included N-queens, Knapsack, and K-colours as the most useful problems in real life, most visually understandable, reasonably hard to run, and find global optima, and as a set in which 3 different algorithms become problem-solving champions. All selected problems are NP-hard. The Knapsack problem has a tremendous field of implementation for logistics, and transportation needs. I do not have numbers on my hands but it is easily billions of dollars per year. And we solve this problem not only in companies but during our regular life on a daily basis. K-colors are one of the best constrained satisfaction problems and are very useful for map coloring to have maps with the smallest number of colors, clean, clear, and aesthetically pleasing. I wanted to put the travel salesmen problem as the third as it also has tremendous importance in logistics on billions dollar per year and is very important for Google Maps, uber, lift, Instacart, Uber Eats, door dash, etc, I got very nice results with RHC, SA, and even GA on this task (the corresponding performance graphs in the folder) but MIMIC algorithm fails there for some reason, I will study why after the assignment submission and by now I chose N-Queens problem as a fun problem to solve but maybe not very practical to be honest.

Important note

For each task, and each algorithm I ran grid-search for the best hyperparameters in a reasonable range (by reasonable I mean that I did not include very long iterations or total population sizes since it is not practical to run) but in the main run file I included the runners with the optimal parameters only because genetic algorithms, and especially MICIC algorithm takes a lot of time to run for good fitness score on many hard times, and I targeted the total run time of fewer than 30 minutes on 4 cores CPU so you as evaluator could easily regenerate all the graphs without a very excessive pain (I was also a TA and do not want you to suffer). Hope My care for you will not count against me, I am just trying to save you time.

For the same reasons I aggregated 4 randomized algorithm performance plots in one to save, space, your time, and make the comparison more straightforward.

N-Queens

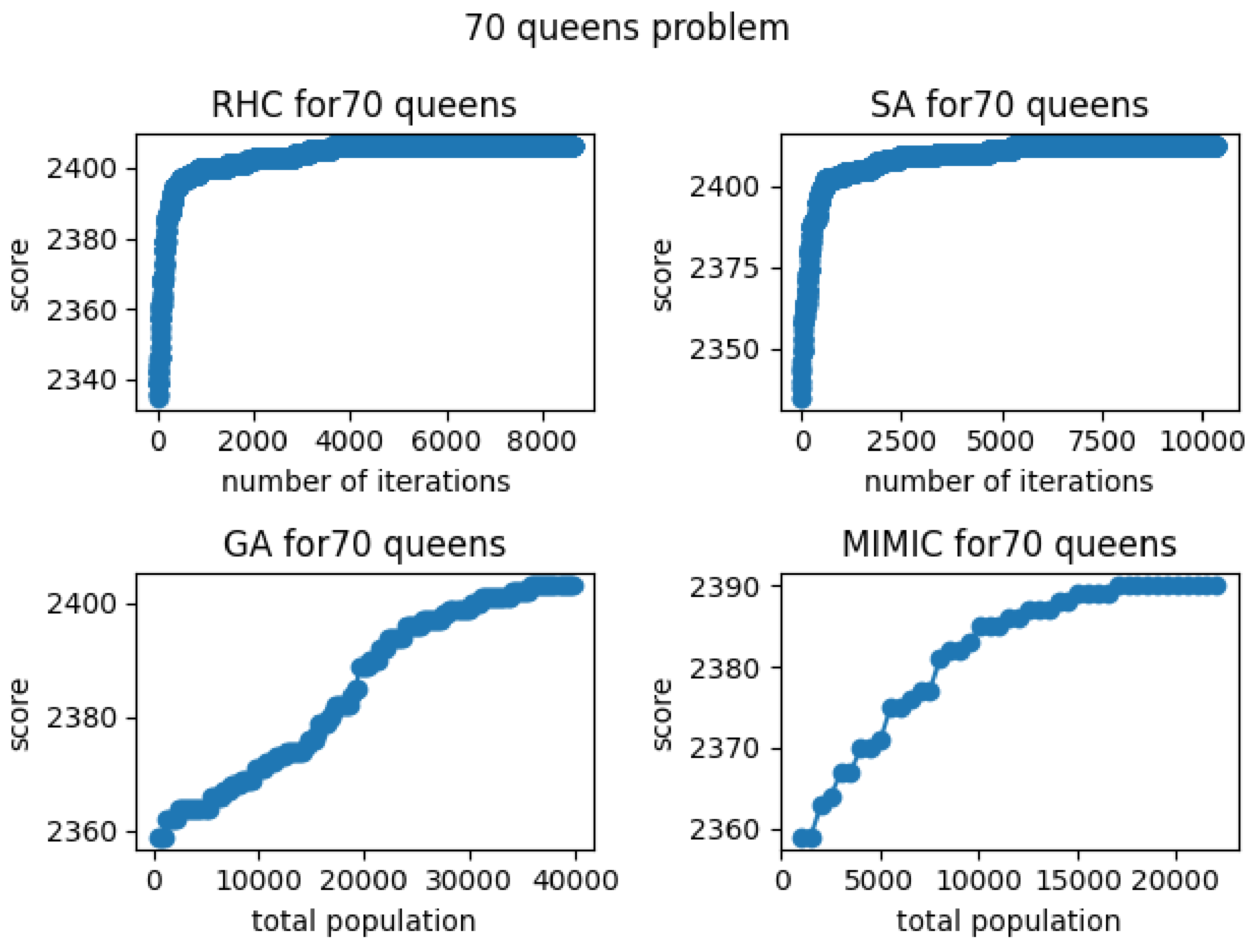


Fig. 1 Performance of Randomized Search algorithms on N-Queens problem

For the N-Queen problem, I had to use a higher N than the regular 8 since for N=8 all algorithms find global optimal solutions in a split of a second. Any N more than 50 or 60 can be a reasonably hard problem here. I chose N=70 in order to not overcomplicate things for the MIMIC algorithm which is notorious for struggling with high-dimensional problems. Even though MIMIC showed the worst performance out of 4 algorithms with the lowest performance score of 2390, and the longest run time of 104 seconds, while the optimal solution has a score of 2415 (the score metric counts only one attack per attack pair when 2 queens attack each other, and also it counts attack with “penetrating strike” when it count all attack through the line even if there is already at least one queen on the strike path). The MIMIC algorithm also does not reach the optimal solution until quite a high total population number of 17000, and there seems no room for further improvement.

The genetic algorithms with a population size of 200, and mutation probability of 0.1 steadily increase fitness score to the value of 2403 with a run time of 57 seconds. The algorithm may still have some small room for improvement but higher mutation rates do not catapult us there, and other remaining algorithms find considerably better scores in a low time frame.

The randomized hill climbing algorithm gives us the fitness score of 2406 and runs for 11.7 seconds while converging to the local optimal value in less than half of the run.

The simulated annealing algorithm gives us the fitness values of 2412 which is very close to the global optima of 2415 and does it in 14 seconds while converging to the local optima just above the half time. Variation of starting temperature, and temperature decay does not affect the performance significantly.

Simulated annealing, or randomized hill climbing can be named as the best algorithm here depending on selected success criteria. The great performance of these two algorithms can indicate that there are useful fitness score gradients in solutions space with relatively large attraction areas, it also slightly helps the MIMIC algorithm but not to the point when it has a really good score. Lack of structure in the solution space prevents it I lean toward the Simulated annealing algorithm as the winner here. The genetic algorithm took almost an order magnitude longer time to run to be a good contender, while their fitness scores are not too far off from the leaders.

Knapsack problem

I need to say that it was quite hard to make a really difficult Knapsack problem set. Even an increase in the total items number at other default values does not make the task very hard but significantly tanks the MIMIC algorithm run times since it significantly longer on high dimensional problems. Only the increase of maximal collected items present quickly made the problem really hard. So I used 0.9 as the max weight percent.

The MIMIC again was easily the worst-performing algorithm. Its maximum fitness score of 7365.21 is significantly lower than for any other algorithm and it takes 808 seconds to find the solution which is 2-4 orders of magnitude longer than the run time for all other algorithms. I even had to reduce the total population size to make a reasonable fitting run which takes less than 15 minutes. The high dimensionality of the solution and lack of structure lead to the low performance of the algorithm.

The simulated annealing algorithm goes miles away from the MIMIC algorithm here, it reaches the score of 11319.13 in about 15% of the total run time of 0.18 seconds, reaches the convergence, and does not increase any further, increasing of starting temperature or decreasing temperature decay do not enhance performance as in case with the N-Queens problem.

The behavior of the randomized hill climbing algorithm is similar to Simulated annealing. It reaches an even higher score of 11360.12 as well at around 15% of an even lower run time of 0.09 seconds. Which makes Randomized Hill Climbing a better algorithm for the task. The algorithm also reaches a plateau, and there is no hope for significant performance improvement.

Genetic algorithms showed the best performance here of 11662.26 which is 300 hundred points higher than the second contender. The performance of the algorithm rises almost the same way and the same values as in the case of randomized hill climbing, and simulated annealing methods, but it does not suddenly level up once previous algorithms reached local optima but continues slowly rice its performance. The algorithm ran for 3.89 seconds which is pretty fast and makes it the winner of the Knapsack problem.

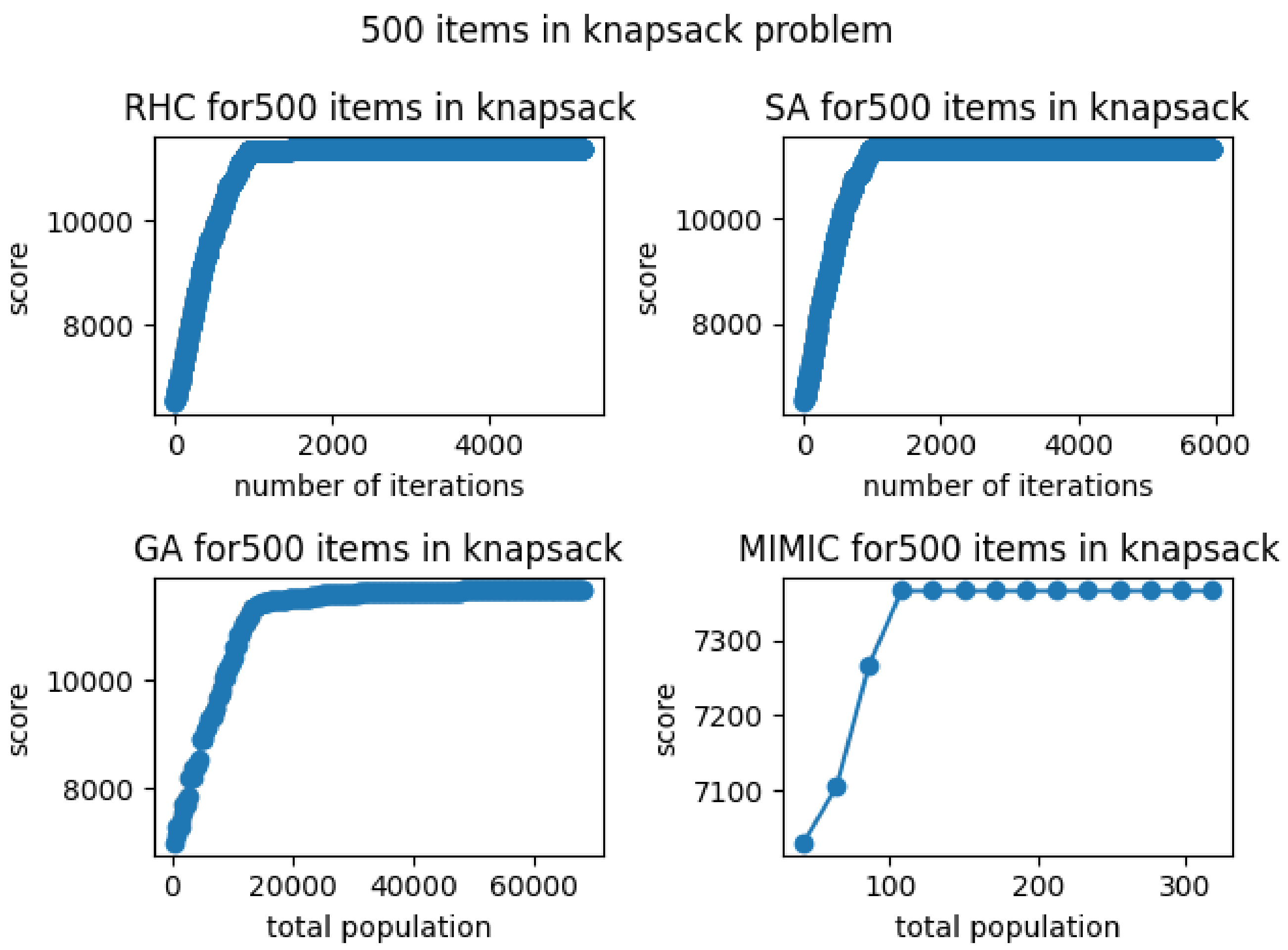


Fig. 2 Performance of Randomized Search algorithms on Knapsack problem

The apparent lack of structure and large attractive areas near high score solutions of the problem prevent randomized hill climbing, and simulated annealing to reach the highest score here.

K-colors

For K-colors, I generated the net of 100 nodes, and 400 random edges, and solved the problem for 3 colors. It was a reasonably hard problem for the algorithms.

The worst score of 339 showed a randomized hill climbing algorithm but the run time was only 1.42 seconds, and it reached the convergence at 20% of the run time.

The MIMIC algorithm did just slightly better with a score of 340, and a runtime of 339.8 seconds. The high dimensionality of the solution space and lack of structure do not help the algorithm here.

Genetic algorithms were better and steadily increased performance until reaching the score of 347 in 27.5 seconds, but the champion was the Simulated annealing algorithm with the score of 352 and run time of 9.62 seconds, and it reached the convergence in 20% of the run time which makes it the champion on K-color problem.

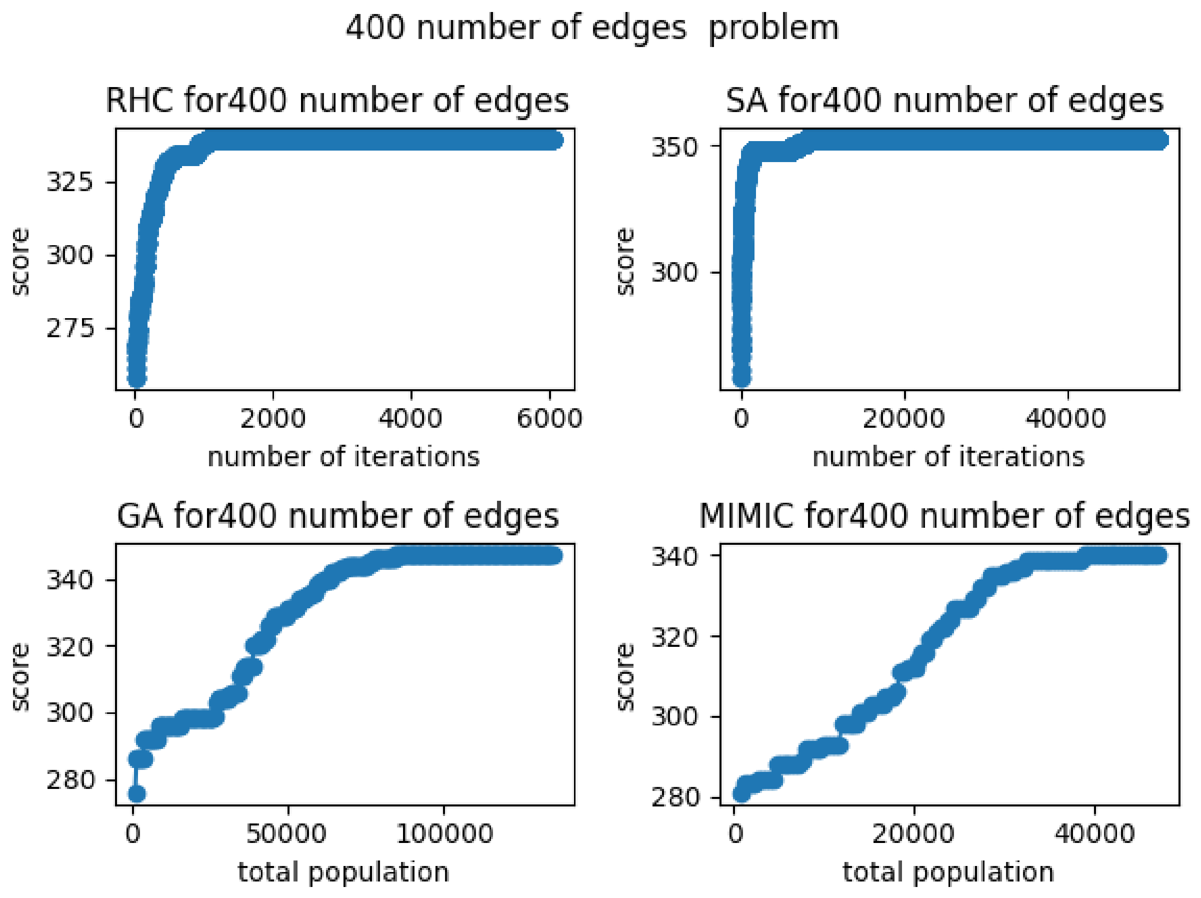


Fig. 3 Performance of Randomized Search algorithms on K-colors problem

Algorithms performance conclusion on the selected problem sets

The high dimensionality of the solution space and lack of structure suppressed the performance of the MIMIC algorithm on all problems. It was consistently a worse overall algorithm. If the solution space had large or reasonably large attraction areas (like in the case of N-Queens, and K-Colors problems) then the fitness score of the MIMIC algorithm was not very far from the leaders, but if the attraction area to local maxima is low (like in case of Knapsack problem) then the performance of the algorithm is very low. The run time of the algorithm is the worst as expected since it has to do many fitness function evaluations, and high dimensionality does not help here and makes things worse.

The lectures paid more than half of the time to the MIMIC algorithm which has very limited application. It is wise to spend more time on the standard algorithms like RHC, SA, and GA. Charles did a bad favor to us then he focused on the MIMIC algorithm. He is not even the first author of the paper. This paper has only 700+ citations over 30 years which is not a lot. The MIMIC algorithm did not have a high impact on the community. Charles's selfishness put give to MIMIC algorithm more attention than it deserves, and it was done in the expanse of the algorithm which does work (RHC, SA, GA).

The genetic algorithm did not prove its name as "the second best approach" It was slower than randomized hill climbing or simulated annealing and performed really well only on the tasks with small attraction areas around good solutions (Knapsack problem) and I included it only because I needed to highlight performance some algorithms other than RHC or SA.

The Randomized hill climbing and simulated annealing algorithms were consistently good on all tasks, and the run time was low. The Simulated annealing was always better than RHC and surprisingly was not very affected by starting temperature, and decay. RHC on the other hand could offer a slightly faster run time since it is the simplest to execute the algorithm of all four of them. The simulated annealing algorithm is just slightly more complex than RHC and does not require to much more process time to run.

Randomized search algorithms for weight find in neural networks

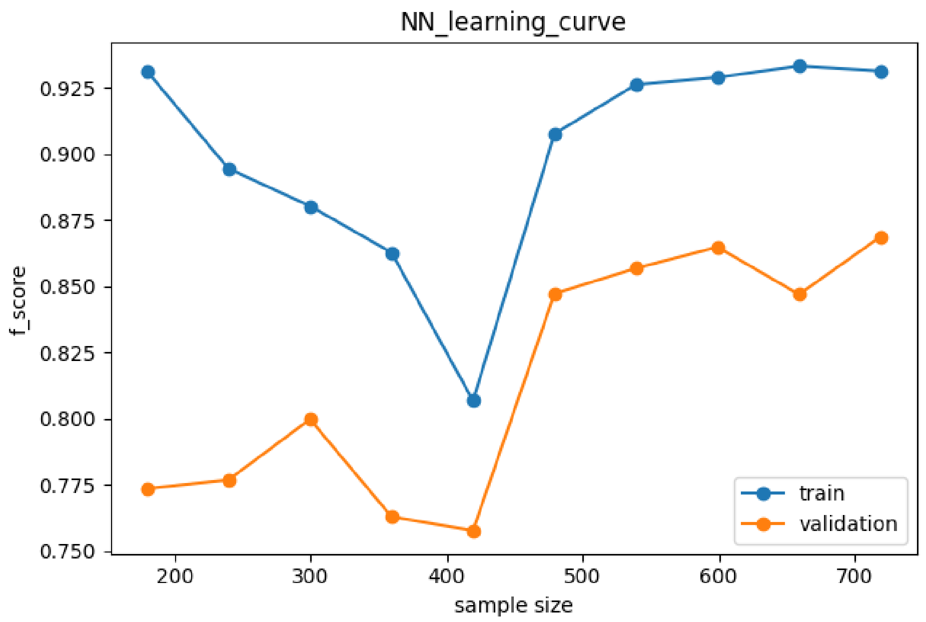
For the second part of the assignment I decided to take the fruits dataset [1]. The task here is a clear classification problem since we have distinct types of fruits I am a person who one time bought cilantro instead of parsley, and my wife was made on me for that. After this incident, I started to pay closer attention to fruits, and vegetables that I buy in a grocery store because I do not want my wife to be mad at me once again I think the ask of fruits classification is pretty important for many fellas. Also, I found that the fruits dataset contains fruits I have never seen in my life, and it would be handy if I could utilize ML algorithms to classify them. The data set contains 34 features, which is a lot. I do not think that I will do 34 measurements of each fruit in a grocery store but find the task fun anyway. It is also an interesting way to learn about new fruits. Also this data set have many different distinct entities.

I did a preprocessing for my data sets. First of all, for the fruits dataset, I changed the names of the fruits on numbers from 0 to 6 to facilitate the classification work. Also, I scaled all features linearly to range from 0 to 1 where 0 is the minimal feature value, and 1 is the maximum feature value. To be consistent with home work #1 where I did it for reasonable performance of KNN method because without any extra knowledge, I cound not give any parameter axis any preference, and hid to keep them equally scaled.

All datasets were shaffled and split into two sets: training (80% of data), and the testing one (20% of data) which I put aside, and never used for training, or validation purposes only for final testing of the ML algorithms performances evaluation.

The champion network's last one was 25 neurons per layer in the net of 25 layers. So it I not very hard to train. The activation function I had to substitute from logistics to relu since mlrose library does not work with the logistics activation function in neural nets. So I had to run learning and test once again with relu activation function this time.

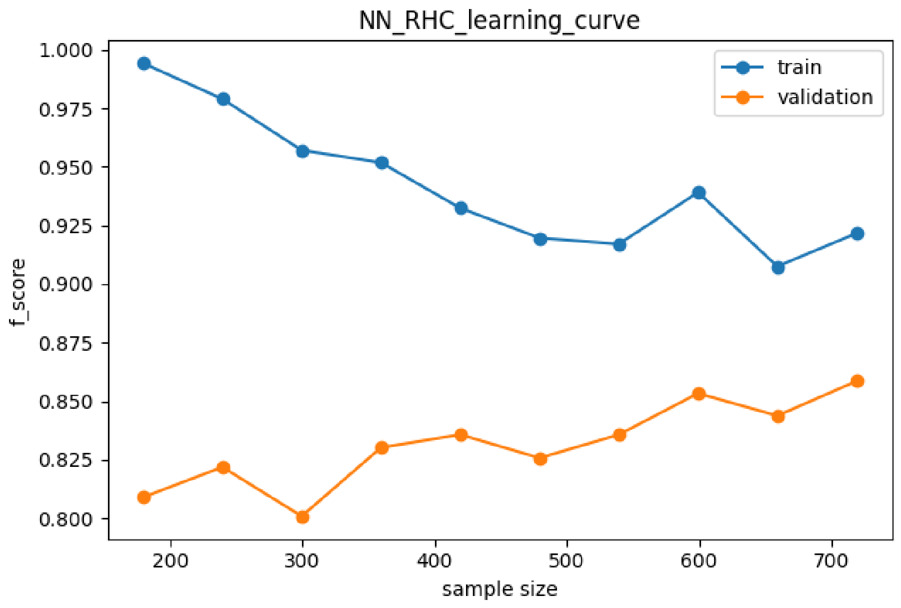
The classical network took only 0.23 seconds to be trained, showed a high f-score of 0.9045 on the test sample (see Fig 4), and detected all different fruits with high accuracy. The learning curve shows a steady increase in the network performance with an increase in the sample size but has an unusual drop in the middle which says that the neural network did not stabilize its performance, it also can be an issue of low cross-validation splitting which I had to keep equal to 2 to run other algorithms in reasonable amount of time. The neural network is undertrained and can enhance its performance with a larger training sample size since there is still a sizable gap between test and validation set performance.



	precision	recall	f1-score	support
0	0.9231	0.8571	0.8889	14
1	0.8500	0.7727	0.8095	22
2	0.9412	0.9412	0.9412	34
3	0.8235	1.0000	0.9032	14
4	0.8750	1.0000	0.9333	28
5	1.0000	0.9767	0.9882	43
6	0.7826	0.7500	0.7660	24
micro avg	0.9006	0.9106	0.9056	179
macro avg	0.8851	0.8997	0.8900	179
weighted avg	0.9019	0.9106	0.9045	179
samples avg	0.8939	0.9106	0.8994	179

Fig. 4 Learning curve and test set performance of the classical neural network

The implementation of the randomized hill climbing algorithm benefited from a low initial learning rate of 0.02 and very high iteration numbers of 100000 and produced a network that even slightly beat the performance of classical neural networks. F -score on the test sample was 0.9092, which is 0.5% better than the neural network trained the classical way. All fruit species are determined with high accuracy But the drow back here is the learning time of almost 148 seconds which is two orders of magnitude higher than in the traditional case. For the creation of the learning curve, I had to increase the learning rate and decrease the iteration number 10 times in order to let the code run in the reasonable time frame of 77 seconds for four cores of CPU.

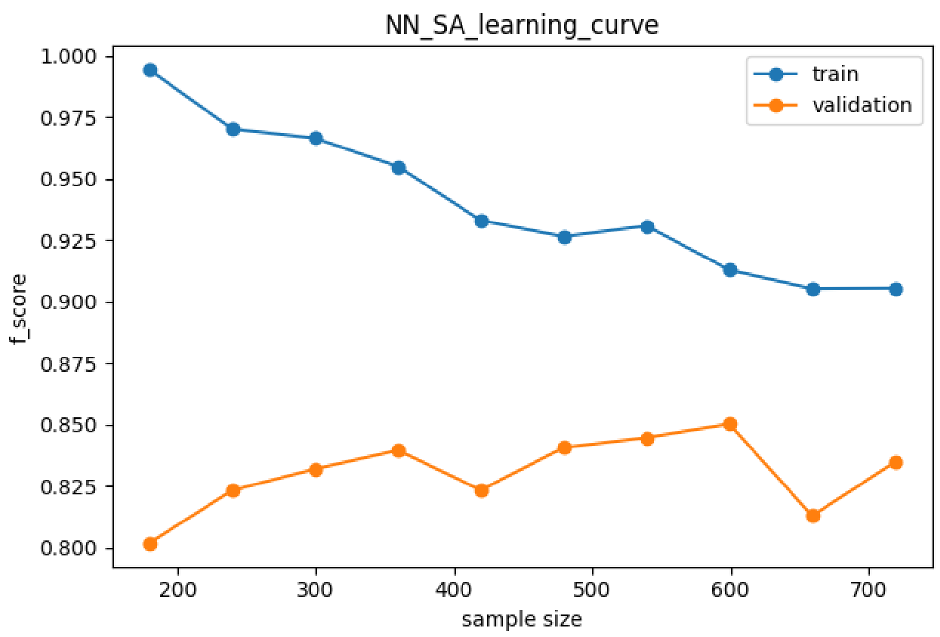


	precision	recall	f1-score	support
0	0.9286	0.9286	0.9286	14
1	0.7826	0.8182	0.8000	22
2	0.9143	0.9412	0.9275	34
3	0.9286	0.9286	0.9286	14
4	0.9000	0.9643	0.9310	28
5	1.0000	1.0000	1.0000	43
6	0.8500	0.7083	0.7727	24
micro avg	0.9106	0.9106	0.9106	179
macro avg	0.9006	0.8984	0.8983	179
weighted avg	0.9101	0.9106	0.9092	179
samples avg	0.9106	0.9106	0.9106	179

Fig. 5 Learning curve and test set performance of the neural network trained with RHC algorithm

The learning curve shows a steady rise of the neural network performance on the cross-validation sample, there is very substantial room for growth at the larger train sample, especially for the neural net which is trained with a lower learning rate of 0.02 and iteration number of 100000 (not included in the report due to long run time of more than 15 minutes for this setting the learning curve is even smoother and shows higher f-score it is just not practical to run and aim for in this task).

While the simulated annealing algorithm performed better in previous tasks it was not better than RHC this time maybe because the ideal weight for NN is impossible to predict and they can be very far away one from another and it is hard to find it when you sometimes go against the gradient without a random restart.



```
             precision    recall  f1-score   support

          0     0.7500    0.6429    0.6923        14
          1     0.6667    0.7273    0.6957        22
          2     0.8421    0.9412    0.8889        34
          3     0.8000    0.8571    0.8276        14
          4     0.8387    0.9286    0.8814        28
          5     1.0000    1.0000    1.0000        43
          6     0.7500    0.5000    0.6000        24

  micro avg     0.8380    0.8380    0.8380       179
  macro avg     0.8068    0.7996    0.7980       179
weighted avg    0.8351    0.8380    0.8317       179
 samples avg    0.8380    0.8380    0.8380       179
```

Fig. 6 Learning curve and test set performance of the neural network trained with SA algorithm

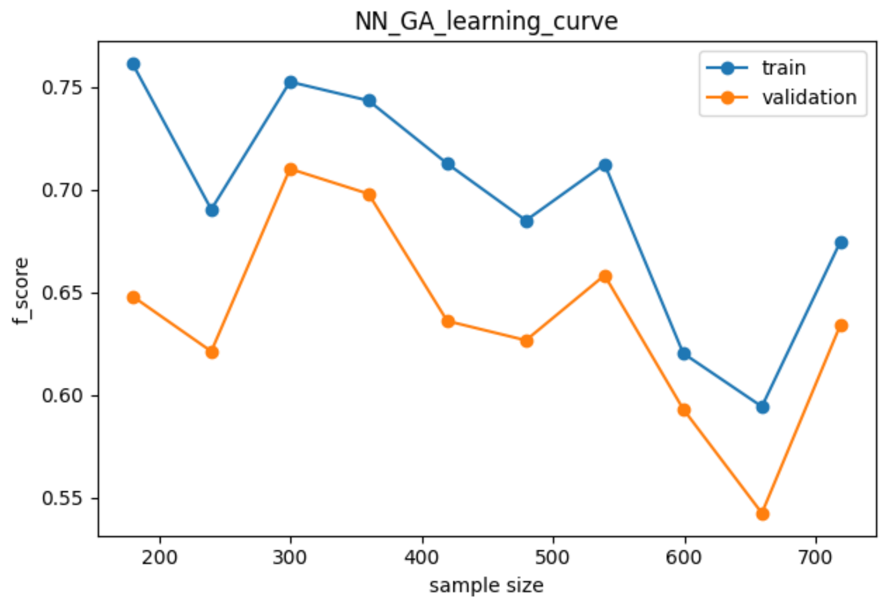
The Simulated algorithm annealing algorithm has a low initial learning rate of 0.02 and very high iteration numbers of 100000 but the f-score on the test sample is only 0.83. The algorithm got stacked in local optima, but an increase of the initial temperature, and slowing of the decay did not increase the performance. It is really hard to predict good weight, and we need random restarts here or some other mechanisms to go into another area of solutions space. The algorithm also takes 221 seconds to run 50% more than RHC.

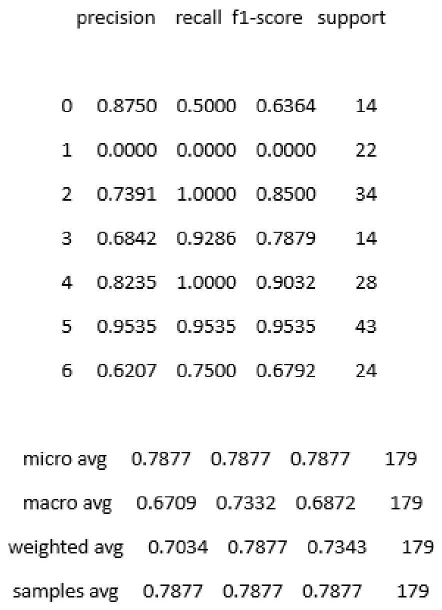
For learning curve plotting I also had to increase the learning rate and decrease the iteration number 10 times in order to let the code run in the reasonable time frame of 160 seconds for four cores of CPU.

There is no consistent upward trend in the cross-validation learning curve. The neural networking hardly will start to perform significantly better if we increase the training sample.

The genetic algorithms demonstrate the lowest performance here. The f-score on the test sample is only 0.7343. And the network can not identify deglets at all even though it takes a longer time to train – 357 seconds. I made multiple attempts to enhance the performance of the algorithm. A decrease in the population value below 50 reduces -the score on the test sample while an increase in the population number does not provide benefits and just slows down training. I expected to see that a high mutation rate would increase the performance since in this case genetic algorithm starts to work more similar to a randomized hill climbing, and I thought that it is a way to go since it is hardly possible to predict good weights but increase mutation rate over 0.01 lead to decrease of the f-score.

The learning curve shows that the neural network updated with genetic algorithms does not enhance cross-validation performance with an increase in the training set. There is very limited possible room for growth since the performance on the training sample remains low and close to the performance on the cross-validation data set.





```
                precision    recall  f1-score   support

            0      0.8750    0.5000    0.6364        14
            1      0.0000    0.0000    0.0000        22
            2      0.7391    1.0000    0.8500        34
            3      0.6842    0.9286    0.7879        14
            4      0.8235    1.0000    0.9032        28
            5      0.9535    0.9535    0.9535        43
            6      0.6207    0.7500    0.6792        24

    micro avg      0.7877    0.7877    0.7877       179
    macro avg      0.6709    0.7332    0.6872       179
 weighted avg      0.7034    0.7877    0.7343       179
  samples avg      0.7877    0.7877    0.7877       179
```

Fig. 7 Learning curve and test set performance of the neural network trained with GA algorithm

The genetic algorithm here is the slowest and worst-performing one with no potential for improvement. It is not the “second best choice” at least on the selected fruits dataset.

Overall the best method to update weights here is the classical method via the loss function it provides performance on par with the randomized hill climbing method but does it 300 times faster. Random search algorithms could help if classical methods fall in local optima and can not leave it since the loss function weight update works negative feedback loop and is stuck to the local optima by its nature, but in this case, on our fruits dataset the local optima does not seem like a real issue and we can use a classical and fast way to update the weights.

Conclusion

The simplest algorithms showed the best performance in both parts of the assignment they are easy to execute, fast, and have the best fitness scores. Simulated annealing was the best for classical tasks, and randomized hill climbing was the best for neural network weight fitting probably due to many random restarts in the search process since it is impossible to predict good weight for complex nets and optimal solutions can be vastly different.

Genetic algorithms were not “the second best option” in the tasks, and run time was pretty long which does make sense. I think that the phrase that genetic algorithms are the second best option is mainly a joke in the community, everything knows that it is not really a truth, but it got to much hype because it sounds sexy if I am allowed to say so, looks familiar, it is very similar to natural evolution process happening in real life, and we should be helpful to real genetics since it created intelligent life on the planet but it took 4.5 Billion years to do so. Our earth's crust was created during a process very similar to simulated annealing – we can call it real annealing:).And the process was much faster than the creation of human beings! Also, genetic algorithms sing in unison with the free market idea, which has efficiently driven our humanity last centuries. It also establishes the idea that we can get the best solution by merging the best solution available now, The idea may sims great but it is not necessarily the truth and the best way to find the optimal solution. Anyway, the idea is very influential by itself. It is also reasonably complicated, but at the end of the day very stupidly simple ideas of randomized hill climbing, and simulated annealing can be much faster to implement with results that are often much better., we should implement Occam's razor and do not multiply entities without the need, many times we just need to stick with the simplest solution. In other aspects of machine learning it also saves us from overfitting. I think that Occam would vote for RHC or GA.

The MIMIC algorithm was last in both fitness score and run time. Maybe it is an issue of lack of structure in solutions I do not think that we should have been paying attention to it as much as it was in lectures. It is not that influential algorithm and number of citations on the original paper, which Charles claims to be his but in reality, he is only the second author remains low to have such big attention on the lectures.

References

[1] https://www.kaggle.com/datasets/muratkokludataset/date-fruit-datasets