Randomized Optimization
CS 7641

Abel Aguilar
aaguilar61@gatech.edu

1 INTRODUCTION TO PROBLEMS

In this assignment the three optimization problems that I chose were, the 4 peaks problem, the Knapsack problem, and the Queens problem. The 4 Peaks problem is an optimization problem that is very popular and is often used to test optimization algorithms. Some of the characteristics that make it interesting is the fact that it has multiple local max and just one global maximum. In this it can trick our algorithms into thinking it has found the solution when in reality it has only found a local max. This problem requires exploration in order to be solved but that exploration also needs to be balanced and directed. The Knapsack problem is a rather simple optimization problem that takes a knapsack with a maximum weight and a group of items with weight. The goal of the optimization is to maximize the total number of items that fit in the knapsack without going over the weight limit. This is an interesting problem for optimization because of the nuance that the best solution might not reach the weight limit, it also prioritizes lighter items over heavier items. This is also interesting because of the possible traps that our algorithms can fall into such as focusing too much in one direction and maybe refusing to "take everything out of the bag" to take over. The Queens problem is an interesting take on chess. The problem is, given an NXN chess board, can you position N queens in such a way that none of them can attack any of the other queens. This is an interesting problem because there isn't always just one correct answer. There could be multiple and any one solution isn't really any better than any other. This is also an interesting problem because it forces our algorithms into a space finding solution. Any one position on the chessboard isn't necessarily any better than any other space, it is what is around that space and where the other queens are that determine how good of a space it is.

Lastly I choose to use the Pima Indians Diabetes Database neural network I created for the last assignment to attempt to optimize the weights of the neural network using the optimization algorithms we have learned. This provides an extra challenge to the algorithms because PIDD is unbalanced, most samples are people that do not have diabetes. This means that a way to optimize is simply to guess 0 for all and that would give a pretty large percentage of correct answers. That is a trap that I am hoping the algorithms can overcome.

2 EXPERIMENTAL METHODOLOGY

In order to test each algorithm I will be testing them all (Random Hill Climbing, simulated annealing, a genetic algorithm, MIMIC) against all 3 of my optimization algorithms. Some of the things I will be looking for is how quickly the algorithm can reach its peak fitness both in terms of iterations and wall time. I want to balance that along with what peak fitness is reached as well.

I will also be using visualization to be able to compare the results of each. This will be especially important with the neural network example as not just accuracy will be checked but also precision to make sure that our algorithm isn't simply just guessing 0 for all samples.

3 ALGORITHM RESULTS

3.1 4 Peaks Problem

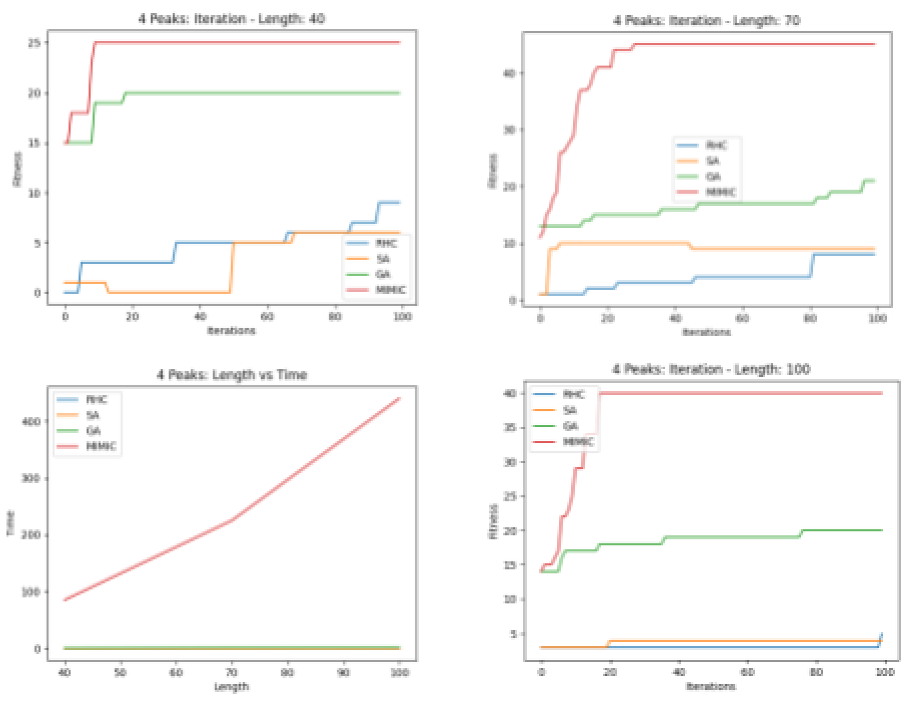


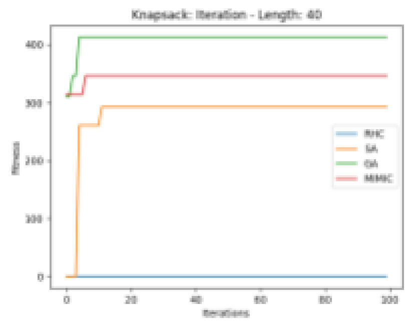
Figure 1- 4 Peaks Problem

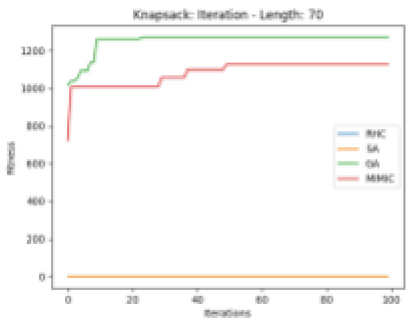
Looking at the above we can see that MIMIC regardless of the length of the problem was able to outperform the other optimization algorithms. However, because of the complexity of MIMIC the time difference that it would take to run that optimization over any other algorithm is very large. Regardless because its performance is so much better than all the other algorithms, in terms of fitness, combined with it reaching its maximum fitness with a very low number of iterations makes it the best algorithm for this specific problem.

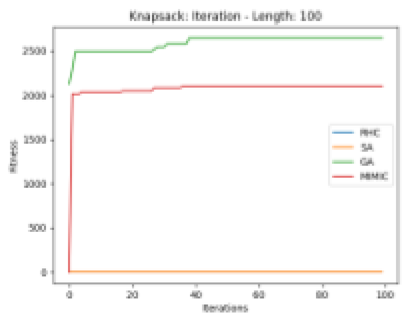
I believe that the reason that MIMIC performs the best in this specific problem is because of the fact that it initially generates a large number of possible solutions and then uses probably models to iterate over those solutions and create new ones. This means that it is unlikely to fall into the same traps that other algorithms might fall into with the 4 peaks problems. For example even if MIMIC finds a local max, given the fact that it also checks many other solutions it is unlikely to only find that local max and stick with it.

This also explains why GA also outperformed SA and RHC. SA and RHC only start with a single solution and we need to rely on the fact that they will explore the correct solution, but that does not always happen. In this case the probabilistic models of MIMIC are better suited than GA but that general principle explains why MIMIC is the best algorithm.

3.2 Knapsack







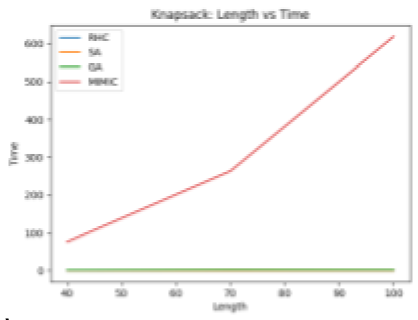


Figure 2- Knapsack

For the Knapsack problem as you can see from above that GA gives the best result. This is a combination of the best fitness but also its run time, clocking in a very low time to execute. This is then compared to the other algorithms where MIMIC also produces pretty great results however its run time makes it not the best choice for this problem. SA and RHC both did not perform well. SA had a good run with length 40 but I attribute that more to luck than the algorithm. Both were rarely able to crack 0 fitness.

The reason GA is performing well in this particular problem is because it does not have the limitation of starting with one solution and then slowly trying to modify it to get a better solution like SA and RHC. Because the knapsack problem requires “taking out everything and starting over” a couple times to get to the best solution, starting with a wide range of solutions gives it an advantage over its competitors. This does not mean that SA and RHC won’t produce a good solution ever but it does mean that it needs to guess right on the first try in order to get close to a good solution. This is why we see one iteration of SA do well and others do poorly. It got lucky on that run. I also believe that using crossover gives it the advantage against MIMIC. This is because it simulates taking an item or items out of the bag and replacing them. This means that it is trying many different combinations before reaching its best solution.

3.3 Queens

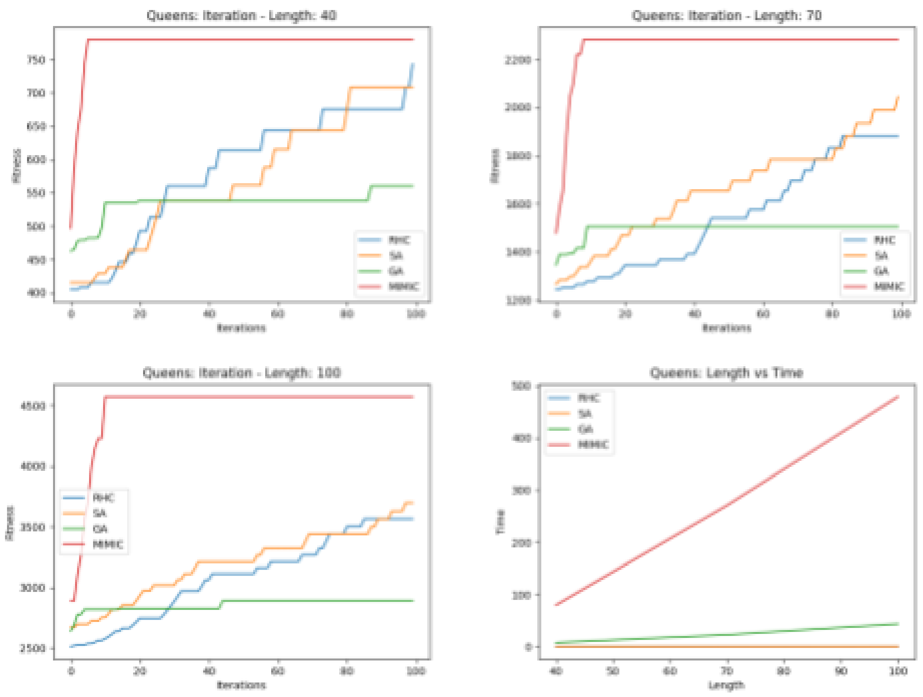


Figure 3- Queens

For the Queens problem we can see that both MIMIC and SA perform well in terms of fitness. However, MIMIC time to reach an optimal solution is much longer than SA. While SA technically had a lower best fitness as the charts suggest it was trending in the right direction for all of them and did not seem to have become flat yet, this suggests that more iteration would have reached or maybe even passed MIMIC in terms of fitness. With all these factors combined it is clear that SA is a better algorithm for this specific problem.

The reason why SA is able to reach such good fitness is because this type of problem does not need a huge amount of solutions and there aren't many places where it will get stuck in "local max" because there is always a way to move a queen on the chessboard. SA starts with a random possible solution and then slowly moves the pieces around the chessboard until to maximize the number of queens that can not attack each other. This is also why RHC does similarly well in the problem. GA does not do well because it is attempting to move multiple pieces at once to a better fitness but in that moving it is not taking into account where the pieces will be and thus it can undo its progress. With this kind of problem it is best to move one piece at a time vs many pieces at a time.

3.4 Neural Network

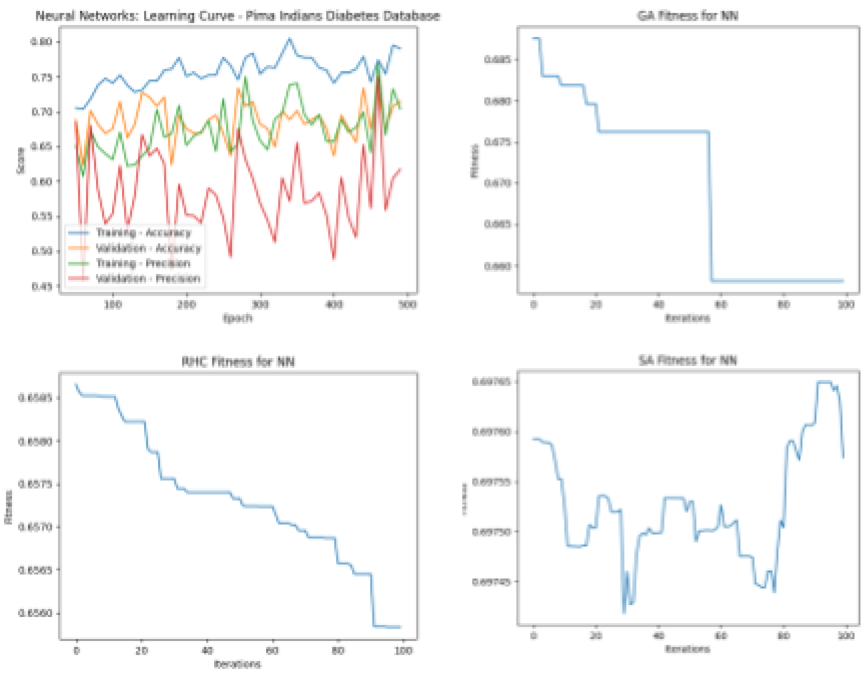


Figure 4- Neural Network Fitness

Looking at the results it looks like none of the three algorithms were able to improve much on their initial guess. All of them stayed roughly around the same percentage. I believe that it most likely has to do with the fact that the data that we are using is unbalanced so the algorithms are unsure what to change to make the fitness better. Look at the figure below.

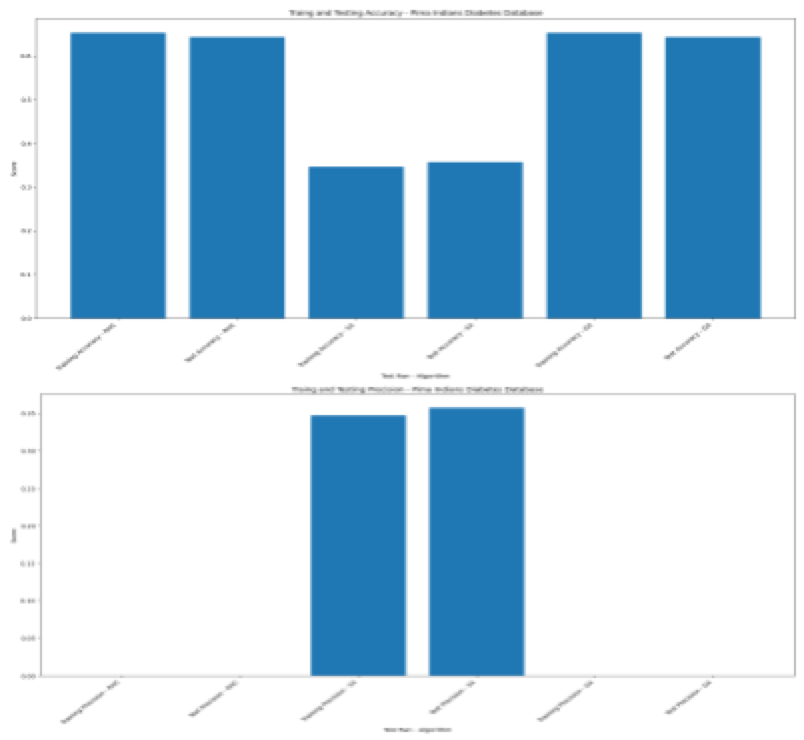


Figure 4- Neural Network Accuracy vs Precision

Looking at the above charts it is clear that in 100 iterations RHC and GA come to the conclusion that guessing 0 for all samples is the best way to optimize the function because that will give you over 60% accuracy. SA is the only algorithm that attempts to learn more about the problem but that costs it when it only gets about 30% correct. This is because the weights of Neural Networks do not have the simplest way of increasing fitness given their complexity.

4 CONCLUSION

In conclusion it is clear that there is no algorithm that fits all problems best and it all depends on the problem itself. MIMIC is best suited for problems that have traps that our algorithm can fall for if not a lot of possible solutions are tested.

GA is best suited for problems that can benefit from cross-over and combining possible solutions to get the best possible results. And SA and RHC are best in problems that do not have traps like local maxes and are best solved only changing very few variables at a time. Optimization algorithms are also not complex enough to understand the nuance that goes into neural networks as even the best optimization algorithm did not come close to the neural networks optimal weights.

5 REFERENCES

1. “Algorithms¶.” Algorithms - Mlrose 1.3.0 Documentation, mlrose.readthedocs.io/en/stable/source/algorithms.html. Accessed 17 Oct. 2023.
2. “Fitness Functions¶.” Fitness Functions - Mlrose 1.3.0 Documentation, mlrose.readthedocs.io/en/stable/source/fitness.html. Accessed 17 Oct. 2023.
3. Learning, UCI Machine. “Pima Indians Diabetes Database.” Kaggle, 6 Oct. 2016, www.kaggle.com/datasets/uciml/pima-indians-diabetes-database.
4. “Mlrose/Mlrose_hiive/Neural /Neural_network.Py.” GitHub, github.com/hiive/mlrose/blob/master/mlrose_hiive/neural/neural_network.py. Accessed 17 Oct. 2023.
5. “Optimization Problem Types¶.” Optimization Problem Types - Mlrose 1.3.0 Documentation, mlrose.readthedocs.io/en/stable/source/opt_probs.html. Accessed 17 Oct. 2023.