Randomized Optimization: A2 - CS7641

Perry Francois-Edwards (pdfe3@gatech.edu)

1 INTRODUCTION

This report will investigate the ability and functionality of randomized optimization (RO) algorithms on maximized optimization problems and neural networks. When comparing the algorithms, reviewing fitness, iterations, function evaluations (FEvals) and clock times, will be imperative. The algorithm's convergence ability will also be observed to highlight characteristics.

1.1 Optimization Problems

Three optimization problems, Knapsack, TSP and FlipFlop, were chosen to highlight the performance of the RO algorithms Random Hill Climbing(RHC), Simulated Annealing (SA), Genetic Algo(GA), and Mutual-Information-Maximizing Input Clustering (MIMIC). Knapsack was chosen because it maximizes the value in the knapsack while remaining within a weight constraint [4]. TSP, originally a minimization problem, was chosen because its structure calls for an algorithm that has the ability to remember optimal paths and information to build the best route between cities and back to its starting city, mixing randomization with searching which may bode well for MIMICs structure or GA's mutation. FlipFlop was chosen because it is simple and the only direct bit-string problem which should bring interesting results, especially from SA's simplistic search and accept approach. Each optimization problem brings a different challenge. Convergence, which is the basis of comparison, is the RO algorithm's ability to reach the local optima. Within the assignment, discussion on whether the algorithm reached the maximum fitness, costs and pace will be related to convergence.

1.2 Dataset for Neural Network Optimization

The Cars Acceptability Evaluation (CAE) dataset was chosen from problem A1 over the Red Wine Quality Measure (RWQM) dataset because it has less features, less target values and it is categorical, while also producing better f1 scores. It is a cleaner dataset to attempt the RO algorithms with more structure and high f1 scores are expected from the RO algorithms.

1.3 Hypothesis

I hypothesize that Knapsack will highlight the SA algorithm because it can accept criteria and keep improving, TSP will highlight MIMIC because of its complex nature to 'remember' past iterations and relationships and FlipFlop will highlight GA because of its crossover ability to hold info with a bit-string. Also, GA will outperform the other algorithms in with the Neural Network.

2 METHODS

2.1 Optimization Problems

For each optimization problem, the RO algorithms generators were tested and optimized based on their tunable hyperparameters. RHC was quite rudimentary and the only modified hyperparameter was restarts. SA's tuned hyperparameters were the initial state value and scheduled decay. GA's tuned hyperparameters were population size and mutation rate. MIMIC's tuned hyperparameters were population size and keep percent. Each algorithm's optimal hyperparameter(s) are referenced and their optimal hyperparameters were used to produce the data in each plot, graph and table. Each RO algorithm was averaged by 10 seeds to produce outputs (fitness, FEvals, etc) that appropriately represent the performance. The data was run with 3 different data sizes, with the medium problem size parameters being the base for the small and large problem sizes to keep continuity. When the data was collected, it was compared using Fitness vs Iteration Plots, FEvals vs Iteration Plots, Fitness vs Problem Size Tables, Time to Max Fitness and Wall Clock Times. When creating the experiment, it was imperative to compare at least two different max iterations. The two max iterations sizes, 1000 and 200, highlight the algorithm's characteristics. 1000 iterations was used for Knapsack to show that with enough time, most RO algorithms can reach the maximum for a problem. For TSP and FlipFlop, 200 iterations were used to accentuate RO algorithm aspects, but both iteration sizes have drawbacks. It's all dependent on what specific problem the user is running. The max attempts value was set to the max iterations value for all algorithms, which can affect time and complexity. The addition of variance curves were too overbearing and were not included on the graphs. In the future, that could provide feedback on algorithms acceptance. All plots were produced for each problem size, but due to similarity and paper space, tables were used to inform on the max fitness scores.

2.2 Neural Networks

The CAE dataset and Neural Network learning curve process from A1 were used to test how well the RO algorithms were in finding the optimal weights via hyperparameters to maximize the f1 score. Similar to A1, a random 80/20 train/test split was needed. Then the train data was scaled, while the test data was OneHot encoded to properly read using the mlrose_hiive functionality. The three RO algorithms used the common hyperparameters from A1, such as activation='relu' and hidden_layer_size=[80]. The other hyperparameters will be tested and found optimally by the NNGSRunner and experimentation. The weight performance comparison is visualized by the learning curves convergence to an f1_score.

3 KNAPSACK

The mlrose_hiive generator, KnapSackGenerator, was utilized. The problem parameters are number_of_items_types = 6, max_weight_per_item = 7, max_value_per_item = 7 and a max_weight_pct = .8. The max_item_count was used to adjust the problem size, where: Small = 10, Medium = 25 and Large = 40.

3.1 Optimization Outcomes

Knapsack highlighted the performance of SA because of the algorithm's ability to find local optima when sampling new points. SA was highlighted over the other algorithms for its ability to reach a high fitness value, low FEvals and low clock time. Over the 1000 iterations, the SA clock time converged to the same optima as GA and MIMIC. Within the FEvals chart, SA had the lowest function evaluations, with MIMIC vastly above the others, which proves it is low cost in terms of computation and time. In regards to clock time, SA had a lower convergence time to the optima fitness than GA and MIMIC. An argument can be made that the fitness of the other algorithms beat that timing. However, for the medium problem size shown in Figure 1 below, if MIMIC converged at 10 iterations and GA converged at 50 iterations, they are still lower (or equivalent) to the SA time of convergence to the optima. Proving that the chart can be deceiving in terms of time. However, with larger (SA is slower) iterations or smaller iterations (SA has lower fitness), SA wouldn't have been highlighted as performing well.

Figures 1 and 2 showcase the medium size problem where **RHC** used restart = 7. **SA** used Init_Temp = 3, Decay = 'ArithDecay'. **GA** used Pop_Size = 50, Mut_Rate = .47. **MIMIC** used a Pop_Size = 1500 and Keep_Perc = .5. Small and Large problem sizes used the same parameter values.

GA and MIMIC did not struggle converging to a maximum, but their time and FEval inefficiencies were highlighted for Knapsack over larger iterations. Knapsack was able to highlight the added complexity within GA and MIMIC due to it being one dimensional, where MIMIC performs better on higher dimensionalities.

If the max_attempts for MIMIC and GA were lowered, that could have improved the efficiency of the algorithm in Figure 2. However, that would also be lowering all other algorithms and it could still prove to be the lowest for SA.

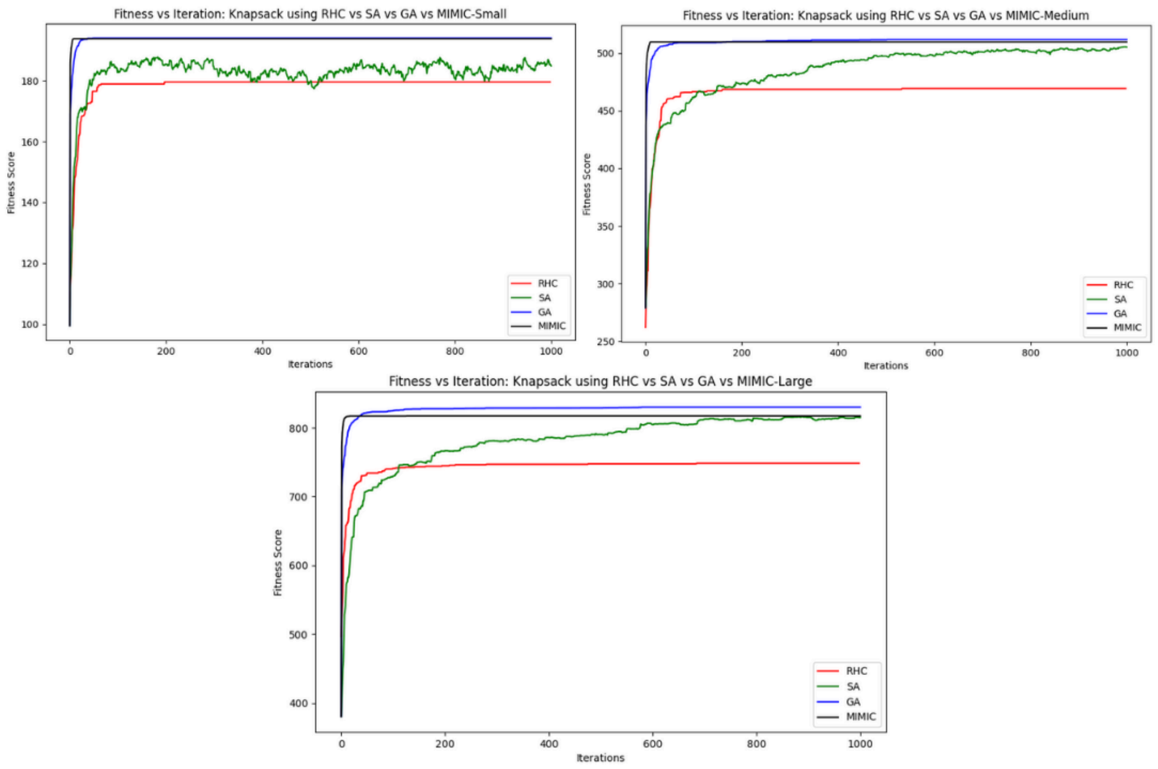


Figure 1—Fitness vs Iterations for all 4 algorithms.

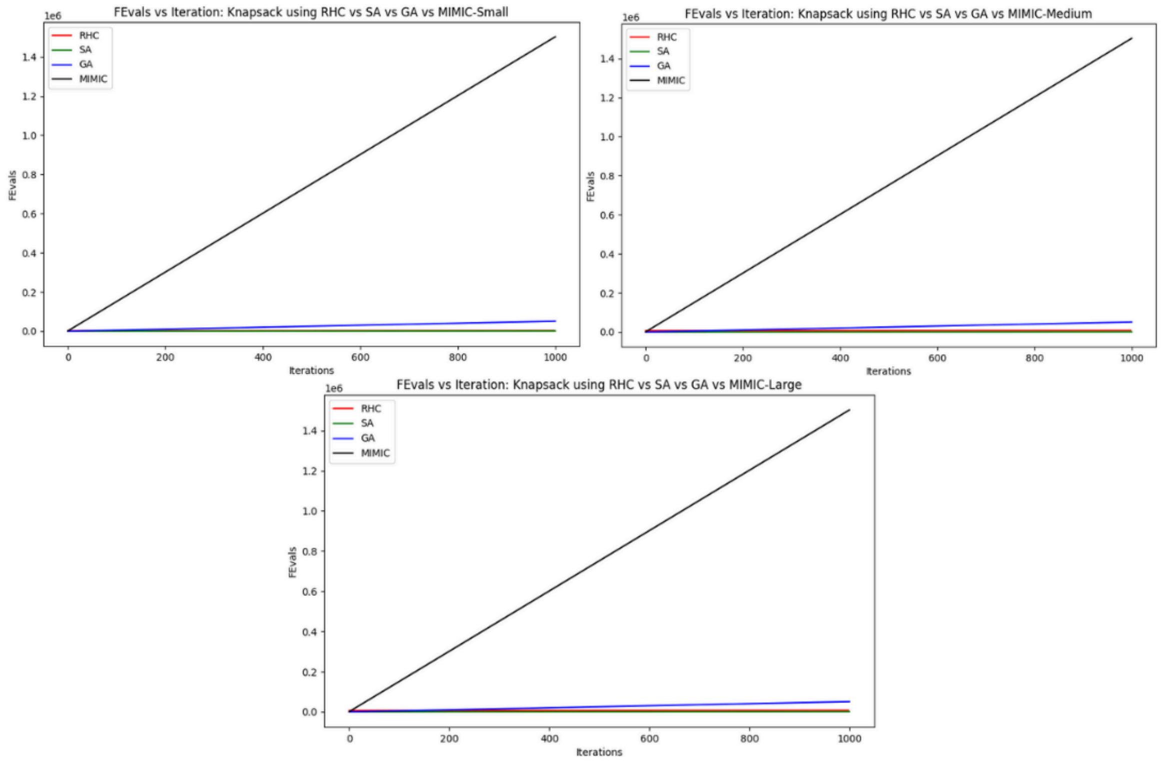


Figure 2—Fevals vs Iteration for all 4 algorithms.

Table 1—The max fitness for each algorithm's size.

Algorithm Size/Max Fitness @ Iter	RHC @ iter	SA @ iter	GA @ iter	MIMIC @ iter
Small	179.6 @ 197	187.8 @ 183	194.0 @ 48	193.8 @ 8
Medium	469.3 @ 533	505.5 @ 998	511.7 @ 502	509.7 @ 10
Large	748.8 @ 685	816.1 @ 888	830 @ 579	817.1 @ 126

Table 2—The iteration clock time to run each algorithm for Knapsack and its associated sizes. Shows that MIMIC is expensive.

Algorithm Size/ Fit Time (s) & Time to Max Fit (s)	RHC	RHC Max Time	SA	SA Max Time	GA	GA Max Time	MIMIC	MIMIC Max Time
Small	.0468	9.2196	.0465	8.5095	1.0803	51.8544	11.0917	88.7336
Medium	.0463	24.6779	.0502	50.0996	1.1066	555.5132	12.3201	123.201
Large	.0439	30.0715	.0468	41.5584	1.0379	600.9441	13.4448	1694.448

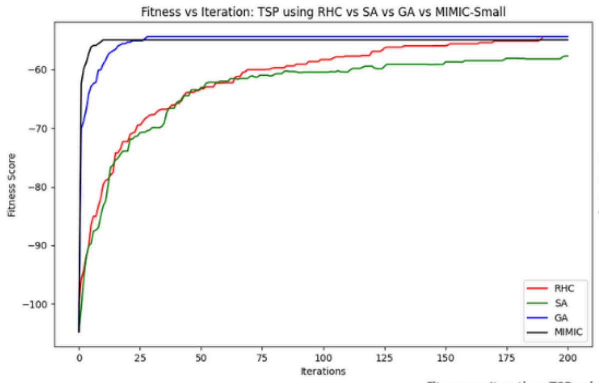
4 TSP

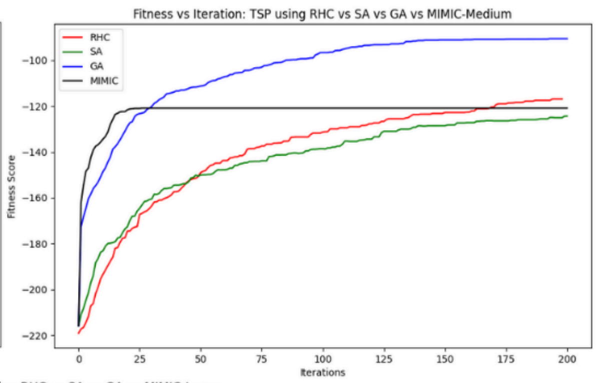
The mlrose_hiive generator, TSPGenerator, was utilized. TSP, a known minimization problem, was converted into a maximization problem by multiplying the fitness parameter by -1. The TSPGenerator parameters used were area_width = 10 and area_height = 10. The TSP problem sizes represent the cities within a grid, where: Small = 10, Medium = 20, and Large = 40.

4.1 Optimization Outcomes

TSP highlighted GA because of its ability to complete local searches with mutation within the grid and how well crossover was able to hold information of the optimal path. GA was highlighted over the other algorithms for its high fitness scoring ability. It was able to consistently converge to the optima higher than the other algorithms, especially as the complexity (problem size) increased. Whereas the other algorithms converged to lower values. GA's FEvals were middle of the pack (Figure 4) and they did not perform better than SA and RHC, but that is a tradeoff for higher performance. GA is more time consuming than RHC and SA, but it becomes a consideration for speed or ability to reach the maximal fitness. If RHC, SA and MIMIC can't reach the optima, the tradeoff of added time becomes inconsequential.

Figures 3 and 4 showcase the medium size problem where **RHC** used restart = 20. **SA** used Init_Temp = .5, Decay = 'ExpDecay'. **GA** used Pop_Size = 70, Mut_Rate = .2. **MIMIC** used a Pop_Size = 700 and Keep_Perc = .4. Small and Large problem sizes used the same parameter values.With TSP, an algorithm like MIMIC seems to be limited because of its dependency tree complexity. Within TSP, the path is not necessarily dependent on another city, which causes issues for MIMIC. SA does decently well in comparison but its iteration count to converge to the optima may take drastically longer. A more robust experiment with different max iterations could prove necessary for further discovery.





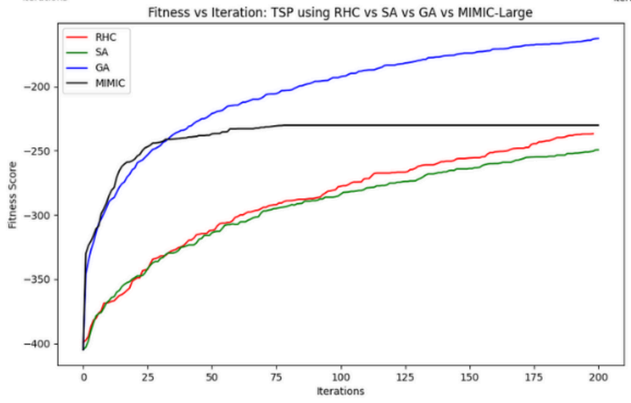


Figure 3—Fitness vs Iterations for all 4 algorithms. Negative fitness values represent an original minimization problem that now converges to a max of zero.

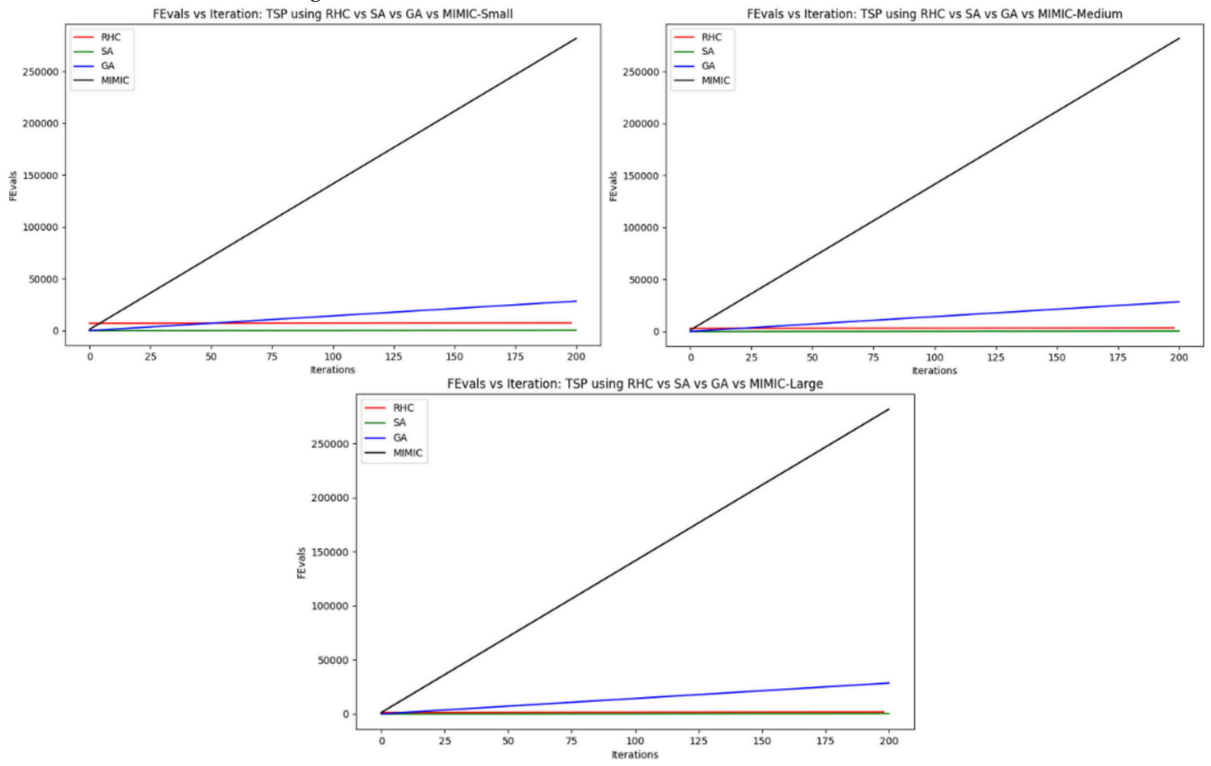


Figure 4—Fevals vs Iteration for all 4 algorithms.

Table 3—The Max Fitness for each Algorithm's size variation.

Algorithm Size/Max Fitness @ Iter	**RHC @ iter**	**SA @ iter**	**GA @ iter**	**MIMIC @ iter**
Small	-54.40 @ 190	-57.71 @ 199	-54.40 @ 50	-54.97 @ 10
Medium	-116.96 @ 194	-124.41 @ 199	-90.62 @ 193	-120.92 @ 26
Large	-236.84 @ 198	-249.34 @ 200	-162.78 @ 200	-230.23 @ 78

Since the max fitness for SA and GA converged at the max iteration, a larger fitness could be found with more iterations since fitness did not plateau.

MIMICs clock time grows at magnitudes higher with problem complexity, in comparison to the Knapsack problem.

Table 4—The iteration clock time to run each algorithm for TSP and its associated sizes. Shows that MIMIC is expensive.

Algorithm Size/ Fit Time (s) & Time to Max Fit (s)	**RHC**	**RHC Max Time**	**SA**	**SA Max Time**	**GA**	**GA Max Time**	**MIMIC**	**MIMIC Max Time**
Small	.0123	2.337	.0145	2.8855	.8881	44.05	44.7089	447.089

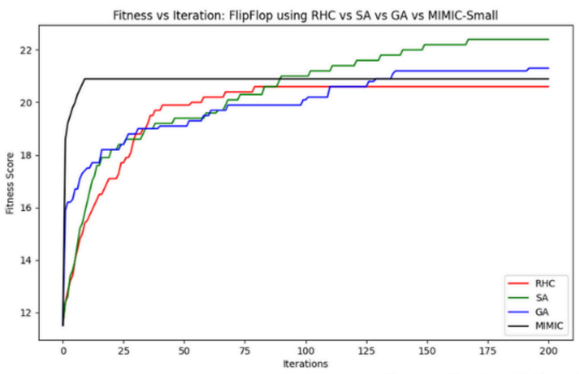
Medium	.0150	2.91	.0156	3.1044	1.2428	239.8604	96.9701	2521.2226
Large	.0156	3.089	.0166	3.32	1.8181	363.62	230.1303	17950.1634

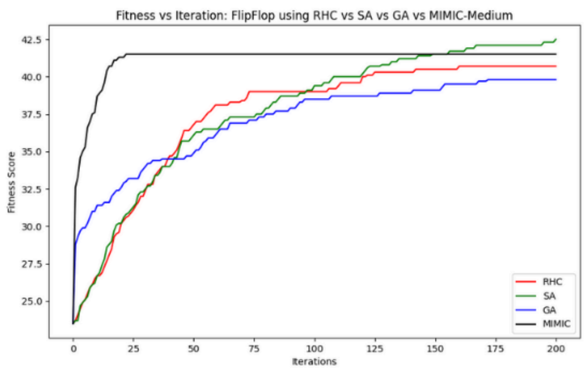
5 FLIP FLOP

The mlrose_hiive generator, FlipFlopGenerator, was used. The FlipFlop problem sizes represent the size of the bit-string, where: Small = 25, Medium = 50, and Large = 75.

5.1 Optimization Outcomes

FlipFlop highlighted MIMIC because of its dependency tree structure in relation to a bit-string. MIMIC was able to perform well because it executes well on high dimensional, complex, spaces and variables with dependencies. MIMIC was mainly highlighted over the other algorithms because this is the first time the FEvals were the lowest and they remained consistent with an increased problem size. Whereas, the SA algorithm that performs well, has increasing FEvals, when SA usually has the lowest FEvals. Though MIMIC is still time expensive in comparison to other algorithms, its ability to reach a high fitness score at low iterations is impressive. Even when reducing max_attempts, MIMIC performed exceptionally. Here, GA's performance may be improved with more complexity and a larger problem size. With a lower iteration and larger problem size, MIMIC has the potential to perform the best repeatedly on one iteration. Figures 5 and 6 showcase the medium size problem where **RHC** used restart = 0. **SA** used Init_Temp = .2, Decay = 'GeomDecay'. **GA** used Pop_Size = 5, Mut_Rate = .4. **MIMIC** used a Pop_Size = 45 and Keep_Perc = .55. Small and Large problem sizes used the same parameter values.





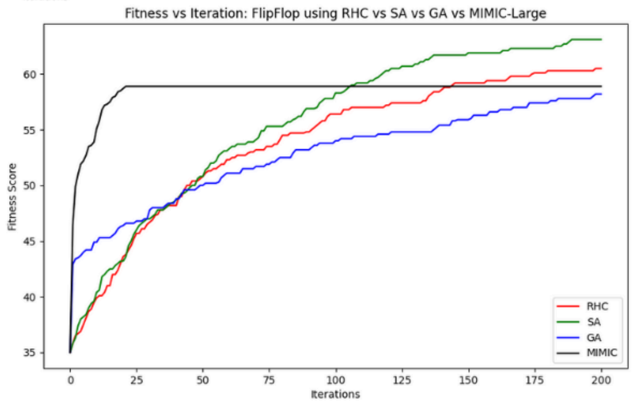


Figure 5—Fitness vs Iterations for all 4 algorithms.

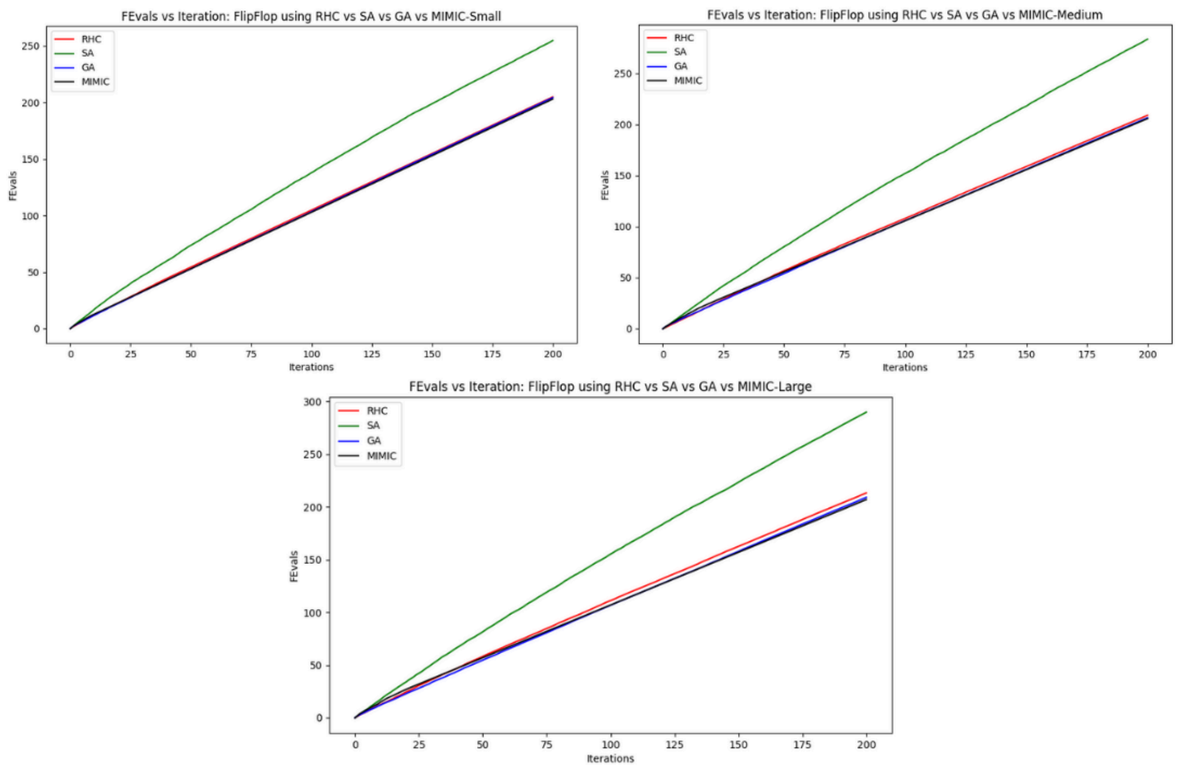


Figure 6—Fevals vs Iteration for all 4 algorithms.

Table 5—The max fitness for each algorithm's size.

Algorithm Size/Max Fitness @ Iter	RHC @ iter	SA @ iter	GA @ iter	MIMIC @ iter
Small	20.6 @ 79	22.4 @ 167	21.3 @ 192	20.9 @ 9
Medium	40.7 @ 160	42.5 @ 200	39.8 @ 172	41.5 @ 22
Large	60.5 @ 198	63.1 @ 189	58.2 @ 198	58.9 @ 21

Table 6—The iteration clock time to run each algorithm for FlipFlop and its associated sizes. Shows that MIMIC is expensive.

Algorithm Size/ Fit Time (s) & Time to Max Fit (s)	RHC	RHC Max Time	SA	SA Max Time	GA	GA Max Time	MIMIC	MIMIC Max Time
Small	.0052	.4108	.0043	7.181	.0171	3.2832	13.3186	119.8674
Medium	.0064	1.024	.0075	1.5	.0310	5.332	52.8520	1162.744
Large	.0086	1.7028	.0101	1.9089	.0304	6.0192	220.0634	4621.3314

6 NEURAL NETWORK

The mlrose_hiive, NNGSRunner and NeuralNetworks, were used, along with RHC, SA and GA RO algorithms to find the optimal weights. The following parameters, max_iters = 1000, max_attempts = 100, and learning_rate = .5, were consistent for all algorithms. For **RHC**, restarts=5. **SA**, schedule=GeomDecay(7), and for **GA**, Mut=.45 and Pop=75.

6.1 Learning and Validation Curves

Overall, the performance of the RO algorithms on the CAE dataset performed well, as expected. The learning curves converge to optimal f1 scores between the train and test around .65, which is similar to the value discovered in Supervised Learning. The datasets' categorical structure made it achievable for these algorithms to succeed. However, of all the algorithms, RHC performed the best. I think this happened by 'random' chance and further experiments would prove that SA's nature to explore and exploit would provide a higher f1 score by changing the decay values. Thought RHC could perform even better with more RHC restarts. The expectation was that GA would perform the best with a neural network in general because of its increased complexity with population iterations increasing, versus the other algorithms not having that ability.

To improve upon performance for further testing, I would refine all parameters that were consistent, especially hidden layer size. During experimentation, lowered hidden layers provided a higher accuracy score, which could translate to better performing f1 scores. For SA specifically, changing the parameter within specific decays would be an emphasis.

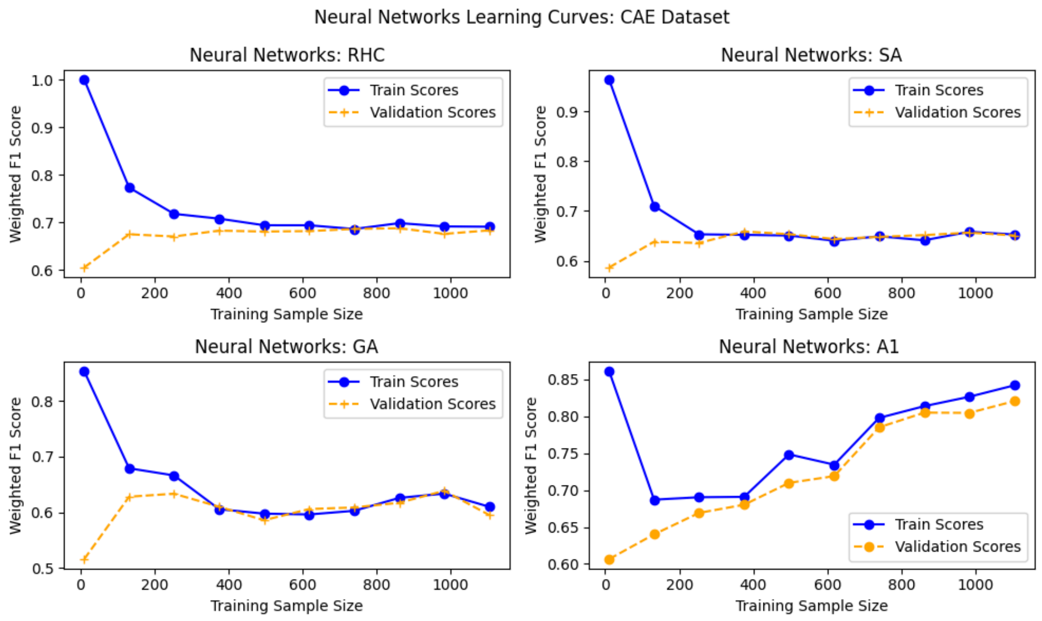


Figure 7—The Learning Curves RHC, SA, RHC and A1 with optimal algorithm specific hyperparameters.

7 CONCLUSION

Overall, this experiment yielded interesting results about the RO algorithm's characteristics and proved that the type of problem plays a large role in the expected outcomes, besides MIMICs ability to be slow, and each excel in different aspects. Whether it be minimizing iterations, time, and FEvals or maximizing fitness, a specific algorithm can suit the problems needs. Though they can be tuned for a result, they all have the ability to improve. RHC could run more restarts. It's random, but the more available data, the higher probability a new optima is found, while simultaneously being inexpensive. It was interesting that SA's performance for each problem was dependent on one of its three decay schedules. SA was also improved by a lower initial temperature. GA performed better (time, fitness, FEvals) with a mutation value of less than .5. GA and MIMIC improved clock times and FEvals with reduced max attempts, which should have been attempted here. MIMIC improved its fitness score with an increased population size, but it also increased its clock time, which hurt performance. I would have also used parameters that were not investigated, like GA's hamming values. The original hypothesis was also wrong because TSP did not highlight MIMIC at all and GA did well with FlipFlop, but its performance was similar to the other optimization problems. GA also underperformed expectations with the Neural Network.
In retrospect, further tuning the algorithms on the small and large problem sizes, besides miniscule changes, to their peak optimal performance could have yielded different outcomes in terms of peak fitness and clock times.

8 REFERENCES

1. Bohanec,Marko. (1997). Car Evaluation. UCI Machine Learning Repository. https://doi.org/10.24432/C5JP48.
2. Mimic: Finding Optima by estimating probability densities. (n.d.-a). https://faculty.cc.gatech.edu/~isbell/papers/isbell-mimic-nips-1997.pdf
3. Aditya Pal and Theodore LaGrow, & Commenter, A. W. (n.d.). *Home*. OMSCS 7641 Machine Learning. https://sites.gatech.edu/omscs7641/2024/02/19/simulated-annealing-methods-and-real-world-applications/
4. *Educative answers - trusted answers to developer questions*. Educative. (n.d.). https://www.educative.io/answers/what-is-the-knapsack-problem