Andy Cheng

CS 7641

Assignment 2: Randomized Optimization

3/6/2024

1. Introduction

Four different randomized optimization (RO) algorithms were applied to three different maximization problems to analyze the impacts and interactions that each method had on the data. The four algorithms are Random Hill Climb (RHC), Simulated Annealing (SA), Genetic Algorithm (GA), and MIMIC. The first problem is Six Peaks which is a combinational optimization with a search space containing multiple local optima to navigate. The second problem is Max K-Color which is a graph optimization with the need to satisfy the constraint of neighboring vertices having different colors. The third problem is Knapsack which is a classic combinational optimization with the goal is to maximize the value of items carried in a limited capacity sack. These three problems were chosen for their different levels of problem complexity and type such as graph vs combinational and having local optima vs not having local optima to navigate. These difference allow unique performance traits of the four algorithms to appear which develops a more in-depth understanding of these RO algorithms. For instance, Six Peaks can be used to test the ability of RHC and SA to escape local optima using their respective random initialization and temperature decay properties. GA's ability to rapidly explore the solution space with its population set and mutations will also be tested thoroughly through the Max K-Color problem growing in complexity. Finally, MIMIC will be able to showcase its intrinsic ability to discover underlying structures in the problem during the Knapsack problem. RHC, SA, and GA were also used to replace Gradient Descent (GD) as the algorithm used to calculate the network weights for a neural network trained on the Apple Quality dataset from Assignment 1. This data set is on apple quality with features such as size, weight, sweetness, crunchiness, juiciness, ripeness, and acidity with an evenly split overall classification of Good/Bad (Elgiriyewithana, 2024). There are 4000 total data points in the apple data set. This data set is interesting as it represents a relatively limited feature space with seven features that should identify which weights algorithm and corresponding hyperparameters perform better accordingly. Through both experiments, the idea of what constitutes the "best" algorithm performance will be explored in terms of fitness/accuracy, iterations required, wall time, function evaluation, and convergence to try and reach a consensus of the optimal algorithm for different problem types.

2. Methodology

a. Optimization Problems

The general methodology for each optimization problem analysis was similar with the same steps being performed for each problem. All analysis was conducted using the mlrose Randomized Optimization and Search package framework (Hayes, 2019). Problem definition

was first done to achieve a small, medium, and large problem size for each of the optimization problems. This process involved changing the number of dimensions for the Six Peaks problem, the number of edges for the Max K-Color problem, and the number of items for the Knapsack problem. These values were set to 5, 20, and 50 for small, medium, and large problem sizes respectively for all three problems. All problems were then set to maximization problems to stay consistent when comparing fitness performance. Each test was run five times and averaged to account for the randomness. The algorithms' hyperparameters were then tuned for each problem and problem size, resulting in nine different tunings for each hyperparameter. The hyperparameters tuned for each optimization problem are shown in Table 1:

Table 1 - Tuned Hyperparameters for Optimization Problems

RHC	SA	GA	MIMIC
Max Attempts	Schedule Type	Population Size	Population Size
# of Restarts	Schedule Temperature	Mutation Rate	Keep Percentage

The tuning criteria was maximizing fitness regardless of iterations or wall time concerns to simplify the parameter grid search. These tuned hyperparameters were then used on their respective problems to determine final performance metrics such as fitness and Fevals against iterations and wall time. The results for the medium size problems were then plotted for the fitness vs iterations/wall time curves as well as the Fevals vs iterations curve for each of the three optimization problems. A bar chart comparison of final iteration fitness for each problem size was also generated to get insight on the performance trends with increasing problem size.

b. Neural Network

The neural network analysis methodology involved an iterative process where results were generated, and model parameters were tweaked as a result. Preprocessing was first done to split the apple quality dataset into training/testing sets. The features consisted of all float values for both sets and were normalized to achieve zero mean and standard deviation of 1 for each feature. The classification was mapped to integers 0 for 'bad' and 1 for 'good' apple quality as well. 85% of the processed data was separated to be the training set with the remaining 15% saved to be the final test set to evaluate on after tuning. Finding the ideal hyperparameters for each algorithm was then conducted by first tuning the parameter that had the greatest predicted impact using learning curves then using this parameter to tune the second hyperparameter. These curves display both training accuracy and 5-fold cross validation accuracy. The ideal set of hyperparameters for each algorithm was then saved off and used to generate final learning curves. The final learning curves display model accuracy vs iterations to determine the convergence for each algorithm and overall model performance. Each tuned model was then evaluated against the test set to determine final performance in terms of accuracy and wall time.

The hyperparameters tuned for the neural network problem are shown in Table 2:

Table 2 - Tuned Hyperparameters for Neural Network

GD	RHC	SA	GA
Learning Rate	Learning Rate	Learning Rate	Learning Rate
# of Iterations	# of Iterations	# of Iterations	# of Iterations
~	# of Restarts	Schedule Temperature	Population Size

Learning rate was the most important parameter to tune for each algorithm as the model would not train at all if the learning rate was too high or low. The hyperparameters chosen for the RO algorithms were the most impactful coefficients found during the tuning process for the optimization problems.

3. Optimization Problems

a. Six Peaks (SA)

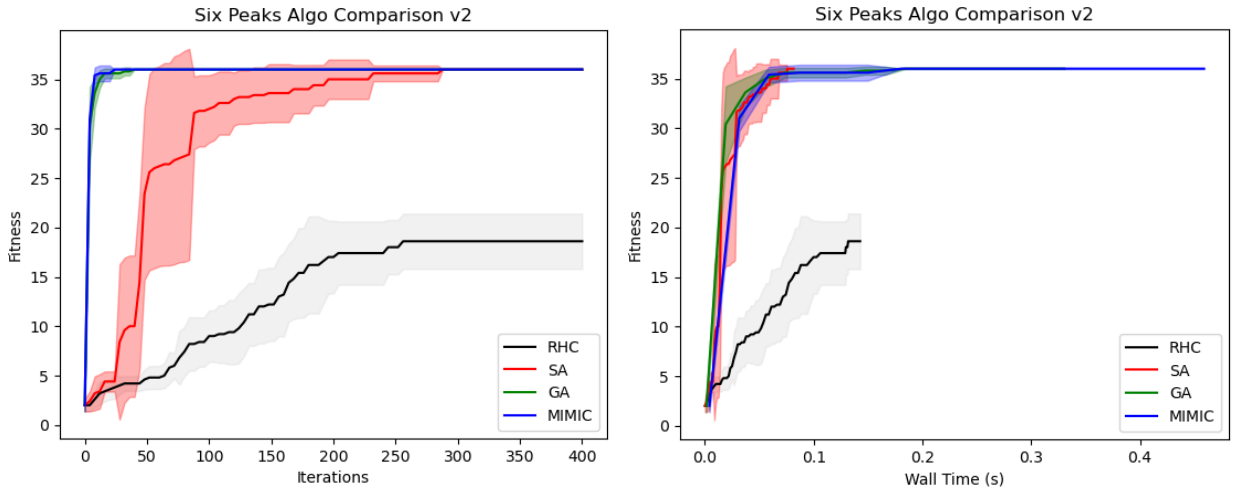

Figure 1 - Six Peak Fitness Curves

Figure 1 shows the fitness curves against both iterations and wall time. The graphs are limited to 400 total iterations which show the number of iterations needed to converge for both MIMIC and GA is small compared to the number needed for RHC and SA. Random Hill Climbing could not find the maximum fitness solution compared to the other three algorithms The wall time plot shows a different story though when compared to just looking at iterations. In this figure, the simulated annealing line converges to the maximum fitness at about the same time as both GA and MIMIC. This result indicates that SA can run a lot more iterations per unit of time when compared to the other algorithms. The iterations figure on the left also shows another interesting behavior of SA where there are multiple plateau regions with high variance before performance suddenly improves. This observation is consistent with the Six Peaks problem having multiple local optima that algorithms can converge to rather than the highest fitness solution. Simulated Annealing explores the search space by accepting solutions that worsen the objective function with a certain probability controlled by temperature, allowing it to escape local optima. These escapes can be seen with the step function-like curve for SA and is a benefit

of the algorithm when compared to the other three. If the GA algorithm was not tuned properly, then it could become stuck on a local optima without being able to escape like SA.

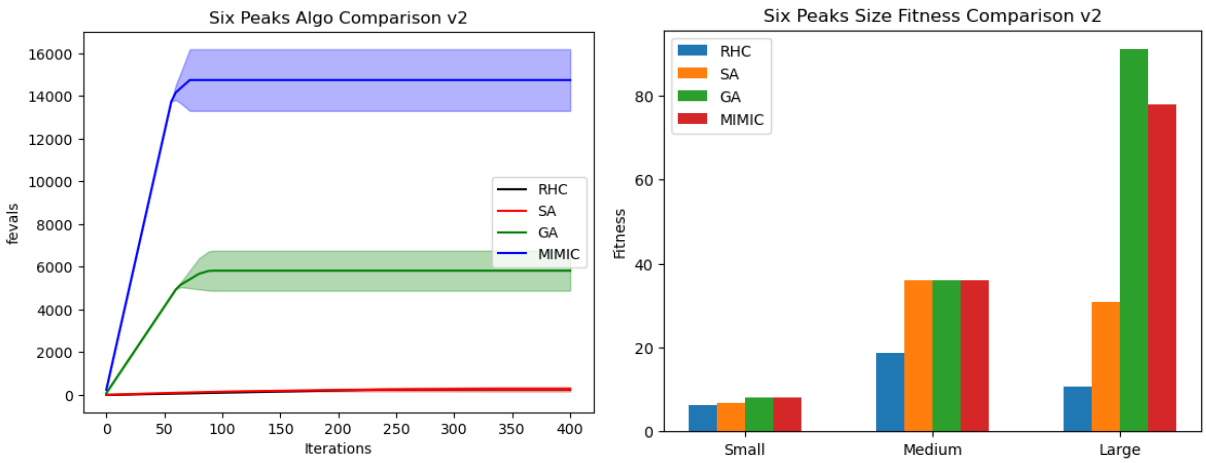

Figure 2 - Six Peaks Feval and Problem Size Comparison

Figure 2 shows more behavior highlighting SA. The Feval plot on the left builds on the idea that SA can process more iterations per unit of time as the number of function evaluations per iterations is multiple order of magnitude smaller than either GA or MIMIC. SA is an efficient algorithm for the medium sized Six Peaks problem in terms of computational resources with GA and MIMIC being much more demanding algorithms. This observation is a huge positive for SA as it can find the global optima with several thousand less function evaluations. The problem size comparison figure shows a drawback of SA, however, as the performance is much worse for the large problem when compared to the medium problem. GA and MIMIC handle the higher complexity problem much better than SA.

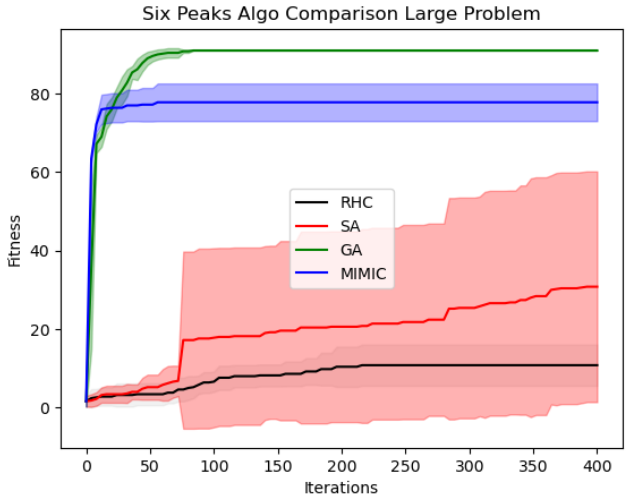

Figure 3 - Six Peaks Large Problem Fitness Curve

The iterations curve for the larger problem was examined as well in Figure 3 and SA was still converging to the optimal solution after 400 iterations, but very slowly which matches its deterministic convergence property that guarantees the global optimum will be eventually reached if tuned properly. Given enough iterations, it would also find the global optimum.

b. Max K-Color (GA)

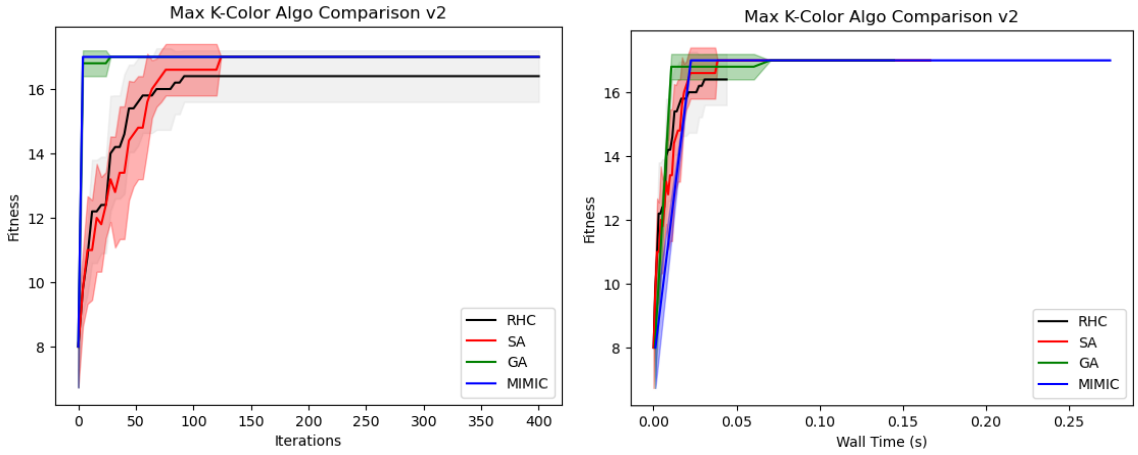

Figure 4 - Max K-Color Fitness Curves

Figure 4 shows fitness curves for the Max K-Color problem. Here, GA can be seen taking very few iterations to converge to the optimal solution. MIMIC takes a similar number of iterations and similar wall time to converge. GA takes the shortest amount of wall time to reach a good solution with low variance but gets stuck on a local optima for a while before escaping and finding the global optimum. This escape behavior makes sense as genetic algorithms maintain a population of candidate solutions throughout the optimization process. This population-based approach enables genetic algorithms to simultaneously explore multiple regions of the solution space. The population-based approach helps maintain high solution diversity alongside the population mutating naturally to be able to escape local optima. This algorithm design is very beneficial for finding high-quality solutions to the Max K-Color problem.

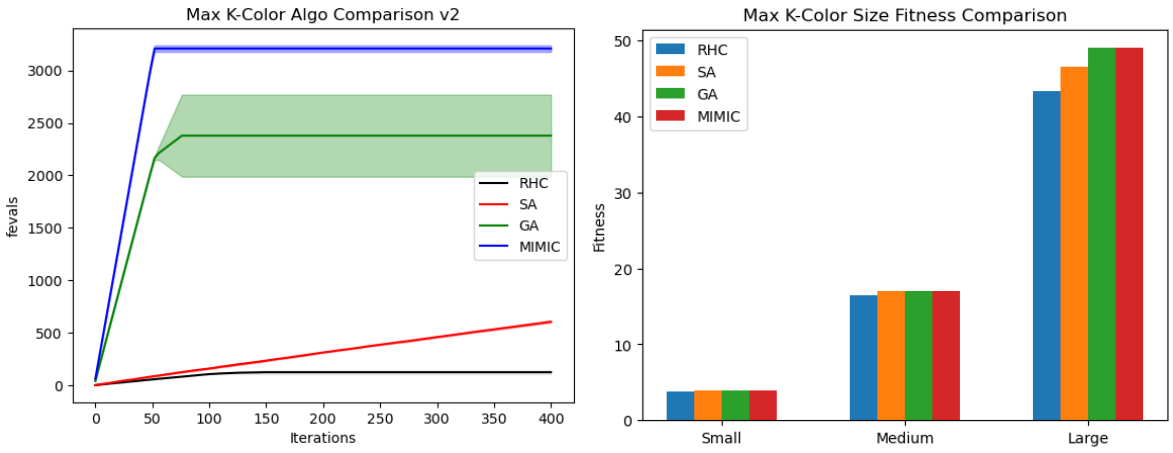

Figure 5 - Max K-Color Feval and Problem Size Comparison

Figure 5 shows that GA performs very well for the Max K-Color problem across all three problem sizes. It finds the global optimum for all three sizes. This performance makes sense as GA can easily represent this problem with each individual in the population representing a vertex of the graph with a coloring. The hyperparameter tuning for population size supports this correlation with [5, 40, 80] corresponding to the [S, M, L] problem size populations respectively. As the problem size grew, the population size grew accordingly as well with higher populations being needed to explore more complex solution spaces more efficiently. The Feval plot shows another reason why GA is the best algorithm for this problem. GA finds the highest fitness for all problem sizes like MIMIC, but it takes over 500 less function evaluations to do so in comparison. GA is more computationally efficient for this problem than MIMIC which matters when expanding the problem complexity to even larger sizes.

c. Knapsack (MIMIC)

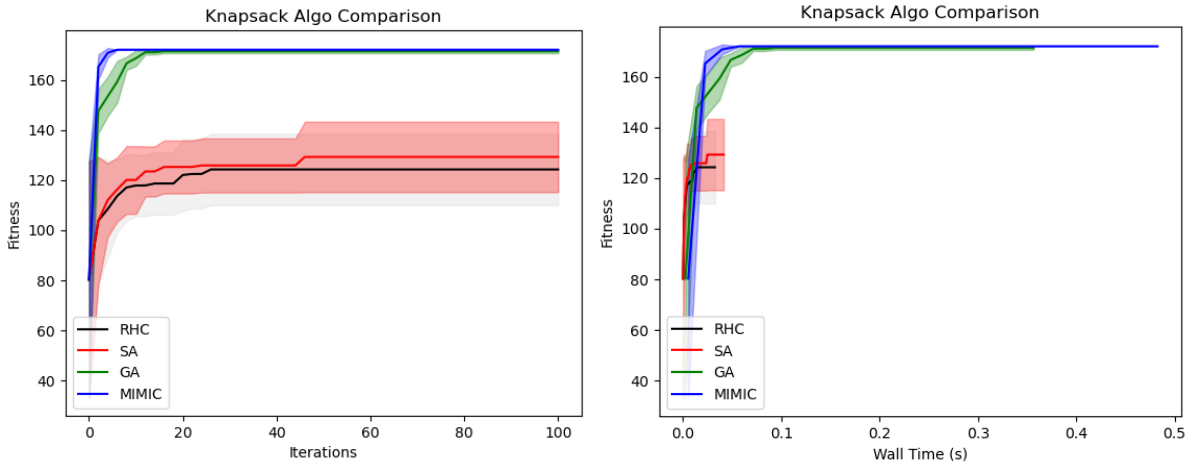

Figure 6 - Knapsack Fitness Curves

Figure 6 shows that the MIMIC algorithm performs the best for the Knapsack problem with MIMIC taking the shortest number of iterations and wall time to converge at the highest fitness. GA is a close second with the other two algorithms not being able to find the global optimum. This behavior makes sense as MIMIC is designed to converge faster and more reliably than other algorithms. It is the best at identifying underlying structures to the problem such as the correlation between items and their contribution to the fitness function in terms of value and weight. It is the only algorithm tested that passes information about the cost function forwards between successive iterations which helps find the highest fitness solution. For a problem like Knapsack where there is a clear connection between the items, their weight and value, and the maximum value that can be carried in the sack, MIMIC can key in on this structure and quickly converge to the best solution. Problems like Six Peaks, where the underlying structure is a lot less clear, do not benefit as much from this ability as much as shown in Figure 2 before where GA finds a higher fitness solution for the larger problem size.

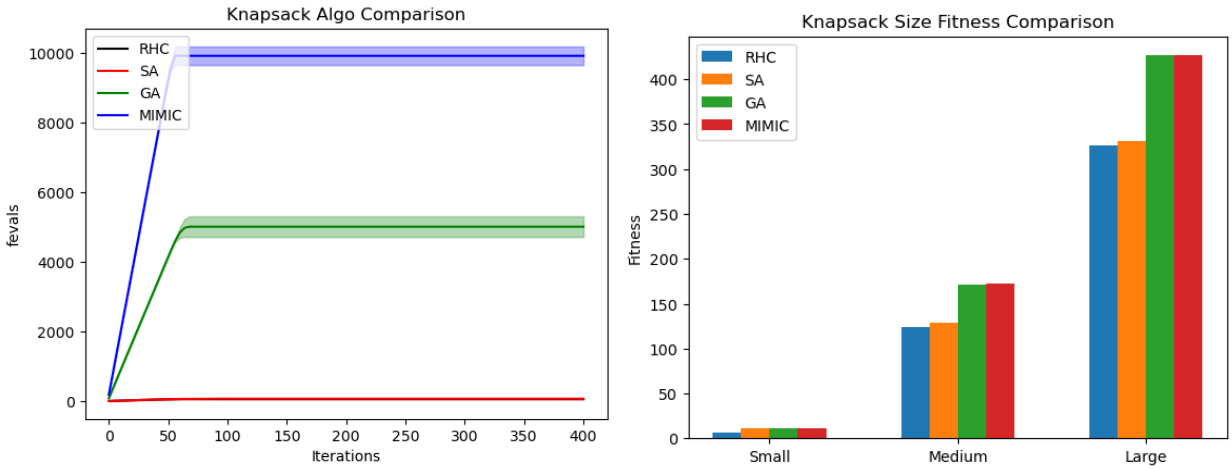

Figure 7 - Knapsack Feval and Problem Size Comparison

Figure 7 and the problem size comparison further illustrates the strength of MIMIC for the Knapsack optimization. MIMIC found the highest fitness for all three problem sizes, along with GA. MIMIC does converge faster than GA, as shown in Figure 6, which makes MIMIC the best algorithm for this problem. The Feval plot shows that MIMIC has a high number of function calls per iteration before convergence. MIMIC is a costly function in terms of computational resources compared to the other algorithms, but it is worth it for the Knapsack problem as it finds the highest fitness solution in the least amount of time. For problems with underlying relations like Knapsack, MIMIC is the best RO algorithm.

4. Neural Network Weights Algorithm Comparison

a. Final Learning Curves

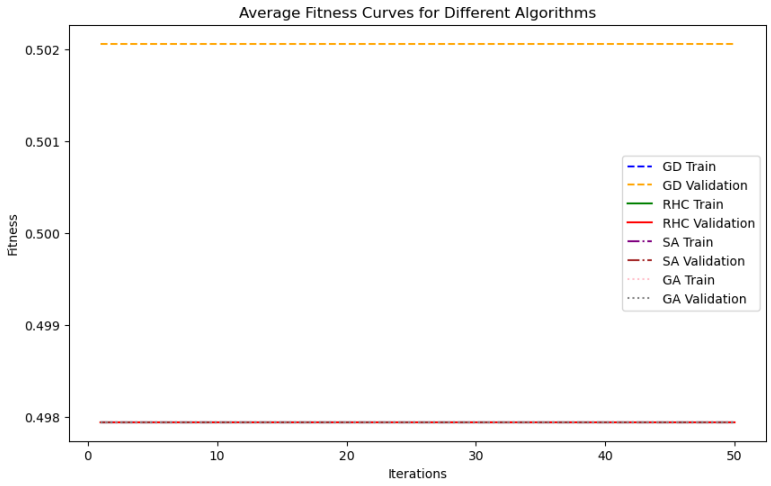

Figure 8 - Neural Network Pre-tuning Learning Curves

Figure 8 illustrates the performance of the algorithms before tuning occurred. More specifically, it showed performance before the right learning rate was found for each of the four different algorithms. If the learning rate was not in the right range, the neural network would not learn

at all and be stuck at 50% accuracy. The mlrose library and the SKLearn underlying functions do not automatically determine learning rate which necessitated manual tuning. Getting this learning rate was vital to properly training the network and these rates varied wildly between each algorithm. These values were [0.0001, 0.36, 0.31, 0.01] for [GD, RHC, SA, GA] respectively.

Gradient descent was very sensitive to overshooting corrections to the network weights and having the weights diverge to infinity which necessitated the low learning rates. This occurs because GD takes excessively large steps at high learning rates, causing it to overshoot the optimal solution continuously until it diverges completely. On the other hand, RHC and SA required high learning rates to get the weights to converge in a timely number of iterations. This behavior makes sense for RHC and SA as a higher learning rate allows much quicker exploration of the solution space which can help with escaping local optima. These algorithms do not have the diverse initialization of GA with its population set and are not GD with its ability to change weights by varying magnitudes dependent on the loss function gradient. RHC and SA require higher learning rates to allow the random optimization behavior to properly explore the network weights solution space. GA had a middle ground learning rate which makes sense with GA being able to explore a diverse solution set with its population-based approach alongside population mutations promoting continuously unique network weight solutions.

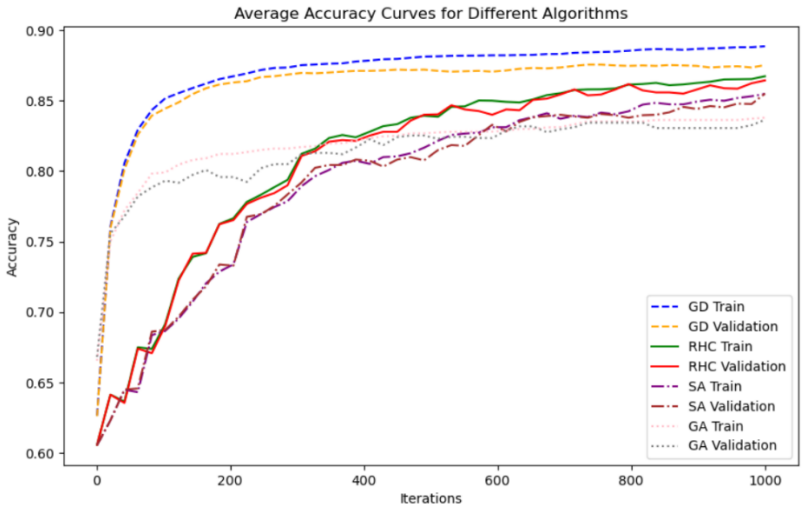

Figure 9 - Neural Network Final Learning Curves

Figure 9 shows the final learning curves post tuning all the hyperparameters chosen. For this plot, gradient descent was able to converge to a high accuracy in the lowest number of iterations. GA was also able to converge in a low number of iterations but had the worst performance in terms of accuracy. RHC and SA took more iterations to converge and continued to improve accuracy steadily with more iterations. There was not an accuracy plateau until near 1000 iterations. This behavior matches the need for these two algorithms to have a high learning rate as they take many more iterations to converge compared to other algorithms through their slow exploration of the weights solution space. If the learning rate is too low, then the learning curves show the neural network is not learning at low iteration counts as

shown in Figure 8. In terms of bias and variance, all four algorithms perform well with small differences between the train and validation accuracy. This performance indicates that all algorithms are not overfitting to the data and that the hyperparameters selected were reasonable for the apple quality problem.

b. Final Results for Test Set

Table 3 - Test Set Misclassification and Wall Time Results

Dataset	Gradient Descent	RHC	SA	Genetic Algo
Apple Quality (Accuracy Rate)	0.872	0.853	0.822	0.777
Apple Quality (Wall Train/Predict Time (s))	6.06	3.29	3.46	14.08

Table 3 shows the final accuracy performance for each of the algorithms when trained on the training set and tested with the hold-out test set. The wall time to train and predict was also measured as another comparison metric. SA and GA are the worst performing with GA in particular showcasing low accuracy and high wall time. GA is too complex of a model for weights training here and the wall time makes it unfeasible to use in comparison to the other algorithms. Gradient descent does the best in terms of test accuracy for the apple quality data set. However, RHC does only slightly worse than GD and takes almost half the wall time in comparison. This wall time difference makes the 0.02 accuracy difference not worth the trade-off compared to RHC. RHC is a very efficient algorithm compared to GD and GA and can run more iterations per amount of time in comparison. For the apple quality dataset, RHC is the best algorithm tested due to its balanced time efficiency and good accuracy performance.

5. Conclusion

The optimization problems chosen, and the neural network weight algorithm comparison provided interesting contrasts that highlighted the strengths and weaknesses of these random optimization algorithms. Six peaks showcased the ability of simulated annealing to always converge to the global fitness optima given enough iterations. SA also was a very efficient algorithm taking low computation resources in terms of Fevals to arrive at the global optima in a similarly low wall time. Max K-Color showcased the ability of the genetic algorithm to find the best fitness solutions quickly in terms of iterations and wall time. GA was able to find this global optimum for all three problem sizes as well. GA is well suited for this optimization problem with its ability to represent each member of its population as different graph vertices with different colors which allows the algorithm to quickly explore the solution space. The mutation rate also allowed GA to escape the local optima quickly and find the global optima. Knapsack showcased the ability for MIMIC to find underlying structures in the problem such as the connection between the items and maximizing the value in the sack. No other algorithm tested here has

this ability and makes MIMIC the best algorithm for problems with clear underlying connections due to its fast convergence to the highest fitness solution. As for the neural network, the key take away was the importance of finding the correct learning rate for each of the algorithms used. Each of the algorithms intrinsic exploration behavior dictated these learning rates and not finding an optimal rate causes the network to be unable to train to the data. RHC was the best algorithm for the apple quality dataset used due to its great wall time efficiency while maintaining a high level of accuracy.

For all these problems, the idea of "best" was evaluated and a mixture of result performance in terms of fitness/accuracy and computational efficiency in terms of number of iterations/wall time/Fevals was used to determine the "best" algorithm for each problem. Just having the highest result performance did not equate best as shown with the neural network problem having the wall time almost doubled for a tiny performance increase. A balance of these metrics is needed to come to a definition of "best" and that balance will come down to the overall goal of each machine learning problem. A potential research area to explore further is to conduct a more thorough search of hyperparameters for SA to explore the idea of SA always finding the global optima given enough iterations. It would be interesting to see if fine tuning the temperature decay behavior to balance exploration rate and global optima refinement would allow SA to match GA and MIMIC in terms of performance for all three problems.

6. References

Elgiriyewithana, N. (2024, January 11). *Apple Quality*. Kaggle. https://www.kaggle.com/datasets/nelgiriyewithana/apple-quality

Hayes, G. (2019). mlrose: Machine Learning, Randomized Optimization and Search package for Python. https://github.com/gkhayes/mlrose. Accessed: 3 March 2024.