Kevin McCarville
CS - 7641 - Machine Learning
2/12/2024
Assignment 2

Optimization Problems:

In the following paper, I will be comparing the strengths and weaknesses of 4 separate optimization algorithms: Hill Climbing, Simulated Annealing, Genetic optimization, and MIMIC. To do this I will compare their performance on 3 separate discrete valued optimization problems: Count the Ones, 4 Peaks, and Graph Coloring:

- Count the ones - simply entails maximizing the total number of ones in a binary string. The binary string for this exercise will be 100 characters. This problem is very straightforward which should shed some light on which algorithms perform best given uncomplicated solutions spaces.
- 4 Peaks - The 4 peaks problem is defined by maximizing between the number of leading 0s and the number of trailing 1s. If both the number of leading 0s and the number of trailing 1s are above a threshold defined as a fraction of the size of the problem, a bonus is given. For this example, I will be using a string of length 50 and a bonus of 100 and a threshold of 20. This means there are two obvious local optimas of all 1s or of all 0s. These optima have a large basin of attraction. But the bonus creates two global optima by just getting over the threshold of necessary leading 0s or trailing 1s, so that both are over the threshold to receive the bonus. This should be difficult to reach as it's not very obvious the bonus exists making it difficult for hill climbing to reach it. In other words, the basin of attraction is small. The higher the bonus threshold is, the narrower its basin of attraction is and the harder the global optima will be to reach. Which algorithms get caught in the tempting local maxima will be interesting to see.
- Graph Coloring - The graph coloring problem involves assigning colors to the vertices of a graph in such a way that no two adjacent vertices share the same color, while minimizing the total number of colors used. Formally, given an undirected graph G = (V, E), where V is the set of vertices and E is the set of edges, the goal is to find a coloring function c: V → {1, 2, ..., k} such that adjacent vertices are assigned different colors and k is minimized. This problem also takes the form of coloring a map such that neighboring countries all have different colors. For my example, I will assume only 2 colors are available and will experiment with different graph sizes. This algorithm involves more structure to the data and the relationships between the bits is much more intricate than in the previous two problems.

I will now briefly describe the functionality of each optimization algorithm.

Algorithms:

Random Hill Climbing (RHC) - a standard hill climbing approach where optima are found by exploring a solution space and moving in the direction of increased fitness on each iteration.

Simulated Annealing (SA) - a variant on random hill climbing that focuses more on the exploration of a solution space, by randomly choosing sub-optimal next-steps with some probability based on the initial temperature and cooling hyperparameters. This increases the likelihood of finding global optima instead of getting stuck in local optima.

Genetic Algorithms (GA) - a subset of evolutionary algorithms that produce new generations based on fitness of prior generations.They start with a population of potential solutions represented as individuals, encoded as binary strings. Through the iterative process, individuals are evaluated based on their fitness, which represents how well they solve the problem. During selection, individuals with higher fitness have a higher chance of being chosen for reproduction.

Crossovers, inspired by biological recombination, involve exchanging genetic information between two parent individuals to create offspring. This process helps explore different combinations of traits from the parent solutions. Meanwhile,

mutations introduce random changes in the offspring's genetic information, allowing for exploration of new areas in the search space.

MIMIC - or the "Mutual Information Maximizing Input Clustering" algorithm, is a probabilistic optimization algorithm that operates by modeling the probability distribution of the search space using a multivariate normal distribution. It begins by sampling a population of potential solutions from this distribution. During each iteration, it refines this distribution to better represent the promising areas of the search space. Through this iterative process of sampling, estimating dependencies, and updating the distribution, MIMIC aims to efficiently explore and exploit the structure of the problem space, ultimately converging towards optimal or near-optimal solutions. Its strength lies in its ability to handle complex optimization problems with large solution spaces by effectively capturing dependencies between variables.

Below, I will analyze each problem in turn and assess which optimization problems performed the best over these problems. First, one important observation I found while working with these algorithms is that there are a myriad of hyperparameters to tune. One could fairly easily manipulate these hyperparameters to obtain whatever cardinal ranking they desired. Here are the hyperparameters for each model:

Random Hill Climbing - generations, max iterations to improve before exiting, random starts
Simulated Annealing - generations, max iterations to improve before exiting, random starts, initial temp, cooling rate
Genetic Algorithms - population size, generations, mutation rate, tournament_size
MIMIC - population size, sample size, generations

For clarity, max iterations to improve, is how many iterations the hill climbing algorithm will do trying to improve performance, where if none of those iterations improve results, the run will exit with that as its sample solution. Thus, just by increasing max iterations to improve / restarts for RHC / SA or generations for GA / MMC, we can obtain arbitrarily optimal solutions. Because of this fact, I did not utilize restarts or max iterations to improve other than for testing purposes. With this caveat in mind, the analysis below will analyze performance with reasonable parameters for each to demonstrate the strengths and weaknesses of each algorithm. The default parameters I used for each problem were the following:

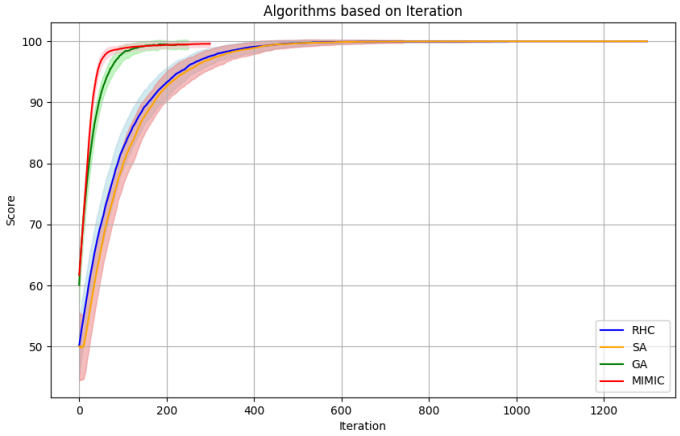
- Random Hill Climbing:
    - Random starts = 0
    - max iterations to improve before exiting = iterations (thus I let the algorithm run til the end of the trial)
- Simulated Annealing
    - Random starts = 0
    - max iterations to improve before exiting = iterations (thus I let the algorithm run til the end of the trial)
    - Initial temp = 100
    - Cooling rate = 0.99
- Genetic Algorithms
    - population size = 100
    - Mutation rate = 0.01
    - Tournament_size = 4
- MIMIC
    - population size = 70
    - Keep percent (aka sample size) = 0.2

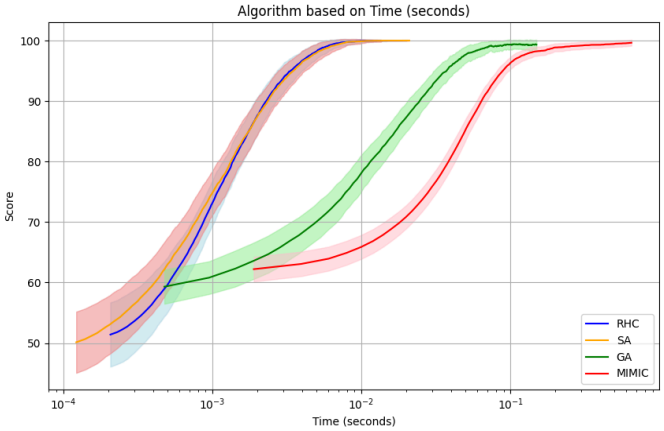
I then tuned these parameters to each problem to observe changes and try to increase performance. Generations and iterations were varied throughout to see the changed performance vs. training time

Count the Ones:

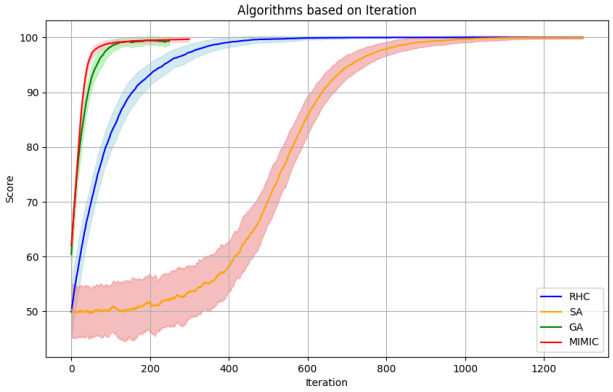
Here are the results from a 100-trial run of the Count the Ones algorithm for performance vs time and performance vs runs. The max score was 100 and I ran 100 simulations for each to obtain a more robust mean and standard deviation. I

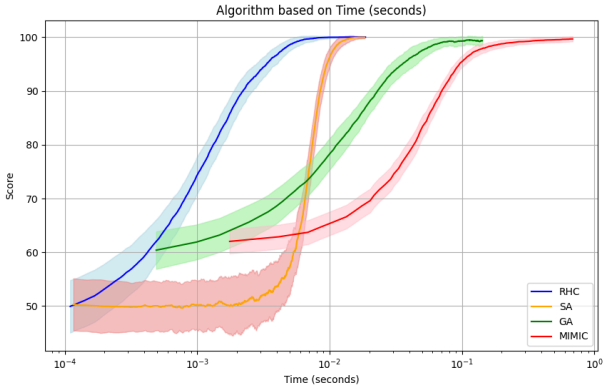
chose to stop each simulation at the number of iterations for each algorithm at the point it reached the maximum score within a small margin to create an intuitive comparison:





All of the algorithms reach the maximum score fairly quickly. In the end, the SA was the best performer. In terms of generations the ranking is MIMIC, GA, SA, and then RHC with all of them converging pretty quickly but that is not incorporating wall clock time. Originally SA was the slowest to converge in terms of iterations but then I lowered the temperature from 100 to 70 and eventually 0 and it caught up and even barely surpassed RHC's performance. Here are those results before lowering the temperature with the SA performing suboptimally:





This is understandable as when the temperature is high for SA it will try random potentially suboptimal solutions as part of its exploration. This has the effect of preventing it from converging in the short term. This can be seen in the large standard deviation to SA in the early steps. For this problem, the basin of attraction is actually the entire search space and there are no local optima that are not the global optimum. This means there is no reason for SA to ever accept a suboptimal solution for exploration. I set the temperature to 0 to test this and SA only switched solutions when its exploration yielded a superior solution and now it outperformed RHC as it metaphorically jumped to a higher point on the hill sometimes which is what we see in the original two charts.

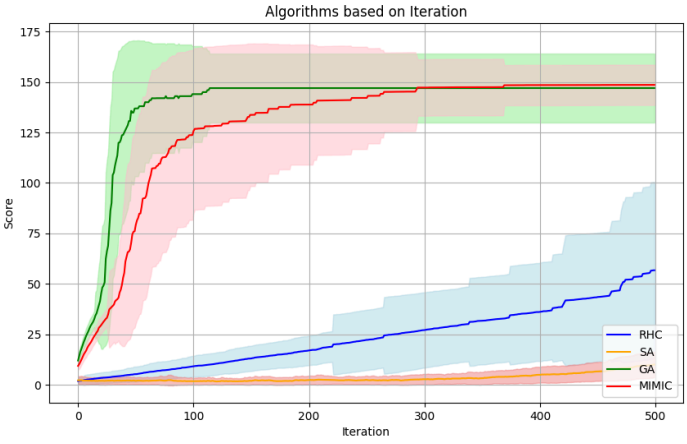
The real interesting analysis is in the algorithm vs time. The SA and RHC really shine here and show their utility for simple algorithms such as this which they are appropriately suited for. They converge to the max first of the algorithms. We also see the slow nature of MIMIC here. Considering the logarithmic scale of the x-axis, the SA converges orders of magnitude faster than MIMIC. The MIMIC algorithm struggled in the time comparison as its complex nature prevented it from quickly solving the problem. The count the ones algorithm is very simple which should naturally favor simpler solutions learning faster. Another key aspect of the problem is that there is no structure to the data necessary to understand how to improve

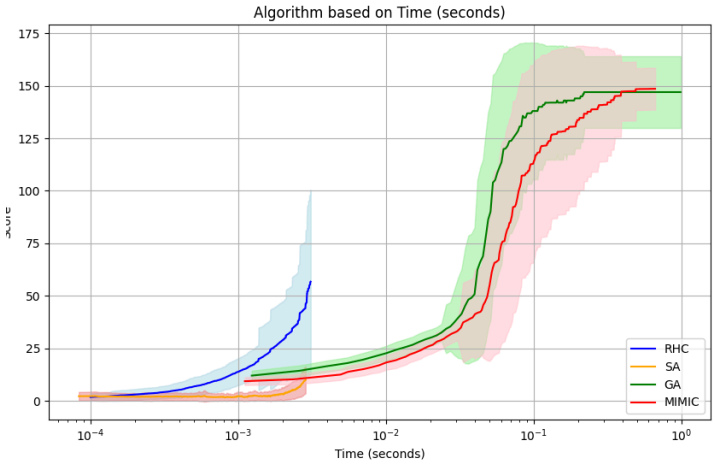
upon a solution. The solution is rewarded for switching any bit from a 0 to a 1. There is no complex relationship between the bits that need to be modeled which would favor a more complicated solution such as MIMIC. The optimal solution of 100 straight 1's is not hidden in some edge case among the sample space so there is little advantage to searching the space that would benefit the genetic algorithm. This explains the simplest algorithm, RHC, keeping pace with and in some ways outperforming the other algorithms. The structure of the data should help GA somewhat though as mixing half of one solution and half of another can lead to far superior outcomes as a solution such as 11111…00000 and 00000…11111 can be combined to quickly reach the optimal solution. This did happen, and the algorithm converged quickly in terms of generations but the slow nature of the GA's calculations caused it to converge much slower in terms of time than SA or RHC.

Four Peaks:

Here I repeated the process for the 4-Peaks problem. The results are below for a 100-trial run for performance vs time and performance vs iterations. The length was 50 and the threshold was 20, creating a max score of 150. Again, I ran 100 simulations for each to obtain a more robust mean and standard deviation. Later I will compare the problems over varying lengths.

Here the best performing algorithms were skewed towards the more complex algorithms such as MIMIC and GA as opposed to RHC and SA. The GA in this case is the best algorithm in terms of performance in convergence and speed to

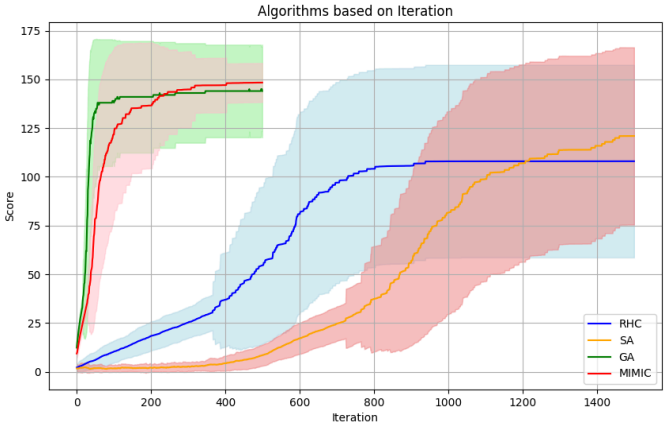


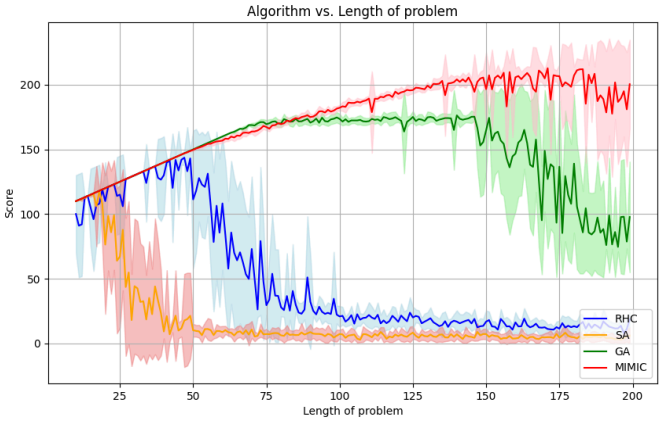


converge. It was the quickest to converge in terms of iterations, and while it was slower in wall clock time than SA and RHC, it vastly outperformed them in terms of accuracy. It was also faster than MIMIC in terms of clock time. One point of note is that MIMIC did in the end achieve a very slightly more robust maximum value but I think GAs speed to converge and wall clock time give it the edge. The GA is particularly suited to this optimization problem as there is a small basin of attraction around the bonus that will really cater towards an exploratory algorithm such as GA that can find the small areas in the solution space with particularly large payoffs. There is also a niche reason GA will outperform here because of the data structure. The GA can combine algorithms of all 1's and all 0's to create the solution that receives the bonus thus this problem is uniquely catered to an algorithm like GA that utilizes crossover. For this same reason, RHC and SA underperformed. The RHC and SA are susceptible to fall into the local maxima, which prevents them from finding the optimal solutions and we see this in their performance. While the SA does have some defenses for this, it will take more iterations in order for the algorithm to overcome these shortcomings.

I ran each for 500 iterations for comparison as that is about where GA and MIMIC are fully converged. We see both the RHC and SA starting to increase performance at this point. For clarity, I presented below the performance of these algorithms, if we let them continue to improve. As we can see, the performance for both does indeed improve but with the RHC leveling off around 110, and SA suprassing RHC at the 1200 iteration, but still below the 150 max of the GA and

MIMIC algorithms. We also see the standard deviation explode at around the 400 iteration for RHC and 800 for SA. This is about the time that both algorithms start finding the bonus on some iterations. For SA, I moved around the temperature and cooling and any decreased exploration really hampered performance in the long run as it needs those random jumps to potentially find the bonus. It still can't find it consistently though which leads to the higher standard deviation. Also noteworthy is the RHC flatlining completely at around 900 iterations. At this point, the RHC run is either completely stuck at a local maxima at 50 or has found the max at 150. The solution will stay static for the rest of the iteration which is why the line completely flattens. The 110 number is the mean, which tells us RHC is finding 150 slightly more than 50 and the standard deviation is static as well as each additional iteration has the same sample of scores as the last. The RHC and SA did perform their operations faster than the more complex algorithms as expected in all experiments but the performance was so much lower that even given more processing time, as done here, they still were unable to catch up to GA and MIMIC in terms of performance.





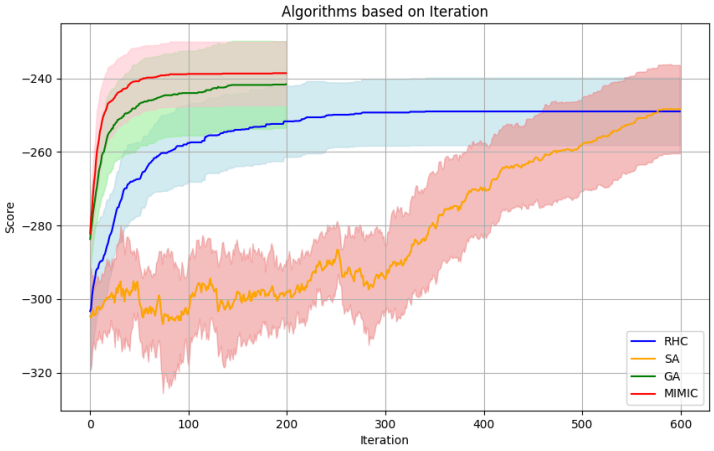
Finally, I looked at performance vs. length of problem. I ran lengths of 10 to 200 with the threshold 20% of the length on each side so 2 and 2 when the length is 10 and 40 and 40 when the length is 200. I limited the iterations for RHC and SA to 500 and the generations for MIMIC and GA to 200. The bonus was kept static at 100, which does mean the basin of attraction is wider as the length gets longer as the bonus becomes less valuable compared to the potential max.

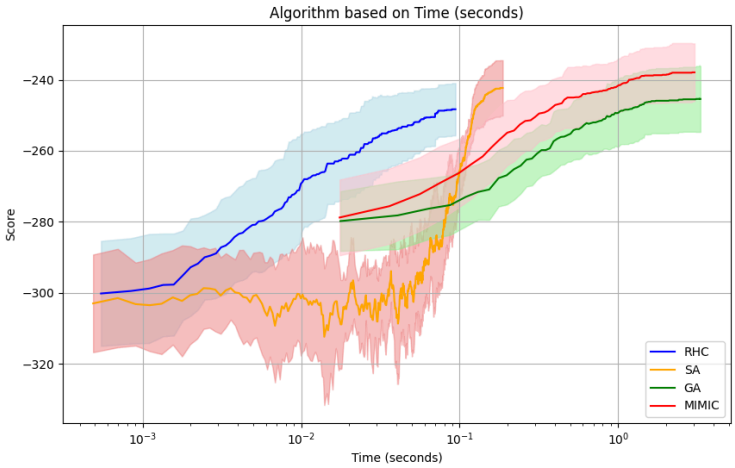
There is a lot to unpack here with these results. First of note, is that as the problem gets more and more complex, it more and more favors the MIMIC algorithm followed by GA. I explored different hyperparameters for both of these algorithms and found that increasing population size above 100 really just extends convergence time without improving performance, while lowering it below 20 started to have negative effects as the algorithm was not choosing from enough solutions to find improvement. MIMIC really excels at the longer lengths. Another observation, is the results across algorithms actually get worse as the length gets longer even though the max potential reward is higher. The GA and RHC really show this as they deteriorate quickly at the 120 and 65 lengths respectfully. This is because the threshold continues to expand and the bonus becomes harder to find. It also means that each iteration of RHC and SA has more neighbors as the length extends, and more neighbors that break the string of 1s or 0s and hurt performance. Thus the algorithm is less likely to find a neighbor that improves performance which is why the performance deteriorates as it gets stuck in the smaller local minima. The SA underperforming the RHC is interesting. I found that at the longer lengths, neither were finding the bonus anymore so any random jumps by the SA were strictly negative for performance so the RHC was able to outperform in those scenarios. It's also noteworthy that I did not allow the algorithms to run for very long compared to the longer length problems in order to keep the iterations consistent throughout. So it's natural the performance may decline as proportionate to the problem they are given less time to run.

We've now seen an iteration favoring a simpler algorithm such as the SA, and a problem slightly more complex with valuable edge cases favoring the GA. For the last optimization problem, we will look at a more complex problem: graph coloring.

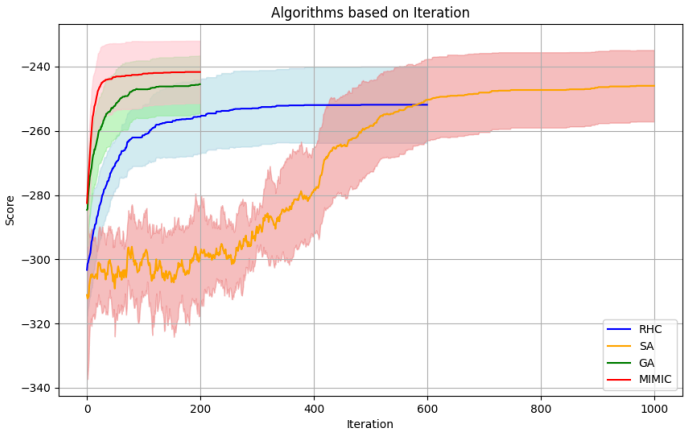
Graph Coloring:

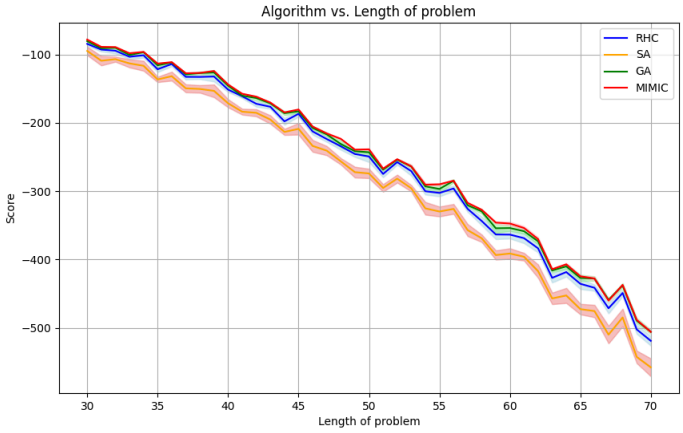
Finally, I ran the optimization algorithms on the graph coloring problem. I limited the problem to 2 colors represented as 1 and 0. I ran 100 simulations and averaged the results. For each simulation I created 50 vertices and made connections between them randomly with a 50% probability. This ensured a diverse set of graphs to compare the results. Every collision of two vertices with a connection that had the same color was a value of -1 and the total was the algorithm's score. I did this so the fitness function is still trying to maximize, which in this case is finding the least negative score. Here are the results:





We see the more complex algorithms outperforming the simpler ones with MIMIC performing the best followed by GA and then RHC and SA. The theoretical best solution is an NP-hard problem to solve and will also vary by graph structure so we can't easily say how close these algorithms are to maximum performance. Again we see the RHC and SA performing much quicker than the more complex algorithms by orders of magnitude. The MIMIC flatlined around -240 with the GA slightly below that around -245 and then RHC at ~250 and the SA finishing around that level as well. I limited the iterations for the RHC and SA to 600 to improve the visualization. I ran the SA further to show that it does flatline around that point. I also ran the algorithm for a variety of lengths and the results are below.





First, the SA, given more time, does improve slightly though with a lot of volatility and never really reaches the stronger algorithms though it did surpass RHC. Adjusting the cooling and temperature moved around the training speed but did not materially improve the final performance. For the problems of different length, the ordinal ranking of the algorithms stay

constant with what we saw in the original simulation. It's noteworthy that MIMIC doesn't really expand its lead over the other algorithms as the problem becomes more complex. I moved around the population size and sample size for MIMIC but the results were negligible. The lead does expand somewhat though. This is a little difficult to see in the graph as the scale is large for the chart as the best performance scores vary considerably as the length of problem expands. I was expecting greater out performance by MIMIC, though. One reason this may be is that I limited the run time for MIMIC and GA to 200 generations and RHC and SA to 600 iterations in an attempt to keep the data consistent with my original analysis, even though longer run time may be necessary on the larger problem sets. This also may explain SA's underperformance. However, the results indicate that it may just be the nature of the problem and the algorithms that these outcomes are robust regardless of run time. For example, randomly setting the graph colors and then adjusting a few bits is likely to get an okay solution though not optimal. Though this problem favors the more complex MIMIC algorithm, it has a high floor for solutions for the weaker algorithms. As opposed to the four peaks problem that can keep certain algos stuck much closer to 0.

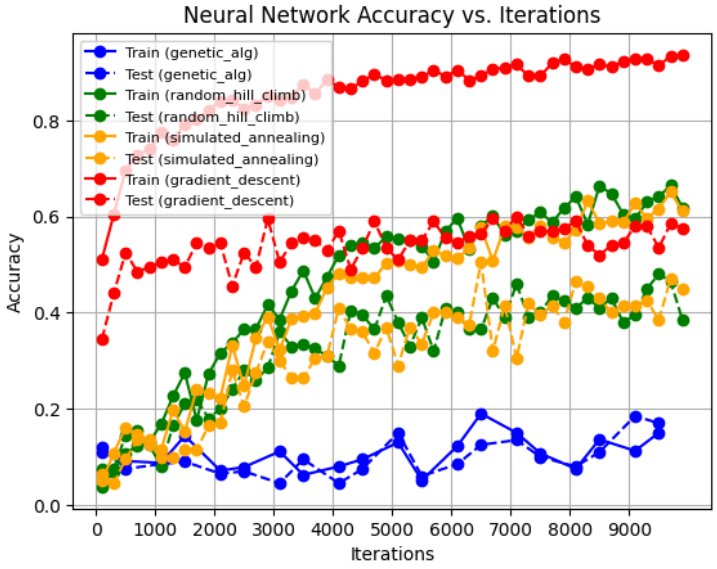
Neural Network Optimization:

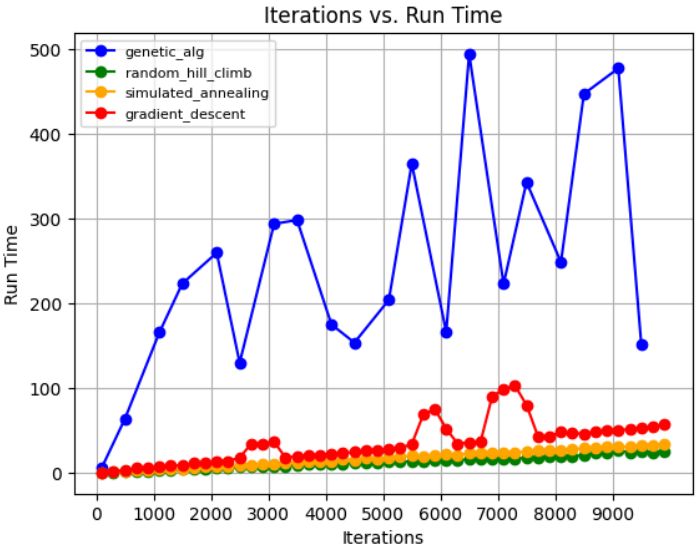
Finally I will use 3 of the random optimization algorithms above (RHC, SA, GA) and compare them to stochastic gradient descent (SGD) when training a neural network on one of my datasets from the previous assignment: the GTZAN music genre classification.

Before I begin the analysis, a developer's note: I changed a number of features of the original neural net for easier analysis. First I switched from pytorch to MLrose. MLrose provides much better integration with the random optimization algorithms as the optimizer for the neural network nodes. Second, the architecture I used for assignment 1 to train the neural network was a convolution NN with a number of layers. This was extremely slow to train when not using stochastic gradient descent and more or less did not converge in any reasonable amount of time. Instead I used a single hidden layer with 64 nodes. I experimented with numerous architectures and this yielded the best results. I also kept the "relu" activation function for all models which was used in assignment 1 for continuity purposes.

I varied the maximum iterations to observe performance through training time and then also kept track of training time in seconds. The parameters I used for training were max iterations = 2000, which was high with the max run having 10,000 iterations, but did allow for some early stopping. Step size was 0.5 for RHC and SA, 0.01 for GA and the learning rate was 0.0001 for SGD. After some experimentation, these yielded the best results. The population size for GA was 25 and the mutation rate was 0.05. Again, I preprocessed the GTZAN dataset to normalize all the values between 0 and 1. This is noteworthy for the optimization algorithms as their step size will be directly related to the scale of the input parameters. Another big distinction from the previous problems is that the neural network node weights are continuous as opposed to the discrete solutions we saw above. This can lead to situations where a large step size may step over a local minima and into another basin of attraction. Or the step size may be too small and the weights take a very long time converging to better solutions. For the GA, it also has the weird characteristic that the crossovers can only use parameter weights from the original population that it constantly mixes and matches. The mutation rate will be the only way for it to see new potential parameter weights outside the population. This is true for discrete solutions as well, it's just not nearly as noteworthy as it is more likely the population will be comprehensive to all potential solutions that could lead to an optima. This fact will almost certainly hinder the performance of GA. Here were the results which I will then analyze individually:

Name	Final Test Accuracy	Final Train Accuracy	Final Run Time
Adam	55.5%	94.8%	
SGD	57.5%	93.63%	57.00 s
RHC	38.5%	61.88%	25.08 s
SA	45.0%	61.13%	33.43 s
GA	17.0%	15.0%	152.24 s * (early stop, prev run = 483 s)





The first thing that jumps out is the SGD way out performing the other optimization functions. I included the Adam results from assignment 1 for comparison (though this is apples to oranges as that was a different NN structure). SGD even outperforms this though they were very close in assignment 1 during hyperparameter tuning so this is not so surprising. The outperformance also happened very early in the training as the other algorithms took time to start learning. This speaks to what we learned about the exploration aspect of the algorithms slowing them down from exploiting the information available to them, which SGD is more focused on. SGD did overfit by the end which was to be expected as that occurred in assignment 1 as well. I will discuss this more later. Finally the run time for SGD was slower than RHC and SA and much faster than GA. There were a couple of bumps in train time which seem to be more a product of my CPU potentially being slower over that period working on other tasks than anything related to ML. Otherwise the train time is linear with iteration size.

The RHC and SA performed very similarly throughout the experiment. There is noise in the results but overall they were far better than a weak learner of 10% accuracy. At the end the SA was slightly higher than the RHC but that was probably just the randomness of the final run as they tracked each other very closely throughout the training. They continued to improve with an increase in training size and were still improving at the end of the experiment. Overfitting did start to play a role as the train accuracy bifurcated from the test accuracy but the train accuracy was still far away from any upper bounds so more train time should improve performance for both algorithms. While the overfitting is something to be aware of, the models are considerably underfit to the data with train accuracies below 70, so it is not too much of a concern until performance improves. Also, the neural network architecture only has one hidden layer and is considerably simpler than the model from assignment 1, which should work against the models overfitting as we learned in the lectures. Both algorithms ran very quickly in a relative sense with RHC consistently quicker than SA. This leads me to believe the SA was not really benefiting at all from its exploration from the annealing process. The performance was more or less identical to the RHC and yet the extra exploration was most likely the cause of the slower run time. This is likely due to the nature of the GTZAN data.

With only 800 training examples but 58 features and 10 classes, there was not a lot of data to learn off of which created a noisy environment. Also the nature of the data is noisy in and of itself as what genre a piece of music belongs to is subjective with considerable overlap. Compare this to the MNIST dataset where the numbers are much more likely to belong to a well defined class. This is probably what contributed to the overfitting seen by SGD and throughout assignment 1 as the algorithms learn the noise instead of the ground truth and the noise is a larger percentage of the data presented. For each class there were only 80 train examples and 20 test examples. This is also what probably made the exploration not helpful for simulated annealing as there is no reason to believe the maxima has a small basin of attraction.

It was more likely to send the algorithm to some local minima and waste training time before it moved back closer to a more optimal solution.This would also explain the RHC outperformance in the early runs with less iterations, as it was less distracted by fruitless exploration. This noisy landscape also plagued the GA.

The GA is the worst performer by a wide margin. I did numerous hyperparameter tuning around NN structure, population size and mutation rate but was barely able to improve performance. I found that increasing these parameters just slowed down training time immensely to the point where the algorithm didn't even really outperform a random guess. By the end of the exercise, I was able to get the GA to perform as a weak learner as its performance hovered in the 15% range, outperforming a 10% random guess, so it did indeed learn something at least. The noisy dataset just did not lend itself to genetic combinations of parameters. Again, the basis of attraction of the optimal solution was not small and random, working against the genetic algorithm. Also, the dataset having 10 classes made the algorithm that much more difficult to converge on anything useful. The performance also exhibited a lot of volatility from run to run as expected as the algorithm struggled to improve. Finally, the issue around discrete vs. continuous solutions certainly was working against the GA through all of this. Having a good random starting population may have been the biggest catalyst for performance. It was also the slowest algorithm by a lot which we saw throughout this assignment. We can even see in the graph where the early stopping was triggered with a spike down in time of run when we experienced a full 2000 iterations of no improvement. The GA optimization technique was clearly not well equipped for this dataset.

I also duplicated this experiment with the MNIST dataset to compare the results and the training and test scores were considerably worse. With so many more input parameters with the MNIST dataset, the training was destined to take longer as the RHC and SA have much more potential local maxima to explore and get caught in. The GA did particularly poorly. I believe this is due to the fact that the entire pixel data defines what number is drawn. So taking portions of solutions of one solution set and matching them with half of another isn't particularly likely to produce improved results except by chance. This is yet another form of the discrete vs. continuous problem I discussed above. Even with the training and test accuracy being considerably higher in assignment 1 for the MNIST dataset, the training results were worse than the GTZAN when the random optimization techniques we are exploring were used.

Conclusion:

The biggest takeaway from this assignment is that different algorithms are better at different tasks. We saw the RHC perform very well on a simple task where the basin of attraction was the entire solution space. This was actually the best algorithm on “count the ones” before we lowered the temperature for SA. Where there are some local minima traps, SA outperforms RHC as some exploration is necessary to avoid those pitfalls. GA was the best performing algorithm when the problem shifted to a situation where the optimal solutions were more difficult to find with smaller basins of attraction such as four peaks. And finally as the problem complexity increased with our graph coloring problem, the MIMIC algorithm reigned supreme.

When we switched to analyzing these algorithms' performance in training a neural network, however, they all performed considerably worse than stochastic gradient descent (and Adam for that matter). One realization I had during this analysis was how revelatory the implementation of SGD, backpropagation, and the compute power to run them, must have been for the industry when they were first created. The optimization algorithms we commonly use for training neural network nodes, just completely blew our random optimization techniques out of the water. It took 1000 iterations (and even more for GA) for our optimization algorithms to produce a neural network that could even qualify as a weak learner with an accuracy above 10%. Meanwhile after 100 iterations, SGD already had a train accuracy over 50% and a test accuracy close to 40%. As we explored above how these algorithms perform better in different scenarios, we see that in stark detail here. SGD is far better equipped to handle this type of optimization problem than SA, GA, or RHC.

Part of this can be attributed to the GTZAN dataset. First, having 10 classes and only 800 training examples will make it difficult to learn for an algorithm, particularly when the dataset has as much noise as the GTZAN has. We saw this with SGD as well as it suffered from some severe overfitting by the end, with a test accuracy 40% below the train accuracy. I would bet that if I was using the Iris dataset or wine classification, or some other linearly separable dataset or close to it,

RHC and SA would be able to achieve solid performance just at a much slower training speed than SGD. For GA, I'm not so sure as the nature of crossover really is hindered by the continuous nature of the neural network node weights, as mentioned above, though I would still predict improved performance vs. what was achieved on the noisy GTZAN dataset. One other factor of note about the GTZAN dataset is that there were only 200 test examples. That left only 20 samples per class. Any supervised learning algorithm could struggle in this regard. There just isn't enough data to really know if it's representative of the actual data population. A supervised learning model could also run into the issue of having a test dataset, just through stochastic selection, that is unfortunately significantly different from the training population. There really isn't much the model can do to improve performance in this scenario. It is simply not seeing training data that can prepare it for the labels it will see in the test set. The solution to this is of course, more data.

Many of the results of these experiments mimicked the lessons from our lectures:

- RHC getting stuck in the local maxima of the 4-peaks problem
- Random starts improving performance of RHC (only used for testing as this could be used to skew results for a given problem).
- Annealing of SA to randomly explore the environment, slowing down convergence in the short term but improving performance over RHC in the long term.
- And that exploring may not improve performance for SA if the solution does not necessitate random exploration. We saw this in count the ones where the exploration was slowing down performance until I lowered the temperature so the algorithm only explored to points higher than its current solution.
- GA outperforming in problems where the solution has a small basin of attraction in need of exploration
- GA outperforming in problems where the solution can be found combinatorially via crossover
- GA does poorly when the input space doesn't lend itself to the locality of bits being essential or some linear combination of inputs producing improved results (like the weights of the neural network nodes).
- When the problem requires some structure and not just points in the space, MIMIC performs well
- MIMIC runs considerably slower per iteration than simpler optimization techniques. Yet this is still worth it when the cost of computing the fitness function is very high (more complex problems)

In conclusion, the most useful lesson from this assignment is that there is no one size fits all when it comes to machine learning algorithms. We saw these optimization techniques all have their strengths and weaknesses in terms of accuracy, run time, and ability to navigate a diverse set of hypothesis spaces. Simply looking at the accuracy of a runtime will not tell the whole story for these algorithms. There are a myriad of characteristics to consider such as how long they take to run, how consistent they are, how quickly they converge, how likely they are to achieve a certain maximum, how susceptible they are to get stuck in a poor performing solution, how their performance changes as the supervised learning task changes such as number of classes, number of training examples, or noisiness of the data. For example, our best performer on the neural network training, stochastic gradient descent, requires that the function being optimized must be differentiable, otherwise we must use a different algorithm. These are just a handful of the potential questions we can be asking when analyzing an algorithm's performance. This is a crucially important insight to internalize as we continue our education on supervised learning algorithms and how they perform in the real world.

References:

- README file including for instructions for running the algorithms included
- https://bambielli.com/posts/2018-07-22-comparison-of-four-randomized-optimization-methods/
- https://louis-dr.github.io/machinel2.html
- https://www.kaggle.com/code/aftereffect/musicgenreclassificationfinal/notebook
- https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification/code
- https://mlrose.readthedocs.io/en/stable/source/neural.html
- All libraries used are listed in the import statements in the code. Main libraries: sklearn, keras, tensorflow, pandas, numpy, matplotlib, xgboost