Analysis of Randomized Optimization Methods with Applications to the Retail Industry

CS7641 Fall 2023

Tyler Burki (tburki3@gatech.edu)

***Abstract*—Three problems that permeate the retail industry--maximization of margin for shipping, efficient logistics and product display cohesion--were examined through the lens of randomized optimization algorithms. For each problem, a comparison of the performance of different algorithms leads to a suggested method for solving the problem. This culminates in an analysis of replacing backpropagation in a neural network with the optimization algorithms.**

1 INTRODUCTION

In a retail organization, such as the one I work for, the variety of operating costs such as shipping, labor and purchasing inventory weigh strongly on a business's ability to deliver consistent profits. Below, I examine three problems related to these costs in an attempt to maximize the benefit provided to the company. Four algorithms will be explored for each problem: **random hill climbing (RHC)**, **simulated annealing (SA)**, **genetic algorithms (GA)** and **MIMIC**. Results for each algorithm will be generated by the **mlrose-hiive** ("Overview — Mlrose 1.3.0 Documentation," n.d.) library for python. Optimal results will be compared between algorithms for the given problem domain and analysis performed to determine which algorithm would produce the best results on real-world data. All results were generated by averaging runs over three random seeds (8675309, 7331, 10102020) to reduce the impact of receiving poor results from a single bad seed.

In addition, **random hill climbing**, **simulated annealing** and **genetic algorithms** will be used to generate weights for a neural network tasked with the problem of breast cancer diagnosis from my previous assignment. Analysis will compare the three algorithms as a function of their success and failures in improving upon the backpropagation of the neural net from that assignment.

2 MERCHANDISING COHESIVENESS

A primary focus in the retail industry is a process called merchandising. Broadly, merchandising refers to both the assortment of products available and how those products are presented to customers on store shelves and online. The prevailing notion in the industry is to group similar products together to both ease the process of locating a particular product and provide an array of options for a product class in the same space.

2.1 Motivation

For the above reasons, merchandising similar products in the same space is generally considered to be more beneficial than the alternative within retail organizations. B in eing able to measure and maximize the product cohesiveness of displays could provide information on how well the display will be received by customers and translate into sales dollars.

2.2 Method

A simple, conceptual model of this problem was tested using the **MaxKColor** ("Fitness Functions — Mlrose 1.3.0 Documentation," n.d.) fitness function. The problem asks the algorithms to maximize the score of a graph where connected nodes with the same value receive one point and zero otherwise. In our scenario, the goal is to maximize the score of three intra-connected regions of a display, essentially defining the boundaries of the areas, to achieve maximum fitness. The display consists of two square regions that can hold four products in a two by two shape and a third square region that can house 16 products in a four by four shape. Each product (node) in a region is connected to all other products in its region that it touches vertically, horizontally and diagonally. The problem is therefore maximized by making all values in a region the same (cohesiveness) and as a result defining the boundaries of the regions. A diagram of the regions can be seen in *Figure 1*. The problem size is described by the maximum value each node can take on (e.g. a size of "2" means a node can be 0, 1 or 2). Problem sizes of 1, 5, 10 and 15 are investigated. In-depth analysis is performed for a problem size of 10. This process would need to be extended in a manner such that the distribution of potential products for a display was factored into

decision making, but the complexity and availability of tooling to solve this problem is outside the scope of this analysis. Parameters for the algorithms are available in *Table 1*.

Algorithm	max_attempts	restarts	temperature	population_size	mutation_rate	keep_percent_list
Random Hill Climbing	1000	25				
Simulated Annealing	1000		1			
Genetic	10			25	.4	
MIMIC	1000			50		.6

Table 1

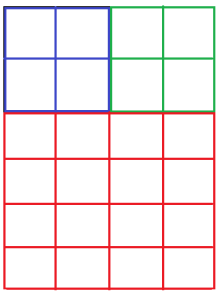

Figure 1

2.3 Results

In terms of fitness, GA is unparalleled, scoring 25% better than the next-best algorithm. GA also performs consistently optimally for all problem sizes, while the other algorithms see a strong decline in fitness as the problem size increases. GA performs worse than the hill climbing algorithms in function evaluations (FEvals) and Time, but significantly better than MIMIC. RHC and SA perform similarly in all respects, with SA having a much higher fitness variance in all cases. MIMIC reaches its maximum value more quickly than GA, but fails to converge quickly. *Figure 2* displays graphs of the results.

2.4 Analysis

RHC and SA see moderate to good success for small problem sizes. This is because when the maximum value is low, fewer iterations are necessary to discover that the same values in a region produce a better score. For example, in the two by two regions, all nodes connect to all other nodes in the region, so placing any two of the same value in the same region will unilaterally improve fitness. As the problem size increases, the odds of this situation deteriorate due to the increased hypothesis space. RHC performed better than SA for high temperatures, so the SA parameter was cooled to 1 to focus on hill climbing while still providing an opportunity to climb through small troughs. Certain runs of SA performed better than RHC as a result, but with a similar mean and a much higher variance for SA there is no reason to select one method over the other.

MIMIC quickly reaches a local maxima in the first few iterations. The performance decrease necessary to improve fitness, by increasing population size generated, did not end up being worth it. MIMIC failed to reach the performance of the other algorithms in almost every scenario, so a sub-optimal fitness was selected in the hope that quick convergence could mitigate some of the downside. However, the algorithm fails to converge before its peers, in part due to low population size and a middling keep percent. As a result, the algorithm seems to believe that a better fit exists based on the distributions it estimates, but fails to find those solutions. Increasing the population size and keep percent would likely produce a better result at the expense of FEvals and Time, although it would still be dwarfed by the performance of GA.

GA is the clear-cut winner having unrivaled performance in fitness for all problem sizes while still using fewer FEvals than MIMIC in Time that is only marginally slower than the hill climbing algorithms. The secret to GA's success lies in its breeding method. Due to the layout of the nodes, two states that are homogenous for an entire region on either end of the input in a two-value input space (e.g.

[1,1,1,1,0…0] and [0,0,0,0,0,0,0,0,1…1]) will produce an offspring that is necessarily maximally fit for two of the problem's three regions when splicing on a region boundary. This allows GA to reach local maxima quickly and hop to a higher maxima within only a few iterations. From this point the problem becomes one of minimizing FEvals and Time, which is approached here by selecting a low population size and mutation rate. Nearly all combinations of these parameters produced the same best result, but the particular values used were selected to be a bit higher than necessary in the hope that they would continue to generalize well as the problem size increases. This appears to have worked as GA does not degrade with regard to fitness as the problem size increases.

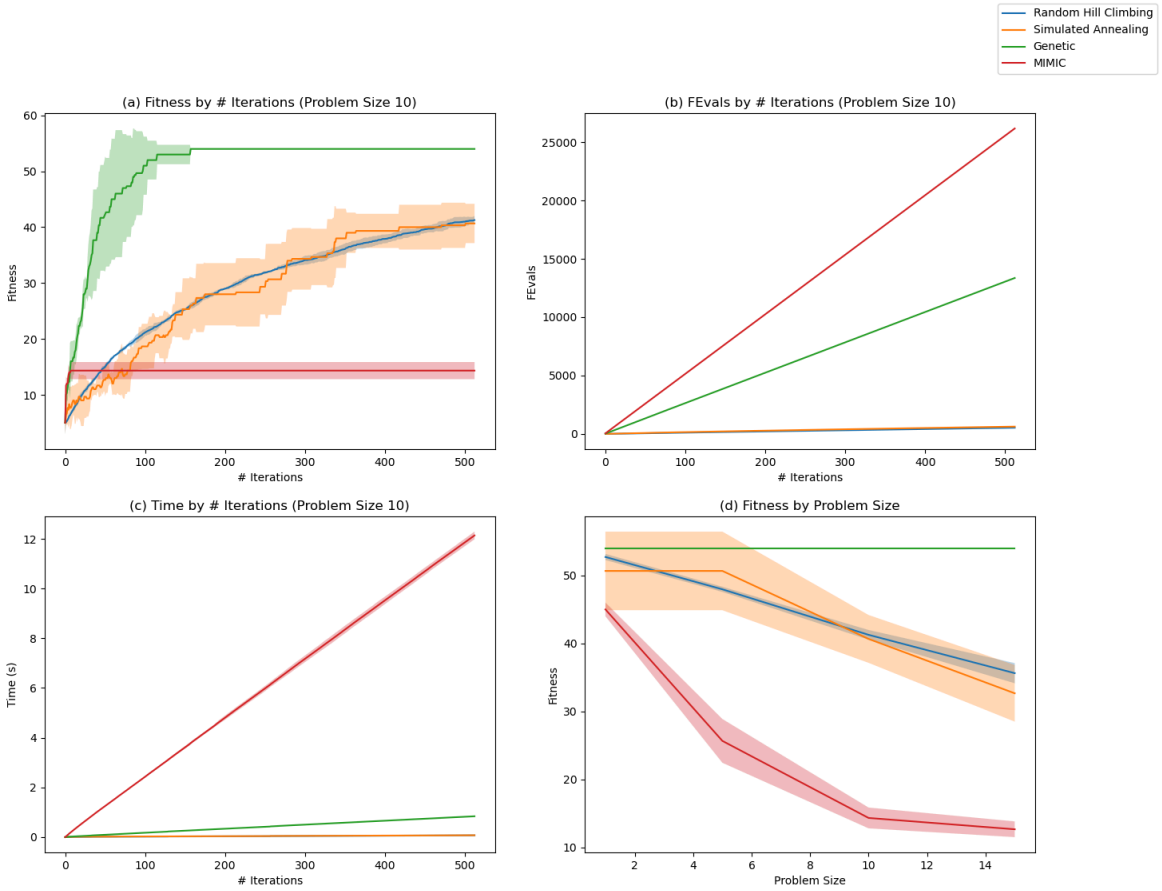

Figure 2

2.5 Conclusion

With its consistent discovery of global maxima, marginal performance losses with increased iterations, and ability to generalize to larger problem sizes, GA is clearly the best algorithm for this problem.

3 EFFICIENT LOGISTICS

For long-term storage and flexibility to respond to customer demand, retail businesses consolidate inventory at distribution centers before sending that inventory (via truck, in this case) to the stores that need stock. This method also simplifies the purchasing process and prevents retailers from needing to know where inventory will ultimately end up (i.e. a store) at the time the order is placed.

3.1 Motivation

Shipping goods between warehouses and stores is a significant cost in the retail industry. It can cost upwards of $250,000 per year to keep a single truck on the road ("An Analysis of the Operational Costs of Trucking: 2023 Update," 2023). Fewer trucks would be needed if their routes could be optimized to deliver more product to more stores in less time.

3.2 Method

This type of optimization is defined by the **TravellingSales** fitness function ("Fitness Functions — Mlrose 1.3.0 Documentation," n.d.). The fitness function attempts to minimize the distance "traveled" between a series of nodes with the requirement that each node is visited. Since we are trying to maximize a fitness score, it is sufficient to maximize the inverse of the function's score. The coordinates of my employer's stores were used as inputs to the function. The coordinates were normalized on the location of the main warehouse, which resides at (1,1) in the Cartesian plane. Problem sizes ranging from traveling between 5, 15, 25 and 35 uniformly selected stores were examined. Parameters for the algorithms can be found in *Table 2*. *Figure 3* shows a plot of the relative locations of the stores normalized on the location of the warehouse.

Algorithm	max_attempts	restarts	temperature	population_size	mutation_rate	keep_percent_list
Random Hill Climbing	1000	25				
Simulated Annealing	1000		1			
Genetic	10			25	.6	
MIMIC	1000			25		.6

Table 2

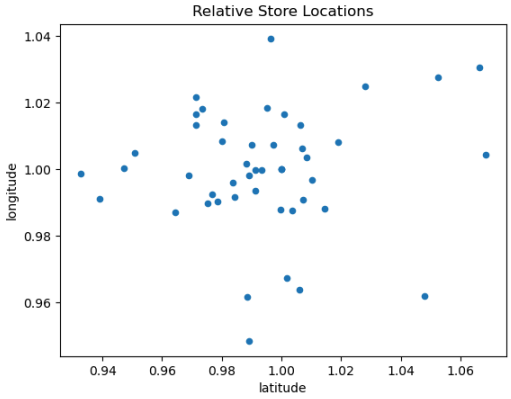

Figure 3

3.3 Results

All algorithms find a fitness within about 6% of the highest scoring algorithm. RHC, SA and GA all perform similarly with respect to FEvals and Time, with GA being marginally slower. MIMIC is the outlier in this regard, with its comparatively steep, linear FEval and Time curves. MIMIC also performs the worst on fitness for all problem sizes. All algorithms see a steady increase in fitness as the problem size increases up to 25, which should be the case as the amount of distance to travel (and in this problem, not travel) increases with the problem size. However, fitness for all algorithms declines after this point, albeit at a slower rate than the previous increase. *Figure 4* displays graphs of the results.

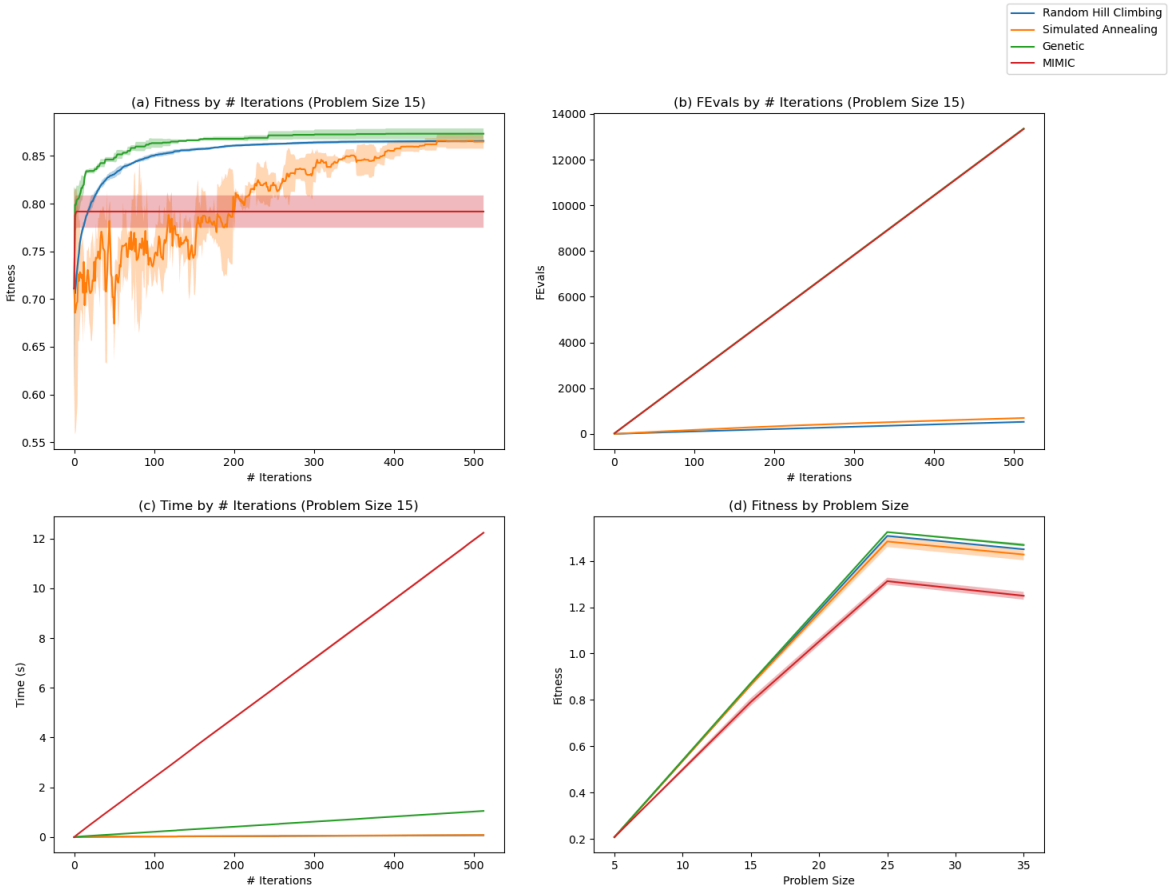

Figure 4

3.4 Analysis

Upon first glance, it appears that this is either a problem that is well-suited for randomized optimization or is so difficult that all algorithms perform equally poorly on it. RHC reaching a local maxima nearly as quickly as GA and MIMIC (< 50 iterations) tells us that there are many "pretty good" solutions to this problem. This is because RHC can proceed in any direction and still come out with a high fitness score. To add credence to this claim, RHC has the smallest variance of any of the algorithms, implying that for every run it was able to arrive in roughly the same place.

MIMIC reaches a local maxima within the first few iterations but does not converge and stop running before its peers. The benefit of MIMIC in this scenario would be to reach one of the best "pretty good" solutions quickly and stop. This would hopefully reduce the amount of FEvals and Time needed to produce an answer relative to the others. To improve performance in this way, we could increase the population size and keep percent while lowering the max attempts the algorithm uses to find a better solution. It would be highly probable that MIMIC would still reach a near-optimal maxima but believe it has correctly described the underlying distribution and converge more quickly.

GA has success across fitness, FEvals and Time that can be explained in how it is applied to the problem of wayfinding. RHC takes one road at a time, seeing if it is a shorter path to the goal. SA may backtrack a bit to try out different roads, but the sentiment is the same. GA, on the other hand, determines lots of good routes between different stops on the path and then tries to piece them together and see

if they make sense. In a way, GA is solving many simple problems while RHC and SA attempt to solve one big problem. This ability allows GA to find a more optimal route more quickly.

While SA performs comparatively poorly for the first few hundred iterations, in *Figure 4 (a)* it reaches the fitness of GA and RHC by the 500th iteration. With fewer FEvals and less Time spent per iteration over RHC and GA, it is possible that extending the running period further for SA could produce a higher fitness than either. Lowering the temperature from 1 to better mirror RHC in the given scenarios could potentially achieve this end, but it would be worth investigating further if the current temperature or higher could increase fitness even further after more iterations with marginal FEval and Time performance loss. This is because SA has the ability to backtrack into a worse state in search of a higher maxima. In scenarios where a seemingly terrible route from a distance actually ends up in the optimal solution, SA would be the most likely to find that route.

3.5 Conclusion

Although it performs comparatively poorly at low iterations, SA's low FEval and time cost mean that more iterations can be run. Those additional iterations have shown to rival the performance of the other algorithms at higher iterations, with the chance that SA can find a more optimal path that the others dismiss. SA should be used to solve this problem.

4 MAXIMIZATION OF MARGIN FOR SHIPPING

Often, when a customer orders products from an online business they choose to have those items delivered directly to their home. The majority of businesses must then pay a third party to deliver those goods on their behalf. The margin of a product is the difference between its cost and the price the customer pays for it, and therefore how a retail business earns profits. The cost of paying a third party to deliver packages deteriorates the income generated from these transactions. For simplicity, concerns over the volume/size of the package have been ignored in the following.

4.1 Motivation

Because shipping can be costly, it would be best to maximize the amount of income generated from each shipment to ensure profitability. Many carriers charge increased fees as the weight of a package increases. FedEx, for example, increases delivery price from $71.79 for a 49-lb package sent within 150 miles to $72.12 (.5%) when the package is 50 lbs. Interestingly, when the package reaches 51 lbs the cost jumps sharply to $78.69 (9.1%) ("FedEx® Standard List Rates," n.d.). Therefore, maximizing the margin of the products in a 50-lb package could take advantage of the biased price-to-weight distribution. One way to implement this in a retail setting would be to offer pre-determined assortments of goods whose contents are determined by solving this maximization problem.

4.2 Method

The problem is readily described by the **Knapsack** fitness function ("Fitness Functions — Mlrose 1.3.0 Documentation," n.d.) with the intention to maximize the value of a "sack" of objects relative to their weight such that the total weight of the sack does not exceed some threshold. In this formulation of the problem, a product may only be included in the sack once. One hundred twenty-two products from the same product family, with their margins and weights, were obtained from my place of work for the analysis. Weights were increased by a factor of 10 and then rounded in order to represent fractional weights as integers. Similarly, margins were increased by a factor of 100 and rounded. This procedure makes the domain discrete while preserving the relationships between products. Each optimization algorithm was tasked with optimizing the margin of a 50-lb box with input spaces that varied from 5 to 35 products selected uniformly without replacement from the data set. All products have a weight less than 25 lbs. *Table 3* shows the parameters used for each algorithm on the problem while *Figure 5* presents the data with margin as a function of weight.

Algorithm	max_attempts	restarts	temperature	population_size	mutation_rate	keep_percent_list
Random Hill Climbing	1000	25				
Simulated Annealing	1000		5000			
Genetic	10			100	.8	
MIMIC	100			100		.2

Table 3

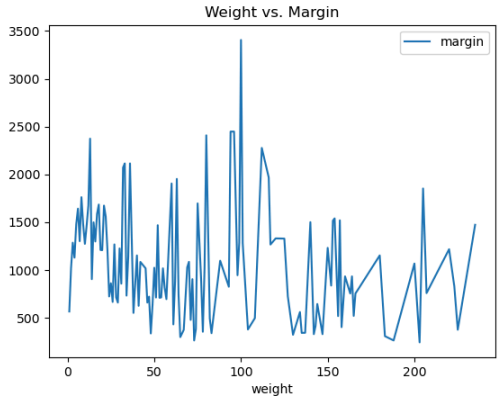

Figure 5

4.3 Results

SA, GA and MIMIC all perform similarly well for problem sizes up to 15. As the problem size grows beyond that threshold, SA performance degrades rapidly while GA and MIMIC see a reduction in performance increases. For the 25-input case, MIMIC converges rapidly and ceases completely around 100 iterations. Variance in fitness is small for GA and RHC while MIMIC varies between 30-40% and SA varies wildly at over 100% of the mean. FEvals with respect to iterations are the same for GA and MIMIC up until the point MIMIC terminates, while RHC and SA have comparatively minimal FEvals. Time by iteration follows suit where MIMIC has a marginally higher time than GA up to the point of convergence while RHC and SA barely register on the chart. *Figure 6* displays graphs of the results.

4.4 Analysis

RHC performs so poorly due to the large hypothesis space presented in the problem. For a problem size of 25, there are 2^{25} or 33,554,432 ways to represent the state. Randomly selecting a state that is viable takes a large amount of luck. However, when a viable state is found, RHC performs similarly, although still not as well, as the other algorithms. The low fitness and variance of RHC is due in large part to not being able to find a good initial state. It follows that we could improve the performance by initializing the algorithm with a naive viable state. FEval and Time scores for RHC can be misleading because while the per-iteration FEval and Time values are low, the random restarts baked into the algorithm cause it to perform many more iterations than the others. More restarts could potentially improve performance with regard to fitness, and the best fitness found is still likely a good solution even though the total Time eclipses its competitors.

SA suffers from the same issues as RHC with regard to the large hypothesis space. Unlike RHC, SA has the ability to traverse through less-optimal values to find higher scores. This accounts for the higher mean fitness in *Figure 2 (a)*. The higher variance than RHC is the result of having runs that were not able to find any viable solution, while the runs that succeeded were able to do a good job of maximizing fitness. The selection of temperature plays a major role in the success of the algorithm, with the temperature used here set to 5000. The temperature needs to be in-line with the expected differences between states, which for this problem is likely to be > 1,000. If the temperature is too low, the probability of selecting a worse state to traverse through is low as well. Since the hypothesis space is so large, it behooves us to select a high temperature to explore as many states as possible before cooling (geometrically, in this case) into hill climbing. Like RHC, SA would benefit from being initialized with a naive, viable state. With no initial state, performance could be improved by setting the temperature to positive infinity before decaying at the desired rate once a viable state has been found. When the number of restarts for RHC are taken into account, SA has the best performance with regard to FEvals and Time because it is simply checking neighbors and potentially jumping to worse points with a probability dependent on temperature.

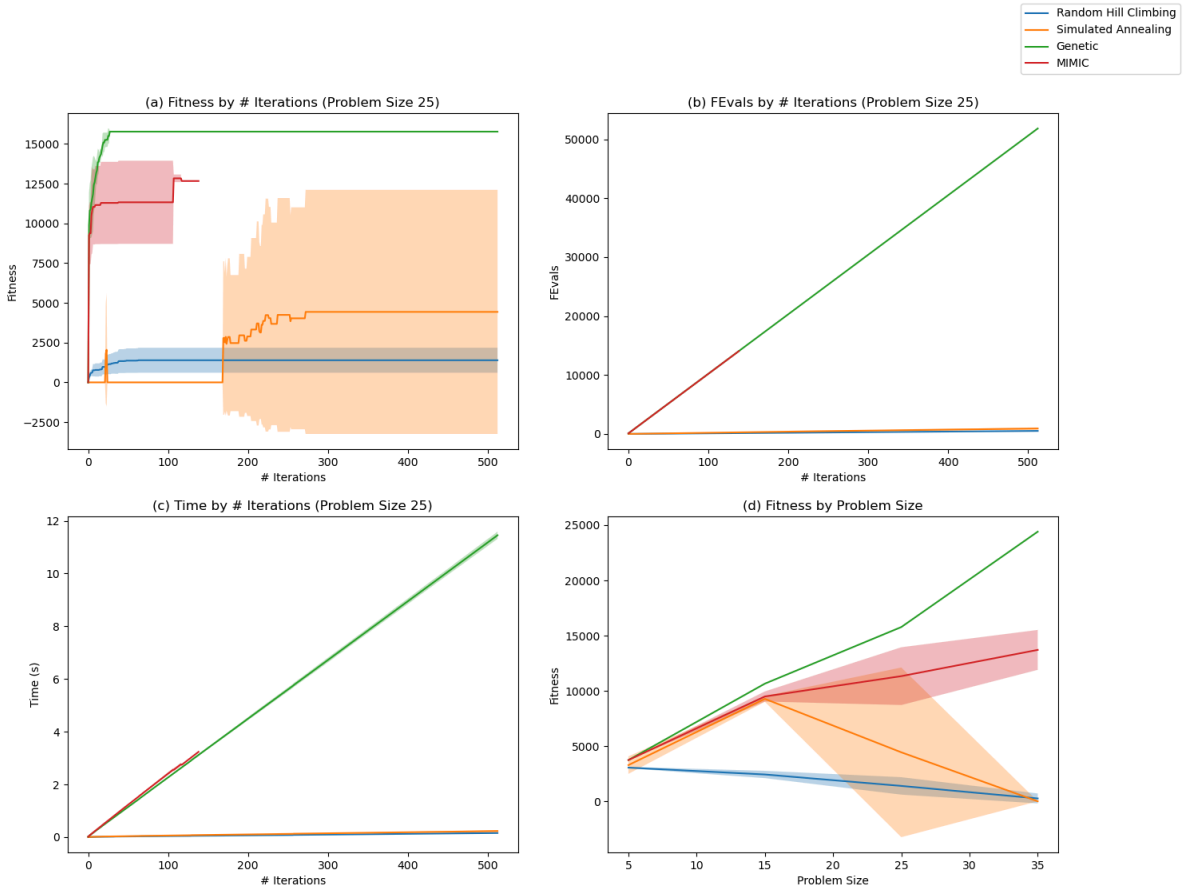

Figure 6

GA consistently has the best fitness over specific examples and all problem sizes. This is due in large part to the structure of the problem that rewards searching widely and how closely viable states are related to one another. For example, for a problem size of 5 products, it is quite possible that all 5 products could be included and still be under the target weight limit. GA could take states [1,1,0,0,0] and [0,0,1,1,1], breed them, and (potentially) end up with the maximal solution of [1,1,1,1,1]. Because GA tries so many states, and takes the best attributes from each to create new states, it finds a very good fit extremely quickly. However, once it has achieved a state that is near optimal, it struggles to converge because the attributes of the state that give it a good score may not, in fact, be optimal (i.e. there is a lighter product with a higher margin) but the algorithm continues to breed with the attributes it knows are good and ends up receiving results that are not viable or slightly worse. This long convergence leads to GA performing the worst in FEvals and Time on a per-iteration basis because MIMIC converges much more quickly.

Possessing many of the same traits as GA, MIMIC finds a high fitness quickly by combining the best attributes of valid states. Unlike GA, MIMIC estimates the underlying distribution of the data in an attempt to give the problem structure and make predictions on the likelihood of the existence of a better solution. It is for this reason that it is able to converge in about 100 iterations. The algorithm finds a high fitness and then deems it unlikely that a better solution exists and settles for a sub-optimal, but still highly successful state. For this reason it outperforms GA in FEvals and Time while still finding a solution that is, with lower variance, consistently more fit than SA.

4.5 Conclusion

While GA performs better with respect to fitness, the tradeoff with FEvals and Time gives MIMIC the upper-hand thanks to its quick convergence to a highly fit solution.

5 USE OF RANDOMIZED OPTIMIZATION IN NEURAL NETWORKS

Neural networks are learning algorithms that attempt to determine coefficients (weights) for a function representative of the true concept that a model is trying to emulate. These weights are improved incrementally after each iteration through the process of backpropagation using **gradient descent** (GD). If the algorithm overshoots its target in a particular aspect, it attempts to reduce the weights (and vice-versa) on those pieces of the network to produce a more desirable result. In this section, we examine the effect of replacing this process with randomized optimization algorithms and analyze the results.

5.1 Motivation

If randomized optimization algorithms could replace gradient descent and backpropagation for certain cases, those cases could see improved performance in terms of fitness and/or fit/score times. Higher fitness increases the accuracy of the model, and lower fit/score times lower compute costs as well as increase the amount of data that can be used to train or query the model.

5.2 Method

The breast cancer dataset from scikit-learn ("Sklearn.datasets.load_breast_cancer — Scikit-Learn 0.24.1 Documentation," n.d.) was used to evaluate the algorithms (RHC, SA and GA) as a replacement for backpropagation in a neural network. The dataset is a binary classification of 569 samples, 212 (37.3%) of which are classified as malignant while 357 (62.7%) are classified as benign. The **NeuralNetwork** class ("Machine Learning Weight Optimization — Mlrose 1.3.0 Documentation," n.d.) from **mlrose-hiive** was used to train and test the network. Default parameters were used except in the cases of activation ('tanh'), clip_max (5), max_iters (250), max_attempts (100), hidden_nodes ([250]) and restarts(100, RHC only). The data was standardized using scikit-learn's **MinMaxScaler** class ("Sklearn.preprocessing.MinMaxScaler — Scikit-Learn 0.24.1 Documentation," n.d.). Eighty percent of the data was reserved for training. The three random seeds used previously in this paper were reused and their results averaged.

5.3 Results

GD presents a smooth fitness curve that is monotonically increasing in a logarithmic manner and is several multiples better than the fitness achieved by the next-best algorithm, GA. GA finds a local maxima quickly and sees moderate improvement in fitness over the remaining iterations studied. RHC and SA perform comparatively poorly and barely register on the fitness chart in *Figure 7*. GA and GD both have high accuracy (> 90%) on the mean of cross-validated test sets, with GA performing better at lower sample sizes and GD pulling ahead at larger ones. RHC and SA struggle to be 50% accurate. GA is by far the slowest algorithm regarding fit time. For large sample sizes, training the network can take between 2-3 minutes whereas GD is near zero for all sample sizes. RHC and SA negatively affect training time, although not as egregiously as GA. All algorithms have negligible score times.

5.4 Analysis

The hill climbing algorithms (RHC and SA) perform significantly worse than GD and GA with regard to fitness over the same number of iterations. These algorithms perform comparatively poorly because, unlike GD and GA, they must pick each new step individually without knowledge beyond their current location. GD's use of gradients to determine the direction the weights should approach and GA's ability to combine well-suited states means they will--usually--provide superior performance over the same number of iterations. The difference between the fitness of GA and GD is due to GA's dependence on randomization to find and breed good states, while GD uses calculus to mathematically determine the direction weights should move to provide a better result. Based on the shape of their fitness graphs (logarithmic for GD vs. step/linear for GA) it is possible that at some high number of iterations the fitness score of GA could eclipse that of GD. However, due to the significant additional cost in training time to use GA and the likely extreme number of iterations of the algorithm to achieve this state, we can assume that the benefit would not be enough to select GA over GD for this problem. Because GA has a higher accuracy for small sample sizes, we could argue that it does a better job of generalizing in those

situations--particularly when we see the disparity in fitness scores. However, as more samples are added the accuracy of GA begins to wane while GD continues its strong performance. This is due to GD being able to generalize better with more examples.

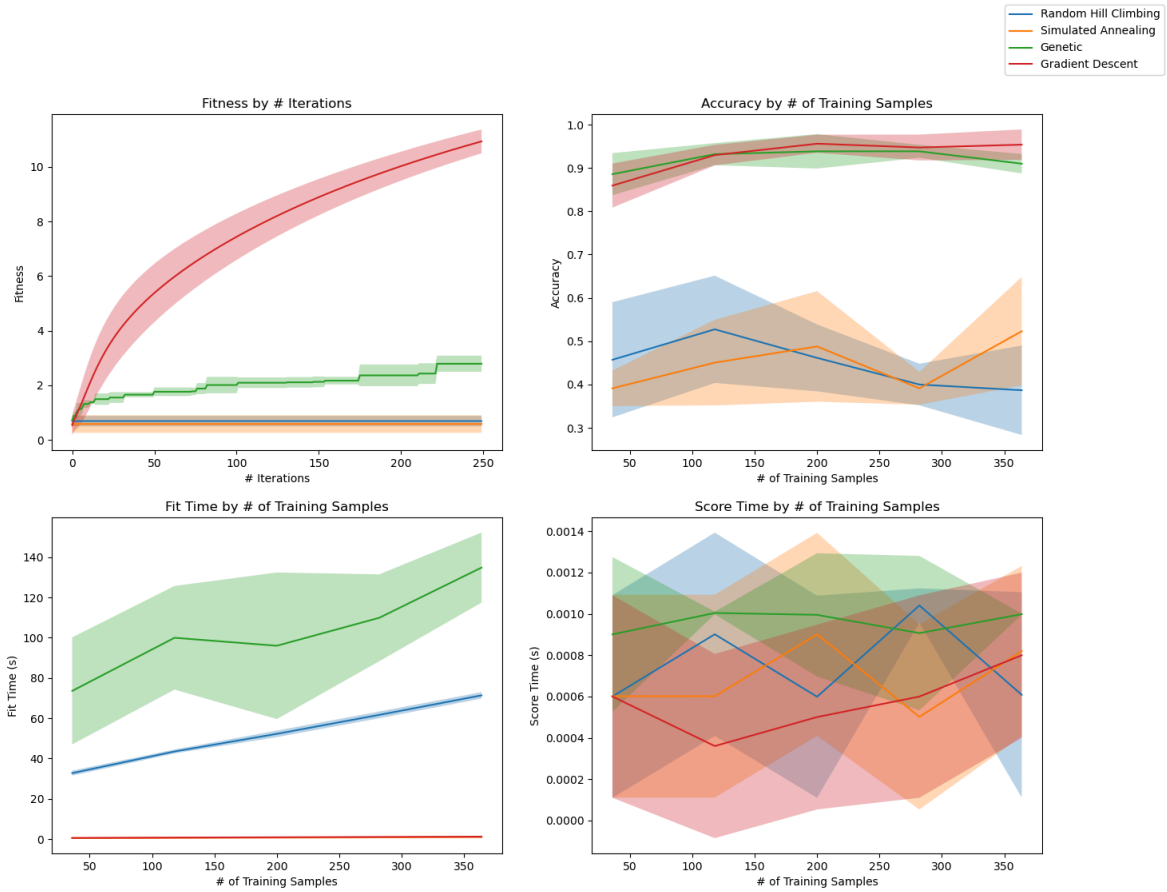

Figure 7

5.5 Conclusion

While GA performs better with regard to fitness for very small sample sizes, GD's train speed and continued fitness/accuracy improvement over new data samples makes it the correct choice for this network.

6 REFERENCES

1. An Analysis of the Operational Costs of Trucking: 2023 Update. (2023, June). Retrieved October 18, 2023, from https://truckingresearch.org/wp-content/uploads/2023/06/ATRI-Operational-Cost-of-Trucking-06-2023.pdf
2. FedEx® Standard List Rates. (n.d.). Retrieved October 18, 2023, from https://www.fedex.com/content/dam/fedex/us-united-states/services/FedEx_StandardListRates_2021.pdf
3. Fitness Functions — mlrose 1.3.0 documentation. (n.d.). Retrieved from mlrose.readthedocs.io website: https://mlrose.readthedocs.io/en/stable/source/fitness.html
4. Machine Learning Weight Optimization — mlrose 1.3.0 documentation. (n.d.). Retrieved October 18, 2023, from mlrose.readthedocs.io website: https://mlrose.readthedocs.io/en/stable/source/neural.html#
5. Overview — mlrose 1.3.0 documentation. (n.d.). Retrieved from mlrose.readthedocs.io website: https://mlrose.readthedocs.io/en/stable/source/intro.html
6. sklearn.datasets.load_breast_cancer — scikit-learn 0.24.1 documentation. (n.d.). Retrieved from scikit-learn.org website: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html
7. sklearn.preprocessing.MinMaxScaler — scikit-learn 0.24.1 documentation. (n.d.). Retrieved from scikit-learn.org website: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html#sklearn.preprocessing.MinMaxScaler