Abstract

This report analyzes 4 Randomized Optimization algorithms over two areas of problems.
The 4 algorithms used are Randomized Hill Climb, Simulated Annealing, Genetic Algorithms and MIMIC.
GitHub Readme and repo under Project 2– https://github.com/Maimoons/7641/blob/master/README.md

Introduction

The report has two sections for the two different problem domains.

- First problem - The first problem set analyses running 3 of the aforementioned randomized optimization algorithms to find the weights for a neural network from project 1 on one of the datasets - Titanic dataset. A 5-fold cross validation was performed with different hyperparameters specific to each algorithm.

- Second problem – The second problem set analyses running the above 4 randomized optimization algorithms on 3 optimization problems: continuous peaks, traveling salesman and knapsack. Each of these problem highlights the strengths of one algorithm over the others based on the structure of the problem and the algorithm.

Randomized Optimizations Algorithms:

Randomized Hill Climb – This algorithm iteratively attempts to find the optima by starting at a point and searching the neighbors and moving in the direction that maximizes it. This is a local search technique and is susceptible to getting stuck in local optima by the nature of it. To overcome this problem, multiple restart positions are tried randomly which is one of the hyper parameters that was tuned.

Simulated Annealing – This algorithm has metallurgy analogy of heating and then allowing for cooling. This algorithm tries to optimize for global optima. It selects its point based on a temperature dependent probability space. It starts with a high temperature which is slowly decreased based on a decay schedule, both being hyperparameters that were tuned. Decreasing temperature allows to explore worse solution and be search heavy for those temperature, which allows it to not get stuck in local optima.

Genetic Algorithm – This algorithm has natural selection analogy where it starts with a population of points and iteratively generates a new generation by taking the more fit points from the generation before and with mutation, cross-over generate new fit points. This process of natural selection is continued to eventually reach the optimized population.

Problem 1: Neural Network weights Randomized Optimization

Datasets

The Titanic dataset was used with the following statistics after preprocessing:

Titanic dataset: The titanic dataset is a famous dataset taken from Kaggle's list of datasets.

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare
count	1098.000000	1098.000000	1098.000000	1098.000000	1098.000000	1098.000000	1098.000000
mean	-0.021624	-0.111052	0.106857	0.010601	-0.007654	0.011317	0.069395
std	1.015757	1.037098	1.027408	1.034430	0.946710	0.975422	1.052189
min	-1.730108	-1.566107	-0.737695	-2.253155	-0.474545	-0.473674	-0.648422
25%	-0.896157	-1.566107	-0.737695	-0.592481	-0.474545	-0.473674	-0.488854
50%	-0.019439	0.827377	-0.737695	0.000000	-0.474545	-0.473674	-0.324253
75%	0.850474	0.827377	1.355574	0.407926	0.432793	0.457304	0.095367
max	1.730108	0.827377	1.355574	3.870872	6.784163	6.974147	9.667167

Statistics after preprocessing

Raw and unclean data do not offer as well as inferences as clean, standardized data do. Preprocessing was done to remove inconsistencies and noise and fill up the missing values. Preprocessing steps included feature selection and resampling to have a more balanced dataset. The dataset is a classification problem predicting if each passenger survived or not survived. 7 input features in total were trained that did not have high correlation of over 95%. It was resampled to 1098 rows with an equal split for both the y labels to prevent skewed learning.

Learning Technique:

The data was split into train 80% and 20% test datasets. Mlrose – hive library was used to use libraries for the randomized optimization algorithms. The neural network classifier was used with the desired algorithm and parameters options passed in. Sklearn's GridSearchCV was used for the 5 folds cross validation and the best hyperparameters were then used to test on the held-out test set. Time to train the with the best hyper parameters, time to test with best model on the test set and the test accuracy was compared across algorithms.
Validation was also performed with the best hyper parameters on different sized problem sets to observe the validation and train accuracy for each algorithm to analyze which algorithm was more heavily dependent on the input size.
Finally, the randomization aspect was analyzed by looking at the fitness function for the best hyperparameters across iterations for each of the algorithm and looking at the trends comparatively.

1500 maximum iterations were used with a maximum attempt of 150 for each iteration. A hidden layer of 15 nodes were used with early stopping enabled. This structure was what performed well from project 1 which used back propagation. So that was retained, and the randomized algorithm specific hyper parameters were tuned.

Experiments

1 Randomized Hill Climb

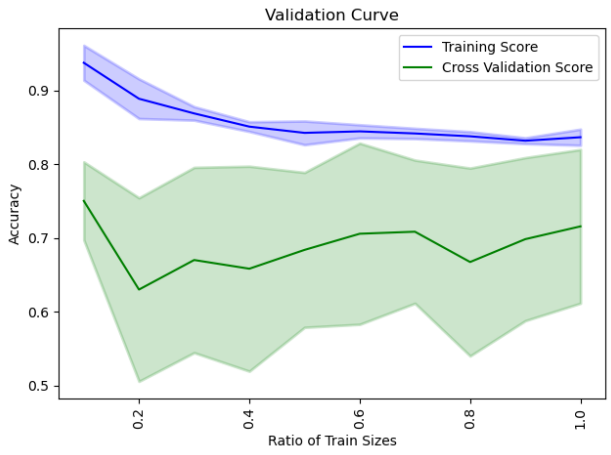
The Random Hill algorithm was used from the mlrose -- hive library. This allowed us to abstract the core implementation and focus on tuning the hyperparameters for the classifier to perform better.

Best hyperparameters:

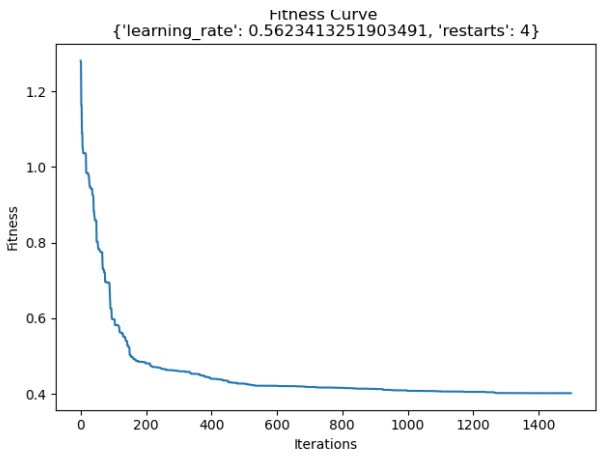
Titanic: {'learning rate': 0.5623413251903491, 'restarts': 4}

The grid search technique was performed on the above hyperparameters to ensure that all possible combinations were tried to pick the best combinations of the parameters.

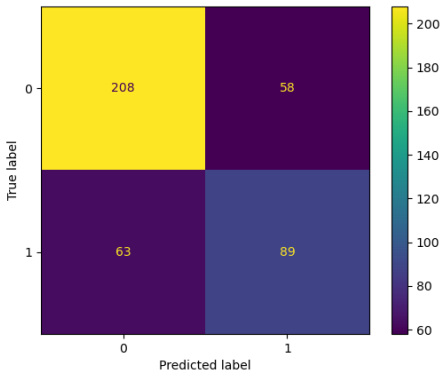
Learning Curve



Fitness Curve



Test Metrics



RHC performed poorly overall. This is because the problem of weight optimization is not strictly convex. From the learning curve, the training accuracy decreased with train size and the validation accuracy decreased as well. The algorithm did not even fit the training data that well, mainly underfitting by converging to a local optimum. The fitness decreased as well with increasing iterations suggesting RHC is not suited for this problem set and dataset. It potentially found a local optimum and with increasing number of restarts performed worse with increasing iteration which explains a low best number of restarts of 4. The train accuracy of the best classifier was 82.6 % and the test accuracy was 71.1%.

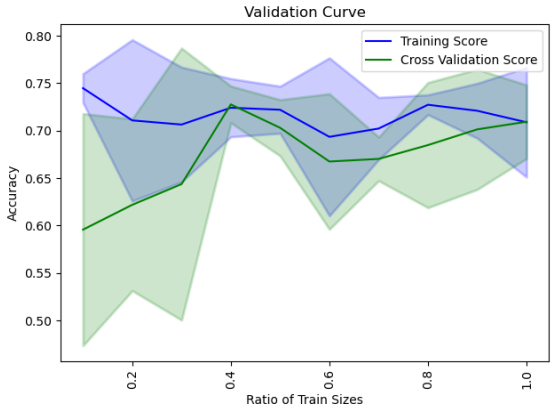
2 Simulated Annealing

Learning rate and different schedules for the temperature were hyper tuned.

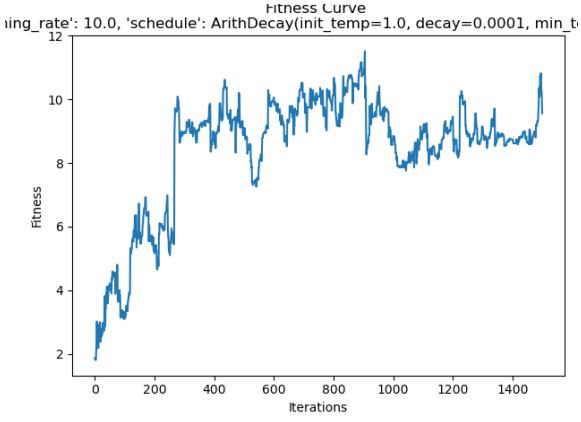
Best hyperparameters:

Titanic: {'learning rate': 10.0, 'schedule': ArithDecay(init_temp=1.0, decay=0.0001, min_temp=0.001)}

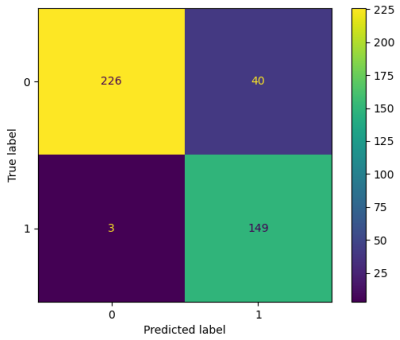
Learning Curve



Fitness Curve



Test Metrics



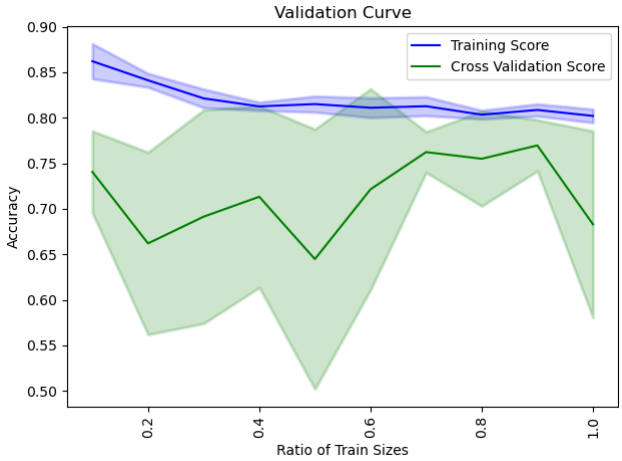
The simulated annealing performed better than RHC as expected. That is because the weight optimization has local optima which the RHC was stuck on, but simulated annealing was able to escape with an increased temperature to explore the input space. The fitness increased with increasing iterations and continued increasing so SA could have performed better with more. Arithmetic schedule to decrease the temperature performed the best with a high learning rate suggesting large strides per iteration. It achieved a high best fitness of around 12 compared to 2 of that of RHC. The train accuracy of the best classifier was 89.7 % and the test accuracy was 72.3%.

3 Genetic Algorithm

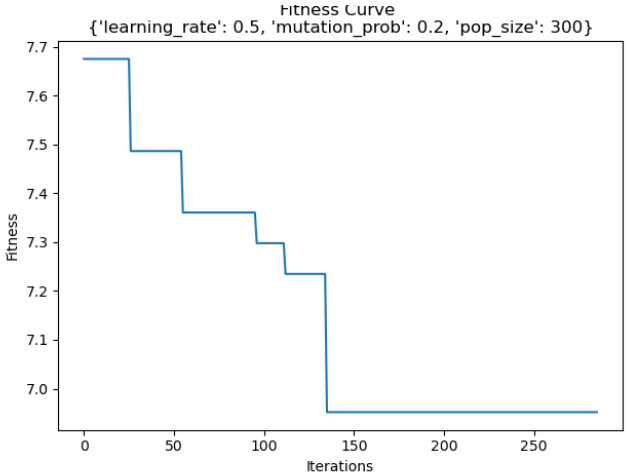
Best hyperparameters:

Titanic: {'learning rate': 0.5, 'mutation_prob': 0.2, 'pop size': 300}

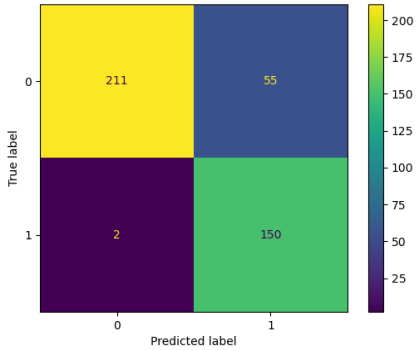
Learning Curve



Fitness Curve

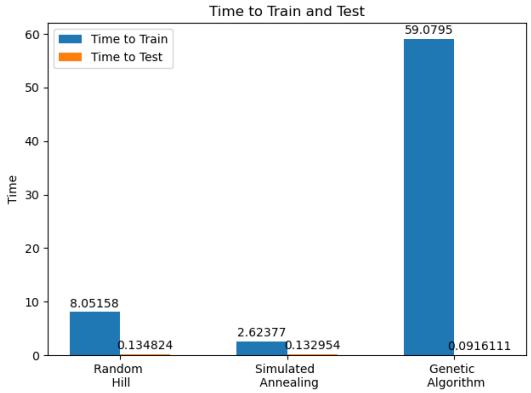


Test Metrics



Genetic algorithm performed better than RHC and comparable to SA. A population size of 300 with mutation rate of 0.2 performed the best. It achieved high fitness of around 8 but not as high of that of SA. Of the three algorithms, it generalized well and had the highest test accuracy. This suggests that probabilistically searching for the optima worked well for this problem space. The train accuracy of the best classifier was 86.9 % and the test accuracy was 79.8%.

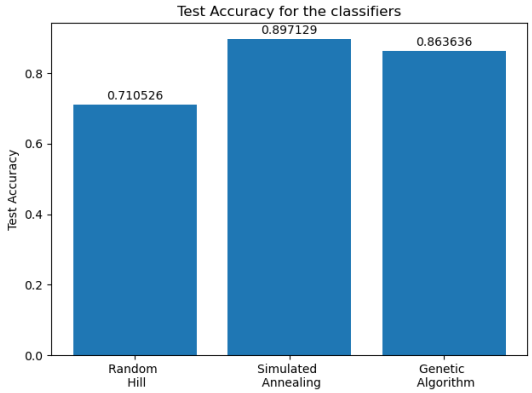
Time

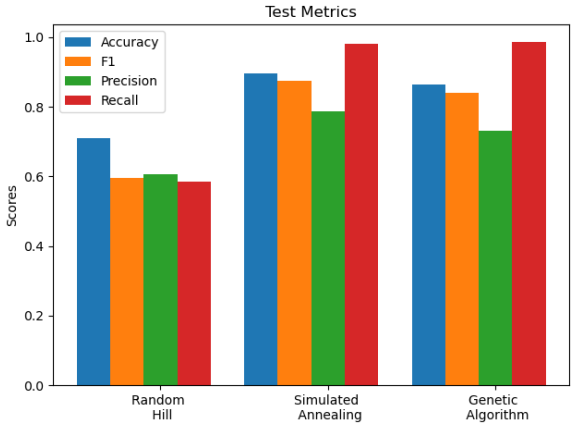


Genetic algorithm took significantly more time to train suggesting that finding population at each iteration was a costly procedure compared to the iterative steps in RHC and SA. However, it was the quickest to test.

Summary

Accuracy comparison





SA and GA had higher test accuracy of 89.7% and 86.3% compared to 84.9% of test accuracy with back propagation gradient descent taken from project 1. This was probably because gradient descent was susceptible to local optima trapping whereas these randomized algorithms could overcome that weakness by their structure. Gradient descent however still performed better than RHC and that could be dependent on the number of iteration and maximum attempts that were used for it. It could have performed better with a better setup for those parameters.

Problem 2: Continuous Peaks, TSP, Knapsack with Randomized Optimization

Experiments

Random Hill Climbing Algorithm: parameters
"restarts": 10,"max_attempt": 100, "max_iters": 1500,

Simulated Annealing Algorithm: parameters
"schedule": mlrose_hiive.GeomDecay(10, 0.5, "max_attempt": 100, "max_iters": 1500

Genetic Algorithm: parameters:
pop_size_genetic": 200, "max_attempt": 100, "max_iters": 1500

MIMIC Algorithm: parameters
pop_size_mimic": 200, "max_attempt": 100, "max_iters": 1500

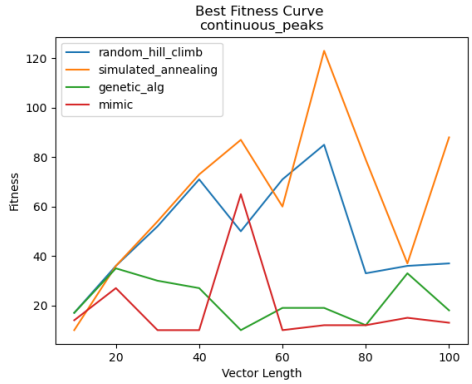
Optimization Technique:

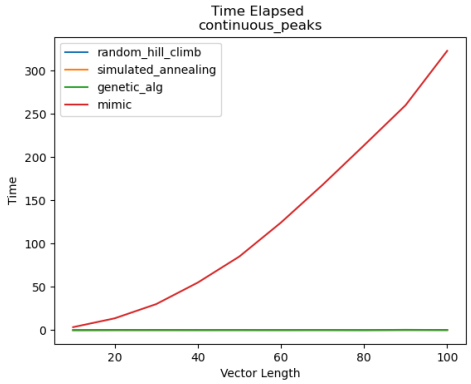
All the algorithms were tested with varying length of the state vector for increased complexity. The best fitness across all the iterations were plotted for all algorithms together to observe the overall trend of the fitness and determine the best performing algorithm on the problem. The fitness curve for each length of the state vector was also plotted and a few, but not all are shown below. Lastly, the time elapsed was plotted as well against the state vector size.

1. **Continuous Peaks**

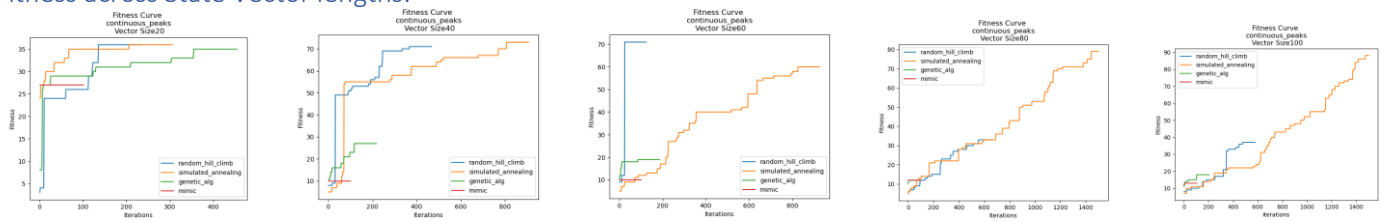
This is a maximization problem for a bit-string, consists of counting the number of 0s and number of 1s. The fitness function evaluates the length of the maximum runs of 0s and 1s and if they are above a certain threshold, it adds a bonus to the function. This problem contains multiple local optima and is good to analyze the randomized algorithms on it. There are multiple maximal peaks but only a few maximum peaks which is the global optima.

Best Fitness Curve





Fitness across State Vector lengths:



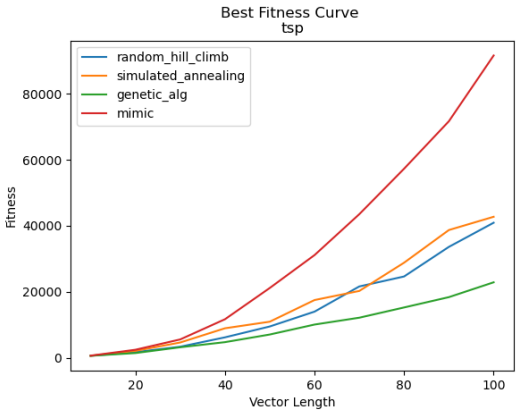
Setup: The length of the state vector was 10 to 100 with a step size of 10. This was a discrete problem with the state vector being a bit string. A threshold of 0.15 of the size of the input string was used in the fitness function.

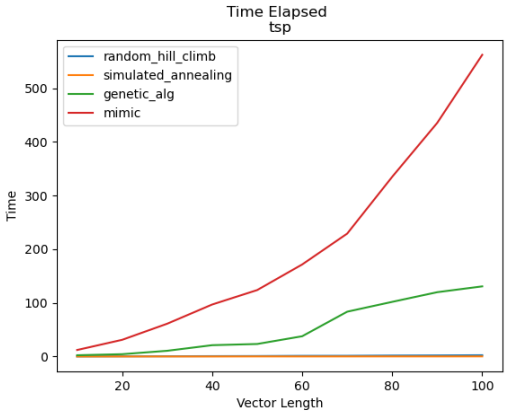
Performance: The best performing algorithm was **Simulated Annealing**. Across multiple lengths of the state vector, the fitness increased over the iterations and it reached a high fitness, close to the maximum in most of them. Simulated Annealing plataued later across iterations than the other algorithms, depicting how it continued to optimize. This was because the problem had multiple local optimas, it was able to overcome this problem by varying the temperature and both searching out and optimizing. It was also the fatest performing algorithm. The worst in terms of fitness and time performance was mimic and this could have largely been due to the parameter set of the mutation and population because of which it plateaued on a lower fitness for most problem sizes. Because of the complex underlying structure of this problem, Simulated Annealing plateaued faster on optimal fitnesses on a lower iteration numbers, suggesting that given the setup the algorithms were able to find the optima relatively quickly.

2. Traveling Salesman Problem

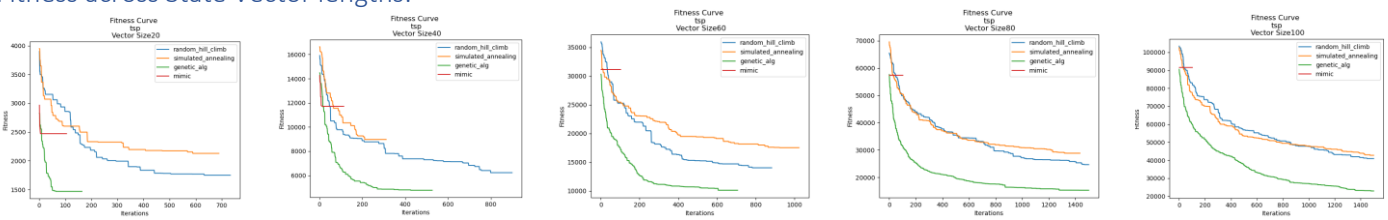
This is a minimization problem which is NP hard. The problem consists of pair of cities and distances between them with the aim of visiting each city once and finding the shortest total distance travelled. Modeling this as a graph, the cities are coordinates. The fitness function therefore evaluates the distances between the coordinates with the same constraint of visiting every point once and trying to minimize the sum of total distance.

Best Fitness Curve:





Fitness across State Vector lengths:



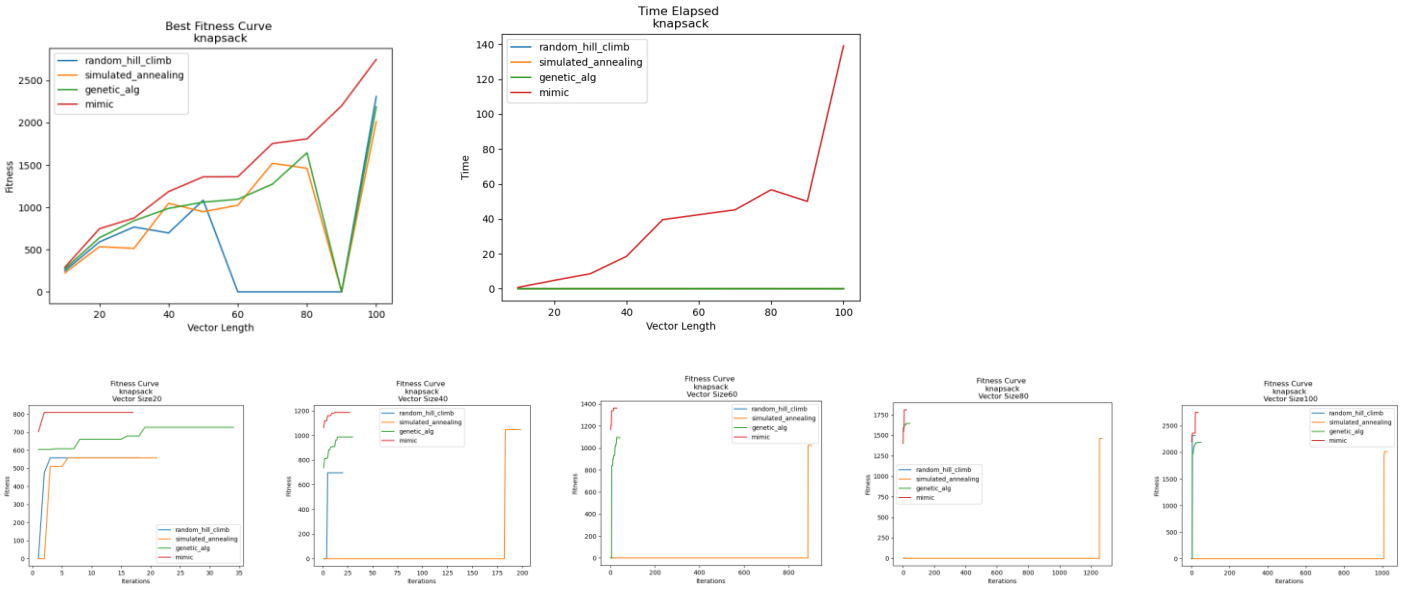
Setup: The length of the state vector was 10 to 100 with a step size of 10. This was the number of cities and the coordinates between them were randomly, uniformly selected from a negative to positive range dependent of the vector length.

Performance: The best performing algorithm was **Genetic Algorithm**. This was a minimization problem and GA reached the lowest fitness of around 2000 for the largest problem size which is the smallest, optimized distance. It also had significantly good performance time comparable to Simulated Annealing and RHC that also performed well. The fitness for both genetic algorithm and simulated annealing increased with increased vector lengths of the problem as expected because of increased number of cities to visit. For each vector length, the finess did decrease but was significatnly lower than MIMIC for bigger problem sizess.

3. Knapsack problem

This is a maximization problem. It consists of a set of items each with a weight and a value with the aim of maximizing the total value of the items chosen alongside the constraint of total weight of the items chosen to be below a certain capacity. The fitness function is therefore calculating the values of the items and maximizing it. This problem too can have multiple local maximal values and is a good candidate for observing performance of randomization algorithms.

Best Fitness Curve



Setup: The length of the state vector was 10 to 100 with a step size of 10. The weigths werer randomly and uniformly chosen between 0.1 to 1 whereas the value was randomly chosen between 1 to 100.

Performance: The best performing algorithm was **MIMIC** however it was also very costly and took the longest time to converge across all vector lengths. The best fitness for MIMIC was above the other three algorithms significantly across all vector lengths. It's fitness also increased across the number of iteration, and had not plateaued, suggesting it could have continued to improve further with more iterations. This could be due to the nature of the problem where MIMIC took advantage of the structure and capitalized of probabilty mass to reach best convergence.

Conclusion

Conclusively, MIMIC performed well on complex problems where it modeled the input space in a structure probabilistic manner such as the knapsack. Genetic Algorithm performed well on the the minimization problem of Traveling Salesman. Genetic Algorithm was significantly faster in all scenarios than MIMIC. Simulated Annealing performed well in cases where

there were multiple local optimas such as continuous peaks as well as optimizing the weights of the neural network. It performed better than gradient descent back propagation highlighting the curse of getting stuck in local optimas which gradient descent can't escape as well as Simulated Annealing with the temperature control probablisitic approach. RHC and SA reached plateaus faster than other algorithms, and across all problem spaces had comparable performances.

References

[1] https://github.com/hiive/mlrose
[2] https://readthedocs.org/projects/mlrose/downloads/pdf/stable/
[3] https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3
[4] https://towardsdatascience.com/optimization-techniques-simulated-annealing-d6a4785a1de7