Randomized Optimization Project

Part 1: Neural Network training

The UCI Magic Gamma Telescope Data set [3] contains data generated from a Monte Carlo program to simulate an atmospheric Cherenkov gamma telescope observing high energy gamma rays from the Earth. If the threshold is correct, the scope takes a pattern of Cherenkov photons called a shower image. Following, the image is put through some pre-processing to generate the 10-dimensional features in the dataset. The output holds an even distribution, roughly 1 hadron to 2 gamma.

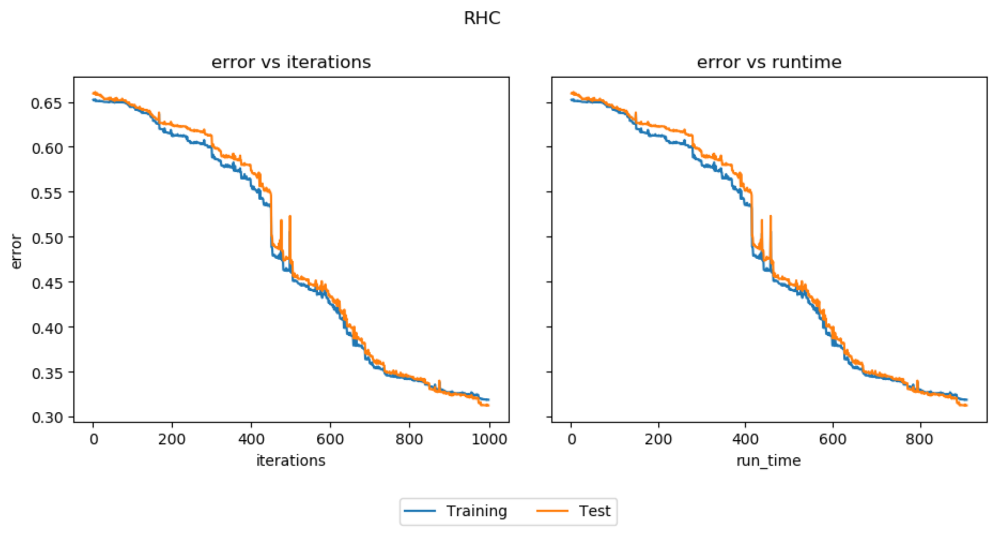
Backprop Neural Network

The previous paper covered the experimentation on neural networks trained using backprop. Two different hidden layers were used 100 and 130,130. Both produced similar accuracy and trends. Additionally, each configuration was tested against 9 combinations of learning rate and momentum. Overall the better choices of hyperparameters produced training and test accuracy around 80% while seemingly not overfitting at 2000 iterations [4].

Randomized Optimization

To further explore neural networks on this dataset, three randomized optimization algorithms were tested to set the weights of the neural network instead of backprop. The ABAGAIL library was used in specific a modified version of their abalone_test.py. Randomized hill climbing, simulated annealing, and genetic algorithms were all tested to see the error over 1000 iterations. To set a good comparison, the network's hidden layers were set to 130,130 as they were in the first paper.

Randomized Hill Climbing

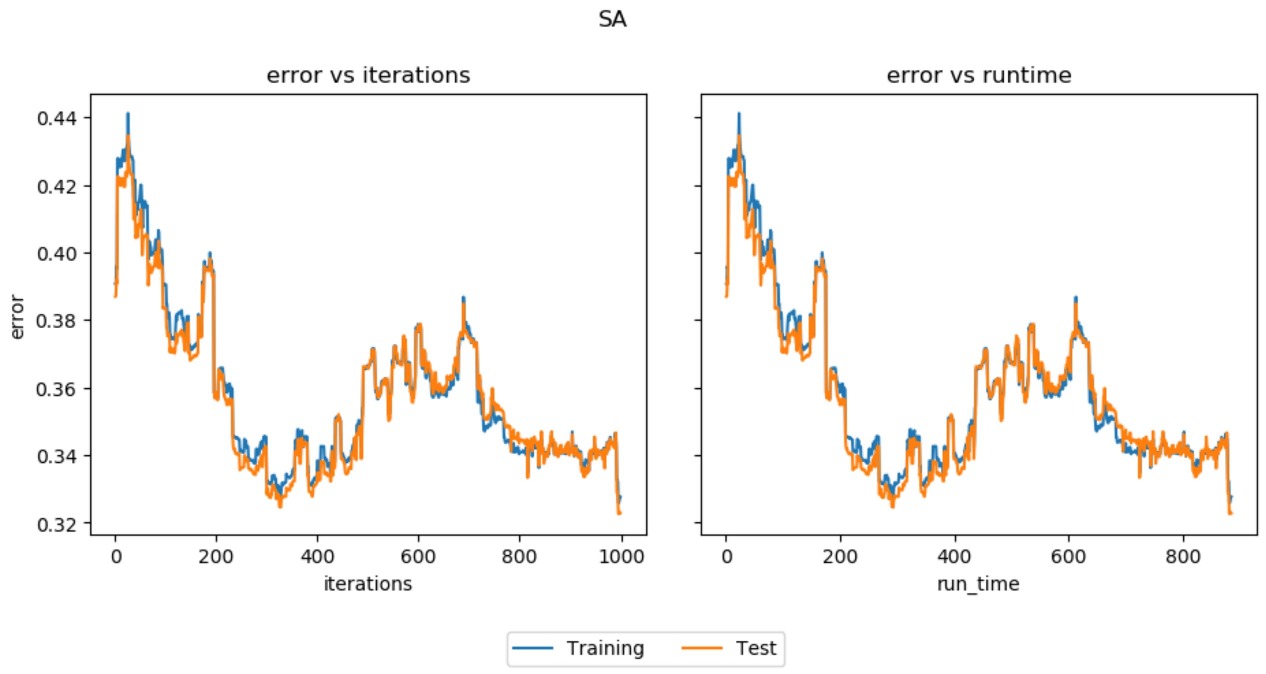


Randomized hill climbing searches from its random starting point moving generally in the direction of the gradient with some randomization elements to occasionally move in another

direction. This makes it susceptible to local optima; however, it uses random restarts to try to escape them.
When applied to the neural network to find the weights, the error decreases steadily towards 30%. A full 10% more error than the ANN used in the previous paper. It has seemingly followed mostly a single direction to improve the error and has plateaued in improvement around 800 iterations. An interesting note is that there are a few spikes in error around 500 iterations. These might be due to some of the randomness in the algorithm where it moved to a less favorable setting than continues towards a better one.

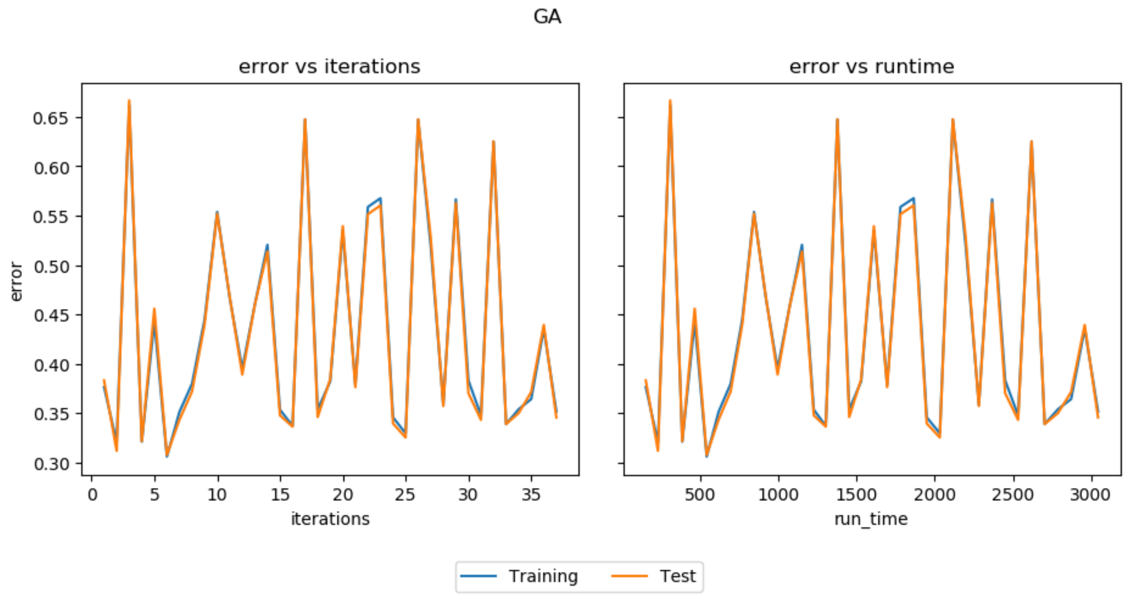
Simulated Annealing

Simulated annealing searches from its random starting point similarly to randomized hill climbing however it adds a twist. The analogue made is to blacksmiths annealing metal to make it less brittle. The process starts at a high temperature so the optimal point moves quickly in the direction of the gradient. As the iterations go on it cools to a smaller temperature slowing down. This allows the algorithm to explore the hypothesis space early on and move past local optima, but then as it cools it will optimize to some maxima, hopefully the global one. In theory it should be able to more quickly find better optima than just RHC.
When applied to the neural network to find the weights, the error decreases slightly jaggedly towards 30%. Again, a full 10% above the 20% of the original network. Interestingly it follows a much different path than RHC. This makes sense considering the exploration part of the algorithm thus I would expect it to smooth out some larger iteration.



Genetic Algorithms

Genetic algorithms searches by pulling a population of the hypothesis space. It then uses mutation of bits and crossover of members of the population like chromosomes. It uses the fitness function to determine which is the optimal of the generation. The crossover function is very important to the domain GA is being applied upon. The population allows this function to form a sort of memory of what it has seen and hopefully cluster together data of more optimal data unlike the previous two greedy algorithms.
When applied to the neural network to find the weights, the error jumps between ~55-65% and ~35%. This suggest the data does not lend itself to the GA crossover function's ability to remember data. Likely there are many adjacent very poor selections surrounding optimal or crossing the data to form new generations can easily result in poor selection. So, GA's attempts to learn and explore the set causes it to rapidly jump between these basins of good and bad selections. Interestingly, GA reached near the lower end of the error in less than 3 iterations. However, 10 iterations take roughly the same time as 1000 RHC or SA iterations.



Conclusions

The neural nets overall performed worse when using randomized optimization; however, the RHC and SA trained nets showed more improvement over time suggesting that they could trend towards equivalent performance given enough time. It is interesting to note all three random algorithms experience some spikes. GA especially is affected by this. This suggest that the error landscape of the dataset has a lot of local minima, but possibly no true global minima for this sized net. GA is hopping between these as it goes through its crossover between generations causing the spikes. RHC and SA are more forced into these local minima. It likes likely that the neural network from the first paper got lucky with its starting weights leading it into one of the better local minima and backprop was unable to search elsewhere.

This displays that GA can be ineffective when the underlaying problem has no strong structure while greedy solutions like RHC, SA, and normal backprop can solve to local minima much faster to take advantage of the roughly equivalent minima in this problem.
Finally, it is interesting to note none of the nets experienced overfitting. In fact, all four have testing and training errors tightly coupled. This suggests that the data is largely uniform between the two sets. However, the fact that all four find an optimal in the 20-30% error range likely means roughly 25% of it does not fit with any local optima at all thus a tradeoff was made to mislabel those in favor of the other ~75% of the data. This dataset is emulating a sensor which often contain a lot of noise. Those 25% could be caused by noise or garbage data being thrown in 75% clean data. Alternatively, the sensor may not full cover all the data needed to classify the full set.

Part 2: Optimization Problems:

Four randomized optimization algorithms, randomized hill climbing, simulated annealing, genetic algorithms, and MIMIC, were tested on three optimization problems to demonstrate different properties and utilities of the algorithms. The tests were implemented using ABAGAIL.

Method

All tests were run to 10000 iterations except MIMIC as it is designed to converge faster but typically takes longer per iteration thus it was run to 5000 iterations. Following are the hyperparameters of each algorithm that were used in a grid search to find the best.
Simulated Annealing (SA)

- Temperature: 1E2, 1E5, 1E10
- Cooling: 0.6, 0.8, 0.95

Genetic Algorithm (GA)

- Population Size: 20, 200, 300
- toMate: 0.5, 0.75, 1

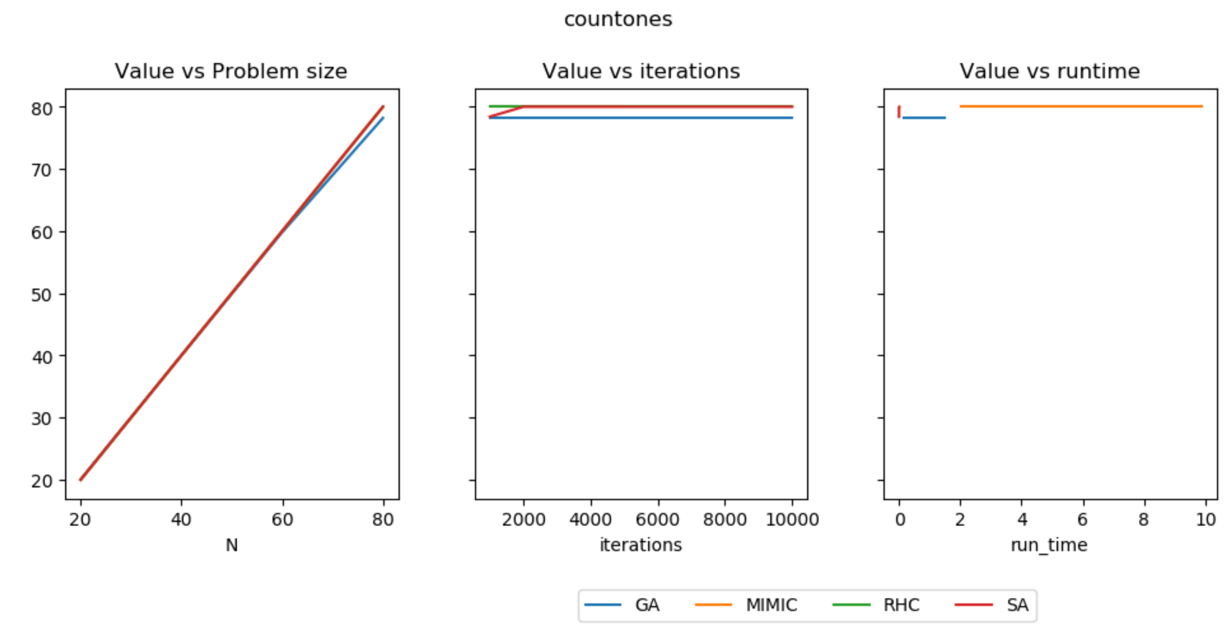
MIMIC

- Samples: 50, 100, 200
- toKeep: 0.1, 0.2, 0.5

The best hyperparameters were defined as the set which achieved the highest fitness without sacrificing orders of magnitude more time when N=80. The best setting was used in the graphs.

Count Ones (SA)

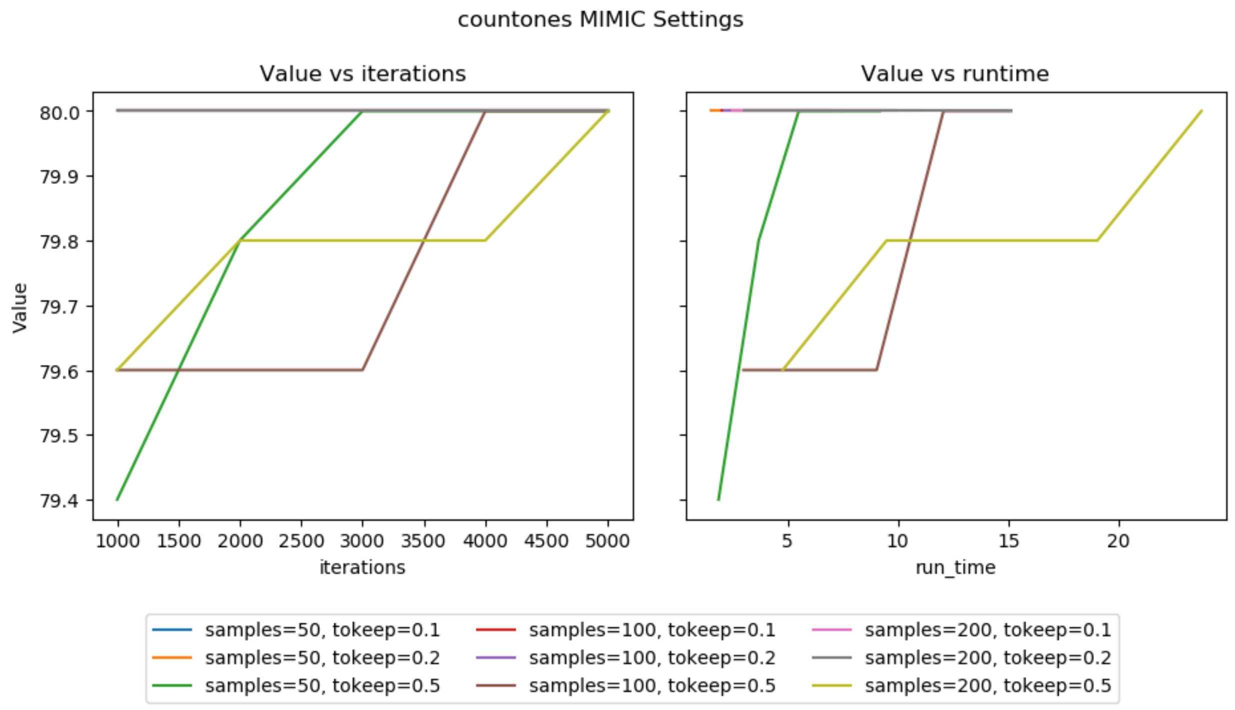
Count ones is a relatively simple concept. To goal is to optimize the number of ones in a N bit vector. While straight forward in concept, the difference between 255 and 256 is 7 ones in a N=9 bit space. Thus, the function is not flat at all.



This problem was run for N = 20, 40, 60, 80. All Algorithms except GA on larger N reached the optimal value. This is reasonable since overall the problem is straight forward and thus even a greedy algorithm such as RHC and SA can easily reach the correct answer.

Genetic algorithms found optimal or near optimal results for all problem sizes tried. However, it is interesting to note that it has begun to drop below optimal at N = 80. This trend is likely to continue as its adhoc way of utilizing previous information will help less and less as the size of the problem increases as its method of memorization does not work well when the pattern is so volatile which can be seen with the little improvement of 10000 iterations. Additionally, it took significantly longer time to find these subpar results. However, the largest factor on GA's optimal value is the population size thus it is possible that continuing to increase it would stave off dropping from optimal; however, this massively increases runtime.

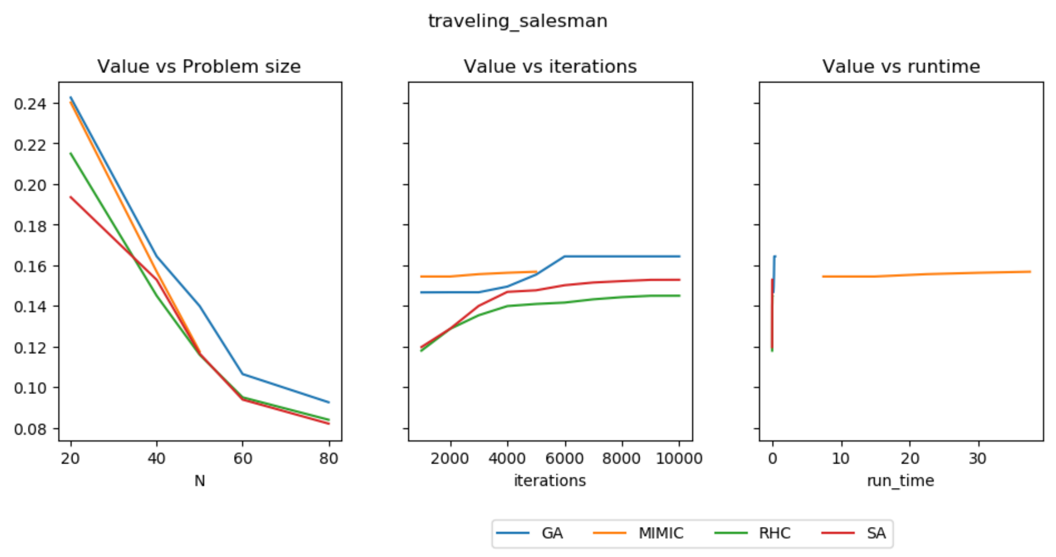
MIMIC produces optimal results at all values. This suggest its method leveraging the past works much better than GA on this problem. However, it takes even longer than GA and order of magnitudes longer than SA and RHC to find these results. For MIMIC the number to keep was the driving force in its optimal value as seen below. This suggest that most samples from the initial uniform distribution were poor and mislead the algorithm at first.



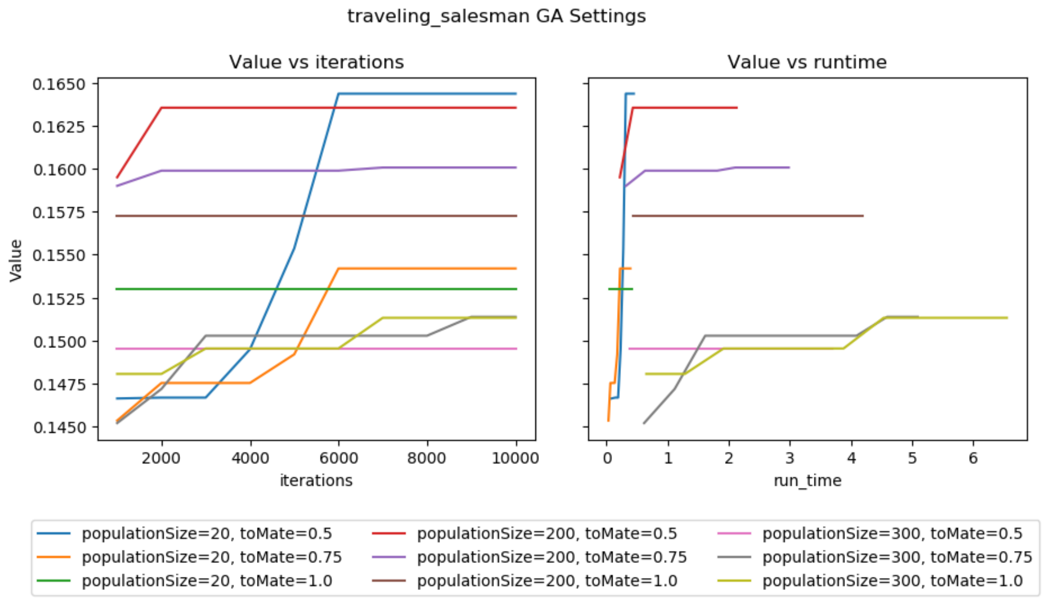
SA produces optimal results at all problem sizes. This suggest that the temperature property allows it to easily move through the error landscape which has a lot of divots and find the optimal location. The error landscape can be thought of as ramp leading towards the optimal N ones solution; however, the ramp is littered with large drops when conversions like 255 to 256 occur. Thus, SA quickly rolls down this ramp ignoring those drops. RHC does so similarly but slower as it rely on random chance to move out of them. RHC was slower than SA during the tests. SA was 3 orders of magnitude smaller in clock time over MIMIC and GA. This holds for all 9 settings tested for SA.

Traveling Salesman (GA)

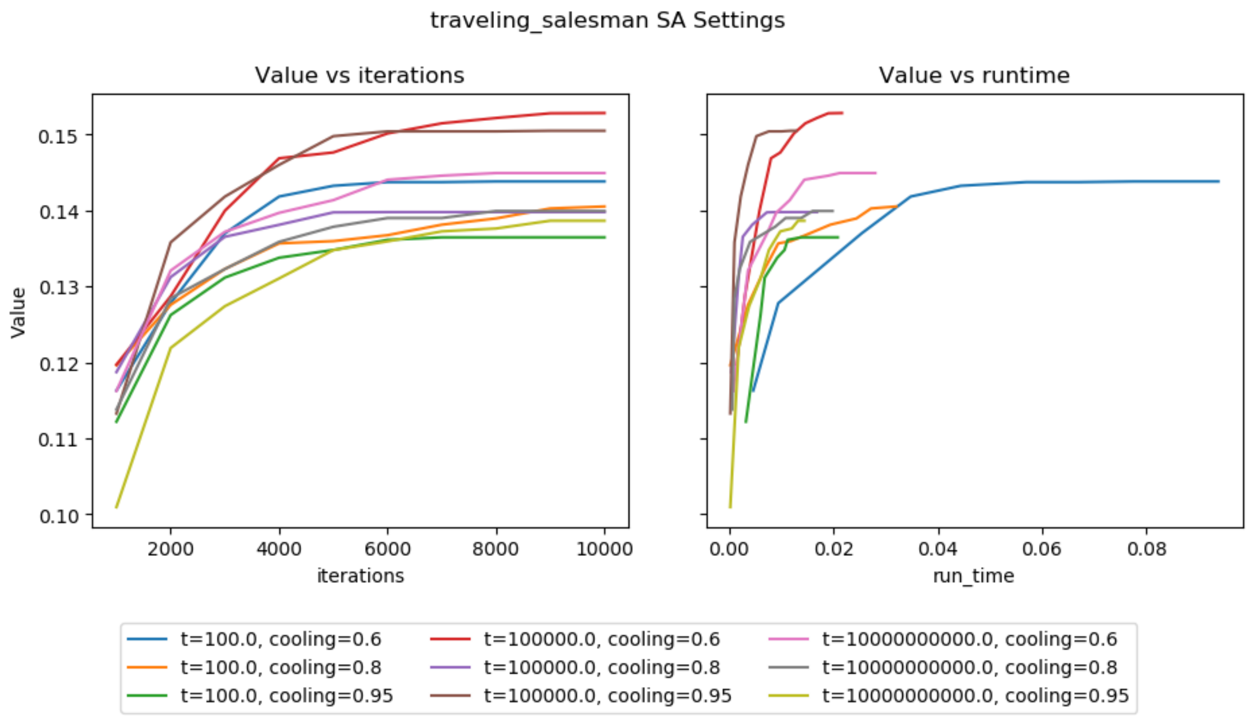
Traveling Salesman is a more complex problem as one of well-known NP-hard problems. The problem is best described as a graph where there are N nodes which represent cities. Each city is connected to nearby cities with some weight. The goal is to from the starting city travel to each other city and back to the starting city for as cheap as possible.



Genetic algorithms produce the best results for all problem sizes. This suggest the encoding of the problem into chromosomes and the crossover and mutation routines help the algorithm find more optimal solutions. Likely, this allows the algorithm to focus on subsets of the problem and find better solutions for subpart of the graph then when crossing over combining multiple optimized subparts to a globally better solution. The lower the mating rate, the better the algorithm performed. Likely this is due to the lower rate allowing more time to explore subparts. Additionally population size seems to parabolic with the middle size largely making up the higher values. This suggest the space of poor choose is high and introducing too many convolutes the data while too few reduces the chances of a good set. It is important to keep in mind that genetic algorithms for traveling salesman are not trivial to implement as the encoding, cross over, and mutation must be done such that the children chromosomes are legal (do not travel between unconnected nodes). Naïve solutions such as filling out the graph with infinitely weight edges would result in massive time and performance losses.

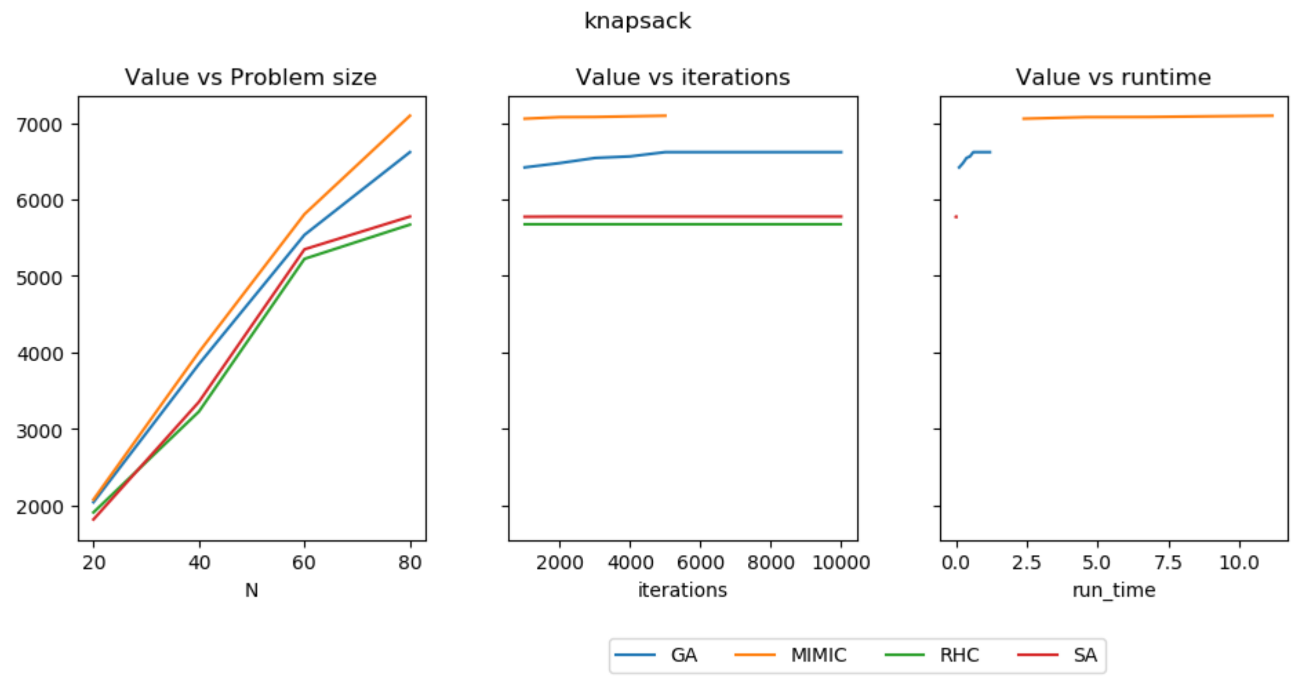


MIMIC performs similarly to GA on small problem sizes. However, as the problem size increases, the hypothesis space grows too large for it to create as accurate of a distribution resulting in losses in performance. Additionally, as the cost function is very expensive, MIMIC takes order of magnitude longer to solve this problem.
SA and RHC produce subpar results as this problem set is simply too large and too varied for the greedy approach to work well. The greedy algorithms all trend to flat as iterations increase suggesting they found local optimal and were unable to escape them. This might lead one to think the algorithm needs to search more thus a higher starting temperature or slower cooling rate. However, as seen below, the best two settings which are closely coupled in results is the middle temperature, 1E5, and the extrema cooling, 0.6 and 0.95. Additionally, the next best setting is again two closely coupled settings. This time it is the two extrema of temperature, 1E2 and 1E10, with a cooling schedule of 0.6. This suggest that the opposite is likely true. The tradeoff of exploring vs find a maximum rapidly favors the maximum as iterations go on. Thus, low cooling schedule seem to perform better as they transition from exploration quicker.

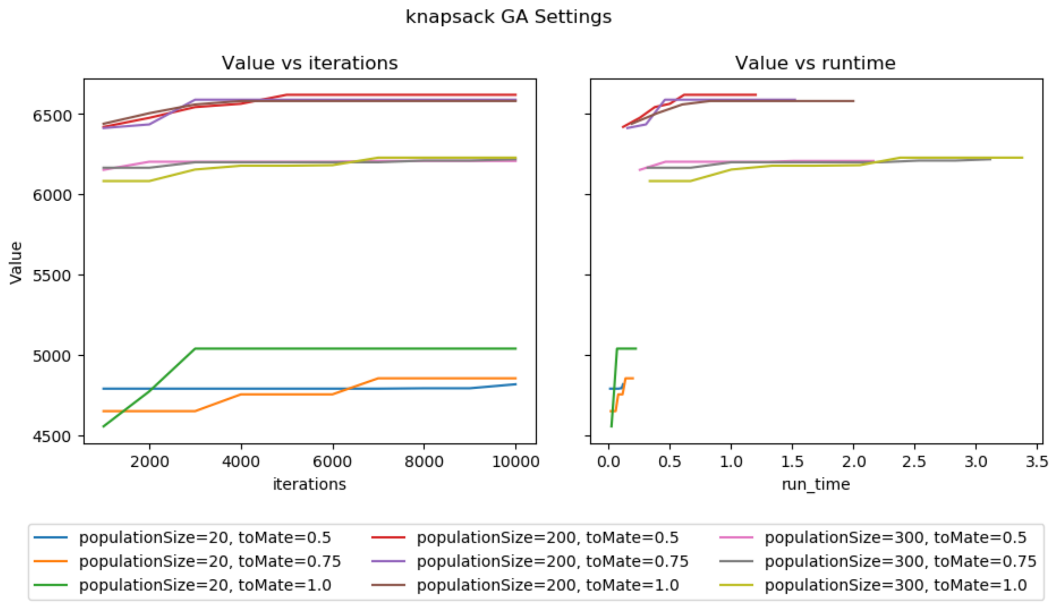


Knapsack (MIMIC)

Knapsack is another well-known optimization problem. In this problem, a given set of N items are given each with a specific weight and volume. A knapsack is filled such that is volume is maximized while the weight is minimized. Additionally, there is a max weight which can not be exceeded. This problem is NP-hard like the previous problem.



Genetic algorithms performed favorably well on this problem. The crossover function within genetic algorithm means it essentially samples the space; however, the crossover is only helpful when the two chosen clusters give insight into the underlying structure of the problem [1]. As GA, performs well, it suggests that it can provide insight during its crossover functions. Interestingly, the population size not the mating rate decides the effectiveness of GA on knapsack. On top of that, the preference is again parabolic favoring the middle hyperparameter over the smaller and larger population sizes, seen below, which suggest poor clusters are picked more easily thus it is a careful balance to have a large enough population without overshadowing it. Additionally, GA converges rather quickly to it final answer as compared to MIMIC.



MIMIC performs the best on the knapsack problem. MIMIC was not detrimentally affected by pseudo sampling like GA was by its crossover function allowing it to get a more representative distribution of the problem space. This allowed it to perform the best on this problem over GA. It is important to note that it did take much longer time wise while iterations wise took very little time. This is indicative of MIMIC's ability to use previously seen data to perform much better over fewer iterations which is especially helpful when the fitness function is expensive. MIMIC's runtime on knapsack was also much smaller than on traveling salesman while GA was a bit longer. This further reinforces the effect of the fitness function's cost.
SA and RHC both performed similarly and poorly. This np-hard problem does not lend itself to their greedy nature. The ability of SA to break from local maxima early on in its iterations seems to a have a marginal benefit in this problem. This illustrates the utility of using previous information in this problem.

Conclusions

These experiments showed the importance of understanding the problem space and how the different algorithms will act within them.
SA and RHC perform similarly with SA produces requiring typically less time and producing better results. SA however requires its temperature and cooling parameters to be correct fitted which as seen above intuition does not always work and empirical evidence might be the key. Both of those greedy algorithms fall to the side when the problem space is complex but has an underlying structure which can be used to find better results. MIMIC and GA both perform much better in these environments where their generally slower times are balanced out by far exceeding results. GA suffers when the problem does not lend itself to be well represented by the crossover function or when the fitness function is expensive. MIMIC generally produces strong results in few iterations. However, each of its iterations are much longer thus for some problems maybe untenable, but MIMIC does a great job of making the most of each call to the fitness function thus does much better in run time when the fitness function is expensive.

References

[1] De Bonet, Jeremy S, et al. "MIMIC: Finding Optima by Estimating Probability Densities."

[2] Kolhe, Pushkar. "ABAGAIL." Github, 8 Mar. 2018.

[3] Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[4] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.