George Maroun CS4641- Guzdial
Assignment #2

Randomized Optimization Report

Dataset

Diabetes: This is a binary-valued dataset gathered from a population in Phoenix, Arizona for identifying potential signs of diabetes according to the World Health Organization. This has been used as an adaptive learning routine to forecast the onset of diabetes mellitus. There is a total of 768 instances with 9 attributes including class.

The Problems Given to You

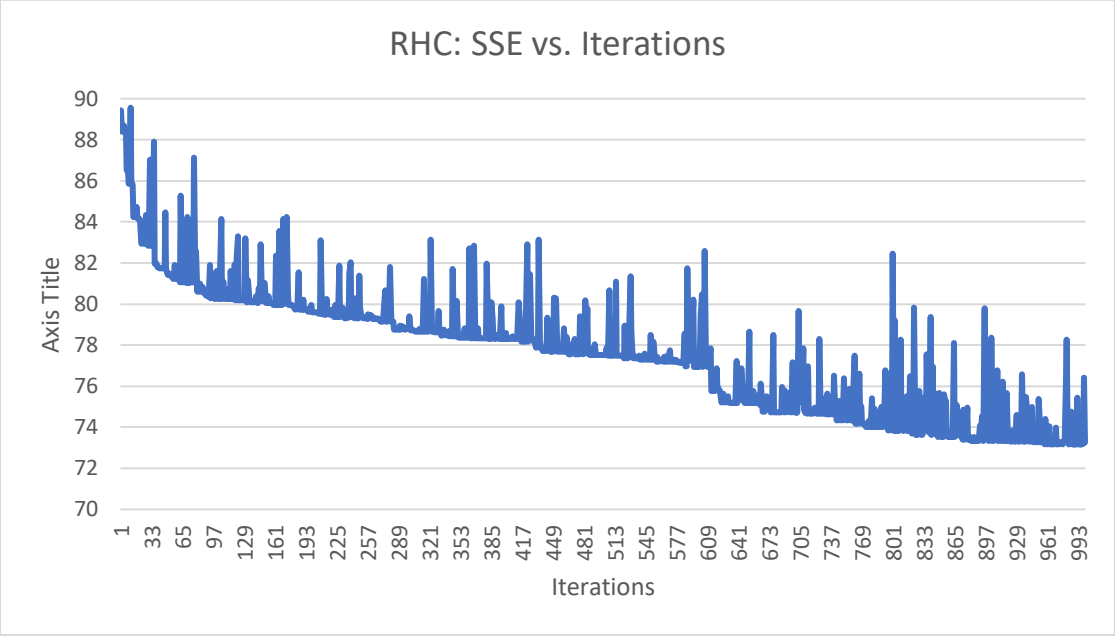
1. For this problem, I implemented the Randomized Optimization techniques and ran them to find good weights for a Neural Network instead of using backpropagation. As a baseline algorithm, I ran a Neural Net with backprop on WEKA as a Multilayer Perceptron. I used static hyper parameters such as setting the number of hidden layers to 20 and setting the number of iterations to 1000. The baseline trained insanely quickly, due to the hidden layer size, and received an accuracy of 81.5104%. Something to note is that the point of comparing these RO algorithms is to test the differences between random and non-random algorithms in regards to this specific dataset. Since all of these are using the same underlying approach for the neural network implementation structure, overfitting and other phenomena within the classifier are static as well, so none of these tests used cross-validation or a testing set. Instead, we compare the different standards based on the accuracy of the training data at a specified epoch size.
2. Also, all of the RO algorithms are being implemented with the ABAGAIL source code from the GitHub Repository, along with minor tweaks tailored to my specific dataset.
3. Lastly, as an introduction, because these algorithms I recorded many different trial runs using the same parameters to see how the results changed due to said randomness.

Randomized Hill Climbing

This algorithm chooses a random representation and compares with neighbor representations and then updates the function if said neighbor performs better, along with random restarts on the heuristic.

Trial	Training Time(s)	Train Accuracy %
1	4.034	71.094
2	4.214	71.354
3	4.107	69.141
4	4.095	67.188
5	4.147	70.052

For the Randomized Hill Climbing(RHC) as weight optimization instead of backprop, there were no individualized parameters to change, so instead I ran the same set on five different trial runs to flesh out an increased potential for accuracies to happen by coincidence alone. As shown through data comparison, the accuracy fluctuates considerably over trials (4%). Still, even through maximization, the backprop. accuracy beat this implementation out. I suspect that the nature of my dataset makes room for the RHC approach to settle on local maxima.



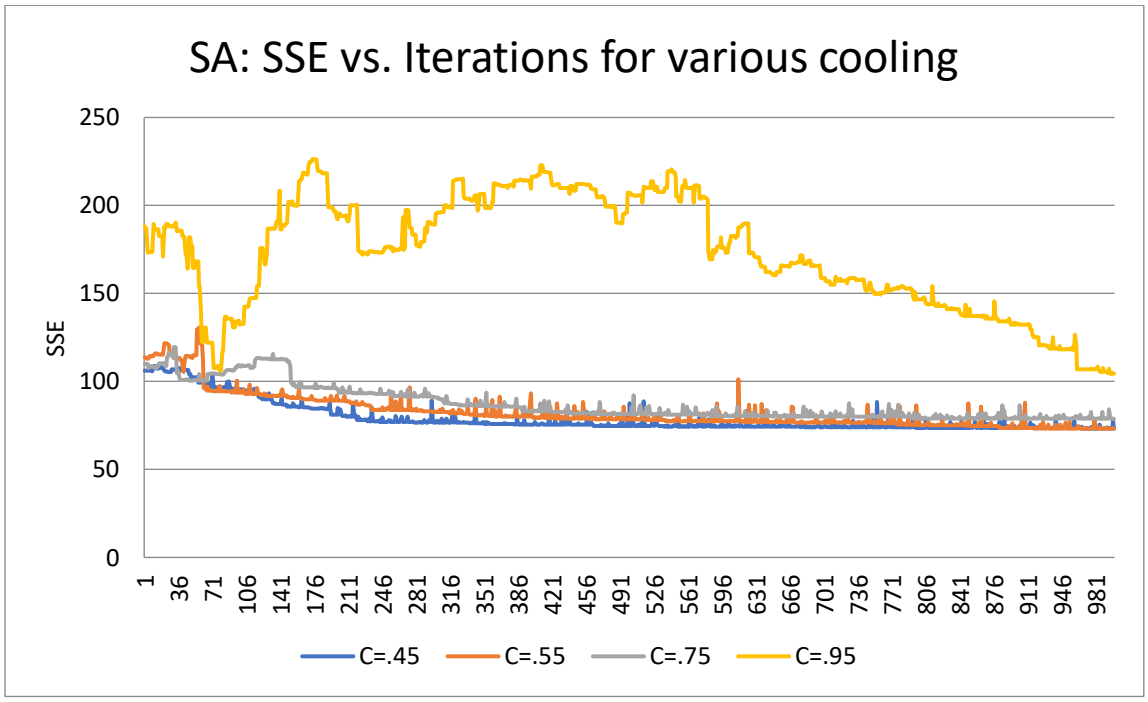
Notably, the graph plotting SSE vs. Iterations for this can be holistically seen as monotonically decreasing, although there are some exceptions to this general observation such as the constant spikes in error, as well as a considerable drop around 600 iterations. It can be equally seen that the rate at which the error decreases is slowing down. The algorithm is choosing points in which the SSE is minimized, because it is working under a set of conditional assumptions such as a Gaussian Distribution and a greedy algorithm. This helps to converge to a specific accuracy higher than the one it started with. Nonetheless, my dataset likely allows for the noise to inhibit performance of this algorithm compared with backpropagation.

Simulated Annealing

This algorithm chooses a random representation to start, but this time has conditions of temperature and cooling rate, which describe the algorithm's volatility with respect to how susceptible it is to perform a random walk and take risks by choosing worse representations locally.

Trial	Cooling Exponent	Training Time (s)	Train Accuracy %
1	.45	3.562	72.266
2	.55	4.184	71.354
3	.75	3.61	68.62
4	.95	4.196	65.234

For Simulated Annealing(SA) weight optimization, unlike RHC, there were in fact hyper parameters that we can see changes in accordance with. To keep some element constant, I set the initial temperature to 1E13, and varied the cooling rate. The training times were on a general uptrend as cooling exponent was increased, which is a result of the temperature being increased at a lower rate, respectively. The relative training accuracies are inversely proportional to the increase in cooling exponent. This is likely due to the fact that, specific to my dataset, the more room allowed for random restarts, there is a certain inflection point in which that is damaging in terms of locating the globally optimal weights for the neural net. The range in my accuracies was not extremely high, yet enough to draw these comparisons (~7%). As compared with the other approaches we've seen, SA has the potential to perform both worse and better than RHC, which is what we expect considering the inflection point from earlier. Even with these subtle crests and troughs, SA performs worse than a neural net with backprop still.



Perhaps the most interesting phenomenon to look at is to consider the SSE for the specific value of .95. The three others, albeit abstracted to a large scale, are relatively equal in their SSE decrease rates with plateaus happening around 200 iterations. The former, however, sees a large dip to match the others, but then corrects to a comparably

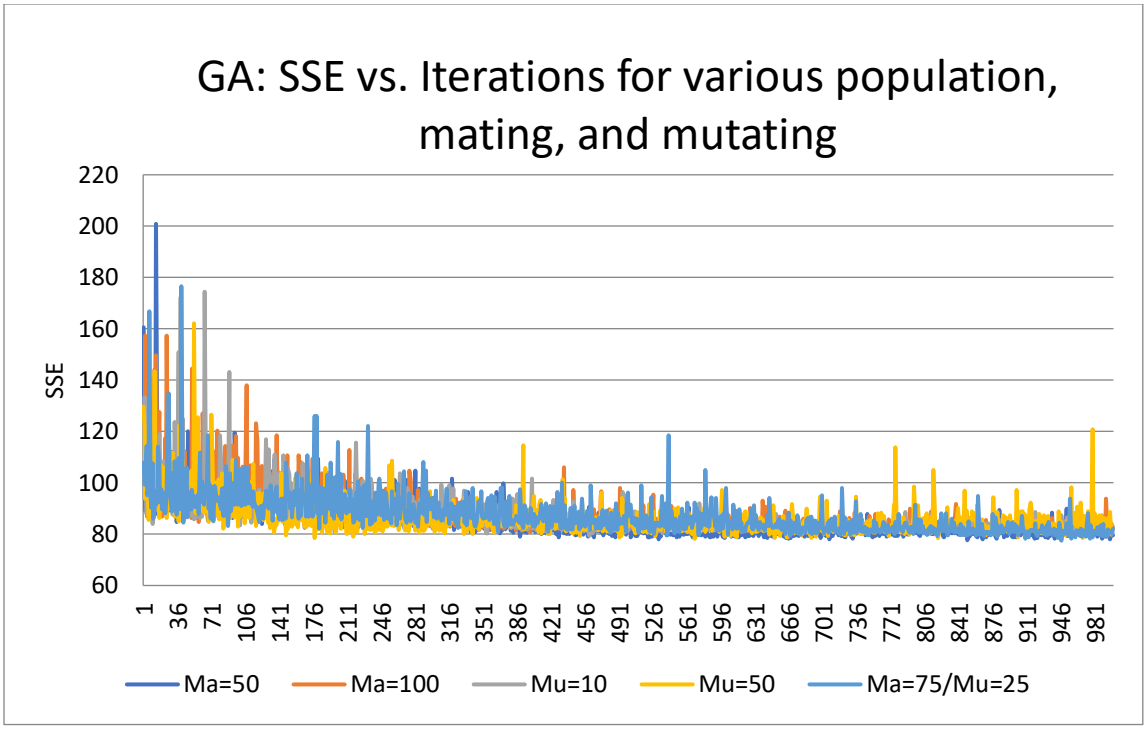
large average which indicates a flaw manifestation in this algorithm which is that too many random walks are assigned that it chooses too many bad representations and gets stuck, which is something to consider when building this up.

Genetic Algorithms

This algorithm chooses a variable population now instead of one individual, and undergoes mating and mutating functions in which individuals are randomly selected to crossover and mutate their characteristics for the nth generation.

Trial	Population Size	Mating Size	Mutation Size	Training Time (s)	Train Accuracy %
1	200	50	10	74.708	67.969
2	200	50	50	132.175	68.099
3	200	100	10	132.175	66.406
4	500	75	25	132.225	68.88

For Genetic Algorithm(GA) weight optimization, there are also a set of parameters to tune and compare in accuracy. I decided on AB Testing, which is tuning parameters one at a time to see specific changes in results. Increasing the mating size (*2) alone, even across trials, seemed to have little effect on the performance, although the training time doubled. One might point out the possibility of these representations being too homogeneous such that crossovers would have minimal effect because no new characteristics are being introduced. Further, the mutation size was equally tuned and saw a slight increase on average in performance. This can be contributed to the idea that there is an optimal level of mutation in which we can see Pareto Optimality take place, but only to a certain point. I decided to compare all of these to a hodge-podge of population, mating, and mutating size and saw little effect. Overall, tweaking these variables yet stagnated the accuracies. As compared to the 3 other weight optimizer's we've discussed, GA actually perform worse on avg. (~68%) than all of them. Sometimes, the dataset is a simple binary classification that putting too much variance into has a negative effect on generalization, which is most likely what we see here.



As we just discussed above with this intra-algorithmic comparison being very subtle, we would only expect the SSE over iterations graph to reflect the same notion. No matter what parameters we choose, the overall shape and kurtosis of the different plots is very similar.

A final comparison

In order to solidify what we are actually getting after with these different implementations, it's worth circling back to the fact that we are trying to optimize weights in a neural network, while comparing to a conventional baseline of backprop. Looking in aggregate form now, we see that the general accuracy path is Backprop > SA > RHC > GA. Having had some experience with implementing these algorithms in the past, I can say that this path is something of an anomaly, but it's very rich in that it tells us how randomized these optimization techniques really are. The diabetes dataset is a binary classification with a small-medium amount of instances. As shown, there are cases in which random walks or a certain amount of mutations across a population in fact take too much of a risk to where the algorithm ends up underperforming, which is exactly what we're seeing here.

**** All 3 of these techniques were run on 1000 iterations, statically**

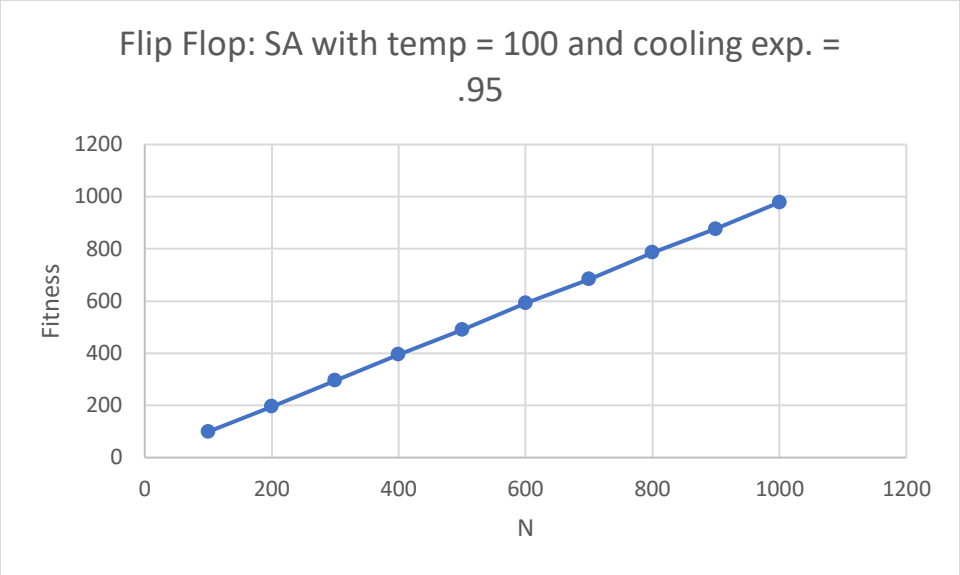
The Problems You Give Us

Flip Flop Problem- Simulated Annealing Optimization

The Flip Flop Test is an evaluation functions that gives the number of times bits within a bit string alternate, starting from the first bit. For example, the bit string “1100” would give a number of 2, while “1010” returns 4. The goal would be to have an alternating value equal to the length of the string, so every value would alternate from its neighbor. This test has a considerable amount of local maxima.

Trial	N	RHC Time (s)	RHC Fitness	SA Time (s)	SA Fitness	GA Time (s)	GA Fitness	MIMIC Time (s)	MIMIC Fitness
1	10	.0745	7	.2354	9	.1755	9	4.3312	9
2	50	.1224	39	.3381	49	.5832	46	21.8454	47
3	100	.1853	79	.3260	99	1.003	87	43.1690	93
4	200	.1965	164	.3932	199	1.2641	160	84.2278	189
5	1000	.8316	833	.5562	977	1.7752	593	226.1800	804

All 4 of these algorithms were tested with a bit string of length 10 – 1000 (N), while the number of iterations for RHC and SA were 20,000 and GA/MIMC were 1,000 for expense normalization. With so small of bit strings such as the first two, the SA’s dominance is not established because there are not enough values to work with yet. However, as we approach higher string lengths, SA starts to perform the best by a considerable margin. This is explainable by the fact that SA shows power in non-continuous with close margin between neighboring representations and local heights. One might argue that this set of conditions would not discount RHC from equally performing; however, this argument can be discredited with respect to the algorithm not making locally bad decisions, such as changing a bit string that would perform worse, allowing it to get stuck. Given the susceptibility to terminate in these plateaus for RHC as well as genetic algorithms, the relative temperature of SA allows it to avoid this same problem. The MIMIC algorithms took the longest time ~by a lot~ to run, and even with the time taken to converge on values performed decently on the global scale. SA was essentially exploiting the literal maximum that can be achieved given various lengths, and doing so in very short training times, making it hands down the best approach for this test. SA > MIMIC > RHC ~= GA.



I thought it would be worth comparing relative Fitness evaluations on the algorithm that beat out the others with respect to this problem to see how to the size of the bit string affects the performance. Turns out there is an extremely monotonically increasing function with a slope of ~1. This shows, given that all other parameters are static, how robust the SA is to the model and the 1:1 x→y ratio shows how perfect of a representation it is for the specific classification.

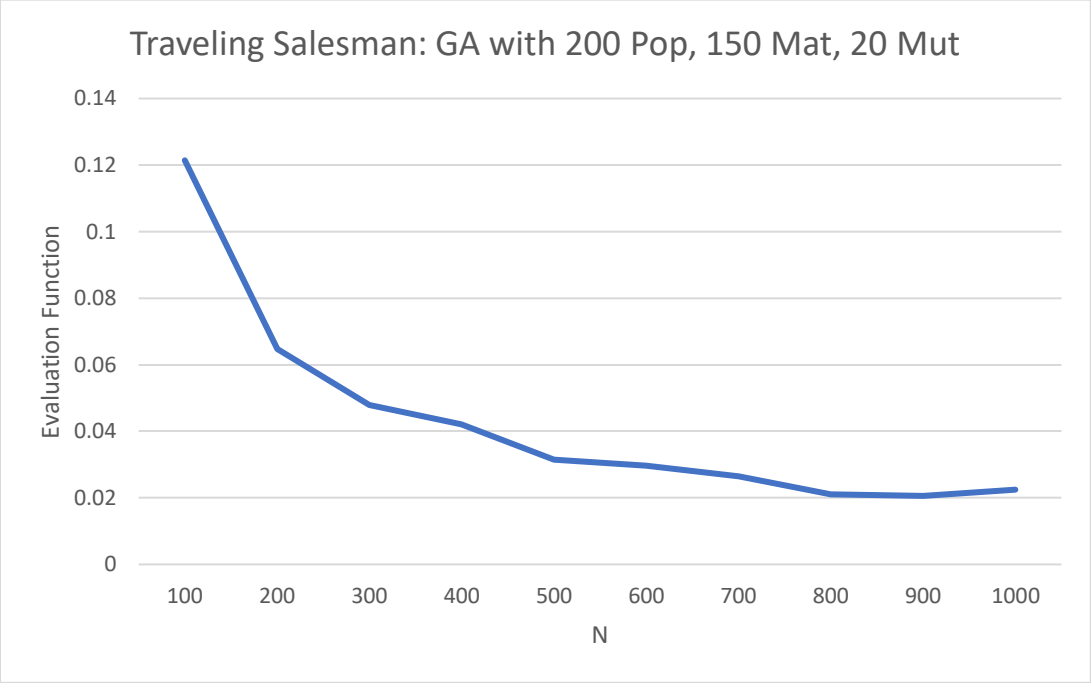
Traveling Salesman Problem- Genetic Algorithm Optimization

The Traveling Salesman Test is a problem in which one seeks to find the minimal distance to circumnavigate every city in a graph with N amount of cities while maintaining the property of making full circle back to one's original starting place.

Trial	N	RHC Time (s)	RHC Fitness	SA Time (s)	SA Fitness	GA Time (s)	GA Fitness	MIMIC Time (s)	MIMIC Fitness
1	25	.0466	.17687	.3375	.16917	.7848	.19821	3.864	.16275
2	50	.0985	.11275	.4509	.12536	1.2326	.15445	5.993	.08332
3	75	.1139	.09689	.6764	.08638	1.9964	.12482	9.612	.05696
4	100	.1357	.06995	.8869	.07678	2.3518	.10651	12.606	.03646

All 4 of these algorithms were tested with a city population of 25 – 100 (N), while the number of iterations for RHC and SA were 20,000 and GA/MIMC were 1,000 for expense normalization. As a premise to the numbers comparison, this task is a very a complex one seeing as the hypothesis space increases more than monotonically with increasing N (HS = ((N-1)!) / 2), so we can already theorize based on what we know about each algorithm that GA's would perform the best. Not only do they handle complex HS's better, even so they generate a POPULATION, rather than one individual,

which is ideal for a path-targeting problem such as this one in narrowing hypothesis space. Since the superiority of GA's becomes apparent with more complex hypothesis spaces, this is why we see the order in which the algorithm does better is increasing through N's. One can see the other algorithms struggling to solve the task, as RHC and SA see 60% and 55% drops across the range, respectively and MIMIC a whopping 78%, while the rate of decline in GA is around 47%. While GA's took more time, marginally, than the RHC's and SA's, the tradeoff to how well it can deal with computational complexity is much more rewarding. Through this competitive analysis, it becomes clear that Genetic Algorithms trump all else in this test. GA > RHC ~= SA > MIMIC.



I thought it would be worth comparing relative Fitness evaluations on the algorithm that beat out the others with respect to this problem to see how the size of the city population affects performance. The path could potentially be described as logarithmic from the surface shape, showing that the evaluation becomes significantly more complex (too much for the best algo. to handle) but then levels off to converge about ¾ down the gamut of populations. This shows how computationally cumbersome it is to narrow down the path of least distance given the increasing hypothesis space. What's interesting to me is how steeply it drops off in the beginning, perhaps coincidentally (RO) or due to variables outside of the scope.

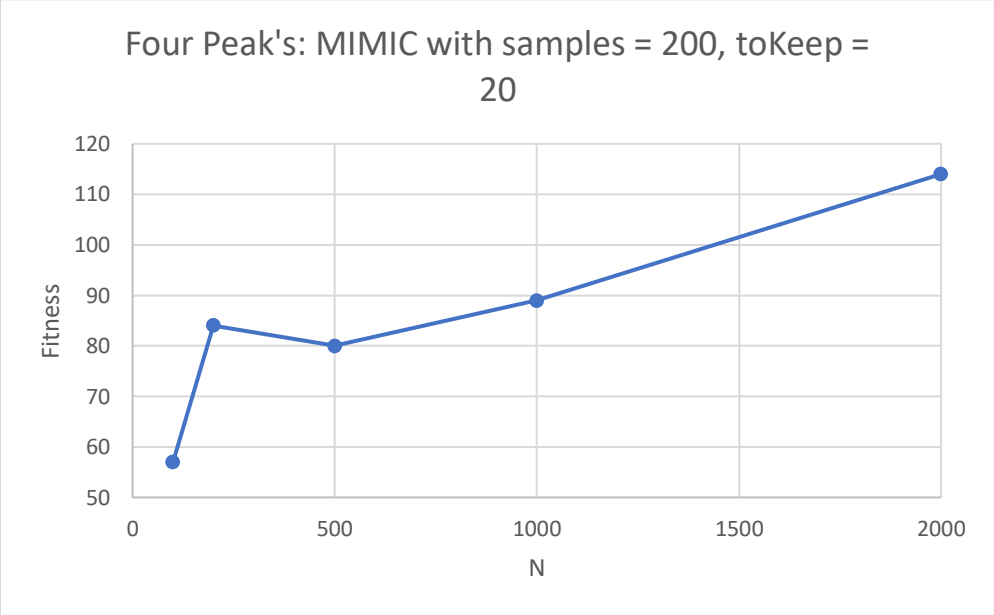
Four Peak's Problem- MIMIC Optimization

The Four Peak's Test is another problem with a bit string optimization underpinning like Flip Flop. Instead, however, what this does with N is takes the bit string length and evaluates the function on having continuous 0's or 1's up to this threshold T, followed by

the unchosen binary value from the latter trailing up to the Nth value of the string. The larger T becomes, the more prevalent the resulting issue of local optima becomes.

Trial	N	T	RHC Fitness	SA Fitness	GA Fitness	MIMIC Fitness
1	100	25	7	6	18	72
2	100	50	8	5	31	67
3	200	50	7	2	20	66
4	200	100	14	3	17	68

I decided to run multiple values of the bit string length (N) as well as the threshold value (T) with another AB Testing approach. Starting with N = 100 and increasing T, there is not much to comment on RHC and SA at this point except for their quite unfortunate performance, and MIMIC absolutely blows any other function out of the water. As we mobbing on to changing N while keeping T constant, we don't see much fluctuation across the board, but what we can notice that is exceptional about MIMIC is its level of resistance to variance is very robust. The algorithm doesn't start poorly, and from there it never falters. It even performs well under the circumstance of a very complex function to classify (Trial 4), where, oddly enough, RHC sees an interesting spike in fitness at this point. I surmise that with the increasing complexity but not an impossible one an exhaustive approach lends itself to finding a better local optima. It is increasingly clear through these 4 trials that the advantages of MIMIC are manifested in this problem. The reason for this is that the algorithm starts with a probability density distribution of the most likely optimal locations on the gamut. An advantage that should be highlighted with MIMIC is its ability to converge with the least amount of iterations, because the way that probabilities in this approach increase is they compound through those iterations.



I thought it would be worth comparing relative Fitness evaluations on the algorithm that beat out the others with respect to this problem to see how the size of the bit string affects performance. What we see, now, is a spike in the original increase of N followed by a pretty constant linear slope upwards. Something to keep in mind is the fact that the fitness value is still with respect to the original length of the string, so naturally we would expect this value to increase with an increased hypothesis space. As far as if the rate at which it is increasing is optimal or not, I would suspect that we would want to see a higher rate with such drastic changes to the N value, but that isn't the case. One could point to the notion that since it takes so long to converge in the first place there is a certain point at which that becomes more of a background variable that affect performance, and the algorithms plateaus. Nevertheless, it performed well.

**** All 3 tests were run with optimal hyper parameters explored in part 1 of this assignment, for all algorithms**

Conclusion

What have we learned about applying different algorithms to these test sets? The low-level nature of the alg's becomes more apparent than ever, such as RHC becoming stuck in minima, MIMIC taking unprecedented times to crunch probability densities, GA's mutation and mating functions producing unforeseeable populations, and SA taking risks that reward fruitfully long term. These are important things to note when implementing these algorithms to anywhere from simple binary with a few instances to multi-label classification with millions.