

## Randomized Optimization Report

### ***Datasets Description***

Sentiment Labelled Sentences – A dataset with 3,000 examples of online reviews (taken from amazon, imdb and yelp). Each example includes a binary label indicating whether the review is positive or negative. To use this example, we had to create a Bag of Words, and use the presence in a certain sentence as the features of that sentence. To perform well in this classification problem, the algorithm must learn the association that some words have with either positive or negative sentiments.

### ***Foreword***

#### *On the Choice of Network Shape*

In the first assignment, I had determined that any number of hidden layers in a neural network trained with BackPropagation was able to achieve extremely good and generalizable results when classifying sentence sentiment. This is partly due to the simple nature of the sentiment dataset. In that assignment, I stuck to 2 hidden layers, with a fair number of hidden units (number of inputs / 2 and number of inputs / 4).

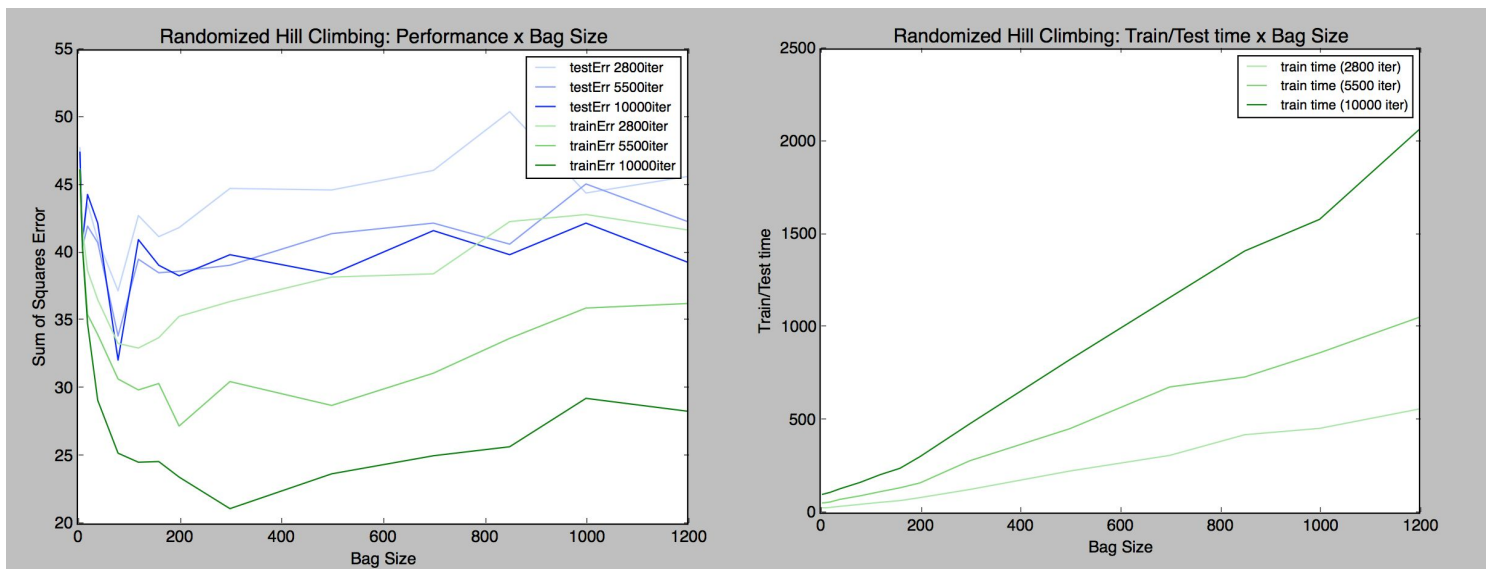
For this paper, however, I determined that a smaller, simpler neural network would be our best shot at achieving good results. The main reason for this is that a fewer number of hidden nodes/layers (i.e.: fewer degrees of freedom) also make the search space smaller, which makes it feasible for randomized optimization to find an optima in a relatively small number of iterations. Thus, the shape chosen was one with 1 hidden layer, containing a fixed number of 10 hidden units.

#### *On the Size of Bag of Words*

As we've seen in the first assignment, a bigger bag of words tend to perform generally better, if we're learning with BackPropagation. However, because I had already changed network shape, I decided to run a sanity check experiment to make sure that we were choosing a reasonable bag size for this assignment. To do this, I ran Randomized Hill Climbing, with varying bag size.

The first thing we notice from the graphs (on the next page) is that, as we increase the number of optimization iterations, we get NNs that perform better on both the training and test set. The converse of that is the fact that fewer training iterations tend to do very poorly, especially with bigger bag of words. This is due to the curse of dimensionality: with big bag of words, the RHC algorithm requires much more training (and training data) to learn which features matter and

which don't. For this reason, we weren't able to match the 2% test error that backpropagation achieved in the first paper.

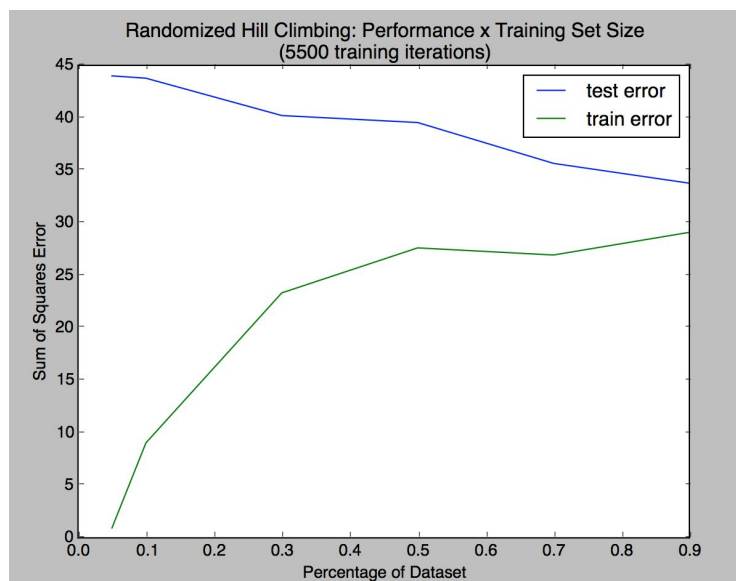


To achieve an accuracy comparable to that, we'd need both bigger bag of words, and a much bigger number of iterations. The reader will notice, however, that training time increases linearly with both bag of words and training iterations, which makes our job of achieving low accuracies very time consuming. Ultimately, what really matters is consistency, so that we're able to compare the different algorithms in this paper. For these reasons, I decided to use a smaller bag of words (80) and a reasonable number of iterations (~6000).

## Algorithm Runs

### Randomized Hill Climbing

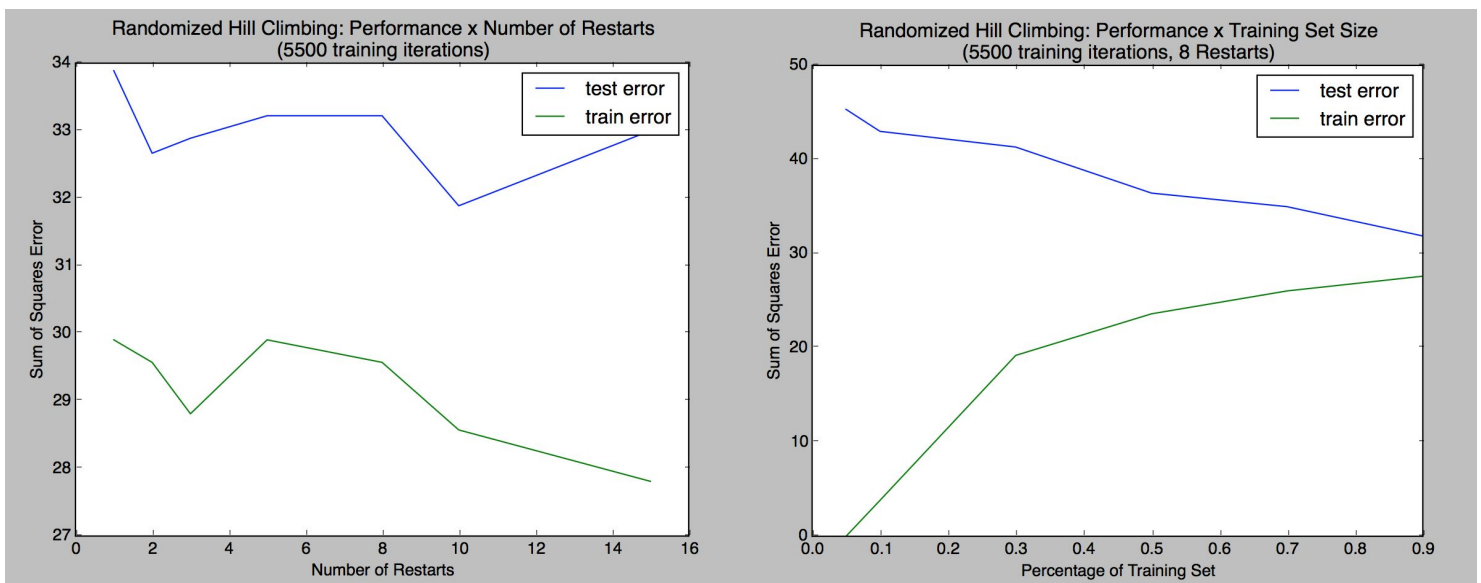
After having determined a good Neural Net shape, and a bag size that would allow us to compare the algorithms' performances, I ran multiple experiments, with varying training set percentages. The idea behind these was to see how well each algorithm performed with different quantities of data. The first experiment run was for simple Randomized Hill Climbing.



It is worthy of mention the discrepancy observed in train vs test set accuracy, when small amounts of training data is used.

But the first noticeable thing in the above graph is the aforementioned very low accuracy. RHC was only able to classify the test set with 64% accuracy at best. This is most likely because RHC was able to find some local optima that wasn't able to correctly classify both datasets as accurately as desirable. One of the reasons such an optima was found is the fact that RHC starts in a random configuration and only explores that neighborhood.

To decrease the chances of getting stuck in local optima, and to explore more of the hypothesis space, we can Randomly restart RHC.

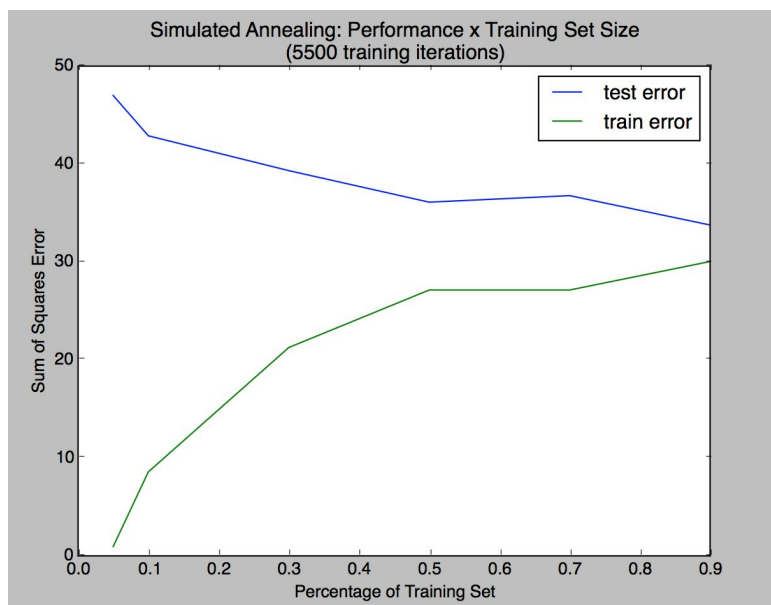


As can be observed in the leftmost graph, randomly restarting gives us much better results. In fact, any number of restarts above 4 made it possible for RHC to consistently achieve results equivalent or better than what it got in the previous experiment. With more and more restarts, the algorithm was able to find an optima that is more likely to be the global optima (because it explores more of the search space). In the rightmost graph, I used 8 restarts to show how much better RHC can do, achieving as low as 31% error, which is a big improvement over the previous experiment.

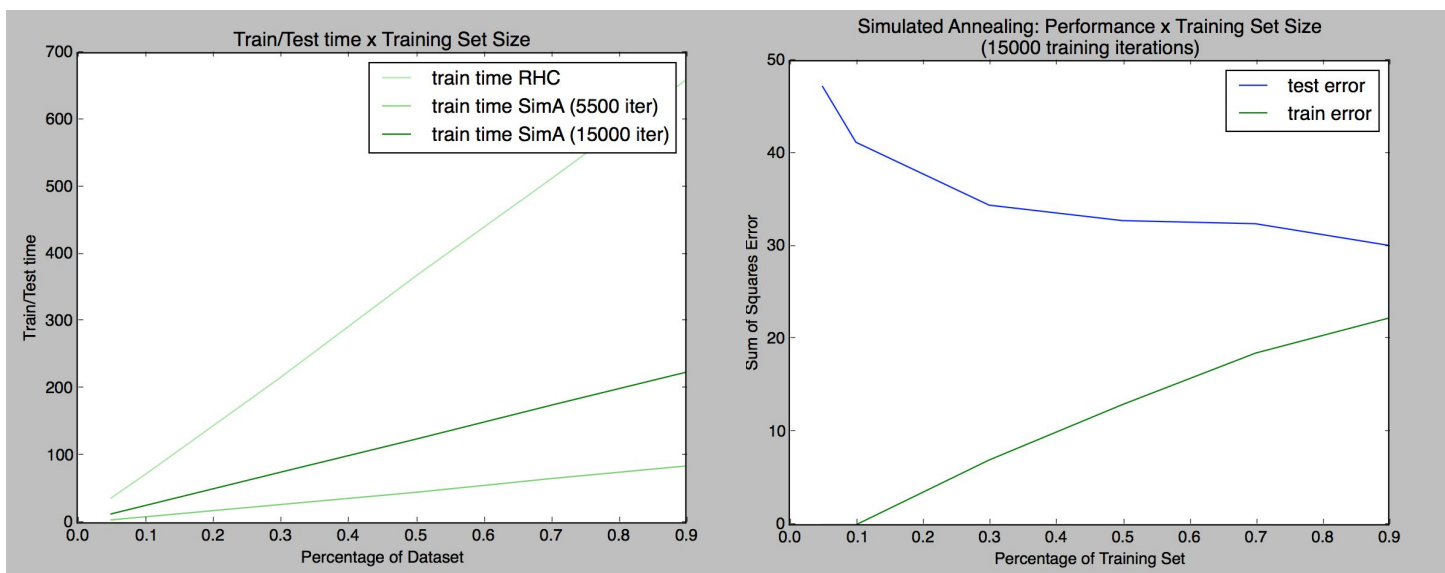
But if the problem of getting stuck in local optima is solved by exploring the search space even further, there is one more thing we can do to improve our results: use Simulated Annealing.

### Simulated Annealing

The first experiment run for Simulated Annealing was of performance as a function of training set size. With this experiment, we are surprised to see that Simulated Annealing performed only comparably as well as randomly restarting RHC: it was able to achieve  $< 33\%$  error.

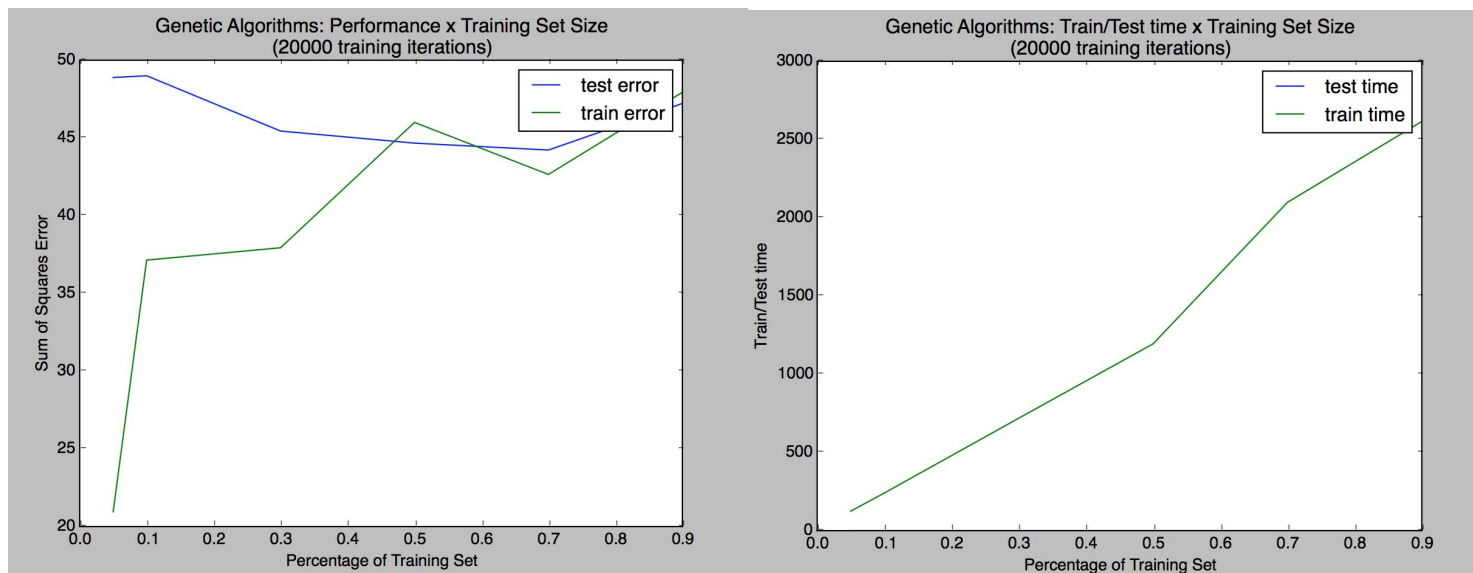


However, when we analyze the two algorithms further, we see that SimA took a very small fraction of the time that RR-RHC took to train (graph below, to the left). Because of this, we can afford many more training iterations of SimA, which makes it able to achieve comparable results to RR-RHC with a much smaller fraction of the dataset, and get much accuracies better ( $< 30\%$  error on the test set) than RR-RHC overall (graph below, to the right). The main reason for this is because, with more iterations, SimA is able to explore the *entire* search space much more, while RR-RHC only explores certain neighborhoods.



## Genetic Algorithms

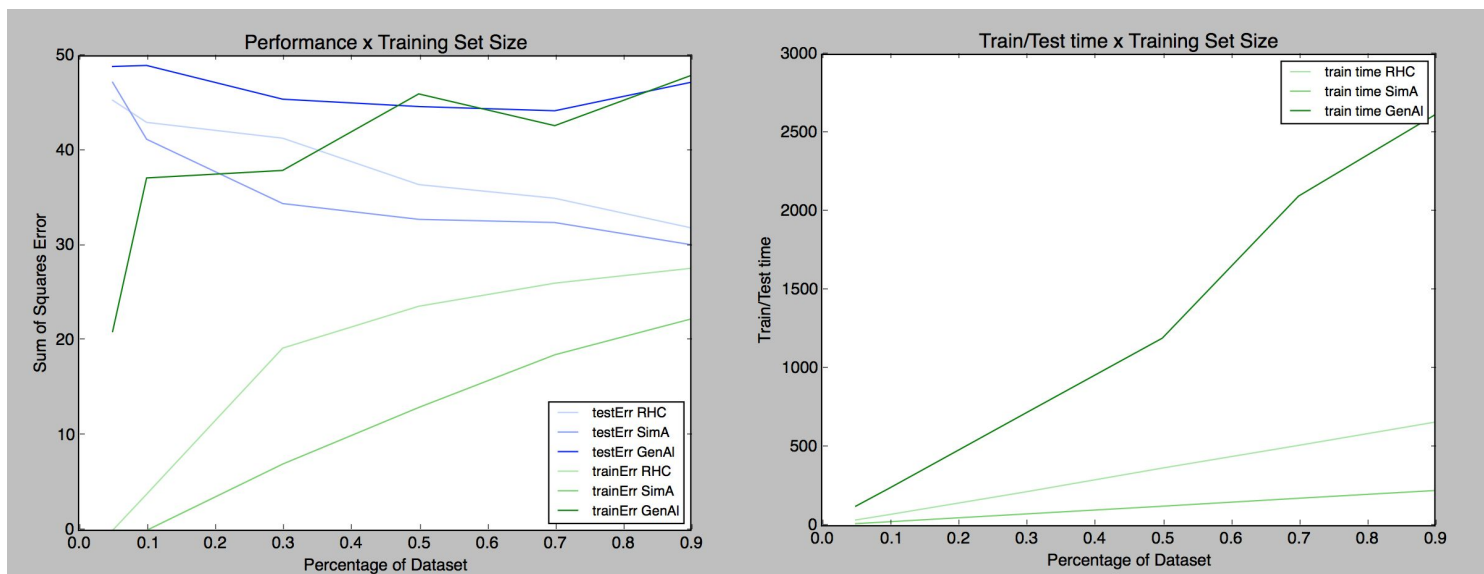
The last experiment run was for Genetic Algorithms. Surprisingly, Despite the theoretical strengths of it, it performed very poorly when compared to Simulated Annealing or Randomized Hill Climbing. One explanation as to why that is happening is that the sentiment dataset simply isn't well suited for genetic algorithms. This could be because the problem isn't well structured enough to take advantage of GenAI's crossover.



The second thing that's noticeable is that time was a huge constraint for running GenAI experiments. This algorithm took about an order of magnitude longer than even RHC. I would like to have run more experiments with varying population size, number to mate and to mutate. I believe that by doing so I could have found better parameters that maybe would have achieved better results. But unfortunately, these were made impossible by the long running time.

## **Comparison of Algorithms**

Here, I present a small comparison of the algorithms seen so far, according to the results we obtained.



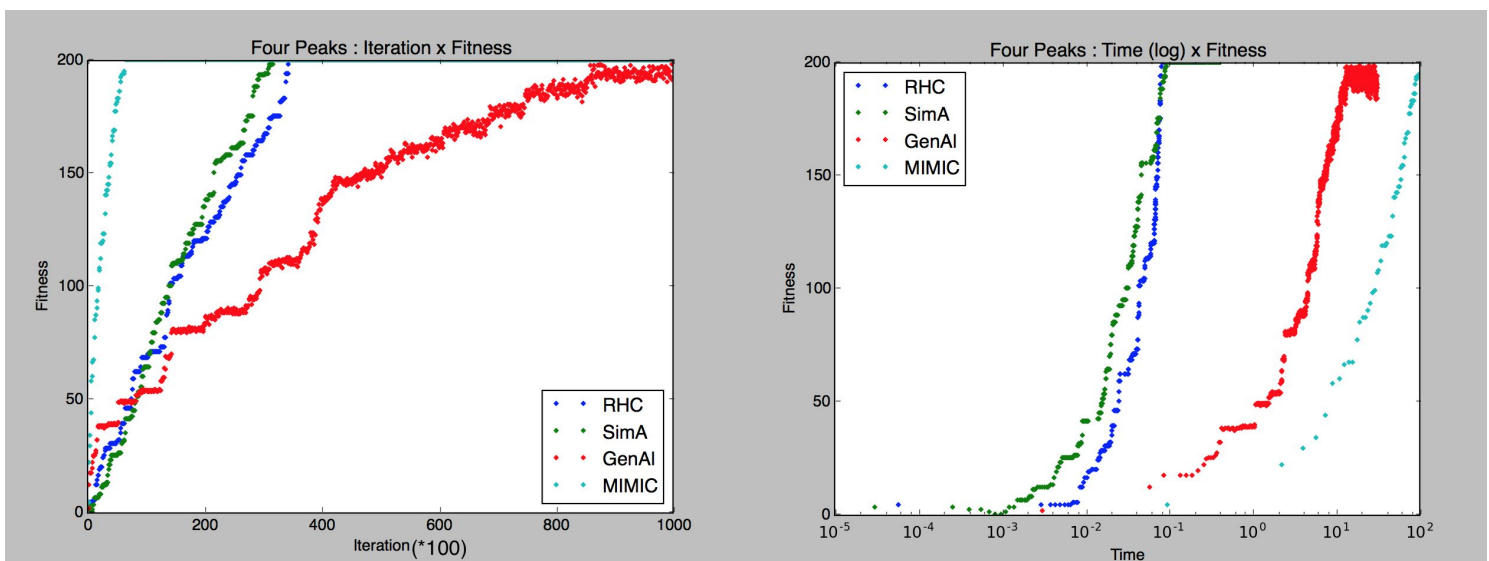
As is clear by the graphs above, SimA performed the best out of all algorithms, achieving lower error rates quicker than any other algorithms. In fact, the time plot clearly shows SimA's performance advantage over RHC and GenAI. It's worth mentioning, once again, that the theoretical performance of GenAI is better than the results we obtained. With further testing, maybe its accuracy could have been so much better that the longer training time wouldn't have been a problem.

Finally, the other hypothesis I can come up with to explain the low accuracy of GenAI, is the definition of the problem space. GenAI works best when the problem is well structured, meaning each neighbor only modifies the "fitness" by a small amount. This way GenAI can take advantage of crossover to improve itself. I believe our neural network may not be an example of a well structured problem, because changing one of the weights can have drastic impacts in the output we observe.

## ***Interesting Problems***

### ***FourPeaks***

This is an optimization problem that tries to find the n-th dimensional vector  $X$  that maximizes the FourPeaksEvaluation function. This function has two global maxima, and two local maxima.



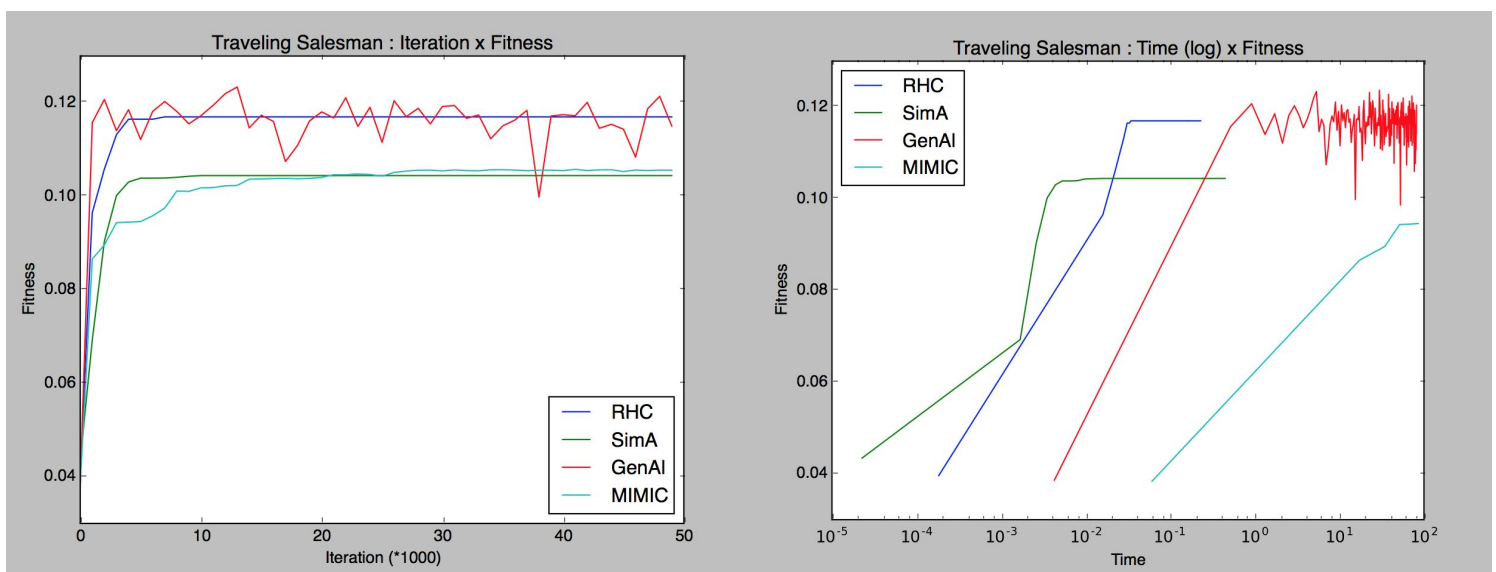
The two graphs above show the value of the FourPeaksEvaluation over iteration, and time. The first thing that is noticeable is the low number of iterations that MIMIC required to find the global optima, when compared to the other algorithms. RHC and SimA performed well with respect to each other, and genetic algorithms took the most number of iterations (and wasn't able to find the perfect optima at all, despite getting very close to it).

However, when we look closely at the Fitness vs.  $\log(\text{Time})$  graph, we see that, despite needing only a small number of iterations, MIMIC took the longest time (by a few orders of magnitude) to find the optima, a result similar to GenAI. Unsurprisingly, SimA seemed to perform the best,

finding better results sooner than the other algorithms. But its results were not dramatically different from RHC's, as they were in the NeuralNet optimization problem. I can come up with one reason for this observation: the big leverage SimA has is the fact that it searches much more of the search space than RHC does. But if the fitness function isn't too complex (i.e.: there aren't too many local optima), then this extra search over different neighborhoods isn't required. As such, RHC is able to find the global optima quickly and perform really well when compared to SimA.

### Travelling Salesman

This optimization problem concerns with finding the best route a travelling salesman would take, to go through multiple cities. In this case, "best" means least distance travelled. The interesting thing about this problem is that it can present many good solutions, that are still not the optimal.

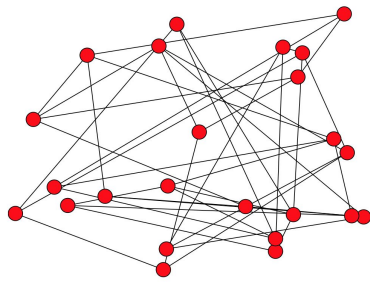


From the graphs above, we can see SimA and MIMIC found at least one such sub-optimal solution, while GenAI and RHC found a much better solution. This result relates closely to our previous discussion about GenAI: the travelling salesman problem is one "well structured" one, so GenAI was able to take full advantage of crossover and find an optimal solution. In fact, it did so in fewer iterations than RHC, but because GenAI is basically parallelized, crossed over RHC, it took much longer to run (as the rightmost graph shows).

### Pretty Graphs

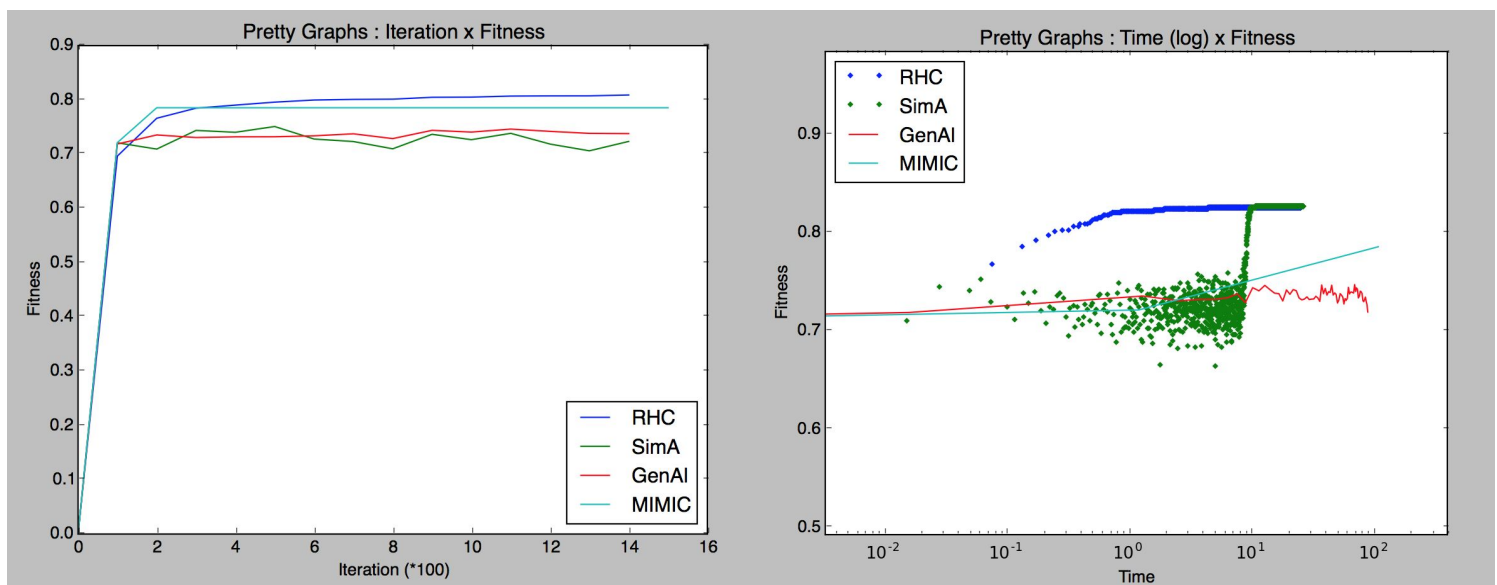
This is a problem I developed a couple semesters ago, while at The Agency (GT's AI Club). Me and Jessica Rosenfield worked on porting it to a version that worked with ABAGAIL for this paper. The problem is simple: given a graph (a list of nodes and their edges), find the positions of each node that makes the graph the prettiest, where "prettiest" is defined in our evaluation function. It consists of a combination of multiple heuristics:





Area = minimize area occupied by the graph  
 EdgeVariance = make edges about the same size  
 EdgeDistribution = distribute a node's edge evenly around it  
 Intersect = minimize edge intersections  
 EdgeLength = tend edges towards a specified desired length  
 Overlap = minimize node overlap with other nodes and edges

The resulting score is a number from 0 to 1, that shows how well the graph fits the (manually weighted) heuristics. The image above shows the graph we used in the experiments (a “web” graph, with 24 nodes), in a randomized configuration.



The graphs above show a comparison of each algorithm’s performance when dealing with the PrettyGraphs problem. It is clear that all of them performed very well, taking a very small number of iterations to reach a good result. Interestingly, longer runs of RHC and SimA kept fine-tuning the graph they obtained, and were able to obtain even better scores.

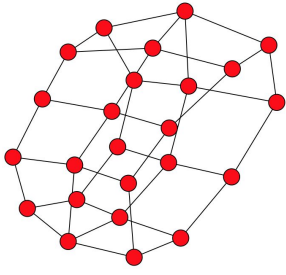
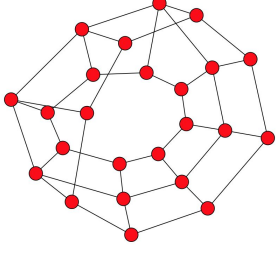
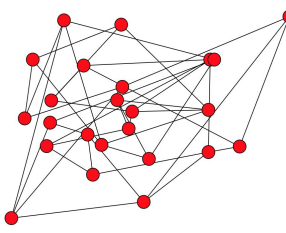
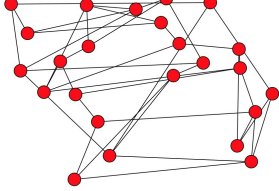
Even more interesting is the comparison between SimA and RHC: SimA took longer to find great scores, most likely because it spent a longer time searching over the entirety of the search space. But when it does find a good score, it scored better than RHC, which shows how SimA probably found a better optima than RHC

Once again, we see that this problem, being well-structured, allowed GenAI to show its potential, even though its results were not as great as the other algorithms. Perhaps the most interesting result in this experiment was that of MIMIC. MIMIC performs extremely well in situations when the cost/score function is computationally expensive, because it learns the most of all algorithms per iteration (which is the reason it has always needed only a small number of iterations to find the optima).



The PrettyGraphs evaluation function is not trivial. It required a lot of computation to get a single score. So we see that MIMIC, for the first time in our experiments, took no more than three orders of magnitude longer than the other algorithms to find a good results. It still took longer to find such an optima, but the fact that the time gap was much smaller shows that MIMIC can be very useful in certain scenarios.

As a last addendum, I wanted to present the resulting graphs obtained by the different algorithms, along with the scores they achieved.

	<b>RHC</b> area: 0.1869 / 0.2 edgeVarScore: 0.6943 / 0.7 edgeDistScore: 0.0523 / 1.0 intersectScore: 1.676 / 1.7 edgeLenScore: 0.2502 / 0.3 overlapScore: 1.999 / 2.0		<b>SimA</b> area: 0.1849 / 0.2 edgeVarScore: 0.6901 / 0.7 edgeDistScore: 0.0546 / 1.0 intersectScore: 1.686 / 1.7 edgeLenScore: 0.2532 / 0.3 overlapScore: 1.999 / 2.0
	<b>GenAI</b> area: 0.1838 / 0.2 edgeVarScore: 0.6316 / 0.7 edgeDistScore: 0.0140 / 1.0 intersectScore: 1.540 / 1.7 edgeLenScore: 0.2091 / 0.3 overlapScore: 1.929 / 2.0		<b>MIMIC</b> area: 0.1855 / 0.2 edgeVarScore: 0.6358 / 0.7 edgeDistScore: 0.0166 / 1.0 intersectScore: 1.612 / 1.7 edgeLenScore: 0.2169 / 0.3 overlapScore: 1.978 / 2.0

## Conclusion

As mentioned in the beginning of this paper, our randomized optimization algorithms performed rather poorly on the sentiment dataset when compared to BackProp (the results of which we've seen in the previous assignment). Here, I offer one reason as to why that might be.

BackProp is propagation and correction of errors, which means it knows what to modify in order to achieve better results over time. But these random optimization algorithms are just that: random. They try things out and stick to what works. This means they can take much longer to achieve good results. The number of iterations used wasn't enough to achieve the extremely high accuracy that backprop did.

However, as previously mentioned, our goal was to provide a comparative analysis of the RandOpt algorithms, not to achieve extremely high accuracies. I hope this paper was able to do that by being consistent with the choice of hyperparameters, and putting the results in context, relative to the other algorithms.