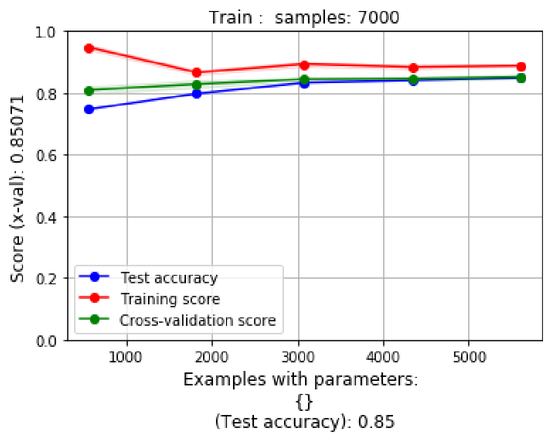
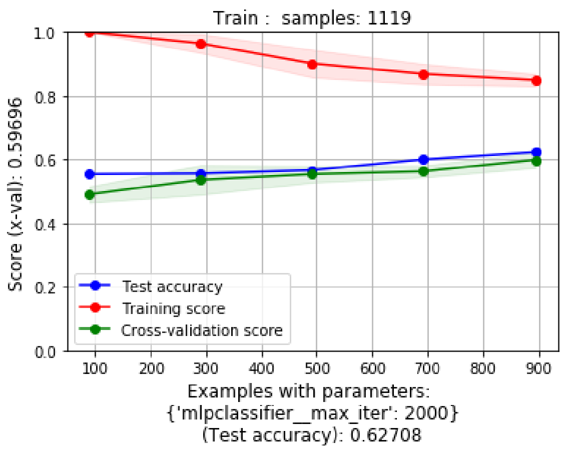
Randomized Learning Algorithms

Part 1

Problem Description

The problem set chosen from the first assignment for this assignment's part one is the TV commercial set where still images from television are classified as being either a commercial or not a commercial. This set was chosen for a few reasons. The commercial set contains twelve attributes each with one or more columns upon that feature (e.g. the motion distribution attribute contains multiple columns of continuous values between 0 and 1). Unlike the wine quality dataset from the first assignment, columns within the same attribute should show some correlation between each other. Additionally from working with this dataset with such a high dimensionality of column space (122 inputs) the neural network doesn't suffer as much from local optima as much as the wine set with only 12 attributes (and only 1 column per attribute) when using a neural network since if a particular weight is stuck in a local optima, the odds are that the rest of the weights in the network are not stuck in local optima and changes to those weights will help out any particular weights stuck in a local optima. This problem is also a binary classification problem which is useful when dealing with the continuous domain. Outputs will need to be encoded to reflect possible outputs. This task can be done in as little as one node. Technically multiple outputs can exist in [0,1] (e.g. (0.1,0.3,0.5,0.7,0.9) for 5 features) but there are more boundaries where false classification can occur. Based on advice from the Tom Mitchell ML book, thresholds were changed from their default values {-1,1} that were generally useful for comparing different algorithms in assignment one to {0.1, 0.9} for assignment two. The movement to positive space was more a result from slightly better performance. More importantly though is the choice of 0.1 instead of 0 and 0.9 instead of 1. This is due to the sigmoid activation function being unable to actually reach 0 or 1 and again for performance reasons, the target values were brought in slightly. The decision boundary still remains at 0.5 for classification.





Baseline performance is shown above for the problem sets for assignment one with the commercial set on the left and the wine set on the right for neural networks.

Part 1 Problem Setup

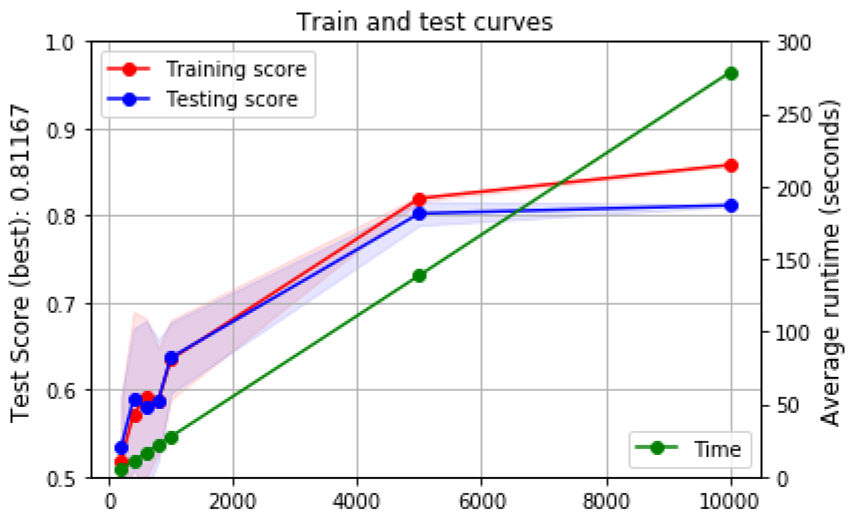
Most of the methodology for assignment one carries over into assignment two. A brief overview though is included. There are five sources for examples (5 different television networks). Each source is then processed to pull out 1000 positive examples and 1000 negative examples. Within each set of 1000 examples, 700 are chosen for the training set and 300 are chosen for the testing set. Combined, this yields 7000 total positive and negative examples for training and 3000 positive and negative examples for testing.

As the five files themselves are too large for submission, indices are instead stored to reference lines within the files of the desired examples and the files downloaded separately. With preset indices, less biased comparisons can be made between the two assignments as the same underlying data is ensured. If the above procedure were followed but with varying indices, that would just be another variable in attempting to make any conclusions.

Data is normalized as is a crucial step for neural networks. All these processing steps do occur every time a new function is run. While ABIGAIL is used instead of scikit-learn in python which has a persistent kernel, Java load times are only a few seconds and are very reasonable. Also, although ABIGAIL generated the results in this paper, some of the graphs used python files and matplotlib to generate consistent looking graphs as those from the first assignment.

On a last note it is important to point out that the bag of words attribute was not used for any neural networks. This one attribute contained ~4000 columns and made some of the randomized algorithms like simulated annealing impossible to search across a space since even after huge iteration counts, the odds that an individual weight would be visited even once was small. This analysis will be further explained later. As for comparing to the first assignment neural network, there was a section in that assignment specifically for comparing the network with and without the bag of words and it was observed as only a 3% drop in accuracy from 88% to 85% (shown in left graph above) when using neural networks. The neural networks in either assignments consist of 122 inputs units, 100 hidden units, and one output unit.

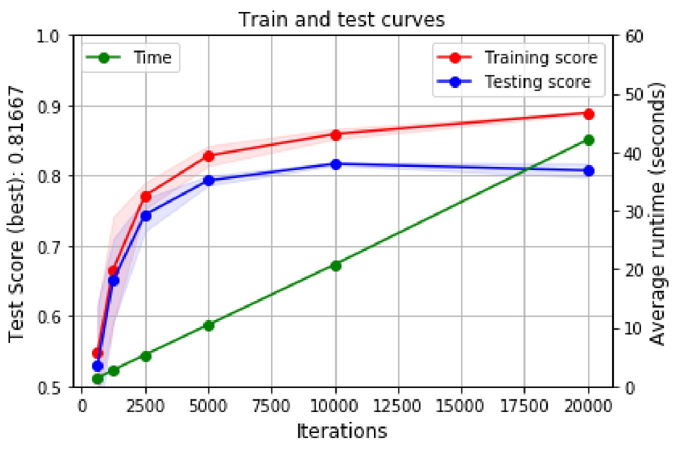
Random Hill Climbing



Random hill climbing takes many attempts to find good optima. From the training set, approximately around 5000 are needed before overfitting begins to occur (which can be seen at the 10,000 mark). Accuracy for random hill climbing should be near that of the neural network with enough attempts made at climbing. Tom Mitchell's book suggests that although there are plenty of local minima for a particular weight in a network, the odds are that most weights are not in a local minima and altering other weights (random, gradient, etc...) can unstick weights suffering from being in a local minima. Runtimes are also down, although comparing against different environments and languages (Java vs. Python) time to train decreased from 247 seconds with back propagation to the times shown above in a feed forward network.

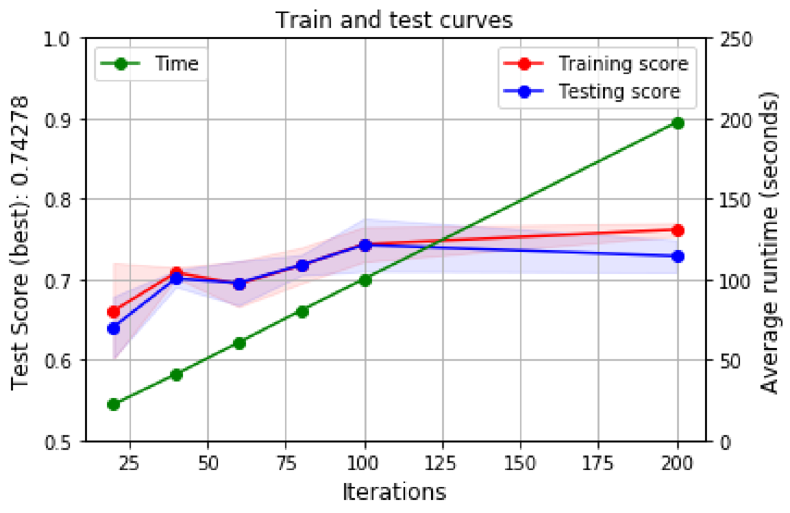
It is not surprising that a large number were required due to the neighborhood to search through. Random hill climbing has to choose a neighbor to modify and check for performance. The space to choose from is the weights that express the network. This comes out close to the number of input nodes multiplied by the number of hidden units and lastly the output units added in or about 122 x 100 + 100 = 12300 weights to look at. Most weights will not even be looked at if only 5000 are chosen and some of those are likely to be duplicates. Of the 5000 viewed, there is probably around a 50% chance that a movement up or down will even improve the network so an assumption can be made that ~2500 weight changes will be made from an initially randomly instantiated neural network which is approximately what was observed though more often weights were adjusted earlier in the lifetime of the random hill climbing search. The results seem to make sense in that some features when present don't depend on the present of other features in the graph for proper classification and such weights between layers could be removed. A complete bipartite graph that a neural network starts as (inputs mapped to hidden layer) is almost assuredly overkill for expressiveness. As an area for improvement for performance on this dataset and possibly for ANY hill climbing in general would be to apply the modified version of Kruskal's minimum spanning tree algorithm to start with a fully bipartite graph and (randomly) remove edges of the graph until it becomes too hard to remove edges without sacrificing the accuracy of the test scores. It would be expected that in such removals that accuracy would go up initially as during retraining, unrelated features wouldn't be offering input to the model. I have similar thoughts on such a process for other uses of neural networks including simulated annealing to be discussed later.

Simulated Annealing



Simulated annealing was the most difficult to extract good results for of the randomized algorithms for the neural network comparison. With default settings in ABIGAIL of 1Exp11 starting temperature and a 0.95 decrease per iteration in temperature. Even for a very high iteration count, simulated annealing was often returning training and testing accuracies around random (50% for binary) and additionally with a high degree of variance with results as low as ~30% to as high as 70%. Tuning of the cooling factor and temperature didn't have much of an impact on the results. What did however have an impact was changing the underlying model to have fewer hidden units. Recall that the model was 122 input, 100 hidden, 1 output. A change was made to use only ten hidden units to increase the odds that any particular weight in the network would be visited. The results can be seen in the above graph with much better results than 50% and high variance. Results converged very closely to what random hill climbing managed as well as the back propagated neural network. Results did converge much faster though than either of the former. For the ~5000 mark, only a smidgen over 10 seconds were needed which is greater than an order of magnitude improvement over random. While random hill climbing was not affected by the large neighbor search space, simulated annealing was likely affected due to its design in that it needs to be able to move downhill at high temperatures. Quite possibly a downhill movement of a weight could be the only time that that weight was visited causing the algorithm to take a metaphorical eternity to start to climb. And by the time simulated annealing began to climb, the temperature would be very low. Looking at the console print statements, this would take a temperature of ~1E-8 before climbing would really start with the default 100 hidden units. The reduction to 10 increased the odds of a weight's adjustment by a factor of 10 which is believed to have helped simulated annealing produce any worthwhile results. Yet even with a much smaller network size, scores for movement coming from (1/ (sum squared error)) only made significant improvements once temperatures had cooled significantly. This could suggest that hill climbing was a much more valid use of iterations rather than getting out of nearly nonexistent local minima for a weight that wouldn't likely be visited all that often. It also may not be obvious from the runtimes and graphs but the times expressed were for one run of an algorithm instead of total for a particular count and there are faint red and blue regions marking the score deviations which give a sense on the variance. 3-5 runs were used based on how many attempts were made. Also for both random hill climbing and simulated annealing, changes to weights were in the range of +- (0.5).

Genetic Algorithm

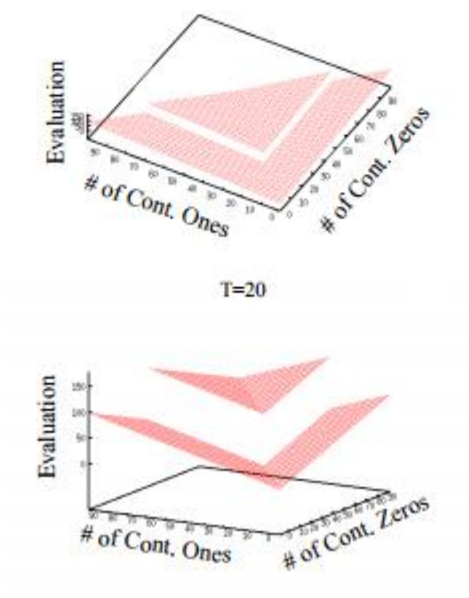


Genetic algorithms performed much better at low iteration counts than either random hill climbing or simulated annealing. Using the standard 100 hidden units, genetic algorithms took the longest time per iteration. Genetic algorithms behave different than the earlier mentioned algorithms in that the changes to weights occur among the most fit candidates to breed. Of the population per iteration, 20% were chosen to breed and 5% were chosen to mutate. Increases in mutation rate allowed similar test scores but with higher variance in scores even after convergence. Above about a 5% range can be seen but with a mutation rate of ~20% had a range of ~10% on accuracy. Mutation rate didn't help the search converge that much faster. Breeding rate had more of an impact on converging faster to higher test scores. Breeding allows for large changes in a member of a population which is probably a reason why so much change can be seen between iterations compared to RHC and SA but also explain the significant increase in runtime per iteration. The process of breeding also needs some explanation here since the weights involved are continuous. While generally thought of as an affine combination, literature has referred to this crossover as Haupt's method. The idea is that no particular value is chosen to carry over to a child but rather a weighted combination of both parents. With x = random(0,1), (1-x) weight comes from parent A and x from parent B. This gives the result of interpolating between the weights of both parents A and B. Important to note is that the genetic approach never quite reaches the 0.8+ accuracy seen in RNC, or SA. This can be explained by the fact that the algorithm isn't designed as climbing a hill explicitly and will only indirectly climb any hills if not completely jump off them to another (I suppose not terribly different than SA). Since weights can be crossed over in large blocks (1 split method), the hope is that two parents each have a better fraction (not necessarily a 50/50 split though) of some amount than the other parent to contribute to the child. The combination of the best of two parents is what allows genetic algorithms to potentially converge faster even if it means ignoring local search like RHC and SA (SA can jump from its current region but only to those nearby).

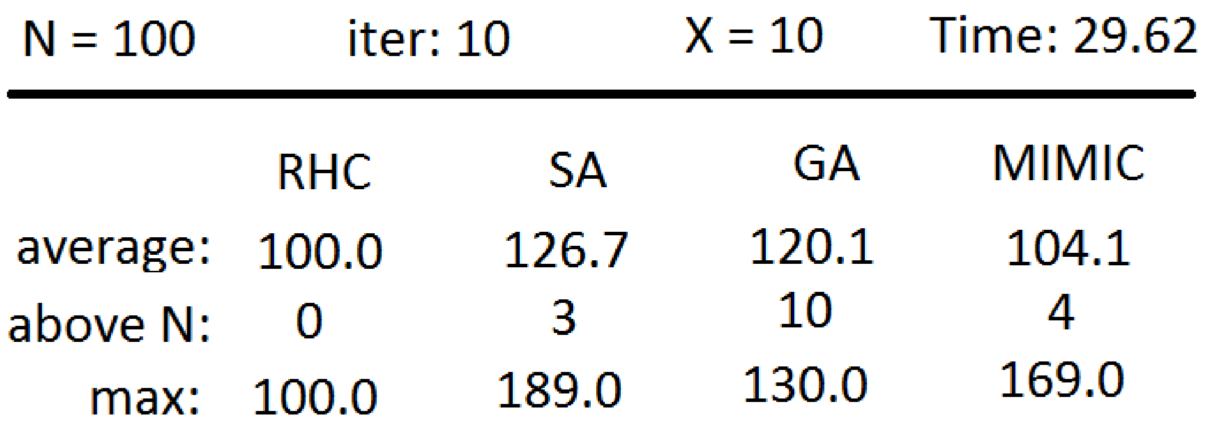
Part 2

Simulated Annealing

Speaking of moving between nearby local optima, this is an area that SA excels at especially on steep or even discontinuous terrain. Enter the problem of the 4-peaks. Those familiar with the 4-peaks problem remember that on a bitstring, score is given based on lengths of consecutive bits both at the beginning of the string as well as the end of the string. This scoring method is proportional so far in that its surface should be relatively smooth (It's still discontinuous as scores are discrete but still small on the order of a value of 1). However making this problem more interesting is that there are thresholds given that will boost the score of a bitstring by a score equal to its total length if enough bits are consecutive at both the beginning and end of the string. For example, A bitstring of length 100 will score 1 point for each consecutive 1 from the starting position and 1 point for each last trailing 0 at the end of the string. This could give a possible score of 100. But another 100 points can be earned if at least the first and last X bits are of a certain length perhaps 10. This causes a discontinuity jump of 100 or double the original possible score. Also, the max possible score is 2*(length) - X - 1 as the problem is defined.



Seen right is a rough visualization of possible scores for the 4-peaks problem. The scores are discrete and there is no surface between adjacent points but height differences of 1. The mentioned jump in height by a score of 100 can be seen between the two "V" shapes of the score space. If you can imagine what type of algorithm might be able to reach the top peaks (there are 2 with a score of 189). Looking at the space naively, one could think that perhaps 1/6 of the space in the top image lies in the space where hill climbing would reach the top. This is of course very misleading as the space visualized on top is ONLY relevant for strings with leading and trailing ones and zeroes with no regard to the middle contents of the bitstring. Thought another way, a starting random bitstring is much more likely to be in the corner of (0 leading, 0 trailing) approximately 50% of the time (based on random initialization). Climbing that space will lead to a lower peak. It would be really unlikely that RHC could get to a point where it's first and last X bits meet the bonus criteria to then start climbing that space.



N = 100 iter: 10 X = 10 Time: 29.62

	RHC	SA	GA	MIMIC
average:	100.0	126.7	120.1	104.1
above N:	0	3	10	4
max:	100.0	189.0	130.0	169.0

Results shown above for 4-peaks. 4-peaks indeed does miss out on the bonus and only scores 100. A row is shown as "above N" which indicates how many runs out of 10 did the algorithm reach the higher space. Lastly there is a row for the max value obtained. Note in the case of Genetic algorithms that it always reached the higher scoring space each of 10 times. However, its max value achieved was only 130. This makes sense as genetic algorithms can make large jumps through search space (not limited to neighbors) and can combine with the best of the populations. The genetic algorithm's ability to max out though at 189 is limited in that it can't actually reach such a condition without pure luck from breeding/mutations since it isn't climbing the score space. Still on any particular run, GA on average will score the highest. Simulated annealing takes the best of both RHC and GA in that it can both climb a space locally as well as take jumps between spaces. SA is typically good for finding "good" optima even if not the best but here it is the only algorithm that consistently finds the best (3 times in this case). This does hold true as iterations increase. However with increases to the consecutive bits X, the likelihood of SA reaching the higher "V" decreases so more iterations will be needed. Still, there is a considerable amount of downhill movement required before reaching the higher "V" (again, the actual space of movement includes lots of interior bits). Looking at logs of print statements when SA earns the bonus, it is often preceded by multiple downhill movements as SA will need to likely trade some of its points it has gotten temporarily for the first/last bits of the bitstring in order to reorganize to achieve the bonus. Once the bonus is achieved, the f(x') and f(x) difference will likely be large and with a cooling temperature, less likely to move down such a huge gap. If the bonus isn't that large (e.g. much less than bitstring length), SA could end up down again but with a large bonus and cooling temperature, SA performs excellently. An improvement here is easy to this particular algorithm due to domain knowledge knowing that this bonus exists. Noticing that GA always is reaching the higher "V". This space can be found and then handed off to SA or RHC to finish the journey up to the max score. This was in fact easy enough to implement and SA for example would always at least reach the global optima at least as often as GA could. Again it can be greater than GA since SA can sometimes find the higher "V" on its own.

Mimic doesn't have as high an average score as SA or GA despite the fact that Mimic often takes many fewer iterations to achieve good scores in arbitrary problems. Mimic tends to score fairly well at its max but still relies on good random number generation to get there. Mimic and GA could potentially score better if there were "better" underlying structure to exploit which will be seen in the next 2 examples.

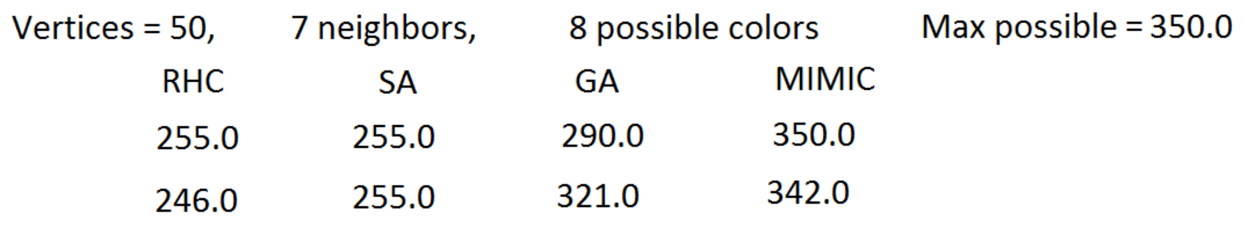
Genetic Algorithms

Genetic Algorithms really shine in problems where it's hard to climb or know when you've reached the top. A good example of this would be the Traveling Salesman problem. The TS problem is an NP-hard problem requiring brute force to prove optimality of a path. Although algorithms like RHC can quickly climb through a space, it is too easy to get stuck on a local optima. Different regions of the space being search are likely to maintain their connections based on the greedy structure of RHC. Two cities that are close together are likely to stay connected. RHC will trade a link between two such cities links between another pair of cities if the combined distance is reduced. However, this is more likely to occur with close by city pairs and such an exchange of links in the path will be harder for cities that are far away. And once RHC gets a good hold onto a hill (at a point where changing any pairs will reduce score) then it will only be able to handle that local optima. While this may be the bane of RHC, this is where genetic algorithms really shines. As mentioned earlier in the 4-peaks problem, genetic algorithms excel in sharing information in farther reaches than other algorithms since it isn't limited to its local domain. You can have a well optimized path in a certain region of a member of the population but lacking somewhat elsewhere. If two members of the population can share their own excellent paths and exchange good city pairs regardless of the pair's absolute distance to their offspring, you can create a better cycle. A possible improvement for handling this problem could be a flavor of RHC And GA. RHC will latch on to local optima quite well, if an algorithm starts off randomly and maxes out RHC, it could switch to GA and potentially reconnect well optimized local regions to achieve a goal of a better path. Mimic performs unusually bad here (multiple runs were performed and the order still stayed the same as reflected below where higher scores are better (SSE can be rederived by taking the reciprocal of the below score values). N = 50 implies the number of cities used. Here higher mutation rates (~15-25%) were successful compared to the neural network for the commercial data. This isn't that hard to see as genetic switches are still strongly random and nudges in the right (or wrong) direction help evolve a lacking group of GA population out of a rut. This is still noticeable later on in the GA's lifetime although there is still a large increase in variance that comes with higher mutation rates. A breeding rate of 20% performed reasonably well although increases did lead to losses possibly because good structure is lost and doesn't get enough time to form for any individual.

	N = 50		
RHC	0.1351772141984194	SA	0.12364980244116114
GA	0.14084753607357572	MIMIC	0.10141687388280057

MIMIC

One test where MIMIC shines is the K-Coloring problem. MIMIC is already structured to perform well in fewer iterations based on its design but with increased cost per iteration with distribution creation and such. In the K-Coloring problem, while various algorithms can accept changes based on improvements in scores they often do so blindly without considering the best change or even where to make such a change now or later. GA attempts to manage some of this structure though allowing crossover but only at random points. Even in successful breeding of two parents, children involved in future breeding can often lose the properties that made its parent's good candidates to begin with. MIMIC's can handle these with its emphasis on structure like GA (which ends up being a close second in scores to come) discussed so far, but with more emphasis on making better than random choices to make decisions. MIMIC creates and uses probability distributions that hopefully will match ideal distributions to maximize the score over generations. The process can be slow per iteration (mutual information is exponential in time to compute) but MIMIC can capture structure to score potentially well.


Vertices = 50, 7 neighbors, 8 possible colors Max possible = 350.0

RHC	SA	GA	MIMIC
255.0	255.0	290.0	350.0
246.0	255.0	321.0	342.0

From the results, MIMIC and GA performed well. Many runs were performed an MIMIC was able to max out its results about 1/2 the time while GA could do so around 1/5 the time for the settings of vertices=50,neighbors=7,colors=8. In some combinations of neighbors and colors, GA actually performed better but in general MIMIC would come out on top and RHC and SA never did. This isn't surprising since those two algorithms don't capture structure in their design.

The scores above are just general representations of how the algorithms performed. Sometimes there were hard graphs generated that none of the algorithms could handle at least on the first try. To see how these algorithms handled much harder graphs, a restart counter was implemented to count how many reattempts it took for GA and MIMIC specifically to converge to the best possible answer. The same <50,7,8> parameters were used and the results were:

	Restarts for GA	Restarts for MIMIC
Hard run 1:	239	0
Hard run 2:	548	7
Hard run 3:	492	1

Scores of course were 350.0 as that was the terminating condition but at least both of these algorithms could eventually reach it. RHC and SA never could max out the score on <50,7,8> even on the non-hard problems where GA and MIMIC got max score on the first try.

Resources:

Abigail repository (original before modification):

https://github.com/pushkar/ABAGAIL

Diagram of space for 4-peaks problem:

https://pdfs.semanticscholar.org/cd4f/e89d8dd6060e2957041f90fc699a30058d01.pdf

Repository of this branch of ABIGAIL (git link):

https://github.com/rabbitweasel/cs7641_p2.git