David Li

Cs 4641, C. Isbell

March 2nd, 2019

Randomized Optimization

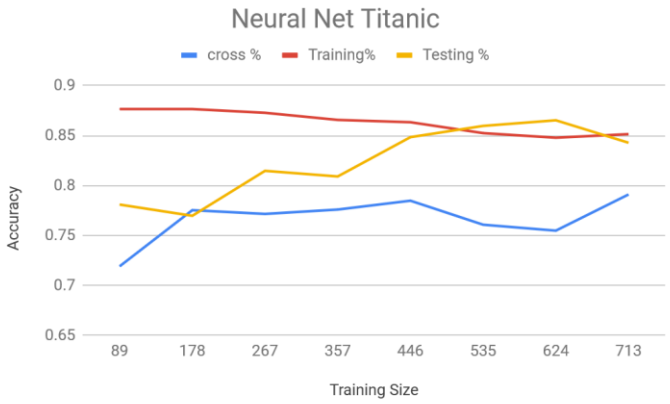
Overview: The following is an analysis on four randomized optimization algorithms: Randomized Hill Climbing, Simulated Annealing, Genetic Algorithm, and MIMIC. The first part of the project takes the data set from the last assignment and use the first three algorithms for the back propagation of neural nets, where we will analyze how each algorithm did and why that happened. Secondly, we will look at three different problems, and see how each of the four algorithms performs based on their strengths and weaknesses.

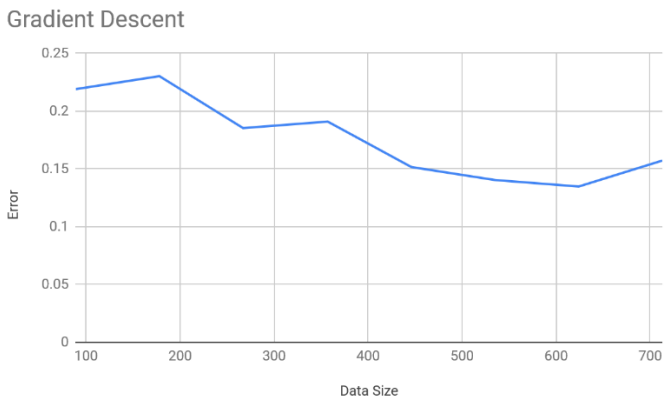
Data: We chose to use the Titanic dataset from the first assignment as it had very little noise based on the supervised learning algorithms performed on it, and did not have a lot of missing data, making it the more comprehensive and complete set of the two. I think the aggressive discretization and missing values on the Adult data set would cause a lot of issues in terms of back propagation as many attributes that were dissimilar or missing might've been grouped into one category.

Part 1

For each of the algorithms, I performed training and testing using different hyperparameters, and graphed their error over 2,000 iterations. Using this we shall analyze how hyperparameters affect each of the algorithms and in the end, we shall analyze how they do compare to each other.

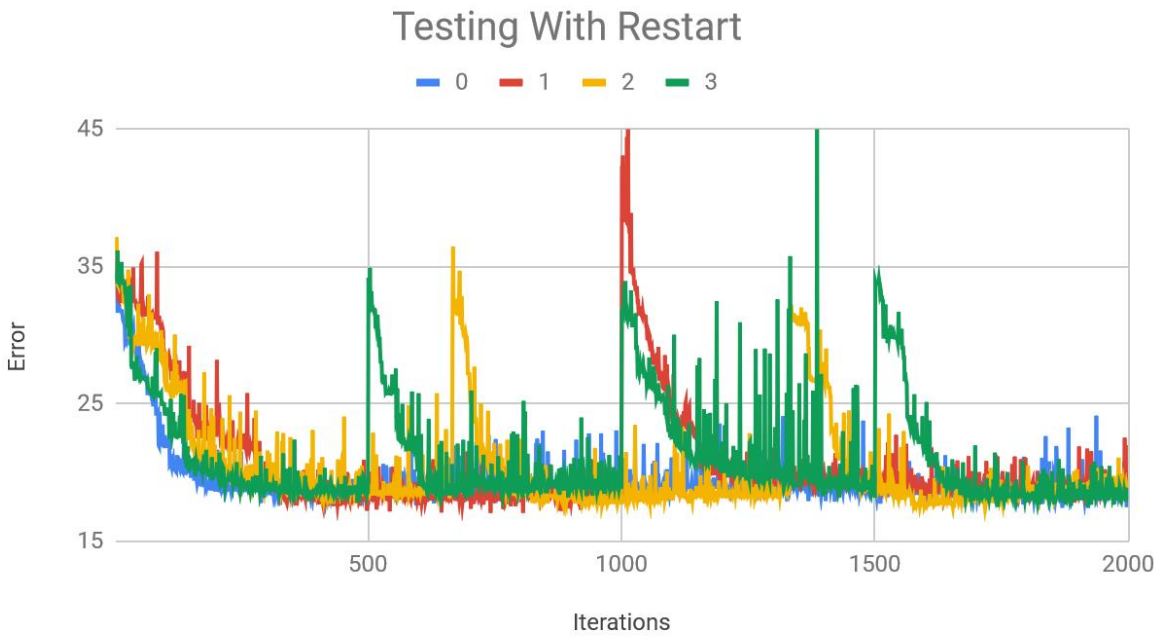
During part 1, with WEKA's neural net using gradient descent in the back propagation. As data size increases the testing accuracy increased, while error decreased. Since during assignment two, we split the data into 70 30, we can see that about 70% of data used, we were at around 14% error, and this will be the baseline we use to compare the other three back propagation algorithms.





Randomized Hill Climbing

For randomized hill climbing, the hyper parameter looked at was the number of restarts during the run, which are spread evenly throughout the iterations.



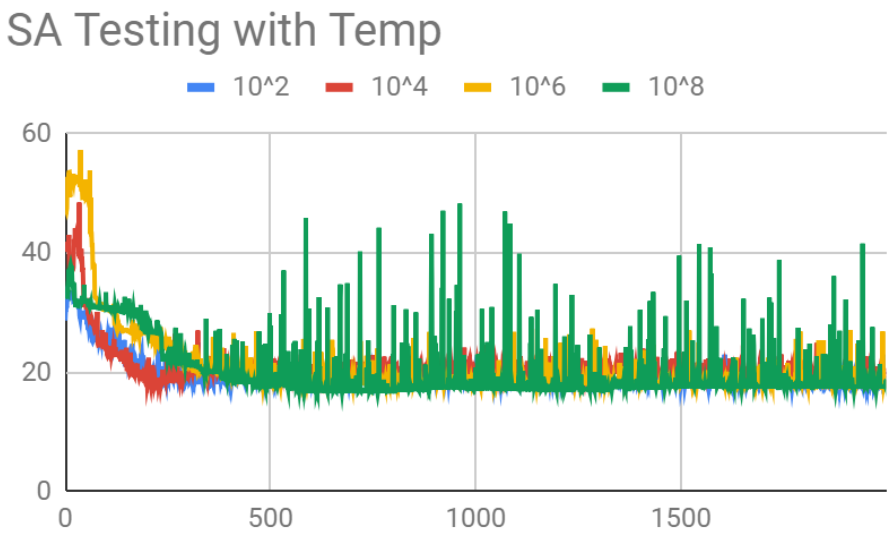
Restart	0	1	2	3
Accuracy	82.09	81.716	80.597	82.09

Although difficult to see here, randomized hill climbing performed the best in testing using either no restarts. Using one restart let to a huge spike during the restart, a phenomenon that is expected, but not to this degree as during restarts for 2 and 3 the error spike was much lower. With 3 restarts, there is a huge spike between the second and third restart, this is likely due to one point or section of the titanic data having some sort of discontinuity or inconsistency, which the algorithm landed upon. In fact, the variance and error within the whole entire section between second and third restart were relatively high, supporting the idea the randomized hill climbing landed in a basin of some sort of a smaller local maxima and while exploring to either left or right it experienced huge drops.

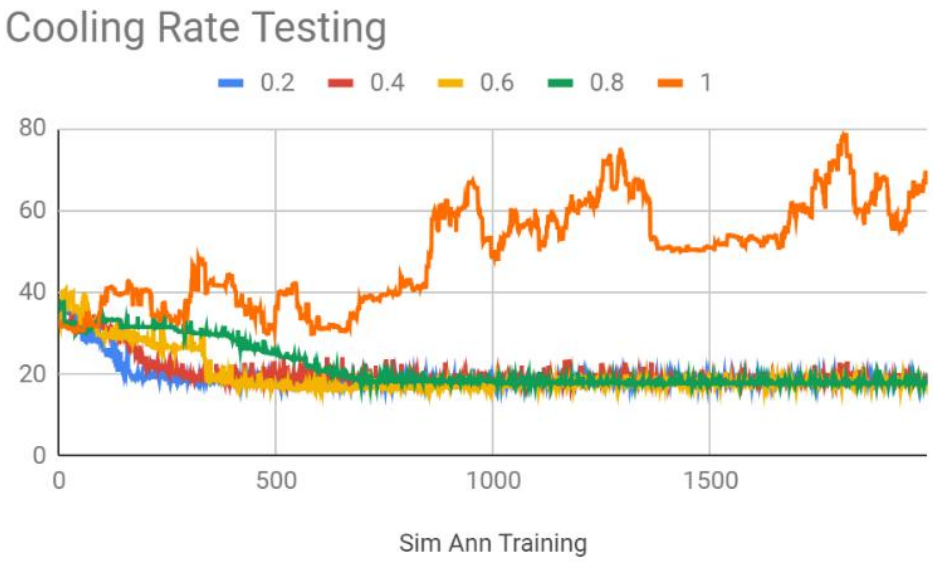
The graph became too complicated when taking other restarts into consideration, but as you can see, there is little benefit to multiple restarts, at least with this data set. No restart, 3, and 5 restarts all performed with the same amount of accuracy, but more restart leads to more variance in the data and a less consistent curve.

Simulated Annealing

For simulated annealing, we looked at both hyperparameters for initial temperature as well as cooling rate, both of which affects how long it takes to reach the optimal point, determining how long it takes to decide on a point and how often it will change its target throughout. With complicated data sets, the random jumping and a lower cooling rate will allow the algorithm to escape local maximas, and a higher temperature to start off with will allow more time to look around. With our simpler data set, however, it is mire important to move towards an optimal solution rather than to focus on escaping, as the maxima found will almost always be the correct one.



Based on the results, all the temperatures had relatively similar results, with a lower temperature doing better. This is probably because of how direct the Titanic data set is that it is very easy to find a optimal point with given data based on weights of attributes, and easy to trace back the influences of different attributes. With a higher temperature, we can see that since it would take more iterations cooling down and at a given iteration, it would jump around a lot, thus the 10^8 temperature had the largest amount of variance, at points many times that of the rest of the 3 temperatures. It can be argued with a simple data set like this, the fact that lower temperature is preferred shows that there are few if any plurality of local maxima's, and that the random switches are unnecessary.

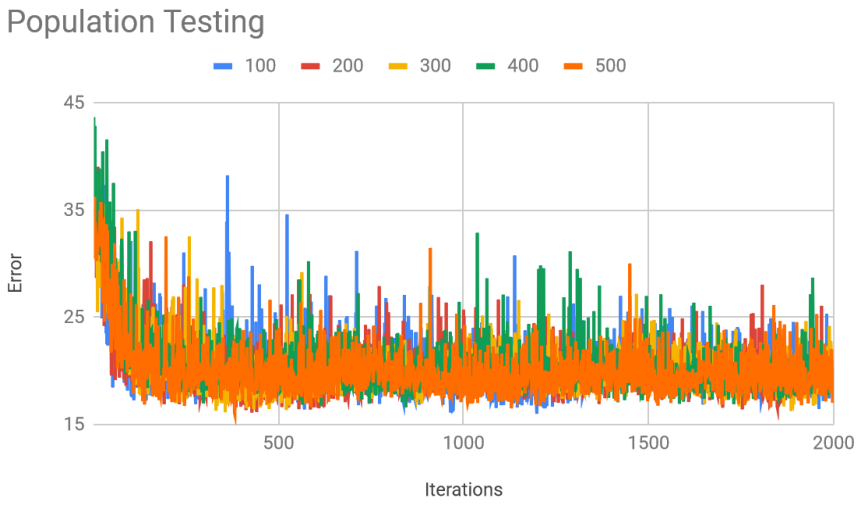


While testing cooling rates as parameters, we found that the lower the cooling rate, the earlier it was able to approach its horizontal asymptote, giving less variance, which furthers the idea that there are few local maxima in the data set, as even with a higher temperature at different iterations, the algorithm still rarely jumped, if ever, and rarely found other maxima by jumping to worse solutions and always converged back to the same one. It makes sense that with a cooling rate of 1, the errors were insanely high and did not decrease as it basically just picked a random point and did not change afterwards as upon landing it picked that point immediately and stuck with it, with no improvement, and it averaged about 50% error on the testing runs.

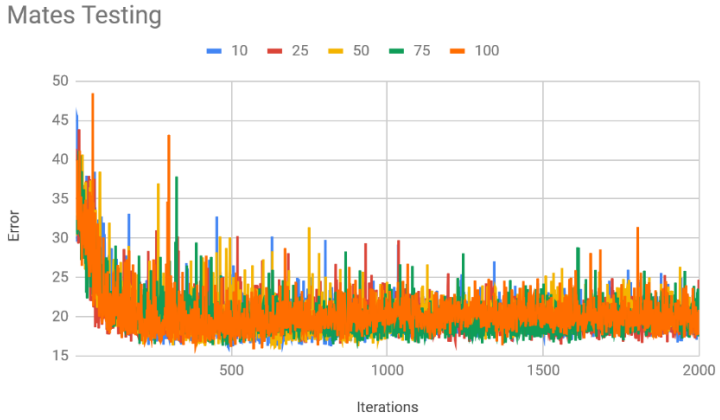
Genetic Algorithm

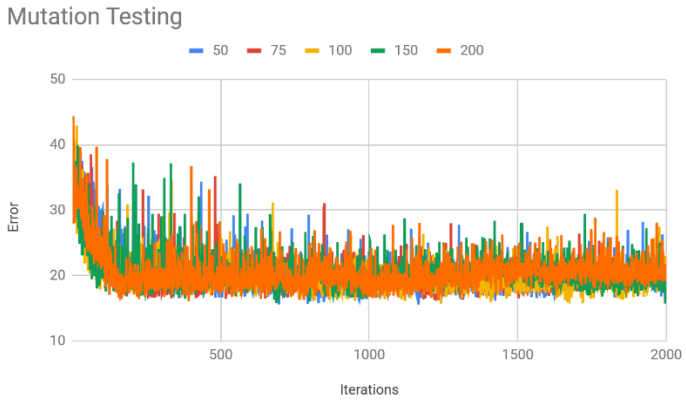
With this algorithm, different variations of the distribution will be created based on successful ones rather than just going based off given distribution like simulated annealing and randomized hill climbing. While with complicated distributions this could find beneficial or more fit populations and distributions, it may create large variances when only using a simple distribution.

First, we looked at the hyper parameter for initial population.



Despite the relatively clumped graph, we can see that with a population of 500 there was very little variance in terms of the training error (and thus accuracy). Since genetic algorithm brings in mutations and cross over, it is important to note that a low variance in testing accuracy does not imply a low variance in distribution, as this algorithm is the first of our analyzed with any distribution variance at all. This makes 500 population the best choice in this case, as over all at any given iterations, it has a lower deviation from the horizontal asymptote and is quickest to reach it, giving it the lowest mean squared error as well. This may be since a higher population will start the algorithm and continue it with more possibilities and individuals to mutate and cross over, but since this is a really simple distribution, almost every mutation and mating result will be getting closer to the answer and more correct ones are made and saved throughout each generation.

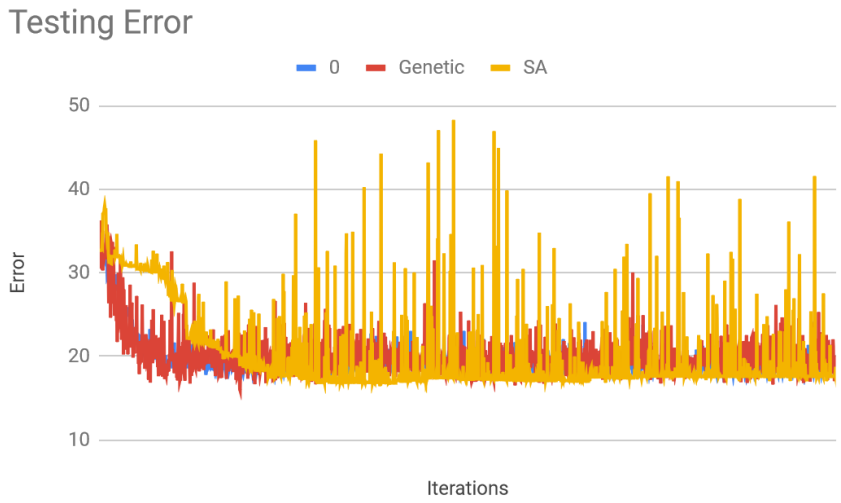




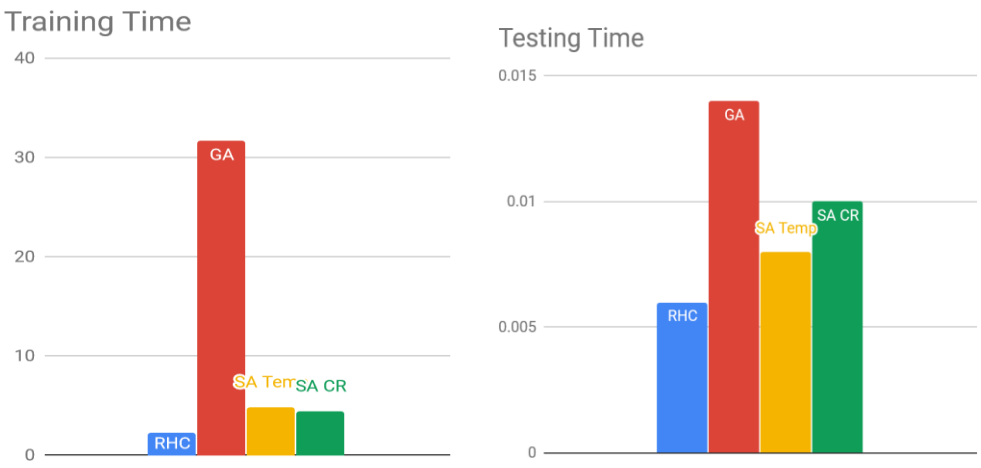
When looking at mating mutation amounts as hyperparameters, there ends up being very little difference between a high and lower rate. The all reach about the same amount of error and accuracy, while doing so at similar rates and variances. It can be argued that at the highest value for both sets, the accuracy was a bit lower. I believe this is again,

since this data set is relatively simple, and that at most levels, mutations and crossing over cannot deviate from the results too much, but at an extreme level, it becomes noticeable. Even then it is barely worse than other levels.

In the end, the most favorable hyperparameters for each are taken, and compared to each other:



It becomes immediately obvious that simulated annealing had an insane amount of variance in terms of error compared to the other two data, followed by genetic algorithm, and finally randomized hill climbing with no restarts. Although some argument could be made that the low variance in error with randomized hill climbing is because of the lack of restarts, since restarts did not benefit the algorithm, following Occam's Razor, we will use the simplest one. Simulated annealing's tendency to jumping around especially at early iterations is what made the error so high, and since this data set is so straight forward, none of the jumps really helped it. Similarly, genetic algorithm's mutations and crossing over brought in variance and increased its mean squared error without really contribution to the accuracy in terms of back propagation and what attributes and weights had the most affect. With RHC, with a relatively trivial dataset and "Correct" weights, the algorithm climbed one of the few, if not only, hill in the distribution and found the optimal solution.



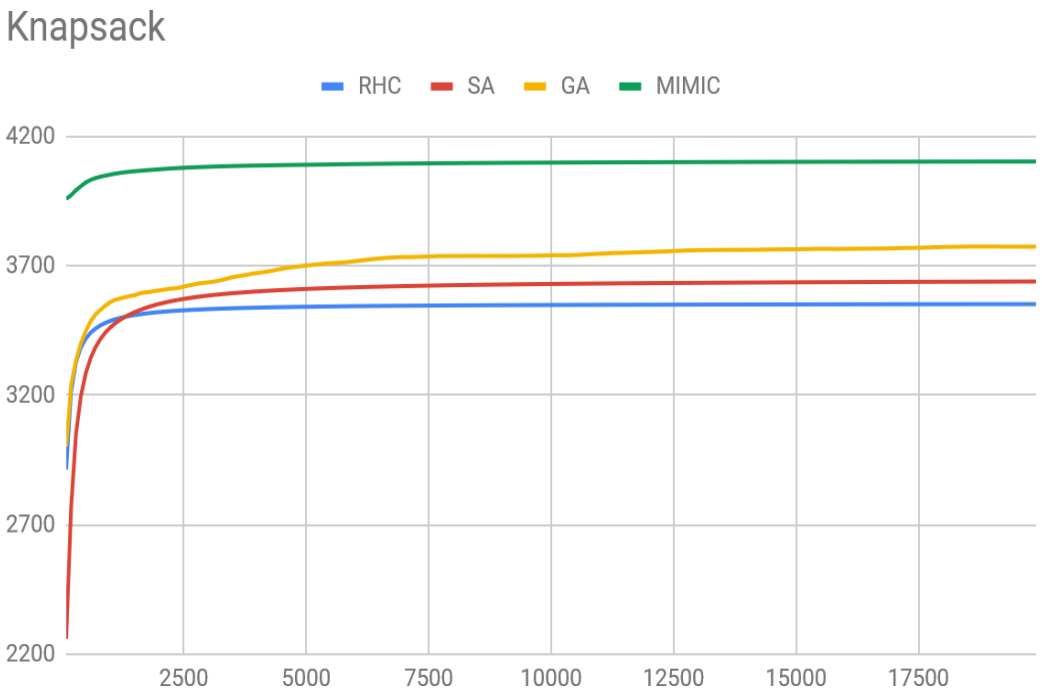
Based on training time, genetic algorithm took a significantly longer time than the rest, and even in testing it took the longest, while RHC was the quickest. Testing fitness functions for GA is expensive while RHC and SA basically looked at its location and value in the distribution, and given their relative accuracies, in Titanic survival's straight forward and simple data set, RHC is by far the best algorithm to use in this circumstance, while GA's expensiveness is not made up for its equivalent performance. SA performed poorly relative to the two with higher run time than RHC but lowest MSE.

Part 2

In this section we shall compare RHC, SA, GA, and MIMIC in three different problems to see how they perform and their strength and weaknesses. For each of the problem sets and each of the algorithms, we used nested for loops to test all combinations of hyperparameters to find the optimal conditions for each algorithm and problem, and the comparisons made between each algorithm are based on that. We then graphed the fitness at each iteration for the algorithms up to 20, 000 iterations to not only see their result, but how they compare at earlier times.

Knapsack

This problem tries to maximize the value you hold based on a maximum volume you cannot pass and certain weights you can chose to take.



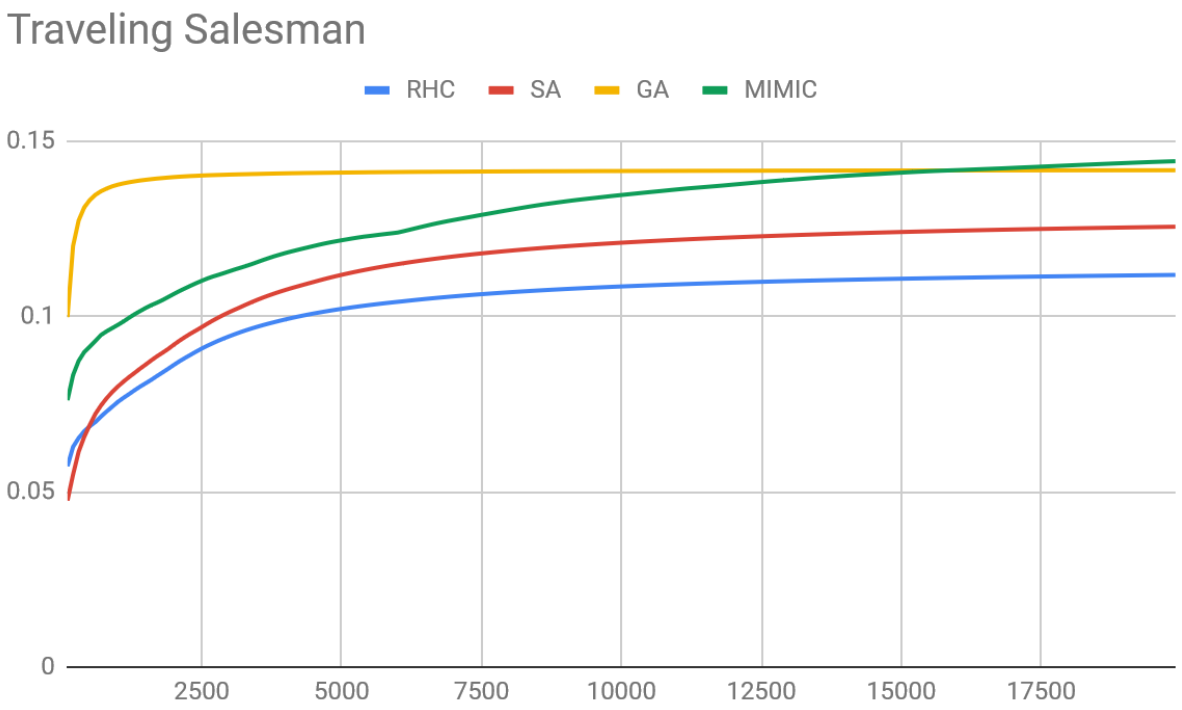
	RHC	SA	GA	MIMIC
Runtime(seconds)	0.039876362	0.040384141	3.105843742	34.288

The graph is the fitness values at each iteration, while the chart shows the runtime, where RHC and SA were magnitudes smaller than GA and MIMIC. It is quickly obvious that RHC and SA performed the worst. This is due to the fact this problem has many local maximas, you can put one large item in the bag and increase the value, but then afterwards be unable to put more in there later. This and many other combinations of weights leads to small peak with underutilized volume left over, creating local maximas, and destroying SA and RHC.

Genetic algorithms get hindered because of crossing over. While normally beneficial distributions are mix and matched, in this case two almost full knapsacks that might become fuller with small weights are mix and matched with each other, taking on values that don't go well together or add together past the max volume of the problem. Despite its high run time, MIMIC the preferred method to solve this problem. It creates similar knapsacks with successful values and work with them, some of which may free up space for higher values, and it is important to keep track of past high value knapsacks to add to them, making MIMIC successful in these conditions.

Traveling Salesman

In this problem, we look at a graph with an individual attempting to visit all of the nodes while minimizing distance, a well-known NP complete problem.



	RHC	SA	GA	MIMIC
Runtime(seconds)	0.064550657	0.053107244	3.856249273	339.1148398

From very early, GA was able to take the lead, plateauing before 1500 iterations. MIMIC eventually catches up at 15000 iterations, which in terms of actual runtime, taking over 5 minutes, vs the 4 seconds of GA just to reach the same result makes it less fit for this problem. Meanwhile RHC and SA dd noticeably worse.

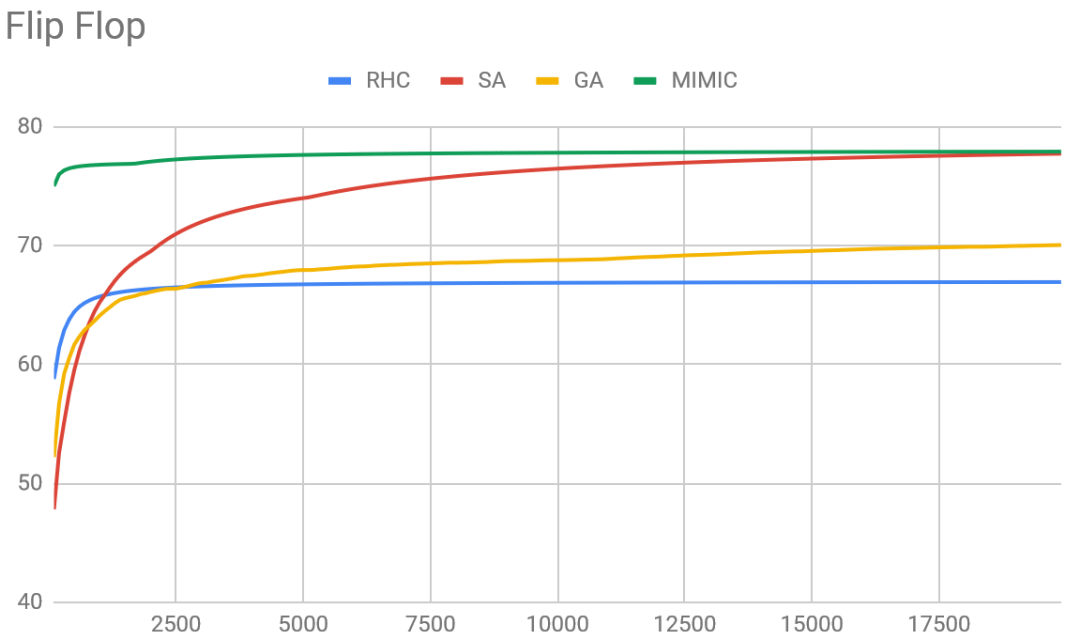
This is because in a graph or network, similar results from the past are greatly favored, since many graphs are possible, but it is important to remember the bottlenecks to reach certain subgraphs. RHC and SA may get stuck on random solutions, because of the almost infinite hypotheses space any solution is considered a maxima for finding a path at all, and there are so many that even SA's randomization does not save it.

GA's crossing over may find two graphs with similar bottleneck paths being used and keep that path, allowing future generations to hold on to important edges of the best solution. MIMIC always performs well in these situations overtime but because of the randomness in fitness and sampling, sometimes important edges are lost. With any amount of time or iterations, GA can succeed as the best for this problem, thanks to its focus on different optimal solutions.

Flip Flop

This problem looks at a binary string, and every time there is a change between consecutive bits it counts as one point for the total value of the string.

	RHC	SA	GA	MIMIC
Runtime(seconds)	0.064365134	0.0519	2.271478386	47.37958636



In terms of computation, this is a very simple question, going through the string to find how many alternations there are. RHC suffered however, because of the many local maximas. One seemingly successful bit string will lead to another one by altering the current one, which means at the algorithm hill climbs, it looks to its next-door neighbors and perhaps changes the next bit to a 0 and one based on the previous, when the previous was wrong or suboptimal already. SA can escape these suboptimal spots with the nature of its algorithm's jumping around. What makes this different from the previous two problems despite having multiple maximas to get stuck on is, this problem has much less, as the bit string with highest score is just constantly alternating starting with 0 or 1, making it so there's only two parity type of peaks, if not only one as the solution. RHC might just get stuck on the few peaks while SA can escape them if there is not too many. GA suffers because you can take two strings that are successful, but if there is a parity or ordering mismatch, the crossover child may in fact lose some of its alternation. With mutation as well, where there was an alternation a mutation on the second subsequence can change it from a 101 to a 111. MIMIC does well because randomly, it may find the perfect string, or over time, get closer and closer, and using its generation of sample, it can find optimal answers.

However, with time taken into consideration, MIMIC took around 1000 times longer time tan SA, while both having the same results, and SA is much less expensive computationally, and should MIMIC have a poor generation function it would hinder the performance as well, making SA the winner when there are some but not many local mixmas, and the problem is straight forward.

Conclusion

When looking at datasets, with Titanic being a super straight forward problem with few if any suboptimal maximas, RHC performed the best, while SA suffered randomly going to different places just to end up at the same place. GA was successful but did a lot of randomization and added variance for little performance improvement. SA was favored in flip flop since it is also a simple problem, but a few local maximas is enough to shut RHC down and bring SA up. GA is dependent upon the past, while adding mutations and cross overs, it is great for large hypothesis spaces where many things can be similar but there is still large degrees of freedom, and where highlighted commonalities with some variance is helpful. Lastly, with enough time and computational power, MIMIC is able to excel in all subjects, being able to highlight past successes without injecting unnecessary variance or depending on individual cases too much.

Acknowledgements

Diego Milla (2018Kaggle[Kaggle.com]) Introduction to Decision Trees(Titanic dataset) for dataset and python script

Pushkar, “Abagail”, https://github.com/pushkar/ABAGAIL

Rohan Ramakrishnan, “Abagail Notes” and driver file, https://docs.google.com/document/d/1mizXqkLVq2kXJN83HmDZqpE5DuY3YugGETCMo-PTp8M/edit

Ronny Kohavi and Barry Becker (1996). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Data Mining and Visualization ,Silicon Graphics.