Randomized Optimization
CS 7641 Assignment 2
Mikey Ling
mling7 903278121

Introduction

The purpose of this report is to compare four common randomized optimization algorithms: random hill climbing (RHC), genetic algorithms (GA), simulated annealing (SA), and mutual information maximizing input clustering (MIMIC). The report is broken up into two sections. The first applies the first three algorithms to find weights in a neural network on the abalone dataset. The results of these neural networks will be compared to the neural network created via back propagation in Assignment 1. The second applies all four algorithms to three different optimization problems and provides analysis for each.

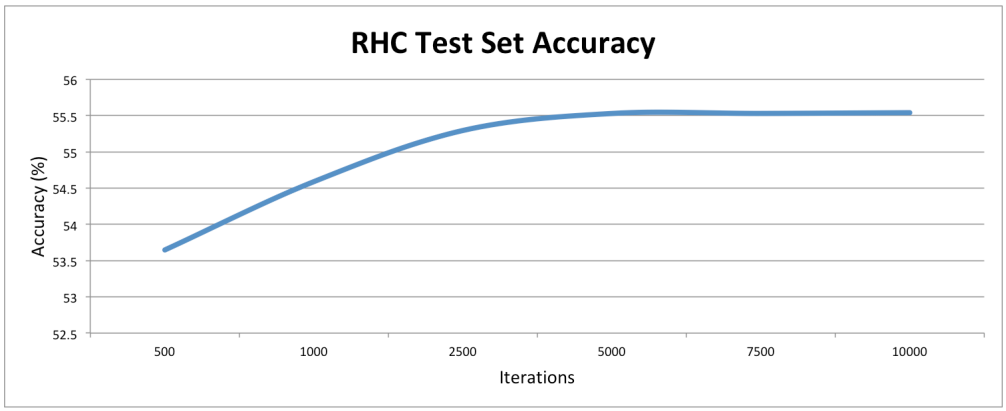
Dataset Description

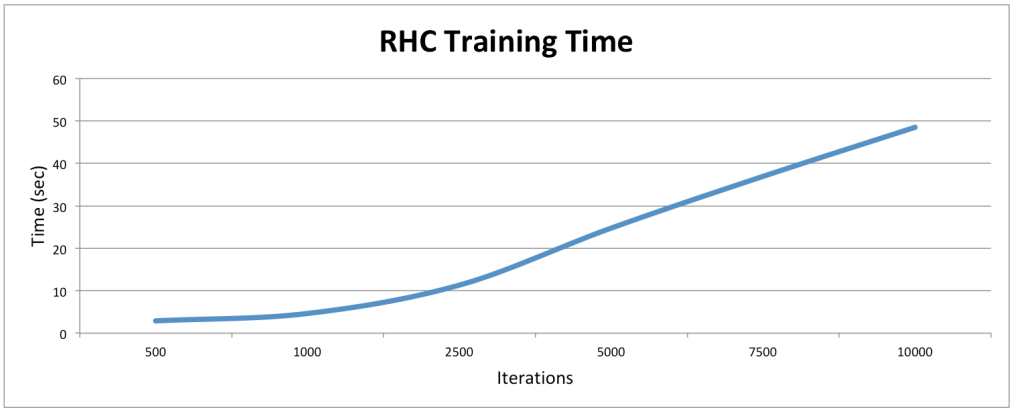
The raw Abalone dataset contains 4177 samples with 9 attributes: gender, length, diameter, height, whole weight, shucked weight, viscera weight, shell weight, and number of rings. The 'gender' parameter originally had three classes: I, M, and F (infant, male, and female). For the sake of this experiment, all samples labeled as an Infant were taken out of the dataset leaving 2835 samples left. This now turns the problem into a binary classification problem. The same procedure was performed in Assignment 1. Now the performance of the neural network used in Assignment 1 can be compared to the performance of the neural networks produced in this report.

Part I: Random Optimization in Neural Networks

Random Hill Climbing

Hill Climbing is a straightforward search algorithm that iteratively loops through independent variables in the direction the dependent variable(s) increases, in other words uphill. The search terminates when it discovers a maximum where no other neighbor has a higher value. The algorithm doesn't keep track of its search history nor does it look beyond the immediate neighbors of the current state. While this leads to the algorithm being simple and easy to understand/visualize, its simplicity can lead to getting stuck in local maxima when there exists a higher global maximum. Random Hill Climbing (RHC) helps alleviate this problem: it conducts a series of hill climbing searches that begin at random values of the dependent variable (within the given bounds, of course). Let's define P as the probability of each hill-climbing search returns a local maximum value. We can now estimate it will take 1/P iterations to find the function's global maximum. This means the success and time complexity of RHC is highly dependent on the overall shape of the function that's we are trying to optimize and the maximum of iterations we allow the algorithm to run through.



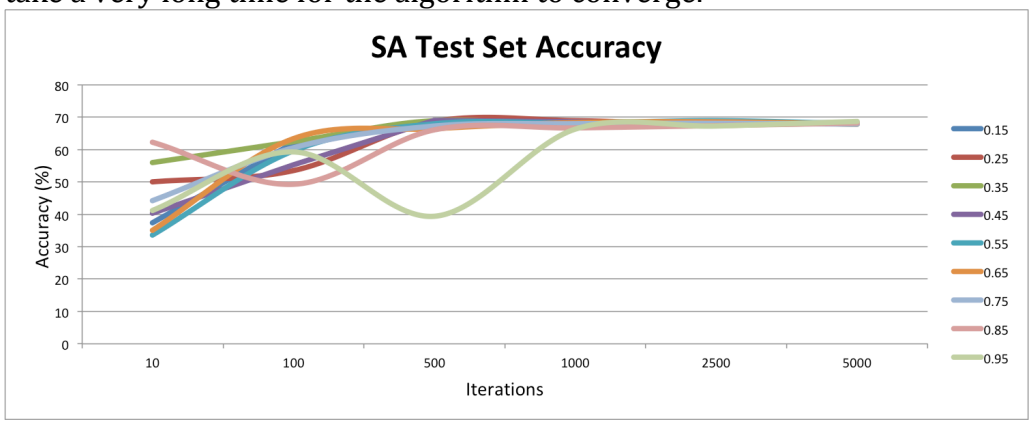


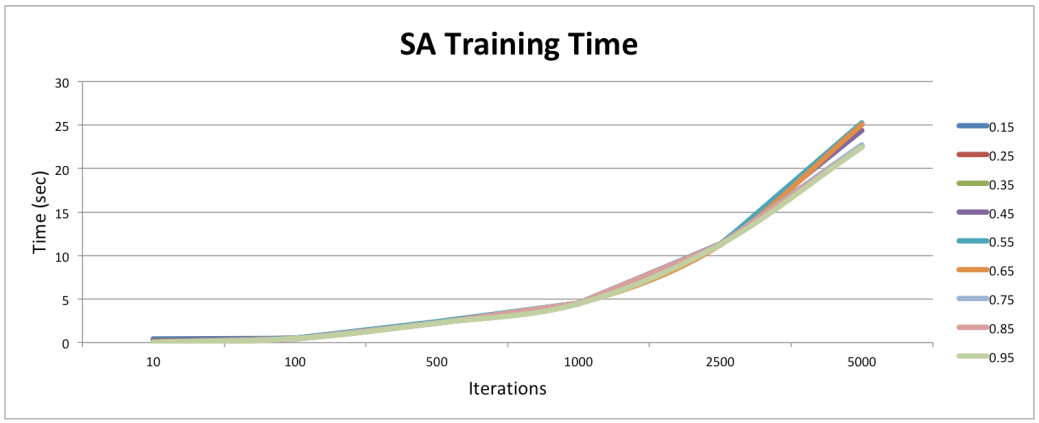
Simulated Annealing

Simulated Annealing (SA) is a version of stochastic hill climbing that introduces downhill capabilities to the RHC algorithm to increase efficiency and success rates. Downhill movements are executed at a rate dependent on the following probability function:

$$P = e^{\frac{\Delta E}{T}}$$
$$\Delta E \rightarrow neighbor\ value - current\ value$$
$$T \rightarrow temperature$$

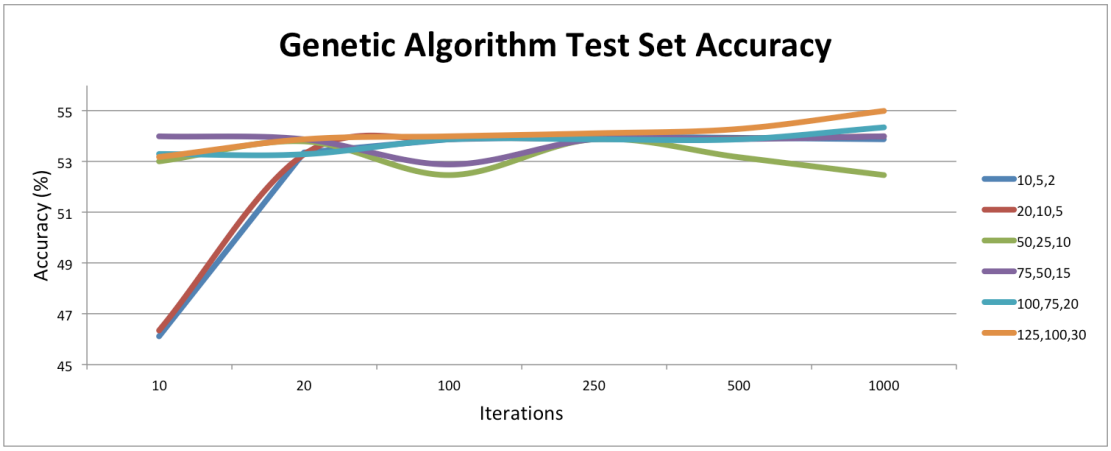
The probability of downhill movements decreases as the difference between neighbor and current values decreases and as the algorithm "cools off" (T goes down). The cooling rate of the algorithm is determined by the user and greatly influences the behavior/success of the algorithm. The temperature and cooling rate of the algorithm is a demonstration of the exploitation vs exploration paradigm that's prevalent in reinforcement learning. Faster cooling rates will lead to quicker convergence times; however, there's a chance the algorithm doesn't converge to the global maximum. On the other hand, if the cooling rate is too low, it may take a very long time for the algorithm to converge.

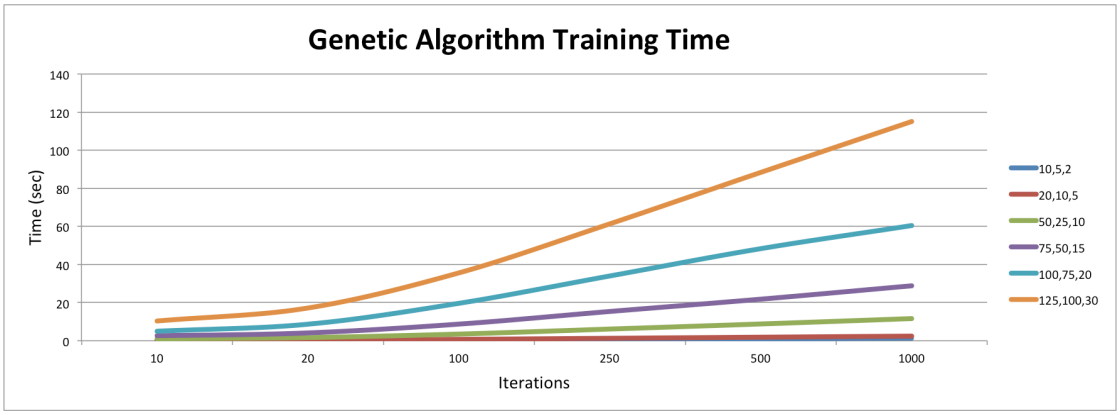




Genetic Algorithm

The Genetic Algorithm (GA) attempts to converge to an optimal solution by applying biological phenomenon to the hypothesis space. It begins with a set of k randomly generated states (known as the population) with each state representing a finite "genetic code" of 1s and 0s, much like chromosomes of a given animal. The algorithm then rates each state via a fitness function defined by the user. Higher rated states are chosen to reproduce children, and this rate is directly proportional to the fitness of the state. The offspring of the states are created via crossover and/or mutation (swapping 0s and 1s at certain indices or randomly changing them). GA combines the random exploration aspect of RHC and the bi-directional exploration capabilities of SA. GA converges to very accurate solutions; however, it is much slower than SA and RHC due to the more complex operations that occur during runtime.

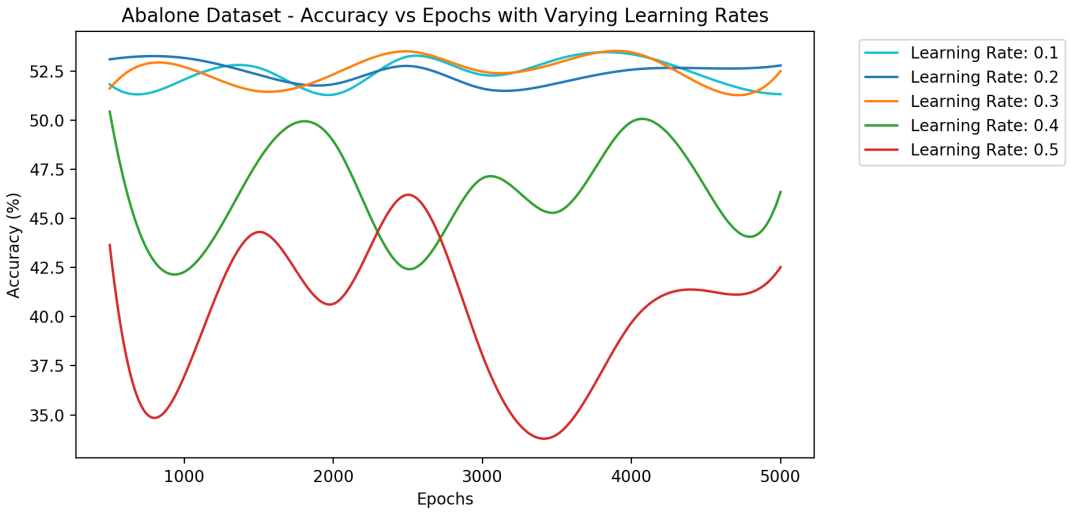




Part I Conclusion

The table below is a summary of the performance of the three random optimization algorithms featured in this section of the report. SA, with a cooling rate of 0.65, outperforms RHC and GA by nearly 12%. A graph from Assignment 1 was provided to compare how these three algorithms compare to using backwards propagation to determine the weights for a neural network.

SA (Cooling = 0.65)		**RHC**		**GA (500, 350, 30)**	
Iterations	*Accuracy*	*Iterations*	*Accuracy*	*Iterations*	*Accuracy*
10	35.059	500	53.647	1	53.294
100	63.412	1000	54.588	5	53.882
500	66.471	2500	56.353	10	53.647
1000	68.353	5000	55.294	50	53.882
2500	68.588	7500	55.529	100	54.353
5000	67.882	10000	55.53	150	53.882



The maximum accuracy achieved by the neural network from Assignment 1 was about 53%. GA and RHC resulted in a neural network that achieved similar results. SA, however, produced a network that achieved an accuracy of 68.7%! This is *significantly* higher than the results found in Assignment 1.

Simulated Annealing			
Cooling Rate	**Test Accuracy**	**Training Time**	**Test Time**
0.15	68.471	22.641	0.002
0.25	68.235	22.476	0.002
0.35	67.882	22.454	0.002
0.45	68.353	24.367	0.002
0.55	68	25.232	0.003
0.65	67.882	25.067	0.002
0.75	67.882	22.747	0.002
0.85	68.235	22.448	0.002
0.95	68.706	22.488	0.002

Referring to the results produced by Simulated Annealing (shown above), it's interesting to show the miniscule effect the cooling rate has on the test accuracy of the neural network. This lack of effect can be the result of the overlapping properties the dataset has. In other words, two samples with very similar feature values can have different labels and vice versa. This inherent randomness in the dataset could

have some sort of nullifying effect on the influence of the cooling rate (how often the algorithm accepts random/bad movements). A similar behavior is demonstrated in the Genetic Algorithm graph. Notice after 20 iterations how the accuracies of the different configurations converge to approximately the same accuracy (though the accuracy is lower than SA's).

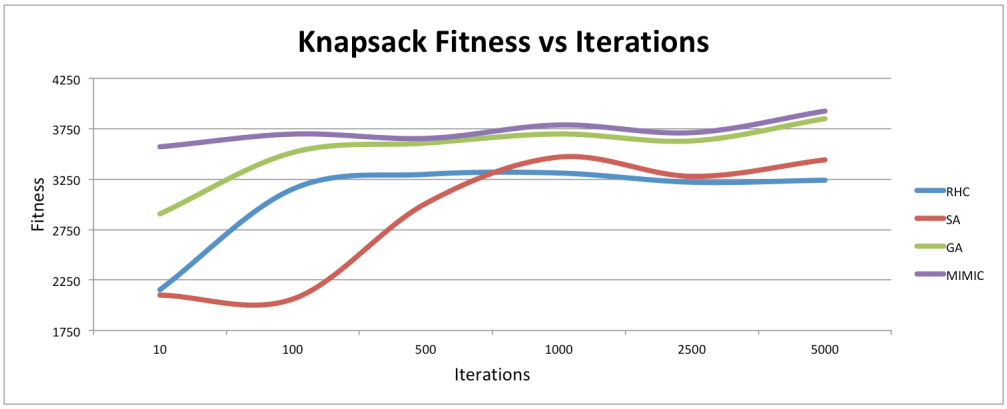
I think SA was able to produce much better results than RHC and GA because it's able to work with smaller datasets than GA, which requires more data to converge, and allows for more bad (downhill) moves than RHC due to the temperature/cooling mechanism. This combination of the ability to work with relatively small datasets and random exploration is the reason why I believe SA is able to achieve such good results for a dataset that contains a lot of overlapping labels.

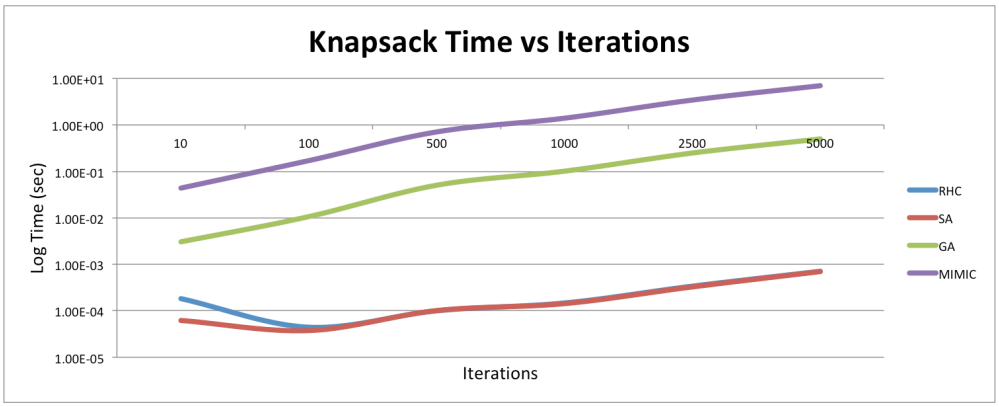
Part II: Analyzing Other Optimization Problems

In this part of the report, we apply four random optimization algorithms (RHC, GA, SA, and MIMIC) to three optimization problems, Traveling Salesman, Continuous Peaks, and Knapsack. We compare the performance of the four algorithms by analyzing the output of their fitness functions and computation times. One of the four algorithms is chosen for further analysis based on these two metrics.

Knapsack Problem

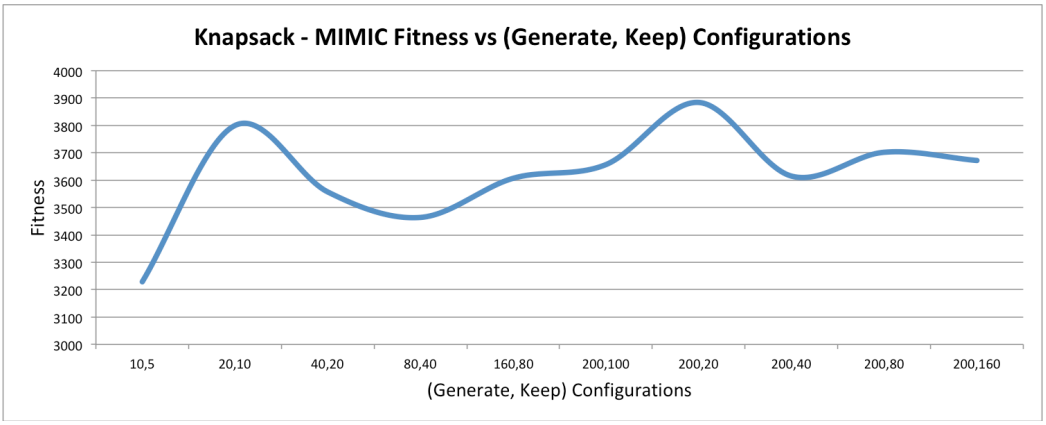
The Knapsack problem is a combinatorial optimization problem. Given a set of items, each with a weight and a value, we need to determine the number of each item to include in a collection, so the total weight is less than or equal to a given limit and the total value is as large as possible. ABAGAIL is used to simulate a Knapsack problem with 40 different items (4 duplicates of each item, so 160 total) each with different weight and volume values (both with max values of 50).

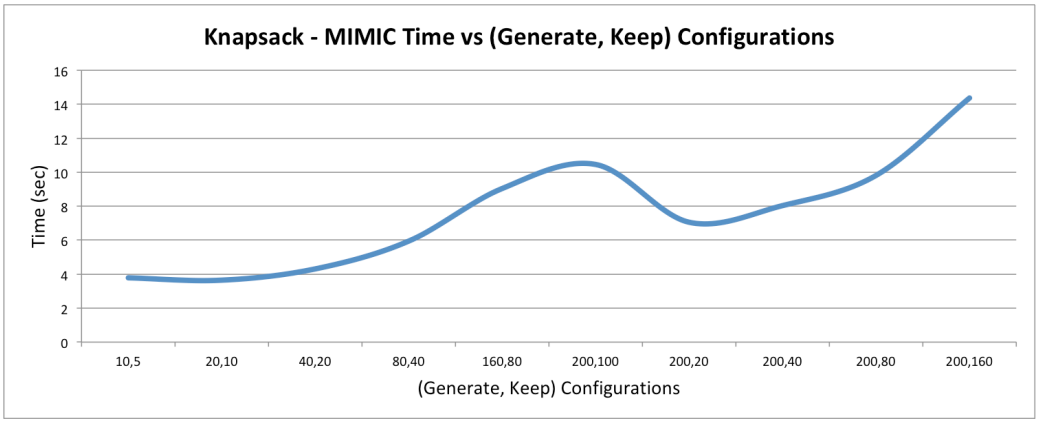




Given the two graphs above, we can determine MIMIC provides the highest fitness value, though it takes the longest to do so. The next question is *why* does MIMIC outperform the other algorithms? My intuition leads me to believe this problem is simply too complex for "simpler" algorithms like SA and RHC. As mentioned in the lectures, MIMIC is the only algorithm that keeps track of the structure of the data *and* defines a clear probability distribution from which the samples are taken from. It could be argued that GA keeps track of some sort of structure, but it's not done so at the same scale the MIMIC does. And it's true that SA defines a proper probability distribution. MIMIC takes these two characteristics of GA and SA and combines them so it can solve complex optimization problems like Knapsack.

We can conclude MIMIC is the superior algorithm when solving this combinatorial optimization problem when it comes to acquiring the highest fitness value; however, this comes at the cost of having the longest running time. To further analyze how MIMIC performs when solving this problem, a separate experiment was run with a fixed number of iterations (5000) and differing configurations of Generate and Keep values.



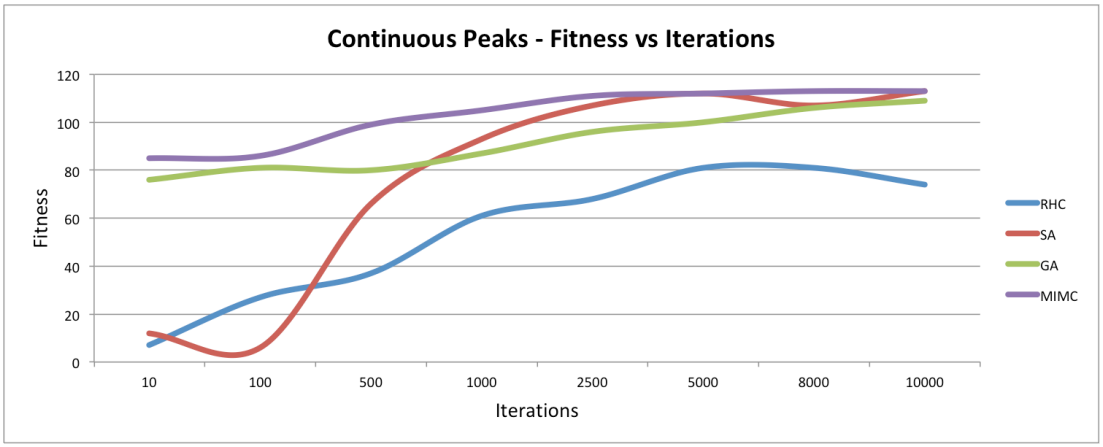


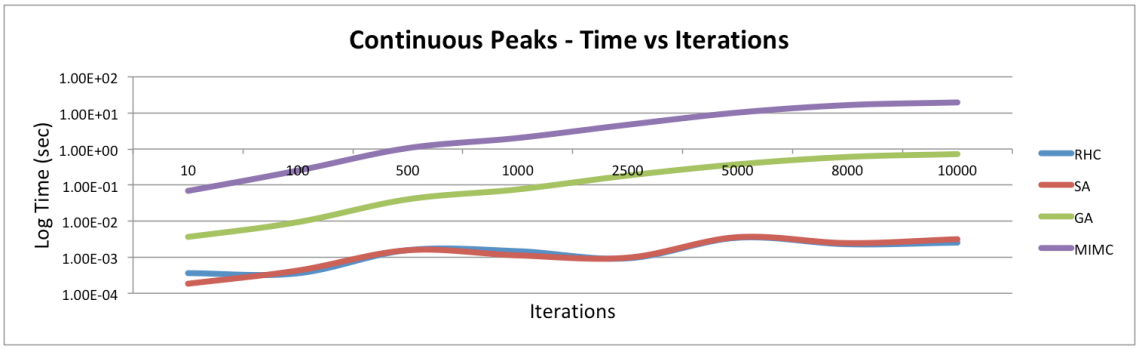
The experiment reveals MIMIC's performance (usually) increases as the values of Generate and Keep increase. It's interesting to mention the dip the fitness function shows at Generate values of 200 and any Keep value above 20. This downward movement can be attributed to the possibility of the algorithm keeping too many solutions from the distribution it chooses from. In other words, some of these solutions may NOT lead to optimal solutions. Whereas keeping 20 solutions seems to be the optimal amount of samples to keep between iterations.

Continuous Peaks Problem

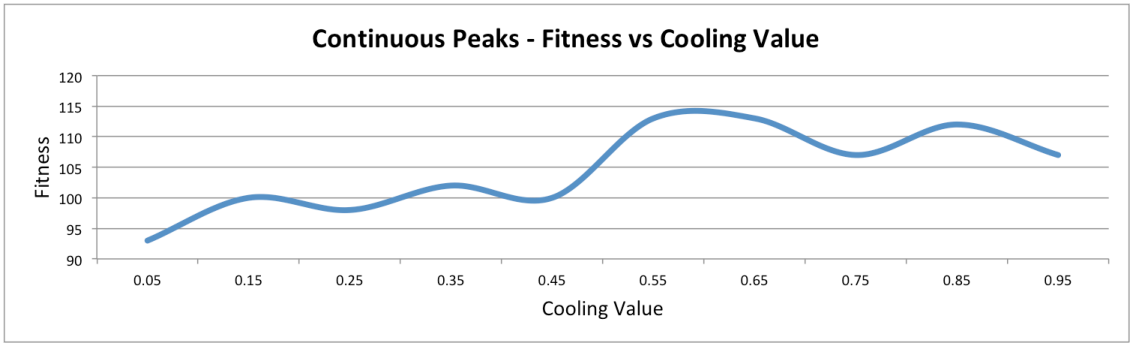
The Continuous Peaks problem is a continuous optimization problem that tasks us with finding the global maximum of a continuous function that contains multiple local maxima. For our implementation of this problem, we set N = 60 and T = 6, a detailed description of this problem and its relevant variables/parameters can be found in Section 6.1 of Isbell's Paper.

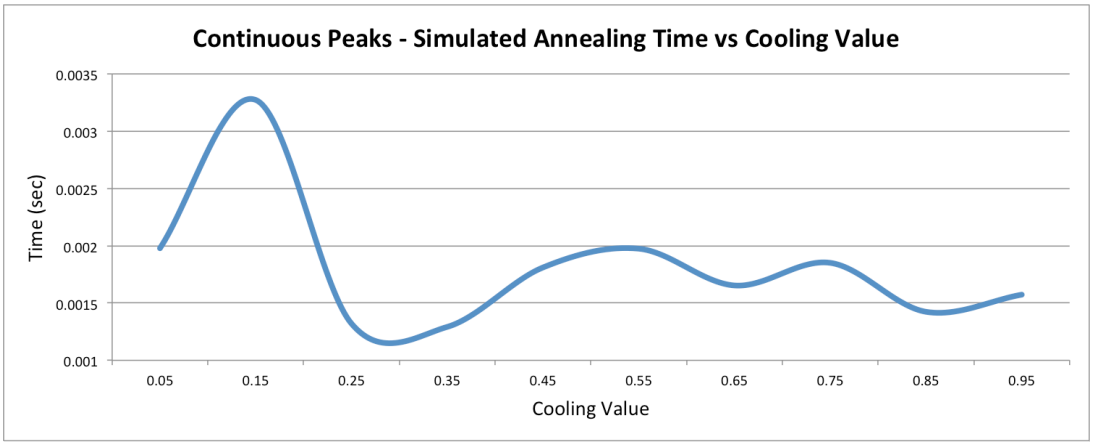
The two graphs below visualize the how the fitness and computation time of the optimization algorithms as the number of iterations increases. MIMIC is the superior algorithm when it comes to fitness, though it's the slowest in terms of computation time. Simulated Annealing comes close to the amount of accuracy MIMIC is able to accomplish when 10,000 iterations are performed, and it has a much faster computation time. Given a large enough number of iterations is allowed, SA is the algorithm of choice because it's able to achieve the accuracy MIMIC is able to accomplish *and* it takes much less time to reach this accuracy. How does SA perform at such a higher speed than MIMIC? Well, SA doesn't have to go through the process of choosing samples within the n^{th} percentile of performance nor keep track of $P^{\theta}(x)$, the probability distribution being represented and kept throughout all of MIMIC's iterations. This is a perfect example of some things Professor Isbell mentioned in the lecture: MIMIC performs orders of magnitude fewer iterations than SA to converge to a solution (500 compared to 5000). And although an algorithm may perform more iterations to converge to a solution, the *amount of time* each iteration takes must also be taken into account.





We provide further analysis of SA's performance for this problem by conducting an additional experiment that involves a fixed iteration number of 5000 and an initial temperature of 1E11. We then use different cooling rates to see how they affect the performance of the algorithm:

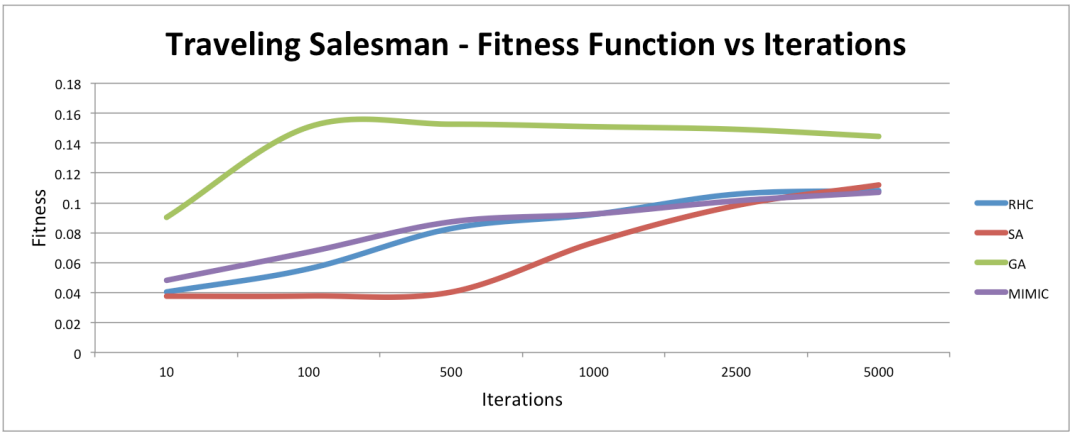


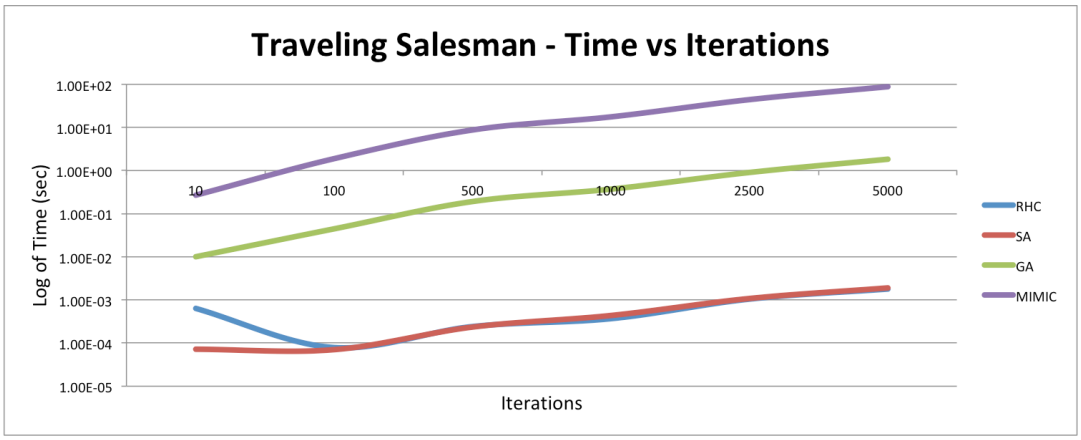


It seems Simulated Annealing achieves its highest fitness value when the cooling rate is 0.55 (though 0.85 isn't too far behind). It's interesting to note the fitness value starts to decline at a cooling rate of 0.95. This could infer the algorithm doesn't allow for enough randomness in its search and settles on a local maxima; however, if this were strictly the case, then a cooling rate of 0.85 shouldn't have a higher fitness value than that of with a cooling rate of 0.75. This strange anomaly is most likely due to some randomness within the algorithm. It'd be interesting to see the results of an averaged graph from performing a large number of trials to see if the dip at 0.75 is recurrent. It's reasonable to predict these small perturbations in the graph would be "averaged" out, and the Fitness vs Cooling Value graph would end up looking more parabolic.

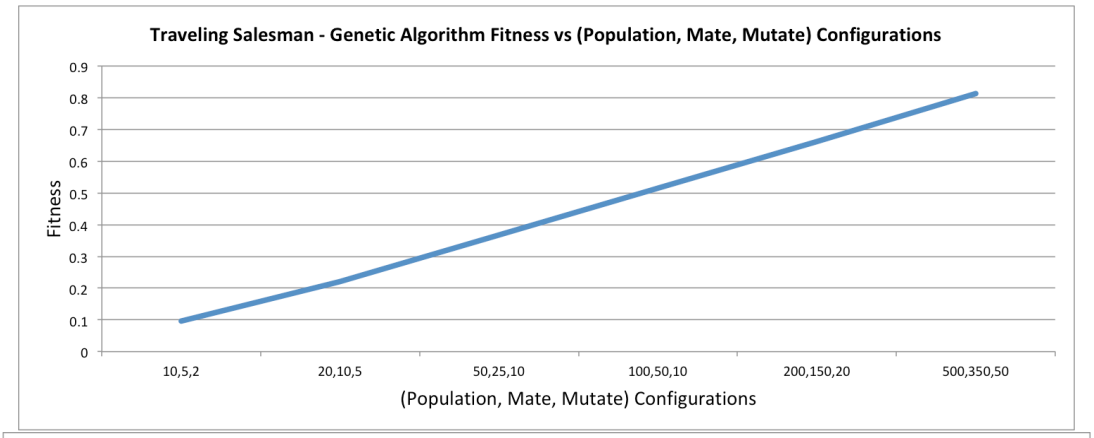
Traveling Salesman Problem

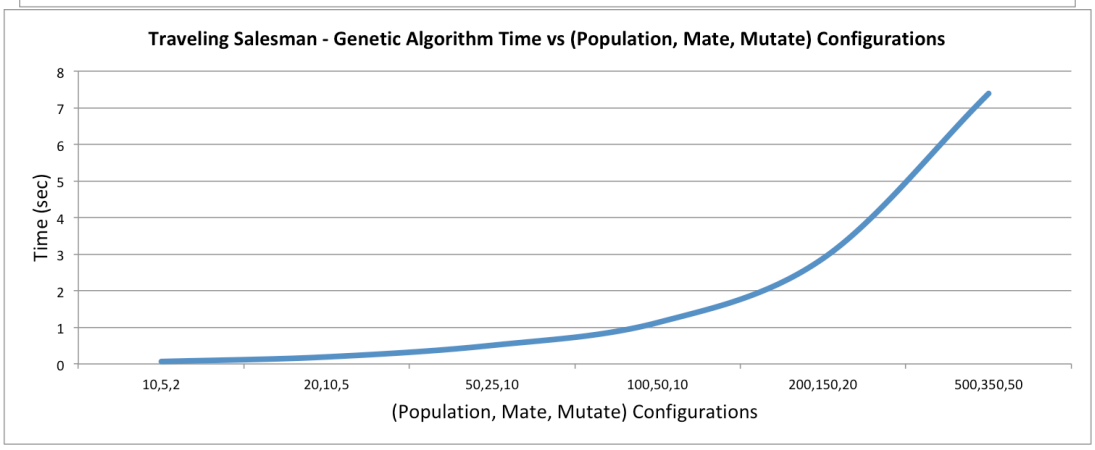
The Traveling Salesman Problem asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" It's an NP-Hard problem of combinatorial optimization that's often used to benchmark optimization algorithms (hence, why we're using it!). Below are two graphs that demonstrate the algorithms' performance when solving this problem:





Genetic Algorithms are clearly the best algorithms to use for this problem. The fitness value it achieves is significantly higher than the other three algorithms, and its runtime is *not* the longest. How is it able to outperform MIMIC in this case? Although the algorithms are very similar, MIMIC needs to go through the task of maintaining a distribution function in through each iteration, and although less iterations is a plus when evaluating the cost function is laborious, MIMIC may not introduce the same magnitude of randomness or stochasticity needed to find the optimal solution for this problem. Again, we chose to run an additional experiment to see how GA performs when the number of iterations is kept at 5000 and different combinations of (population, mate, mutate) parameters are used:





It's interesting to note the fitness increases in a linear fashion (although the variable represented on the x-axis isn't one dimensional) while the computation time increases exponentially. The positive linear trend of the fitness function most likely does not continue infinitely, so it'd be interesting to see at which combination of (population, mate, mutate) it starts to flatten out and reach a maximum.

Conclusion

The optimization problems in Part II were chosen because they reflect how the different optimization algorithms perform depends on the type of problem they are trying to solve. The next portion of the report will summarize these results:

Simulated Annealing

Simulated Annealing performed best in **Continuous Peaks** because it's an algorithm that's good for time sensitive, simple, low cost problem spaces. The Continuous Peaks problem is the simplest of the three problems features in Part II of the report as it's an expansion on a typical hill-climbing problem that can be solved with simple gradient ascension and random walking (which is essentially what SA does). SA requires much less computational resources than GA or MIMIC, which is why it's able to converge to a solution that's equally as good as MIMIC but in a fraction of the time.

Genetic Algorithm

I believe GA performs best in the **Traveling Salesman** problem because it's able to explore larger solution spaces than SA and it is able to explore more solution spaces than MIMIC. Exploring larger solution spaces is a result of GA's "picking a population" mechanism that's chosen from the sample space based on which samples produce the highest fitness value, whereas SA only tracks single solutions at a time. The mutation, crossover, and mating mechanism allow the algorithm to explore a larger variety of samples than MIMIC because they introduce randomness into the sample space. These two characteristics bode well with the Traveling Salesman problem because the problem lends itself to being solved with stochastic/random solutions in order to find the most efficient path between destinations.

MIMIC

MIMIC performed the best in the **Knapsack** problem. The MIMIC algorithm iteratively samples from a uniform distribution and only "keeps" the samples that produce a fitness value that's within a given percentile. It repeats this process until the results cease to improve. This process is allows MIMIC to solve large-scale problems with complex data interactions because it doesn't "keep" the data samples that interact in a meaningless (low fitness) way. The Knapsack problem is arguably the most complex problem featured in Part II of the experiment, so it's no surprise MIMIC produced the best results.

In sum, the size of the dataset and the allowed computation time are two factors that are likely to influence your choice of algorithms. After completing the experiments mentioned in this report, we can conclude, in general, MIMIC and GA are the algorithms to use if you have a large scale dataset and flexible time constraints whereas RHC and SA are more suitable for smaller datasets and stricter time constraints. MIMIC may take a longer time to converge to a solution, but it's able to do so with *significantly* less iterations than SA. That being said, MIMIC is favorable over SA if the cost of evaluating the fitness function is high (for example rocket simulations). On the other hand, if you're working with a fitness function that can be evaluated extremely fast and you do *not* want to know the probability distribution of the sample space when the problem has been solved, SA should be your algorithm of choice.

References

https://www.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf
https://en.wikipedia.org/wiki/Knapsack_problem
https://en.wikipedia.org/wiki/Travelling_salesman_problem
https://github.com/pushkar/ABAGAIL