Project 2 – Randomized Optimization

Anthony Garber – agarber7@gatech.edu

1 Introduction

In machine learning we frequently are required to find the maximum (or minimum) of a function. However, due to the complexity of these functions it is often not possible to explore all states (NP-hard). Therefore, we are left to devise the best way to search the input space to identify the global maximum. In this paper we will explore how several randomized search algorithms perform on problems of various complexity.

2 Random Search Algorithms

In this paper we have chose to focus on four randomized search algorithms: random hill climbing, simulated annealing, genetic algorithms, and MIMIC. As the experiments will show, each of these algorithms have its own strengths and weaknesses which depend on the complexity of the optimization problem.

2.1 Randomized Hill Climbing

The random hill climbing algorithm operates by picking a random starting location and evaluating the function at adjacent points. It repeats this process, continually moving in the direction of higher function evaluations, until it find a maximum. However, random hill climbing suffers from finding local maximum rather than global maximum. To account for this we incorporate the concept of "random starts" where upon finding a maximum the algorithm starts over again – giving it another chance to find a better maximum. However, random restarts were not used in the experiments in this paper to illustrate how easily random hill climbing can get stuck in local maxima.

2.2 Simulated Annealing

Simulated annealing gets is name from metallurgy where blacksmiths will perform a controlled cooling of metals allowing the atoms to rearrange themselves into a compact structure resulting in a harder metal. In the context of search algorithms, this is meant to be an improvement on random hill climbs by allowing the algorithm to jump to worse function evaluations as a function of "Temperature" (stolen from metal annealing) and the decrease in function evaluation. The algorithm will jump to this new state in the function with higher probability if the Temperature is high or if the decrease in function evaluation is low. As we iterate through the algorithm the temperature decays (as would a cooling metal) to allow the for the algorithm to converge on a maximum. In effect this allows for the search algorithm to break out of local maxima while the temperature is high, but as the temperature decreases it begins to operate identically to a random hill climb – only making the move if the function evaluation is strictly higher.

The annealing probability formula (likelihood that the algorithm will jump to a worse state) is calculated as:

$$P = e^{\frac{f(x_t)-f(x)}{T}}$$

2.3 Genetic Algorithm

The genetic search algorithm introduces the idea of a "population". Rather than picking a single random function evaluation the genetic algorithm evaluates the function for each member of the population (size K). Then, the "most fit" individuals (members with the highest function evaluations) are paired up and their input parameters are mixed/crossed over/bred to produce offspring with a mix of their traits. This process is repeated until the algorithm has been executed a set number of times. The genetic algorithm is good at quickly converging on the maximum but does so by executing may function evaluations in parallel.

2.4 MIMIC

The shortcoming of the three previously discussed algorithms is that each iteration of the search only carries over a point or set of points which serve as the starting point for the next pass of the algorithm. The MIMIC algorithm attempts to solve for this by sampling a population from a probability distribution (the first pass being a sample from a uniform distribution). After sampling from the distribution, we generate a new probability distribution which satisfies based only off the most "fit" individuals. This new probability distribution will serve as the sampling distribution for the next iteration of the algorithm. These probability distributions are estimated use dependency trees which assume that each input variable is dependent on exactly one other input. This assumption allows us to generate relatively accurate probability distributions without the covariance from multiple variable dependencies causing the problem space to become exponential.

3.0 Optimization Problems

To illustrate the various strengths of each of the algorithms discussed above, we chose to solve three optimization problems: Four Peaks, Max K-Color, and the Traveling Salesman problem.

3.1 Four Peaks

The Four Peaks algorithm has precisely two global maxima and is described by the formula [3]:

$$Fitness(X,T) = \max[tail(0,X), head(1,X)] + R(X,T)$$

Where

$$tail(b,X) = number\ of\ trailing\ b's\ in\ X$$

$$head(b,X) = number\ of\ leading\ b's\ in\ X$$

$$R(X,T) = n, if\ tail(0,X) > T\ and\ head(1,x) > T$$

$$R(X,T) = 0, otherwise$$

In these experiments we let T = 10% as was done in Isbell, 1996 [3].

3.2 Max K-Color

This optimization problem involves a system of connected nodes. The goal of the problem is to assign each node one of K colors such that we minimize the number of connected nodes with the same color.

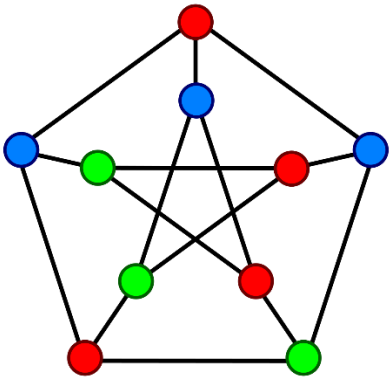


Figure 1: An example K-Colored (K=3) graph where no connected nodes share the same color [1].

3.3 Traveling Salesman

The traveling salesman problem involves a set of points on a two-dimensional grid. The goal of the optimization is to visit all the points exactly once, and to minimize the total distance traveled (i.e. a salesman traveling door to door and wishing to do it as quickly as possible). A visualization is show in figure 2.

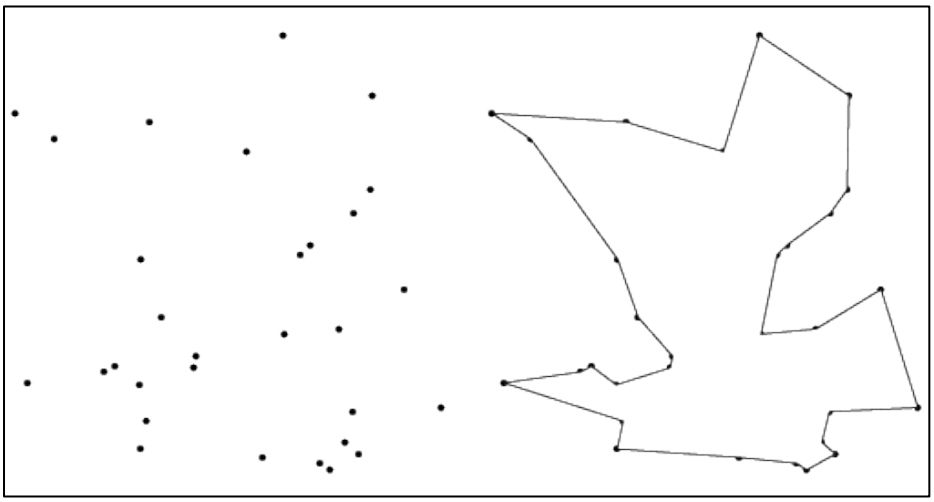


Figure 2: A sample traveling salesman problem along with its corresponding shortest path [4].

4.0 Results

For each of the optimization problems we used the four previously discussed algorithms to determine which would perform the best. In some experiments the best algorithm is subjective as multiple algorithms were able to identify the global maximum. We will evaluate model performance based on run time and number of iterations required to converge on the global maximum.

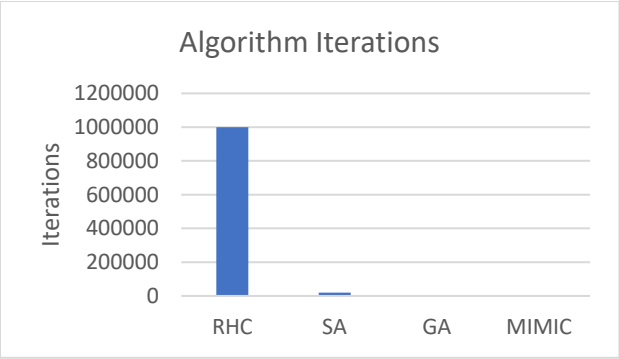
4.1 Four Peaks

For the Four Peaks problem we used a 64-bit string as a discrete valued parameter space. The optimal parameters to solve the four peaks problem for each of the algorithms are listed in table 1. For all non-listed hyperparameters the default values were used as specified by the MLRose documentation [2].

Algorithm	Attempts	Iterations	Fitness	Time
RHC	1000	1000000	64	0.065
SA	100	20000	64	0.077
GA	10	100	120	1.53
MIMIC	10	10	81	12.82

Table 1: Optimal hyperparameters to solve Four Peaks problem.

Of the four algorithms, only GA was able to converge with default hyperparameters (aside from attempts and iterations). It is worth nothing that some algorithms required a higher number of attempts to converge (the default attempts specified by MLRose is 10) therefore making a direct comparison of the iterations somewhat misleading. While we were able to observe better fitness scores when playing with the other hyperparameters (such as population size for MIMIC) – we thought it best to focus on showing the shortcomings of each algorithm by only tuning iterations. Figures 3 and 4 below show the tradeoff between iterations and convergence time.



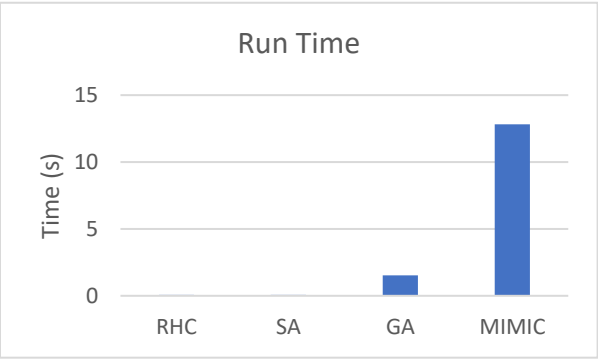


Figure 3 (left): Number of iterations require for algorithm to converge on Four Peaks problem. Figure 4 (right): Run time for algorithm convergence.

While MIMIC took the fewest iterations to converge, it was not able to find a global maxima in a reasonable run time. This is due to the algorithm incorporating the additional step of fitting the points to a population distribution rather than using a fixed probability distribution (i.e. simulated annealing) or using the data points as a proxy for the distributrion (i.e. genetic algorithm). The genetic algorithm successfully located a global maximum and required very few iterations to do so. It also preformed reasonably well on the run time making it well suited to solve the Four Peaks problem.

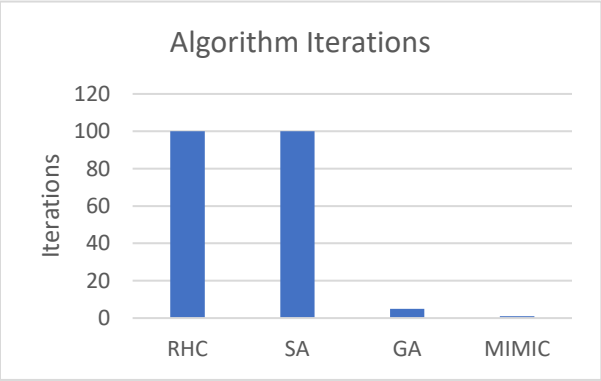
4.2 Max K-Color

For the K-color expirment we ran 16 nodes with 30 connections and 2 possible values.

Algorithm	Fitness
RHC	24
SA	30
GA	30
MIMIC	30

Table 2: Fitness function optimum identified by the respective algorithms.

Even in a problem space as simple as 16 nodes the random hill climbing algorithm was unable to identify a global maximum. The random hill climbing algorithm suffers from being unable to explore outside of local maxima. Therefore, to identify a global maximum it must randomly start its search in the region of the fitness function which is increasing all the way to the global maxima. For particularly complex problems it may be nearly impossible to luckily find the global maximum. Ideally we could have continued to increase the problem complexity until some of the other algorithms had difficulty locating global maximum in reasonable amounts of time. However, due to the data structures involved in K-Color problems, it is difficult to scale to large complexities.



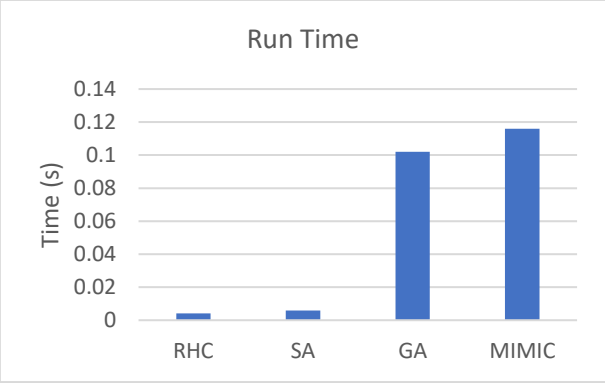


Figure 5 (left): Number of iterations required for algorithm to converge on Max K-Color problem. Figure 6 (right): Run time for algorithm convergence.

Again we see GA and MIMIC with much fewer iterations required for convergence. Each iteration of GA and MIMIC sample hundreds of observations from a population making the “iteration”measurement a relatively poor comparison. Run time is probably a more relevant convergence metric for real world applications and we see that simulated annealing does the best of the three algorithms which were able to identify a global maximum.

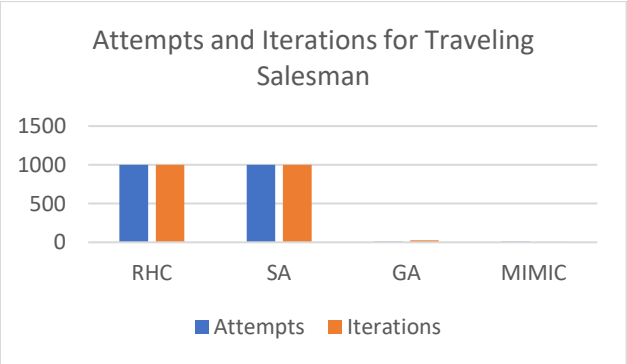
4.3 Traveling Salesman

We set up the traveling salesman problem with 13 points for traversal. For the traveling salesman problem only the genetic algorithm was able to find a global maximum in the time provided.

Algorithm	Attempts	Iterations	Fitness	Time (s)
RHC	1000	1000	23.65	0.122
SA	1000	1000	25.54	0.126
GA	10	25	22.6	0.75
MIMIC	10	5	27.9	1.07

Table 3: Optimal hyperparameters for traveling salesman problem

Here we see the shortcomings of evaluating only attempts and iterations. Ideally, we could have explored larger hyperparameter spaces for each algorithm to see if they are capable of finding global maximum, but in doing so making it more difficult to compare the algorithms against one another (no longer apples to apples). GA and MIMIC used the default population sizes while SA used the default Temperature cooling hyperparameters. Although it is not ideal that run times of algorithms did not permit a full exploration of parameter space, from a practical point of view the most relevant metrics are (1) did we find a global maximum and (2) how many real hours did it take to find the global maximum. Perhaps we could have found global maxima for the other algorithms as well had we explored wider population sizes or slower cooling schedules, but those additional experiments have a real time cost associated with them. The genetic algorithm was able to provide the best results under a reasonable exploration of the hyperparameter space.



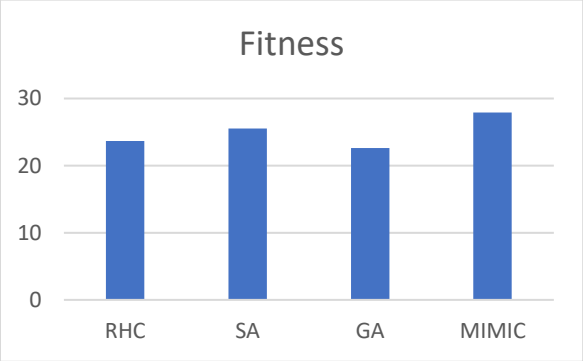


Figure 7 (left): Comparison of attempts and iterations for each algorithm's best solution. Figure 8 (right): Optimal solution (travel distance) for each algorithm for the traveling salesman problem.

Simulated annealing was unable to find a global maximum – this shows that this algorithm is weak on problems which involve an ordinal spatial routing as it may be difficult for it to break out of local maxima for these functions.

4.4 Neural Network

Beyond the three optimization problems discussed previously, we also evaluated the ability of randomized hill climb, simulated annealing, and the genetic algorithm to find good weights for the neural network which was created in project 1. The neural network in project 1 was created using the python library sk-learn. The neural network was used to classify MNIST digits (handwritten digits from 0-9). The optimal hidden layer structure was determined by sk-learn to be (9,6,6) and yielded an 89% accuracy. Figure 9 and 10 show how the various algorithms compare in training time and model accuracy.

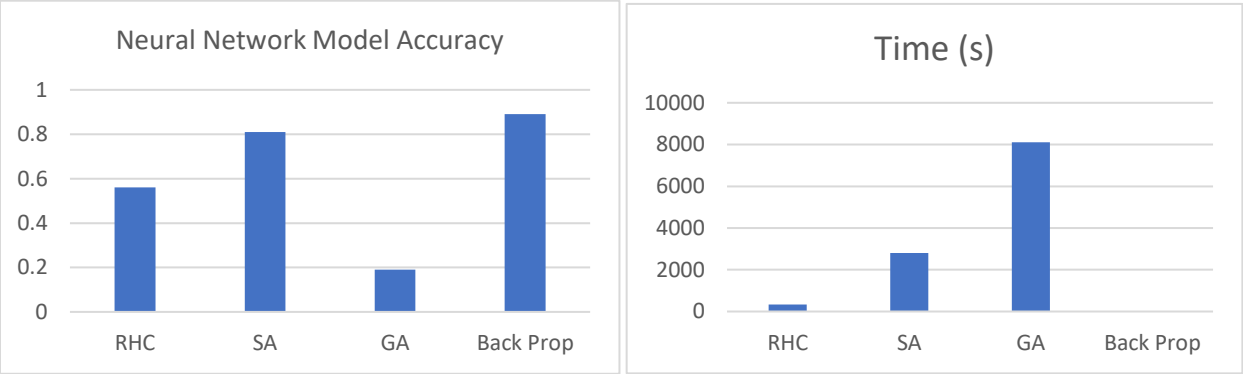


Figure 9 (left): Comparison of model accuracy of neural networks tuned by various random search algorithms. Figure 10 (right): comparison of model training time for each search algorithm.

Unlike the prior optimization problems (Four Peak, etc.) the MNIST classification problem does not have a fitness function describing the effectiveness of the algorithm at finding the correct maximum. Instead, we must use the trained model to evaluate how accurate it is at classifying each sample as the correct digit. Figure 10 shows that the sk-learn library's standard back propagation was the most efficient at finding good weights for the neural network. However, it should be noted that simulated annealing was still showing signs of improvement as iterations went up, but due to time constraints it was not feasible to test higher iterations for better model accuracy. However, given the vast model training time difference it would be hard to argue that standard back-propagation was not the all around best algorithm.

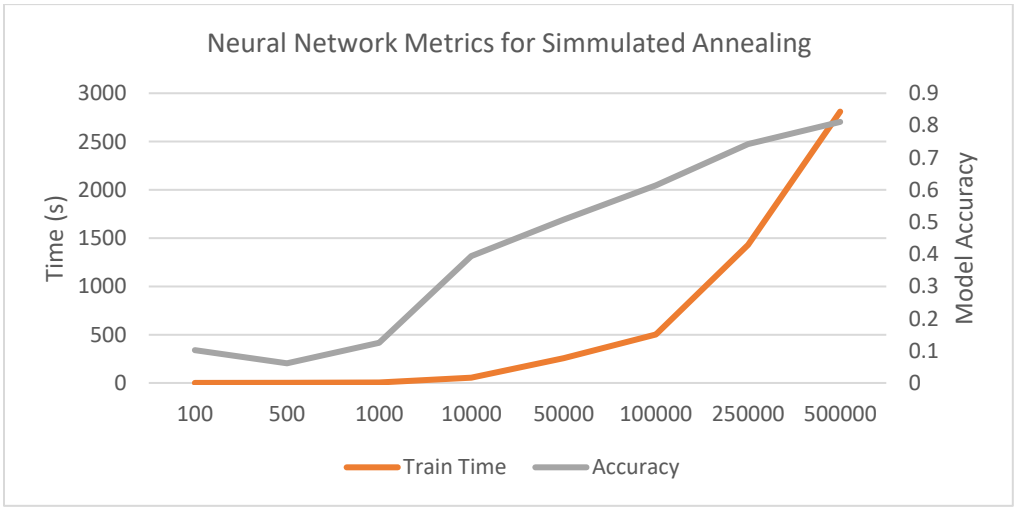


Figure 11: Visualization of how training time and model accuracy are affected by number of iterations for simple annealing on a neural network.

5.0 Conclusion

Our goal in running these experiments was to see which types of problems were best suited for each of the learning algorithms: random hill climbing, simulated annealing, genetic algorithms, and MIMIC. While our results began to give us insight into this question, time/computing constraints prevented us from fully exploring both the complexity of the various optimization problems (i.e. only 16 nodes and 2 colors in the K-Color problem) as well as the full extent of the hyperparameter space of the various algorithms.

We saw that random hill climb consistently performed the worst as we did not permit it to do random restarts. It was therefore susceptible to getting stuck in local maxima and not finding the optimal fitness function value. MIMIC performed exceptionally well with very few iterations, but in some instances the added complexity associated with calculating a probability distribution inflated the run time such that we were unable to explore higher iterations and population size hyperparameter values. This resulted in the algorithm underperforming since on several occasions we did not find the global maximum in the allotted time.

Simulated annealing consistently performed well with very low run times but did struggle on the traveling salesman problem (did not find a global maximum). This algorithm outperformed random hill climbing (which it shares many similarities) because of its ability to break out of local maxima. However, it was unable to do so in the traveling salesman problem suggesting a weakness in distance routing.

Finally, we tested the Genetic Algorithm which was the only algorithm able to find the global maximum on the 64-bit four peaks problem. The population component of the genetic algorithm made it effective at identifying global optimum and it consistently did so in less time than MIMIC (a similar algorithm which also shares a population hyperparameter).

The neural network results were particularly difficult to compare because time constraints made it impossible to test higher iterations for the non-standard algorithms. However, even with the limited trials that we were able to run it was apparent that back-propagation performed much better in considerably less time. Time constraints did not just affect the neural network trials. We would also have liked to test additional hyperparameters for each of the algorithms – Temperature and cooling rate for simulated annealing and population sizes for genetic algorithms and MIMIC. But, even without these data points we were able to glean some of the strengths and weaknesses of each of these algorithms.

Citations

1) "File:Petersen graph 3-coloring.svg." *Wikimedia Commons, the free media repository*. 13 Oct 2019<https://commons.wikimedia.org/w/index.php?title=File:Petersen_graph_3-coloring.svg&oldid=364818523>
2) Hayes, G. (2019). ***mlrose: Machine Learning, Randomized Optimization and Search package for Python***. https://github.com/gkhayes/mlrose. Accessed: *10 Oct 2019*.
3) Isbell, Charles L. Jr. "Randomized Local Search as Successive Estimation of Probability Densities." https://www.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf
4) "Traveling Salesman Problem". http://mathworld.wolfram.com/TravelingSalesmanProblem.html