Keith Mertan
CS 7641 - Machine Learning
Assignment 2

Randomized Optimization

Optimization is often a difficult, even NP-Hard, problem. Brute force searches through a parameter space often are ruled out quickly due to this. Four alternative methods for searching for optima in complex parameter spaces are discussed in this paper. These include Randomized Hill Climbing, Simulated Annealing, Genetic Algorithms and MIMIC.

Randomized Hill Climbing is the most simplistic of the bunch. As implemented in the code preceding this paper, the algorithm randomly chooses a beginning state for the problem and checks its neighbors for improvement. If a neighbor of the current state is found to be an improvement, that neighbors state becomes the current (and best) state and the algorithm begins again by checking its new neighbors. Once no more improvements can be made the algorithm is said to have converged. This method is subject to falling into local optima and can be improved with randomized restarts at different states to attempt a more exhaustive search of the parameter space. The code used did not implement these randomized restarts so a single search to the nearest optimum is used.

Simulated annealing borrows its concept from the physical process of annealing. The algorithm takes two parameters: T (initial temperature) and C (cooling factor). Like Hill Climbing, Simulated Annealing starts from a randomly generated state. Neighbors of that state are considered as candidates for the next iteration of the algorithm. Where this differs from Hill Climbing is how the two deal with neighbors that are less favorable than the current state. Hill Climbing will never accept a neighbor with a lower fitness score, but Simulated Annealing will accept a worse state with a probability dictated by the current temperature. The temperature decreases over iterations, allowing the algorithm to jump around the search space early on then take less risky updates toward the end.

Genetic Algorithms attempt to preserve and integrate the qualities of the most fit candidates by selecting those candidates above a certain threshold at each iteration and generating offspring from them using a method called crossover. The algorithm introduces randomness by allowing for mutation after crossover. There are three parameters varied in the Genetic Algorithm used for this analysis: The population, the crossover rate and the mutation rate. The population determines the number of instances to consider in each generation. The crossover rate determines the fraction of the population to be replaced within each generation. The mutation rate dictates the the percentage of the population that will have a single bit flipped after mutation. Each instance has a probability of remaining in the data proportional to it's relative fitness among the population at each iteration of the algorithm.

Lastly, MIMIC uses estimated joint probability distributions to generate instances at each iteration and retain the most fit. MIMIC takes two parameters in this implementation: The number of samples and the fitness threshold. The number of samples determines how many instances are generated from the probability distribution at each iteration. Any instances with a fitness score below the fitness threshold are dropped. The the joint distribution of the remaining points is estimated again and the process repeats until convergence. This method allows relationships between variables to be modeled which is a feature not present in the

other algorithms. Those algorithms are limited optimizing each individual value within the instance.

The algorithms described are designed to operate in discrete valued spaces. A modification has been made to allow for part one of this assignment: training neural network weights in a continuous space. Instead of a neighbor being the previous and next discrete value, for example, the neighbors are generated as a random continuous distance from the current state. For part two of the assignment, these algorithms are compared across three different discrete valued optimization problem domains.

The code used comes from Jonathan Tay's GitHub repository which utilizes ABAGAIL.

Part 1 - Neural Network Weight Optimization

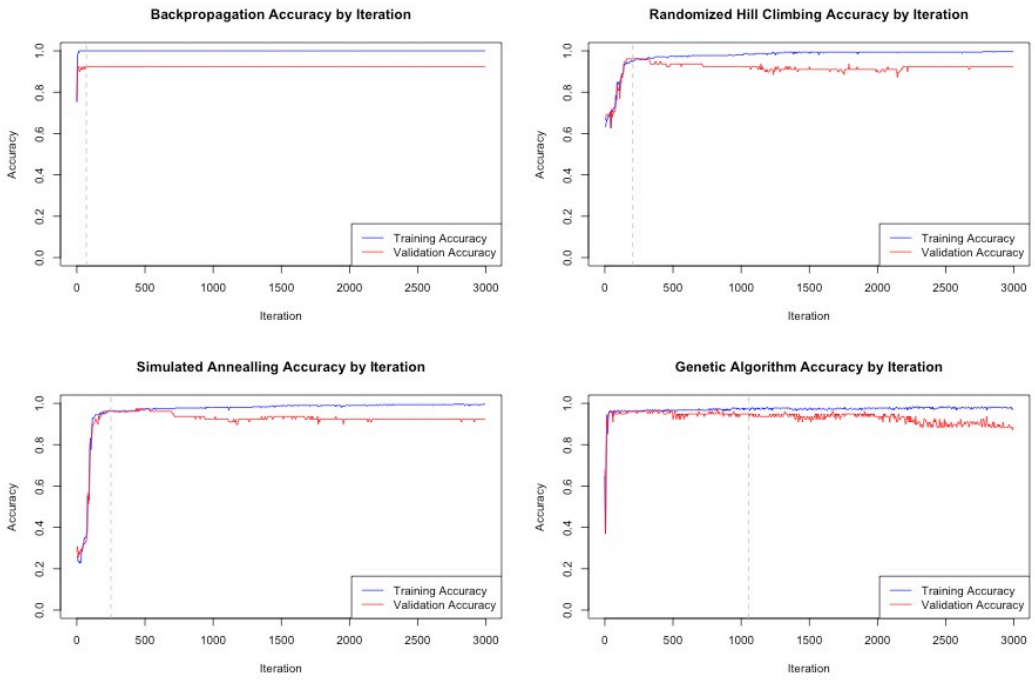
Part 1.1 - Introduction

Optimizing the weights of a neural network is a nontrivial task. In terms of the error, there may exist many local optima over the space of possible weights and convergence to a global optimum is not guaranteed. Randomized Hill Climbing, Simulated Annealing and Genetic Algorithms are compared here to each other and to the benchmark of Backpropagation.

The data set the network is trained on is the breast cancer data set from mldata.io. The data contains 569 rows which each represent a patient, and 10 independent variables. All 10 variables are integers which measure attributes of cell nuclei, such as shape and size. The dependent variable is binary, taking values "malignant" or "benign". The data has been split into 70% training and 30% testing, with 20% of the training data reserved for validation.

The network architecture comes from the optimal parameters found in assignment 1 of this class. The network is a single hidden layer with 18 nodes and a ReLU activation function. Each optimization algorithm is run using these parameters to create a fair comparison.

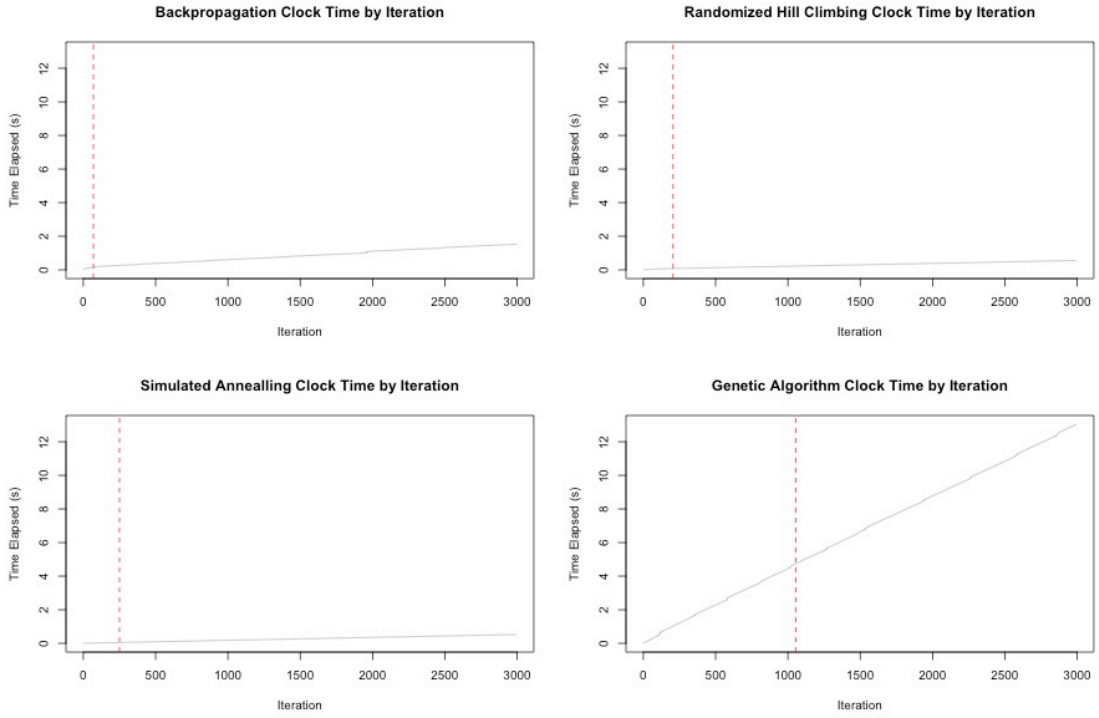
Part 1.2 - Performance



The plot above shows the performance of each algorithm over 3000 iterations. The vertical gray dotted line shows the iteration of converge with epsilon = .000005. Surprisingly, backpropagation performed the worst when extracting the set of parameter values at the iteration of convergence. This may be due to untuned parameters. Since ABAGAIL does not support cross validation out of the box, the optimal network structure used in assignment one was duplicated as closely as possible and rerun in ABAGAIL to be scored on the same criteria as the other algorithms during training. One feature that is appealing of backpropagation despite its poor performance in this situation is its smooth convergence. The other three algorithms overfit the data past a few hundred iterations and begin to suffer a loss in validation accuracy. Backpropogation suffers from overfitting a bit as well, but the training and validation accuracy stop diverging at some point. It is also the most quickly converging algorithm in terms of iterations.

Both Simulated Annealing and Hill Climbing reached the maximum validation accuracy upon convergence, tied at 96.15%. Simulated Annealing used a cooling parameter of 0.35 in this instance. This was the best of the five parameters tested from 0.15 to 0.95. We see in the first few hundred iterations that Simulated Annealing may have been stuck in a local optima and found its way out as it's designed to do. The randomness of the Genetic Algorithm did not allow it to build on the relationships between the weights. That random disposal of useful information may have caused it to do poorly and have more erratic behavior than the others.

Part 1.3 - Time



The plot above shows the number of iterations of an algorithm against the elapsed time in seconds over 3000 iterations. At convergence, Simulated Annealing is the fastest with 0.048 seconds. The longest running algorithm before convergence is the Genetic Algorithm. It took 4.738 seconds to converge. This speaks to its expensive computation and its inability to capture meaningful relationships in data that has them, such as neural network weights. We also see a steeper slope for backpropagation due to its need to compute gradients at each iteration. Overall, Simulated Annealing is the fastest converging algorithm in seconds and achieved the best performance optimizing the neural network weights.

Part 2 - Three Optimization Problem Domains

Part 2.1 - Introduction

To compare the performance of Randomized Hill Climbing, Simulated Annealing, Genetic Algorithms and MIMIC, three discrete valued optimization problem domains are chosen. These problems are known as Flip Flop, Continuous Peaks and The Traveling Salesman Problem.

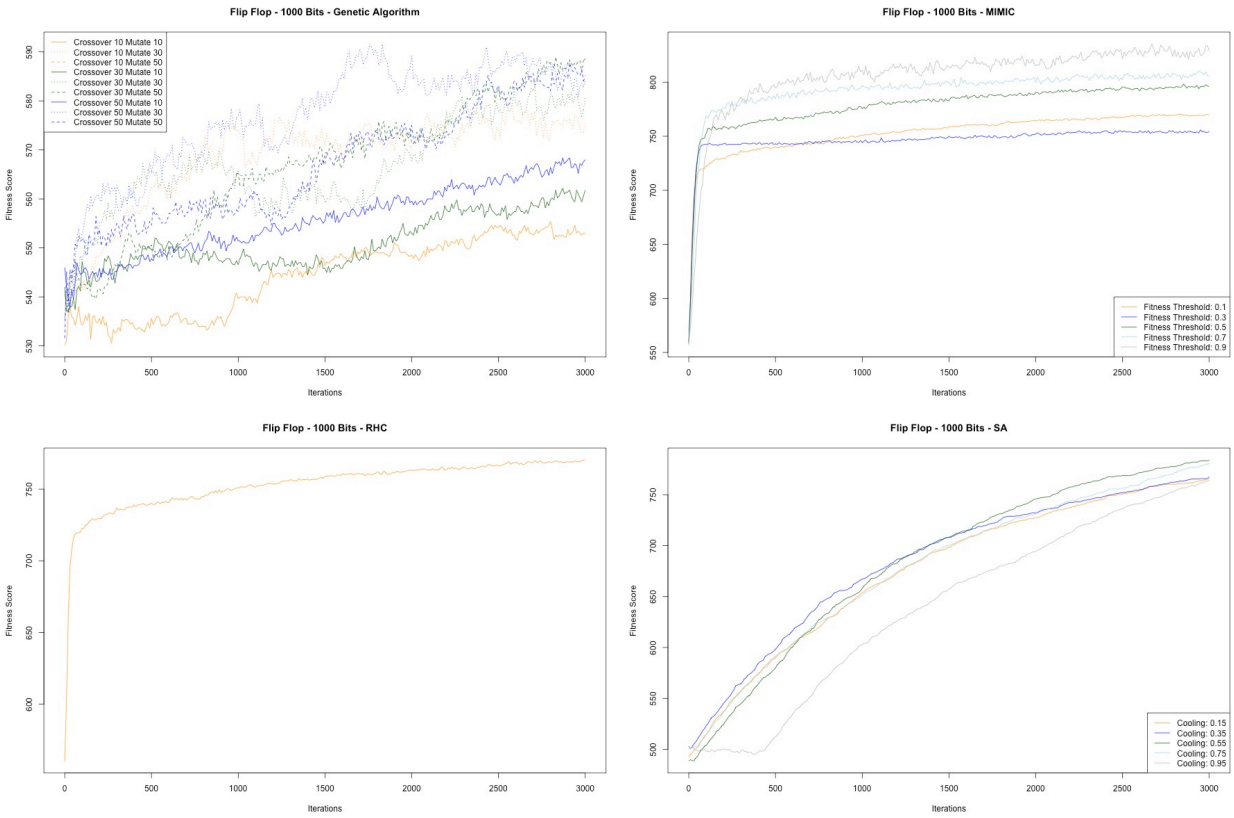
The Flip Flop problem scores instances based on the number of alternations between bit values along the bit string. N is the only parameter for the problem and is the length of the bit string which is randomly generated. The optimal value is N - 1 and occurs under two scenarios: one when the bit string begins with 0 and alternates at every position throughout the string and the other when it begins with 1 and alternates in the same fashion. The fact that the optimal value is known beforehand makes this a valuable problem for evaluating the performance of algorithms and their ability to converge to the global optimum.

Continuous Peaks attempts to maximize the length of consecutive zeroes or ones in a bit string, with an additional bonus added in the case where the length of the longest string of zeroes and the length of the longest string of ones is greater than or equal to T. T an arbitrary parameter to threshold the bonus score and N, the second parameter, is the length of the entire bit string. As is the case with the Flip Flop problem, the deterministic solution to Continuous Peaks is also known. N + (N - T - 1) is the optimal value of the fitness function and allows us to compare our algorithm results directly to the global maximum.

The Traveling Salesman Problem aims to optimize the distance traveled between a given set of points. In this analysis, N is the only parameter set for the problem. N is the number of destinations along the route. A series of N continuous values is randomly generated and distances between them are calculated and summed at every iteration for each algorithm. The inverse distance is used as the fitness function to allow this to be a maximization problem, not a distance minimization problem. This has many practical applications such as discovering optimal delivery routes for companies like Amazon.

Part 2.2 - Flip Flop

Flip Flop is a problem with structure. Bits are dependent on the other bits around them to score points. This relationship is best captured in this case by MIMIC. MIMIC, using the highest fitness threshold tested which is 0.9, far outperformed its competition in terms of fitness. The other algorithms struggle due to the little information available when considering

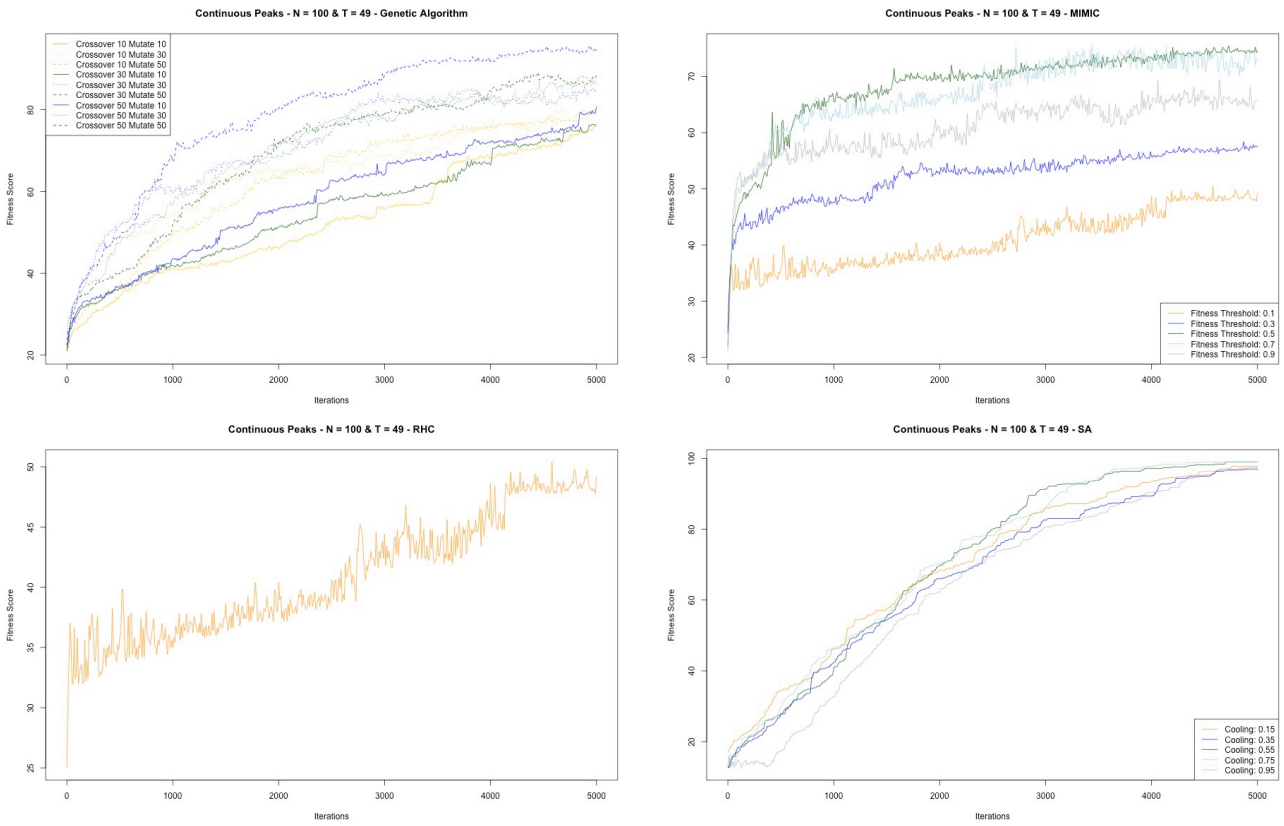


each value of the instance individually and not the joint distribution of the entire instance. Hill Climbing is relatively successful, but may have had a lucky start and landed near the global maximum or a high local maxima. Simulated Annealing did not converge but showed promise as its fitness increased steadily through its 3000th iteration. More time to run the algorithms may produce better results for Simulated Annealing. The Genetic Algorithm saw erratic results due to its randomization in this structured setting.

Part 2.3 - Continuous Peaks

Simulated Annealing performed best with a moderate cooling coefficient of 0.55. It was able to explore the parameter space while cooling quickly enough to find a promising optimum. With a maximum fitness value of N + (N - T - 1) = 100 + (100 - 49 - 1) = 150, it can be seen that none of these methods have approached that value very closely. Most appear not to have converged. This is a computationally expensive task and if it were more feasible to complete additional iterations of the problem they would be included in this analysis.

Since the Genetic Algorithm had such high crossover, it could be that many useful strings of bits were being combined generation over generation with additional help from mutation where the bit strings were incomplete. Since the MIMIC implementation always produces 100 instances at each iteration, this perhaps is not sufficient in the case of a length 100 bit string. These instances may have quickly caused the estimated joint probability distribution to shift away from the true distribution and become unrecoverable since MIMIC assumes that every new generation could essentially have been a subset of the prior.

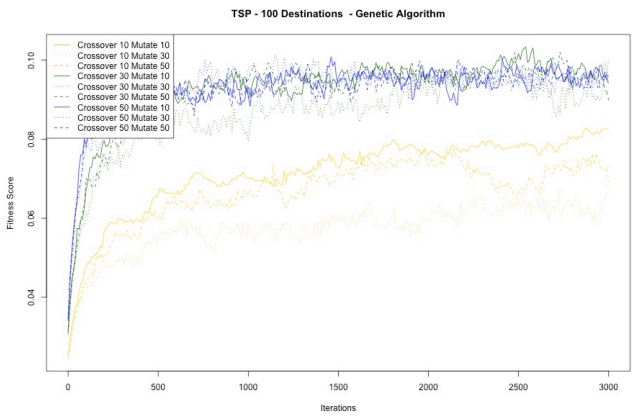


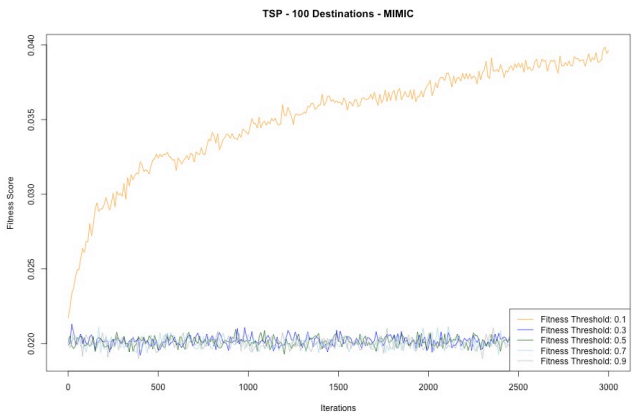
Part 2.4 - Traveling Salesman Problem

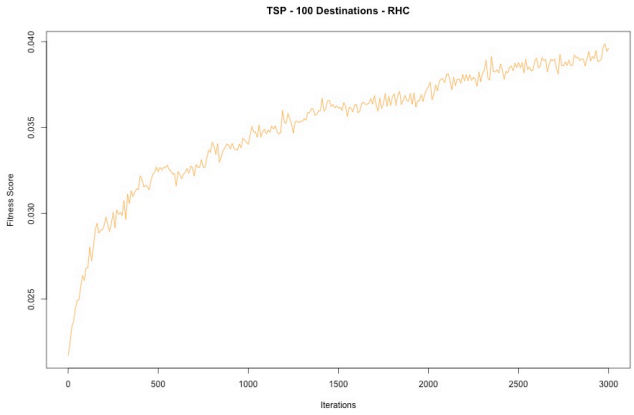
Finally, the Traveling Salesman Problem saw its best performance from the Genetic Algorithm. This makes sense since the fitness function may not be smooth and the crossover of the algorithm allows it to experiment with different combinations of short path snippets. The best parameters for the algorithm were crossover of 30 and mutation of 10. With such a low mutation, most of the benefit of the algorithm came from crossing over beneficial path subsets. Simulated Annealing also benefitted from being able to explore the search space more widely, seeing its best result with a fairly low cooling coefficient of 0.35. MIMIC did not fair well in this situation since the joint distribution did not provide much information. Although there may be relationships between connecting points and the inverse distance we measure as fitness, these relationships are not as important as the total path length which cannot be informed by the connection between any two values of an instance.

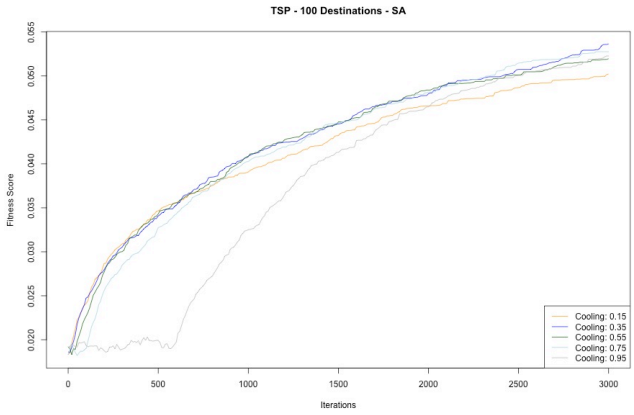
Part 2.6 - Time

Although the plots above show fitness as a function of iterations, this can be misleading. Each fitness function evaluation takes a differing amount of clock time depending









on the algorithm. The algorithm that takes the longest to run is MIMIC. At each step there is a sampling from a population, dropping and an estimation of a probability distribution. It is a computationally expensive task, but it does allow the algorithm to converge in fewer iterations than others. Conversely, an algorithm like Hill Climbing runs very quickly. It simply checks its two neighbors, decides if one is an improvement and continues on. With this increase in speed however, we lose valuable information about the distribution of our data.