RANDOMIZED OPTIMIZATION REPORT

1. FINDING NEURAL NETWORK WEIGHTS USING RANDOM SEARCH ALGORITHMS

In this section, we will use the Randomized Hill Climbing, Simulated Annealing, and a Genetic Algorithm to find good weights for the Neural Network we implemented in Assignment 1 for the Adult Census dataset.

DATASET AND METHODOLOGY

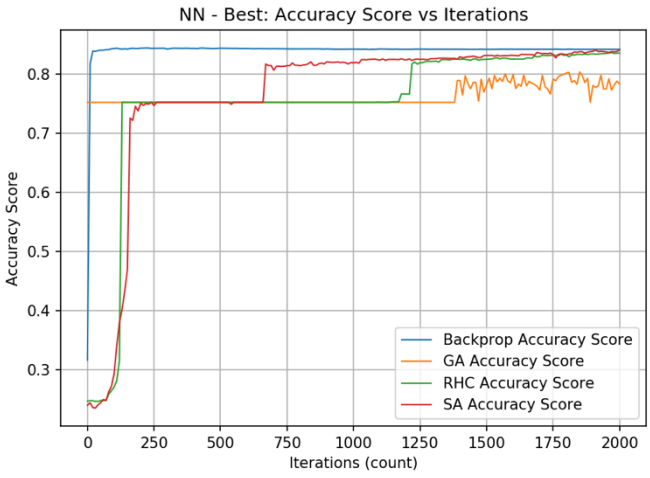
The goal for this binary classification dataset is to predict if a person earns more than 50K a year. For the sake of comparison with the solution obtained using Backpropagation from Assignment 1, we maintain the same neural network structure with 5 nodes in the first hidden layer, and 2 in the second. We use the ABAGAIL library to obtain weights for our neural network with the optimization algorithms.

RESULTS

As a baseline for comparison we present the accuracy curve in blue (figure below) for our neural net with backpropagation using ABAGAIL's implementation. The way backpropagation works is that it calculates the error at the output layer and then propagates it backward through the network's hidden layers updating the weights at each iteration. In other words, weights are updated at each iteration to reduce the output error. Each algorithm was run for a total of 5000 iterations, and most results can be highlighted within the first 2000 iterations as displayed in the figure below. Using backpropagation, we obtained 84.41% accuracy which took the least number of iterations and time to get to that value. This result gives us almost the same accuracy as the one we obtained using Scikit from Assignment 1.

Algorithm	Best Accuracy	Best Num. Iterations	Best Time (sec)
Backpropagation	0.844112	220	5.055250
Simulated Annealing	0.840574	1910	22.423487
Random Hill Climbing	0.836594	1970	22.424033
Genetic Algorithm	0.803427	1850	747.603629

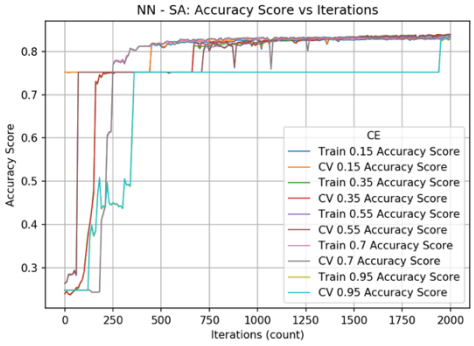
RANDOMIZED HILL CLIMBING



This optimization algorithm starts by searching the space with an arbitrary solution, and then makes incremental steps towards a better neighbor. However, it is susceptible to getting stuck in local optima for which random restarts help to get it to the global optimum. The way RHC works is that it generates a NN with random weights and sees if it has a better fitness (sum of squared error) than the previous one. If it does, then it takes it. A neighbor is any network whose weights are slightly different from the current NN, so if a better neighbor is found, it then makes a switch. From the plot we can observe that the green RHC Accuracy curve is stuck at about 75% accuracy for the first 1200 iterations. In that range until 1200 iterations, RHC is stuck in a local optimum, but as the number of random restarts and iterations increase, it eventually converges closer to the accuracy value backprop obtains: 83.65% accuracy in 1970 iterations. Because backprop uses gradient information to update the weights rather than random moves with smaller error as RHC, it converges much faster. If we had a very large number of hidden nodes, this would make this algorithm struggle as it would take many iterations to converge.

Simulated Annealing

This optimization algorithm helps us approximate the global optimum of a function by performing random walks when temperature is very high, and as temperature slowly decreases and approaches zero, it climbs the hill of the optimal solution. In that sense, simulated annealing can reach to the optimum point if temperature is decreased slowly enough. In the context of our weight search optimization problem, the way it finds better weights is similar to what RHC does, however since it makes random moves at the beginning when the temperature is high, it tends to get to a better weight network much faster avoiding being stuck in a locally optimum solution. From our plot we can observe that our red SA accuracy curve (previous page) until about iteration 750 gets an accuracy of about 75% while it makes these random walks and moves with a certain probability towards a better accuracy, and after iteration 750 it climbs up to a hill with an accuracy of 80% and beyond. As the temperature keeps decreasing and the number of iterations is increasing, it keeps approaching towards the optimum solution rather than moving towards a suboptimal one. Eventually it reaches the accuracy of 84.05% which is close to the one obtained via backprop, at iteration 1910. It is worth to point out that however both RHC and SA eventually get to 84% accuracy at 2000 iterations, SA gets a higher accuracy much faster within fewer iterations (at about 700 iterations) than RHC does (at about 1200 iterations). To get improved performance for simulated annealing, we experimented with Temperature and the Cooling Exponent values `0.15`, `0.35`, `0.55`, `0.70`, `0.95` and found that a cooling rate of `0.35` along with a temperature of `1E10` gave the highest training and validation set accuracy for 5000 iterations. In fact, after experimentation we would like to note that no matter what cooling rate was used, SA eventually converges to the optimal value with varying number of iterations as seen in the image to the right with higher cooling rates taking longer to converge than smaller cooling rates.



Genetic Algorithms

This optimization algorithm helps us approach an optimal solution by generating a population of points at each iteration and then taking the best points from the population to approach the optimal solution by performing selection, crossover, and mutation (which are the GA parameters we tune for finding our weights). In the context of our weight finding problem, GA would score each neural net, sort by accuracy and keep the most fit networks (selection) to breed children (crossover) and randomly mutate some of the weights for the network (mutation). Performing this operation multiple times, it would eventually converge to finding a network from the evolved population with optimal value. In our case it took 1850 iterations (orange accuracy curve in previous page) to obtain a comparable accuracy value of about 80%. However, it is notable to mention that each iteration for GA took much longer than SA and RHC. To get improved performance for genetic algorithms, we experimented with selection, mutation, and crossover values `(50, 10, 10)`, `(50, 10, 20)`, `(50, 20, 10)`, `(50, 20, 20)` and found that `(50, 20, 20)` rate gave the highest training and validation set accuracy for 5000 iterations.

Conclusions

- Finding weights for a neural network is an optimization problem. Though it is certain that the randomized optimization algorithms that we implemented can be used to find good weights for neural networks, for our dataset backpropagation is better as it achieved the highest accuracy in the fewest number of iterations in comparison with the other implemented algorithms.

- Backpropagation is better because it minimizes the loss function between the expected and the generated output by the network, and then backtracks that information to update the weights. It leverages the gradient to update the weights, which makes it converge faster to an optimal set of weights.

- All randomized optimization algorithms implemented for finding weights for our neural network got us to a similar accuracy value. This explains that there is a single global optimum take makes all these random search algorithms converge to a similar set of weights.

- Finally, as we can observe, all three randomized optimization algorithms gave weights that were close to what backpropagation got, and in fact gave us results close to what the other Supervised Algorithms we implemented from Assignment 1 obtain. It appears based on our results that using randomized optimization algorithms can help us solve a problem differently, but do not necessarily help us do any better, because there seems to be an upper limit, no matter the algorithm, on the learner we can create.

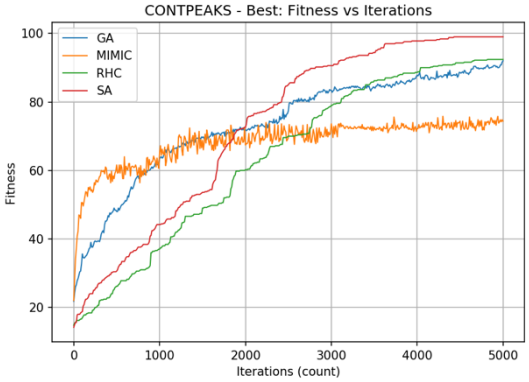
2. Optimization Problems

When we talk about optimization problems, we are looking to maximize a *fitness* function. In this section we will tackle three optimization problems and for each we will implement four randomized search algorithms: Randomized Hill Climbing, Simulated Annealing, Genetic Algorithms, and MIMIC. We will compare all the algorithms and see which works best for the particular optimization problem we are analyzing. In the previous section, we have described a general idea of what Randomized Hill Climbing, Simulated Annealing, and Genetic Algorithms are. Before we delve deeper into the optimization problems analysis, lets describe what is MIMIC.

MIMIC is an optimization algorithm that conveys the structure of a problem by performing probability distribution estimations and refining a model such that it maximizes the fitness. MIMIC samples a probability distribution, keeps the most fit samples above a certain percentile, estimates a new probability distribution which conveys structure, and repeats the process. The version of MIMIC implemented uses dependency trees to represent the probability distribution.

2.1 Continuous Peaks Problem

The continuous peaks problem consists of multiple peaks with multiple local optima and a global optimum. The goal is to find the highest peak. For our particular implementation, we want to find the highest peak within 100 peaks. In the figure to the right we can observe the result of running our four optimization algorithms plotting the fitness vs the number of iterations. The goal is to maximize the fitness. As we can observe, Simulating Annealing yields the best result. Let's go into a detailed analysis for the behavior of each algorithm for this problem.



Randomized Hill Climbing

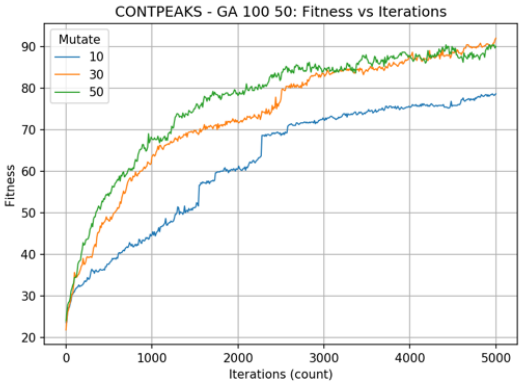
RHC achieves a decent performance, however it does not converge to the best fitness in the same amount of iterations SA does. RHC would eventually get there but will take longer. This is because it can get unlucky and be stuck in local optima, given that there are many basins of attraction and only one contains the highest peak. Therefore, it needs random restarts. This would take longer to converge. If there were few basins of attraction, and few local optima, then RHC would achieve a result closer to SA in a similar number of iterations (as in the *Flip Flop problem* described in the next section).

Simulated Annealing

As mentioned earlier it achieves the best fitness for this problem. This result makes sense because SA is made to avoid local optima by making random movements between hills when the temperature is high, and then climbing up the best hill when temperature decreases slowly. Therefore, as the temperature decreases, it eventually finds the best hill and gets to the optimal peak.

Genetic Algorithms

GA is sensitive to the initial population initialization. For example, if the two parents are located at a small hill with a local optimum, then their child could be a point in that same hill that is introduced into the population. Hence in this case GA would be climbing up a hill with a suboptimal peak with the generated population. It is possible that while performing crossover, GA can discard the best answer while evolving the population. Alternatively, a higher mutation rate can help us overcome this problem. In fact, as we can observe in our plot to the right, the higher the mutation rate, the higher the fitness. It is certain that we can get decent results, but each iteration takes more time in GA, and the nature of this problem due to crossover/mutation used by GA will make it take more iterations to converge than SA for this problem.



MIMIC

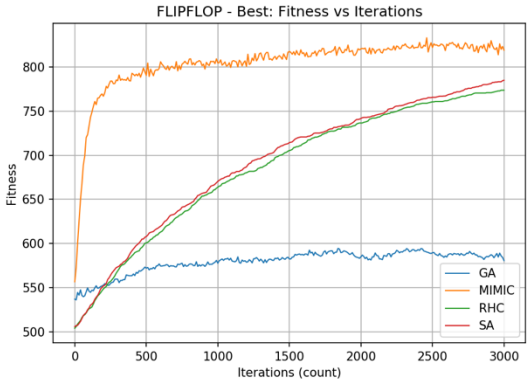
As we can observe in out plot at the beginning of this section, MIMIC obtained the lowest fitness. Since it takes the most fit samples to calculate the probability distribution at each iteration, if the basin of attraction is small, there may be very few values that are sampled from the highest peak basin and may not be part of the final distribution. Therefore, it does not perform well when there are a lot of peaks and we have to get to the best one. However, it does perform good on problems with small peak numbers, like *4 Peaks*, because more points can be sampled from the entire underlying distribution and an accurate probability distribution could be estimated.

Comparison

Algorithm	Parameters	Best Fitness	Best Iterations	Best Time (seconds)	Best Fevals
SA	0.55	99.00	3620	0.009089	3631.0
RHC	None	93.32	4040	0.013998	4051.0
GA	100 50 30	93.16	4230	0.727013	266919.0
MIMIC	100 50 0.9	78.83	4240	36.965709	467400.0

SA performs best for this problem and takes the lowest number of iterations and time in seconds to get to the best fitness. Also, it took the lowest number of calls to the function (fevals) to calculate the fitness value, about one per iteration. RHC and GA on the other hand took more iterations to get a fitness that was lower than the one SA got. Despite RHC and GA taking almost the same number of iterations, GA took more time. Given more iterations, RHC and GA could eventually get to the value SA got. MIMIC got the lowest fitness and took the most time among all of them. MIMIC converged to a suboptimal distribution which did not contain the highest peak, which makes it the worst algorithm for this problem.

2.2 Flip Flop Problem



This problem consists of a bit string with 0 and 1. Our goal is to maximize the alternation between zeros and ones. In the figure to the right we can observe the result of running our four optimization algorithms plotting the fitness vs the number of iterations. As we can observe, MIMIC yields the best result. This is because to maximize the fitness, we need the information of the previous bit to then determine the following bit and increase the fitness. MIMIC helps us represent that dependency with the underlying distribution. Let's go into a detailed analysis for the behavior of each algorithm for this problem.

Randomized Hill Climbing

The way RHC would work in this scenario is changing one bit at a time and see which is better and take a step in that direction. Given a fixed length bit string, there is one solution and its inverse that maximize the flip flop alternation. Given that, there are only two hills to climb on, with each peak being the optimal value (two global optima). Getting to any peak will get an optimal solution. However, since only one bit is changed at a time, it will eventually get to the best fitness at one of the peaks, but it will take very long.

Simulated Annealing

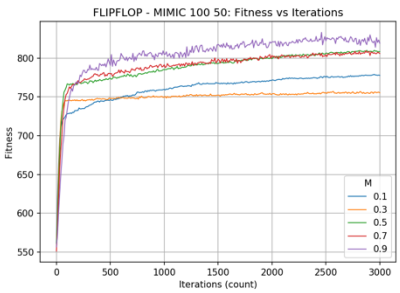
Given the nature of this problem as described in the RHC analysis, there is just one of the two hills we have to climb on to get to the best fitness. Therefore, the result that SA provides is almost the same to the one that RHC provides, as this algorithm would likewise make one bit change to get closer to the optimal value at each iteration. Making random movements at the beginning when temperature is high would not make much of a difference, since it will be bouncing between both hills with similar fitness, and when temperature decreases it will just stick to only a single hill to climb on in the end and get the fit value at the peak. Just as RHC it would eventually get there, but it will take more iterations.

Genetic Algorithms

GA performs the worst on this problem. This is because the way GA performs crossover between two fit parents would not necessarily increase the fitness of the child. This is because the fitness of a bit string depends on the entire string and not parts of the fit ones. Their combination would not necessarily yield the best solution. Later when we perform mutation, we just change one bit which would be a similar result as RHC, however it would not be enough to take it to a better population due to what crossover does. Doing this many time seems to converge to a population that has a very low fitness as can be observed in the plot above.

MIMIC

MIMIC performs the best for this particular problem. It converges faster because it approximates the distribution given a bound on the cost function. And as the we keep iterating; the optimum value cost becomes more likely as we sample points from the new distribution. For the flip flop problem, we need the entire bit string sample to know if it's good or not. Also, it is important to know if the current bit is a 0 or a 1 and then use that information for selecting the neighboring bits to maximize the flip flop fitness. MIMIC would then sample from the current distribution to generate the updated distribution. This bit dependency can be represented with the distribution generated, because it looks for the highest dependency via the maximum spanning tree, which gives us the final dependency tree. As it improves the probability distribution it narrows down the space we need to sample to, hence the fittest says at the end.



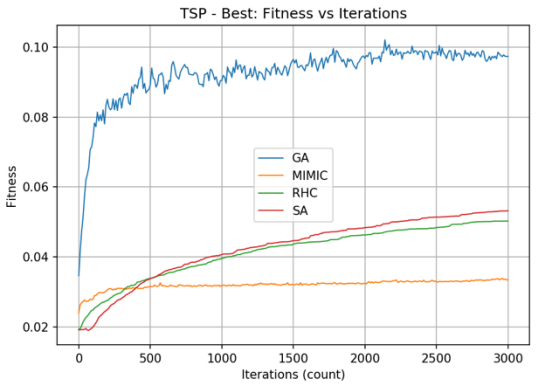
As a side note, as we can observe in the image at the right, keeping 90% of the samples for each iteration and reupdating the distribution takes us to the highest fitness. MIMIC converges closer to the higher fitness value within the least number of iterations compared to the rest.

COMPARISON

Algorithm	Parameters	Best Fitness	Best Iterations	Best Time (seconds)	Best Fevals
MIMIC	100 50 0.9	834.8	2370	1755.024190	261700.0
SA	0.55	787.8	3000	0.040844	3011.0
RHC	None	776.8	2960	0.039244	2971.0
GA	100 30 50	596.0	590	0.561168	34712.0

MIMIC gets the best fitness among the rest of the algorithms with the lowest number of iterations compared to SA and RHC. On the other hand, GA converges to an inferior fitness value within fewer iterations. In terms of wall clock time, despite MIMIC getting the best fitness, it takes the most time due to calculating probability distributions, generating dependency trees, and sampling at each iteration. On the other hand, RHC and SA take the least time, more much more iterations than MIMIC and get to a fitness value that is not too far away from the one obtained by MIMIC, however it still will take more time and iterations to get to the best fitness value. Also, they perform the least function evaluation calls, since it's just one at each iteration.

2.3 TRAVELLING SALESMAN PROBLEM



The travelling salesman problem's goal is to minimize the distance that a salesman needs to travel to visit all cities and get back. In the figure to the right we can observe the result of running our four optimization algorithms plotting the fitness vs the number of iterations. As we can observe, the Genetic Algorithm yields the best result, because merging knowledge of each good path found can get us to an even better path. Let's go into a detailed analysis for the behavior of each algorithm for this problem.

RANDOMIZED HILL CLIMBING

As we can see in the plot, at 3000 iterations RHC has still not converged to the optimal value GA got to within that many iterations. RHC takes longer to get to a higher fitness value as it would tune one single path for the salesman at each time to get to the optimal journey. Eventually with more iterations it would get to the value.

SIMULATED ANNEALING

Similar to RHC, SA grows logarithmically (slowly) because you just pick one route and make one change at each iteration. Given a lot more iterations it would eventually get to the best fitness. However, since it is very slow, it is not the most apt algorithm for this problem.

GENETIC ALGORITHMS

This algorithm performs the best for this problem. The concept of crossover and mutation are helpful in the context of this problem. To find a better path you use a combination (crossover) of good paths you have found already (selection) with maybe some slight changes (mutation). This eventually yields a better path population that converges faster towards the best journey with highest fitness that the salesman should take. GA crossover leverages information about locality which leads us to better paths.

MIMIC

MIMIC performs the worst gets the lowest fitness for this problem compared to the other algorithms. This is because we have to sample the best paths at the beginning to find it later as we update our distribution. Initially they are not known, so MIMIC takes the best bet at choosing the most fit paths and regenerating the probability distributions. However, there is no scope for MIMIC to go and find some other alternative path that may be better, hence MIMIC converges to a suboptimal solution in this case.

COMPARISON

Algorithm	Parameters	Best Fitness	Best Iterations	Best Time (seconds)	Best Fevals
GA	100 50 50	0.102508	1440	0.472602	101299.0
SA	0.75	0.054040	2860	0.004424	2871.0
RHC	None	0.051500	2980	0.004797	2991.0
MIMIC	100 50 0.1	0.034140	2290	443.496993	252900.0

As we can observe GA gets the best fitness among the rest in the least number of iterations. In terms of time, it takes more time than SA and RHC during the first 3000 iterations but converges to the best fitness than the latter two because it has to perform crossover and mutation to evolve the population. MIMIC on the other hand gets the worst fitness and takes the most time and function evaluation calls.

CONCLUSIONS

Each of these random optimization algorithms work well for certain problems:

- *Randomized hill climbing* works well for problems where we are finding for an optimum value with incremental improvements, multiple iterations, and presence of a single global optimum and no local optimum, such as in a *Single Peak* problem. Due to its randomized nature, it is very fast. When applied to problems with several local optima, to avoid getting stuck in them, random restarts help to take it to the global optima.

- *Simulated annealing* helps us find the global optima in the presence of multiple local optima such as in the *Continuous Peaks* problem.

- *Genetic Algorithms* work well when the nature of the problem admits crossover and mutation, such that the knowledge of *selecting* two good solutions *crossing* it over and *mutating* it gets us to a better solution population. For example, choosing two fit paths and generating a more optimal one as in the *Travelling Salesman Problem*.

- *MIMIC* works well when knowing information of the state is important such as knowing if my neighbor is a 0 or a 1 to maximize the fitness as in the *Flip Flop Problem*. It is also useful when the cost of computing the fitness is very high, because it generally tends to converge in much faster in terms of iterations. However, because each iteration requires creating a distribution (using dependency trees in our case) and generating samples, each training iteration takes longer wall clock time than the above algorithms.