Assignment 2: Randomized Optimization (RO)
CS 7641
Connor Beveridge – cbeveridge3 – 902451314

Introduction

In order to explore random search, I have analyzed three algorithms and three problems which highlight the benefits and costs of each algorithm. Random Hill climbing (RHC), Simulated Annealing (SA), Genetic Algorithms (GA), and Mimic (MM) are analyzed for each problem. The first three of these algorithms are analyzed in use for finding neural network weights against gradient descent.

Three Optimization Problems

1. **The Knapsack problem** is that of finding the best combination of items to put in a sack so that the total values of those in the sack is maximized while not exceeding the max weight capacity of the sack. In this analysis, there is only 1 of each item available. The problem state is thus represented by a series of 1s and 0s indicating an item is added to the sack or not respectively, thus representing the state of the sack. This problem is interesting because a maximum fitness is not known in advance and it has a conceptualizable space structure with many local maximum close to the global maximum's fitness. A randomized set of weights and values is provided by me. The maximum weight capacity of the bag is 35% of the total weight of all objects.
2. **The Random Rule** problem is a problem I have come up with myself. It is simply a series of *i* 1s and 0s which represent true or false for each rule R_i where each input bit X_i is the value for that rule. It is not known to the person desiring optimization if there are dependencies between a few of the rules. For simplicity in this analysis there is not, but the person does not know that, so they assume there may be some local maxi. The true value for each rule is given in the fitness function and each correct rule gives one point. The maximum fitness is thus the number of rules *i*. This search space is interesting because although it is very even, there are no local maximum at all in the i-dimesnional Euclidian space. The high dimensionality however still makes the problem non-trivial. The search space can be visualized somewhat is a multi-dimensional pyramid with no flat points except for the peak and no negative slopes at all.
3. **The Love 7s Problem** is another problem I have come up with myself. The main thing that is interesting with this problem is that it has a very time expensive fitness function. In this problem, your employer likes numbers, and he wants you to bring him good numbers. Any set of numbers 1-9 will do. You are not sure what 'good numbers' means so you must learn what numbers he likes based on ratings of the sets of numbers you give him to get a promotion. He is not aware because he is an amnesiac, but he had a bad gambling problem in the past and just really like 7s. Each 7 in the set gives one point and other numbers give 0 points in the fitness function. It takes you one minute to show your employer a set of numbers, get the fitness, and enter it back into the computer. This minute time delay is incorporated into the analysis. This problem is also interesting because it has single consistent rule that applies to all inputs. This fitness function is implemented in code comments and not run for obvious time constraint reasons, but all other processing times per iteration become negligible compared to the minute-long fitness evaluation.

The Knapsack Problem Analysis (Genetic algorithm Strength)

1. Initial hyperparameter search

A custom implemented grid search was done on a 30 input problem on all 4 algorithms to determine optimal parameters for each. Convergence checking values were found after the grid search for optimal fitness within 1000 maximum iterations for all algorithms. which is simply the number of times the algorithm keeps checking for a better state before it returns its optimal state. Sometimes, many iterations are needed to find a better state in all algorithms. The maximum number of attempts was set at a basic 10 and increased until the algorithms maximized the problem. In order to reduce computation time, I picked search ranges I thought were most likely best.

For RHC, the best number of restarts was found to be 1000. This makes sense because the search space is very bumpy with many local maxima. An initial state space of an empty bag was found best as random initial states often put the state coordinate in a vast flat minimum which is difficult to get out of.

For SA, an initial temperature of 10 and decay rate (geometric) of .998 where found best with grid search. This makes sense as the search surface is very bumpy, so much slower cooling and high initial exploration tendency is needed to settle into the right maximum. A convergence checking count of 50 was needed. An empty bag initial state was also best for the same reason as random hill. This high temperature makes sense as it gives the algorithm a better chance to settle into the global maximum.

For the GA, a population size of 100 and mutation rate of .1 where found best with gridsearch. A max convergence of 15 was needed to maximize. This population is equally well as bigger populations and so was selected.

For MM, a large population size of 900, and a small keep percent of .05 was best.

2. Algorithm Comparison Analysis

Given the hyperparameters found best in the grid search, an analysis was done on time and iteration count to reach maximum fitness. Maximum fitness is unknown for this problem, however, is assumed to be the highest value given by these algorithms, which was 187 given by the genetic algorithm. The results of the analysis can be seen in **figure 1a**. All algorithms were constrained to 1000 iterations, and a limited computation time. Given these constraints, random hill climbing, and simulated annealing did not achieve a maximum even with the best hyperparameters. The times for each individual execution on an average of 5 runs for each algorithm are shown in the legend. SA was the quickest by far with MM being the slowest by far. **Figure 1b** shows the number of iterations needed to maximize fitness with each algorithm given smaller problem sizes than the initial analysis so that all algorithms would maximize, however still, SA did not maximize in all cases, and it had significantly many more iterations although it is the fastest algorithm for all problem sizes. GA was determined to be the best algorithm for this problem.

2.1 RHC

Given time and iteration constraints, RHC was not able to achieve maximum with 30 items surely even with 1000 restarts because the search space is so large with very many local maxima that many thousands of restarts may be required. RHC is so much slower than SA here because it does so many restarts while SA does none. Given the smaller search space of the bag sizes as shown in figure 1b, it was able to maximize with only a small linear increase in iterations needed. This is because 1000 restarts are still used for these small problems and so in many cases the max may be found without any climbing needed. As the number of items grows, the search space grows exponentially and so RHC becomes pretty useless for such a massive and bumpy fitness surface. In figure one the curve shown is for the best run out of the 1000 restarts, and so the true number of iterations for this algorithm are closer to 50,000. Each restart begins with an empty bag and adds an item in randomly until no more items can fit with the room left. Because of this, it will search many areas of the search space repeatedly.

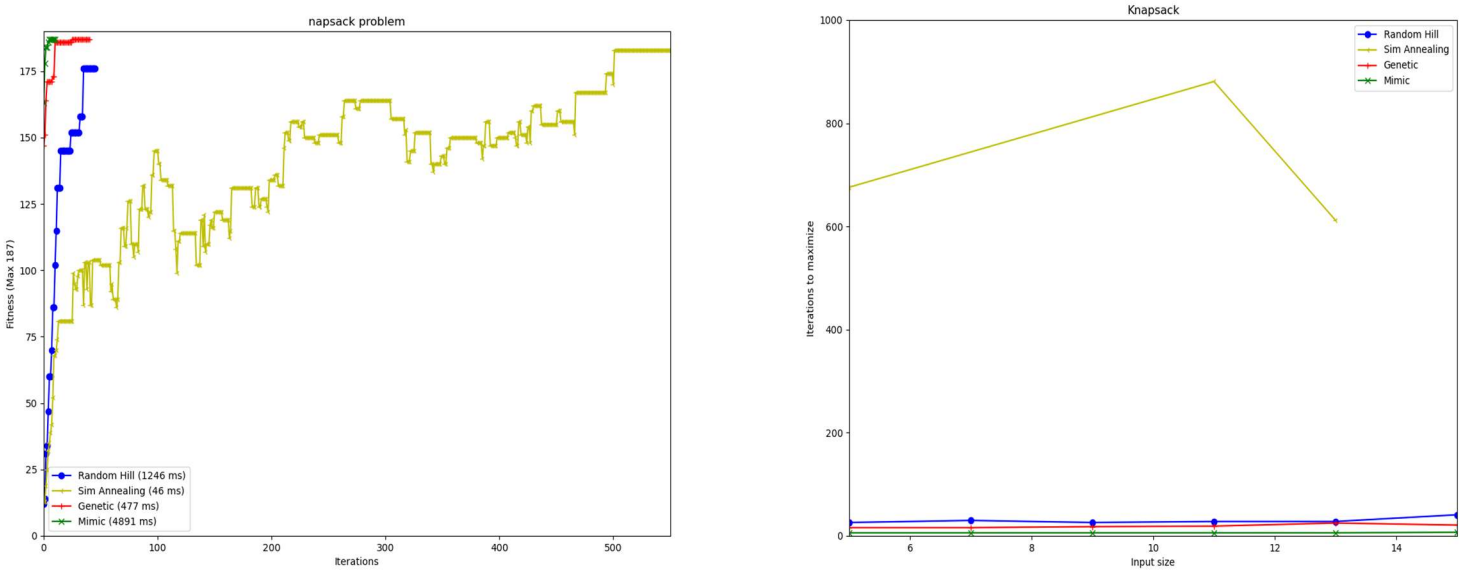

Figure 1 | RO application to Knapsack Problem. **A.)** Fitness curves for each algorithm are shown for the 30 input size case with MM and GA achieving the maximum of 187. The number of iterations for RHC are for the best restart run out of 1000 **B.)** The number of iterations to maximize fitness for each problem size are shown. Maximum fitness is determined by best performing algorithm, in this case GA.

2.2 SA

Given iteration constraints, SA was close but did not maximize with iteration constraints. It needs so many iterations because it does not restart like RHC. Its benefits over RHC for large, bumpy search spaces is illustrated in figure 1a, performing better with much less time. This is because it is not almost destined to fall into one of the thousands of local maxmima like RHC and theoretically should come to rest at the global max given enough time and optimal hyperparameters. In the 30 item case it does get stuck in a local max but would surely find the global one if given more iterations. Surprisingly, it did not maximize for some of the smaller problem sizes given more iterations, but I attribute this to the large initial temperature used which is likely not suitable for smaller search spaces. It starts with an empty bag and slowly adds, and unlike RHC, removes items. This removal affords it the freedom to not restart.

Many areas of the search space will still be searched repeatedly, but more benefit derived as more time is spent at high fitness areas of the space instead of restarting at 0 often as with RHC.

2.3 GA

GA was found to be the best algorithm, finding the maximum and doing so much faster the MM which also found the maximum. It did not need a large population to do so and it makes sense why it is so good. This is because there are surely combinations of items that have high value to weight ratios. Different combinations will bubble up in the population and before to long the optimal combination of will emerge. The extremely bumpy fitness surface that plagued RHC and SA does not hinder GA at all. An increase in problem size does not seem to require more iterations. In figure 1a you can see at the first iteration, there exists an individual in the population with very high fitness (~150).

2.4 MM

Mimic was very effective at finding the maximum in few iterations but took so long that it can easily become unreasonable to run. Because of the large population size, an individual in the first iteration has high fitness.

Random Rule Problem (Simulated Annealing Strength)

1. **Initial hyperparameter search**

A custom implemented grid search was done on a 50 input problem on all 4 algorithms to determine optimal parameters for each. Convergence checking values were found after the grid search for optimal fitness within 1000 maximum iterations for all algorithms. The default value of 10 is used for grid search. The search space for this problem has very high dimensionality but no local maxima and is best thought of as a multidimensional pyramid surface. Although as mentioned, the person who desires optimization does not know if there are some dependencies between the rules so they assume there could be some local maxima.

For RHC, the best number of restarts was found to be 100. This makes sense because the dimensionality is high and the search space has 2^50 coordinates.

For SA, an initial temperature of 2 and decay rate (geometric) of .98 where found best with grid search. These hyperparameter values however were just the first in the grid search to achieve the max of 50, so other values could be much faster. I would think lower temperature would be best, maybe 0 temperature would be best in this case because climbing up at every iteration is best.

For the GA, a population size of 30 and mutation rate of .5 where found best (fastest as most parameters maximized) with a grid search. This makes sense as although the problem size is relatively large, each rule represented by a ‘gene’ in the DNA will always improve fitness with the right value so the right values will pop up fast with a high mutation rate and proliferate; then soon some item in the population will achieve the max, even with a small population.

For MM, a smaller population was fine of 200 with a keep percent of .1.

2. **Algorithm Comparison Analysis**

Given the hyperparameters found best in the grid search, an analysis was done on time and iteration count to reach maximum fitness. Maximum fitness is simply the problem size for this problem. The results of the analysis can be seen in **figure 2a**. All algorithms were constrained to 1000 iterations, and a limited computation time. Given these constraints all algorithms achieved a maximum. The times for each individual execution on an average of 5 runs for each algorithm are shown in the legend. SA was the quickest by far with MM being the slowest by far. **Figure 2b** shows the number of iterations needed to maximize fitness with each algorithm given smaller problem sizes than the initial analysis so that all algorithms would maximize. Simulated Annealing is determined to be the best algorithm for this problem because of its speed and because there is an assumption there could be some local maxima (even though there is not in this implementation. I would think RHC would have been better if the person knew there was no local maxi, but it also did poorly with no local maxi since the problem space is so large (2^{50} coordinates), it takes much more time to climb the massive multidimensional pyramid. The algorithm is just faster given it does not restart. SA also seems to do a better job of finding a better state when getting towards the top of the pyramid.

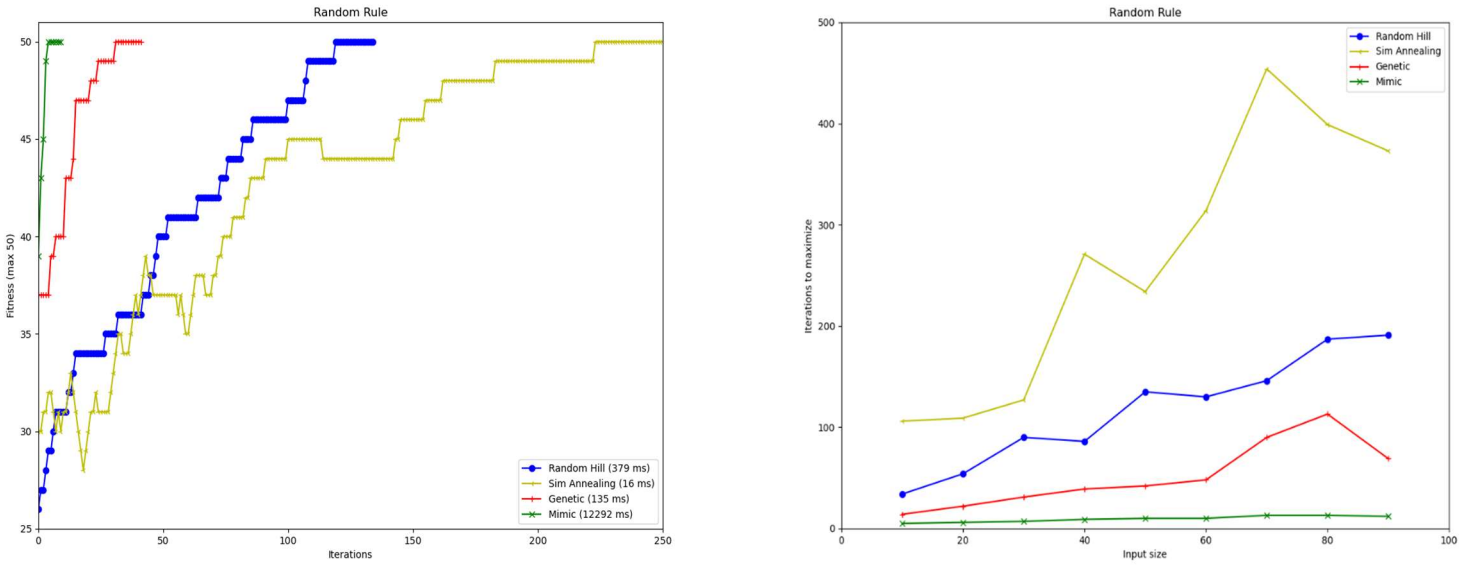

Figure 2 | RO application to Random Problem. **A.)** Fitness curves for each algorithm are shown for the 50 input size case with a maximum of 50. The number of iterations for RHC are for the best restart run out of 100 **B.)** The number of iterations to maximize fitness for each problem size are shown.

2.1 RHC + SA

This problem is great for RHC and SA with a low temp and fast decay as there are no local minima. It makes sense that iterations needed increases with time in an apparently linear fashion as the rules to solve grows linearly. Figure 2a illustrates how when the algorithm cools, it is much less likely to backtrack down the pyramid like it does initially. In regards to time, SA just does maximize much quicker, and it seems to be in the way the algorithm checks for adjacent better states.

2.2 GA + MM

While these algorithms performed well on this problem, they do too much for what is required to solve it, thus taking a large amount of unnecessary time.

Love 7s problem

1. Initial hyperparameter search

A custom implemented grid search was done on a 10 input problem on all 4 algorithms to determine optimal parameters for each. Convergence checking values were found after the grid search for optimal fitness within 1000 maximum iterations for all algorithms. The default value of 10 is used for grid search. The search space for this problem is fairly large but with lower dimensionality. What makes this problem special is not its structure as it is similar to the random rule problem, but its fitness function, requiring one minute for each evaluation.

For RHC, the best number of restarts was found to be 100. This makes sense because the dimensionality is high and the search space has 10^10 coordinates.

For SA, an initial temperature of 5 and decay rate (geometric) of .99 where found best with grid search. These hyperparameter values however were just the first in the grid search to achieve the max of 10.

For the GA, a population size of 10 and mutation rate of .6 where found best (fastest as most parameters maximized) with a grid search. This makes sense as the problem size is relatively small, and each 7 represented by a 'gene' in the DNA will always improve fitness by one so and so 7s will pop up fast with a high mutation rate and proliferate; then soon some item in the population will achieve the max of all 7s, even with a small population.

For MM, a smaller population of 600 with a keep percent of .025 was best.

2. Algorithm Comparison Analysis

Given the hyperparameters found best in the grid search, an analysis was done on time and iteration count to reach maximum fitness. Maximum fitness is simply the problem size for this problem. The results of the analysis can be seen in **figure 3a**. Run times were approximated given each fitness evaluation takes 1 minute, and so each iteration will be approximately 1 minute, then the run time is equal to the number of iterations in minutes. All algorithms were constrained to 1000 iterations. Given these constraints all algorithms achieved a maximum. The times for each individual execution on an average of 5 runs for each algorithm are shown in the legend. Contrary to the other 2 problems, MM was the quickest by far with SA being the slowest by far. **Figure 3b** shows the number of iterations (and minutes) needed to maximize fitness with each algorithm given smaller problem sizes than the initial analysis so that all algorithms would maximize. MM is determined to be the best algorithm for this problem because of its vast speed superiority. Given the need to have a person do the fitness scoring, the other algorthms could lead to impossible time requirements while mimic can shorter the needed time down to minutes. Thus, MM is essential to optimization problems requiring high fitness evaluation times.

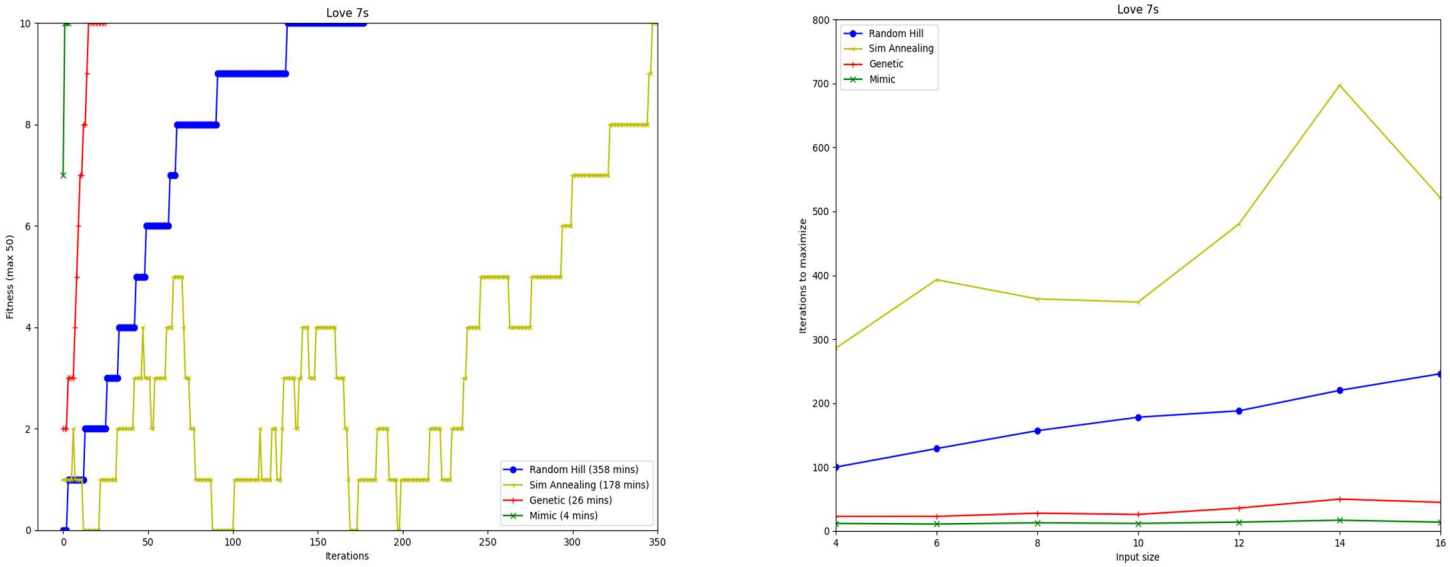

Figure 3 | RO application to Love 7s problem. **A.)** Fitness curves for each algorithm are shown for the 10 input size case with a maximum of 10. The number of iterations for RHC are for the best restart run out of 100 **B.)** The number of iterations to maximize fitness for each problem size are shown.

2.1 RHC + SA

This problem would great for RHC and SA if it were not due to such a long fitness eval time. It makes sense that iterations needed increases with time in an apparently linear fashion with 10 more possible values to check for or search for each additional size increase. Figure 2a illustrates how when the algorithm cools, it is much less likely to backtrack down the pyramid like it does initially, and actually goes back to 0 fitness a few times in the first half of the run. Because these algorithms need so many iterations, they are impractical for such eval times.

GA + MM

While MM has performed equally well as GA in the previous problems, its drawback has been its very long run times. In this case however, it has the smallest run time by far given it maximizes in so few iterations. This is very valuable quality to have as it is unique to these algorithms and the others can be no help for similar long eval time problems.

RO Problem Conclusion

From this analysis, I have come to conclusions on the strengths and weaknesses of these 4 RO algorithms. The strength of MM is found in its power in maximizing fitness in few iterations compared to the other 3 algorithms when fitness evaluation is time expensive. It can make impossible problems due to time constraints possible. However, its weakness is time related too, often taking an impractical amount of time when fitness evaluation is cheap compared to the others. Putting aside long run times, it does appear to be just as good as GA at optimizing large search space, structured problems with a great

many local maxi. This must be because it is able to find structure in the problems, model them according to probability distributions, and thus have 'memory' derived from states it has previously processed. The random rule problem, although still solved in few iterations, had the highest relative iterations needed for MM compared to the other algorithms. This is likely because it does not have much structure or applicability to probability other than 100% and 0% true. Which is still something MM handles, it just must not help it too much on this type of problem with minimal relationships between inputs.

GA proved to be best when the fitness surface is very large and very bumpy with way too many local maxi for simulated annealing to settle into when considering time constraints. The bumpiness of the fitness surface did not hinder GA in its search for the global max. GA seemed to perform well when each bit of the input had a direct, constant influence on the fitness and when gestalt properties were present so that bits together create 'modules' that can be passed down to the next generation. GA appears to be able to less hindered by very bumpy fitness surfaces by keeping track of many good candidate coordinates at once.

SA appeared best when there are a 'small' to 'medium' number of local maxima. Given the state space in this analysis that range is estimated to be about 50-5000 local maxima. When the fitness surface is extremely large and bumpy, GA would be much more accurate and finding the global max. With SA it can be hard to have confidence you have truly found the global max. You could waste a lot of time tuning SA and letting it run and extra long time to help find the global max, when GA would be more likely to find it.

RHC revealed it usefulness in problems with a very small number, preferably 0, of local maxi and a small search space. When it needs to restart a great many times it just takes too long to run relative to SA, even when there are no local max.

Neural Network Optimization

I am comparing the tic-tac-toe endgame supervised learning problem from the previous assignment among different kinds of weight optimization algorithms reducing loss to create the best network. These include gradient descent, random hill climbing, simulated annealing, and genetic evolution. This is done in mlrose and the sklearn created network in assignment 1 is recreated in mlrose with gradient descent. A perfect network can be created with 8 hidden units, each representing a different win state all for x's or all for o's. However, I imagine an equally effective network but less complex network could be created with fewer nodes. 8 Nodes are used in this analysis. Each input to the network is the state of each letter (and blank) for each space which comes to 27 total inputs. Each of the 8 hidden nodes of the perfect network would be of the form (x_{ij} ^ x_{kl} ^ x_{mn}). This is just the 8 possible ANDS of three Xs that indicate a win for x. The Output node would just need to be an OR perceptron as only 1 hidden node can output true at a time.

1. Initial hyperparameter search

A custom implemented grid search was done on the training set. Convergence checking values were determined practically as no algorithms converged fully. The hyperparameters were selected on how well they did on the validation set which is 200 of ~600 training examples. 200 of the ~800 total training samples were initially separated as a testing set.

For RHC, the best number of restarts was found to be 10. More may have been better but began to take considerably more time.

For SA, an initial temperature of .5 and decay rate (geometric) of .99 where found best with grid search. This indicates the search space does not have very many local maxima.

For the GA, a population size of 100 and mutation rate of .1 where found best (fastest as most parameters maximized) with a grid search, however with much more time available, a bigger population likely would help more.

2. **Algorithm Comparison Analysis**

Given the hyperparameters found best in the grid search, an analysis was done on loss, time and iteration count to try and reach maximum fitness. The results can be seen in **figure 4a** with time to reach the final iteration shown in the legend. The algorithms were not run to convergence, but to practical iteration and time constraints. The loss for each iteration can be seen in figure with the time to last iteration as shown in the figure displayed in the legend. Accuracies for iteration sizes for each algorithm are shown in **figure 4b** with the time to reach peak accuracy shown in the legend.

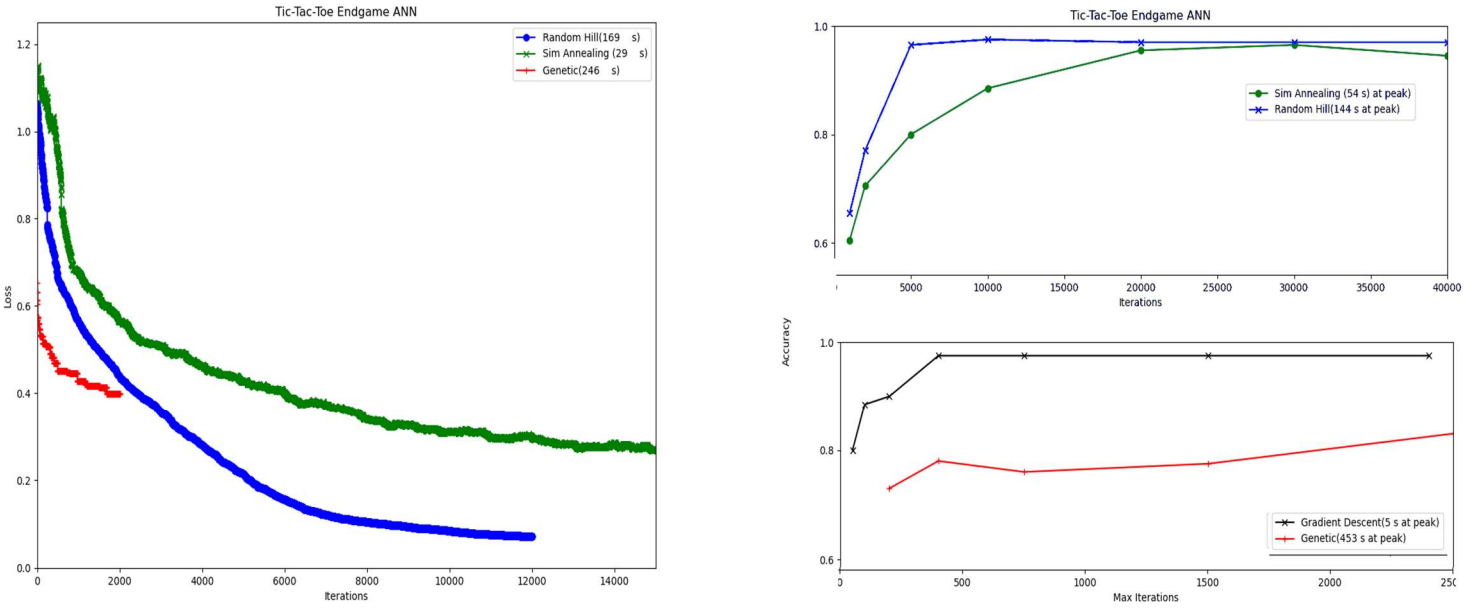

Figure 4 | RO application to Neural Net. **A.)** Loss curves for each algorithm are shown for the Tic Tac Toe dataset The number of iterations for RHC are for the best restart run out of 10 **B.)** The Accuracy over number of iterations is shown on the test set. The max range for the lower graph is 2500.

2.1 RHC + SA

All three algorithms appear to be approaching an optimal set of weights but were cut short due to time constraints. They actually appear to be settling into a local optimum. Although RHC performed better with a constraint of 15000 iterations, SA performed much faster. It takes much longer to run these algorithms than with

just gradient descent as they have a very large, technically infinite space to search. While these two algorithms achieve decent accuracy compared to gradient descent, that take much longer to run. This is because gradient descent is specialized for continuous search spaces, and these two algorithms are better for discrete. They have a step size of .1, and with 27 inputs with weights from 0 to 1, the state space has 2^216 coordinates which is a very large space to explore. With enough time however, they may reach the global max. In figure 4b you can see that as as the number of iterations is increased as compared to figure 4a, both the algorithms start to overfit.

2.2 GA

GA did the worst out of these algorithms, simply because the state space is so large, there is lots of aimless searching, while in SA and RHC, there is a steady hill to climb. GA does seem to be approaching the optimum however, it is just a more time intense algorithm for such a massive state space with likely few local maxima as compared to the other two. It takes more time and is less accurate overall, however just looking at iterations and not time, it is converging better.

3. Conclusion

Gradient descent performed much better than all of the other algorithms for this problem. These sets of algorithms are fundamentally different as gradient descent checks the slope of the current state to determine the changes that should be made while the others check the actual neighbors to see if they have less loss. The other ROs move from state to state with a limited step size instead of one that is determined by the slope. Instead of just changing one weight by .1 , gradient descent can change many weights by a variable step size for each weight, leading to much quicker convergence, especially with steep slopes.

The fact that there was overfitting with many iterations allowed may indicate that the global optimum solution I mentioned earlier is not achievable with the data given due to the presence of rare outliers, however with more data, all of these algorithms should be able to achieve it, with gradient descent being the best as it is suited to continuous search spaces.

Based on this analysis, I determine that gradient descent is best suited for smooth curvy fitness surfaces (continuous), while the other algorithms are better for fitness surfaces without smooth curves (discrete). If the optimal solution has weights that can be found with large step sizes, the other RO algorithms would be much quicker and effective at maximizing. However, even with step size of .1 in this analysis, they were still much too slow.