OMSCS_CS7641 HW2
Randomized Optimization

Clement Li (cli620) Fall 2020

Introduction:

The purpose of this project is to explore random search on discrete optimization problems. These techniques include 1) randomized hill climbing (RHC) 2) simulated annealing (SA) 3) genetic algorithms (GA) and 4) MIMIC. Optimization problems means fitness functions trying to maximize return.
There are 2 parts to this analysis: 1) three problems will be introduced to the randomized optimization methods to distinguish the benefits/flaws of each and 2) evaluate the randomized optimization methods would be used as a backprop to a neural net. This analysis paper will leverage the mlrose library, particularly some work that ezerilli [1] and make modifications to his code and generate my own results based on different problem sets.

Part 1: Random Search Algorithms

This section explores the pros and cons of four random search algorithms on finding the optimized solution. These are randomized hill climbing, genetic algorithm, simulated annealing, and MIMIC. The problems being introduced to these algorithms are the traveling salesman problem (TSP), four peak problem, and max k-color problem.

Traveling Salesman Problem

Problem Description

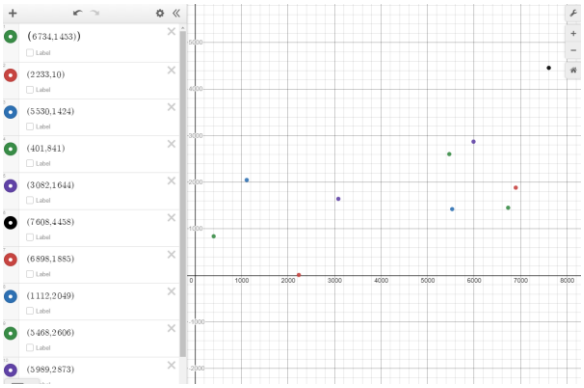
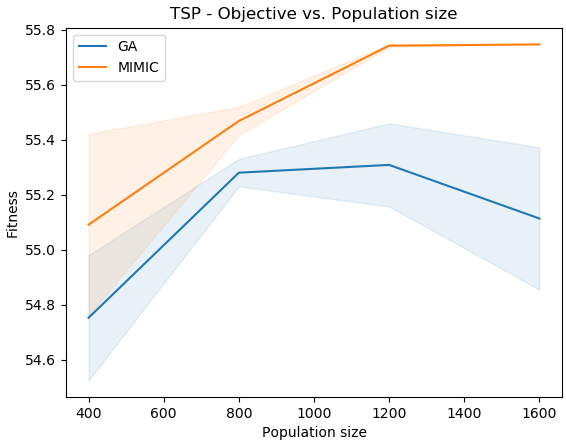
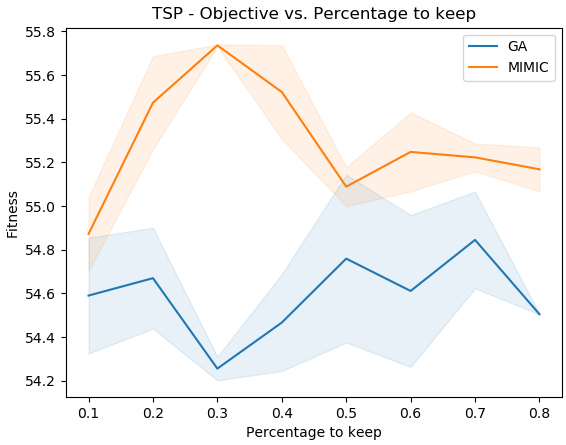


Figure 1: City coordinates inputs for the Traveling Salesman problem obtained and truncated from https://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html . The coordinates respective from top to bottom are: (6.734, 1.453), (2.233, .010), (5.530, 1.424), (.401,.841), (3.082, 1.644), (7.608, 4.458), (7.573, 3.716), (7.265, 1.268), (6.898, 1.885), (1.112, 2.049), (5.468, 2.606), (5.989, 2.873)

This problem takes the city coordinates, calculates the distances between all pairs of cities and finds the shortest possible route that visits each city exactly once and returns to the origin city.

Results/Analysis





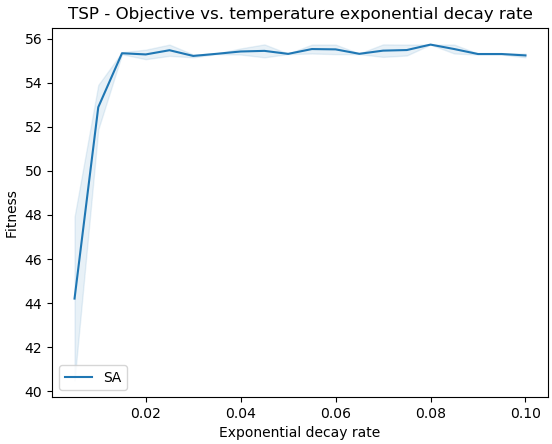


Figure 2-4 (topleft, topright, bottom): These plots analyzes the effects of different hyperparameters tuning available in the mlrose toolkit.
For GA and MIMIC, shown in Figures 2 and 3, the API allowed tuning to population size and keep percentage, for which I chose 200 to 1000 for population size and 0.1% to 0.8% for keep percentage.
The available parameter for SA (Figure 4) is exponential decay rate, for which we choose 0.005 to 0.105 with increments of 0.005.
These numbers were chosen based on having chosen a larger range and truncating it after convergence.

From figures 2 to 4, we analyze the optimization of parameters using GA, MIMIC, and SA. The objective here is described as the distance taken to visit all the cities exactly once. Therefore, the best algorithm is the one with the lowest cost since we want to take the least distance for the trip. There is a clear separation between the fitness of all three algorithms. This demonstrates that GA has the lowest fitness (54.5) compared with (55.2) and (55.3) for MIMIC and SA respectively. With this new information, we chose the best parameters for each algorithm to compare the objectives and run times with each other.

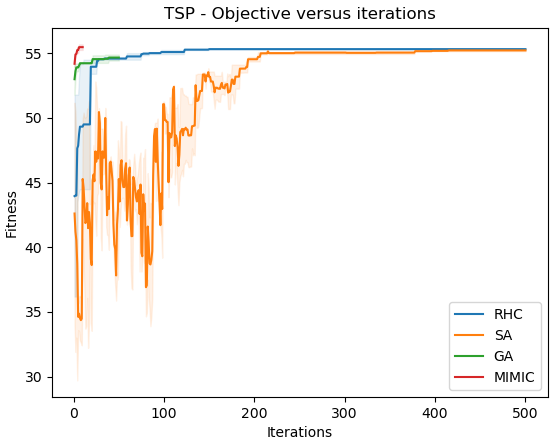


Figure 5: Final path distances versus iteration for the 4 search algorithms

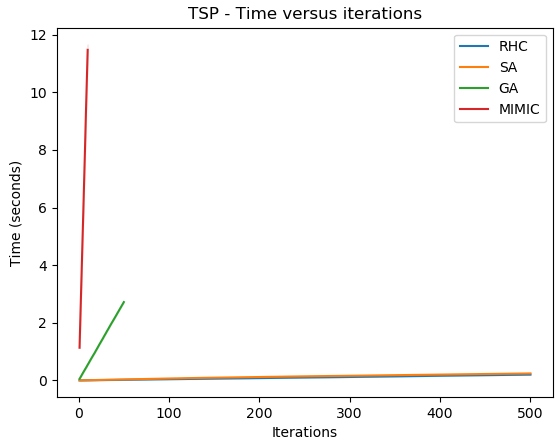


Figure 6: Times taken to run these algorithms on the optimized parameters.

The results in Figure 5 and 6 shows performance of the search algorithms on the traveling salesman problem; here we observe, like figures 2-4, that GA performs with the lowest cost. MIMIC clearly ends with a higher cost than GA. SA converges with RHC after 400 iterations and the GA curve ends lower than RHC before it converges. Figure 6 shows that GA is faster than MIMIC but slower than SA and RHC to find a solution. The times are linear with clearly higher slopes as we move between each algorithm.

The rationale of the performance lies in the data that is provided. For GA, the algorithm takes 2 parent paths (ex: [**1 6** 4 2 3 5] and [**3 5** 2 6 1 4]) and generate a child path (ex: [3 5 4 2 3 5] and [1 6 2 6 1 4]) with chances of mutation of randomly changing a “chromosome” with a random city. With such this will explore combinations of cities faster. In these plots we have the GA and MIMIC algorithm set to keep 20% of the given parent. Here GA works well because there are no specific routes and swapping percentages of cities around quickly and thoroughly explores many different paths as possible. The SA algorithm occasionally randomly, with decreasing randomness with increasing time, choses its next neighbor. This prevents the process from being stuck on a local minimum. The approach takes a city and tries to find the next city with the lowest distance, with the probability of accepting the city even if it is not the lowest distance. However, this is not as effective as GA to find the optimal solution because it is not as random. The first few accepted cities is already set, whereas in GA there is a chance that even the first couple cities will change. MIMIC takes a cost function and randomly guesses the first half of the solution, then generates more samples for the second half based off the distribution of the

first half. This performs worst than GA for this problem because, like SA, it is less random for the first half of the solution. The cost of the MIMIC solution would be heavily dependent on the goodness of the first half guess.

Looking at the time analysis per algorithm, SA/RHC would perform the fastest because it is essentially breadth first search which is known to have time complexity of less than O(V+E). Where V is the number of vertices and E is the number of edges. The acceptance randomness will make it faster because there's a chance it will not touch all the nodes in the tree. MIMIC has O(V log E) time complexity [2]. Per iteration, MIMIC has to find the top population percent to keep, update probability estimates and generate new sample for the next child. GA randomly selects the indices of the two parents and "reproduce" a child output. This shows the number of steps MIMIC has more than GA, explaining the significantly higher MIMIC time complexity. Since the same algorithms are used throughout the paper, the time complexity explanations will be the same.

Four Peak Problem

Problem Description

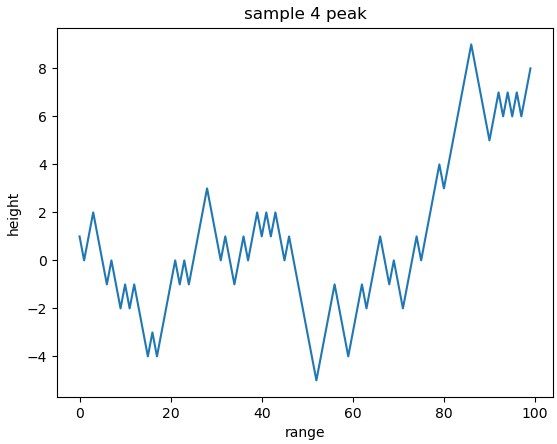
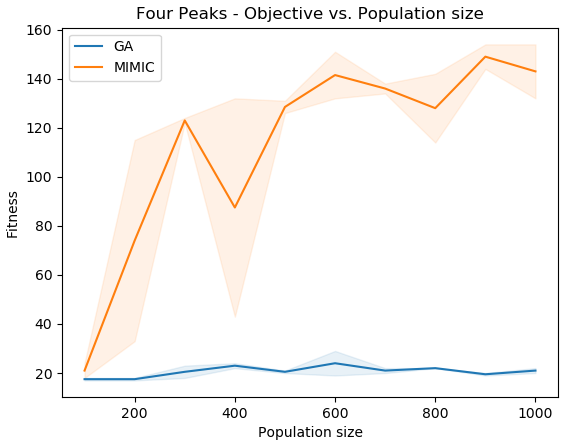
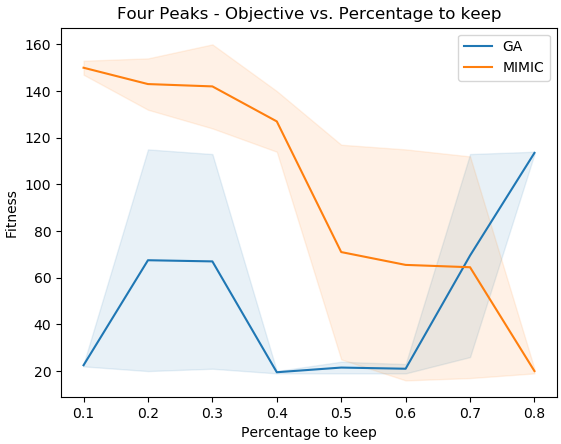


Figure 7: Sample input of the problem used to generate plots.

This problem can be thought of as a mountain range with heights as your data. Solving the problem means finding the highest point of the mountain range. Naïve methods are known to be stuck on top of a local maximum. (i.e.: being stuck on the peak at 27 range because everything to the sides of it is negative.)

Results/Analysis





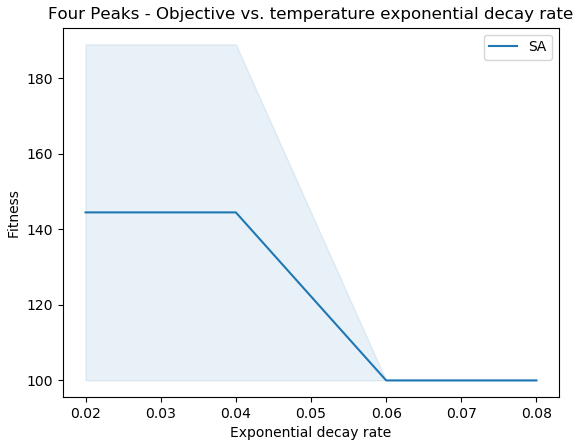


Figure 8-10(topleft, topright, bottom): These plots analyzes the effects of different hyperparameters tuning available in the mlrose toolkit.
For GA and MIMIC, shown in Figures 8-9, the API allowed tuning to population size and keep percentage, for which I chose 200 to 1000 for population size and 0.1% to 0.8% for keep percentage.
The available parameter for SA (Figure 10) is exponential decay rate, for which we choose 0.005 to 0.105 with increments of 0.005.
These numbers were chosen based on having chosen a larger range and truncating it after convergence.

Fitness here is calculated via max(tail(0, x), head(1, x)) + R(x, T), where tail(b,x) and head(b,x) is the number of trailing and leading, respectively, b's in the state x. R(x,T) is the state size if the candidate is higher than the tail and head; otherwise, it is not a peak so R(x,T) will be 0. Essentially, it is outputting the size of the peak and determining if it is a peak based off the tail and head bases of the peak.

From figures 8-10 we can see that SA performs roughly the same with MIMIC with 145 and significantly better than 110 for GA.

This result demonstrated that GA would get stuck in the local minimum compared to SA and GA which was able to find the similar peaks. This would not demonstrate the clear benefits of SA versus MIMIC. To do so we analyze the objective and time performance for each algorithm using its optimal parameters.

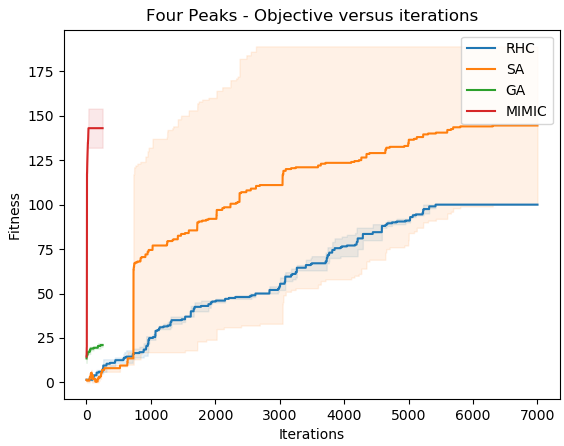


Figure 11: Final heights versus iteration for the 4 search algorithms

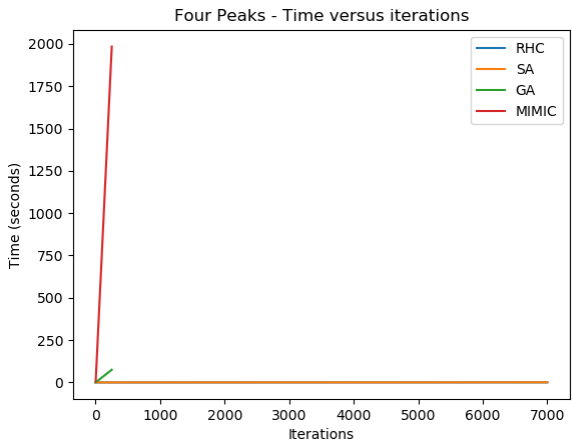


Figure 12: Times taken to run these algorithms on the optimized parameters.

The results once again show SA and MIMIC performance well above GA and RHC. However, the differentiators between SA and MIMIC lies in the time complexity. Both shows are relatively linear time consumption, however the differences in the slopes of each line is vast. In just 200 iterations, MIMIC takes 2k seconds compared to 12 seconds for the SA algorithm. Hence, this shows the superiority of SA over MIMIC for the Four Peak problem.

The performance per algorithm can be explained by looking at the data. Each data point is directly correlated with its neighbors. Therefore, no matter how you swap it around, with GA, it would not improve the performance. However, if we are running along the mountain range and we get stuck on a local maximum, the other three algorithms can randomly "teleport" us to another point on the range and explore a different region for another potential peak. SA is more random in developing the early parts of the solution but gets less random with increasing time. The idea of RHC is like SA, but it would performance worst with multiple peaks like this problem. It will randomly restart the problem but

can still be stuck on a local maximum if there are many local maximums. In addition, it has a higher chance of restarting even if it is close to the optimal solution; with SA it allows the algorithm to finish climbing to an optimal solution. MIMIC works wells here because the probability distribution is not complicated for this problem since the values are just heights with a very clear structure based off the peaks. I believe that if we choose a less well-defined structure problem, such as the continuous peak problem, MIMIC will perform worse than SA with a much clearer delta in performance. The time analysis was done in TSP section. The algorithm times used between TSP, 4peak and max k colors are the same.

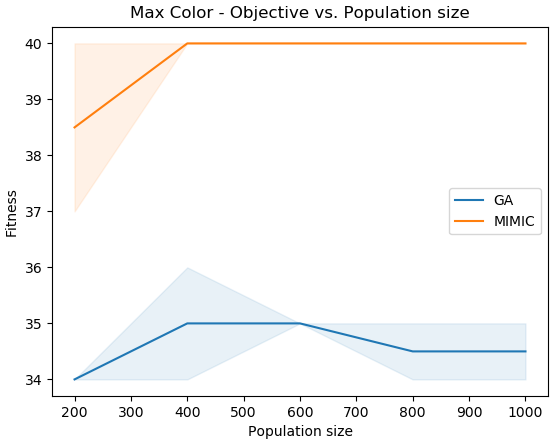
Max K-Colors

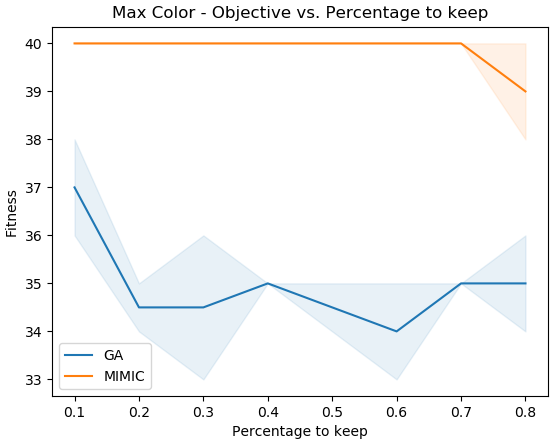
Problem Description

Figure 13: Sample input of the problem used to generate plots.

The input for this problem is a list of edges, representing the country borders, generated by the number labeled vertices. The problem is trying to color each of the neighboring numbered nodes a different color while minimizing the number of colors used.

Results/Analysis





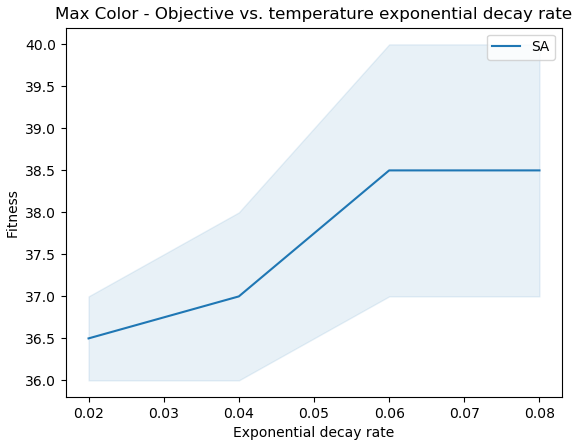


Figure 14-15: These plots analyzes the effects of different hyperparameters tuning available in the mlrose toolkit.
For GA and MIMIC, shown in Figures 2 and 3, the API allowed tuning to population size and keep percentage, for which I chose 200 to 1000 for population size and 0.1% to 0.8% for keep percentage.
The available parameter for SA (Figure 4) is exponential decay rate, for which we choose 0.02 to 0.08 with increments of 0.02. These numbers were chosen based on having chosen a larger range and truncating it after convergence.

In the Max-K color problem, fitness is calculated by how many nodes have adjacent nodes of the same color. It is a good thing if adjacent nodes have the same color because that means that in a set of three nodes, only the minimum 2 colors

are used. With that rationale, the higher the fitness the better the performance of the algorithm. From figures, MIMIC clearly performs the best out of the three algorithms with 40 compared to the 35 and 38.5 for GA and SA respectively. Taking the optimal parameters, we compare each algorithms' optimal performances.

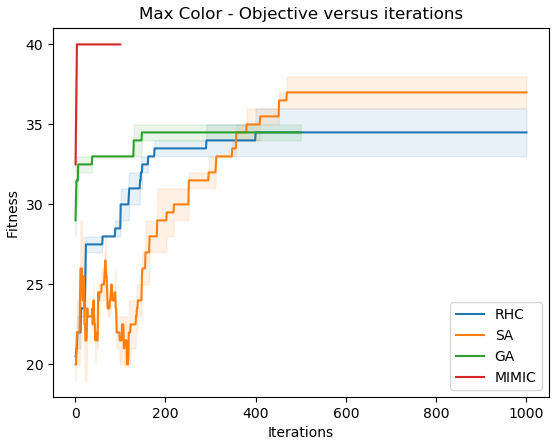


Figure 16: Final 'same-neighbor colors' count versus iteration for the 4 search algorithms

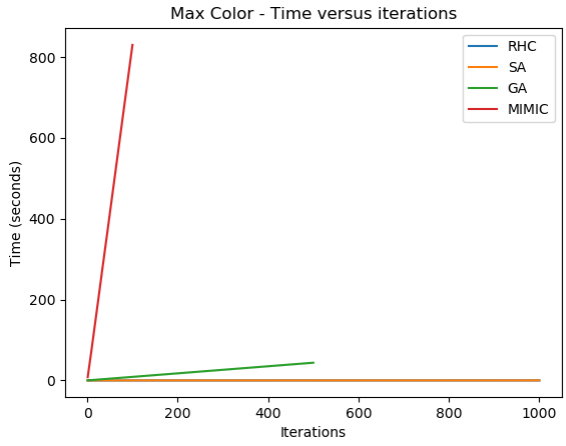


Figure 17: Times taken to run these algorithms on the optimized parameters.

The objectives in figure 16 is consistent to the results obtained in figures 14 -15; this details that MIMIC shows the clear best performance despite the long times it takes per iterations (explained in previous sections). The GA algorithm would take two candidate color assignment to each vertex and generate children out of it. This method would tend to be stuck on a single solution because it will be tougher to introduce a color or take a color away since color assignments are already set for scrambling. SA algorithm would randomly, with decreasing randomness with increasing time, accept a color as it goes through the process of choosing a valid color. An example based off figure 13, vertex 1 (color A), has 2 probable vertices 2 and 28. Right now it does not matter because the max color is the same after assigning 28 or 2 with a color. Assuming we chose 2 to assign color B, the next possible vertices are 3 and 30. The lowest addition, based on the algorithm, is to make vertex 3 color A; however, with SA it can assign the color or based on the probability randomly assign a random valid color (Colors A or new color C). Downstream towards the end of the assignment, the choices of colors to choose will be less and less, leading to a local maximum. This is important in the beginning when the probability of randomness is high because it does not cause the results to be stuck to a certain path. MIMIC is similar here only with complete randomness in the first half of the assignments. Based off what it has after the first assignment half, it will complete the second half based off what the rules and trends permit. As discussed earlier, towards the end of assignments, there are much less color assignment flexibility. Another way to think of why MIMIC works better is that, for the Max-K color problem there is very little knowledge of the optimal solution in the beginning of the assignments. Therefore, the early assignments matter much less than the middle or end of the assignments since we do not know better. Once we have information, it is optimal to use that information to figure out the rest of the assignment for optimal solution. SA works similarly, where it considers the assignments more as time increases. I believe a contributor to the problem is that the problem always starts with vertex 1 as the initial route. If the starting vertex is randomized, the solution may be more optimized than this current solution. However, it appears that a complete randomization of the initial sections of the solution performs the best for this problem. The time analysis was done in TSP section. The algorithm times used between TSP, 4peak and max k colors are the same.

Part 2: Neural Net with Randomized Optimized Weights

Diabetes Detection

Problem Description

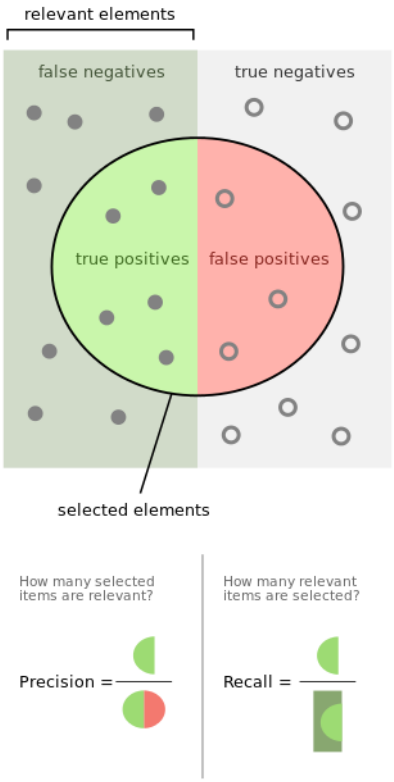
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Age	Gender	Polyuria	Polydipsia	sudden w	weakness	Polyphagi	Genital th	visual blu	Itching	Irritability	delayed h	partial par	muscle sti	Alopecia	Obesity	class
2	40	Male	No	Yes	No	Yes	No	No	No	Yes	No	Yes	No	Yes	Yes	Yes	Positive
3	58	Male	No	No	No	Yes	No	No	Yes	No	No	No	Yes	No	Yes	No	Positive
4	41	Male	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	No	Yes	Yes	No	Positive
5	45	Male	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	No	No	No	Positive
6	60	Male	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Positive
7	55	Male	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Positive

This early diabetes detection dataset, pulled down from the UC Irving machine learning repository, has each row represent an individual patient. Each row describes whether they have any of the 16 characteristics of diabetes and a flag of if they were diagnosed with diabetes. Some example characteristics includes age, gender obesity, etc. There are 521 patients in this dataset. The data is mostly binary besides the age field. Our goal is to use four different neural nets with a different optimizer for its weights to train on this dataset.

We will use common metrics to evaluate the machine learning algorithm techniques, such as data loss, time, and F1 Score. Here we will use the sklearn metrics toolbox's classification report to generate some of these analyses. F1 score is the harmonic mean of the precision values and recall value. Precision is the positive predictive value (true positives/(true positives + false positives)) and recall is the sensitivity value (true positives / (false negatives + true positives)). These two metrics measures 1) how many selected items are relevant and how many relevant items are selected.

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})}.$$

https://en.wikipedia.org/wiki/Precision_and_recall



Results/Analysis

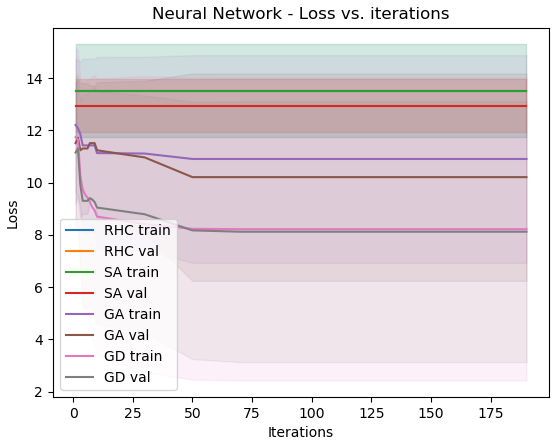


Figure 18: Final Neural Net loss versus iteration for the 4 search algorithms

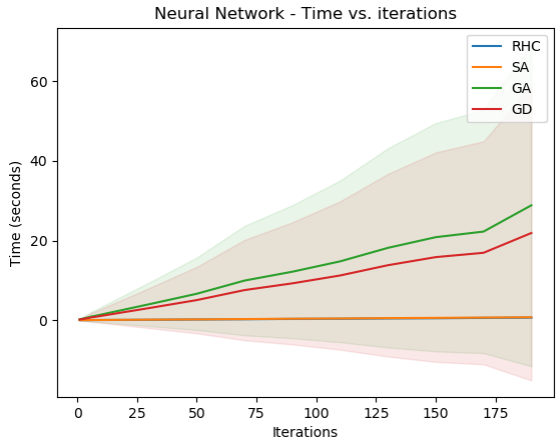


Figure 19: Times taken to run these algorithms on the optimized parameters.

RHC test classification report					SA test classification report				
	precision	recall	f1-score	support		precision	recall	f1-score	support
0	0.7	0.78	0.74	64	0	0.7	0.78	0.74	64
1	0.58	0.47	0.52	40	1	0.58	0.47	0.52	40
accuracy			0.66	104	accuracy			0.66	104
macro avg	0.64	0.63	0.63	104	macro avg	0.64	0.63	0.63	104
weighted	0.65	0.66	0.66	104	weighted	0.65	0.66	0.66	104
GA test classification report					GD test classification report				
	precision	recall	f1-score	support		precision	recall	f1-score	support
0	0.92	0.88	0.9	64	0	0.97	0.98	0.98	64
1	0.81	0.88	0.84	40	1	0.97	0.95	0.96	40
accuracy			0.88	104	accuracy			0.97	104
macro avg	0.87	0.88	0.87	104	macro avg	0.97	0.97	0.97	104
weighted	0.88	0.88	0.88	104	weighted	0.97	0.97	0.97	104

Figure 18 shows the losses of all four algorithms. MIMIC is excluded because it only handles discrete problems which neural net weights are not. Gradient Descent (GD) loss less over increasing iterations, soon the training losses converges with the testing losses. This indicates good training with little overfitting. There is also less loss with GA than SA and RHC after convergence. This would indicate that I would be less confident in the results being accurate during the testing phase since the model is changing as much as it is in the beginning iterations. The time analysis was done in TSP section. The algorithm times used between TSP, 4peak and max k colors are the same as used in the neural net training phase. In addition, to the three algorithms we included gradient descent. Gradient descent moves in the direction of the steepest descent, this timing should be like SA/RHC because it only calculates the gradient and takes the max (min based in which is being optimized).

The classification report shows that GD, GA, SA and RHC has the highest to lowest f1-score performance in that order. Overall, all the algorithms are more reliable at predicting that a patient is positive for diabetes than if they are negative for diabetes. Reliable means, if the algorithm says you are predicted to be positive for diabetes then 1) the chance of the positive being a true positive is higher and 2) the chances of it not being a negative is lower. Therefore, with the dataset provided for training if the patient would be a bit more vary of the results if the output is negative versus if it is positive. The exception is if the algorithm has a gradient descent optimization for its weights then you can be 97% certain that the results you get are reliable regardless if the output is negative or positive. Rationale for why the standard GD performs best may be that the results maybe that we need more iterations for the optimization algorithms to converge higher. Another reason (mentioned in the homework description) may be that weights are non-discrete parameters and gradient descent performs better than SA/RHC/GA/MIMIC on non-discrete parameters like weights. Therefore, strategies such as randomly swapping weights between neurons or initially randomly assigning weights would not have the same effects as we would on discrete parameters.

Conclusion

Algorithm	Pros	Cons
RHC	Quick Easy to implement Easy to converge depending on how many restarts are allowed	Performs poorly for complicated problems Can lose the optimal solution more often due to consistent randomness towards the end.
SA	Quick Consistent in location global optima problems Good at avoiding local maximums.	Not as efficient for problems without structure
GA	Can be quick Performs well with problems with more flexibilities in options between features. Can perform well-enough on non-discrete problems such as weights of neural net	Performs poorly on problems with high defined correlation between neighboring features Slow for complicated problems
MIMIC	Performs well for all problems with definition between features Good against problems starting off with very little knowledge of optimal solution.	Not as efficient for problems without structure Significantly slow for every problem introduced Does not work for non-discrete problems

References:

[1] https://github.com/ezerilli/Machine_Learning/tree/master/Randomized_Optimization

[2] https://www.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf

[3] https://stackoverflow.com/questions/9146086/time-complexity-of-genetic-algorithm#:~:text=Genetic%20Algorithms%20are%20not%20chaotic%2C%20they%20are%20stochastic.&text=Given%20the%20usual%20choices%20(point,the%20size%20of%20the%20individuals.