Randomized Optimization

Yi Zhao

Yzhao644@gatech.edu

Introduction

Optimization is a technique to find the best solution with certain conditions or limitations. It is researched in a lot of area from economy, mathematics to computer science. In computer science area, there many algorithms developed to solve certain problems. For example, dynamic programing algorithm to solve Knapsack problems. In this report, I focused on research on four algorithms. randomized hill climbing, simulated annealing, genetic algorithm and mutual-information-maximizing-input-clustering (MIMIC). I used the above algorithms to solve four optimization problems. They are 4 peak, N-Queens, Knapsack problems and a neural network to solve adult dataset.

The first three problems are designed to show the advantage of individual algorithm and to explain and compare the hyper-parameters in details for the later research for neutral network. Which is again trying to find the best parameters for optimization algorithms and compare with gradient boosting problem.

I started with description on how the algorithms work and how some hyper-parameters are going to affect the algorithm. Followed is my initial thought on the performance of the individual algorithm on each problem. I then showed my process on deciding the hyper-parameters and comparison between different algorithms on fitness, computational time, and complexity. The second part of the assignment is to implement three algorithms on neutral network to solve the adult data.

The library I use is mlrose-hiive.

Keyword: optimization, MIMIC, Knapsack Problems

1. Algorithms

1.1 Random Hill Climbing

Random Hill Climbing comes from Hill Climbing technique, which is method that excels in find the local optima. It starts with a solution and trying to move around to find a better neighbor to improve the solution. The search stops when either maximal iteration or attempts reaches. The highest value of this search is taken as global optimal.

The random come when the algorithm is restarted with another random position to find a new local optimum. The result is the largest of all local optimums. The hyper-parameter to train is the number of restarts of the algorithm.

1.2 Simulated Annealing

Simulated Annealing is algorithm that use ‘explore and exploit’ mechanism to find the global optimum.

The analogy to annealing process is the algorithm starts with a higher threshold (initial temperature) to explore. The threshold is decrease as the temperature goes down, all the way to minimum temperature, according to a decay method (Exponential or Geometrical).

For problems where finding an approximate global optimum is more important than finding a precise local optimum in a fixed amount of time, simulated annealing may be preferable to exact algorithms such as gradient descent, Branch and Bound.

There are two hyper-parameters to tune for SA, init_temp and decay. Init_temp decides how widely do you want to explore, and decay determine how fast you want to decay to only search for local optimum.

1.3 Genetic Algorithm

Genetic Algorithm technique uses evolutionary algorithms to generate new kids based on previous parents' generation. The advantage is information are transferred from parents to next generation, through techniques like mutation, cross over and selection. With those operations within, it can have a better understanding of input data and some memory on the previous generation.

There are multiple hyper-parameters to tune for GA. This report focuses on two of them. Population size, which decides how many parents are selected to generate newer generation. And mutation rate, which decides how to get new solutions which also has some feature from previous generations.

1.4 Mutual-Information-Maximizing Input Clustering (MIMIC)

This is another technique in Randomized Optimization field. The idea is to generate the description of certain parameters and only take the better half (or some other portion, Keep percentile) for next optimization. It gets input parameter's distribution from second order statistics of the input data, which is computationally simple. It also has a memory mechanism to transfer learnings from previous iterations to next ones.

The hyper-parameters I focused on are Population size and keep percentile. They are used to decide how much information are gathered to estimate parameters' distribution and how much information are kept for future iterations.

2. part 1: problems and process of training

In this part, I will introduce the problems I choose to show difference between algorithms and hyper-parameters. How the hyper-parameters are chosen with individual algorithm. An overall comparison of individual algorithm with best hyper-parameter combination in terms calculation time, fitness, and complexity of the problems.

The grid search method I used is the runners within mlrose_hiive.

I also modified the source code to keep track of running times per iteration. And used customized fitness function to keep track of fitness per evaluation. All the plots are generated with different random seeds.

2.1 N-Queens Problem

N-Queens Problem is to put N numbers of queens on the chess board so that they cannot attack each other. It's is a discrete optimization problems. It has multiple optimums, but the maximum fitness function is unique. For a 8*8 chess board, it should be 2C8/2, which is 28.

The uniqueness of the problems is that all the optimums are reached at the same level (28) and the basins between the optimum are not very wide. In another word, it's easy to change from one solution to another better and very hard for algorithms to stuck on local optimum.

2.1.1 Solve N-queens with Random Hill Climbing

As mentioned earlier, one hyper-parameter to tune here, namely the restarts time. One thing to notice is that even with no restarts, the algorithm could succeed in finding the optimum. This is because the N queens problem's local optima are located close to each other and many of them are global optimum.

Restarts	Sum of Time	Average of Fitness
0	1.1968895	25.66666667
10	20.4609738	24.42424242
25	250.0702527	24.51282051

Decay rate	Average of Fitness	Max of Fitness	Sum of Time
0.005	**23.28666667**	**28**	**98.9929119**
1	25.06666667	28	2.3322061
10	23.73333333	28	3.2816466
50	23.26666667	28	4.4976455
2500	23.06666667	28	15.3010815
5000	23.2	28	17.6331149
10000	22.53333333	28	17.4174676
0.05	**25.6**	**28**	**66.5409918**
1	25.6	28	66.5409918

Since it's required to use the random hill climbing algorithm, I am using restart = 25, since it's giving the higher average of fitness. And the reason for that is since there are multiple optimum, having it restart multiple time making the algorithm converging faster and making the average fitness larger than 10 restarts.

2.1.2 Solve N-queens with Simulated Annealing

Two hyper-parameters to train here are tuned for comparison.

Here we can see SA also converge to global optimum no matter what hyper-parameters to choose from. This is due to multiple global optimum of the problem itself.

Decay rate	Average of Fitness	Max of Fitness	Sum of Time
0.005	**23.28666667**	**28**	**98.9929119**
1	25.06666667	28	2.3322061
10	23.73333333	28	3.2816466
50	23.26666667	28	4.4976455
2500	23.06666667	28	15.3010815
5000	23.2	28	17.6331149
10000	22.53333333	28	17.4174676
0.05	**25.6**	**28**	**66.5409918**
1	25.6	28	66.5409918

The decay rate decides how fast the temperature goes to the minimum temperature. Since the problem is easy to solve, no need to use a smaller parameter to avoid local optimum.

The temperature is used to decide how wide the algo goes to find the global optimum. Since there are many optimum and gap is small, the lowest temperature is good to solve the problem.

I chose decay 0.05 and temperature 1 for faster compute time.

2.1.3 Solve N-queens with Genetic Algorithm

There are other hyper-parameters other than pop size and mutation rate, like pop_breed_percent, elite_dreg_ratio or cross over method. I think the first two are tied back to pop size, as it's only a ratio on the pop for reproduction. The cross over one might be a good point for future analysis. Currently I am using uniform cross over.

In below you can find out the performance of certain hyper-parameter combos. Again, all of the combo could reach to global optimum because of the uniqueness of the problem. Here I chose 300 and 0.6.

Population Size	Mutation Rate	Time(max)	Fitness	Time(avg)	Fitness
150	0.4	52.36109	349	17.03123	28
	0.5	50.62864	350	16.14924	28
	0.6	50.00811	351	16.29367	28
200	0.4	68.76631	353	22.25842	28
	0.5	83.28351	351	26.7368	28
	0.6	98.0206	350	30.87064	28
300	0.4	188.5487	350	60.00127	28
	0.5	152.7405	352	48.22158	28
	0.6	148.1845	355	44.99537	28

Row Labels	Sum of Time	Max of Fitness	Sum of Fitness
100	**10.389696**	**27**	**143**
0.25	3.6170854	27	**48**
0.5	3.3788663	26	**47**
0.75	3.3937443	27	**48**
150	**12.5869007**	**27**	**143**
0.25	5.4314956	27	**48**
0.5	3.7969106	27	**48**
0.75	3.3584945	26	**47**
200	**11.6891983**	**27**	**143**
0.25	4.4153314	27	**48**
0.5	3.4455904	26	**47**
0.75	3.8282765	27	**48**
300	**13.1251468**	**28**	**146**
0.25	4.3442034	28	**49**
0.5	4.0933207	27	**48**
0.75	4.6876227	28	**49**

2.1.4 Solve N-queens with MIMIC

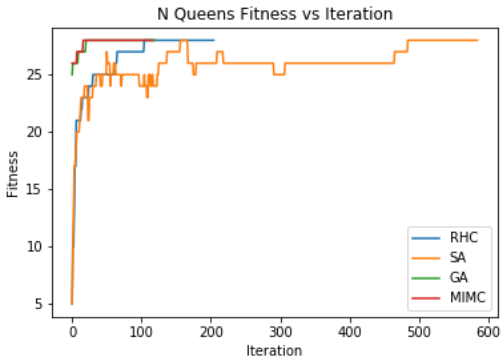
Similar to GA algorithm, there are two parameters to tune, pop size and keep percentile. See up right for details

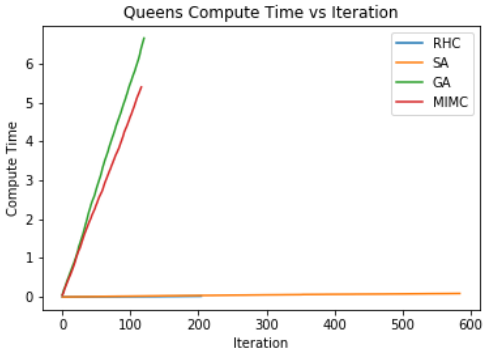
A surprise find is that the smaller number doesn't cut it when trying to reach the global maximum, though they are pretty close to global optimum. A possible reason is that the algorithm really needs a lot of knowledge (population) to get sense of the parameters distribution. And if the percentile is too low, it might cut the optimum and compromise with the second to the best.

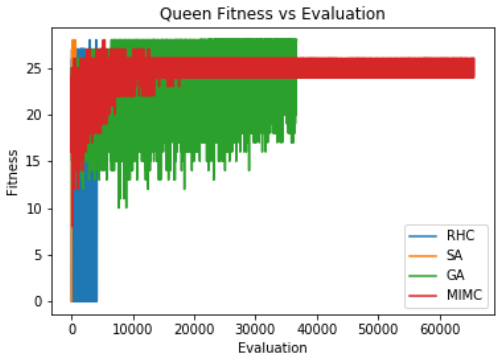
The parameter I chose is 300 and 0.75 for the fitness.

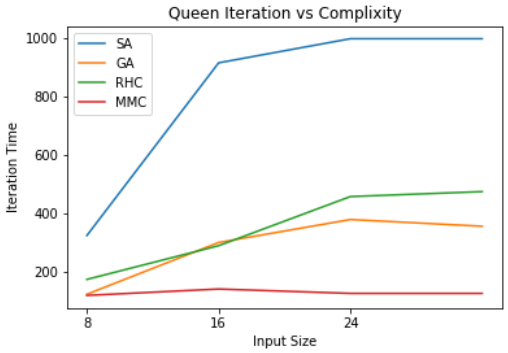
2.1.5 Overall comparison between algorithms

With the best hyper-parameter combo, a final train was performed with four algos. I made comparisons between iteration vs fitness and computation time, evaluation vs fitness and model complexity vs fitness. The comparison is shown in below plots.









First two plots are calculated fitness score and total compute time against iteration time. Source code is modified to keep track of time. It's calculated when each iteration finishes and saved in a list.

Third plot is evaluations during the full training time. I used a list to save the fitness score every time when the fitness function is called.

The complexity is done by increasing the chess boards dimensions and number of queens.

A few findings from the compare.

- All the algorithms finished reaching the global optimum. GA and MIMC need much less iteration than SA and RHC.
- SA and RHC are significantly faster than MIC and GA.
- MIMC are GA needs a lot of evaluation to reach the optimum. Since they both need to get an understanding of the data distribution. There are multiple evaluations happening per iteration.
- When problem comes more complex, MIMC doesn't change iteration time that much, it convergence faster in terms of iterations to run (not necessary compute time), while other three increase and then plateau.

In summary, I'll choose SA algo, due to the faster compute time.

2.2 Four Peak Problem

Four peak problem maximize the reward and continues 1s in the series of 0 and 1 digits. The reward is awarded with a default number T.

There are two global optimal for this problem, and they are far on the edge and local optimal have wide basin for algorithms to be stuck.

In theory RHC and SA should suffer from this problem.

2.2.1 Four Peak with RHC

Below is stats on restart. It's not surprise to see that RHC needs a lot of restart to be better and find the global optimum.

Restarts	Average of Fitness	Sum of Time	Sum of Fitness
0	10.66666667	0.667529	32
5	12.55555556	22.9595037	226
5	15.625	535.3760437	750
25	16.85897436	1629.805812	1159

RHC Hyper Tuning

Decay	Temp	Max of Fitness	Sum of Time
0.0005	1	38	0.186981
0.0005	10	36	0.257713
0.0005	50	25	0.336443
0.0005	1000	11	0.396064
0.0005	10000	13	0.449849
0.05	1	31	1.708365

SA Hyper Tuning

2.2.2 Four Peak with SA

For Simulated Annealing, the best combo is with small decay rate so the algo would have a longer iteration of higher temperature to find the local maxima. For the temp parameter, it's having a better performance with lower temp to start with it. The reason is you don't want to the algo to start searching really wildly in the beginning to be stuck in a unfavorable local optimum.

Decay	Temp	Max of Fitness	Sum of Time
0.0005	1	38	0.186981
0.0005	10	36	0.257713
0.0005	50	25	0.336443
0.0005	1000	11	0.396064
0.0005	10000	13	0.449849
0.05	1	31	1.708365

2.2.3 Four Peak with GA

For GA, there are 3 compo with the largest fitness score 107. I chose the one with fastest running time.

The reason behind is that the Four Peak problem is designed for GA. The mechanisms for the algo to be trained is exactly how you would want your digit to change into. There are only two choices for every digit (0,1). So it doesn't need a lot of population to get sense of the distribution.

For mutation rate. 0.6 is not too close to 0 or 1. Meaning that the algo won't have a huge variance and it changes/mutates in a good pace.

Population Size	Mutation Rate	Time	Fitness
150	0.4	31.355353	60.0
	0.5	34.990566	60.0
	0.6	31.263342	60.0
200	0.4	31.587773	60.0
	0.5	34.917729	60.0
	0.6	30.930705	107.0
300	0.4	49.534204	107.0
	0.5	43.754218	107.0
	0.6	42.539056	60.0

GA Hyper Tune

Keep Percent	Sum of Time	avg of Fitness
100	**854.7021876**	**45**
0.25	263.368163	15
0.5	269.8844086	15
0.75	321.449616	15

MIMC Hyper-tune

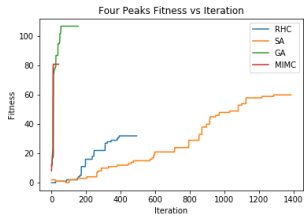
2.2.4 Four Peak with MIMC

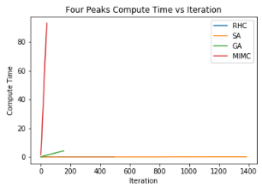
For MIMC problems, the Keep percent was tuned to find the best parameter combo. There's not difference. See top right table.

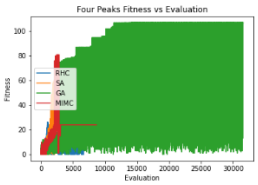
The reason behind it is that once the pop size is decided for the problem, the distribution is not hard to interpret. Whether keep 25% or 75%, the optimum will always be preserved since the algo has a pretty good understanding of distribution.

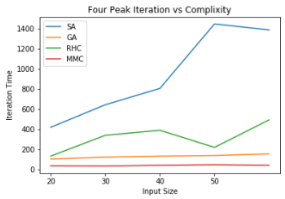
2.2.5 Overall comparison between algorithms

Same comparison was done as RHC for evaluation, iteration, computation time and complexity.









The complexity is achieved by increasing the length of the digits of 0s and 1s.

Findings from the comparisons.

GA and MIMIC out performed SA and RHC by a lot. RHC is performing the worst. This is inline with my perception earlier, since the optimum are on the edge and local optimum will stuck RHC the most and also SA.

MIMC again has overwhelming large computation time.

GA used a lot our evaluation to figure out the distribution and that might be the reason why it outperforms other algos in fitness.

Iteration time to converge doesn't grow much for MIMC and GA when problem becomes more complexed. They all converge faster than SA and RHC in terms of convergence.

2.3 Knapsack problem

Knapsack is an NP hard problem. The goal is to figure out how many items to take but not exceed the maximum weight limit but at same time maximize the total value.

The weight and value I used for this problem is randomly assigned but both under 50. I used 140 items for default analysis and one item could be picked up to 10 times. The weight limit is 0.6 of the total 140 items.

There are not polynomial solution for this problem. So the best way would be generalize the underlying distribution and find the optimal solution.

2.3.1 RHC for Knapsack

RHC needs more random start to get a better fitness of the problem.

Row Labels	Sum of Time	Max of Fitness
0	553.675208	21115
5	4889.756713	21506
15	97525.05169	21511
25	430929.5316	21880
Grand Total	**533898.0152**	**21880**

Decay	Temp	Sum of Time	Max of Fitness
0.005	1	0.2615061	22368
0.005	10	0.1350907	22789
0.005	50	0.1660954	23180
0.005	5000	0.2036325	22250
0.005	10000	0.2645367	21922
0.05	1	1.8282453	16173

2.3.2 SA for Knapsack

SA has the similar situation to Four peak problems. Here the best temperature is higher than earlier because the problem space is really huge. 140*10 vs 10*2 in terms of size of combinations. So it requires a higher temperature for wider initial searching. Decay rate is still lower to best explore the space.

2.3.3 GA for Knapsack

I chose highest pop size and highest mutation size. It needs all the help it can get to best generalize the input data parameter and distribution. Although the maximum fitness doesn't change a lot between all the categories, the maximum combo ends up with largest total fitness.

Row Labels	Sum of Fitness	Sum of Time	Max of Fit
150	**213714**	**67.4938165**	**28206**
0.4	71392	22.5131749	28193
0.5	71039	21.2336123	28196
0.6	71283	23.7470293	28206
200	**214207**	**141.0967213**	**28252**
0.4	71620	31.8827279	28230
0.5	71421	46.3753909	28252
0.6	71166	62.8386025	28226
300	**215005**	**224.9944649**	**28250**
0.4	71835	77.4303075	28222
0.5	71561	68.2278516	28223
0.6	71609	79.3363058	28250
Grand Total	**642926**	**433.5850027**	**28252**

GA Hyper Tune

Row Labels	Max of Fitness	Sum of Time	Sum of Fitness
0.25	**25623**	**952.5829192**	**79686**
150	24139	400.1002366	39101
300	25623	552.4826826	40585
0.75	**25391**	**2783.797245**	**80004**
150	24689	1509.391623	39651
300	25391	1274.405622	40353
Grand Total	**25623**	**3736.380164**	**159690**

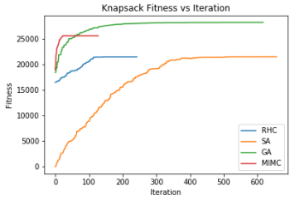
MIMC Hyper Tune

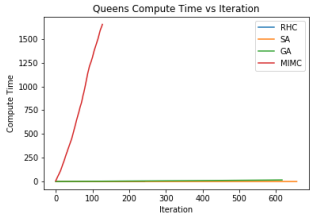
2.3.4 MIMIC for Knapsack

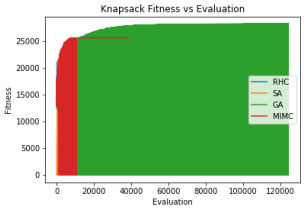
MIMIC's parameter combo also tells an interesting story. See right above table. It also wants to have largest pop size to get the info about the underlying number. The keep percentile is however choosing the lower ones. My guess is that it tries to eliminate as much as possible the lower solutions to better use it's iterations to come up with the global optimum.

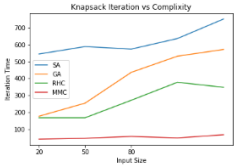
2.3.5 Overall comparison.

Same comparison was done as RHC for evaluation, iteration, computation time and complexity.









The complexity is achieved by change the number of items to choose from between (20,50,80,160)

One thing to point out is that I chose a really bad starting point for MMC and the running time just horrible for MMC hyper parameter tuning. With more parameter's combo, it might improve.

Findings from the plots.

- MIMC and GA outperformed SA and RHC by a lot. GA and MIMC are almost at the same level. This could be interpreted as GA also has ability to understand the distribution, with the reproduction capability and cross over capability.
- GA and MIMC did a lot of evaluation again. MIMC takes a lot of time computing but iteration time doesn't increase as problem becomes more difficult.

3. part 2 neural network parameters for optimization

In part 2 I am using three algos discussed earlier to train a neural network that best predicts adult dataset and compare the result with some bench mark, like gradient descend to measure the performance.

I took two steps for the training part. First is to get the best hyper-params combo. I created a function called nn_sa_test that tries a combination of hyper combo and used it to predict the testing datasets.

Second is to tune the neural net itself. Similar idea is performed as hyper-param train step. I focused on Learning rate and neural network structure.

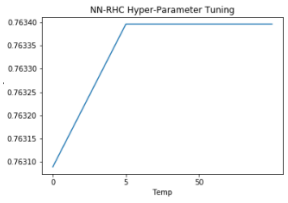
For the benchmark, I tried to use my sklean example but returns terrible, with only 50% plus accuracy. So here I used mlrose_hiive's own gradient descent for comparison.

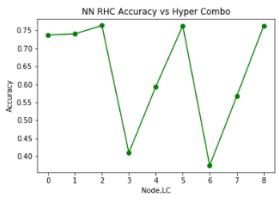
3.1 RHC for Neural Net

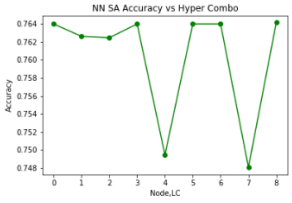
For RHC, I tired 4 restarts amounts. The number 5 and above give me similar metrics in terms of accuracy.

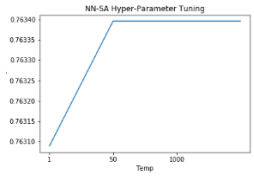
For second part of tunning, I tried multiple combinations of the maximal iteration time and attemps.

The nodes structure and learning curve combination are also calculated to come with the best accuracy.









3.2 SA for Neural Net

For SA part, there are two parameters combo to tune with. I choose the temperature to focus one, as in the previous session, decay rate doesn't affect much. For a complex neural network parameter, you would want you cooling time to be as long as possible for better result. The best combo of the temperature and decay curve is 50 and 0.005.

For the neural network structure. Both [4,2] and [10,2] gives me similar result. I kept [10,2] and a larger learning rate to avoid overfitting.

3.3 GA for Neural Network

With the ability to generate the input data distribution, I'd expect GA to perform better on the neutral network. It turns out to be good.

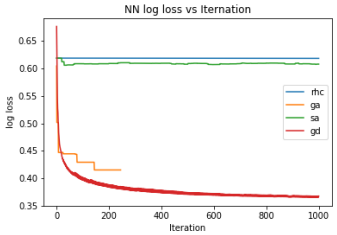
I tried different combo of Hyper-parameters and Neural network structure for better result.

One thing special about GA is it requires a higher max iteration and lower learning rate to perform well, which is understandable giving the nature for GA to understand the underlying distribution.

3.4 Final result of the neural network

Below plot is ploted by the fitness curve of the 4 method.

Gradient descent has the lowest error, followed by GA. Both RHC and SA doesn't generate well and returns a lot log loss in terms of fitness function.



	Accuracy on Testing	Traning time
RHC	0.736527	52.98288
SA	0.75572	37.38421
GA	0.762322	54.72645
GD	0.82401	33.55626

Below table also shows that the accuracy and training time performance.

Gradient Boosting has the best accuracy than other three and the calculation time is also faster than the 3. RHC and GA suffers calculation time because they need to restart and generate the input distribution.

Conclusion

Based on the result from the neural network, Gradient Descent performs better than other three. My theory on this is that the optimization problem is more like a generative learning. It tries to figure out what's the best parameter given the fitness function. Whereas Gradient descent is trying to discriminate by minimizing the loss function. It has a more direct feed back from the loss function where Generative modeling could suffer from not having enough data or stuck at the local optimal points.