Assignment 2: Random optimization

Name: huan wang

Username: hwang680

1.Comparison of Four Randomized Optimization Methods

1.1 Optimization problems

One max

Fitness function for One Max optimization problem. Evaluates the fitness of an n-dimensional state vector $x = [x_0, x_1, \ldots, x_{n-1}]$ as:

$$Fitness(x) = \sum_{i=0}^{n-1} x_i$$

This fitness function clearly has only one local and global optima. This should be the most trivial optimization problem one algorithm could face. The simplicity and time efficiency strength of Random hill climbing algorithm would be highlighted in this context.

Four peaks

Fitness function for Four Peaks optimization problem. Evaluates the fitness of an n-dimensional state vector x, given parameter T, as:

$$Fitness(x, T) = \max(tail(0, x), head(1, x)) + R(x, T)$$

where:

tail(b,x) is the number of trailing b's in x

head(a,x) is the number of leading b's in x

R(x,T)=n if tail(0,x) > T and head(0,x)>T; and R(x,T)=0 otherwise

There are two global optima for this defined function when T+1 leading 1's followed by 0's or when T+1 0's preceded by all 1's. There are two local optima with string of all 0's or 1's. Genetic Algorithms are more likely to find these global optima than other methods.

Knapsack

The knapsack problem is a problem which is more than a century year old in combinatorial optimization field.

Fitness function for Knapsack optimization problem. Given a set of n items, where item i has known weight w_i and known value v_i; and maximum knapsack capacity, W, the Knapsack fitness function evaluates the fitness of a state vector $x = [x_0, x_1, \ldots, x_{n-1}]$ as:

$$Fitness(x) = \sum_{i=0}^{n-1} v_i x_i, \text{ if} \sum_{i=0}^{n-1} w_i x_i \leq W, \text{ and } 0, \text{ otherwise,}$$

where x_i denotes the number of copies of item i included in the knapsack.

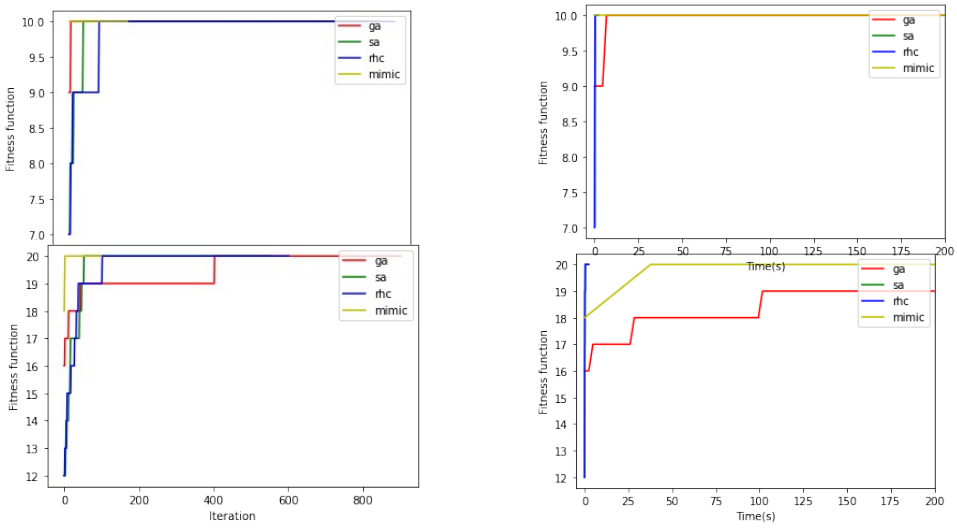
Knapsack problem is a classic NP-Hard optimization problem with no polynomial time solution. The strength of MIMIC was highlighted in this context, as it exploited the underlying structure of the problem space that was learned from previous iterations.

1.3 Results

Varying problem size reveal the strength of algorithms

For each of the three optimization problems, problem length was varied from 10 to 40 by increment of 10. To visualize how each algorithm progress, fitness function vs iteration is plotted for each method (shown in left column graphs in Figure 1-3) for each problem size(10-30, 40 not shown in graph). To also compare the time efficiency of each algorithm, fitness function is also plotted against actual optimization time (shown in right column graphs in Figure 1-3) for each problem size (10-30, 40 not shown in graph).

For **one max problem,** at problem size 10, not much differences are observed for four algorithms. As problem size increases to 20, MIMIC reached to global optima in the least amount of iterations followed by RHC and SA algorithms. If time efficiency is considered, shown in right column graphs in Figure 1-3, RHC and SA are the two most efficient algorithm that reaches global optima in the least time. Therefore, using **a problem size ~ 30** for one max optimization problem, simplicity and time efficiency strength of RHC and SA is revealed.



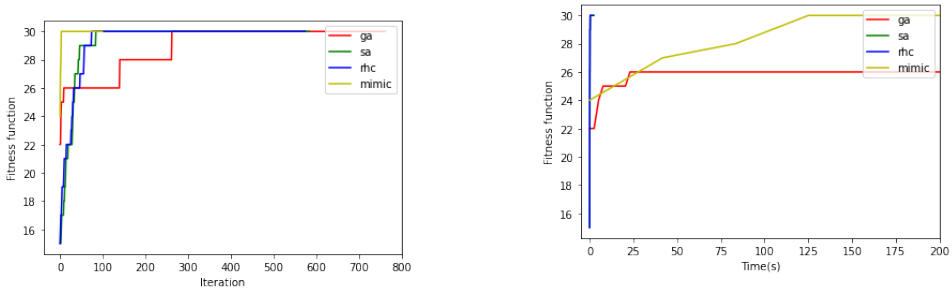


Figure 1. Onemax fitness function vs iteration and time(s) for four optimization algorithms(RHC, GA, SA, MIMIC) Row 1 (problem length 10), Row 2 (problem length 20), Row 3(problem length 30)

For four peaks problem**,** as the problem size increase from 10 to 30, MIMIC and GA are both able to find the global optima while RHC and SA are trapped in local optima. MIMIC and GA both reach global optima in less than 10 iterations, however, GA is more time efficient. Therefore, using **a problem size ~ 30** for four peak optimization problem, GA's capability to exploit complex hypothesis surface and relative time efficiency is revealed.

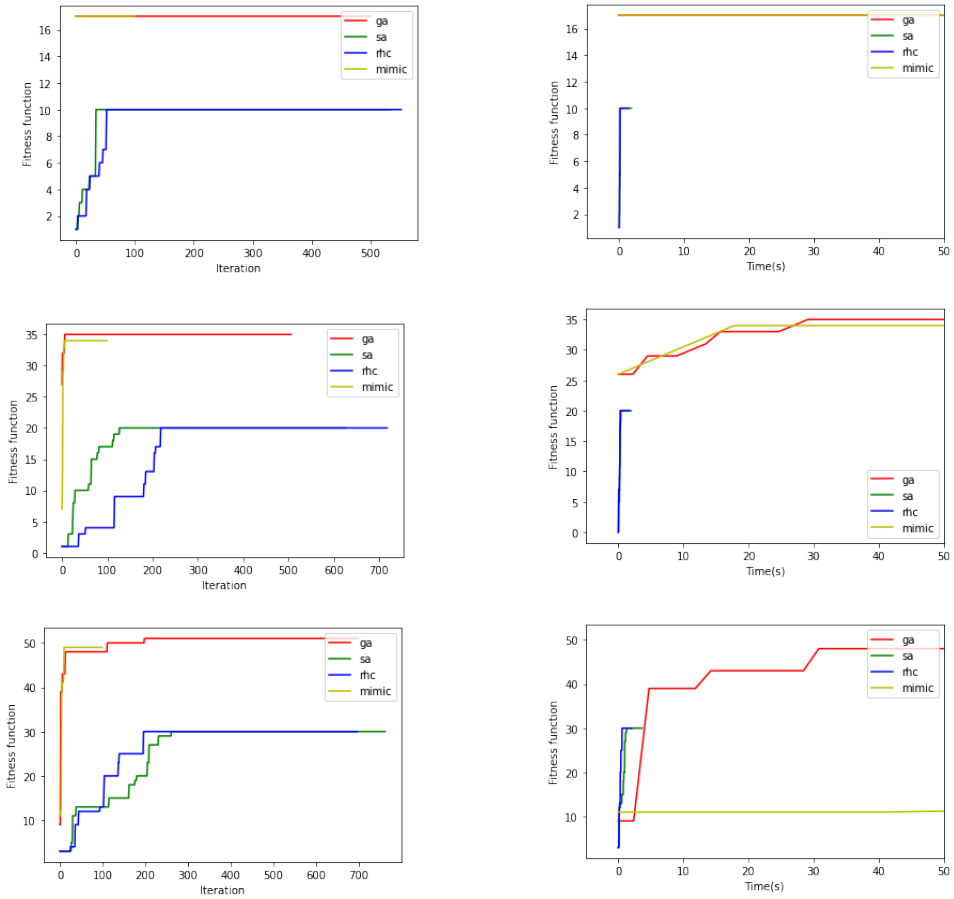


Figure 2. Four peaks optimization fitness function vs iteration and time(s) for four optimization algorithms(RHC, GA, SA, MIMIC), Row 1 (problem length 10), Row 2 (problem length 20), Row 3(problem length 30)

For **knapsack problem,** as the problem size increase from 10 to 30, MIMIC is the only method that is able to find global optima, while SA, RHC and GA are trapped in local optima. Therefore, using **a problem size ~ 20** for knapsack optimization problem, MIMIC's capability to exploit the underlying structure of the problem space that was learned from previous iterations is revealed.

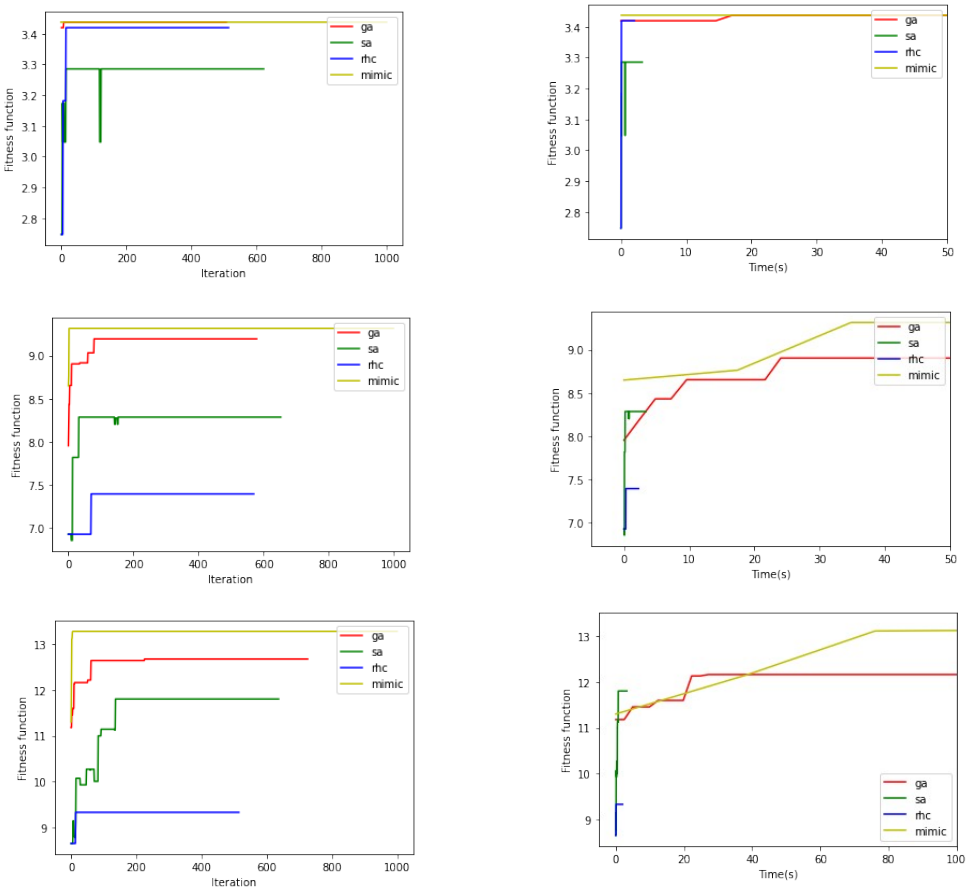


Figure 3. Knapsack optimization fitness function vs iteration and time(s) for four optimization algorithms(RHC, GA, SA, MIMIC), Row 1 (problem length 10), Row 2 (problem length 20), Row 3(problem length 30)

RHC and SA are the best performers for one max problem

As shown in Figure 1, four algorithms all reach global optima within first 200 iterations. Since this is relatively easy task for all optimizers, time expense is important factor to consider to choose optimizers. As in table 1, GA and RHC takes only 0.0004 seconds for each 10 iterations. One max is a simple problem with one single global optima with large basin of attraction. GA and RHC's evaluation functions are inexpensive to compute and there is no local optima to be stuck at.

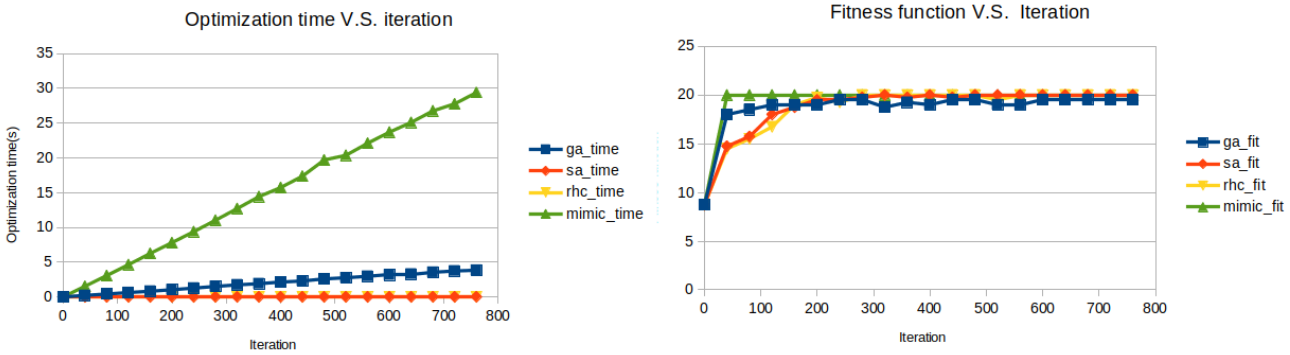


Figure 1. Comparison of four optimization algorithm for one max problem

GA is the best performer for four peaks problem

Four peaks is a problem with four local optima with wide basins of attraction designed to catch simulated annealing and random hill climbing, and two sharp global optima at the edges of the problem space. GA and MIMIC managed to reach global optima within 100 iterations. RHC and SA got caught up with local optima within first 1000 iterations. SA managed to converge to global optima after 2000 iterations (not shown in figure). However, comparing the time expense for each 10 iterations, RHC and SA take significantly less time (~100 times). In conclusion, SA is the best performer for four peaks problem.

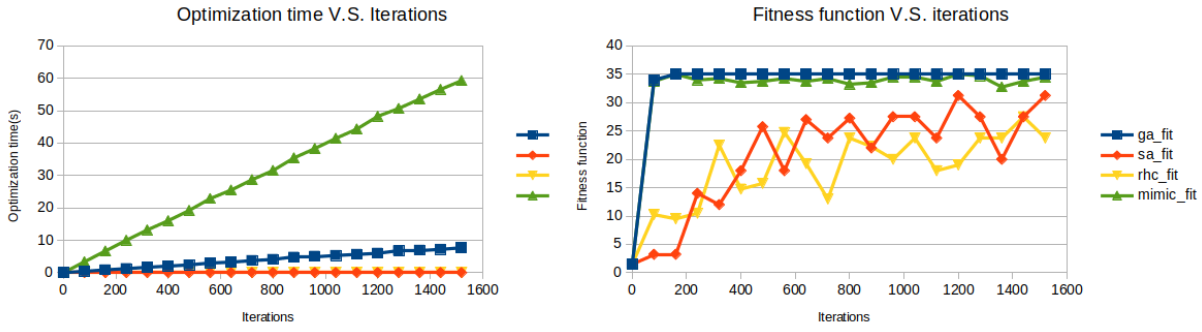


Figure 2. Comparison of four optimization algorithm for four peaks problem

MIMIC is best performer for Knapsack problem

Knapsack problem is the hardest among the three problems. As shown in Table 1, even though I chose slightly less challenging size of the problem for Knapsack problem, the time used by each algorithm is still larger than previous optimization problems. GA and MIMIC results in optimized fitness function relatively quickly within first 100 iterations. However, MIMIC's time expense is lower than GA, proving its power in solving this challenging problem. Knapsack is a classic NP-Hard optimization problem with no polynomial time solution. The strength of MIMIC was highlighted in this context, as it exploited the underlying structure of the problem space that was learned from previous iterations.

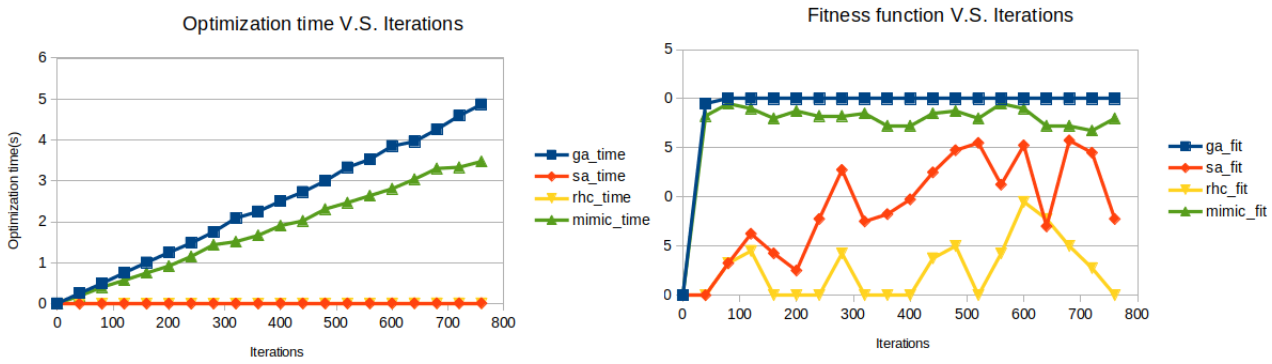


Figure 3. Comparison of four optimization algorithm for one max problem

1.3 Conclusion

Optimization Problem	Fitness Rank	Hypothesis space	Time cost/ 10 iterations(s)
One max	SA~RCH> MIMIC~GA	Small	GA:0.21 SA: 0.0004 RHC: 0.0003 MIMIC:1.60
Four peaks	GA>MIMIC>SA>RHC	Medium	GA:0.12 SA: 0.0007 RHC: 0.0004 MIMIC:3.52
Knapsack	GA~MIMIC>SA>RHC	Large	GA:0.25 SA: 0.0009 RHC: 0.0005 MIMIC:0.19

Table 1. Conclusion of comparisons of four optimization algorithms

In this section, three optimization problems are used to evaluate four optimization algorithms' performances. Difference of the 3 methods is mainly from the different approaches of how candidate solution is generated, or more specifically from how they deal with a worse solution. The idea that the fitness evaluation has to be exploited to avoid brute force searching the fact that a lower fitness solution may lie on the way to global optima has to be both handled in a proper way. Simulated Annealing balance the two by introducing a temperature parameter and took decision probabilistically. Genetic Algorithm solve it by employing multiple searching agents and make them work parallelly and independently, while MIMIC by maintaining and refining an underlying dependency tree.

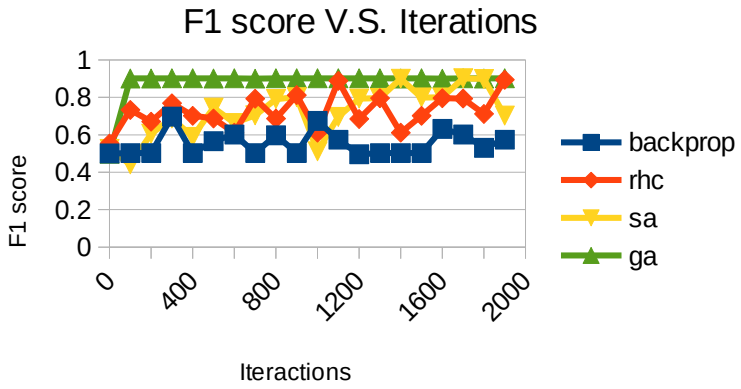
In conclusion, RHC/SA are good at combinatorial optimization problem with small hypothesis space, as they can quickly exhaust the version space and come up with a fair answer when GA/MIMIC are still struggling with refining their decision structure. However, computation investment of GA/MIMIC in early rounds would benefit them in long term for a relative difficult problem, which give them great potential for really challenging tasks.

2. Using various optimization algorithms for weights update in Neural Networks

In this section, I use synthetic classification data sets to compare different optimization algorithms. With data science being increasingly used almost all fields of study, increasingly amount of data becomes available. However, real life data sets suffers from data quality issue. In some machine learning problem, it is not which algorithm to choose but the quality of data defines the success of the project. Therefore, in my problem, I use synthetic data sets to compare different optimization algorithms.

2.1 Method

I used four optimization algorithm (gradient descent GA, random hill climbing RHC, simulated annealing SA and genetic algorithm GA) available in mlrose package. All initial comparisons are done using default hyperparameters of each algorithm. Hyperparameter tuning is then carried out for SA and GA. Parameters tuned are mutation probability and population size for GA. Parameters tuned are temperature and various cooling factor fore SA. To show all four algorithms' performances, 4 trials of training are carried out at iterations from 200 to 2000. Average classification F1 score and MSE are shown in the plots below.



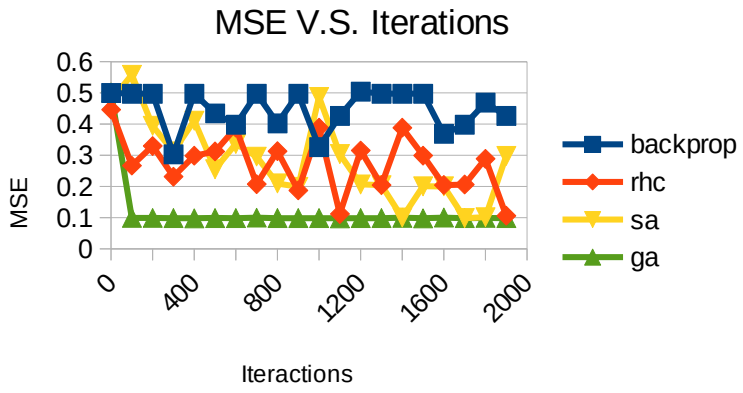


Figure 4. Comparisons of optimization algorithm performances on weight updates in NN.

2.2 Result

Initialization states dependent optimization

Especially for simulated annealing which wasted many initial rounds wandering around the neighborhood, when exactly the algorithms are going to converge is difficult to predict and subject to randomness. As elaborated by the big divergence among the 5 trials of simulated annealing experiments, iterations needed before convergence depends heavily on their initialization states, probably because the network is too simple to differentiate good and bad performers. A more complicated optimization problem such as the one in next part should give a better examination on the true capabilities of the models.

SA and GA prone to be trapped at Local optima

Besides, there are quite a few traps (local optima) that could deceive and hold the algorithms, which results no improvement for the algorithms after continuous training despite the algorithms manage to escape most of time. For example, Genetic algorithm was heavily stuck at accuracy 76.9%; Simulated annealing got fooled at 23.1% in all trials. RHC stumped a few times but escaped quickly. This is consistent with intuition that “Take not a musket to kill a butterfly”.

Performance and time cost comparison

As shown in Figure 4, GA converges to max F1 score within 100 iterations, while GA and SA manages to reach convergence at 2000 iterations. However, the time cost for each iteration for GA is 1000 times back propagation and 100 times of the RHC and SA. GA is the best performer considering convergence rate. SA/RHC are the best performer considering time cost. However, this is more chance for SA/RHC to be trapped at local optima.

	Max accuracy reached(%)	Time per iteration(s)	Min MSE reached
Back propagation	0.69	0.002	0.30
Random hill climbing	0.89	0.026	0.10

Simulated annealing	0.90	0.025	0.10
Genetic algorithm	0.90	2.15	0.10

Hyper parameter tuning for SA and GA

For GA, I varied mutation probability to see if it improve performance. The results shows that mutating 90% of the population doesn't improve overall performances. For SA, I varied initial temperature and cooling factor. We can see cooling factor and temperature seem have much similarity in the way they affect the convergence of SA: very low temperature/small cooling factor makes SA behave like a typical RHC; Very high temperature/big cooling factor significantly elongate the converging process (SA tends to spend more time exploring rather than exploiting). Their differences lie on the starting point and the divergence pattern.