CS 4641 PROJECT 2: Random Optimization

Name: CHANG Yingshan Student ID: 903457645 GT account: ychang363

Neural Network Weight Optimization Problems

1. Problem description
   I reused a binary classification problem from project1, aiming at analyzing what sorts of passengers were likely to survive the Titanic tragedy. Instead of using back propagation to find the weight of the neural network, RO algorithms are used to find the optimized weights.
2. Performance of RO algorithms

Algorithm	Iteration / pop_size (for GA)	Training accuracy	Testing accuracy	Running time (s)
Hidden layer [3]				
BP	——	0.80	0.79	6-8
HC	1000	0.63	0.62	6
	2000	0.68	0.66	7
	5000	0.79	0.82	9
	10000	0.82	0.81	17
SA	1000	0.36	0.35	6
	2000	0.45	0.51	7
	5000	0.78	0.73	13
	10000	0.82	0.76	24
GA	10	0.65	0.57	5
	20	0.69	0.62	5
	100	0.73	0.77	7
	200	0.77	0.79	10
	500	0.79	0.80	13
Hidden layer [2,2]				
BP	——	0.80	0.80	6-8
HC	1000	0.70	0.68	7
	2000	0.75	0.74	8
	5000	0.80	0.77	12
	10000	0.81	0.76	19
SA	1000	0.59	0.44	7
	2000	0.69	0.60	9
	5000	0.74	0.71	14
	10000	0.78	0.77	25
GA	10	0.67	0.64	4
	20	0.77	0.82	5
	100	0.76	0.72	7
	200	0.79	0.82	14
	500	0.79	0.78	15

3. Analysis
   a) Performance of RO algorithms
   It can be concluded that all three algorithms can achieve comparable results as the back propagation does, as long as they are provided with enough iteration times or pop_size.
   Among the three RO algorithms, SA has the slowest convergence speed. This may due to the randomness

of SA, since it randomly chooses a neighbor at each iteration and accepts it with some probabilities. The randomness of SA helps to avoid local optima, but causes to more effort on useless exploration. It is noteworthy that when we have only one hidden layer, the accuracy of SA cannot even achieve 0.5 (the accuracy of random guess) with small number of iterations. As the number of iterations increases, the accuracy of SA exhibits a rapid growth. By slight contrast, the accuracy of HC is higher than random guess even with small number of iterations. And as the number of iterations rises, the accuracy of HC experiences a steady growth. On the other hand, GA converges much faster than SA and HC. Therefore, GA achieves similar results with SA and HC with less training time, reflecting that the faster convergence of GA makes up for the longer processing time required by its complex structure.

b) Neural Network Complexity

I slightly modified the neural network complexity by adding one more hidden layer. The performance of three RO algorithms were very consistent: all three algorithms spent more time on a more complex network, but achieved better accuracy.

Traveling Salesman Problem (TSP)

1. Problem Description
   a) Goal

   The goal of this problem is to find the shortest path that visits every city in a city list for exactly one time before returning to the starting point. For this specific problem, the city list is predefined as a list of 6 coordinates in the x-y plane: [(3,2), (5,8), (4,12), (6,7), (11,13), (9,10)].

   b) State Vector

   A state vector is a permutation of [0,1,2,3,4,5], which determines the order of the cities visited by a path.

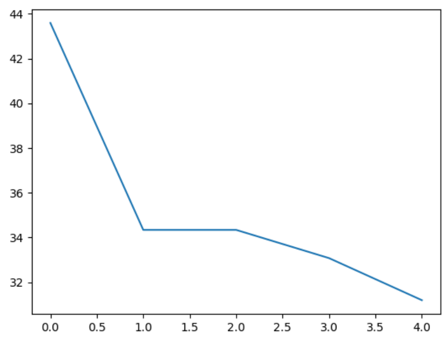
   c) Fitness Function

   The fitness function is the sum of Euclidean distance between two consecutive cities according to a path.
2. Hill Climbing
   a) Algorithm Description

   At each attempt of "climbing", a "neighbor" of the current state is generated by picking a city in the current path and put the city into the very front. For example, the state vector [0,1,2,3,4,5] has neighbors: [1,0,2,3,4,5], [2,0,1,3,4,5], [3,0,1,2,4,5], [4,0,1,2,3,5] and [5,0,1,2,3,4]

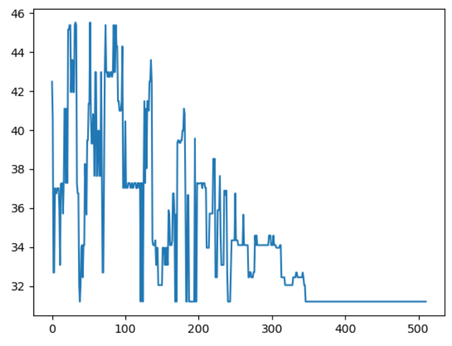
   b) Performance & Analysis

   HC runs quite fast, in less than 1 second. But HC's performance on this problem is very poor. The correct times out of 100 attempts for max_iter = 5, 10, 30 are 34, 37, 35. Increasing max_iter does not seem to help at all. When I plot the iteration curve of HC, I found that HC usually stops after 4-5 iterations, because it finds no better neighbor. This could be attributed to the fact that the fitness value of a state vector does not necessarily to be close to the fitness value of its neighbor. However, the complexity of this problem prevents us from finding an absolutely reasonable definition of "neighbor".



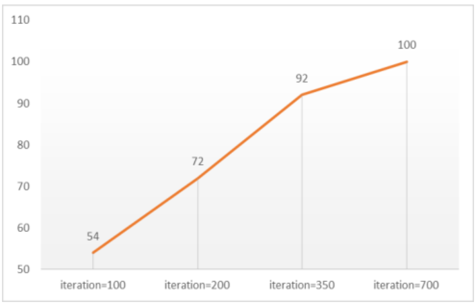
3. Simulated Annealing
   a) Algorithm description

   In each iteration, SA randomly chooses an index in the current state vector and put the corresponding city to the very front of the path.



b) Performance & Analysis

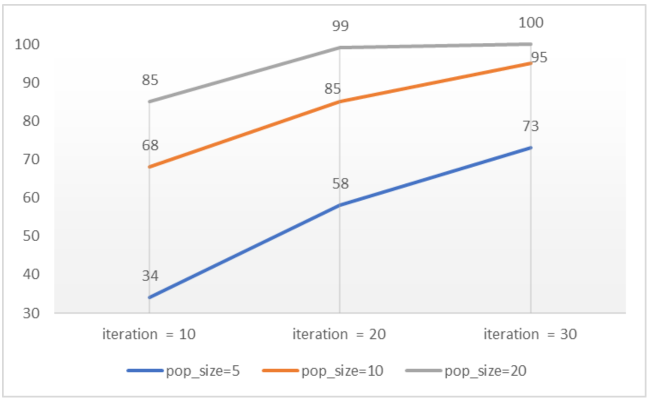
SA has satisfactory performance on this problem. On one hand, it runs very fast, in less 1 second. On the other hand, SA converges to the correct result within 700 iterations. Possible changes in order to enhance performance include increasing initial_T for the purpose of encouraging the exploration of more neighbors and increasing alpha for the purpose of reducing the speed of annealing.



4. Genetic Algorithm

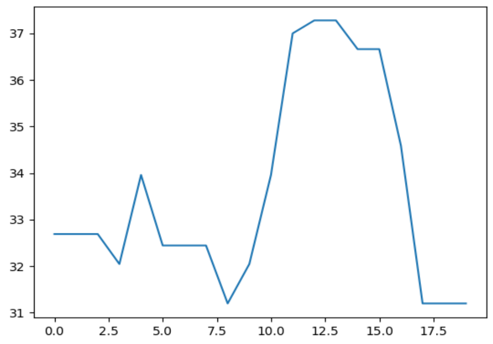
a) Algorithm description

Crossover for TSP is defined as randomly choosing an index and exchange the corresponding element of two parents. Some further processing is needed to avoid repetition in a path. Mutation is defined as randomly choosing two different cities in a path and exchange their orders.



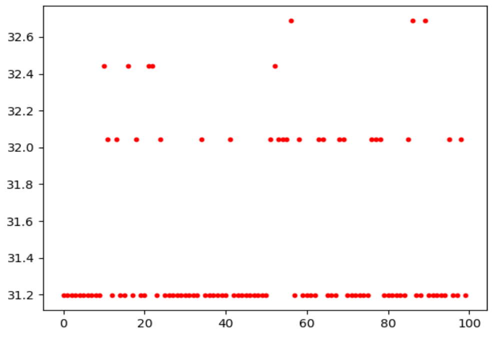
b) Performance

In terms of running time, the algorithm completes 100 times of searching in less than 5 seconds. Generally speaking, GA has the best performance on TSP problem. From the chart we can see that GA achieves high accuracy with small number of iterations and pop_size.



Due to the nature of TSP, while it is hard for greedy algorithms like HC and SA to output the correct answer, this problem highlights the strengths of GA: 1) easy to handle large number of variables; 2) be able to carry out a thorough exploration of the searching domain with cross over and mutation; 3) can jump out of local optima.

GA converges very fast with less than 20 iterations. This could be attributed to the fact that in each iteration, GA explores a relatively large range of the searching domain and records the best state vector. The large "searching scope" also explains why GA can avoid local optimum smartly. Furthermore, even though sometimes GA makes mistakes, the mistakes are acceptable as illustrated by the following chart: all incorrect answers are actually already very close to the true optima.



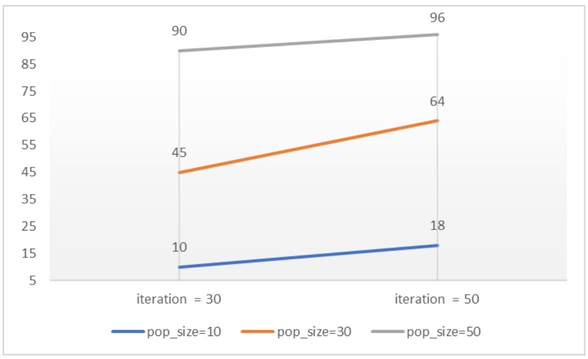
5. MIMIC

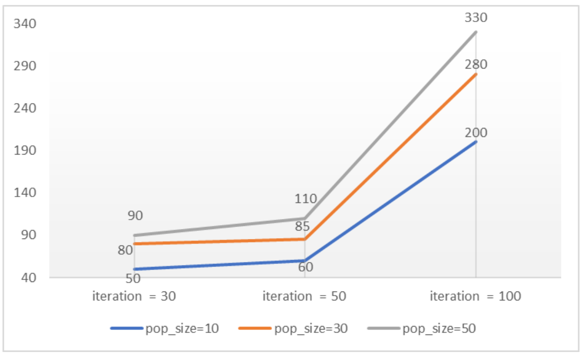
a) Algorithm description

Mlrose has a predefined function for solving TSP problem. I initialized a TSPOpt object by passing the coordinate list of city locations and used mimic algorithm predefined by mlrose to do the search.

b) Performance

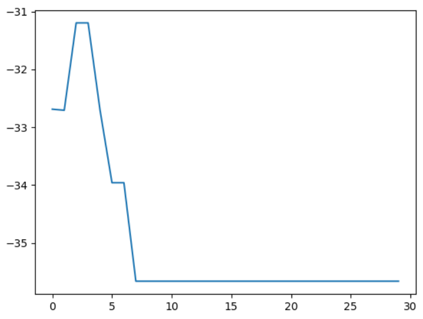
The performance of MIMIC is similar to GA, which converges to the correct optima with 50 iterations and pop_size = 50. However, MIMIC is inferior to GA in a sense that in order to achieve similar results, MIMIC's running time is much longer than that of GA, increasing dramatically as the number of iterations grows.





c) Analysis

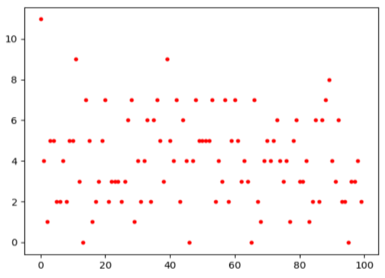
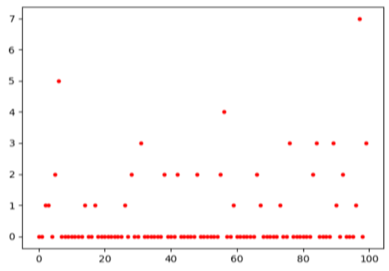
As can be seen, MIMIC performs well in this complex problem by structuring the solution domain and iteratively sampling out subsamples with larger fitness value. This allows MIMIC to converge very fast, as shown in the chart that sometimes MIMIC converges within 10 iterations, far less than the predefined maximum number of iterations.



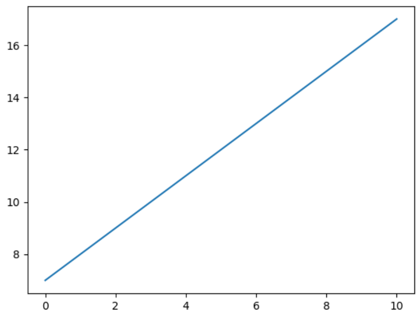
"Single Optima" Problem

1. Problem description
   a) State vector
   Each state vector is an array of length 6, each element of the array can take integer values from 1 to 6.
   b) Fitness function
   Sum of the absolute difference for each array element between the current state vector and a "target vector". The target vector is predefined as [3,3,3,3,3,3]. For example, the fitness value of [1,1,1,1,1,6] is 13.
   c) Goal:
   Find the minimum of the fitness function. The correct minimum value is 0, and the corresponding state vector is [3,3,3,3,3,3].
2. Hill Climbing
   a) Algorithm description
   The initial state vector is randomly chosen to be an array of length 6, whose elements are integer values from 1 to 6. At each iteration, the algorithm looks at "neighbors" of the current state vector and replaces the current state vector by the best neighbor. A "neighbor" of a current state vector is defined as: any array with only one digit different from the current state vector, and the difference is 1. For example, neighbors of our target vector [3,3,3,3,3,3] are: [2,3,3,3,3,3], [4,3,3,3,3,3], [3,2,3,3,3,3], [3,4,3,3,3,3], [3,3,2,3,3,3], [3,3,4,3,3,3], [3,3,3,2,3,3], [3,3,3,4,3,3], [3,3,3,3,2,3], [3,3,3,3,4,3], [3,3,3,3,3,2], [3,3,3,3,3,4]. The updating keeps going on until no improvements can be made within the neighborhood.
   b) Performance
   The parameter <iteration> defines the maximum times of "climbing" before termination. Hill Climbing will converge within 15 iterations. The following table summarizes the running time and accuracy of this algorithm with different iterations. The accuracy is calculated by running the same algorithm 100 times and checking how many times it outputs the correct optimum value for the fitness function. The charts on very right column show the 100 output results.

Iteration	Time	Accuracy	

5	< 2s	4/100	
8	< 2s	41/100	
10	< 2s	73/100	
15	< 2s	100/100	

c) Analysis



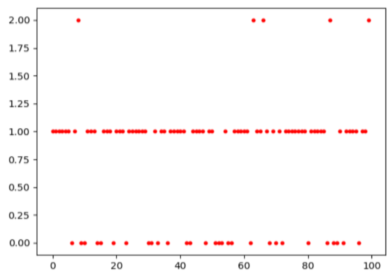
Hill Climbing has very good performance on this problem, in terms of both short running time and small number of iterations for convergence. This could be attribute to the fact that this problem has no local optima, so that HC won't get stuck.. Based on how we define "neighborhood" for a state vector, every time of "climbing" the best fitness value can only be improved by 1. Thus, the speed of convergence is uniform. The following chart shows the iteration curve for a typical convergence process, with x_axis representing number of iterations and y_axis representing the best fitness value currently found. Note that since this is a minimization problem, I made some variations in the code to convert it into a maximization problem. So, the optimal value of the function defined in my code is 20.

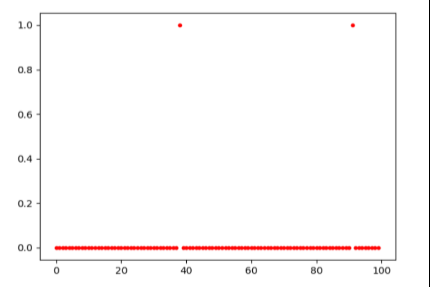
3. Simulated Annealing

a) Algorithm description

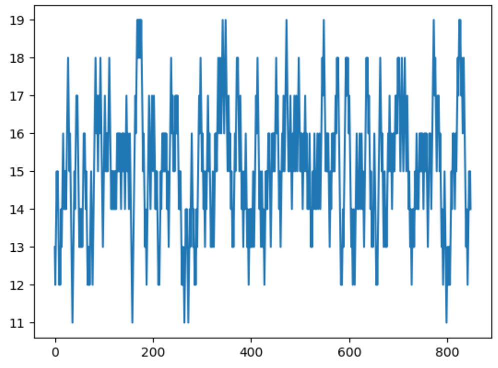
SA models the annealing process in metallurgy, in which random local search that leads to a downhill move is allowed at the beginning of the optimization process. To be more specific, if a local search generates a better fitness value, it is sure to be accepted, otherwise, the new state will be accepted with a certain probability and this probability will go down as time goes by. In this case, each random local search randomly evaluates a vector within the "neighborhood" of the current state vector and decides whether to accept it.

b) Performance

Iteration	Time	Accuracy	
420	< 1s	29/100	
850	< 1s	54/100	
1750	~ 2s	78/100	

4000	~ 3s	98/100	

c) Analysis



SA also performs well on this problem because there is no tricky local optima. Although it seems that SA needs much more iteration to converge than HC, it should be noticed that while in HC, we analysis all state vector in the neighborhood in every iteration and pick the best one, when it comes to SA, only one random state vector from the neighborhood is picked, and it is still not guaranteed that this state vector will be accepted. This significantly increased the randomness of this algorithm (as shown by the fluctuating line in the iteration curve), which leads to the low convergence speed. But it will converge eventually because as time goes by, it is less and less possible to accept a state vector that makes a downhill move.

However, there is still some ways to speed up the convergence. Firstly, I have to make it clear that the number of iterations in SA is determined by initial_T, alpha and terminal_T, where T is the temperature that controls the probability of accepting a worse state vector, and between each iteration T is decreased by multiplying with alpha. Then, it is essential to notice that this problem has no local optima, so it is no use to accept a worse state vector. We may reduce initial_T to decrease the probability of accepting a worse state vector, or reduce alpha to speed up the decrease of T.

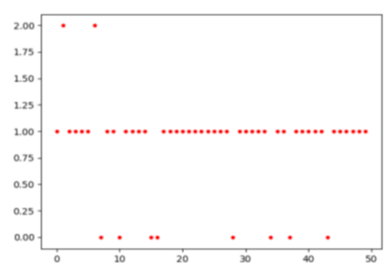
4. Genetic Algorithm

a) Algorithm description

GA models the principles of genetics and natural selection. An initial population is generated by a certain number of randomly selected state vectors. In each iteration, individuals with higher fitness value will survive and they will produce the next generation by crossover and mutation. For this specific problem, crossover refers to exchanging a section of the state vector between two parents and mutation refers to randomly choosing a digit and increase or decrease its value by 1.

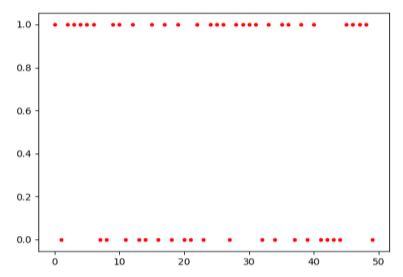
b) Performance

Parameters	Time (for running 100 times)	Accuracy	

Iteration=100
Pop_size=200
	~ 8s	16/100	

Iteration=200
Pop_size=200
	~ 10s	28/100	

Iteration=100
Pop_size=400
	~ 15s	34/100	

Iteration=200
Pop_size=400
	~ 20s	41/100	

c) Analysis

It can be easily concluded that GA is not a good algorithm for this problem. Its running time is significantly greater than HC and SA. Moreover, its accuracy is not satisfactory. However, when we look at the scatter plots of the results, we can discover that at most times GA was very close to the correct answer. This suggests that although GA's performance is poor by just looking at its accuracy, it is not that bad because it can always provide a really close answer, which should be good enough especially for those extremely hard problems defined in complicated domains.

5. MIMIC

a) Algorithm description

MIMIC refers to Mutual-Information-Maximizing Input Clustering. It attempts to communicate information about the cost function obtained from one iteration of the search to later iterations of the search directly. For implementation, I used a python package: mlrose.

b) Performance

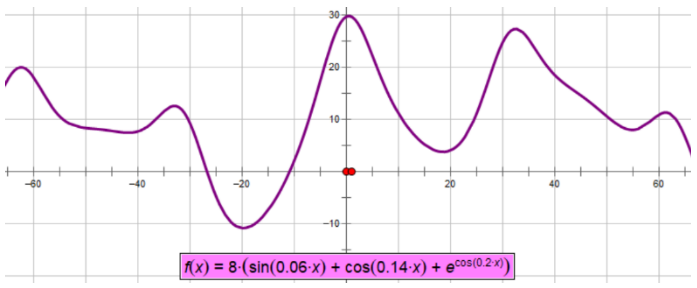
Pop_size	Max_iter	Time (for running 100 times	Accuracy
100	20	~16s	24/100
	100	~ 40s	30/100
200	20	~30s	70/100
	100	~70s	72/100
300	20	~32s	80/100
	100	~78s	90/100

c) Analysis

Just like GA, there is also a trade-off between running time and accuracy associated with MIMIC. If we need higher searching accuracy, we need to afford long searching time.

However, there are two interesting points I'd like to put forth. Firstly, the accuracy has more to do with population size than iterations. With a fixed pop_size, increasing max_iter does not improve the accuracy much. By contract, the running time is more closely associated with max_iter, as can be seen from the table that for each pop_size, the running time for max_iter=100 is nearly double that for max_iter=20

"Multiple Optimum" Problem



1. Problem description

a) State

The state for this problem is a single integer value from -63 to 64

b) Fitness function

f(x) = 8 * [sin(0.6x) + cos(1.4x) + e^(cos(0.2x))]

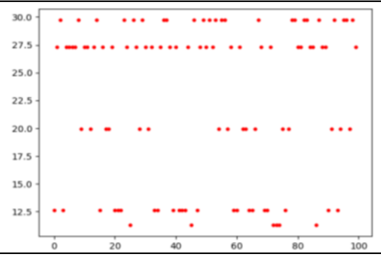
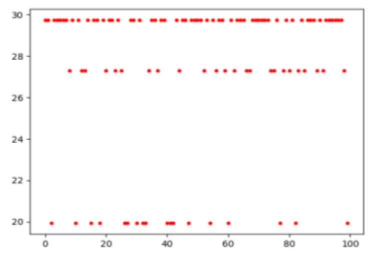
c) Find the maximum value of f(x), where x is an integer in the interval [-63, 64]

2. Hill Climbing

a) Algorithm description

The initial state is a random integer in the interval [-63, 64]. In each step, the algorithm checks every integer value within a neighborhood of the current x and update x with its best neighbor.

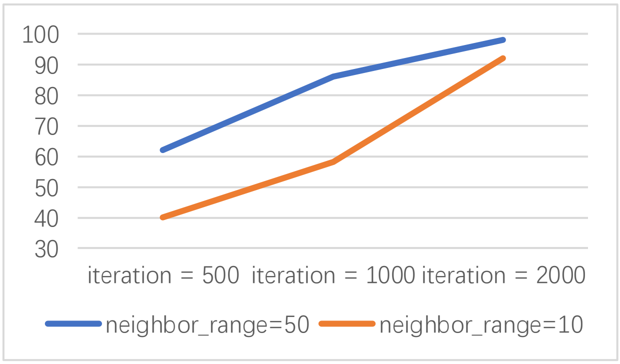
b) Performance

Neighborhood_range	Iteration	Time	Accuracy	
10	10	< 1s	29/100	
	50	< 1s	26/100	
30	10	< 1s	53/100	
	50	< 1s	58/100	

c) Analysis

This problem shows the drawback of HC. Since HC is defined in a greedy way, it keeps on looking for a higher point and immediately stop when it can not see a higher point within a small neighborhood. It turned out that all incorrect outputs were resulted from getting stuck in local optimum and this can be illustrated by the two charts. As can be concluded from the above table, increasing the number of iterations seems useless if the algorithm is "shortsighted", i.e. with small neighborhood range. Enlarging the neighborhood range substantially improves its performance.

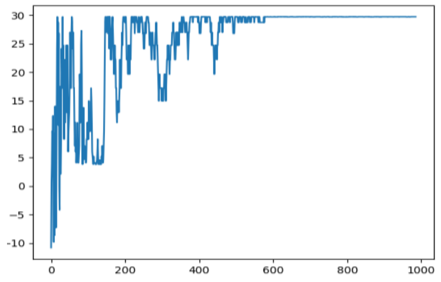
3. Simulated Annealing



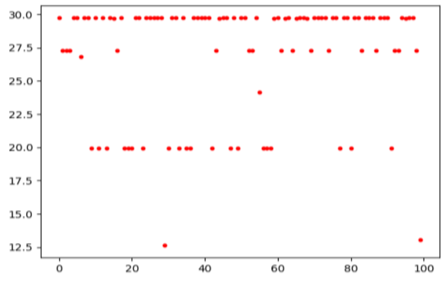
a) Algorithm description

Similar to the first problem, SA randomly chooses an integer value within the neighborhood of x and decides whether or not accept it. This problem does not require complex analysis, so SA runs quite fast, usually in less than 1 second.

The chart summarizes the accuracy of SA with different iterations and neighbor_range. Again, enlarging neighbor_range helps a lot, since it gives the algorithm a broader vision.



A typical iteration curve for SA demonstrates that the fluctuation gets smaller and smaller as the algorithm accepts less and less worse states. Finally, it converges to an optimal point as it can no longer find a better state near the current state. It is also interesting to note that although SA tries to avoid getting stuck in local optima by temporarily accepting worse states, it still makes lots of mistakes. Improvements can be made toward either broadening its horizon by increasing neighbor_range, or encouraging SA to explore more "temporarily worse states" by enlarging initial_T and alpha, in order to enhance its ability to avoid local optimum.



4. Genetic Algorithm

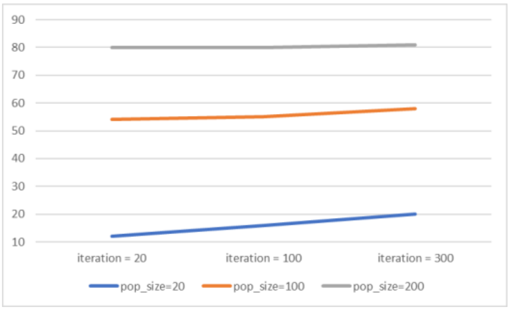
a) Algorithm description

In order to solve this problem by GA, encoding of integer values along x_axis is involved. Binary encoding is used to convert each integer value in the interval [-63,64] into a binary string of length 7.

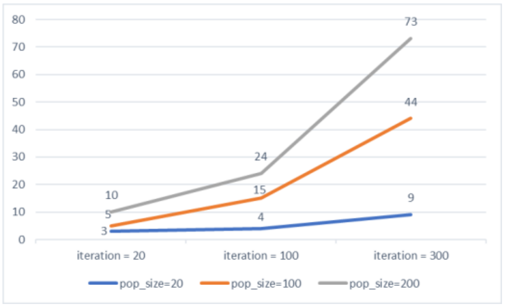
When calculating the fitness function, each state vector is converted back to a decimal number before being evaluated by f(x).

b) Performance

The first chart on the right summarizes GA's accuracy out of 100 times of searching with different iterations and population sizes. It is obvious that pop_size has greater influence on performance. With a fixed pop_size, increasing number of iterations does not show much improvement. This result is consistent with the theory of "gene diversity" in biology, which points out that a population with greater gene diversity will survive. Since the initial population in GA is chosen randomly within the state vector domain, larger pop_size means higher diversity at the very beginning, leading to better chance of finding the global optima. By contrast, increasing number of iterations only lead to more generations, but each generation only involve slight variations based on the previous one. If the population itself is not good enough, it is much harder to reach the true optima.



The second chart on the right shows the time for running the algorithm 100 times, which is positively correlated with both iteration and pop_size. GA runs significantly slower than HC and SA, perhaps not only because the inherently complex structure of the algorithm, but also due to the frequent conversion between binary sequences and decimal numbers.



5. MIMIC

a) Algorithm description

In MIMIC, each integer value is also encoded to a binary sequence of length 7. As for coding, I defined a custom problem object class (fxOpt) to provide functions for calculating the fitness function as well as initializing population.

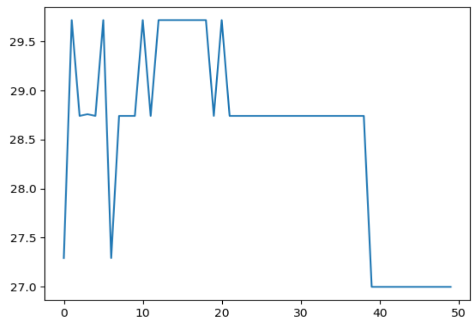
b) Performance

Max_iter	Pop_size	Time (for running 100 times)	Accuracy
20	20	~ 40s	14/100
	50	~ 50s	42/100
	100	~ 60s	51/100
100	20	~90s	23/100
	50	~100s	60/100
	100	~120s	69/100

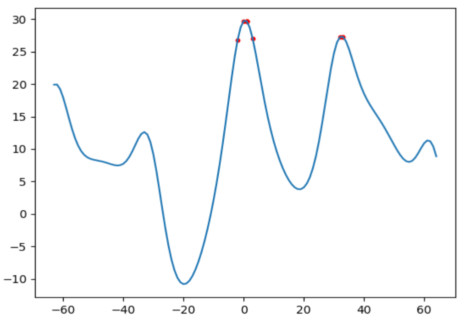
Just like GA, the performance of MIMIC is highly dependent on pop_size, since pop_size determines the "scope" of the algorithm at each iteration. The larger the pop_size, the more likely that the algorithm has a global view of the solution domain. Increasing number of iterations causes the running time to rocket up. Although it does help contribute to enhancing the accuracy, its affect is limited.

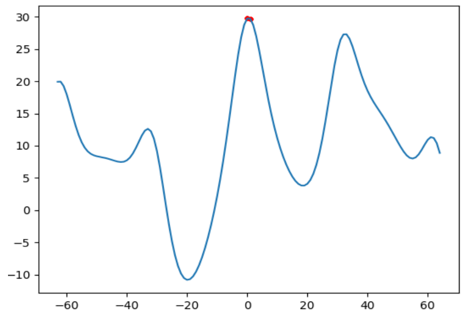
c) Analysis

The above chart shows a typical iteration curve of MIMIC, which keeps track of the best child at each generation. It is noteworthy that there are downhill moves even after large number of iterations. A possible explanation is that when producing the next population, MIMIC samples out parents from the previous generation and makes variations on them without knowing what kind of variation is "good". Thus, it is likely that a child is worse than its parents after reproduction.



It is also helpful to look at where the algorithm ends at when it produces incorrect answers. Judging from the chart on the left, which shows 100 results with max_iter=20, pop_size=20, with only 14 results being correct, it is clear that with small max_iter and pop_size, MIMIC sometimes falls in local optimum, while in other times arrives at somewhere close to the global peak. The right chart shows 100 results with max_iter=100, pop_size=50, with 50 results being correct. It should be noted that although the accuracy is only half, MIMIC have avoided all local optimum. The incorrect answers are only 1 or 2 steps away from the correct answer. In this sense, MIMIC perform quite well to give an approximate optimal answer.





Conclusion

Based on all the analysis above, we can conclude the advantages and disadvantages of HC, SA, GA and MIMIC.

	Advantage	Disadvantage
HC	1) Simple, fast, easy to understand and implement 2) Has the best performance on single-optima problems	1) Greedy nature, easy to get stuck in local optima if the neighbor range at each iteration is narrow 2) Not very good at capturing the underlying nature of a complex problem
SA	1) Still has greedy nature, but tries to avoid local optima and make more exploration of the solution space 2) Capable of locating the global optima if the exploration range is large	1) Low convergence rate due to the randomness nature 2) Easy to be fooled by local optima if the problem has complex underlying structures.
GA	1) Lower training time on complex problems compared with other RO algorithms 2) Capable of exploring a large area in the solution domain, smartly avoiding local optimum	1) Requires relatively longer running time on simple problems compared with other RO algorithms. 2) Sometimes requires an encoding/decoding method to convert a state vector to a gene sequence 3) Good at making close approximations, but may not precisely locate the exact position of the global optima,
MIMIC	1) Converges fast, able to generate acceptable result within few iterations 2) Good at discovering the underlying structure of a problem	1) Long running time 2) Require large memory space 3) Running time increases exponentially as the number of iterations increases

Reference

1. https://www.cc.gatech.edu/~isbell/papers/isbell-mimic-nips-1997.pdf
2. https://mlrose.readthedocs.io/en/stable/source/opt_probs.html
3. https://bambielli.com/assets/pdf/Comparison-Of-Four-Randomized-Optimization-Methods.pdf