Randomized Optimization

CS-7641 Assignment2

Zhengyang Chen GTID#903338861

1 Introduction

This passage indicates to make experiments on randomized optimization. The contents maily divided into two parts, the first part is to use three randomized optimization function - the randomized hill climbing, the simulate annealing algorithm, and the genetic algorithm- to find the best weight for back propagation neural network. The second part is to utilizing all the four algorithms- the previous three plus MIMIC, to solve the randomized optimization problem, compare and highlight the advantage of them.

For the first part I use the dataset from last assignment, the student-por.csv from uci repository, the task ,again, is to predict the final grade of the students in Portuguese class.For the second part, I chose 3 simple problems- the continuous peaks, the traveling salesmen,and the knapsack problem, to implement the optimization, separately highlight the advantage of simulate annealing, genetic algorithm, and MIMIC.

The programming is implemented in jython, and resources and libraries comes from ABAGAIL package.

2 Randomized Optimization in NN weight finding

Given the bp Neural Networks taken as the object to optimize, we picked 3 algorithms - the randomized hill climbing search pattern (rhc), the simulate annealing algorithm(sa), and the genetic algorithm(ga).As what the origin program provides, we run the optimization every time we train a network to define the best value, thus it may concluded to a distribute, and so the diagram I choose to plot is the distribution figures.

Randomized Hill Climbing is, surprisingly, came out to be the best performance among all 3 algorithms. As it is such a simple algorithm, the result comes that it has the mean accuracy of 74.38(see Table 1.1), and are above the other 2, it also received the minimum of standard deviation. The possible reason for this result may be that for the function underlying for optimization, is kind of a smooth one with less distributed local optima, and the global optima is of large influence on the optimization descent.Moreover, as the test runs among all the instances, the randomized hill climbing can have the opportunity to regenerate the start points, so as to perform better in global search.

However, if we take a look at the brother of randomized hill climbing, it is astonished to find that it perform so badly, that the mean is 50.57 - which means, it is not optimizing, it is guessing, or gambling. But if the attributes of generated fitness function should be somehow the same aspect, what can cause the difference between these two? Look back to the program itself we can notice that in this situation, it is the parameter of SA algorithm that should be responsible, the starting temperature is set extremely high while the cooling rate is significantly low (as 0.99,seems not cooling at all), thus let's make an assumption, we are having a pinball on an bumpy pan(yes, a bumpy pan with local and global 'optima'), and the high starting temperature and the low cooling rate means I m violently shaking this pan without stopping, thus the ball will randomly be anywhere, rather than slow down and drop into the optima.

Mathematically saying, the reason is that the SA algorithm does not converge, so that the result is coming out to be absurd.

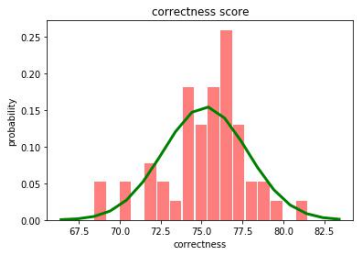

Figure 1.1a -rhc

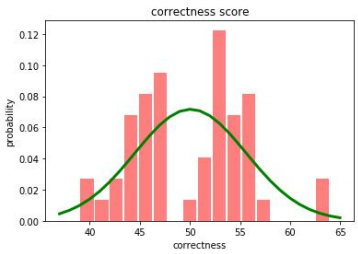

Figure 1.1b - sa

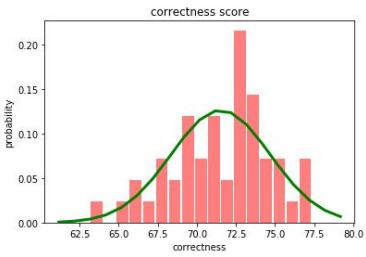

Figure 1.1c - ga

The genetic algorithm is such performing fairly, but the time cost is significantly higher than the other two(see table 1.1), it is because that the nature of genetic algorithm needs to compute every member in the population - and unfortunately, in this case, not in binary strings. The consequence is that it will devour plenty of time ranking them, crossing them, and producing the offspring, as well as dealing with sudden change. Each step, compared with the climbing and annealing, is demanding on time.

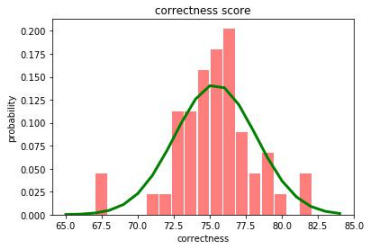

Figure 1.2a -rhc

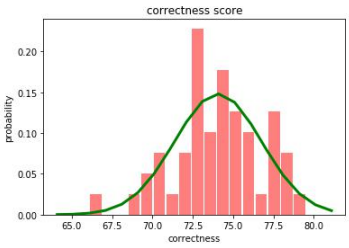

Figure 1.2b - sa

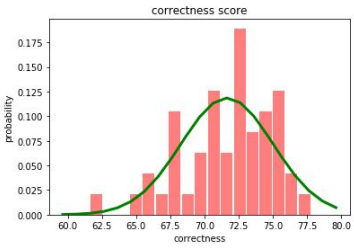

Figure 1.2c - ga

Table 1.1

Algorithm	Hill climbing	Simulation Annealing	Generic Algorithm
Mean accuracy	74.78	50.57	71.34
std	2.35	4.38	2.89
Mean training time	2.100	2.107	28.54

What can we do to make improvement? For sure the SA algorithm is not doing well now, so the first action is to adjust the SA algorithm's parameter - lower the starting temperature, set the cooling parameter to .5 instead of .99,etc, then the optimization come as table 1.2 - and we can see that the improvement is explicit, the SA are performing now quite as well as the hill climbing.

We can also see the improvement from figure 1.3a and 1.3b

Table 1.2

Algorithm	Hill climbing	Simulation Annealing	Generic Algorithm
Mean accuracy	75.39	74.07	71.65
std	2.81	2.69	3.36
Mean training time	2.135	2.128	28.66

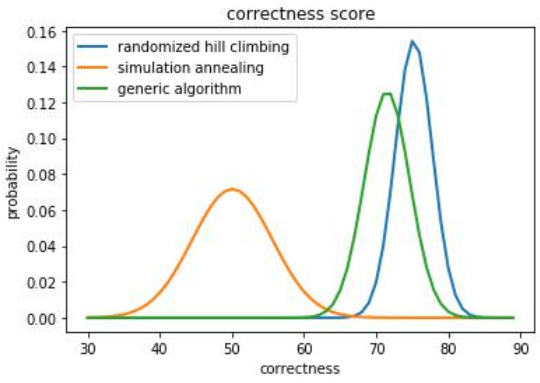

Figure 1.3a-original

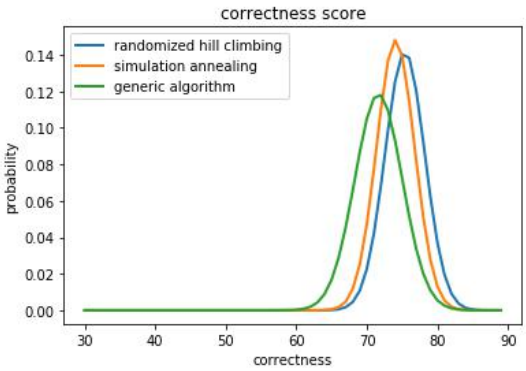

Figure 1.3b-improved

3 Randomized Optimization Problems Application

3.1 Continuous Peaks Problems

3.1.1 Problem description

The continuous peaks problem, is one of the peak problems and is an advanced explore based on 4-peaks and 6-peaks problem. The 4 peaks problem is defined as follows:

$$FourPeaks(T, X) = MAX(head(1, X), tail(0, X)) + Reward(T, X)$$

$$Reward(T, X) = \begin{cases} 100 & \text{if} \quad (head(1, X) > T) \wedge (tail(0, X) > T) \\ 0 & \text{otherwise} \end{cases}$$

$$head(b, x) = \text{number of contiguous leading bits set to b in X}$$

$$tail(b, x) = \text{number of contiguous trailing bits set to b in X}$$

Given input X which is composed of N binary elements, and the peaks is to maximize the above.

Moreover, the continuous peaks problem defines a binary string distribution of anywhere instead of at the start and end of the solution string. The reward is given when they are greater than T contiguous sets to 0, and greater than T contiguous sets to 1.

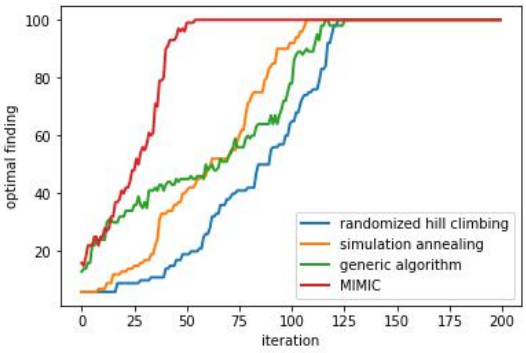

Figure2.1.1

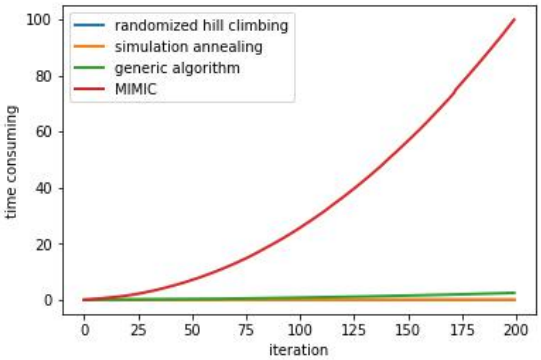

Figure 2.1.2

3.1.2 Analysis of diagrams

The first trial of the continuous peaks problem optimization is by handling two variables -

the given length of input binary string as N, and the iteration of times of the algorithm it self.While we also compare them in time accumulation during iteration and average time consume value when string length increases.

Figure 2.1.1 indicates the optimal found by 4 algorithms during iteration, with the parameter of N set to 100, and Figure 2.1.2 is the time accumulation while iteration continues. As we can see from the diagram, the 4 algorithms all converged to the optima points, but if we made a comparison between as follows:

Randomized climbing are performing fairly while compared with the other algorithms,it takes more iterations to reach the optima, GA algorithm also took longer iterations and perform fairly, the MIMIC seems faster, however, if we take a look at figure 2.1.2, the gorgeous time cost accumulates let mimic take even longer to converge - even it takes less iterations.

Table 2.1

Algorithm	Simulation Annealing	MIMIC	Hill climbing	Generic Algorithm
Converge iteration	102	56	117	121
Converge time	0.032	8.473	0.055	1.748
Converge value	100	100	100	100

However, as we see all 4 algorithms are converged and get the optimal at the same value, thus I began to think that there' s possibility that this function generated are likely to have distinct optimal, or there's possibility that all the 4 algorithms are reaching the local optimal (well, maybe not, because generic algorithm are likely to 'escape from trap'), and based on this doubt I continued the experiment.

Given output figure 2.1.3 and 2.1.4, I am placing restriction on the iteration but change the value of input string length, meanwhile change the cooling rate to a sharp value (low as 0.25) and the output diagram can be seen as the figures. It seems that the distinct difference are coming - in horizontal view the input numbers does not matter too much on the optimal - unless it is too low to cause instability of the diagram at the beginning, however the sharp rate of cooling does has an effect - compared with the figures above, the SA algorithm seems getting trapped, and also it seems no difference in time when input numbers increases.

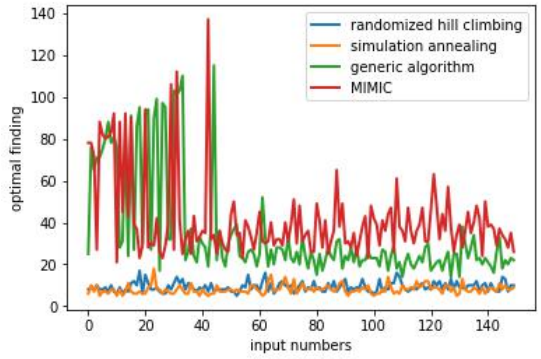

Figure 2.1.3

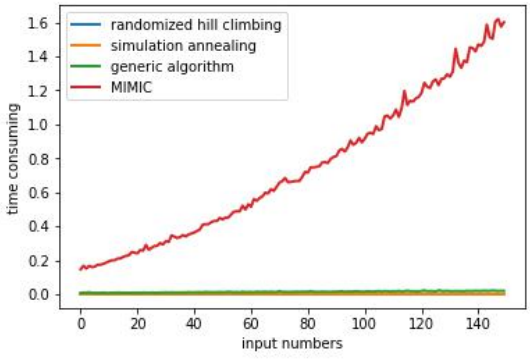

Figure 2.1.4

Thus the next experiment is launched - to see the cooling rate affects the performance. In figure 2.1.5 the cooling rate is set to 0.95 and in 2.1.6 is 0.65, we can see from the graph that the

low value of cooling rate led to the lower converge threshold, and that is obviously because the too much cooling speed let the algorithm get trapped in local optima.

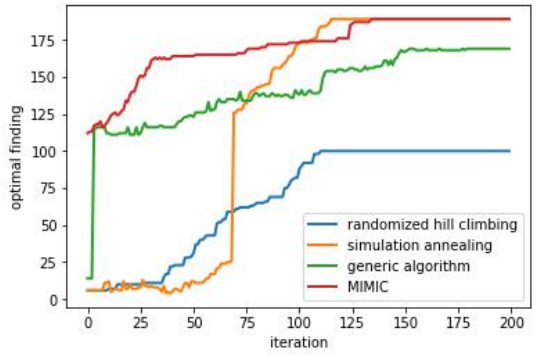

Figure 2.1.5

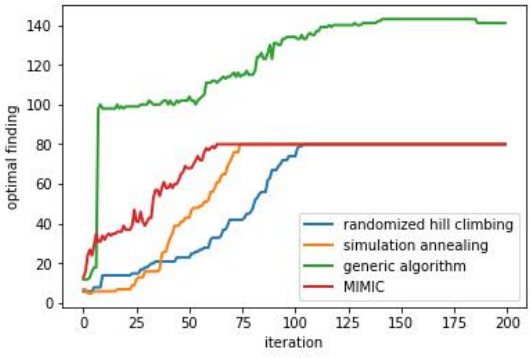

Figure 2.1.6

3.1.3 Conclusion

As the practice experiment given above, it can be concluded that in the typical problem where the size is limited and thus the problem scope is very likely to be the local optima finding, and the advantage of SA algorithm maximizes - it is very powerful to do local search while the time cost is extremely low. However, when utilizing the SA algorithm, the vulnerability of the parameter- the start and end temperature, the cooling rate, are to be concerned. If the start temperature is too high it is doing more random searching, and are likely to jump into a poor result. If the end temperature is set too high, the function is less likely to converge,as for the cooling rate, a faster cooling rate will take less time but if set too low it will exit before start searching. In this task of continuous peaks, the SA algorithm can make better performance if the parameter is appropriately set.

3.2 Traveling Salesman Problem

3.2.1 Problem description

The traveling salesman problem is a classical optimal problem described as a salesman in a point cloud of city and want to find the shortest path that can visit all these cities without duplicating the visit.In graph theory aspect, it is to optimizing a traversal through a graph. The sum of visited graph edges should be minimal and any node in the graph should only be visited once.

3.2.2 Analysis of diagrams

This problem is to find the shortest route, however to implement the algorithm we are figuring out the maximum value of the inverse of route length.

Figure 2.2.1 and figure2.2.2 indicates the curve of the four algorithm by handling the iteration variables, as we can see from figure 2.2.1, the poorest algorithm in this situation should be the randomized hill climbing, since it begins to converge at a value lower than others, moreover, since this graph is the inverse of the lowest optima, the original distinct between local optima and global should be more unsatisfactory.

Take another look at simulation annealing, it performs no better than randomized hill climbing except tiny time cost advantage, since they are from the same basic prototype- to seek graphically - by saying graphically, I mean the searching of explicit value of optima - higher value of the problem output. However, in this problem given, the same limit is that it can be trapped by local optima. The TSP problem is quite a dynamic problem and the local optima can be accessed

easily, thus the limitation of it lead to further thinking of the algorithm that can get rid of it.

Then we take a look at Generic Algorithm and MIMIC, from the figure given we can easily recognize that the GA is of outstanding advantage in convergence and development. It seems not trapped at the local optima.However, figure 2.2.1 are more based on iteration, if we need to clearly present the difference, the other diagrams should be included.

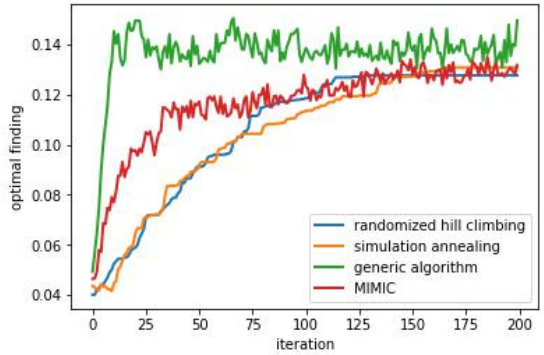

Figure 2.2.1

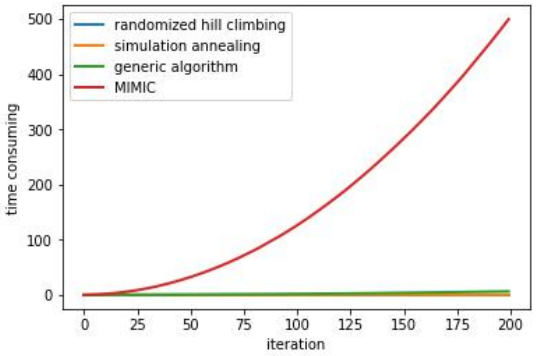

Figure 2.2.2

Let's take a look at the following two figures, when we place a limitation on iteration at value of 100, it is obviously that the GA takes the lead. The other algorithm's weakness is distinct - they seemed all get trapped!(figure 2.2.3)

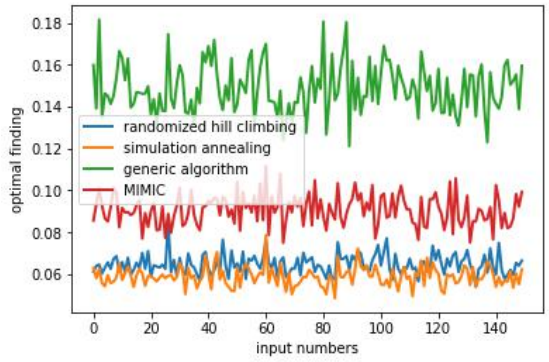

Figure 2.2.3

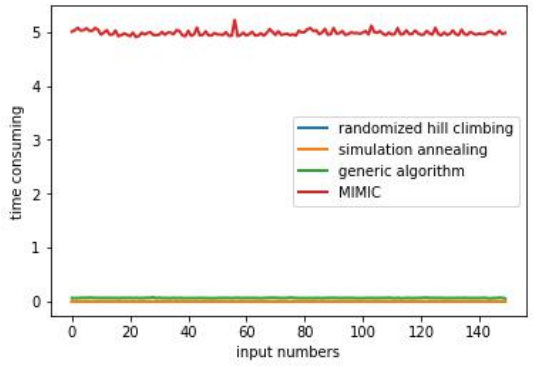

Figure 2.2.4

Why GA algorithm can be a superpower here? We should consider the nature of GA algorithm.Firstly, the GA algorithm start from cluster searching, and have the potential of paralleling, thus in the TSP problem, it can take several comparison between different individuals, and the solution is independent from the initial value- which is the trait obviously what TSP problem also obtain. Secondly, the GA algorithm are based on three parameters, in which one of them is causing sudden change and others are dealing with evolution - which obviously introduce the random variable globally, and thus definitely will not be trapped by local optima. As many local optima may occur in this problem, the advantage of GA is enlarged.

What about MIMIC? It is true that MIMIC also perform better than the other two, however, the advantage of convergence does not really take place in TSP problem, but the possibility of being trapped in local optima, does! Moreover, if we take a look at table 2.2,we can also see that considering the expensive time cost of MIMIC (also can be accessed from figure2.2.4,where input numbers does not really contribute a lot to time cost accumulation), the advantage of MIMIC is counteracted, or even became the negative effect.

Table 2.2

Algorithm	Generic Algorithm	MIMIC	Hill climbing	Simulation Annealing
Converge iteration	21	146	129	166
Converge time	0.156	265.729	0.023	0.03
optima	0.149463	0.133317	0.127612	0.130749

3.2.3 Conclusion

The GA algorithm is absolutely the best algorithm to solve TSP problem compared with those other 3, the greatest advantage is its random nature, multiple start point, the independence from start value, as well as the strong global search ability. However, we can also implement this problem by SA algorithm, which again, have the low cost. The most difficult part for utilizing the SA algorithm here, is to define the best parameter of it, if the parameter defined can help get rid of the local optima(and exactly there's the possibility), we can have maybe even better solution regarding on low time cost, however, the low time cost on the algorithm , are on the cost of high time cost of finding the parameter - hence for me, I choose GA algorithm.

3.3 Knapsack Problem

3.3.1 Problem description

Knapsack problem is another classical problem in algorithm and operational research, and is an NP problem based on combination optimization. The definition of this problem is simple: Given a knapsack of limited capacity of weight, given a group of items which have the weight and value, the target is to find the combination of item putting into the knapsack, and let the sum value of the item be the maximal.

3.3.2 Analysis of diagrams

We can first take a look at the diagram of 2.3.1 and 2.3.2, and notice that the two algorithm soon get stuck in the local optima when we increase the iteration. The other two are performing well, however, you may say as above- the figure 2.3.2 shows the significant time cost of MIMIC.But, in this problem, the time cost of MIMIC, seems not be a problem, and even I can declare that MIMIC is the best solution to this problem, then why?

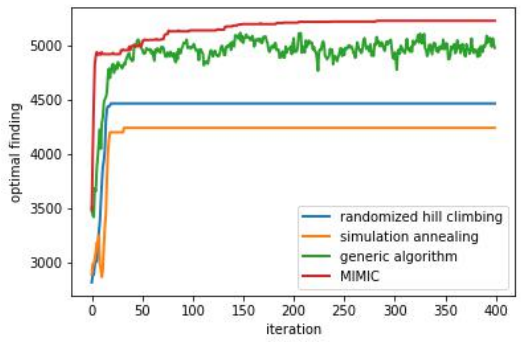

Figure 2.3.1

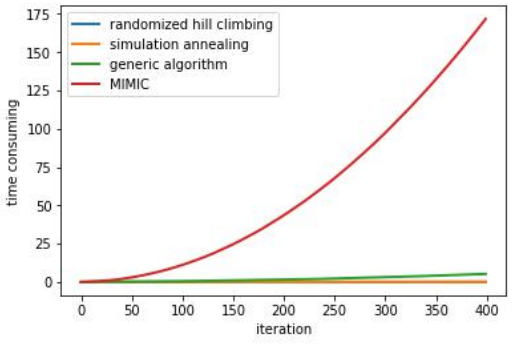

Figure 2.3.2

Yes exactly from the latter figure we can say that mimic is time consuming as always, but notice that from figure 2.3.1, Mimic is the fast to reach the optimal point. As figure 2.3.1 is in large scale of iteration, let us zoom in the graph.

What did you discover from figure 2.3.5 and 2.3.6? The MIMIC has reached the optimal very

soon and what are the time cost line? It is just something like a horizontal line, at very low cost,just as the other algorithm do.

Then what about 2.3.3 and 2.3.4? It is even more explicit that the optimal is the fastest runner, faster than anyone else!

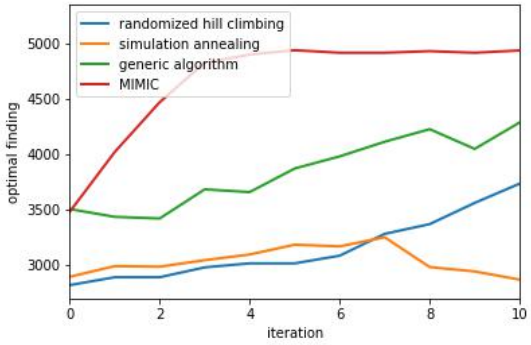

Figure 2.3.3

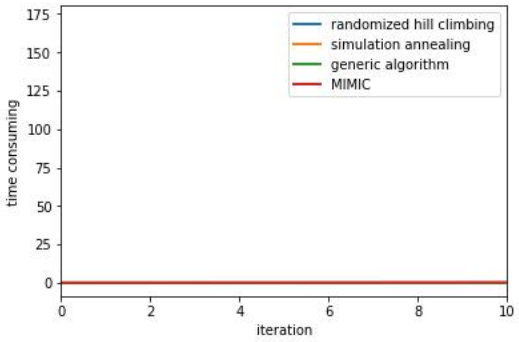

Figure 2.3.4

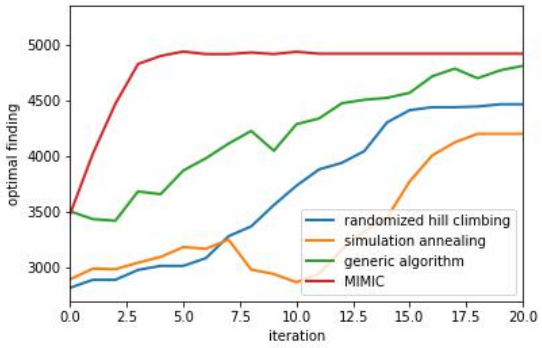

Figure 2.3.5

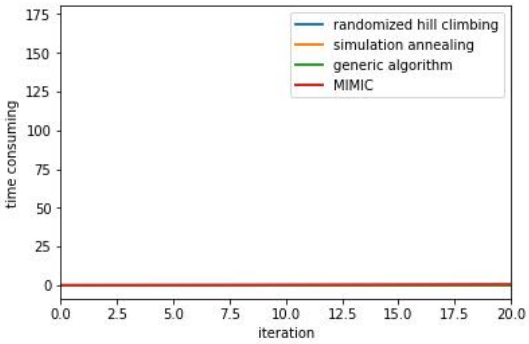

figure 2.3.6

Now it is really distinct that the advantage of MIMIC is enlarged in this kind of question, the speed of converge it takes secures the time cost, the optimization task is rapidly done. And yes, maybe the first point it reaches is the local optimal, but from the figure at very beginning we can also see that it reached the optimal point faster than generic algorithm.

So for this kind of problem, when it is able to have solution rapidly, mimic can be the weapon we use.

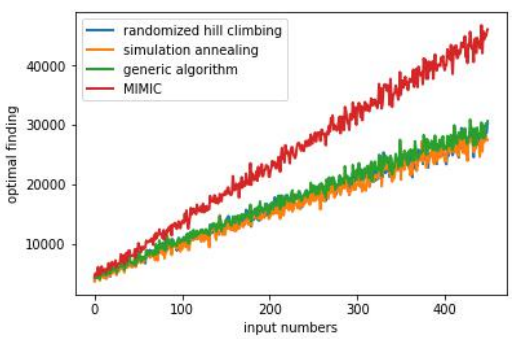

Figure 2.3.7

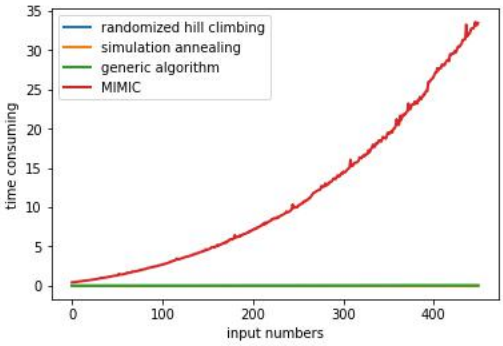

Figure 2.3.8

Let's take another look at the following figure, 2.3.7 and 2.3.8, in these case I placed limitation on iteration, and we can see that not only mimic can be best to converge, but if limited

iteration given, it can be the best solver to find optimal.

We can also see the efficiency of MIMIC by placing 'cutlines' of value and see how long and how much it will take for MIMIC and other algorithms to reach each 'threshold', we can see from this form that the MIMIC is always the first one to reach the desired optima and have low potential to stuck in this situation

Table 2.3

cutline	**MIMIC**			**Generic Algorithm**			**Hill climbing**			**Simulation Annealing**		
	loop	time	val	loop	time	val	loop	time		loop	time	val
4200	3	0.038	4566	9	0.016	4222	15	0.004	4299	33	0.128	4237
4300	3	0.038	4566	12	0.028	4335	16	0.005	4409	NA	NA	NA
4400	3	0.038	4566	13	0.030	4471	20	0.047	4462	NA	NA	NA
4500	3	0.038	4566	14	0.032	4502	NA	NA	NA	NA	NA	NA
4600	4	0.064	4824	17	0.038	4713	NA	NA	NA	NA	NA	NA
4700	4	0.064	4824	17	0.038	4713	NA	NA	NA	NA	NA	NA
4800	4	0.064	4824	21	0.048	4808	NA	NA	NA	NA	NA	NA
4900	7	0.133	4913	37	0.086	4911	NA	NA	NA	NA	NA	NA
5000	48	2.659	5008	45	0.114	5007	NA	NA	NA	NA	NA	NA
5100	74	6.141	5101	330	3.686	5103	NA	NA	NA	NA	NA	NA
5200	184	36.632	5204	NA	NA	NA	NA	NA	NA	NA	NA	NA

As far as I am concerned, the reason why MIMIC get so efficient in this problem, not only a coincidence, but the probability distribution of this problem is somehow that can be inherit, and each step of the knapsack problem can surely affect the following ones. And that is what MIMIC actually good at. As mimic is based on the probability distribution, the dependency of the steps in probability can make it best to solve this kind of problem, and yes, notice that in this situation mimic even not get stuck in the local optima!

3.3.3 Conclusion

To draw a conclusion, MIMIC, in the case of knapsack problems, is the best solution and have the advantage over all other 3. The step complexity of the problem and the probability dependence underlying in each steps contributes to the efficiency of MIMIC.More over, in this case, MIMIC has greatly enlarged its advantage of minimum iteration to rapidly reach the optima.

4 Final comment

In a nutshell, the randomized optimization's task is to find the the best solution to a problem by searching the optima, while the optima can be the highest or the lowest point regarding on the problem itself. To implement algorithm smoothly we can transform finding the low optima task into finding the highest value of the inverse of the optima.

The biggest problem it may face is that randomized optimization may be trapped into local optima, just like the original hill climbing. To get rid of being trapped, we need to consider suitable algorithm ,as well as the parameters of the algorithm, for instance, the cooling rate in SA.

Give a brief comparison and comment to the four algorithm, the randomized hill climbing is simple and low time cost but are not really sufficient in solving mass problems, the SA algorithm is super efficient in finding local optimal in a low time cost but it is vulnerable to parameters, the GA algorithm has the big advantage of getting out of local optimal trap but it is quite expensive to compute, and the local search capability is poor, while also takes longer in convergence. The mimic is really powerful in convergence but it takes significant expensive time cost.

Thus it is essential to figure out the optimization task prior to utilizing the proper algorithm, some testings before mass data get in will be helpful.