CS7641 Assignment#2

Randomized Optimization

Yancheng Liu (yliu723@gatech.edu)

I. Finding weights for a neural network using random search algorithms

1. Analysis of the performance of random search algorithms

Table 1. Summary of neural network training results.

Algorithm	training time to reach 90% accuracy (s)	# of iterations to reach 90% accuracy
BP	9.205	25
RHC	0.512	255
SA	1.681	855
GA	11.516	55

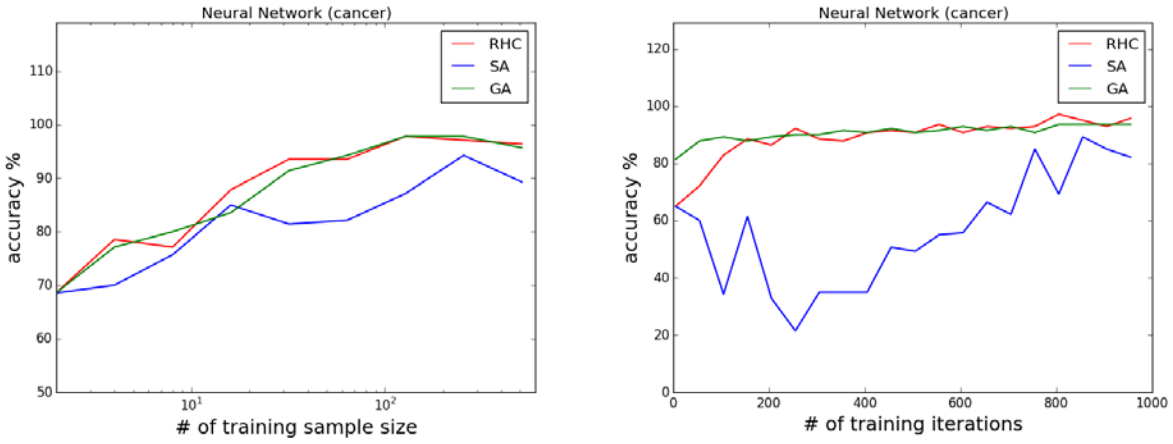

Figure 1. Learning curves of the neural network trained by RHC, SA and GA.

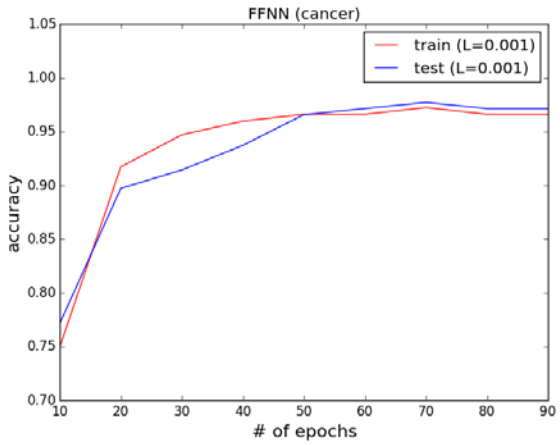

Figure 2. Learning curve of the same neural network trained by Backpropagation (adapted from Assignment#1 for comparison).

As described in Assignment#1, the dataset I chose is the Breast Cancer Wisconsin Dataset. These are real-world data obtained from biopsy analysis of cancer patients from the University of Wisconsin Hospitals in 1990s. There are 699 samples in this dataset and 11 attributes, including cell size, shape, clump thickness etc. The ID numbers are excluded from the fitting. The aim is to predict if a tumor is benign or malignant based on the attributes. The accuracy of prediction is of prominent importance for the following treatment choice and final outcome.

The dataset was split into training and testing sets with 3:1 ratio. Three random search algorithms, namely Randomized hill climbing (RHC), Simulated annealing (SA), and Genetic algorithm (GA), were used to find good weights for a feed forward neural network. The structure of the network is the same (nodes in input-hidden-output layers: 9-12-1) for all search algorithms in order to facilitate comparison. As shown in Figure 1, all three algorithms were able to find good weights in order to reach >90% prediction accuracy given enough training samples and iterations, suggesting they are acceptable alternatives for the commonly used Backpropagation (BP) trainer. In terms of wall clock time, RHC and SA clearly outperformed GA and BP, whereas GA and BP is the winner in terms of using much less iterations to converge (Table 1). This is because GA and BP are more computationally-intensive for each iteration, but they pass more information about the hypothesis space to the next iteration.

Another interesting difference is that although the three search algorithms all successfully found good weights, the learning curves fluctuated a lot (Figure 1). In contrast, the learning curve of BP is much smoother (Figure 2). This is likely due to the stochastic nature of the three search algorithms compared to the gradient descent based BP trainer. In line with this idea, the learning curve of SA fluctuate the most among the three, which is because SA allows more exploration when searching for the global optima.

2. Improve the performance of the algorithms

As shown above, all three algorithms can achieve very good prediction accuracy given enough iteration. The only weakness is that when SA is used, it needs a lot more iterations to reach >90% accuracy, and its performance at early iterations was very poor. As explained above, this is probably due the high temperature at early stage allows more exploration and permits more mistakes. Therefore, in order to improve the performance of SA, I decreased the initial temperature T from 1e11 to 1e2, and tuned up the cooling rate slightly to be more stringent on exploration. As illustrated in Figure 3, this change significantly improved the performance of SA at early iterations.

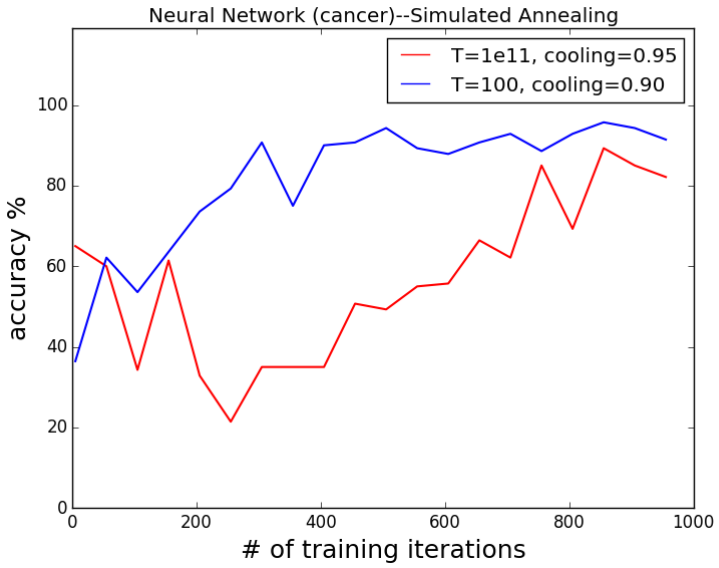

Figure 3. Tweaks to improve the performance of SA.

II. Three optimization problems

1. Introduction for the Problems Chosen for this Analysis.

For this part of the study, I chose 3 well known optimization problems to demonstrate the various strengths and disadvantages of the four algorithms.

(i). Traveling salesman problem (TSP). The task is to find the shortest route between N cities that visits each city exactly once and return to the origin city. This is a NP-hard problem. It has many applications in daily life such as travel planning, logistics, circuit board design, etc. It is also one of the most intensively studied problems in optimization.

(ii). Knapsack problem. In this problem, a person is given a set of items with different value and volume and a knapsack with limited volume. The goal is to fill the knapsack within the volume limit and at same time maximize the total value. This problem has important applications in operational and financial planning where limited resources have to be optimally allocated in order to obtain maximum return.

(iii). One-Max problem. The task is that given a bit-vector x with length N, try to maximize $\sum_{i=1}^{N} x_i$ This is a simple problem equivalent to maximizing the number of 1s in a bitstring, but it illustrates the strengths and pitfalls of the optimization algorithms rather clearly.

2. Results Analysis

(i). Travelling salesman problem (TSP).

Table 2. Results summary for the TSP problem.

Algorithms	Plateau Fitness	# of iterations	# of evaluations	wall clock time
RHC	0.135	60000	60000	69 ms
SA	0.133	60000	60000	62 ms
GA	0.167	120	60000	266 ms
MIMIC	0.123	800	400000	16882 ms

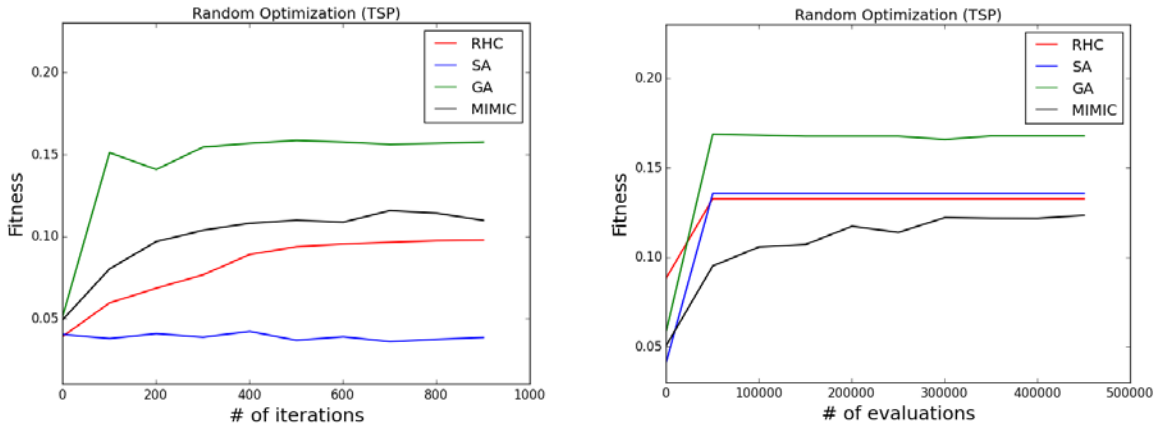

Figure 4. Number of iterations required for optimize the fitness function of the TSP problem.

In this problem, the number of cities needs to be visited is set to 50. 4 optimization algorithms were tested to search for the shortest route. The population size of GA is set to 500, and sample size of MIMIC is also set to 500. The summary results are listed in Table 2. The inverse of the traveling distance was used as the fitness function. As shown, the clear winner is GA. It obtained the best fitness score among the four algorithms with the least number of iterations. Although it is not the fastest in terms of the wall clock time, it is still relatively fast. GA outperforming RHC and SA is expected, as in such a complex problem there could be multiple local optima which RHC and SA are more likely to be attracted. In contrast, the diversity among GA's hypothesis space and its evolution power (through cross-over and mutation) make GA less likely to be stuck in a local optimum.

However, the underperformance of MIMIC in terms of finding the better route is somehow unexpected. One potential problem that may affect the performance of MIMIC is Hamming Cliff. The cities in the TSP problem are encoded by binary code, which is not intuitive. Occasionally, some cities could be really close neighbors but have huge Hamming distance, causing the correlation to be mislabeled. One possible solution for this problem is to encode all

the cities by Gray Code, in which the Hamming distances of all the neighboring numbers are always 1.

There is another interesting observation when compare the performance of SA and RHC at the early stage and late stage of the optimization process (compare Figure 4 left panel with right panel). At early stage, SA performed much worse than RHC, but it caught up as iteration number (which equals to the number of evaluation in the cases of SA and RHC) increased. In the end, SA slightly outperformed RHC. The explanation is that SA allows more exploration, especially at early stage when temperature is high, thus lower fitness is tolerated. But this gives SA a better chance to find the global optimum or better local optima at later stage.

(ii). Knapsack problem

Table 3. Results summary for the Knapsack problem.

Algorithms	Plateau Fitness	# of iterations	# of evaluations	wall clock time
RHC	496	50	50	0 ms
SA	813	120	120	1.1 ms
GA	1149	200	40000	47.0 ms
MIMIC	1162	20	4000	187 ms

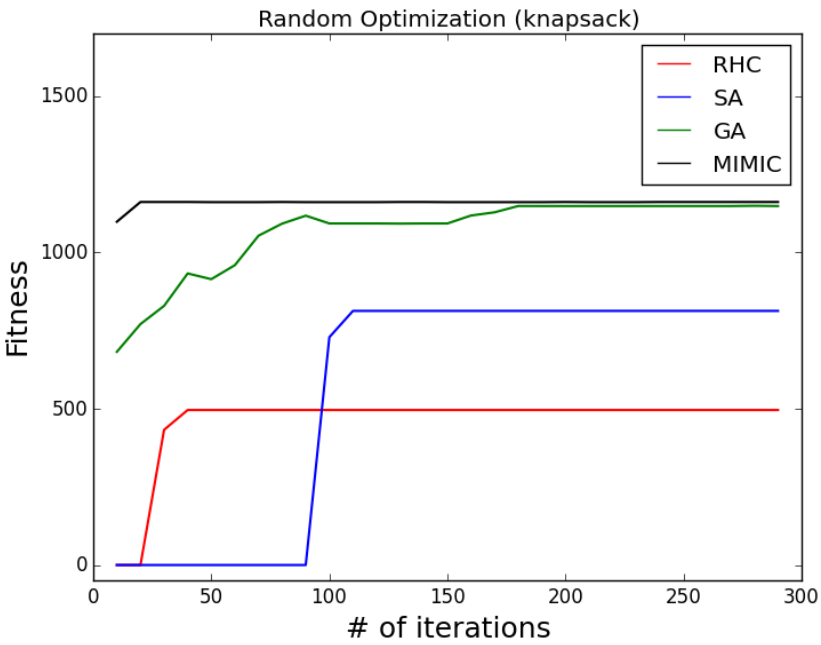

Figure 5. Number of iterations required for optimize the fitness function of the Knapsack problem.

Knapsack is a disparity type of problem. It favors GA and MIMIC, whereas it will be hard for SA and RHC which are more suited for smooth functions. Therefore it makes sense to see that GA and MIMIC are the clear winners in terms of finding the maximum (Table 3). MIMIC required more time than other algorithms, because at each iteration it needs to compute the minimum spanning tree and recalculate distributions to generate samples. But in terms of number of evaluations, MIMIC is 10-fold better than GA. This suggests that in problems where the cost of each evaluation is high, MIMIC is the preferred technique than GA.

As seen in Figure 5, both RHC and SA were stuck at some local optima. Consistent with the observations in the TSP, SA underperformed at early stages because it allows more exploration, but this gave SA the benefit to find a better local optimum at later stage than RHC (See Figure 5, after 150 iterations).

(iii). One-Max problem

Table 4. Results summary for the One-Max problem.

Algorithms	Plateau Fitness	# of iterations	# of evaluations	wall clock time
RHC	200	250	250	16 ms
SA	200	300	300	1.0 ms
GA	134	500	25000	31 ms
MIMIC	200	150	7500	2200 ms

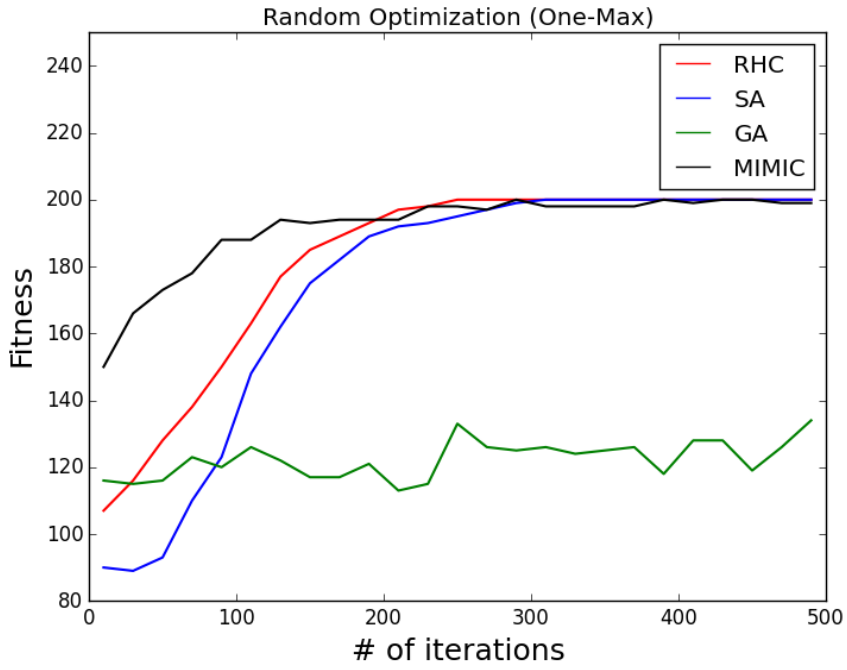

Figure 6. Number of iterations required for optimize the fitness function of the One-Max problem.

In this problem, the length N of the bit-vector was set to 200, so the maximum should be 200. As shown in Table 4 and Figure 6, MIMIC performed best in terms of reaching global optima with fewest iterations. RHC and SA also performed well and found the global optima with least wall clock time spent. Surprisingly, GA performed worst with this straightforward

task. It stuck at ~130 even after 500 iterations and 25000 evaluations. One possible explanation is the problem of crowding. Crowding represents the situation wherein some individual is much more fit than others, therefore it reproduces much faster, and copies of this individual and its very similar relatives dominate the whole population. The consequence of this phenomenon is that the diversity of the population will significantly decrease, which in turn will slow down the evolution process towards global optima and may stuck in a local optima.

To address this issue, I tried to two approaches in order to improve the performance of GA: (a) increasing the population size (Figure 7, left panel) or (b) increasing the number of cross-over per iteration (Figure 7, middle panel). Both approaches aim to increase the diversity of the population to avoid crowding: greater population size preserves more diversity at each iteration; higher cross-over rate facilitates the generation of more diverse individuals. And when the values of parameters are all independent of each other like in this problem, more corss-over usually is beneficial. As expected, the performance of GA increased as population size or cross-over rate increased. In contrast, the performance of GA didn't keep increasing as mutation rate increased (Figure 7, right panel). There seemed to be a sweet-spot value for the mutation rate. This indicates that the GA algorithm needs a certain mutation rate to keep the diversity, but passing over that rate will not further benefit the final selection as the mutation changes are random and not necessarily beneficial. This probably reflects the trade-off between keeping the elites and increasing the diversity.

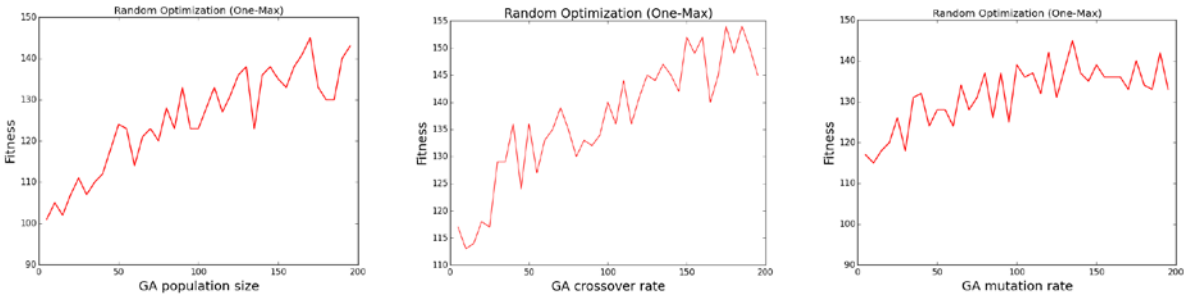

Figure 7. Tweaks to improve the performance of GA.