CS7641 Assignment 2 – Randomized Optimization

zfu66@gatech.edu

Abstract

In this project three local random searching algorithm were implemented into the neural network weighting factor optimization process. Further these three algorithms plus MIMIC were applied to search the solution spaces of three classical optimization problems and their performances were evaluated in terms of fitness function as well as the running time.

I. Introduction

This report outline starts with brief discussion on four local random searching algorithms: random randomized hill climbing (RHC), simulated annealing (SA), genetic algorithm (GA) and MIMIC[1]. A binary classification experiment was conducted to measure the performances of RHC, SA, GA and compare them with the benchmark algorithm backpropagation. After that all four algorithms were employed to solve three classical optimization problems: continuous peaks, Flip-Flop and travelling salesman problems. The results of these experiments were extensively discussed.

1. Randomized hill climbing (RHC)

The intuition of randomized hill climbing algorithm is to consider the movement of the input data state on the surface of solution landscape. We could evaluate the initial state of the input data first, if it is the goal state (local minimum or maximum), the searching step stops and returns success. Otherwise the initial state will be set as the current state, and the algorithm will select a direction from a random movement set that will increase the value of fitness function. The new state will then be selected as the current state and start a new search as mentioned above. Such process will be carried out repeatedly until the stop criteria is reached. And how to define the stop criteria is another important question in RHC algorithm, often it is the maximum number of iterations, or the value of fitness function was converged. The advantage of this algorithm is that it doesn't require any parameter tuning-up so its running time is short, yet the definition of the fitness function is critical and often the algorithm stops at local minimum instead of global minimum.

2. Simulated Annealing (SA)

Simulated annealing algorithm is quite similar with randomized hill climbing method. On each solution landscape searching step, if the value of fitness function of next state is higher than that of current state, the movement will be accepted. Otherwise the movement will be accepted with a probability:

exp{ -[fitness_value(current) - fitness_value(next)] / T}

based on Boltzmann distribution. Here T is the temperature parameter that will be lowered repeatedly by a constant factor alpha. Thus two parameters, the initial temperature T0 and temperature changing factor alpha need to be tuned up for this algorithm.

3. Genetic algorithm (GA)

Genetic algorithm belongs to evolution algorithm family that mimics the process of natural selection. It starts with a group of solutions called ‘Population’. The fitness function here evaluates the ability of one solution to compete with other solutions within a population and produce a fitness score. Solutions with higher fitness scores have more chances to produce offspring and inherit their parameters. The two parents of a offspring may switch their parameters at a random crossover point and the new offspring will be added into the population as well. Meanwhile an offspring could undergo a mutation that modifies one or more parameter values of a solution. The algorithm will stop if the population does not give rise to new generations that significantly differ from the old generations. Thus three parameters, the size of initial population, the crossover rate and the mutation rate need to be tuned up for this algorithm.

4. MIMIC

MIMIC also belongs to evolution algorithm family and could be recognized as an advanced version of genetic algorithm. It starts with a random population of solutions that were uniformly chosen from the input space. Then the median fitness of this population is selected and denoted as theta_0. The next step is to update the probability density function P_theta_i(x) from a sample, and generate more samples based on the updated P_theta_i(x). After that the theta_i+1 will be set as the Nth. percentile of the data and only data less than theta_i+1 will be kept. Thus two parameters, the size of initial population and N (determining how many samples should be kept) need to be tuned up for this algorithm.

II. Methods

1. Data pre-processing

For neural network weighting factor optimization experiment, I still use the wine quality dataset of Assignment 1 that was directly downloaded from Kaggle website. Here I used quality score = 6.5 as a cutoff to assign “Good” and “Bad” labels to the independent variables. Then the categorical labels were transformed to numerical variables via sklearn.preprocessing.LabelEncoder function (‘Good’ = 1 and ‘Bad’ = 0). Note that using quality score = 6.5 (>6.5 is ‘Good’ otherwise ‘Bad’) as the threshold will generate 217 ‘Good’ samples and 1382 ‘Bad’ samples that makes the dataset imbalanced. All independent variables were standardized by removing the mean and scaling to unit variance (Z ~ N(0, 1)). The final dataset was split to 64% training set, 16% testing set and 20% for validation set.

2. Neural network topology

The neural network topology was based on the Assignment 1 result. It has one input layer with 11 nodes (corresponding to 11 features), 3 hidden layers that have 50 nodes on each layer, and an output layer. The learning rate was set to 0.001 with maximum = 50 and minimum = 0.00001. The activation function of hidden layer node is relu function.

3. Randomized hill climbing (RHC)

Randomized hill climbing (RHC) was implemented by ABAGAIL[2] library, no hyperparameters need to be tuned up.

4. Simulated Annealing (SA)

Simulated annealing algorithm was implemented by ABAGAIL library, the hyperparameters for neural network weighting optimization experiment are T(initial) = 1E10, annealing factor = [0.15, 0.25,0,45,0.75,0.95], and T(initial) = 1E10, annealing factor = [0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95] for classical optimization problems.

5. Genetic algorithm (GA)

Genetic algorithm (GA) was implemented by ABAGAIL library, the hyperparameters for neural network weighting optimization experiment are population = [40, 20], mate =

[20,10], mutate = [20,10], and population = [500, 200, 100], mate = [100, 50, 25], mutate = [100, 50, 25] for classical optimization problems.

6. MIMIC

MIMIC was implemented by ABAGAIL library, the hyperparameters for classical optimization problems are population = [200,150], keep samples = [100,75], m = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9].

7. Classical optimization problems

All three classical optimization problems, Flip-Flop, Traveling Salesman Problem and Continuous Peak problem were implemented by ABAGAIL library.

III. Neural network weighting optimization

As mentioned in Assignment 1, the wine quality dataset is a small and unbalanced data set (217 ‘Good’ samples and 1382 ‘Bad’ samples). That means the popular metrics of evaluating binary classifier performance, such as accuracy, precision, recall and F-1 score are not proper for this dataset. For instance, the validation set used in this experiment has 43 ‘Good’ samples and 276 ‘Bad” samples. If we just assign all validation samples as “Good”, we could still have the accuracy score = 276 / (276+43) = 0.865. Thus I manually compute the false positive rate and true positive rate at 100 thresholds between 0 and 1, and calculate the AUC score of the ROC curve as the performance measurement of these four neural network weighting optimization algorithms.

From Figure 1A we could see the benchmark backpropagation algorithm quickly reaches the highest AUC score at 100 iteration then its AUC score continuously drops with the increasing of iteration counts. Hill climbing family algorithms (RHC and SA) reaches its AUC score plateau after 2000 iterations. GA algorithm reaches it maximum AUC score at 3100 iteration then its AUC score goes downhill with the increasing of iteration counts. One explanation of this phenomena is that the backpropagation algorithm will lower the true positive rate while increasing the iteration counts, which indicates it may not be the best choice of this classification problem, although it is fast and do not need to tune up the parameters. The AUC scores of RHC and SA will be fluctuant within a small region once these two algorithms reached the optima. The GA algorithm’s performance indicates producing more offspring and finding better optima may not be positively correlated with the classifier performance in terms of AUC score. Since SA has the best performance in this experiment, I also investigated the impact of different temperature annealing factors

on the binary classifier performance and the results were shown in Figure 1B and SA algorithm with annealing factor = 0.75 has the best performance (Table 1).

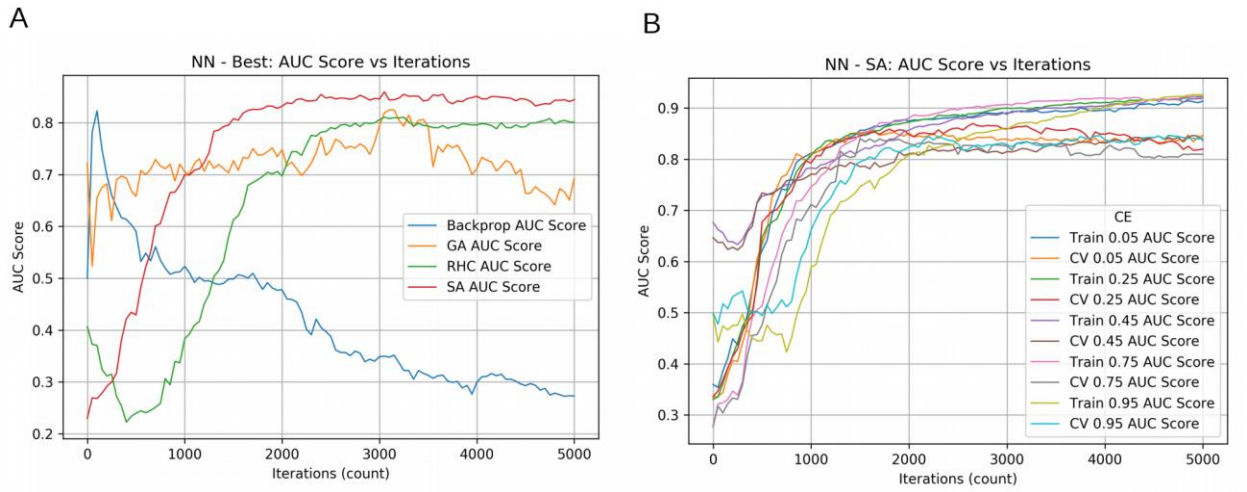

Figure 1. The neural network weighting optimization results.

IV. Solutions for Three Classical Optimization Problems

1. Flip-Flop

Flip-Flop is a function that given a bitstring, it will return the number of times bits alternate. If a bitstring has constantly alternating bits and its flip-flop function values equals to its length, this string will be recognized as an optimized solution of the Flip-Flop function. Actually the Flip-Flop was well-known by its large number of local optima. That's why we want to try the four random optimization algorithms to solve it. In this experiment the bitstring length was defined as 100.

Theoretically this problem favors simulated annealing algorithm to solve it. Since according to the description of simulated annealing algorithm on the "Introduction" part, SA could overcome the local optima trap if the neighbor points are near and the solution space is a small discrete space. The experiment results support this point. From Figure 2A we could see that SA has the highest average fitness score at iteration 5000. Both MIMIC and GA (the evolution algorithm family) have fitness score more than 90 after 1000 iterations and begin to converge. However if we check the times of calling fitness functions (Function Evals, Figure 2B), such values of MIMIC and GA are much higher than that of SA at each iteration point. Thus among the tested four algorithms, the SA is the best one to solve the Flip-Flop problem. And not surprisingly, the RHC has the worst performance

in terms of fitness score. That because it is easy to get stuck at the local optima in which alternating any bits of the bitstring would not generate information gain (Table 1).

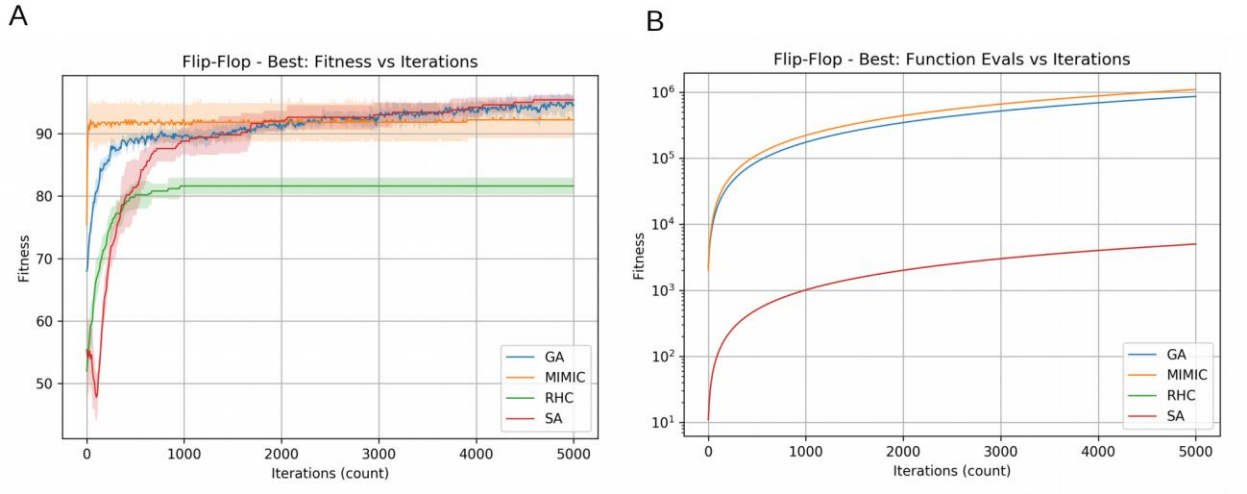

Figure 2. Solving the Flip-Flop optimization problem.

2. Traveling Salesman Problem (TSP)

The goal of traveling salesman problem is to find the shortest possible round trip that visits every city (data points) exactly once. In this experiment the number of cities was defined as 50. Note that since the object of this problem is to identify the shortest path d, we need to use 1/d as the fitness score for maximization process.

Theoretically this problem favors genetic algorithm to solve it. Since unlike other discrete-state optimization problems, we already know the elements of the optimal state vector (a series of integers from 0 to N-1, where N is the number of cities), yet we don't know the order of these integers. By tuning up the mutation rate and crossover rate, the GA algorithm could generate a series of integers which order is quite close to the global optima. The experiment results support this point. From Figure 3A we could see that GA has much higher fitness score than the other three algorithms. As expected, the performance of hill climbing algorithms are worse than GA, since both RHC and SA could only handle one possible solution vector at a time, and the searching space of this problem is the factorial of 50 (a tremendous number). Whereas GA explores the entire sets of potential solutions and has more chance to find the true optima in an area (MIMIC could also estimate the probability of true optima appearing in an area). Yet the trade-off is GA needs marginally more times to call the fitness function comparing with RHC and SA (Figure 3B, Table 1).

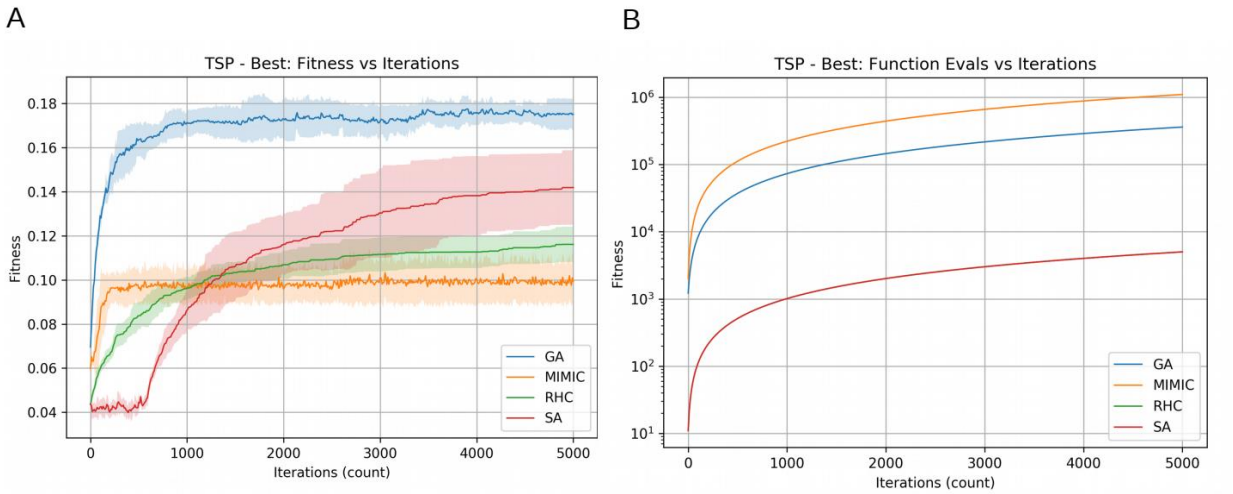

Figure 3. Solving the TSP optimization problem.

3. Continuous Peaks Problem

Continuous peak problem is quite similar with four peaks problem. It takes a bitstring of length N with a “trigger” position T as the input of the function. The optimal solution is a bitstring with contiguous 0s or 1s before the trigger position, and the complementary bits contiguously after trigger position towards the end. The optimal state should have absolute maximum 2N - T - 1. In this experiment the N is equal to 100 and T is equal to 60.

Theoretically this problem favors MIMIC to solve it. As described in the “Introduction” part, the MIMIC algorithm could generate a probability density function to estimate the most likely locations of the optima. By repeating this process and extract samples from the previously built probability density function, this algorithm continuously rises the probability to reach a better optimum. Once a better optimum is reached, the previous suboptimal local optima could be removed to narrow the search area. Such iteration will be repeated until the algorithm finds the absolute maximum 2N - T - 1. The experiment results partially support this point. From Figure 4A we could see that the MIMIC quickly reaches fitness score = 80 after few hundred iterations. Whereas GA’s fitness scores close to 100 after 3000 iterations, and hill climbing family algorithms (RHC and SA) fitness scores approaching 100 after 4000 iterations. One explanation for this scenario is I only uses 5 trials for each iteration points, and I did not check the percentage of optima (2N - T - 1 = 2*100 - 60 - 1 = 139) reached among these five trials at each iteration. It is likely RHC, SA and GA were all converged to the suboptimal, which has 100 0s or 1s and its fitness score is equal to 100. More experiments with larger trials numbers at each iteration point will help us to further investigate the performance differences of these four

algorithms. Note that the MIMIC requires much more callings of the fitness function than other algorithms (Figure 4B), because it needs to find the best K samples out of a population with N instances and establish the probability density function (Table 1).

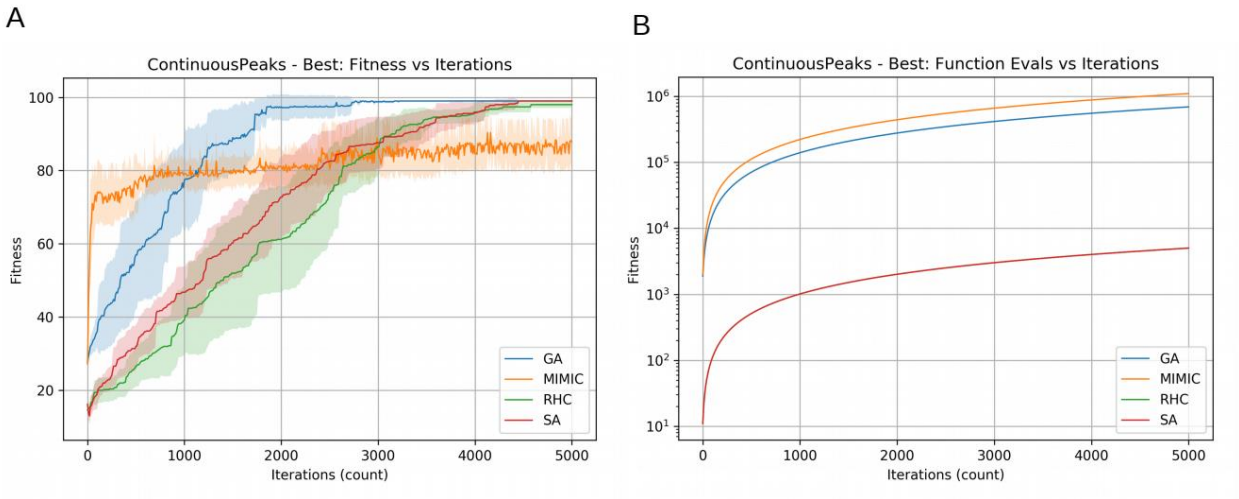

Figure 4. Solving the continuous peaks optimization problem.

Problem	Algorithm	Best Parameters	Best fitness	Best iterations	Best time	Best fevals
NN weighting optimization	BP	N/A	0.82	100	3.5	0
	GA	pop=20, mate=20, mutate=10	0.82	3100	682.3	0
	RHC	N/A	0.81	3050	32.8	0
	SA	annealing factor=0.75	0.86	3050	32.8	0
Continuous Peaks	MIMIC	pop=200, sample=75, m=0.9	92	2790	15.0	615800
	GA	pop=200, mate=50, mutate=50	99	2570	0.1	215236
	RHC	N/A	99	3990	0.0	4001
	SA	annealing factor=0.65	99	4110	0.0	4121
FLIPFLOP	MIMIC	pop=200, sample=75, m=0.8	92	40	0.3	10800
	GA	pop=500, mate=100, mutate=100	94	4210	0.7	728854
	RHC	N/A	82	450	0.0	461
	SA	annealing factor=0.85	97	3920	0.0	3931
TSP	MIMIC	pop=200, sample=75, m=0.7	0.12	3670	30.7	809400
	GA	pop=500, mate=50, mutate=25	0.19	4800	0.7	346745
	RHC	N/A	0.12	4660	0.0	4671
	SA	annealing factor=0.95	0.13	4880	0.0	4891

Table 1. Experiment results summary.

V. References

1. De Bonet, J., C. Isbell, and P. Viola (1997). MIMIC: Finding Optima by Estimating Probability Densities. In Advances in Neural Information Processing Systems (NIPS) 9, pp. 424–430.
2. https://github.com/pushkar/ABAGAIL