Assignment 2

# 1. Requirements

Python 3.6.5, Numpy 1.14.3, mlrose

# 2. Problem Description

## N-Queens Problem

For a N x N chessboard, find the optimal solution of placing N queens on the board so that queens cannot attack each other (<https://en.wikipedia.org/wiki/Eight_queens_puzzle>). Note that the wiki page is specific for the 8-queens problem but the size of chessboard can be any arbitrary integer >= 3.

## 4-Peaks Problem

Given a bit string, find the best combination of 0s and 1s so that the fitness function below will be maximized:

Where:

*Tail(0, bit string)* is the number of trailing 0s in bit string

*Head(1, bit string)* is the number of leading 1s in bit string

*R(bit string, T)* = length of bit string, if *Tail(0, bit string)* > T, and *Head(1, bit string)* > T; otherwise= 0. Here T is defined as 0.2 x length of bit string.

## Knapsack Problem

For a given bag with certain weight capacity W, and n items with weights [w1, w2, ...., wn] and values [v1, v2, …, vn], optimize the items in the bag so that the total weight of items is less than or equal to the weight capacity while obtaining the highest total value.

# 4. Optimization Problems

Even though each of the 4 algorithms evaluated in this assignment have fundamental differences, they share one thing in common: at each iteration step, they all attempt to search its neighbors for a better fitness score. The number of max attempt, as shown in this section, is a key factor determining the convergence speed. Factors particular to each algorithms will be analyzed as well.

## 4.1 N-Queens

I only test the 8-Queen problem in this section. The fitness function is designed to check every pair of queens, and fitness score increases by 1 if that pair does not attack. Therefore, the highest fitness score is 28 in this case. For all runs, the initial positions of queens are the same to facilitate comparison.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Simulated Annealing |  | Random Hill Climbing |
|  |  |  |
| Genetic Algorithm |  | MIMIC |

Figure 4.1 Fitness curve for each algorithm on 8 Queens, with different combination of parameters

There is no constraint on the number of iterations for each algorithm (that is, the algorithm can run forever if better fitness score can be found). Red dots in the figures above show where each run stops. Each curve on the plots shows the fitness score evolution per each iteration. For plots with seemingly one curve, multiple curves in fact exist but overlap. Legends on each plot show the best fitness score achieved and parameter combinations for each run.

All runs for Simulated Annealing (SA) managed to converge to fitness score 27, suggesting there is one pair of queens can attack each other. In this case, different max attempts (even with 100) to search neighbors for better fitness score does not lead to Global Optimal Solution (GOS). Further study, which is not shown here, suggests that once attempt increases to 1000, after more than 2000 iterations, SA is able to find GOS. This makes SA a very inefficient algorithm compared to Random Hill Climbing (RHC) and Genetic Algorithm (GA), both of which are able to reach GOS with less than 200 iterations. The drops in fitness score at the beginning of iteration suggests data points with worse fitness score are chosen due to probability. However, as temperature decreases, this become unlikely and fitness score eventually become monotonically increasing.

Results obtained by RHC indicates the strong effect of max attempts: when max attempt is 10, the fitness score is only 25; while fitness scores all reach GOS with 50 and 100 max attempts. Similar to SA, RHC experiences fitness score drop at certain iteration step. This is because the algorithm restarts per settings. For RHC, the hope is that by restarting with different initial seeds, it increases the probability of data point lay in the vicinity of GOS, so that RHC can eventually converge to GOS.

Two more parameters, other than number of max attempts to search for neighbors, are available for Genetic Algorithm (GA): population size and mutation probability. Theoretically, bigger population size increases the chance of getting better parents, and therefore better children. Mutation probability, on the other hand, could be a double edged sword: mutation could happen in either directions (good or bad), and might not necessarily improve the overall fitness score. As we can see from this figure, mutation actually reduces the final fitness score when max attempt = 100, and pop size=10. It is also worth noting that none of the runs reach GOS, and similar to SA, increasing max attempt finally allows GA to reach GOS, with a total number of iterations of about 400 (not shown in figure).

Fitness curves for MIMIC differ from other algorithms in that the fitness scores almost do not change with iterations. The reason is that MIMIC uniformly sample the entire solution space initially, and for every iteration, it only resamples from those with better fitness scores. In other words, from step 1, the best solution should already be in the sample space, and each iteration reinforces sample space towards the best solution. The beauty of MIMIC is that it does not require many iteration steps to reach a local optimal solution, and could be quite efficient for complex problems.

## 4.2 4-Peaks

10-bit string is used for this problem. According to the fitness function defined in Section 2, GOS should have this form [1,1,1,1,1,1,1,0,0,0], with a fitness score of 17.

|  |  |
| --- | --- |
|  |  |
| SA | RHC |
|  |  |
| GA | MIMIC |

Figure 4.2 Fitness curve for each algorithm on 4 Peaks, with different combination of parameters

For SA, small max attempt is enough to find GOS, with only about 40 iterations. This is much more efficient that RHC, where only max attempt = 50 and 100 are able to find GOS. It should be noted that the fitness curve for RHC above is based on random restart, meaning once a local optimal solution is achieved, RHC will start from a random seed and find the local optimal solution again. The big dive on the above plot shows the random restart. In this example, RHC will not converge to GOS without random restart.

The fitness curves for GA confirms the argument before, that mutation could be a double edged sword. When max attempt = 10 and pop size = 10, larger probability of mutation in fact deteriorates the final fitness score. With larger pop size or max attempt, however, GA is able to find GOS. In this case, max attempt and pop size all serve similar purpose: enlarge the scope of sample space, and therefore improve the likelihood of reaching GOS.

Fitness curve behavior for MIMIC agree with Figure 4.1, that is, fitness scores almost do not change with iterations. In this example, both MIMIC and GA are very efficient: with proper parameter settings, both converge to GOS within 20 iterations, which differs from last problem where GA tends to have much more iterations than MIMIC.

## 4.3 Knapsack

I use 10 items to demonstrate the performance of each algorithm. For these items, their individual weights are [10, 5, 2, 8, 15, 7, 8, 12, 6, 12], and individual values are [1, 2, 3, 4, 5, 4, 6, 2, 4, 7]. For this problem, greedy algorithm can be used to get GOS, which is [0, 0, 1, 1, 0, 1, 1, 0, 1, 1], where 0 means item is not chosen and 1 means item is chosen. With this GOS, the fitness score, which is the total value in knapsack, is 28.

|  |  |
| --- | --- |
|  |  |
| SA | RHC |
|  |  |
| GA | MIMIC |

Figure 4.3 Fitness curve for each algorithm on knapsack, with different combination of parameters

One important set of parameters of SA that have not yet been discussed is the initial temperature and temperature step (delta T). For the previous two examples, their effect is not significant, and is therefore omitted on purpose. For the Knapsack problem, however, final fitness score is very sensitive to the selection of them. To achieve the result in Figure 4.3, initial temperature is 5 and delta T is 0.001. More SA results can be found below.

|  |  |
| --- | --- |
| Figure 4.4 Fitness curve for different temperature combinations for SA | in Figure 4.4 max attempt is set to be 50. With the same delta T, higher initial temperature tends to lead to better fitness score. This is because higher initial temperature increases the chance of using worse candidates as the new seed, and hence increases the chance of getting out of current local optimal and continues search for GOS. Because SA reduces temperature by delta T in every iteration step, having a smaller delta T allows a longer period of picking worse candidates for more iteration steps, and therefore increases the chance of reaching GOS. |

The RHC results in Figure 4.3 are achieved by 5 random restarts. With only 1 random restart, fitness score is only 23 or 24, depending on max attempt selection. As discussed before, theoretically given infinite random restarts GOS can always be reached, but in practice we attempt to minimize the random restarts while achieving GOS. For all the problems tested so far RHC seems to be very efficient.

For this Knapsack problem, GA has a descent performance: out of 8 cases, GA manages to converge to GOS in 6. The only 2 cases that GA fails to achieve GOS are the ones with small max attempts and small population size, which indicates good parents are unlikely to be added to the pool, and therefore algorithm is prone to converge to a local optimal solution.

Similar to other two cases, MIMIC in this case also converge very quickly. In fact, for all cases MIMIC can converge within 20 iterations consistently, which outperforms the other three algorithms in terms of number of iterations.

## 4.4 Some Other Thoughts

All previous tests are done based on problems with fixed settings, and therefore, fixed problem complexity. It is also important to compare algorithm performances as problem complexity increases. In this section, N-Queens problems is used with a set of different Ns. Parameter settings for each algorithm are based on the best performers in Section 4.1. Algorithm parameter settings are listed below:

***SA:*** *max attempt = 100*

***RHC:*** *max attempt = 100, restart=5*

***GA:*** *max attempt = 100, pop size = 100, mutation prob = 0.1*

***MIMIC:*** *max attempt = 100, pop size = 200, keep pct = 0.1*

|  |  |
| --- | --- |
|  |  |

Figure 4.5 fitness scores for 4 algorithms

Figure 4.5 (left) shows N from 5 to 12, and Figure 4.5 (right) shows N from 13 to 20. The fitness scores among different algorithms are very close with less complex problem (smaller N), showing that all algorithms are able to converge to GOS. The fitness scores start to diverge as N increases, with SA and RHC almost always outperform GA and MIMIC. This in part is because parameters for GA and MIMIC are not optimized for larger Ns. For instance, GA relies on the population size to generate children. However in this test, population size does not change with N. When a certain population size is enough to cover the solution space of a simple problem, it might be insufficient for a complex one. Further test with larger population size proves that fitness score can be improved with larger population size.

# 5. Neural Network

In neural network, a technique used to update connection weights among different hidden layers of neurons are usually back propagation, where fitness score information is populated back to hidden layers for weight update. One popular approach is using gradient descent, as long as the activation function among layers is differentiable. In this section, instead of using gradient descent and back propagation, SA, RHC and GA are used to update connection weights.

The same EEG data used in HW1 is reused here, which contains ~14,000 data points and 14 features. The target is a binary classification. For each algorithm, back propagation is used as benchmark algorithm for comparison.

For all runs, a 80/20 split is used for training and testing. The accuracy scores reported below are all based on the testing set. ‘early stopping’ is implements if no change happens for 20 consecutive iterations. 3 hidden layers are used to build the neuron network, with number of neurons being 1024, 128 and 16, respectively. For all runs, ‘relu’ is used as the activation function.

## 5.1 RHC

One set of very important parameters in RHC is the number of restarts. As discussed in the last 3 problems, more restarts increases the probability of random seeds landing in the vicinity of GOS, and increases the chance of reaching GOS. For this problem, I tested 4 different numbers of restarts: 5, 10, 40, 80.

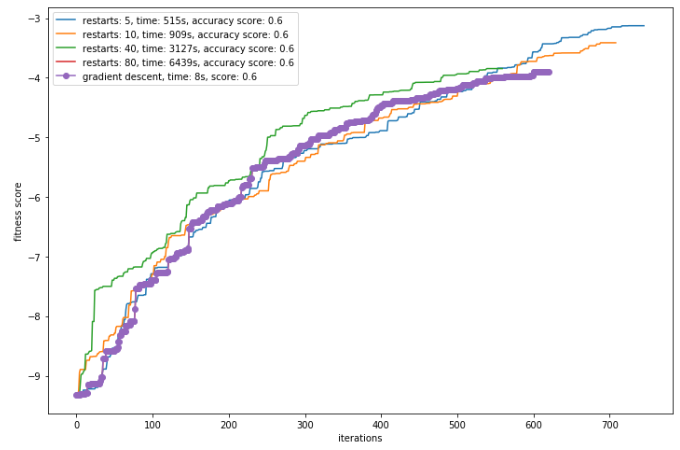


Figure 5.1 neural network fitness curves based on RHC algorithm

Runtime required for simulation differ significantly, and it is almost linearly correlated with the number of restarts. RHC appears to be an inefficient algorithm for neural network weights update, with the fastest run taking about 9 mins and slowest run taking about 1 hour and 40 mins.

For different restarts, accuracy scores do not show any difference. It indicates that no matter with 5 or 80 restarts, new seeds are not able to jump out of the local optimal ‘sink’ and therefore ended up with the same accuracy score. It should be noted that fitness scores are calculated by using probability instead of predicted label, therefore a slight difference exist in the final fitness scores while the accuracy scores are the same.

It is proven that increasing number of restarts will help improve algorithm performance. However, given the amount of time required for this algorithm, I did not increase the number of restarts further.

Compared with back propagation, the current settings of RHC does not offer any benefits. In fact, RHC performs much worse than back propagation in terms of efficiency. Back propagation finishes training in 8 seconds, orders of magnitudes faster than any runs of RHC. Back propagation is also able to achieve similar fitness score with RHC, and the same accuracy score with RHC.

## 5.2 SA

Whether the algorithm will converge, or how fast it will converge, largely depends on initial temperature, ending temperature, and temperature step (delta T). In this example, with a fixed ending temperature of 0.001, I test several combinations of initial temperature and temperature step.

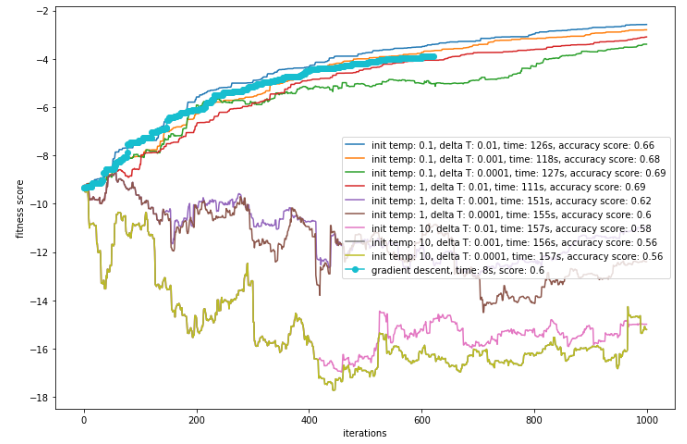


Figure 5.2 neural network fitness curves based on SA

For a total of 9 runs, only 4 have fitness score improvement with iterations. The best accuracy score is obtained with the lowest initial temp and delta temp (initial temp 0.1 and delta T 0.0001). When it comes to fitness score, the one with initial temp 0.1 delta temp 0.01 has the highest one. As discussed before, accuracy score and fitness score are two different metrics, even though they are correlated.

5 fitness curves deteriorate with iteration. Those are all with higher initial temperatures (1 and 10). This leads to an important observation in SA: when initial temperature is too high, it is more probable for the weights to jump to a worse state. As cool down continues, those weights might be stuck in a bad state (in this case, worse than initial state) and be never able to get out. Therefore for SA, picking the right initial temperatures is crucial for the performance of this algorithm: it should not be too high, so that data points are less likely to be stuck at a worse state; and it should not be too low, otherwise the data points cannot jump out of the initial state.

There are certain benefits of running SA, compared to back propagation: 1) SA manages to achieve higher accuracy score (0.69) than back propagation (0.6); 2) SA also achieves better fitness scores than back propagation. However, as shown in Figure 5.2, not every run of SA is able to outperform back propagation. SA can have inferior performance if hyper parameters are not carefully tuned.

Compared to RHC, SA is very efficient in updating connection weights in neural network, and the efficiency is not quite affected by parameter settings (for different parameter settings in Figure 5.2, training time ranges from 111s to 157s). And yet, the accuracy score and fitness score for the best performer of each algorithm show that SA outperform RHC in every aspect. However, SA does not guarantee fitness score is always improving as training continues. Parameter tuning for SA is a subtle task and requires domain knowledge and experience.

## 5.3 GA

The performance of GA largely depends on the size of population pool and mutation probability, both of which affect GA’s capability of getting out of local optimal solution and converging to GOS. 4 different combinations of pop size and mutation probability are tested in this section.

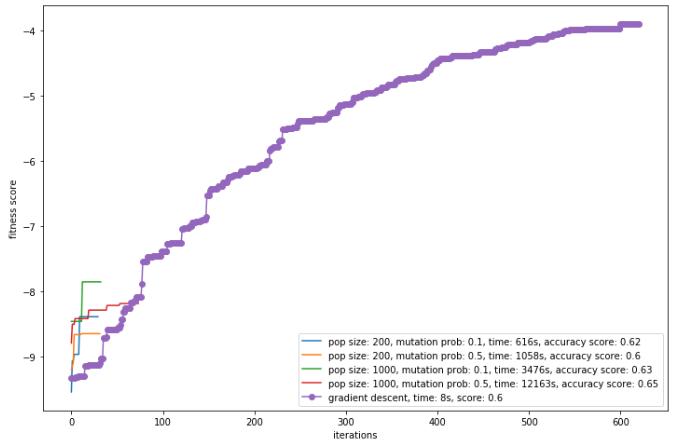


Figure 5.3 neural network fitness curves based on SA

Figure 5.3 suggests that with higher population size and mutation probability, model tends to achieve a better accuracy score, but with a price: significant reduction in training efficiency. The longest training (more than 3 hours) happens when pop size = 1000, and mutation prob = 0.5. With current neural network structure, there are ~150000 weights to be optimized, and a population size of 1000 can only cover a very small portion of all the possible weight combinations. Therefore, even though in theory increasing population size should improve model accuracy, in practice it is too computational expensive to afford.

Runtime for GA depends on parameter settings, and is linearly correlated with population size. Mutation probability affects runtime as well, but not as significant as population size. All GA runs stop in less than 200 iterations, because of the ‘early stopping’ criterion implemented in the simulation. However, runtime is comparable with RHC, with a couple hundred iterations. This shows that each iteration for GA takes much longer than the other algorithms, which further strengthens the argument that GA might not be a good algorithm for complex problem, if efficiency is a concern.

# Conclusion

In this report, 4 different algorithms, RHC, SA, GA and MIMIC, are implemented and compared using simple optimization problems, and RHC, SA, and GA are used to update neuron network connection weights, and compared with back propagation. Each algorithm has its pros and cons, and should be selected case by case.

The performance of RHC relies heavily on the number of restarts. In theory, if the random seeds can cover the entire solution spectrum GOS can always be found. But that might not be practical given limited resources. Therefore RHC could be a very good algorithm for simple problems but its application is constrained for more complex problems.

SA is a dedicated algorithm. The key parameters (initial temperature and temperature step) determines the algorithm performance. In this report, it shows that initial temperature and temperature step should neither be too big nor too small. A good combination of these two parameters give random seed the capability of getting out of local optimal solution, and once it is in the vicinity of GOS, temperature cools down quickly enough to ‘trap’ the random seed. A bad choice of combination might lead to results in Figure 5.2, where a random seed might be trapped in even worse state than the initial state. SA, compared to other algorithms, is quite efficient. For the neuron network application, it is much faster than RHC and GA, and the runtime does not change much with different parameter combinations.

GA’s performance depends on two parameters: population size and mutation probability. While larger population size is always preferable, the choice of mutation probability could be a double edged sword. A small mutation probability does not allow a good update of the population pool, but a big mutation probability might sometimes ‘trap’ the population in a local optimal solution. Therefore similar to SA, a good choice of parameter combination can be a key to the performance of GA. From the tests on neuron network, the efficiency of GA is much worse than SA, even though population size only covers a very small portion of the entire solution space.

The power of MIMIC is its capability to quickly converge to the local optimal solution (and sometimes also GOS). For all the three problems tested in this report, it only takes less than 20 iterations for MIMIC to find local optimal solution, while it might take other algorithms hundreds of iterations. Therefore, MIMIC is a very efficient algorithm compared to the other 3.