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:

$$Fitness(bit\ string, T) = \max\big(tail(0, bit\ string), head(1, bit\ string)\big) + R(bit\ string, T)$$

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

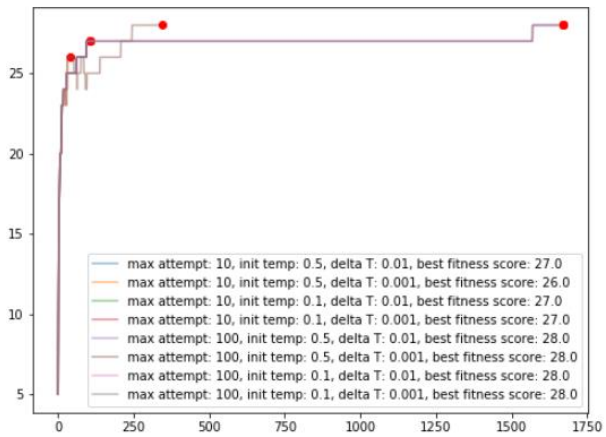
For a given bag with certain weight capacity W, and n items with weights [w1, w2, ..., wn] and values [v1, v2, …, vn], optimize the items in the bag so that the total weight of items is less than or equal to the weight capacity while obtaining the highest total value.

4. Optimization Problems

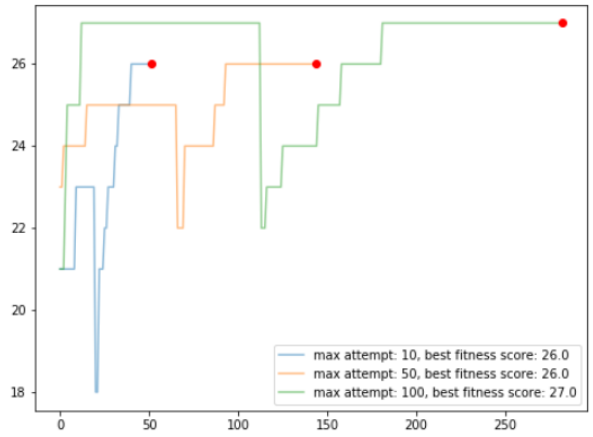
Even though each of the 4 algorithms evaluated in this assignment have fundamental differences, they share one thing in common: at each iteration step, they all attempt to search its neighbors for a better fitness score. The number of max attempt, as shown in this section, is a key factor determining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

4.1 N-Queens

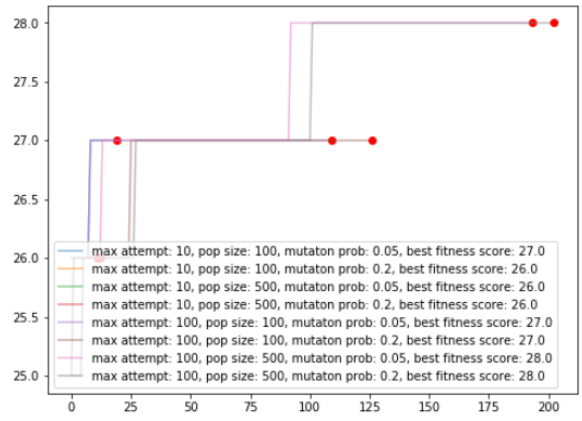
This problem is to highlight the performance of Genetic Algorithm (GA). I only test the 8-Queen problem in this section. The fitness function is designed to check every pair of queens, and fitness score increases by 1 if that pair does not attack. Therefore, the highest fitness score is 28 in this case.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



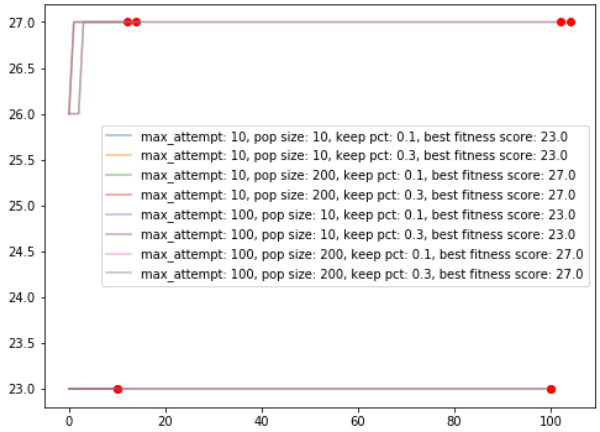
Simulated Annealing



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.

The key parameters for SA are the initial temperature, ending temperature, and temperature step (delta T). While fixing the ending temperature, a careful selection of the combination of initial temperature and delta T could really affect the performance of this algorithm. Since both initial temperature and delta T determine the probability of seeds with worse fitness score are selected for the next iteration, they allow the search for Global Optimal Solution (GOS) to happen in a greater spectrum, other than around the local optimal solution. Therefore, a good parameter settings is the key factor for model performance. Out of the 3 runs tested, the one with max attempt = 100 manages to converge to GOS, while the other 2 converge to a local optimal solution with fitness score of 27. This makes SA a very efficient algorithm compared to the other 3, none of which are able to reach GOS.

Results obtained by RHC indicates the strong effect of max attempts: when max attempt is 10, the fitness score is 26; while fitness scores reaches 27 on 100 max attempts. 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.

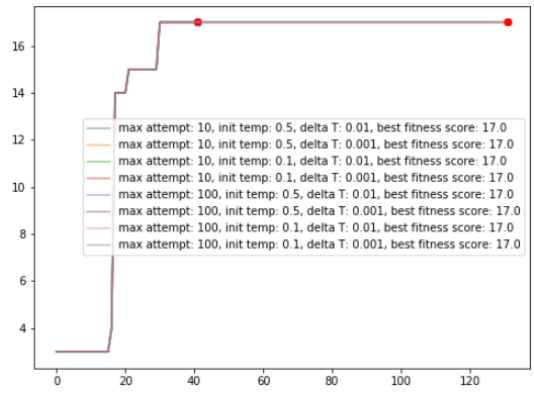
Two more parameters, other than number of max attempts to search for neighbors, are available for Genetic Algorithm (GA): population size and mutation probability. Theoretically, bigger population size increases the chance of getting better parents, and therefore better children. Mutation probability, 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).

Although both GA and SA are able to find GOS based on the parameter settings, it takes much less iterations in GA (around 200), while SA uses more than 1600 iterations to find GOS. For this problem, GA turns out to be a more efficient algorithm than SA.

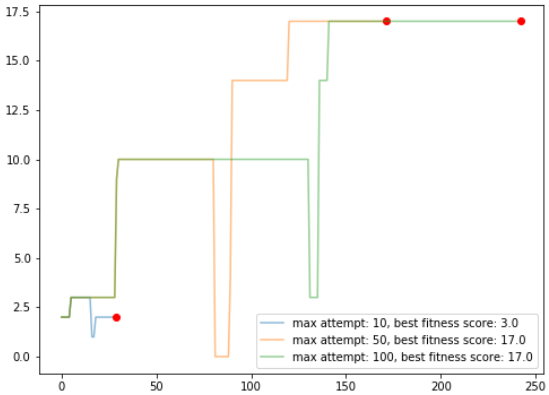
Fitness curves for MIMIC differ from other algorithms in that the fitness scores almost do not change with iterations. The reason is that MIMIC uniformly sample the entire solution space initially, and for every iteration, it only resamples from those with better fitness scores. In other words, from 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

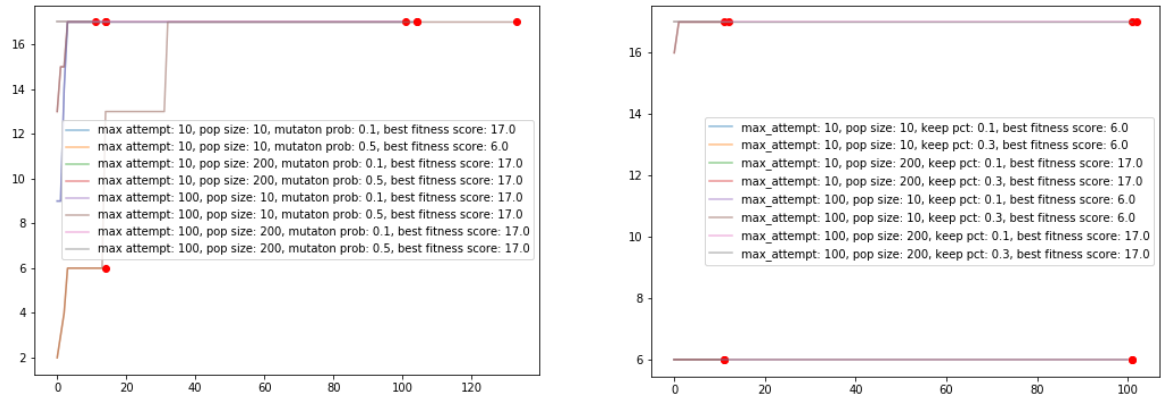
This problem is to demonstrate the benefit of SA. 10-bit string is used for this problem. According to the fitness function defined in Section 2, GOS should have this form [1,1,1,1,1,1,1,0,0,0], with a fitness score of 17.



SA



RHC



GA MIMIC

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

All runs in SA are able to converge to GOS, with the fewest iterations being around 40. For this problem, SA turns out to be a quite resilient algorithm with a descent tolerance on parameter selections. RHC, on the other hand, shows big variation in fitness score (3 vs. 17) due to different parameter settings. It should be noted that the fitness curve for RHC above is based on 1 random restart, which is shown by the dive on each curve. In this example, RHC will not converge to GOS without any random restarts.

Similarly, GA does not show a good consistency either for this problem. A total of 8 runs are performed based on different parameter setting of GA. 7 out of 8 are able to converge to GOS. However, one run only manages to converge to fitness score of 6, a sudden drop compared to the other runs.

The fitness curves for GA with max attempt = 10, pop size = 10 and mutation prob = 0.1 and 0.5 confirms the argument before, that mutation could be a double edged sword. In these two runs, larger probability of mutation in fact deteriorates the final fitness score. That is due to the fact that new children from mutation could end up with worse fitness score than their parents, and higher mutation probability just increases the chance.

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

This problem is to demonstrate the benefit of using MIMIC. 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.

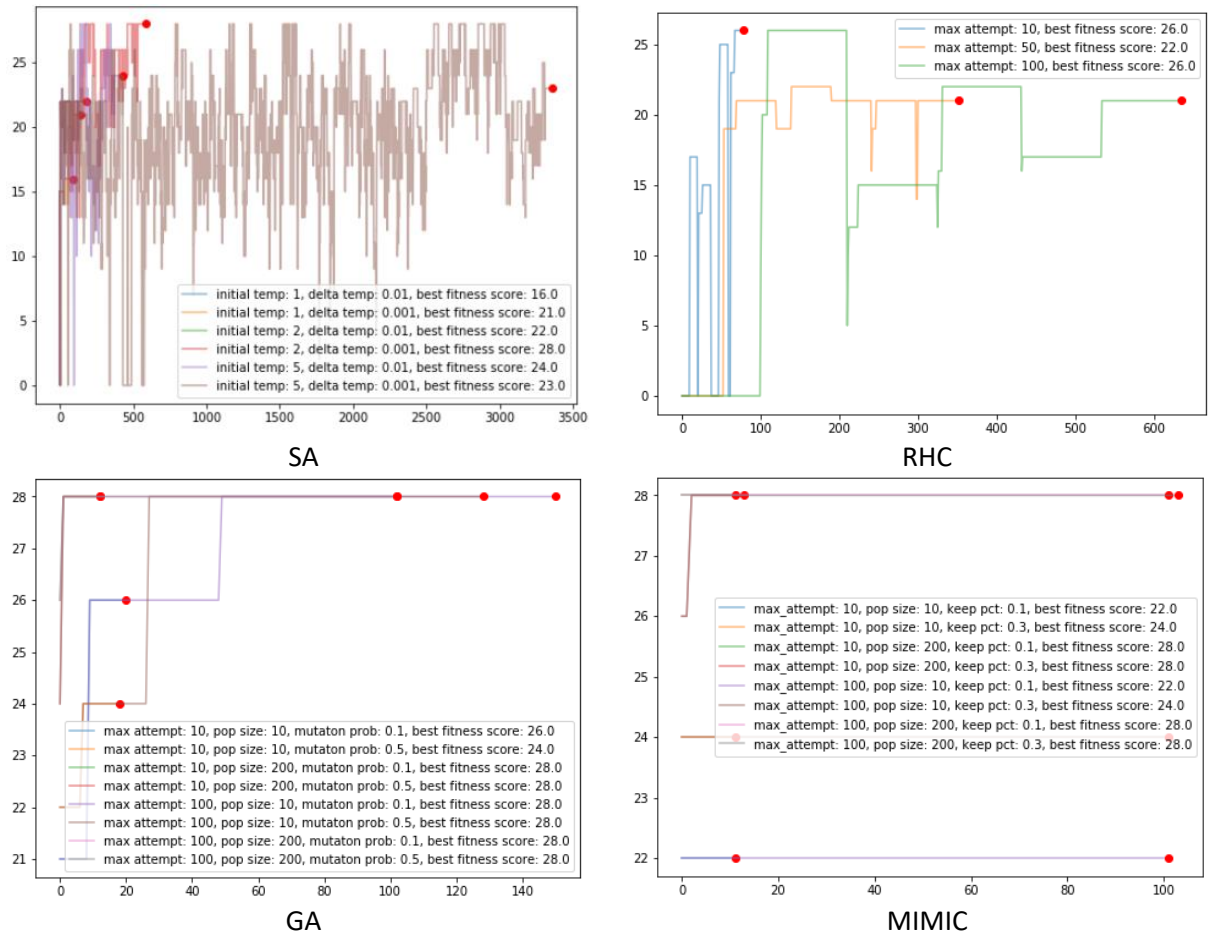


Figure 4.3 Fitness curve for each algorithm on knapsack, with different combination of parameters

The strength of MIMC is already discussed in the previous sections, but here it is worth stating again: that is, MIMIC can converge to the optimal solution very quickly. In this problem, even though 2 runs stop at about 100 iterations, early stopping at 20 iterations, if it happened, also gives the fitness score. In fact all 3 problems, MIMIC converges within 20 iterations. This feature of MIMIC beats all other algorithms, whose number of iterations could vary significantly with different problems and parameter settings.

For this problem, MIMIC also outperforms SA and RHC in terms of fitness score. 4 out of the 8 runs in MIMIC are able to reach GOS, while neither SA nor RHC are able to do so. Note that 1) in this experiment 5 random restarts are used for RHC, compared to only 1 is used for the other 2 problems, and also 2) a max of more than 3000 iterations are performed by SA, much more than the other 2 problems. Both suggest this is a more complex problem that the other two, and MIMIC tends to outperform SA and RHC with more complex problems.

When the number of random restarts is reduced to 1, fitness score is only 23 or 24 for RHC, 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.

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*

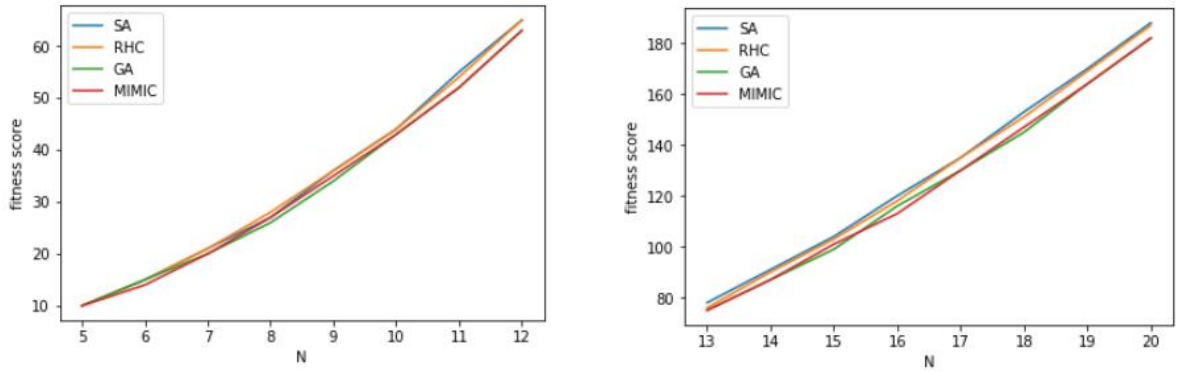


Figure 4.5 fitness scores for 4 algorithms

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.

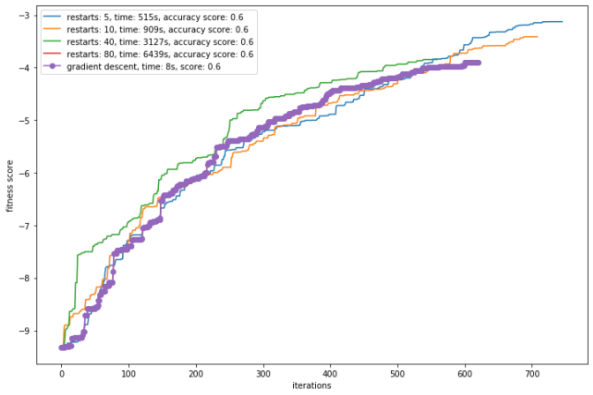
The same EEG data used in HW1 is reused here, which contains ~14,000 data points and 14 features. The target is a binary classification. For each algorithm, back propagation is used as benchmark algorithm for comparison.

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.



	restart	precision	recall	f1
0	5	0.555792	0.529853	0.542513
1	10	0.555792	0.529853	0.542513
2	40	0.555792	0.529853	0.542513
3	80	0.555792	0.529853	0.542513

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.

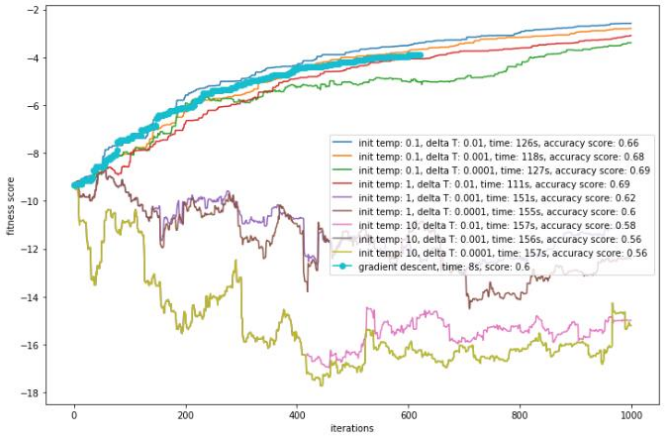
It is proven that increasing number of restarts will help improve algorithm performance. However, given the amount of time required for this algorithm, I did not increase the number of restarts further.

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.

Precision, recall, and f1 scores are also reported for the completeness of the analysis. Precision scores are better than recall scores for all restart values. Since precision measures the proportion of all the correct positive labels from prediction over all the actual positive labels and recall measures the proportion of all the correct positive labels from prediction over all the positive labels from prediction, a higher precision score suggests we have more false negative than false positive, a.k.a, more type II errors than type I errors. F1 score, on the other hand, is just a harmonic mean of precision and recall. The goal is to achieve an F1 score as close as to 1, but depends on the nature of problem (sometimes precision of more important than recall, or vise versa), one metric might be preferable than the others.

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.



	init temp	delta T	precision	recall	f1
0	0.1	0.0100	0.633269	0.503919	0.561238
1	0.1	0.0010	0.590000	0.547875	0.568158
2	0.1	0.0001	0.647471	0.445311	0.527692
3	1.0	0.0100	0.544545	0.437985	0.485487
4	1.0	0.0010	0.578462	0.555201	0.566593
5	1.0	0.0001	0.597692	0.452637	0.515148
6	10.0	0.0100	0.575714	0.463626	0.513626
7	10.0	0.0010	0.539438	0.529560	0.534454
8	10.0	0.0001	0.539438	0.529560	0.534454

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.

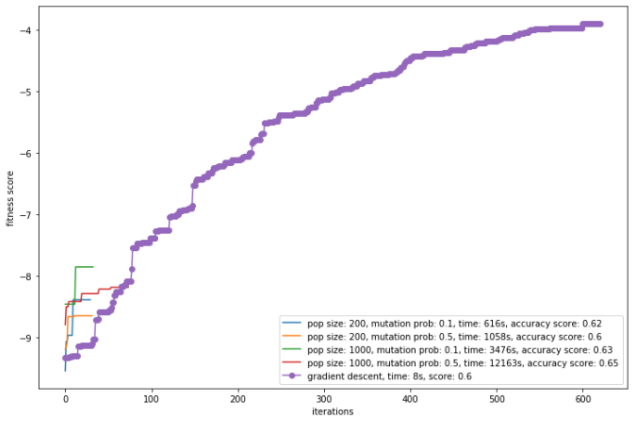
Compared to RHC, SA is very efficient in updating connection weights in neural network, and the efficiency is not quite affected by parameter settings (for different parameter settings in Figure 5.2, training time ranges from 111s to 157s). And yet, the accuracy score and fitness score for the best performer 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.

For different parameter settings, there are some variations in precision, recall and F1. Again, precision scores are better than recall scores for all parameter combinations, which shows neural network

mislabel more 1s to 0s (false negative) than more 0s to 1s (false positive). For all cases, the difference between precision and recall is between 1% to 10%.

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.



	pop size	mutation prob	precision	recall	f1
0	200	0.1	0.562857	0.382344	0.455364
1	200	0.5	0.645526	0.386007	0.483121
2	1000	0.1	0.568621	0.276117	0.371727
3	1000	0.5	0.710000	0.155238	0.254772

Figure 5.3 neural network fitness curves based on GA

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.

Similar to the other 2 algorithms, precision scores are also better than recall scores. However, the difference is widened to 20% to 30%. F1 score for GA is the worst among all 3 algorithms, even though accuracy score is close to the other algorithm.

5.4 Section Summary

It is impossible to compare the performance of each algorithm without knowing the definition of the metric(s). If our metric is clock speed, SA is an easy winner since it is orders of magnitudes faster than RHC and GA. However, if we measure our speed in terms of number of iterations, GA outperforms the other 2 algorithms by a lot.

The comparison becomes tricky when it comes to model accuracy. The default metric used in simulation is the accuracy score. Accuracy score is a good metric when there is no preference for one label over other, and for this EEG dataset, we can argue that the eye state of open (1) is probably as important as the eye state of closed (0). Based on this metrics, the best performer is SA, followed by GA and RHC. If, however, we care more about the accuracy of label 1 than label 0, we will need to select precision over accuracy as our new metric, and as a result, GA is the best performer. It requires domain knowledge and experience to select the best metric for a specific problem. Having all types of metrics listed for comparison is a good exercise (just like what is shown in this report), but in real project it is highly not recommended.

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.