CS7641 ML- Project 2 report

Chenxi Yu

Saturday, Mar 11, 2017

Part 1

1. **Optimization Problems Presentations**

I have chosen the following 3 problems to illustrate the comparative strengths of each algorithm (simulated annealing, genetic algorithm and MIMIC): Four peaks, count ones and Knapsack problem.

The former 2 seems obvious to human but not intuitive to a machine. Four peaks may lead an optimizer towards the local optima, while this may not happen due to the oscillating effect in count ones. The Knapsack problem is helpful for its ability to assess a large number of parameters in different scenarios when comparing different optimization approaches. Moreover, a change in configuration requires different solutions implies that a greedy approach easily fails to recognize the optimal knapsack solution.

A. Four Peaks

The Four Peaks problem is taken from (Baluja and Caruana, 1995). Given a N-dimensional input vector X, the four peaks evaluation function is defined as:

$$f(\vec{X},T) = \max[tail(0,\vec{X}), head(1,\vec{X})] + R(\vec{X},T)$$

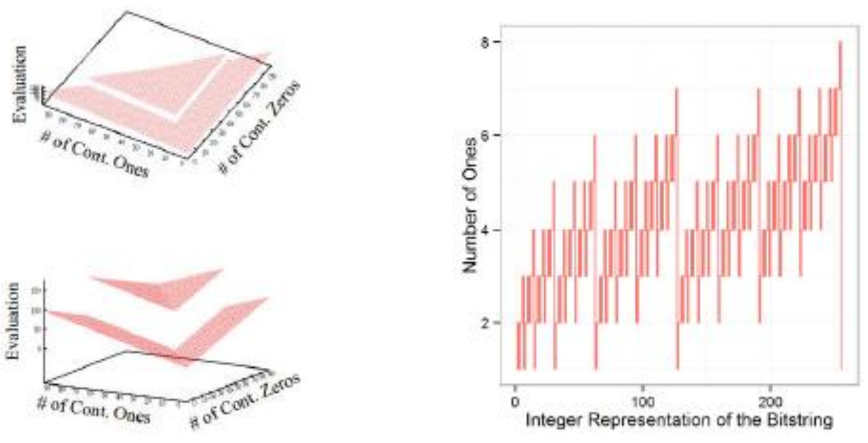
where

$$tail(0,\vec{X}) = \text{ number of trailing 0's in} \vec{X}$$

$$head(1,\vec{X}) = \text{ number of leading 1's in} \vec{X}$$

$$R(\vec{X},T) = \begin{cases} N & \text{if } tail(0,\vec{X}) > T \text{ and } head(1,\vec{X}) > T \\ 0 & \text{otherwise} \end{cases}$$

It is accessible in the Abagail library suggested in the assignment. According to the paper, there are 2 global maxima for this function. They are achieved either when there are T+1 leading 1's followed by all 0's or when there are T+1 trailing 0's preceded by all 1's. There are also two suboptimal local maxima that occur with a string of all 1's or all 0's.



Left: Two views of the same four peaks problem, when N=100 and T=N/5 (Baluja and Caruana 1995).
Right: The value of count ones with respect to the integer representation of a bit-vector.

B. Count ones

Count ones is a discrete function defined for a bit-vector of length N. The goal is to search for a 1 filled solution. This can be defined by trying to maximize:

$$f(\boldsymbol{X}) = \sum_{i=0}^{n} g(x_i)$$

Where $g(x) = \begin{cases} 1 \text{ if } x \ = \ 1 \\ 0 \text{ otherwise} \end{cases}$.

The function oscillating with integer representation brings a great challenge to the machine, since small changes in x (i.e. flipping a bit) will lead to small changes in the results.

C. Knapsack Problem

The knapsack problem is a constrained optimization problem: given a set of items, each with a mass and a value, determined the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.
The knapsack problem can be formulated as follows. Let there be n items, $z1$ to zn where zi has a non-negative value vi and non-negative weight wi. xi is the number of copies of the item zi. This is subject to the constraint W, which is the weight we can carry, and also ci which is the total number of copies we have for each item xi. Then we must attempt to maximize:

$$\sum_{i=1}^{n} v_i x_i \text{ subject to } \sum_{i=1}^{n} w_i x_i \leq W \,, \quad x_i \in \{0, 1, \dots, c_i\}$$

2. Parameter Tuning

The Abagail library is used and each algorithm as required were run 10 times after which the results are averaged. I use an exhaustive grid search to determine parameters for each algorithm.

For SA the parameter varied was:

- Cooling Schedule - 0.65, 0.75, 0.80, 0.95

For GA, five parameters were varied:

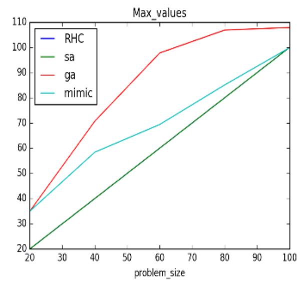
- Population Size - 50%, 100%, 200%, 400% of a Problem Size
- Crossover Type - One point, Two Point, Uniform
- Crossover Rate - 10%, 40%, 60%, 80%, 100%
- Mutation Rate - 0.001, 0.01, 0.1, 0.2, 0.5, 0.8, 1

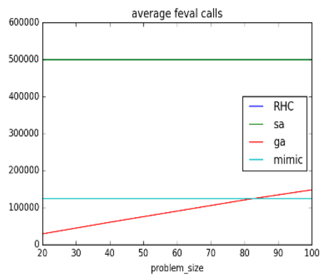
For MIMIC, two parameters were varied:

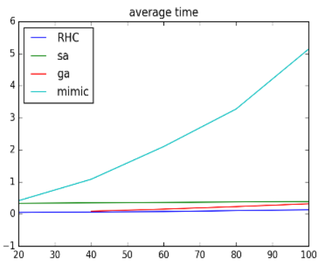
- Number of Samples taken for each iteration - 40, 60, 80, 100, 120, 140
- Number of Samples to be kept each iteration - 20%, 50%, 70%, 90%

3. Results

a. Four Peaks







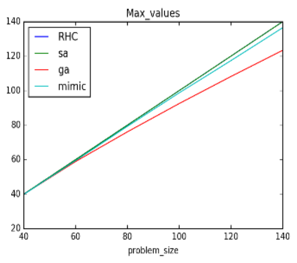
Four peaks was run for n = 20,40,60,80,100, where n is the dimension of the input vector and the three graphs above show the average maximum value, the average of function calls and the average time for each algorithm. Also the RHC is identical with SA in the first 2 graphs but the curve color indicates only by the green one.

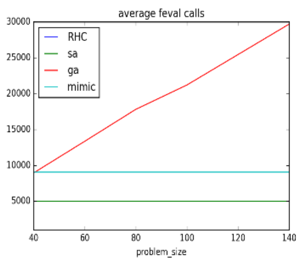
GA is clearly the best with the highest values requiring comparatively fewer function calls in general while taking a relatively low average time. Noticeably, GA finds the global maximum on problem size 20, 40 and 60.In the function evaluations curves on figure 2 above, MIMIC and SA have fewer functions calls while generating better results. However, in figure 3, MIMIC definitely runs longer due to its long unit run time.

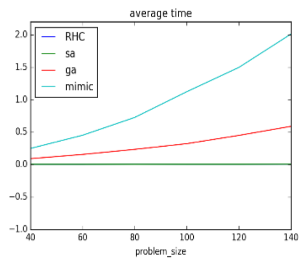
Overall, RHC and SA only make it to the local maxima for each problem size because of their greedy nature thus failing to cross the large gap in the four peaks problem shown in the graph earlier. However, the ability of explaining structures in MIMIC enables it to perform better than the local optima, while GA's crossover operation has allowed it to propagate into a global optimal solution.

b. Count ones

Count ones was run for n=[40,60,80,100,120,140], where n is the size of the problem. Also the RHC is identical with SA in the graphs below but the curve color indicates only by the green one.



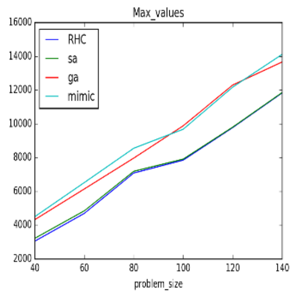


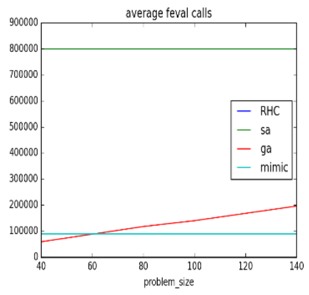


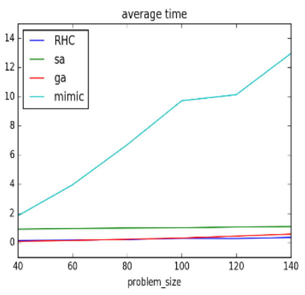
For every problem size, SA and RHC managed to reach the optimal point for the problem. While there is some difference between MIMIC and the optimal result, GA does slightly worse than the others. In figure 2, it is obvious that GA involves a linearly increase in function calls while MIMIC does the same yet almost double the function calls of RHC and SA as problem size increases. In figure 3, the average time performed by MIMIC has an exponential pattern whereas GA grows slower and SA and RHC does not change in this aspect.

The reason why SA and RHC prevails is that the count one problem places less importance in structure exploration or generation shifts where MIMIC and GA do better. In comparison, this problem reveals the computational and time cost at exploring structures and generating features. In short, SA and RHC are better at solving problems with no structures, especially when the global optima is not much different from a local one.

c. Knapsack







The knapsack problem was run by varying the number of items, n=[40,60,80,100,120,140], the maximum weight and maximum volume of the items were held constant at 50 for both, with weights and volume of each item randomized for each particular run. Also, in figure 2 above, both RHC and SA are in the same green line.

The graphs indicate MIMIC performs the best in this problem comparing with the others. Since the knapsack problem is NP-hard for an optimization solution, there is a high chance for RHC and SA to perform worse as expected. Overall, MIMIC and GA do better not only in terms of maximizing the value but also in terms of function calls numbers while RHC and SA have similar results except for a small difference in the average time aspect.

In figure2, there is a striking comparison in curves on the group of RHC and SA with the group of GA and MIMIC. With more than 4 times of function calls, RHC

and SA perform at least 30% worse as problem size grows. Apparently, a greedy search approach by RHC and SA without exploring structures does not do well enough in this problem. But GA's performance is comparable to MIMIC because GA's attempt to generate features by crossover is almost equivalent to the process of enabling GA to find the underlying structure of a problem. Yet, GA does work harder in making more function calls as shown in figure 2 while MIMIC consumes more time in each iteration as shown in figure 3.
In conclusion, MIMIC and GA would perform better when retaining history is essential. Nevertheless, when the optimal problem places less difference in global optima and local optima, RHC and SA could be a better solution as shown in count ones.

Part 2

1. Neural network problem presentation

The image Segmentation classification for every pixel from UCI Machine Learning repository is used again for this portion of the analysis. Each algorithm was run until it met the tolerance of 1e-6 or 20 minutes of run time and MSE was used. Based on my reading (Hagan, Martin T.), I set a value for the upper bound on the number of hidden neurons that won't result in over-fitting exhibited as follows:

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$$

N_i = number of input neurons.
N_o = number of output neurons.
N_s = number of samples in training data set.
α = an arbitrary scaling factor usually 2-10.

(I set Ns as 2100, Ni as 18 and No as 1 and alpha as 5 and thus the number of hidden neurons I use is 22)
For the sake of fast implementations on the algorithm, I use Matlab. Its function "patternsearch" is used for implementing RHC, while "ga" is for GA. SA was implemented through our own codes, this may explain the lower number of function evaluations which SA had. The results were optimized and chosen based on the lowest validation error found and are exhibited as below.

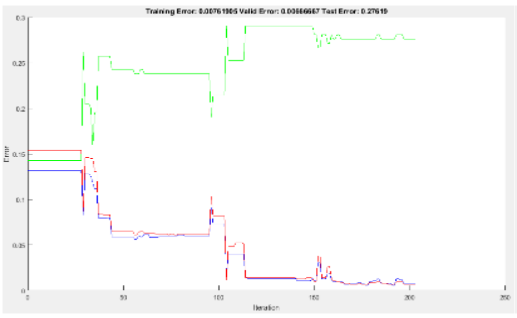
Algorithm	Optimization Time	Function-Evaluation	Training Error	Validation Error	Test Error
Backpropagation			0.1190	0.1400	0.1524
RHC	20 mins	54713	0.1314	0.1543	0.1429
SA	20 mins	56407	0.1314	0.1543	0.1429
GA	20 mins	42600	0.1410	0.1733	0.1857

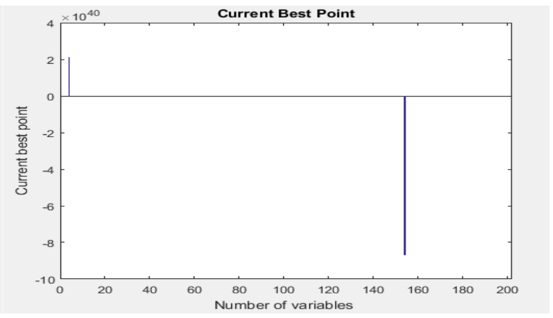
All approaches have very little differences in number shown in the above. It is surprising that the error results above for RHC and SA are the same, while GA seems perform the worst. Yet, it is obvious that backpropagation still prevails among all 4. However, we will further examine each randomized optimization algorithm resulting in different weights for the neural networks.

2. Results:

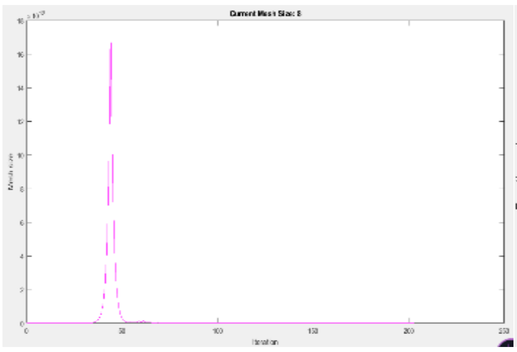
A. RHC

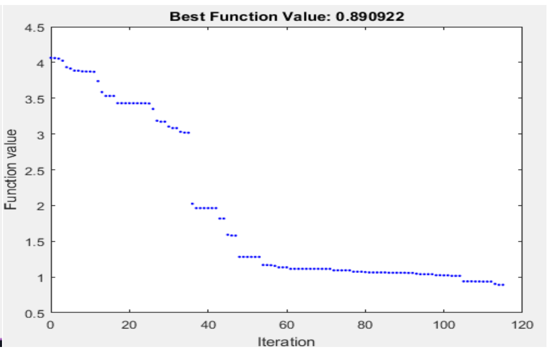
(The green curve is for the test error and the red and blue for the validation and training errors)





(figure 1) (figure 2)



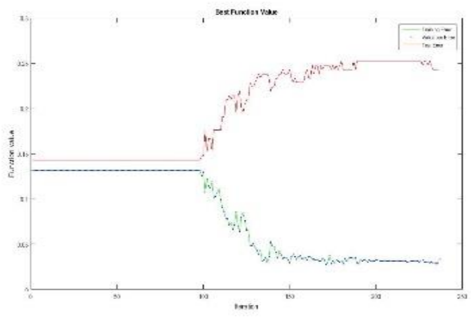


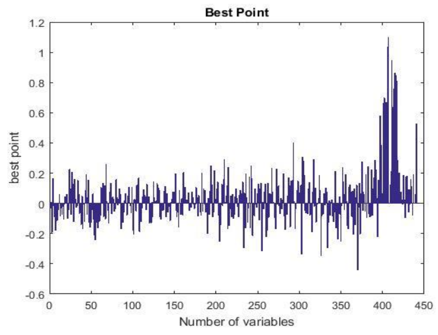
(figure 3) (figure 4)

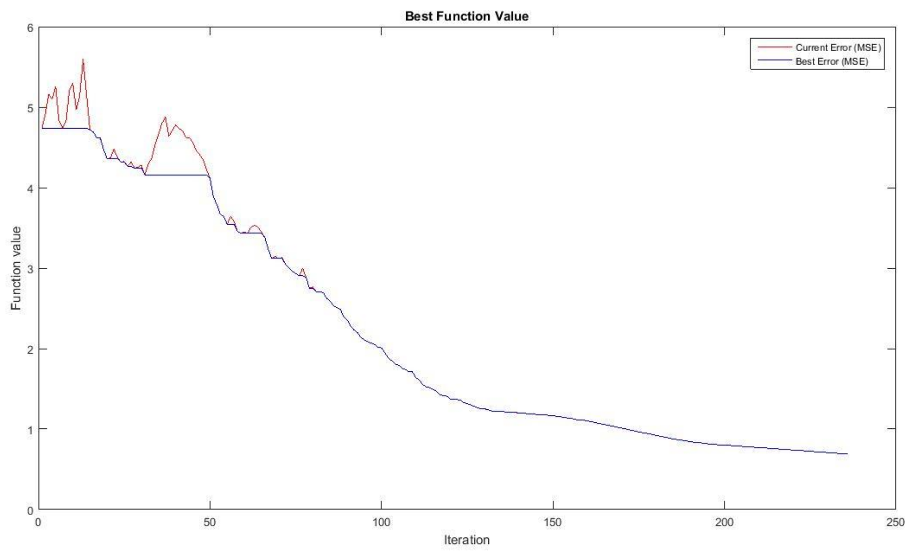
The graph above shows the progression of the search for the minimum point and the number of function evaluations at each iteration. With the timer at 20 minutes, the algorithm did not terminate early whenever caught in a local minima, instead expanding the search mesh to determine if any better point could be reached. It seems the best function value drops suddenly from 3 to 2 in just one more iteration. Also, based on the observation between 35 and 50, we spot a great improvement. The mesh size (figure 2) peaked at around 45 and the best function value (figure 4) started to stall. RHC demonstrates its greediness when examining the weights. It picks up a few features and gives weightings in excess of 2×10^40

B. SA

(The red curve is for the test error and the blue and green for the validation and training errors)



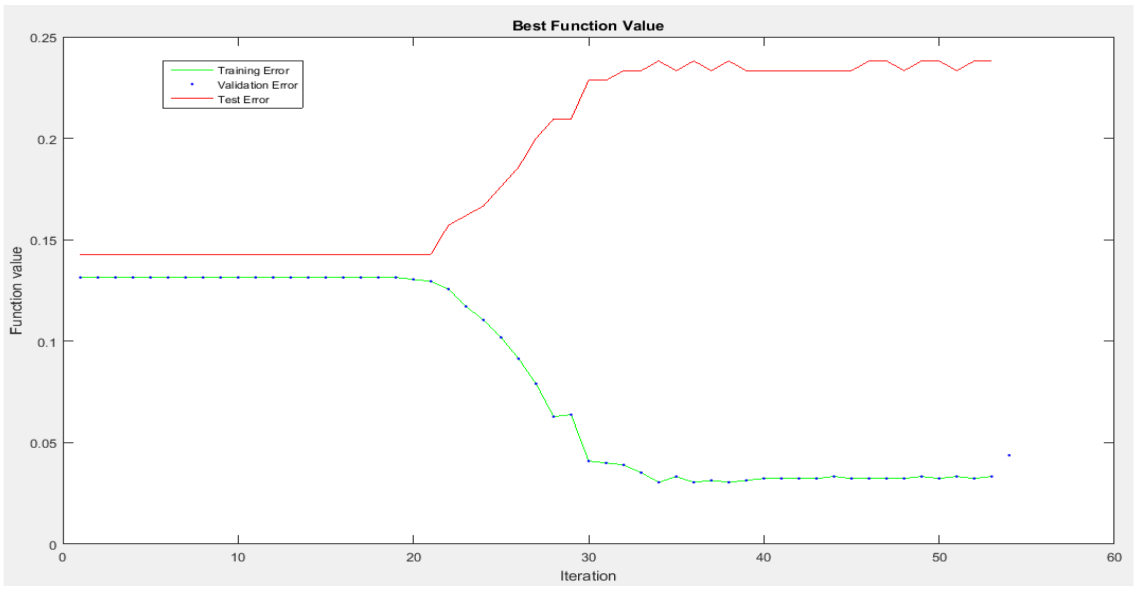




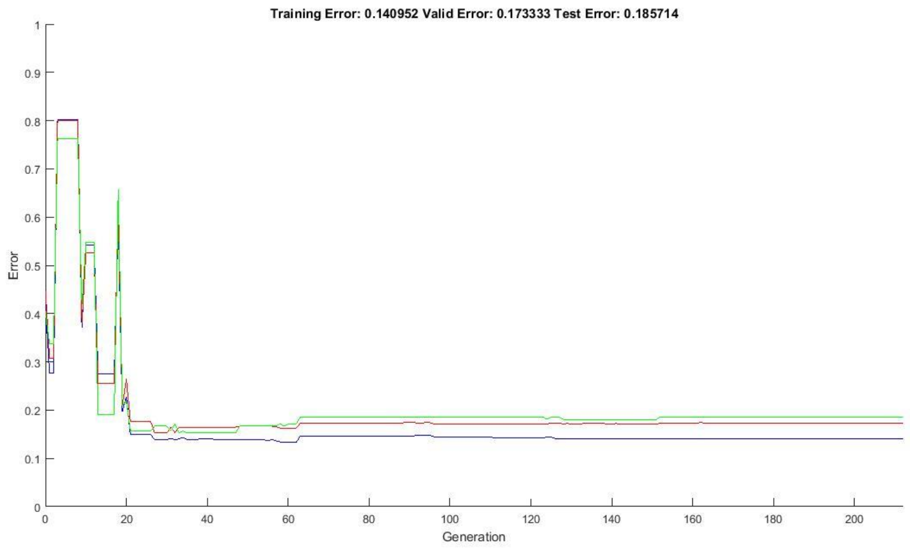
Matlab's native approach was tested but decided not used because the default settings somewhat performs way off my expectation when it comes to errors in validation. Like the 3 problems presented above, different cooling schedules (exponential, fast and boltzman annealing schedule) were tested and the best one—the 95% exponential was chosen. This is because the 70%'s has the temperature drop too fast at the early iterations and thus slowing down each iteration afterward comparatively. As a result, even though the best function value is high in the early iteration on the 95%'s mode, faster early iterations allows it to reach a lower MSE in 20 mins.

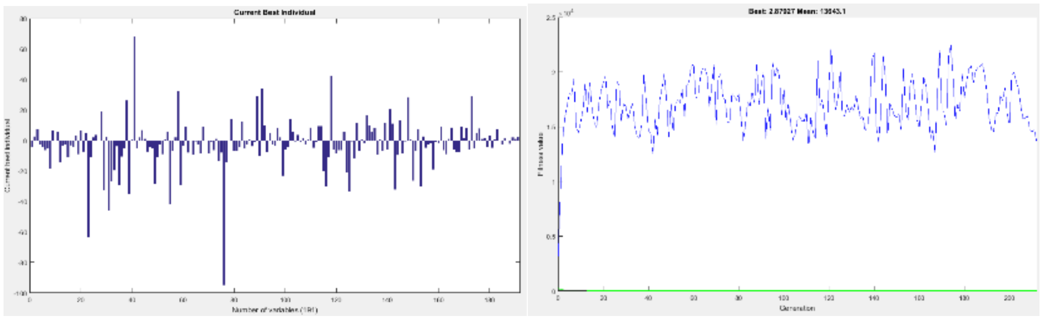
As is shown above in figure 1, after 95 iterations, SA attempts to take in points that may not be the best. However, after the temperature falls off enough, it does indeed behave quite similar to RHC. However, the validation curves in figure1 on SA performed in a pattern similar with the RHC's, except that the SA is likely to accept a local minima after its 95 iteration while RHC does it at around its 25 iteration. This indicates the hybrid nature of the SA via allowing a random walk at higher temperatures and RHC behavior at lower temperatures. This is evident when referring to figure 3 above where best MSE and current MSE exhibit no difference as it iterates further after 70s.

So far, why does the validation and test plot behave in such a way where the gap is widened in the middle of the plot? I believe this is because the data size difference between the training set, validation set and the test set. Their ratio is at 5:5:1. However, It appears that the SA approach suffer from overfitting compared to RHC, since the training error curve and the validation error curve seem to merge into the same position in the graph space. However, this could not be improved via a different cooling schedule (i.e. the 70% exponential shown as below).



C. GA





(Figure 1 shows the validation curves based on MSE while figure 2 indicates the best individual based on minimizing errors and figure 3 reveals the best/mean fitness curve based on the best fitness and the mean fitness. The green curve is for the test error and the red and blue for the validation and training errors. In figure 3 the green curve is for the best fitness and the blue curve is for the mean fitness)

Parameter tuning via exhaustive search was used to determine the optimal parameters as described earlier. In the first figure, if given a more lax conditions, GA would have terminated early by generation 20. Yet, it continues seeking better points. However, the improvement from the 20th generation on was rather limited. This trait is a good reflection on the cross-over model structural changes of connections of the layers. Therefore, it could evolve the generation fast at the very beginning but not much after a tipping point.

The validation curves pattern look very different from the RHC and SA's and GA's has the worst in its 7 or so iteration. This is a disadvantage for GA's application in an optimization problem. As the data size increases, the amount of computation required for generating enough new generations could be uncertain. We never know when it behaves abruptly unless GA explores the solution space thoroughly and the tipping point (asymptotic convergence) is reached.
Ideally, the best/mean fitness curve should be monotonically non-decreasing. The best fitness curve will always be monotonically lower or higher than the mean fitness curve. In our case, we could see the running pattern of best/mean fitness curves in the figure is quite stable, though the mean fitness is fluctuating around certain value. This indicates the algorithm is exploring the solution space adequately and cross-over operation runs normally.

However, based on the best point and how stable it appears in GA's graph, we could believe it is the best model for weighing our neural network. This is mainly due to the number of variables needed in GA's is just roughly 22 while SA requires more than 100 and RHC more than 150. More importantly, our data set classification algorithm based on a high dimension (18 variables) would not intuitively be solved by a pure greedy based algorithm. The MSE curves difference of the GA's and the former 2 algorithms help explaining GA handling overfitting well.
Overall, GA appears to be the strongest performing algorithm since under more lax conditions it would still appear to perform well. In addition, when it comes to managing weights for neural networks, backpropagation itself has not been a bad solution as shown in our error validation analyses table earlier. Even in our matlab experiment, the amount of iteration and time taken have not been costly compared with the other 4 algorithms we tried.

Conclusion:

Base on the above illustration, we can see the close relationship between SA and RHC. Both algorithms have extremely similar performance. While SA requires fewer function evaluations to get a similar RHC's performance if the SA's cooling schedule is set to drop fast. However, if the learning rate is ideally set to be slow enough, SA is more likely to escape local optima. Yet, this is unlikely to happen in reality since a real world problem is complex enough to have the SA run forever.
MIMIC and GA perform better in more complex optimization problems where exploring structure is important to rule out the distractions of local optima. For

exhaustive search problems, SA or RHC would be preferable via setting up searching constraints (i.e. a grid search approach) depending on the function evaluation cost.
In short, it depends on our domain knowledge on the problems nature and what aspects of the analyses are most important therefore determining intuitive algorithms to use. For a time sensitive and simple enough problem, RHC or SA would be preferable. However, if overfitting is a big concern and computational costs is not, GA is a better option. Further, when it is expensive to compute functions, MIMIC prevails among the four.

Baluja, S. and Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. Technical report, Carnegie Mellon Univerisity
Hagan, Martin T., Howard B. Demuth, Mark Hudson. Beale, and Orlando De Jesús. Neural network design. S. l.: S. n., 2016. Print.