Author: Apurva Sharma
GTID: 902490301

Assignment 2 : Randomized Optimization Techniques

Problem Set Description and Motivation:
The assignment asks us to choose three optimization problems, one suited for Genetic Algorithms, one for Simulated Annealing, and one for MIMIC. Being relatively unsure of which problems should work better on which algorithms, I used ABAGAIL to run tests over 100 iterations for :

1. Continuous Peak Evaluation Problem
2. Count One's Problem
3. Knapsack Problem
4. Flip Flop Function
5. Traveling Salesman Problem

I ran Randomized Hill Climbing, Simulated Annealing, Genetic Algorithms and MIMIC on all these problems to find the following results :

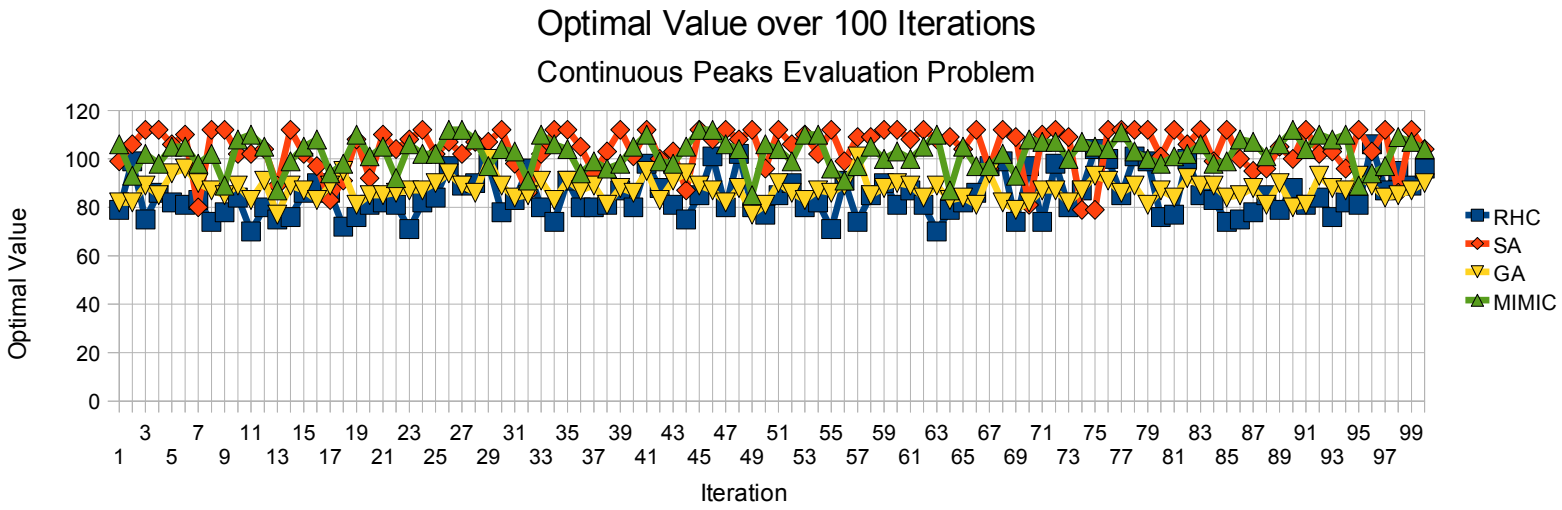


Fig 1 (a)

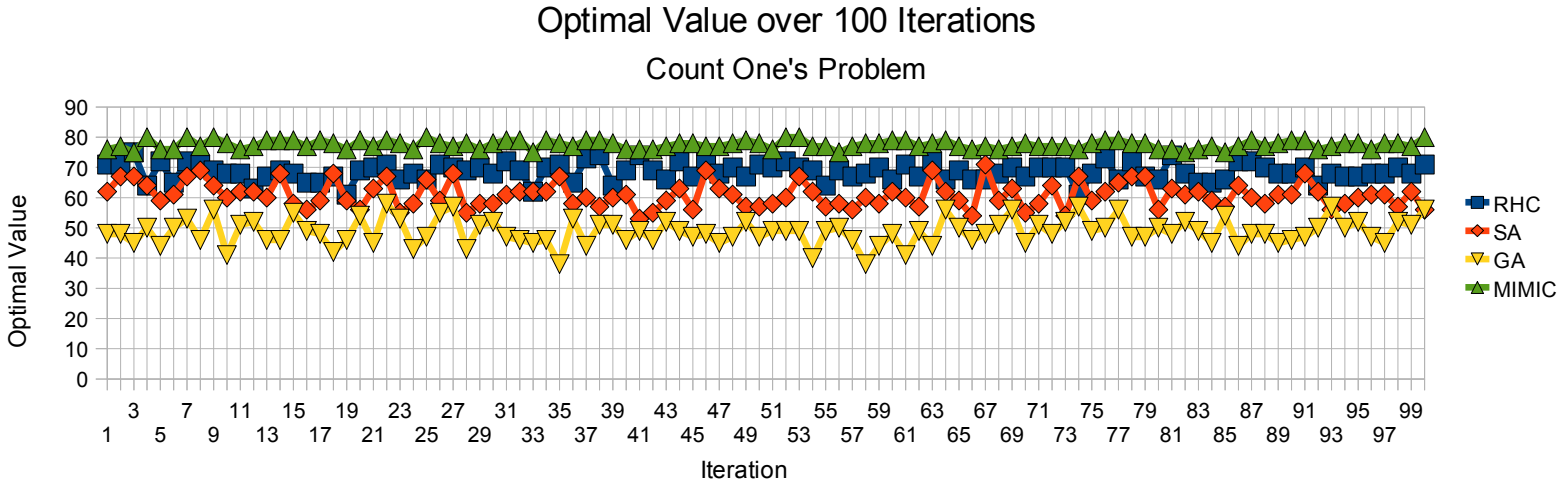
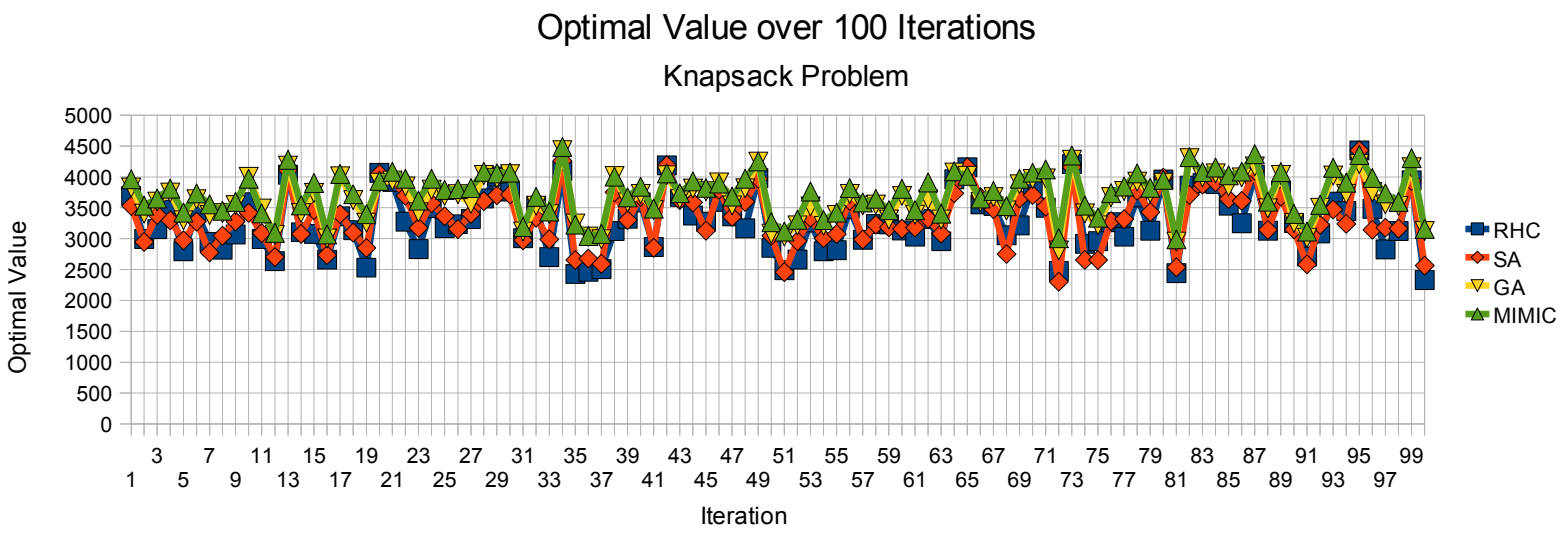


Fig 1 (b)

While this may sound like a hack – running the tests first and then deciding on the problem, I am hoping to pick out the problems in which SA, GA and MIMIC outperform the other algorithms respectively and then try to analyze why this is the case.

Also worth noting is that all the problems have been modified to be maximization problems. For example, the TSP requires minimization of the route distance. I change this to deal with maximizing (1/distance) i.e. smaller



the distance, greater the value of the optimization parameter.

Fig 1(c)

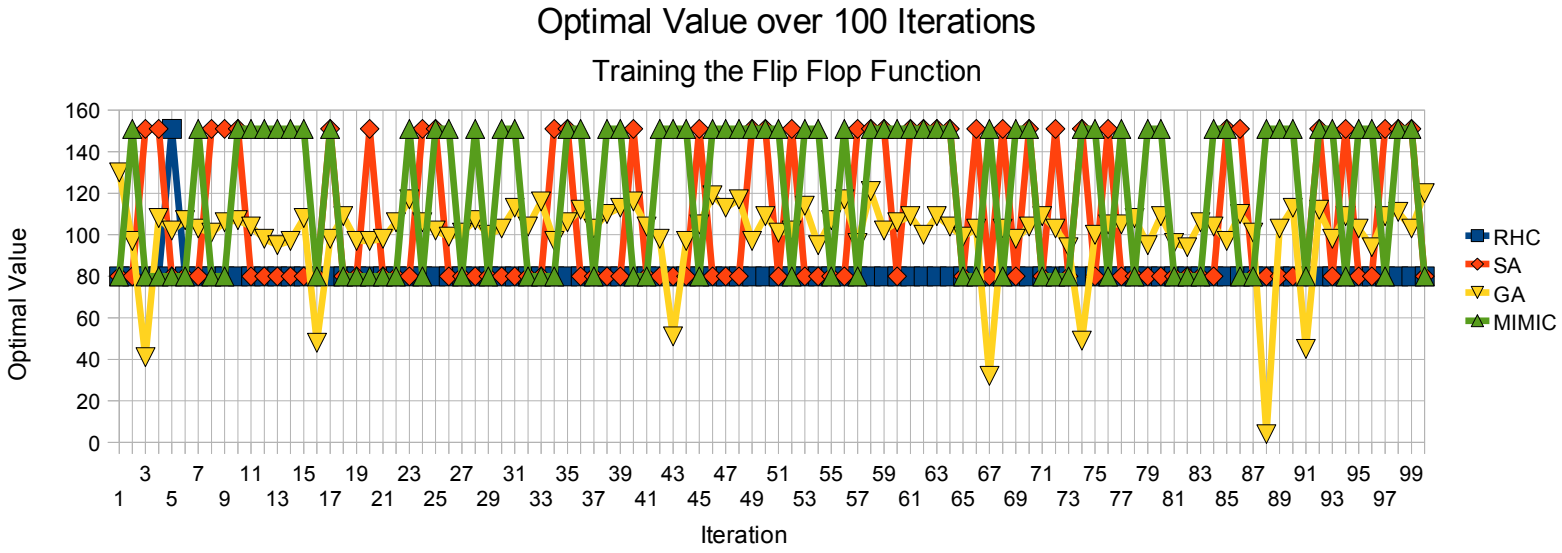


Fig 1(d)

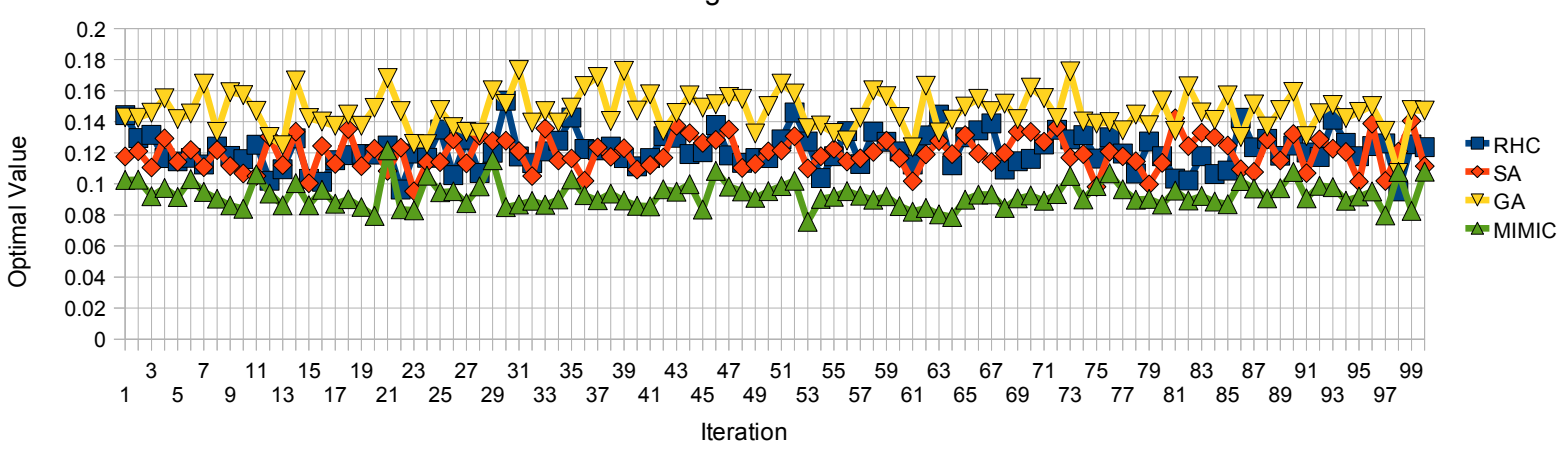


Fig 1(e)

Fig 1 (a) through (e) : Optimal values over 100 iteration for different problems

These readings can be summed up as:

Problem	(Avg,Max) for RHC	(Avg,Max) for SA	(Avg,Max) for GA	(Avg,Max) for MIMIC
Continuous Peaks	(84.5, 106)	(101.5, 112)	(87.05, 101)	(102.39, 112)
Count One's	(68.54, 75)	(60.91, 71)	(48.64, 58)	(77.56, 80)
Knapsack	(3291.97, 4429.5)	(3343.23, 4429.5)	(3645.03, 4431.36)	(3738.81, 4485.17)
Flip Flop	(80.71, 151)	(105.56, 151)	(100.23, 30)	(120.47, 151)
TSP	(0.12, 0.15)	(0.12, 0.14)	(0.15, 0.17)	(0.09, 0.12)

Table 1: Avg, Max Optimal Values of Problems Vs Algorithms

This preliminary test itself has a lot of valuable information which I will describe further in my analysis of problems chosen. For the time being, it is enough to say I chose the following problems based on the average and maximum values.

1. TSP : This seems to be a clear winner for GA with both its max and avg values being better than other algorithms.
2. Count One's Problem: MIMIC does considerably better than other algorithms for Count One's and Knapsack problem. I chose Count One's as it has a much more interesting **Path to Optimal** which I will describe later.
3. Continuous Peaks Problem: This was the trickiest choice. On the face of it, by seeing Fig 1(a), one might be tempted to say that SA does the best. However, Table 1 conveys much greater detail. MIMIC outperforms SA by reaching the same max value (112) while having a higher average. However, as the graph suggests, SA hits the peak more frequently, dropping the average because it does very poorly in some cases, thus, skewing the values.

Traveling Salesman Problem:

Concisely this problem is well defined by Wikipedia as:
"The **Travelling Salesman Problem** (**TSP**) is a problem in combinatorial optimization studied in operations research and theoretical computer science. Given a list of cities and their pairwise distances, the task is to find a shortest possible tour that visits each city exactly once."
TSP has uses in many fields like planning, circuit board design, DNA sequencing etc. and is found to be NP-hard.
For my implementation, I choose ABAGAIL's Traveling Salesman Test which randomly generates x and y coordinates for 50 points (cities). The evaluation function's job, then is to calculate the Euclidean distance between any two given points. Like mentioned before, this is a minimization problem which is converted to a maximization problem by considering (1/distance) as the parameter to optimize instead.

Intuitions before Experiment:
TSP is particularly interesting and hard because, replacing a sub path changes the whole problem greatly. This causes the problem to be particularly discontinuous. Thus, I expect approaches like RHC and SA which rely on the concept of neighbors to not do very well in the context of this problem. I say this because as opposed to count one's functions which typically move steadily and gradually (say, 1 bit improvement at a time), the variations in this problem are a lot more discontinuous and sudden. Because of this, the illusion of arriving at a global maximum is possible. GA and to an extent, MIMIC are not dependent on this concept of neighbors. For MIMIC, I am a little skeptical as if the initial population itself is pretty bad, we will arrive at a good distribution after a fair number of iterations. GA on the other hand, may do a bit better because of availability of operations like mutation and randomly picking points for new population.

Experiment Setup:
I will perform two basic sets of tests for all algorithms for this problem:

1. Run all the algorithms on TSP 100 times. At any given iteration, the same travel map is being used for all the algorithms so that their relative performance can be analyzed.
2. Try to capture how the algorithms are reaching for the optimal value for randomly chosen iterations. This is done by modifying the FixedIterationTrainer to capture the values in each iteration. This will really help us visualize how the algorithms are behaving and how the optimal value changes with each step. Iteration cap for all algorithms was 2000.

Analysis, Possible Explanations and Lessons Learnt:

Figures 1(e) and 2 are observations made for setups 1 and 2 respectively. Both the experiments give a reasonably good picture of what actually goes on in the problem at hand.

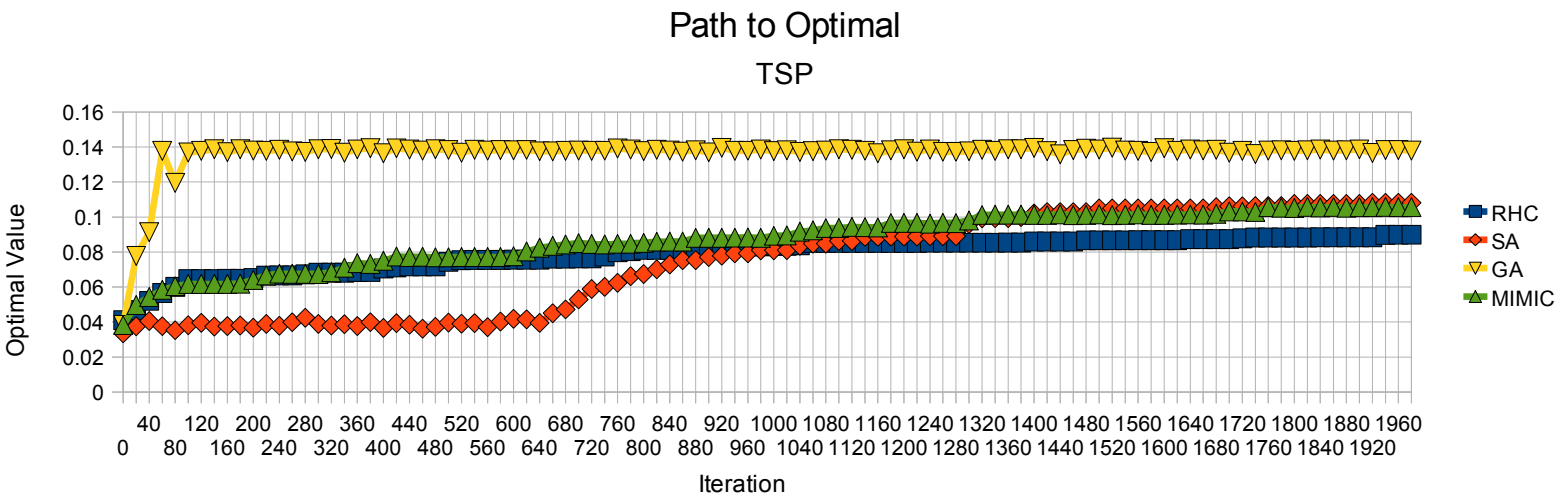


Figure 2: Path to Optimal Comparison for TSP for a sample run (Iteration limit = 2000 for all algorithms)

Experiment 1 (Fig 1 (e)) tells that GA outperforms all other algorithms by a fair margin. Relatively, RHC and SA seem to have struck on local maximum and don't reach the optimal values that GA manages to reach. MIMIC's performance is relatively the worst in this scenario.

The part of RHC and SA not being the best performers in this problem is understandable, as the problem is not smoothly continuous, with changing of sub-paths affecting the whole solution greatly. GA doing well also makes sense as it does not rely on the concept of neighboring values. However, MIMIC being the least accurate is a little counter intuitive. One possible reason for this happening is that the initial population chosen itself is not very good. As a result, it takes a greater time for MIMIC to be able to get to values closer to the global optimum. I think that increasing the number of iterations for MIMIC should lead to better results. GA would have faced the same issues but it has the option of mutations which may have worked well to get to a better population more quickly. Figure 2 substantiates this argument by showing that GA has maybe stabilized after around 120 iterations, but when we stop the run (at 2000 iterations), MIMIC still seems to be on the rise. To confirm this, I ran MIMIC with 5000 iterations instead about 20 times and on 2 out of those 20 times, I got values which were better (~0.15) than the best values for MIMIC in Fig 1(e). While this is not a very impressive figure, it still is possible that this might indeed be the case.
Another possible reason is that one of MIMIC's strengths is that it tries to capture the underlying structure of the problem at hand (Ref: Charle's Paper on MIMIC). This key property is missing in TSP as there is no well defined way

to solve this problem in polynomial time.

Experiment 2 (Fig 2) tells us the true behavior of these algorithms. One of the most clear trends to be seen is the RHC vs SA approach. For this sample, we can see that RHC naively tries to pick only the neighbors better than it and gets stuck at a maximum of near 0.9. On the other hand, SA takes values of neighbors which are not necessarily always better, thus, enabling it to explore other possible maximas. This is seen very clearly as SA initially picks up values worse than RHC, but this makes it get to a better maximum value of nearly 0.11
GA seems to have found a good population very quickly and stabilizes after that. MIMIC on the other hand, still seems to be on the rise at the end of 2000 iterations. Another trend verified is, that for RHC, the graph in Fig 2 will never dip as neighbors are chosen only if they are better.

Lessons learnt are that every algorithm needs its own specific time before it can start showing results. This is not always necessarily a bad thing as we may prefer an algorithm that takes more iterations but has more potential of reaching a global optimum over an algorithm that very quickly settles on a local optimum. Another lesson learnt is that algorithms like SA and RHC are maybe likely to perform better on functions that are smoother. This will be verified further as I tackle other problems. Another key lesson learnt is that MIMIC is better suited for problems whose solutions possess an underlying structure that MIMIC can exploit.

Count One's Function:

This evaluation function is used to count the number of one's in a given input and try to maximize it. While this seems like a fairly trivial problem for humans – just make the bit string as all 1's, this logic is not present in these algorithms. They try to get to the correct value by randomly exploring the search space and trying to get to better values.
In ABAGAIL, N is set to 80, which means that 80 is the correct global maximum.
This is a discrete continuous problem, ie, it is well defined at all points and increases by a discrete value at a time (since the number of 1's is a discrete value).

Intuitions before Experiment:
I think this problem has a much smoother curve because of the fact that the context of the problem doesn't change much depending on the sub-solution chosen. Based on understanding from TSP, I feel RHC and SA should do reasonably well, SA more so than RHC. Also, since MIMIC does not so great on a not so smooth problem like TSP, it will be interesting to see if it does any better on problems which are smoother instead. Add to this the fact that the solution here has a neat underlying structure which MIMIC can possibly exploit.

Experiment Setup:
The experiment setup remains the same as in TSP, as we still have to explore the same things- which algorithms perform better and what does the path to optimal value look like.

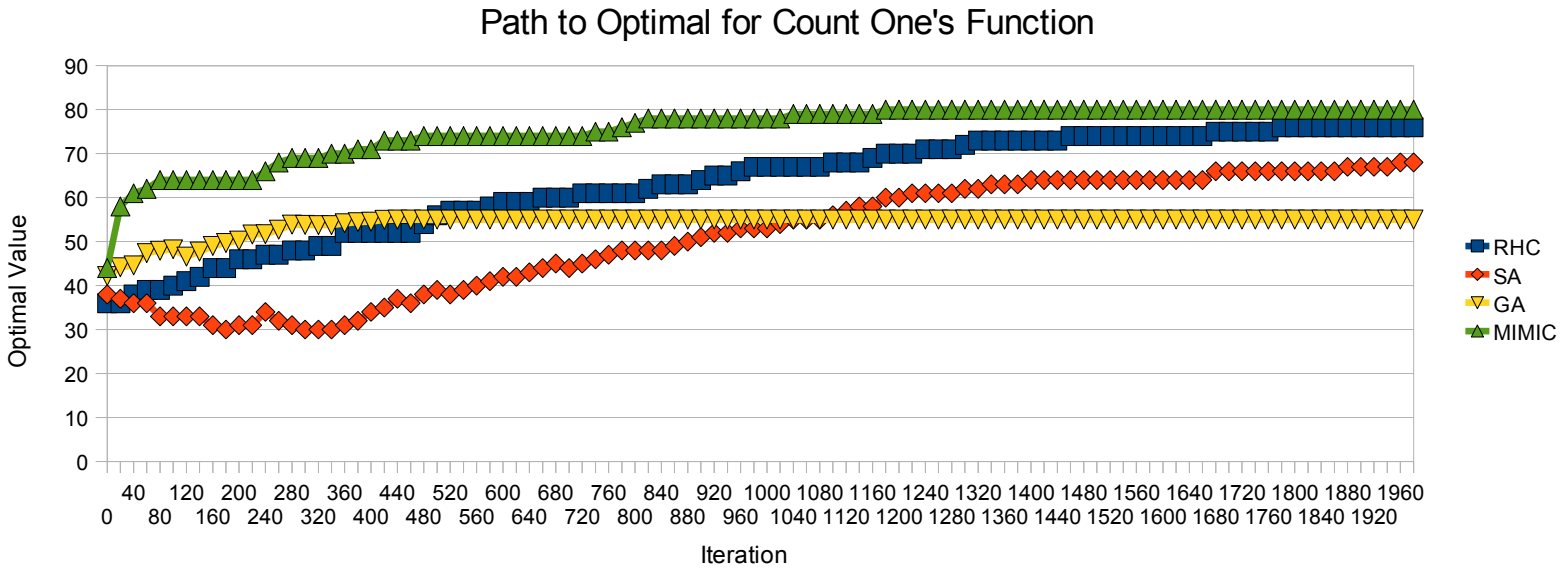


Figure 3: Path to Optimal Comparison for Count One's for a sample run (Iteration limit = 2000 for all algorithms)

Analysis, Possible Explanations and Lessons Learnt:
Figure 1(b) and 3 correspond to the experiments carried out for CountOne's evaluation function.

Figure 1(b) tells compares performance of all algorithms run 100 times. An interesting pattern is beginning to emerge because of carrying out this experiment. While MIMIC did the poorest on TSP, it does the best on a problem which is much more smoother. This gives us some notion of the kind of problems MIMIC is better suited for. Add to this the fact that the best performing algorithm for TSP (GA) performs the most poorly for this problem!
Also, since the solution to the problem at hand has a neat underlying structure, MIMIC is able to combine information form all maximas to yield the best performance.

One of the reasons why the GA may not be working well is because of the population size chosen (20) and the choice of cross overs (20) and mutations (0). This is likely to improve if I tweak the population size (and it does, as I verify later) but more importantly we can see that GA's are prone to local maxima problem too. In Fig 1(b) on numerous occasions, GA has settled on a value near 60 which is clearly sub optimal.

RHC and SA give a performance which is not too bad (Avgs 75 and 71 resp) .
However, there is another observation to be made here. For some reason, SA is not able to find a peak beyond RHC in this case. This means that in spite of taking neighbors which are not always better, SA does no better than RHC in this case. This leads to a dip in the average performance of SA without any betterment in the maximum value reached.

Figure 3 shows how these algorithms go about trying to reach their optimal value for a chosen sample. As discussed before, SA does pick up a lesser value every now and then (defined by a probability distribution function) to try and reach a better solution. MIMIC reaches the global optimum very very quickly in this sample (roughly 800 iterations) because of the fact that the points in the subsequent iterations are chosen from the distribution obtained by the better performing points in the previous iteration. GA does particularly bad, even worse than RHC in this sample. SA in spite of not reaching a global optimum is at least on the upward trend, leading us to believe that an increase in the number of iterations would possibly let it reach the global optimum. GA shows no such promise. It seems be badly stuck at a local maxima of 50 and refuses to get any better!

I then tried to improve the performance of GA's by tweaking the population chosen, the number of cross overs and mutations allowed. With the configuration of (Population : 50, Crossovers : 48, Mutations :2), I was able to get much much better results (~70) which gives me enough reason to believe the performance may improve even more as the configuration is modified.

Lesson learnt is that MIMIC seems to perform better for problems that are much smoother. Conversely, GA seem to be a better choice for functions that change much more sharply. I also learnt that GA performance can sometimes be improved by increasing the population size and varying the other parameters (like crossovers, mutations) accordingly.

<u>Continuous Peaks Evaluation:</u>

Continuous Peaks seems to be an interesting problem. It is an extension of the 4-peaks and the 6-peaks problem. The problem expects a bit string of length N and a parameter T.
But instead of the islands of 0's/1's being only at the head or the tail, continuous peaks allows these islands to be present anywhere in the bit stream. The parameter 'p' to optimize then can be stated as:

$$p = \max[\text{island}(0, X), \text{island}(1, X)] + R(X, T)$$
$$\text{where } R(X,T) = N \textbf{ if } \max(\text{island}(0,X)) > T \text{ and } \max(\text{island}(1,X)) > T$$
$$= 0 \text{ otherwise}$$

The problem tries to maximize such islands of 0's and 1's in a particular bit string. Although not mentioned anywhere, I can imagine this to have applications in computer graphics with coloring algorithms. This is again a maximization problem and we add a reward of N if we were able to satisfy the condition of both islands of 0's as well as 1's.

For my experiment, I chose **N=60**, and **T= N/10** (i.e. 6)

Intuitions before Experiment:
Straight off, we see that there is an underlying structure to the problem which means that MIMIC should probably do well in this problem. Also, the nature of this problem seems closer to Count One's than TSP as a chosen sub solution does not affect the whole solution very drastically. Thus, I expect a not so great performance from GA's with SA and

RHC doing reasonably well. I also hope to see the problem of local maxima as N (and hence, T) chosen is not very small. I also expect the problem to get much more severe as I increase the values of T for the same N.

Experiment Setup:
The experiment setup remains the same as in TSP, as we still have to explore the same things- which algorithms perform better and what does the path to optimal value look like.
Additionally, I want to see the effects of an increased value of T on the quality of solutions. I expect this performance to go down with an increase in the value for T.

Analysis, Possible Explanations and Lessons Learnt:
Figure 1(a) displays the data for experiment 1. As expected, MIMIC does pretty well in this problem. But SA is the one which really shines in this test. Even though its average optimal value over 100 iterations is marginally lesser than MIMIC, it seems to hit the optimal value much more frequently. GA's as expectedly do not give the best performance and do better only than RHC. This seems to be a some what established pattern by now.
Reason for MIMIC's good performance in this algorithm is the same as Count One's – the structure in the solution of the problem being considered.
I explain the reason for SA's good performance as : in this case, there are many local optimum present. As a result of this, it is very easy to get struck in the local maximum thinking it's the best we can do. SA avoids this trap by sometimes choosing the points which are not necessarily the best at that point but may lead to better peaks. However, it is very clear in the figure that this can backfire too – the performance dips a lot in some of the iterations but this is expected. We can always employ hacks like running SA multiple times instead of a single time and pick the best predicted value.

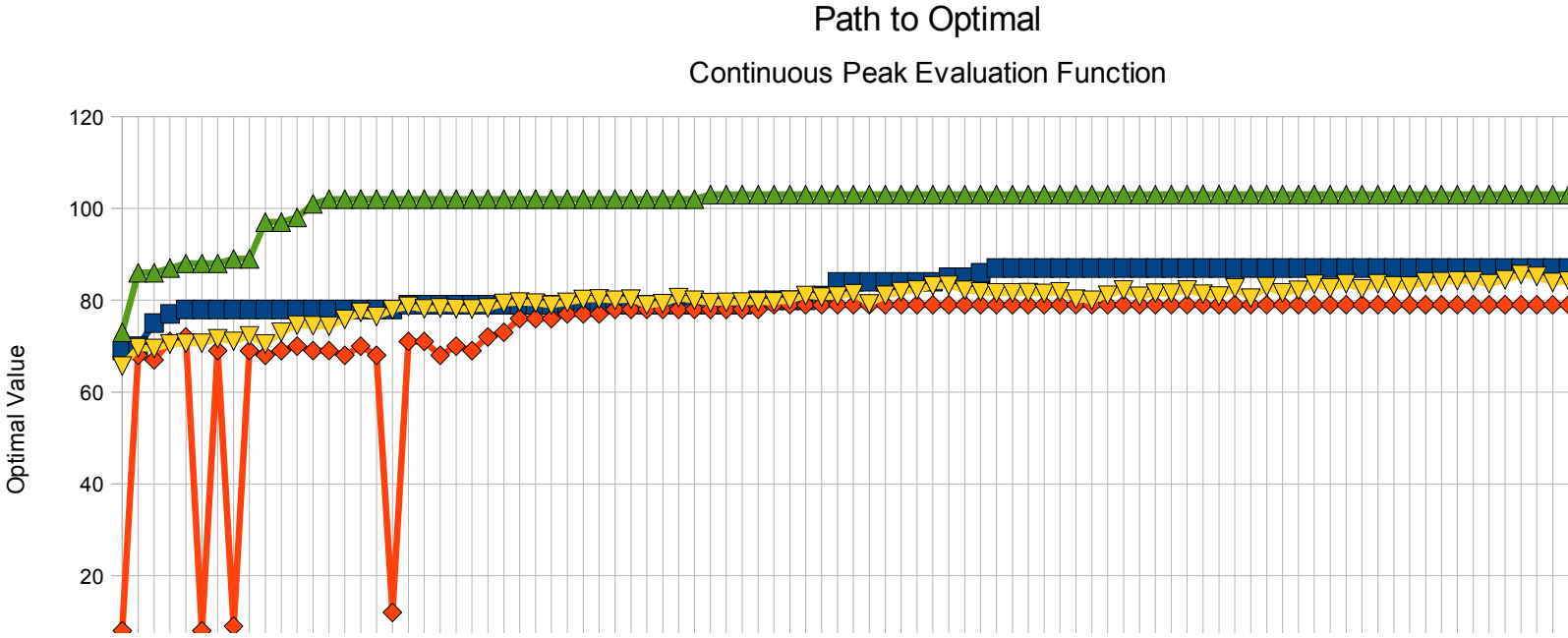


Figure 4 : Path to Optimal Comparison for Continuous Peak Evaluation for a sample run (Iteration limit = 2000 for all algorithms). N=60, T = 6

The path to optimal value can be observed in Figure 4. It is very clear how SA takes points which are much much worse than the current value (see the sharp dips in Figure 4). Even though in this particular sample, SA does not reach the global peak (since there are peak values better than it), still as we see from Figure 1(a), it does reasonably well on the whole.

Figure 5 is where a lot of things are happening. I changed the value of T from 6 to 30, which means "the basin of attraction for the inferior local maxima become larger". (Ref : Charle's handout on MIMIC). As expected, GA has landed in a sub optimal peak. SA earns the reward of picking lesser valued neighbors by climbing to what looks like a global maxima very very steeply. MIMIC yet manages to exploit the structure very very quickly, reaching the peak in far lesser iterations (750) and behaves stably after that.

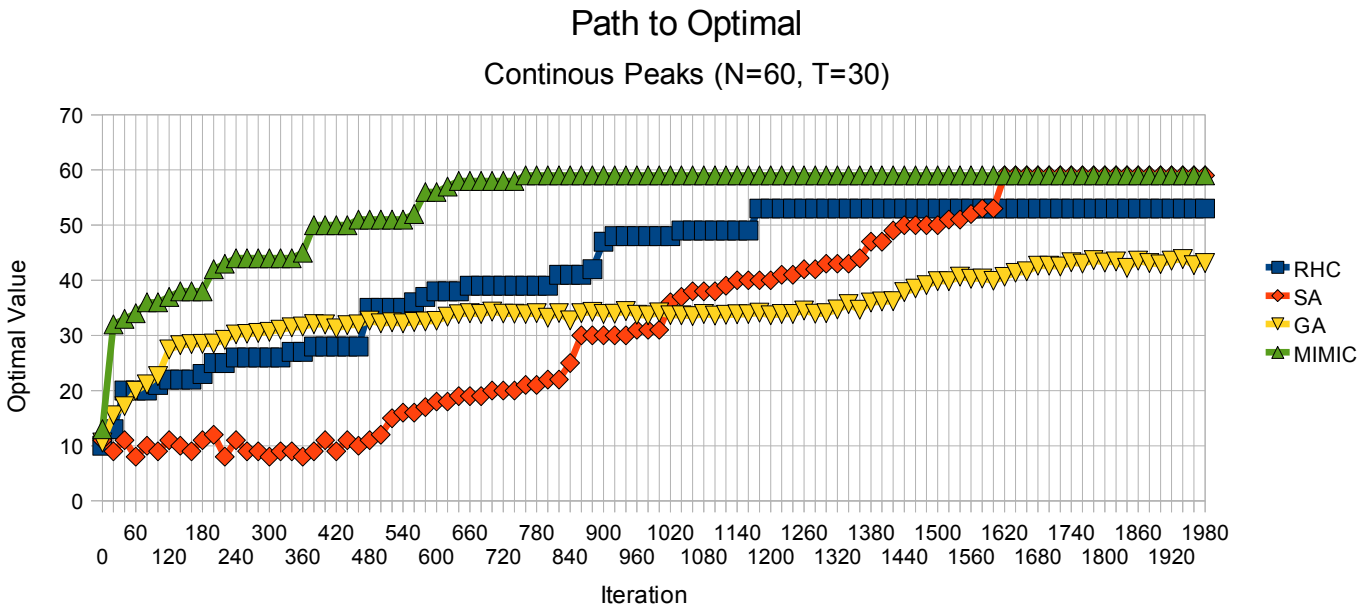


Figure 5 : Path to Optimal Comparison for Continuous Peak Evaluation for a sample run (Iteration limit = 2000 for all algorithms). N=60, T = 30

Also, I ran this particular scenario (N=60,T=30) 10 times to obtain the following results:

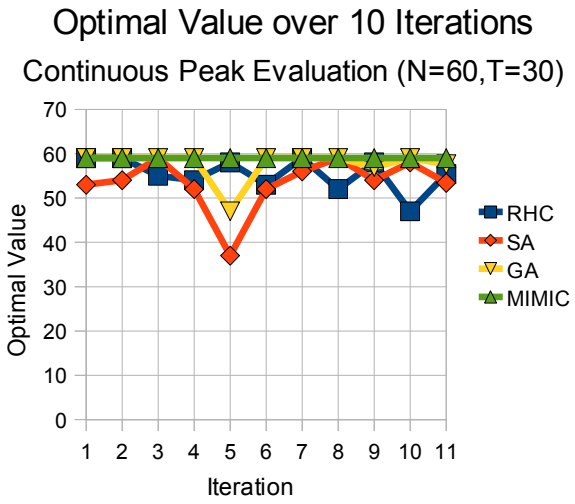


Fig 6 : Optimal values over 100 iteration for Continuous Peak Evaluation N=60, T = 30

on all the algorithms to find the averages for RHC, SA, GA and MIMIC as (55.4, 53.4, 57.6,59). Oddly enough, for this case, SA's performance is the not as great as the first time but this could be possible as the number of iterations are lower, so each bad performance hurts the average much more.

Lessons learnt in this problem include that SA is capable of doing pretty well in certain problems like this. But as we take points which are not better sometimes, it is possible to end up doing pretty badly. One idea can be, always try to run SA in iterations, before predicting the best value obtained from them. Other findings are basically reinforcements of beliefs formulated by the previous problems.

Neural Nets:

I use the spambase dataset which I used in Assignment 1. The dataset has 56(+1 classification) attributes. Classification is binary as a mail can either be spam or non spam.
The Neural Net I have implemented has just 1 hidden layer and hence, it should be compared with the results from

Assignment 1 which had just 1 hidden layer.

I also try to see what the impact of increasing the number of iterations for these algorithms is. This approach is inspired from the findings in the earlier section where I suspected that maybe an algorithm can perform better given it has enough iterations to get to the optimal value.

Worth noting is that unlike the other problems above, this is a minimization problem as we are trying to minimize the error function.

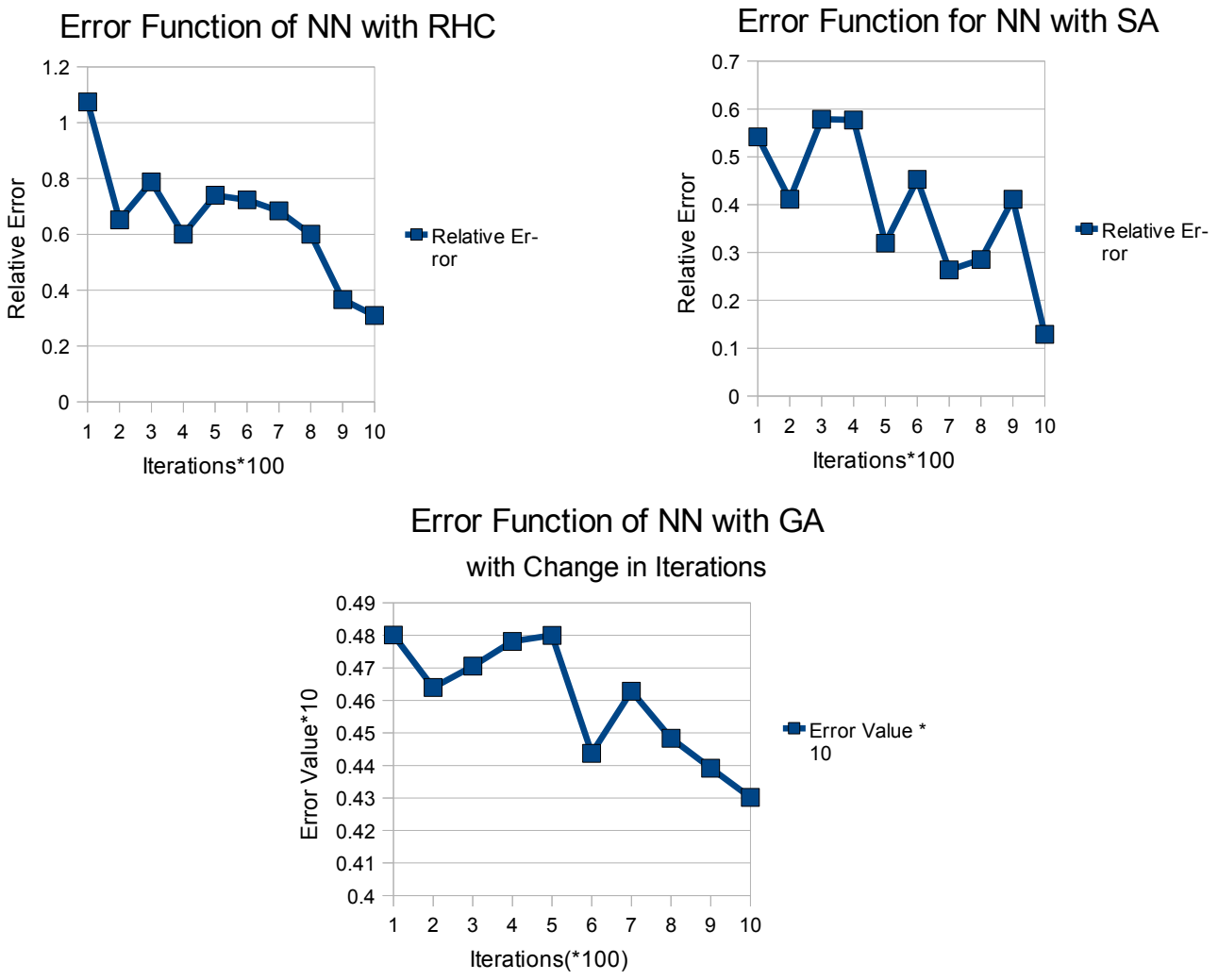


Figure 7: Variations in Error Function vs Number of maximum possible iterations

For this exercise, the weights of the neural network are initialized to random values and then we try to minimize the error function which is dependent on the difference between the predicted and the actual values, but instead of Backpropagation, I use these three algorithms.

The general trend observed is that as all these algorithms are given sufficient iterations, they tend to do well. Also worth noting is that I have given thresholds for the value of this error function. What this means is that in some of the cases, if the value of the function is less than a threshold (say, 0.01) then the function terminates irrespective of which iteration it is on. This obviously has a danger of not attaining the best value.

This is what distinguishes such real valued continuous functions from discrete valued functions like the one's mentioned above. We must define what is an acceptable error limit for us, and this can be known only through the domain knowledge.

In my observations, I see that SA manages to minimize the error function to a lower value than both RHC and GA. GA outperforms RHC though. This seems to be a consistent trend in almost all the problems I have done.

I further tried to optimize both SA and GA algorithms to get better results as the accuracy was still not better than worse case performance for Assignment 1 with 1 hidden layer (worse case 60.8967 % accuracy for a .00001 learning rate value).
For SA with 1900 iterations, I was getting an accuracy of 57.0093% while for GA with 1900 iterations, it was 59.421%.

For GA, I tried doubling the population size from 10 to 20. 10 iterations over this new configuration yielded a best case accuracy of 64.362% which is better than the worse case performance of a single layer NN with back propagation. I feel that this value should have been much higher and been comparable with the best case scenario for a single layer NN of assignment 1 (~93%).
Another possible reason can be the number of nodes in the hidden layer as well as how we choose the population makes a difference. I tried variations in this configuration too, varying the number of crossovers, mutations and randomly chosen offspring, but could not arrive at a pattern which seems to work. Some configurations yielded better accuracies while most did no better than the previous best (64,362%).

Similarly I tried to tweak SA to try and get a better performance. One strategy that came to mind was to decrease the cooling factor. Once again I saw some better values, but no patterns. I expected to get better accuracies by decreasing the value of the cooling factor. In some cases, this did give a better value, but in many cases it did not. Hence, it is difficult to see the correct pattern here.
Another possible reason for such bad performances is that I do no cross validation in this experiment.

There's a very hard lesson in this: things tend to be a lot more complicated for real valued continuous functions. What I mean by this is that in spite of having a working algorithm, continuous valued functions seem to require a lot more tweaking of the parameters and like in my case, it is often very hard to see the pattern of parameters which give these optimal values.

Conclusions:

The first section (of picking optimization problems) was a lot of fun. I was able to discover what Dr, Charles meant by saying that MIMIC exploits the structure of the solutions to optimization problems. It clearly outperforms all the other algorithms in most cases and only in TSP, where there seems to be no structure does it fare poorly. I also learnt that GA's can be reasonably alright for these cases. Although, considering their not so great performance, it is probably a good idea to run some other algorithm to verify if thats the best we can do. SA seems to be the most consistently volatile of the lot. If you notice Figures 1 (a) -(e) , it has the maximum variations. Thus, SA seems to be a reasonably good algorithm to apply provided we are willing to take measurements in multiple iterations. RHC though pretty naïve, and in spite of being the most prone to local minima stll does better than GA some times, which is not to say that GA is worse necessarily, but that a lot depends on the type of population chosen and what the optimization function looks like.

The second section (of optimization of NN's) was a much more tedious task which involved tweaking a lot of parameters leading me to believe such problems are much more complex. Also, it is pretty difficult to see the patterns in parameter variation which can guarantee better solutions. The one thing, how ever that can be said is that sufficient iterations should be given to these techniques because different algorithms have different rates of arriving to the optimum values.