Assignment 2: Randomized Optimization

Ahmad Aldabbagh

October 15, 2016

1. Introduction

This assignment has two main parts that explore random search. The first part is to find good weights for a neural network using three local random search algorithms instead of using back propagation on one of problems in assignment 1. The second part is to select three optimization problems and apply four learning algorithms to them to highlight the advantages for each algorithm.

2. Data Set of Choice

The data set I have chosen is the same set used in assignment 1 which is the 'Breast Cancer Wisconsin (Diagnostic) Data Set with 569 instances. Where it has been acquired from the UCI Machine Learning Repository and it has 30 features extracted from it. The data set has been pre-processed to match the requirements of the programming language of choice and related packages (e.g. order of features & classes), which shall be explained further in the following section. Pre-processing started with changing class labels from strings to numbers. It also included normalizing features which resulted in yielding better results. This would also be noticeable when using algorithms that leverage gradient-descent (many algorithms) to minimize the cost function resulting in a harder path to find the global minimum. Further, the used normalization/scaling was mean-normalization where the mean for a given feature (list of one kind of features) was subtracted from the feature and then the feature has been divided by the standard deviation.

3. Finding Neural Network Weights Using Randomized Optimization Algorithms

The development language that have been used is Java and MATLAB. Specifically, Java (ABAGIAL) is used for all parts except for neural networks with back propagation, where MATLAB was used in the first assignment to find the best network configuration.

For each algorithm (RHC, SA, GA), the data set have been split into training (60%) and testing (40%) data. Where testing data is not looked at till the very end. As for training data, cross validation has been used to tweak the hyper-parameters of the randomized optimizer.

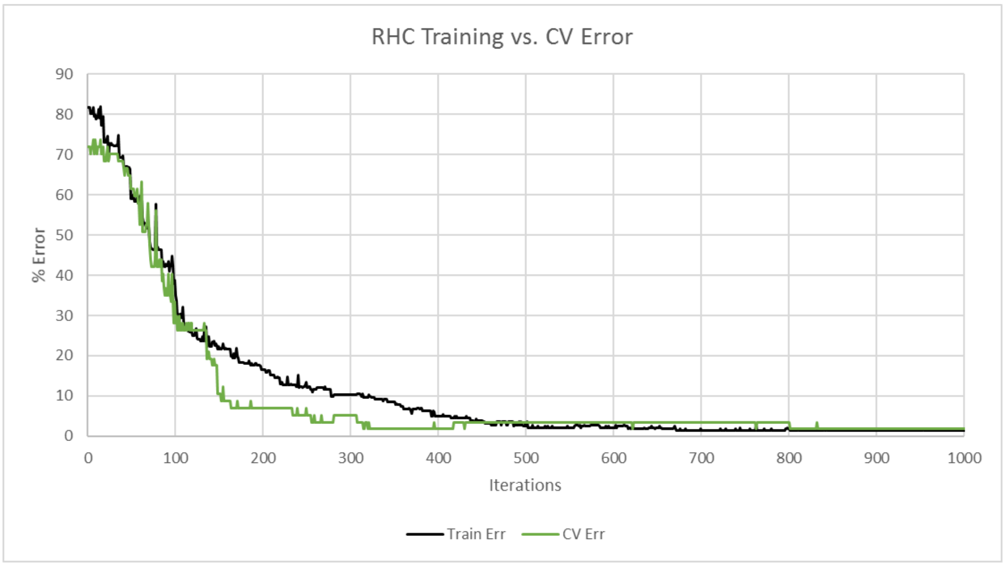
The metrics used to analyze/train neural networks included: learning curves, parameter value vs. cross validation error. As for evaluating the results, the parameters for each algorithm with lowest cross validation (CV) score have been used to test the testing set. Where score is the classification error.

3.1 Model Selection

The network setup used is based on the optimal configuration obtained in assignment 1. Which had 30 input nodes, 12 hidden nodes and 2 outputs. This configuration was fixed and a hyper parameter grid search with cross validation has been used by varying the parameters for both SA and GA (RHC does not have parameters). Also, the number of iterations was set to 3000 for the case of SA & RHC because it was found that they take longer to converge to a reasonable value. On the other hand, GA was run for 2000 iterations because it was observed that it converges to a good result in smaller number of iterations. However, each iteration of GA was slower than those for SA and RHC.

3.1.1 Random Hill Climbing (RHC)

For RHC, the CV and training errors were observed for different iterations to see the effect of number of iterations on the rate of convergence. The lowest CV error was 1.754% at 801 iterations. The graph below shows the CV and training errors.



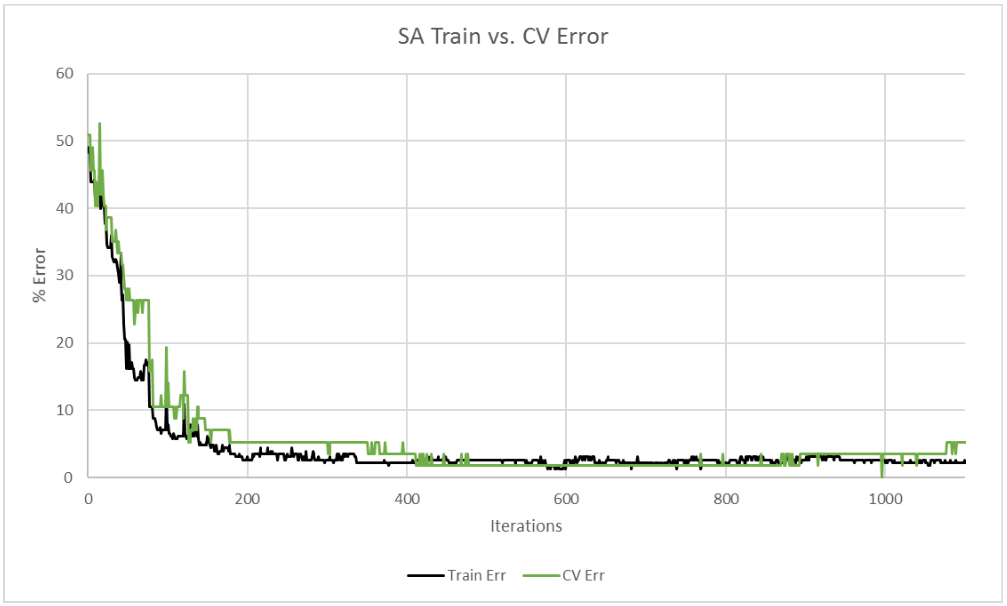
The average accuracy for this network configuration was 96% with a training time of around 2.182 s.

Another aspect examined was the **runtime** with respect to number of iterations which was found to be **linear**.

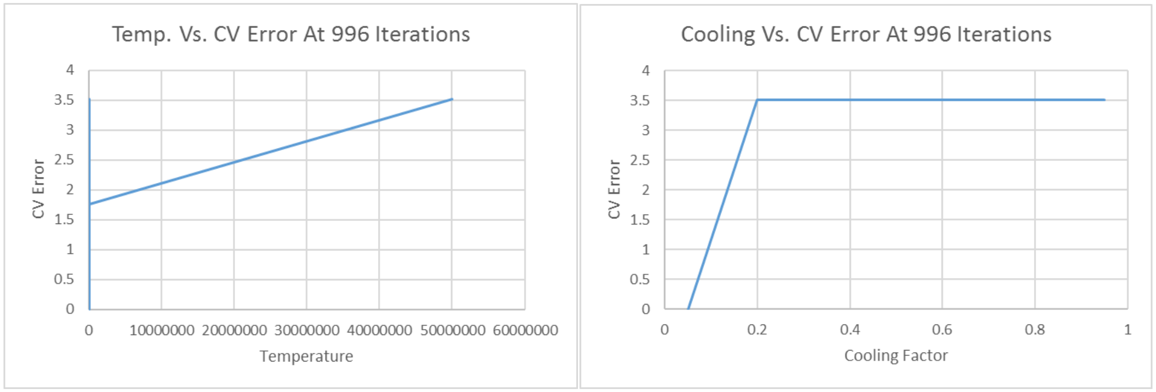
Since RHC starts at random locations without a temperature parameter, it has a tendency to get stuck in local minima so the algorithm was run multiple times to see how well it performs in regards to accuracy: (96+95.6+96.491+97.1)/4=96.23% and average number of iterations: (801+1700+1189+574)/4 = 1066 iterations. A possible explanation is that this problem doesn't have many local minima and only one optima so the initial location of the learner doesn't affect the results that much.

3.1.2 Simulated Annealing (SA)

For SA, T was varied between 5 and 50000000 & cooling was varied between 0.05 and 1.0 while measuring the test error and cross validation error for each iteration. The best combination found with smallest T value was T=5 and Cooling=0.05 at iteration 996, resulting in CV error of 0.0%. The figure below shows the learning curve for this combination.



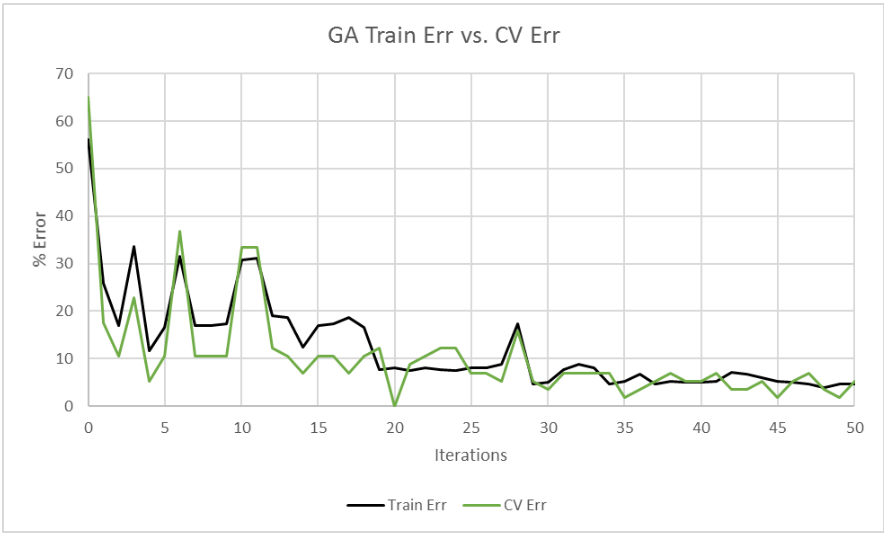
The average accuracy for this network configuration was 97% with a training time of around 3.411 s. It was observed that beyond some point CV error starts increasing and overfitting occurred. Furthermore, to understand the effect cooling and T on SA, they were both varied while fixing the other parameters as obtained from best model.



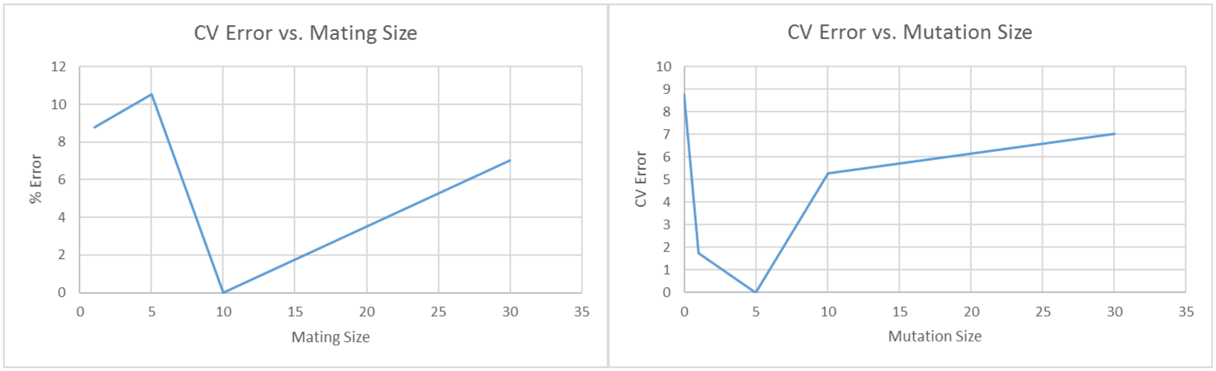
These results were quite expected, it can be seen from the graph on the left that as T increases, it will take more iterations to reach a smaller of CV. Also, as the cooling factor increases, more iterations are needed to reach a smaller error. For example, when T is large (50000000) and Cooling is fixed at 0.05, it takes 2091 iterations to obtain a CV score of 0.0% as opposed to 996 iterations. A note worth mentioning is that even if more iterations are needed (e.g. more complex models), SA is still quite fast in running through iterations as opposed to GA. The same applies to RHC in comparison to GA. **Running times** were also found to be **linear** for SA similar to RHC.

3.1.3 Genetic Algorithm (GA)

For GA, the parameters have been varied as follows: the population size has been varied from 1 to 200, the size to mate from 1 to 100 and the size to mutate from 0 to 100. Following the same approach for the SA, the model with lowest average cross validation error and least complex parameters, as well as number of iteration has been chosen, Popsize = 50, toMate = 10, toMutate=5. Resulting in a CV error of 0.0%, after 20 iterations with runtime of 0.219 s. The test error was around 5%. It is important to note, that other parameter combinations also resulted in a similar CV error but they were avoided as they were larger and required more iterations to converge. The figure below shows the learning curve for the selected parameters.



Furthermore, to understand the effects of the mating size as a percentage of the population size as well as the mutation size as a percentage size both have been varied over the optimum parameters (keeping all parameters constant and only changing one parameter) the following results have been obtained.



As seen in the graphs above, the optimum mating size as part of population size was found to be 20% of the population size. As for mutation size, 10% of the population size yielded the lowest CV error. Although, GA converged in less iterations than both RHC and SA, the average iteration time for GA is much higher than that of RHC and SA.

Runtimes times with respect to number of iterations were also found to have a **linear** relationship.

3.2 Feature Selection

Since the number of features is fairly high (30 features) for a data set with 569 instances, I thought it would be interesting to explore how reducing the number of dimensions would alter the performance, runtimes and model selection for the problem. I have decided to reduce the number of features to 4 using sklearn's extraTreesClassifier.

The following is a summary of the results for each of the 3 algorithms based on the best parameters that have been determined using cross validation in the previous sections.

Algorithm	Best Parameters.	CV Err.	Average Test Error	Iterations	Training Time
RHC	N/A	0.0 %	6.14 %	808	2.01 s
SA	T=5; RT=0.05	0.0 %	6.14 %	630	1.18 s
GA	Pop=50; mate=10; mutate=5	1.76 %	8.20 %	286	2.09 s

It can be seen from above that the overall accuracy of the learners has decreased a bit and this could be due to the fact that some information has been lost when we got rid of the learners. Also, runtimes for algorithms/iterations has decreased in comparison to the case where all 30 features were used. This was specifically noted for the Genetic Algorithm where the runtime per iteration decreased from 0.01095 s to 0.007 s (around 40% reduction in runtime vs. 86% reduction in the dimension of the input or input layer of the neural network).

This would be quite noticeable if the data set was really large and if the neural network was more complex (e.g. has more hidden layers) as well as the nature of the problem (e.g. having many local minima).

3.3 Summary

Overall, all three algorithms performed well in optimizing the weights for the neural network. A possible reason could be that the problem is fairly simple and does not have many local minima. The table below shows a summary of the obtained results as compared to backpropagation.

Algorithm	Average Test Error	Iterations	Training Time	Testing Time
RHC	3.70 %	1000	2.20 s	0.001 s
SA	3.40 %	1050	3.50 s	0.001 s
GA	5.00 %	35	0.36 s	0.001 s
Backpropagation	2.37 %	25	0.001 s	0.001 s

Even though these algorithms provided positive results, they were still outperformed by backpropagation and this is quite understandable since the problem seems to be smooth with a global optimum which can easily be found by gradient descent. Another point worth mentioning regarding GA, is that it might have converged quicker than RHC and SA in this case due to the simplicity of the problem. However, this won't be the expected result if the problem had many more instances with a more complicated neural network configuration.

4. Optimization Problems

The purpose of this section is to use four optimization algorithms (RHC, SA, GA, MIMIC) to find global maximum values for three different problems where each problem should leverage one algorithm over the others. All these problems were implemented based on ABAGAIL's representation of them. For all problems, values have been changed to see the effect of changing each algorithm's parameters on the problem at hand. However, when running the final experiments I decided to go with the provided parameters as I was able to use them to make my point.

4.1 Continuous Peaks Problem

Is an extension of the four peaks problem as mentioned in [4], where four peaks has two suboptimal local optima with fitness of 100 in case of a class of fitness functions defined on 100 bit strings. It is described by fitness function **f(x) = MAX(o(x), z(x))+reward**

where:

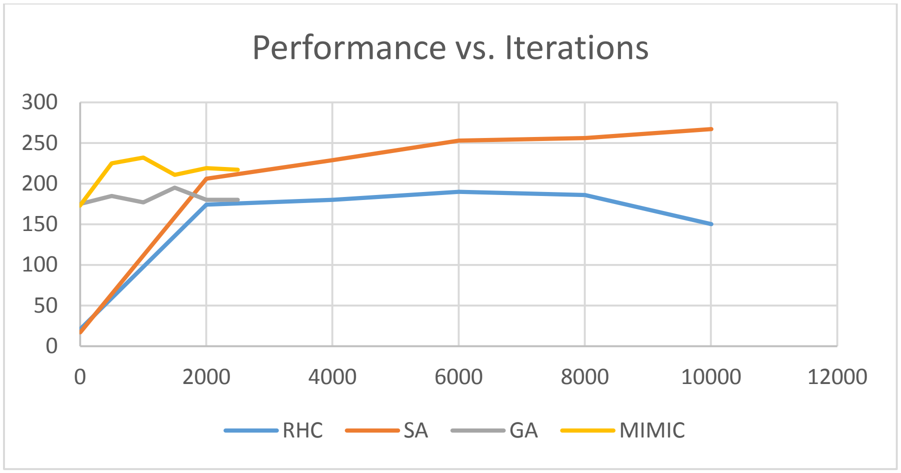
o(x) is the number of touching ones starting in the first position

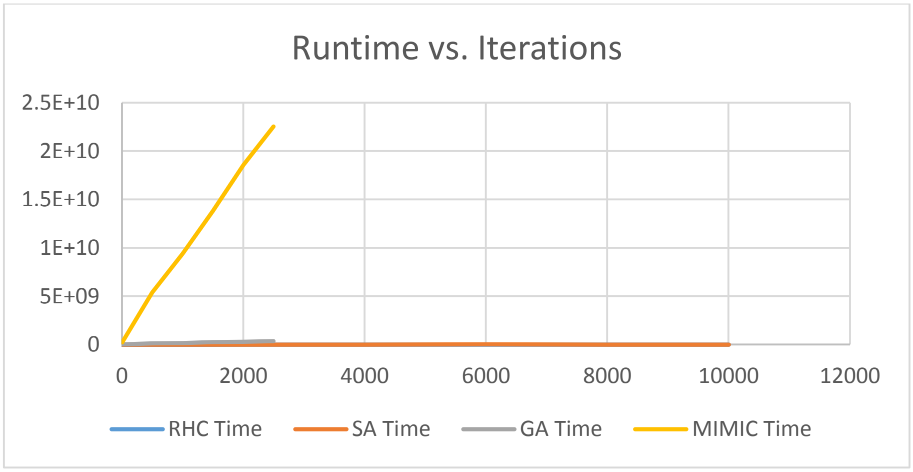
z(x) is the number of touching zeros ending in the 100th position

reward = (100 if both o(x) & z(x) are above a value T; 0 otherwise)

The reason I chose this problem is that it has multiple local minimas so I expect purely random algorithms like RHC to not perform well. Also GA might not perform well if the mutation probability was small since mutation helps GA not to get stuck in a local minma but if mutation probability grows too high, GA will just exhibit a random behavior and even if that did not happen, it along with MIMIC will probably take more time than SA to converge to the same result.

The problem was ran for N=150 and T=15 to compare the different algorithms where RHC and SA were allowed to run for 12,000 iterations and GA and MIMIC were run for 2500 iterations as they are expected to converge with fewer iterations. The reason this constraint was put on MIMIC and GA is because SA and RHC run much faster so they can afford to run in multiple iteration while having a run time lower than GA and MIMIC. So although SA needs more iterations to converge, it still does that in a way faster runtime than all the others.





RHC is expected to not perform as good as others here due to the multiple local minima. And it is evident that SA is better than all other algorithms given the runtime constraint or bound. Also, it can be seen beyond 2000 iterations mimic overtaken by SA. The difference in runtime for mimic with 2500 iterations and SA with 12000 is that MIMIC runtime takes more than 2250 times SA's runtime and still doesn't perform as good. GA is also similar in the sense that its runtime is 36 times of SA without performing as good.

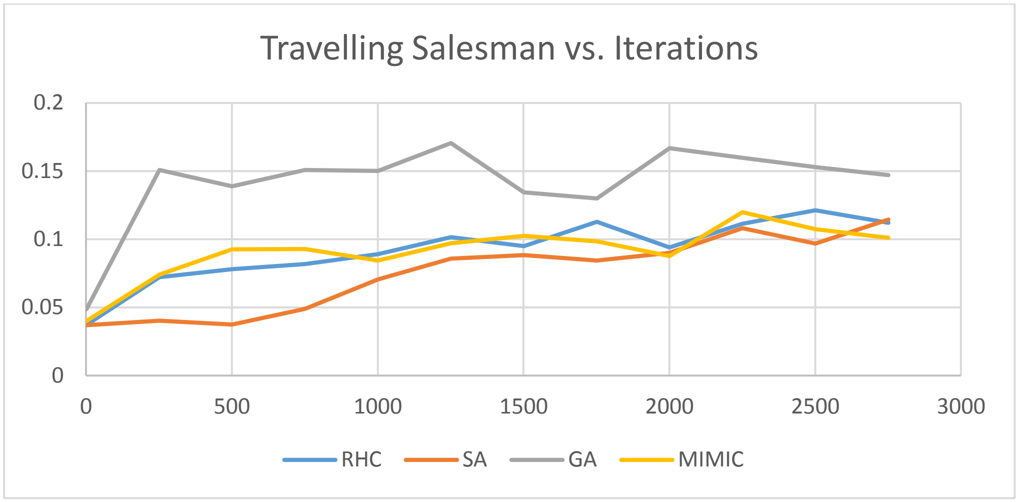
4.2 Travelling Salesman Problem

This is a well known optimization problem in computer science. It can be thought of a graph representing a route that connects multiple cities together and the goal to visit all cities using the shortest distance possible. The fitness function to maximize is 1/totalDistanceTravelled.

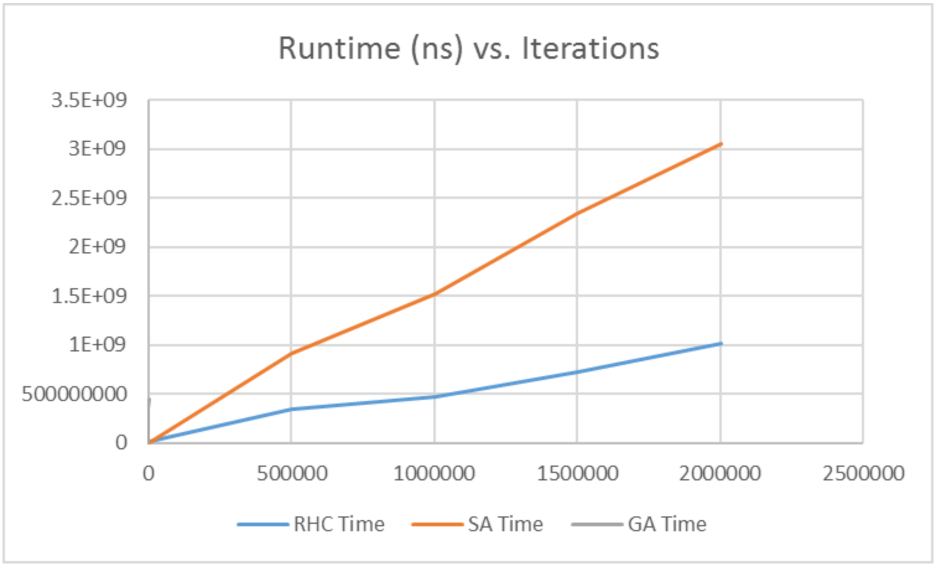
The reason I chose this problem was due to the nature of GA in solving the problem as which was expected to perform better than others will be explained further.

The parameter that was chosen for this problem is N = 50 which is used to create N random points that can be thought of as cities on a map.

In this problem GA outperforms all other algorithms, where MIMIC really falls behind. GA converges to 0.16 in this example much faster than all other algorithms.



As a matter of fact, RHC and SA were allowed to run 1000 folds more iterations than GA with 2 times and 6 times the runtime respectively. Yet, they were not able to reach the optimum value reached by GA. This could be due to the nature of GA in solving this problem where next generations get the best attribute (shortest paths/shortcuts) from the parents so it converges much faster and with better results in each iteration.



RHC doesn't do as well as GA since it is prone to being stuck in local maxima. As for SA, after a large number if iterations, it just behaves like RHC where it might get stuck eventually and even if that did not happen, it just becomes more expensive. Finally, MIMIC simply takes much longer time in this problem.

Similar results have been obtained by varying the size of the problem.

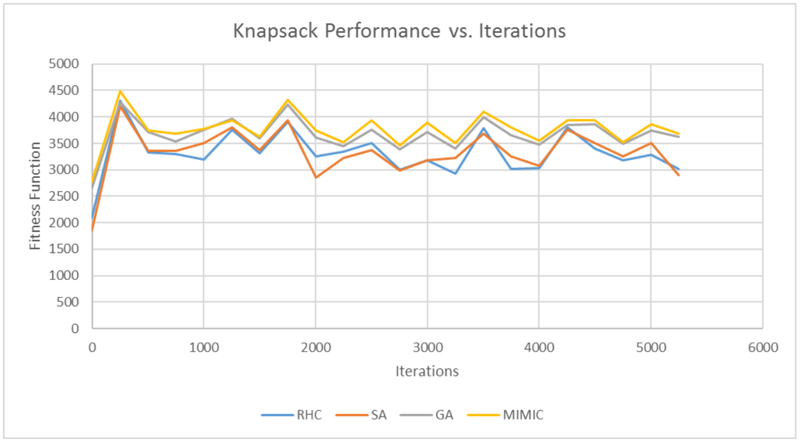
4.3 Knapsack Problem

This problem aims to maximize a value parameter given a weight constraint. Given N elements each with weight (w) and value (v) we want to maximize the total value of all N elements while keeping the sum of weights below a certain level.

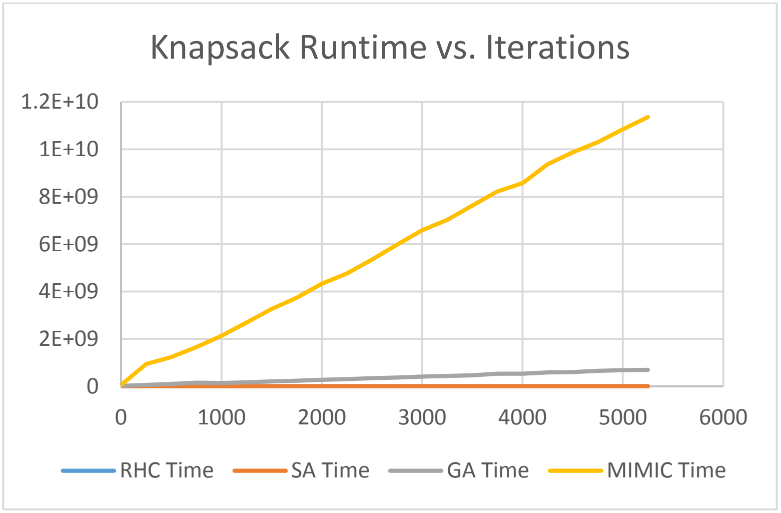
This problem was interesting because it tries to maximize one value while being constraint by another and it was chosen to show how MIMIC, although expensive in runtime, can be used to solve such problems and when it should be used.

The parameters used here were numItems =40 with 4 copies each. And the maximum volume and weight value is 50.

It was observed that all four algorithms reach some sort of a steady state with some fluctuations where the upper bound is set by MIMIC.



Therefore, if MIMIC were to terminate at a low number of iterations, thus having a smaller runtime, and other algorithms kept on running till their runtime reached that of MIMIC, they will still not be able to outperform it. This is an example of a problem where an expensive algorithm like MIMIC is justifiable to use over the other algorithms.



A possible explanation of why MIMIC might have performed better [5] than others in this particular problem is that we have multiple parameters that interact with other and the more we explore the Optimization landscape, the better we understand which parameters are related and which aren't. This kind of global view of the problem, is something that gives MIMIC has an advantage over others that only have a local view.

References

[1] https://archive.ics.uci.edu/ml/index.html
[2] https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29
[3] http://www.ri.cmu.edu/pub˙files/pub1/baluja˙shumeet˙1997˙1/baluja˙shumeet˙1997˙1.pdf
[4] https://pdfs.semanticscholar.org/055a/141d22f56612c0120a0bb7fd24114eb08ca9.pdf
[5] http://www.cc.gatech.edu/˜isbell/tutorials/mimic-tutorial.pdf