Havish Chennamraj
903201642
CS 7641 – Assignment 2 Report

Abstract

This report explores four local random search algorithms namely (1) Randomized Hill Climbing, (2) Simulated Annealing, (3) Genetic Algorithms & (4) MIMIC. The first part of the report evaluates the performance of Neural Network on 'Ford Dataset' in an experimental setup where the best weights are found by applying each of the first 3 optimization algorithms. The second part of the report evaluates the performance of all four algorithms under 3 "optimization problem" setups namely Travelling Salesman, Knapsack and Continuous Peaks.

Algorithms

Randomized Hill Climbing (RHC)

Randomized Hill Climbing attempts to find the optimum value by evaluating the fitness score (the function the algorithm is trying to optimize) of the neighbors and moving towards the neighbor with the highest score until it reaches a peak. It is important to note that RHC doesn't always converge on the global optimum because of its greedy approach. In order to help with this, the algorithm is repeated for a number of iterations with the starting point for every iteration chosen at random.

Simulated Annealing (SA)

Contrary to RHC, Simulated Annealing tries to 'explore' the dataset before exploiting it with same approach used by RHC. SA draws its name from annealing in metallurgy, a technique where a metal is heated and cooled to improve its attributes and reduce defects. At every step the algorithm finds the best neighbor and moves towards it if it has a higher fitness score than the current one. If not, the algorithm attempts to move to the new point based on a probability function which is directly proportional to temperature 'T'. At every step, this temperature T decreases based on a cooling rate 'e'. To summarize, at high 'T's the algorithm tends to be more explorative and behaves like RHC when 'T' tends to 0. Also, lower cooling rate allows the temperature to decrease gradually thus allowing the algorithm to explore more and vice versa.

Genetic Algorithms (GA)

The class of Genetic Algorithms draws its inspiration from the process of natural selection. Every generation (which is analogous to an iteration), the algorithm selects individuals (candidate solutions) from the population to generate offspring using crossover (functionally combining two candidate solutions) and mutation (functionally modifying part of the offspring). The selected individuals and their children constitute the population for the next iteration.

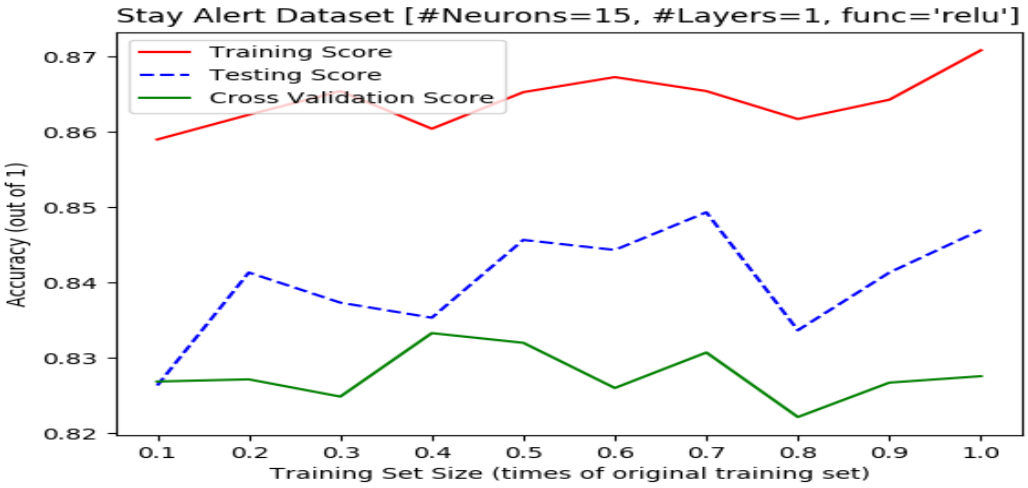
Mutual Information Maximizing Input Clustering (MIMIC)

MIMIC works along the same lines as a traditional genetic algorithm. It attempts to communicate information it has learnt about the cost function in one iteration to the successive one. It proceeds to find the best optimal solution by estimating the distribution of the underlying problem space, and iteratively eliminating sub-optimal regions to arrive at the best solution.

Part 1: Evaluation of the randomized optimization algorithms on Ford Dataset

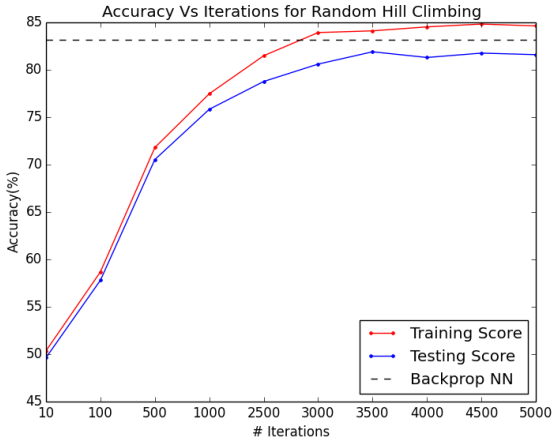
Dataset: Stay Alert! dataset from The Ford Challenge [1]

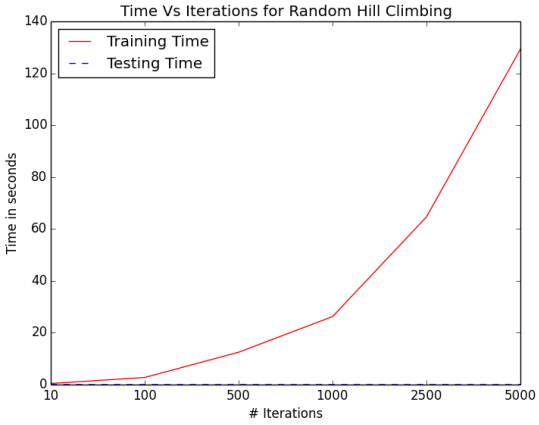
The dataset consists of results of a number of "trials", each one representing about 2 minutes of sequential data that are recorded every 100 ms during a driving session on the road or in a driving simulator. The trials are samples from some 100 drivers of both genders, and of different ages and ethnic backgrounds. Every sample in the dataset has 30 features which are continuous values representing physiological, environmental and vehicular data. It's also got a binary variable "IsAlert" around which the classification problem is built. The training dataset has around 600k records and the test dataset has around 121k records to work with. Please note that in the interest of time, the dataset has been heavily subsampled.



The above graph is the best performing NN model from the Supervised Learning Assignment. The network has **1 hidden layer with 15 neurons** in it and uses back propagation for learning and achieves the highest testing accuracy of ~83.12% on the same dataset and takes around ~13 seconds to train using Backpropagation. Please note that for all the experiments performed in this section, the NN shall have the same hyper-parameters.

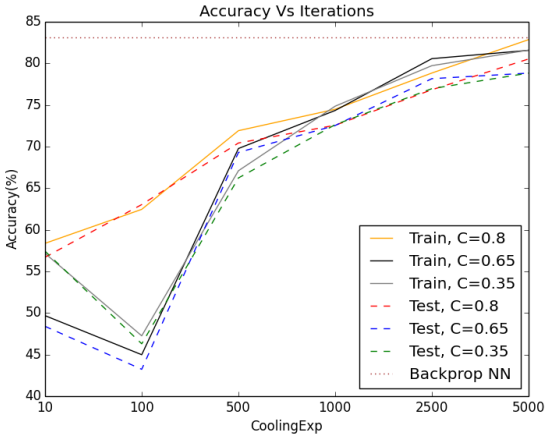
Random Hill Climbing

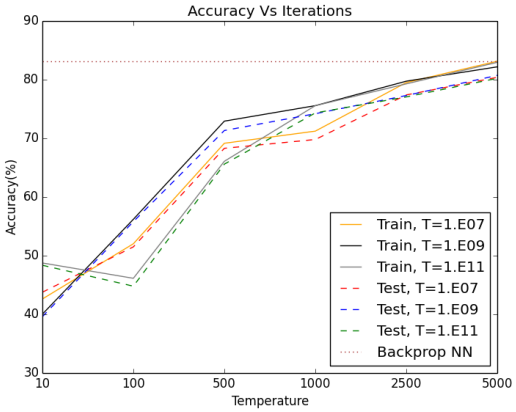




The above graph shows the performance of NN on the "Ford" Dataset against the number of iterations Randomized Hill Climbing was allowed to run. The highest test accuracy achieved is ~81.9% at 3500 iterations. However, both the training and testing accuracy seem to stabilize beyond 2500 iterations. Since we know for a fact that weights of the NN can trained better, it can be inferred that RHC is always converging on a local optimum even with increased iterations. Taking into consideration the fact that "Ford" dataset has 30 features, and is being modelled using an underlying Neural network, it is safe to assume that the search space is enormous with respect to the ground covered by neighborhood function thus making it difficult to land on the Global optimum. The fact that all the iterations beyond 2500 converged on very similar optimums suggests that the optimization problem presented by the dataset has numerous very similar local maxima than span across most of the search space. Thus, even with several random restarts, the starting point ends up in a basin of attraction to the local optimum.

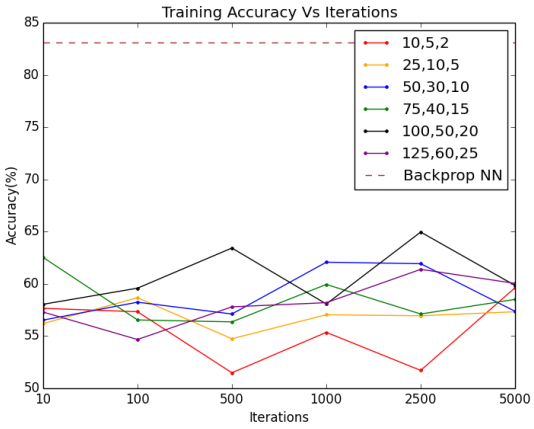
Simulated Annealing

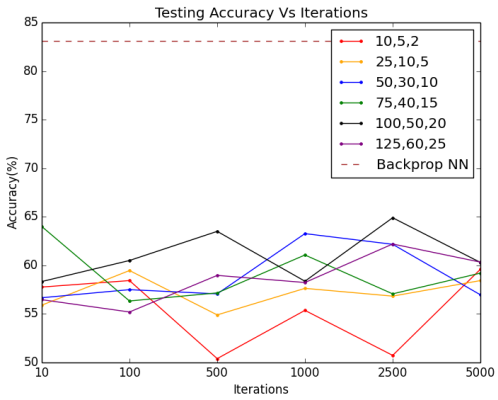




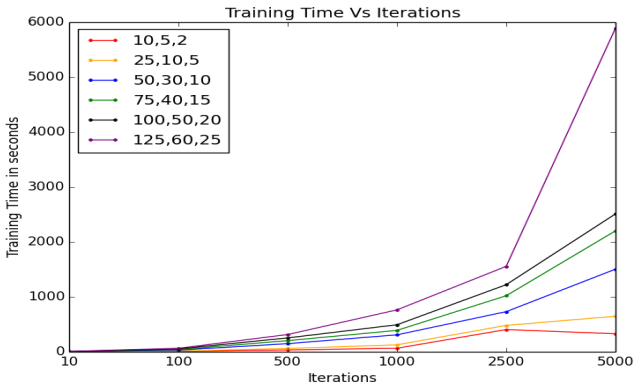
The above graphs show the performance of NN on the "Ford" Dataset against the number of iterations Simulated Annealing was allowed to run for various values of cooling factor [0.35, 0.65, 0.8] and Temperature [1.0E7, 1.0E9, 1.0E11] to see if the algorithm behaves as expected. Ideally, we expect better results with higher starting temperature 'T' and gradual cooling of temperature 'T' because the algorithm gets more opportunity to explore the search space with higher T and doesn't overshoot over optimal solutions if the cooling is gradual. As expected, the training and testing scores are better for C=0.8 (A cooling factor closer to 1 leads to a much gradual temperature decrease) and also for higher temperature T=1.0E11, ~80.5% in both cases. Also, unlike before, the scores don't seem to converge to a particular value but tend to increase linearly, albeit slightly in both the cases. The performance of SA is very comparable to RHC: it is also getting stuck at sub-optimal regions because of the nature of problem space. However, the explorative characteristic of SA seems to be working hence the slight upward trend at the end. This tells us that if the algorithm is granted more time to explore the search space, it is very likely to converge on a better optimum.

Genetic Algorithms





The above graphs show the performance of NN on the "Ford" Dataset against the number of iterations Genetic Algorithm was allowed to run for various tuples of (S, MA, MU) where S – Population Size, MA – Number to mate per iteration & MU – number to mutate per generation. As we can notice, changing the (S, MA, MU) parameters or the iterations doesn't seem to have any effect on the performance of the algorithm, and can be explained as 'erratic' at best. Noticing the trends in the graph, it looks like increasing the iterations wouldn't do much either. It performs worse than SA & RHC by a sizeable margin as well. Before we analyze the performance any further, let's take a look at the training times for the same combination of parameters (S, MA, MU).



Increasing the population size seems to decrease the speed at which the algorithm converges. A possible explanation or this is as follows: Larger population has a wide variety of genes are compared to the smaller population, which also increases the likelihood of the presence of suboptimal genes that hamper optimality of the potential solution set. Such genes are eliminated much faster in the smaller populations as compared to larger ones.

Changing the mutation rate MU is similar to altering the momentum of back-propagation. Increasing the number of samples to mutate every generation obviously increases the time to converge, but also increases the chances of finding a better solution. It should be noted that that if MU is set too high, it will skip over the global optima, and if set too low, it'll not escape a local optimum.

Changing the mating rate MA has a very similar effect. Higher MA grants the algorithm opportunity to check a more diverse population (increases the scope in search space) thus providing a better chance for finding the best solution. And just like MU, if set too high, it might overshoot and if set too low, gets stuck in a sub-optimal region.

The 'Training Time Vs Iteration' graph seems to agree with the above proposition, but none of the combinations of (S, MA, MU) yield an optimum which is substantially better than the rest. Instead, they are more or less comparable. One explanation I can think of is that the algorithm is struggling to model the problem space. In other words, the algorithm's definition of 'best' candidates doesn't seem the adhere to the problem context Consequently, every iteration generates a sub-optimal solution set even with varied population sizes, mutation and mating rates. Considering that the training time also is a big step up from RH and SA, genetic algorithm is the worst choice as an optimization algorithm for training NN on "Ford" dataset.

Conclusion

	Training Accuracy (Best)	**Testing Accuracy (Best)**	**Average Training Time**
RHC	84.83	81.90	39.30
SA	83.20	80.53	123.79
GA	64.96	64.90	2132.00
Backpropagation	87.76	83.13	13.01

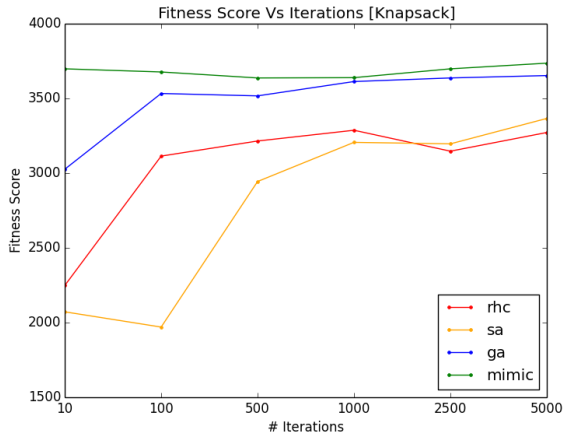
Clearly none of the RO Algorithms outperformed Backpropagation. Backpropagation tends to feed information back into the network after every iteration based on the motive of minimizing squared error enabling it to tune its weights in the direction of improved accuracy. There is no such feedback mechanism in Randomized Optimization Algorithms. The NN weights tend to end up in the local optima, hence it is essential to start off with small random values so that more of the search space is explored. It is quite evident that my RHC was ending up in a sub-optimal solution space because the accuracy plateaued after certain number of iterations. Because of the complexity of the dataset, there seems to be no structure or locality among the data points thus rendering the mutation and cross over strategies of GA ineffective and SA's performance and behavior is comparable to RHC, thus making none of them better than back propagation. However, SA seems to show some potential: there is a linear increase in accuracy, albeit very subtle, with increasing iterations which might indicate that given enough time it might find solutions comparable to that of Backpropagation if not better.

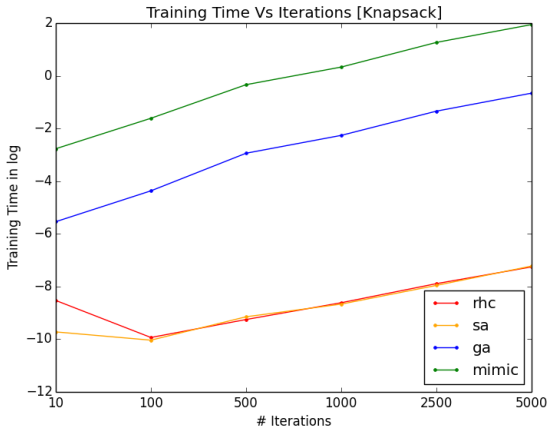
Part 2: Optimization Problems

Knapsack (MIMIC)

Knapsack problem can be defined as follows: Given a knapsack of capacity 'W' and 'N' items of weight $\mathbf{w_i}$ and value $\mathbf{v_i}$, the objective is to fill up the knapsack such that total value of all the items in it is maximum. For the purpose of this problem, I've used the library implementation of bounded knapsack provided by ABAGAIL with the following parameters.

KNAPSACK_VOLUME: 3200
NUM_ITEMS: 40
COPIES_EACH: 4
MAX_WEIGHT (of an item): 50
MAX_VOLUME (of an item): 50





The knapsack is an NP-Hard problem, thus applying randomized optimization algorithms is the best thing we can do to get close to a solution. As we can notice, **MIMIC consistently**

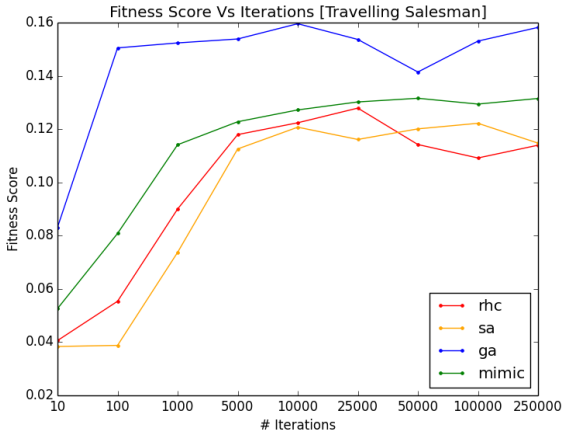
outperforms other algorithms but also takes up lot more time to train. Longer training time comes as a tradeoff for MIMIC trying to develop structure for the search space by estimating probability density functions. In other words, it attempts to learn something every iteration about the underlying distribution.

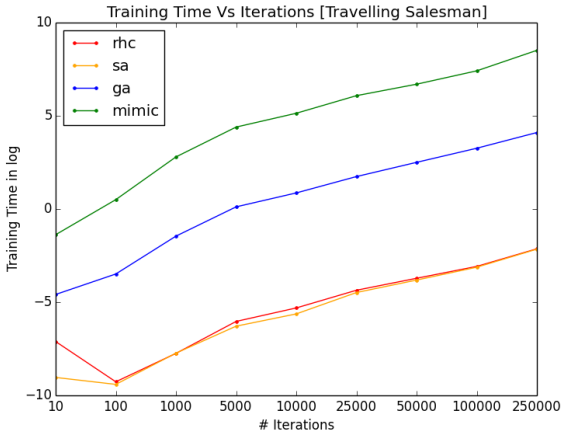
Note that search space for the above knapsack problem is very huge considering it involves selecting from 40 different items each with real-valued attributes. This makes it all the more likely for the search space to have a lot of basins of attraction for the local-optima which explains why RHC and SA perform poorly. However, MIMIC learns about the underlying distribution to eliminate areas of lesser potential (for finding a solution) and focuses on the higher potential ones refining the solution set at every iteration thus outperforming other algorithms.

Increasing iterations wouldn't necessarily guarantee better performance for RHC & SA, because they would continue to end up basins of attraction to local optima even with increased random restarts. We need a solution like MIMIC which gets better with every iteration by learning the underlying distribution and eliminating search spaces with lead to mediocre local optima. Having said all of the above, I'd also like to state that the results are a tad bit surprising because MIMIC and GA tend to perform better for continuous functions (which the above problem isn't) because it is easier for the algorithms to extrapolate from the previous generation every iteration.

Travelling Salesman Problem (Genetic Algorithms)

The Travelling Salesman is also a NP-Hard problem with the following definition: Given 'V' cities and 'E' which is a set of distances (between pairs of connected cities), the objective is find the shortest path to visit each city and return back to the starting point. It is easy to visualize this problem as an undirected graph where nodes are the cities with edges connecting them whose weight reflects the distance between the pair of cities.



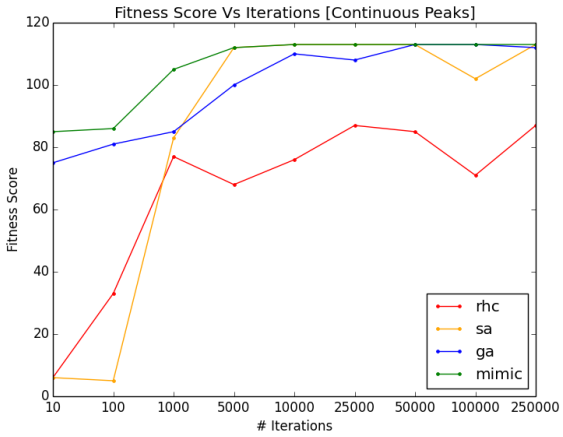


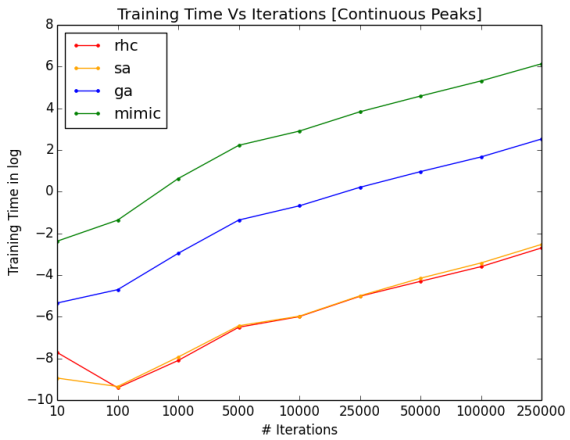
As we can notice, **Genetic Algorithm outperforms the other 3 while MIMIC comes as a close second.** Then goes on to show that iteratively improving the solution set leads to better results. Genetic Algorithms works with a set of potential solutions every iteration and makes them better by selectively picking only the best solutions based on some heuristic and then using the selected

ones to generate further solutions with crossover and mutation. This refines the potential solution set every iteration eventually leading to the one the closest to global maximum if done right. On the other hand, SA & RHC deal with only one solution at a time and wouldn't perform well in a complex hypothesis place such as this because of the existence of large number of local optima. I suspect the reason being same for the slightly worse performance of MIMIC. The complex search space probably made it very difficult for MIMIC to learn the underlying distribution correctly within the iterations the experiment ran for. We can clearly tell MIMIC is struggling at least computationally to learn the problem by looking at its high training time.

Continuous Peaks (Simulated Annealing)

Continuous Peaks problem is basically to find a string with max contiguous 0s or 1s. The problem setup is relatively simple and obviously rewards strings with more contiguous ones or zeroes.





As we can notice Simulated Annealing, Genetic Algorithm & MIMIC perform almost the same. However, SA achieves the same performance in training time orders of magnitude lesser than GA and MIMIC as inferred from the 'Training Time Vs Iterations' graph (note the time values are logarithmic).

The training times can be easily explained: As established before the cost function is realitively simple and computationally cheap, hence the very low training times of RHC and SA. MIMIC and GA involve generating samples every iteration based on probability distribution function and crossover respectively which are computationally more expensive and simply evaluating a simple fitness function.

Simulated annealing works best in small discrete space where neighbors and local optima are close by. Neighbors in this problem are defined as any two bit-sequences which are different from each other at exactly one bit position. This might seem ideal for RHC as well, but it performs poorly because of its unwillingness to explore the search space at the cost of moving to 'poorer' neighbors. I suspect that RHC is getting stuck at sub-optimal bit sequences where changing any of the bits wouldn't result in a better solution. On the other hand, with finely tuned Temperature T and cooling factor 'C', SA is able to jump through such sub optimal plateaus and

basins of attraction. Although MIMIC and GA perform comparable to SA, they are much slower than SA thus making **SA the choice of optimization algorithm for this problem.**

Conclusion

The experiments above highlighted the strengths and weaknesses of various optimization algorithms. From the results of the experiments, it is safe to assume that some problem domains are more suited for certain algorithms thus making it useful to have a knowledge of the domain and structure of the solution space while choosing which algorithm to use. If there is no structure to the underlying distribution, and the problem space isn't very complex, you're better off choosing RHC/SA since random is the best you can do is such cases and they converge much quicker. If there is a defined underlying distribution that can be characterized or some structure to the solution space, MIMIC would do a pretty good job by virtue of eliminating sub-optimal regions and focusing on high potential regions for the best solution. If there is some degree of locality to the attributes in the problem space, genetic algorithms might do the trick.

References

[1] https://www.kaggle.com/c/stayalert/data

[2] Libraries for implementing Randomized Hill Climbing, Simulated Annealing, Genetic Algorithms and MIMIC: https://github.com/pushkar/ABAGAIL

[3] The source code for the experiments has been referenced from here