Assignment 2

I have used Abagail 's help for implementing the Randomized Optimization functions.

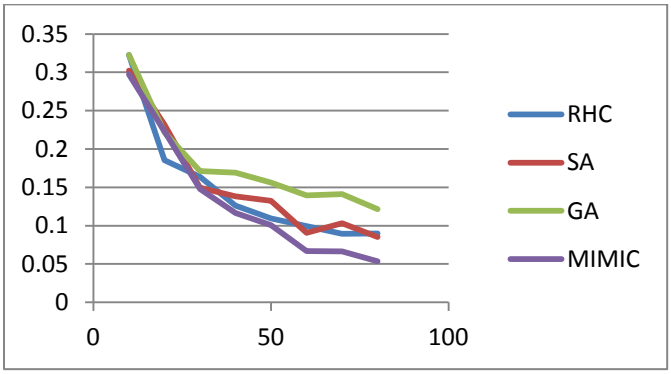
Travelling Salesman Problem

Problem Description

Given a list of 'n' cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city? It is an NP-Hard Problem. The brute force exact algorithm would have a time complexity of O(n!).

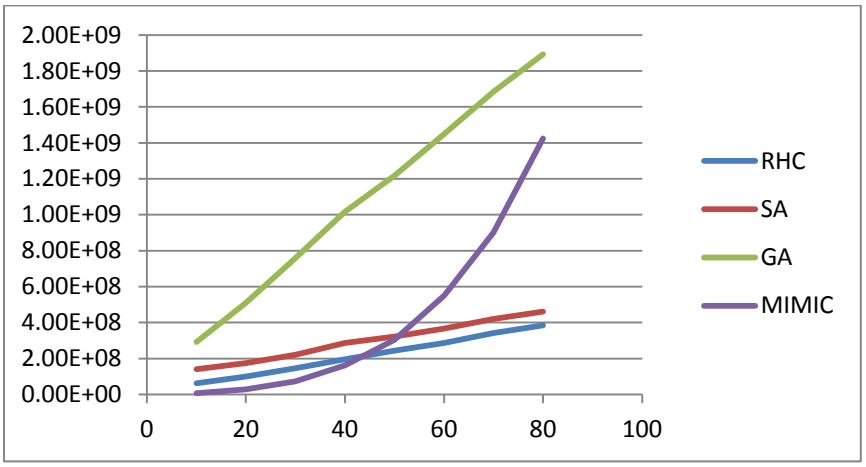
Application of Optimization Algorithms

I have applied RHC, SA, GA and MIMIC to optimize results for the TSP problem. I have returned the inverse of the distance in my maximization function in order to cast this as a maximization problem. The below graphs explain the output obtained. I am evaluating two parameters namely computational time needed for optimization and the maximized value resulting from the use of each method. RHC & SA seem to take minimal computational time, as there are no intensive operations involved in each iteration of hill climbing or SA. From the above graph, GA outperforms all other algorithms. The algorithms were run for about 5 times and the average result for 100k iterations is shown below. MIMIC was an exception was run for only 10k iterations as each iteration in MIMIC is computationally intensive.



N→ Input Size

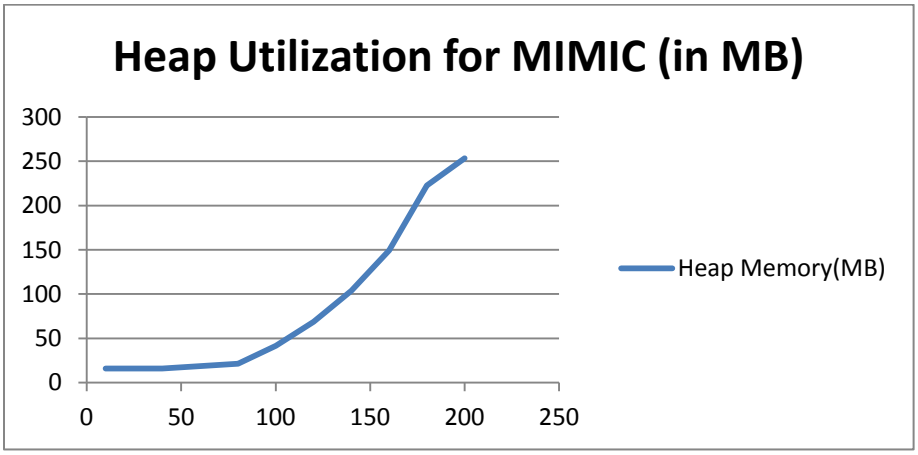
I have plotted the computational time (nano seconds) for each algorithm in the below plot. It appears as though GA, SA & RHC execution time seems to rise linearly against the input size (N- number of cities in the problem).MIMIC execution time seems to have a non –linear relationship with the input size.



N→ Input Size

Problems of Applying MIMIC to TSP

MIMIC has an acute problem of leveraging more space for building the dependency spanning trees and computing mutual Information. At one point, (N=500) MIMIC crashed the process due to lack of memory. So I plotted Java heap size to input size N in the below graph and found out that higher the values of N , MIMIC memory requirements are not scalable(shown below).The same analysis was done against other algorithms and they all showed heap size less than 20 MB and hence not listed below.



N→ Input Size

Result: Although Genetic algorithm takes more time, It seems to be the best choice for optimizing the TSP. The Genetic algorithm was run with multiple crossover functions, uniform cross over and a problem specific cross over(PMX function).The PMX crossover function played a major role in optimizing fitness values for the TSP and helped them distinguish from RHC & SA.

Four Peaks Problem

Problem Description

The fitness functions defined on bit strings containing 100 bits and parameterized by the value T:

$z(x)$ = Number of contiguous Zeros ending in Position 100
$o(x)$ = Number of contiguous Ones starting in Position 1

$$REWARD = \begin{cases} 100 & \text{if } o(x) > T \wedge z(x) > T \\ 0 & \text{else} \end{cases}$$

$$f(x) = MAX(o(x), z(x)) + REWARD$$

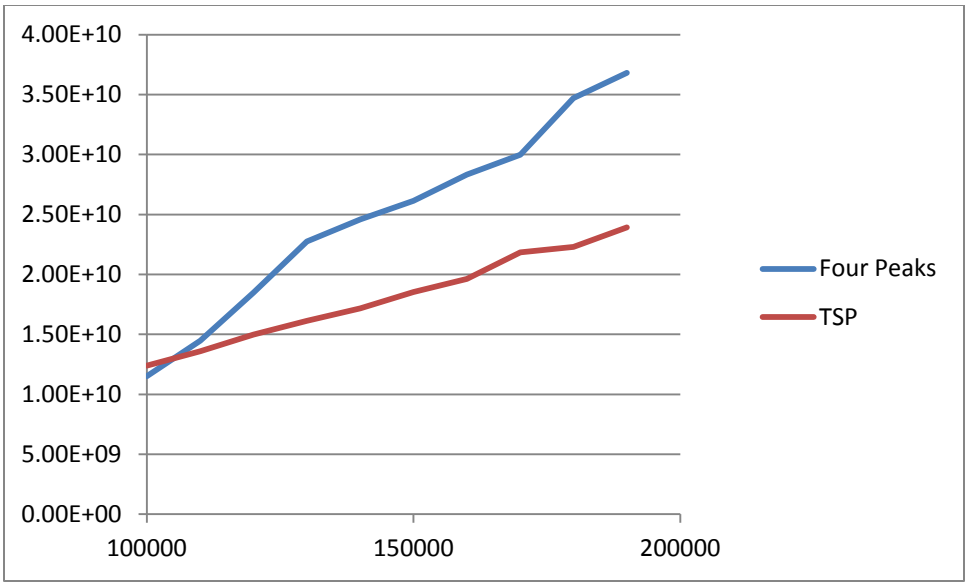
Suppose T=10. Fitness is maximized if a string is able to get both the REWARD of 100 and if the length of one of O(X) or Z(X) is as large as possible. The optimal fitness of 189 (when T=10)

Application of Optimization Algorithms

I have run the RHC , SA, MIMIC and GA against the below problem and have gotten the results as shown below. The below table depicts the number of iterations that are needed for the algorithm to maximize. I tweaked the algorithm to exit if it hit local optima and noticed no change to RHC and SA maximized values ***for my specific tests*.** This means the window for checking plateau conditions (about 10K iterations for SA, RHC and 1K for MIMIC) does not cause the algorithm to prematurely exit before hitting the actual maximum. From the graphs one can say that MIMIC is likely the best choice for the four peaks problem. However, MIMIC does have an issue with getting stuck at local optima at times. I am not sure how to handle this. I checked Piazza and the paper written by Professor Charles and eventually found out that MIMIC is prone to getting stuck at local optima for higher input values. I have changed my parameters to minimize this occurrence, but it cannot be ruled out when testing my algorithms.

```
N = 50
  Iterations  12350 RHC: 50.0
  Iterations  12400 SA: 94.0
  Iterations  331 MIMIC: 94.0

N = 60
  Iterations  13867 RHC: 60.0
  Iterations  13047 SA: 60.0
  Iterations  394 MIMIC: 113.0

N=70
  Iterations  14142 RHC: 70.0
  Iterations  16225 SA: 132.0
  Iterations  604 MIMIC: 132.0

N=80
  Iterations  15836 RHC: 80.0
  Iterations  16674 SA: 80.0
  Iterations  658 MIMIC: 80.0


N=90
  Iterations  16658 RHC: 90.0
  Iterations  18834 SA: 90.0
  Iterations  814 MIMIC: 90.0

N=100
  Iterations  20119 RHC: 100.0
  Iterations  20174 SA: 100.0
  Iterations  719 MIMIC: 189.0

N=120
  Iterations  25958 RHC: 120.0
  Iterations  18622 SA: 120.0
  Iterations  2218 MIMIC: 120.0
```

Results: I understand that MIMIC is the best algorithm for optimizing Four Peaks problem. I took the evaluation functions for Four Peaks and Travelling salesman problem and tried to understand the differences. My understanding is that Four peaks evaluation function is computationally more intensive than the TSPP problem which makes the heavy lifting in terms of space using the spanning trees and MI calculation worth the effort. Since TSP's evaluation function is not that intensive it makes sense to explore using greedy approaches than using MIMIC.



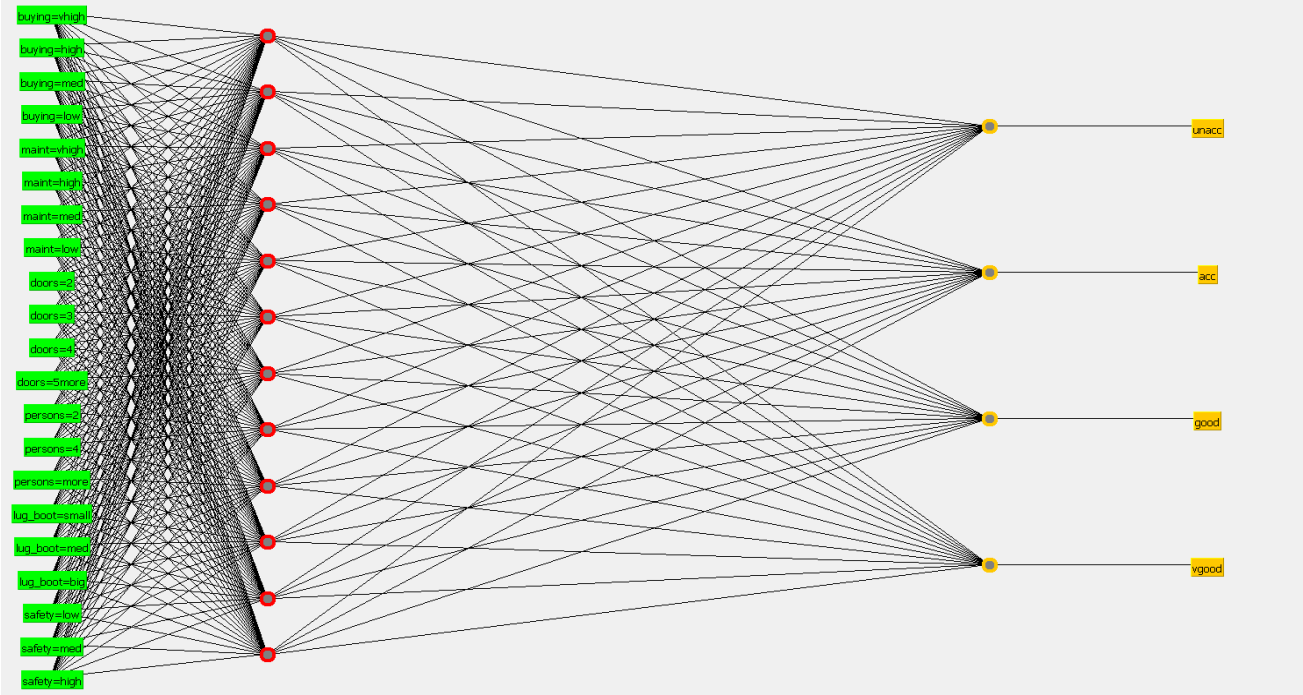
Neural Network Optimization Problem

I applied RHC, SA, Genetic algorithms for optimizing the weights of the neural network. The discrete values have been numerically encoded for being able to be processed by Abagail.

The classification problem being chosen is a problem of classifying cars into four discrete categories: ***Acceptable, Unacceptable, Good, Very Good.***

Problem Description:

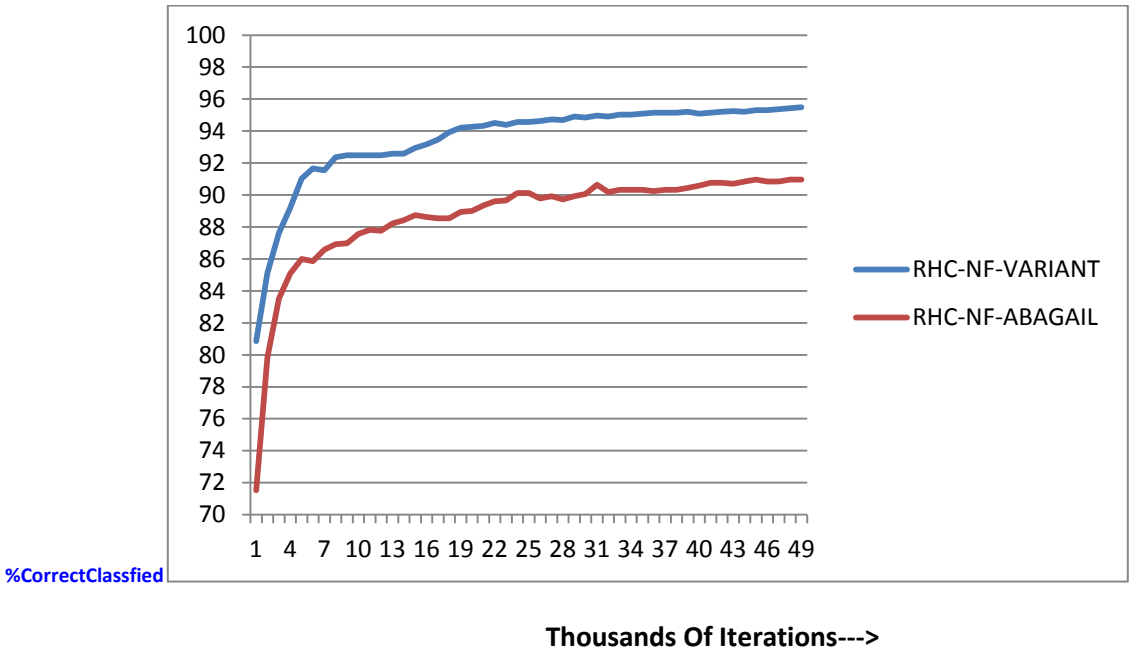
```
Car Evaluation Database was derived from a simple hierarchical
decision model originally developed for the demonstration of DEX
(M Bohanec, V Rajkovic: Expert system for decision
making Sistemica 1(1), pp 145-157, 1990) The model evaluates
cars according to the following concept structure:

   CAR                      car acceptability
   PRICE                    overall price
   buying                   buying price
   maint                    price of the maintenance

TECH                        technical characteristics

   COMFORT                  comfort

   doors                    number of doors
   persons                  capacity in terms of persons to carry
   lug_boot                 the size of luggage boot
   safety                   estimated safety of the car
```



RHC & SA:

I tweaked Abagail RHC's neighboring function and wrote my own neighboring function. The motivation for wanting to tweak RHC came from coming across a paper on "Adaptive RHC neighboring function". However, due to lack of time for reading the paper, I wrote my own neighboring function that would that at any given point maintained three values for adjusting the treading of the algorithm.

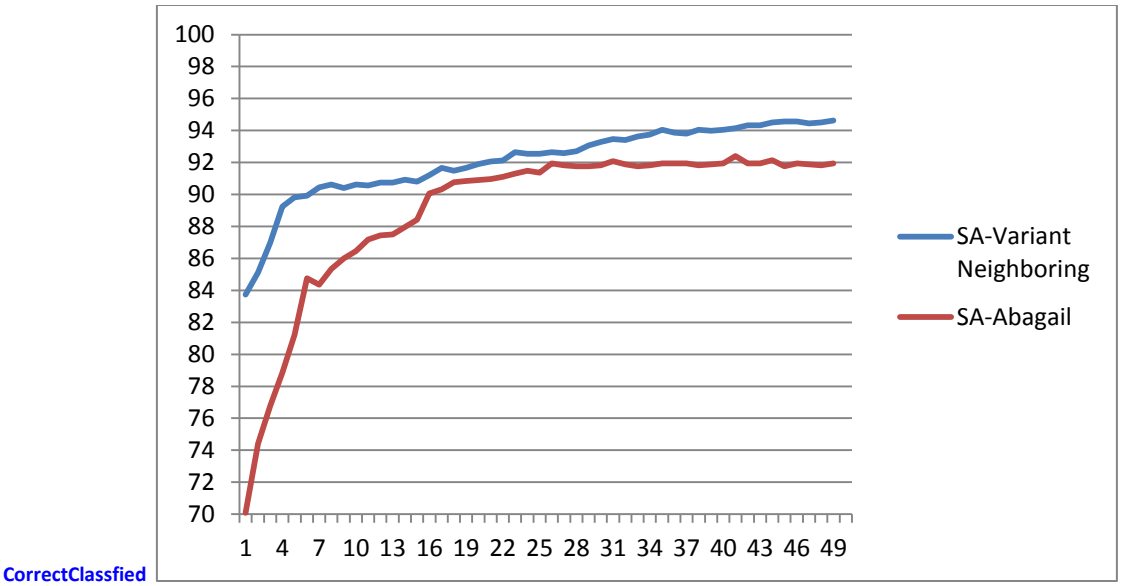
1) It would maintain three values for the adjustment that is being applied to the weights.
   One adds a minimal weight, one a nominal and another one a higher weight.
2) The error values corresponding to all these three values will be tracked. If the error values corresponding to the minimal weight has been flattened, then we apply a higher weight in subsequent iterations and keep doubling the weight adjustment until the error flattens again.
3) Once the error flattens we are again at a decision point where we decide either to tread slowly ,faster or keep the same pace.
4) The other parameter we control for treading RHC is the number of weights to which an adjustment is applied.
   The explained algorithm is present in ContinuousAddOneNeighbor.java. RandomizedHillClimbing.java and SimulatedAnnealing.java have their **customTrain** methods that leverage the tweaked neighboring function when the variant is invoked. To be sure I have made these changes in a separate file, so we can compare the performance before and after the changes.
5) It was also noticed that changing about 5 weights at a time to move around the neighborhood helped improve minimize the error.



The above chart shows the difference in performance between the abagail RHC neighboring function and the variant neighboring function that I have written. As you can see, we are taking a big hit in the training error values compared to gradient descent when using the RO methods to find those weights. So implementing this neighboring function helped alleviate those problems to some extent.

The below chart has been obtained by implementing the variant neighborhood function on Simulated annealing.

I did try tweaking the cooling factor for the temparature by 5% above and below but could NOT notice any significant reduction or increase in error. Hence I chose to use the 0.90 cooling factor for running my SA.



GA& RHC & SA Computational Time

RHC and SA have overall taken about 50,000 iterations (15 minutes of clock time) to provide weights that when applied on a network give yield about 95% classification correctness. However, GA took more than 100,000 iterations and ran for about five hours or so only to converge to 90% classification rates. My choice for optimizing the network would be RHC/SA given as they both yield similar comparable classification rates, however take far more time than gradient descent to minimize errors. GA certainly has serious performance issues.

GA

The GA had to be run for 100K iterations to converge to even 90% classification correctness of the neural network. I tried removing crossover functions to see how they fared and the GA performance dropped to 30% but the iteration execution time reduced which was pointless.

```
Training Results for GA-Min Population:
Correctly classified 1386.0 instances.
Incorrectly classified 142.0 instances.
Percent correctly classified: 90.707%
Training time: 18532 seconds
```