CS7641 Assignment 2 - Randomized Optimization

Ben Parli

March 13, 2016

Introduction

In this assignemnt we are to implement four Random Local Search Algorithms; Random Hill Climbing, Simulated Annealing, a Genetic Algorithm and MIMIC. Random Hill Climbing, Simulated Annealing, and the Genetic Algorithm are to be used to discover the optimal weights for the Neural Network arrived at in Assignment 1. These algorithms will take the place of the Back Propogation algorithm so commonly used in Neural Networks. In the second half of the assigment, the four algorithms will be applied to three new optimization problems, namely the count ones problem, the four peaks problem, and the knapsack problem. These Computer Science problems will be discussed more in depth in their respective sections later in the assignment.

The following sections are organized as follows; the datasets are described and a brief discussion on applicability to this assignment. The next section begins with the approach to unsing Random Local Search to fit a specific (i.e. from Assignment 1) Neural Network. Following that, the results and analysis of each Random Local Search algorithm applied to the Neural Network as well as an overall summary of findings is provided. The second major section of this assignment describes three common optimization problems and the application of each Random Local Search algorithm. The second section will also conlcude with overall findings and analysis.

Datasets

The classification problems chosen for this course are the Wine Quality dataset and Car Evaluation dataset. Both were downloaded from UC Irvine's public repository and both present examples of interesting and potentially far-reaching applications of machine learning and pattern recognition. Wine quality has been shown time and again to be extremently subjective and even based on visual cues. As a business then, the wine-maker may seek some additional data-driven guarrantee to ensure quality. Further, if patterns can be objectively derived to determine quality based on physicochemical properties in as fickle an industry as wine, there is no reason a similar approach couldn't be applied to industries elsewhere. On the other side of the counter and in a similarl way consumers can leverage such Machine Learning output to determine product quality independent of a subjective human "expert." In Assignment 1 this dataset proved very difficult to fit and none of the tuned algorithms were able to perform at better than 25% error. As such, this dataset will be applied to the Random Local Search Algorithms.

The Car Evaluation dataset is the other dataset explored in Assignment 1. This dataset proved to be easily modeled and all of the algorithms of Assignemt 1 performed well. The Neural Network performed with 0% training error and 3% test error. As a result we would only be able to answer whether a Random Local Search algorithm could perform as well, not beat the Back Propogation used in Assignment 1. Because of this, the Car Evaluation dataset will not be used in Assignment 2, only the Wine Quality.

Fitting Neural Networks with Random Local Search

The original, tuned, Wine Quality Neural Network was made up of one hidden layer of 12 nodes. The weights of the optimal model ranged from approximately -20 to 20. As a result of computation time I decided to focus the search some by boudning the maximum and minimum weights. I also decreased the number of hidden nodes to 3 for the same reason. Although this configuration was found to be sub-optimal in Assignment 1 (not complex enough), we can still answer the question of whether the Local Random Search algorithms can perform better than the Neural Network Back Propogation. Under this known sub-optimal configuration the Neural Network showed a 39% Training error rate.

Randomized Hill Climbing

For the Randomized Hill Climbing algorithm, a custom R script was used for the Hill Climbing and, in order to account for the algoirthm's succeptibility to local optima, a for loop used to generate random seeds. The best result of the for loop was taken to be the global optima. The implementation tuning primarily focused on the number of iterations the algorithm was allowed to run and the "location" from which the algorithm would begin its search.

Leaving the weights at a much higher magnitude (100) than the Backpropogation weights resulted in poor performance of around 60% error rate. This is somewhat surprising since larger weights often equate to a more complex model and can often result in overfitting and a potentially overly complex Neural Network. As noted earlier, tweaking these down towards the Backpropogation weights, although cheating somewhat, yielded better performance. The weights were bounded at -30 and 30 (the Back Propogation Neural Network weights ranged from approximately -20 and 20)

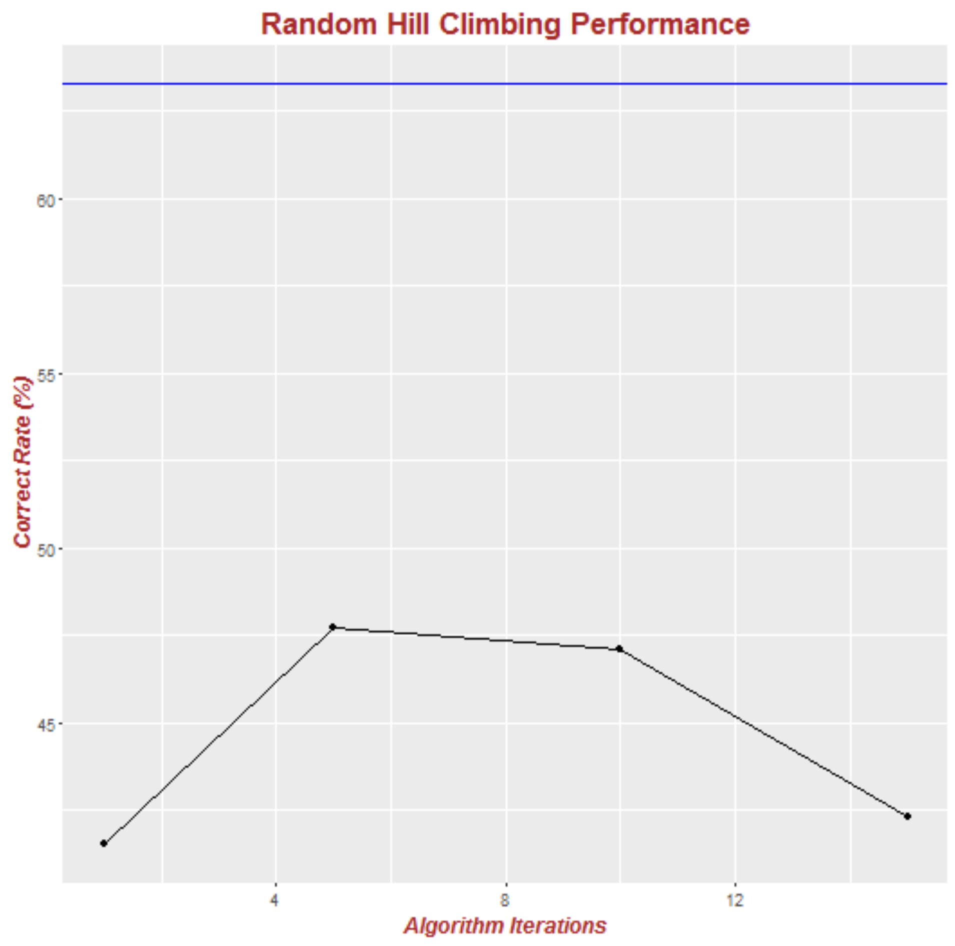
One key to this algorithm (atleast this implementation of it) is generating the random seeds in the for loop. Starting from the same seed, analogous to starting from the same point on the terrain, yielded the exact same result over and over. This is expected in Random Hill Climbing since the algorithm will deterministically proceed in the same way if starting from the same point.

The other key to this algorithm is just as obvious; allow enough iterations of checking neighbors to locate an optima. As can be seen below, if enough aren't allowed the algorithm suffers, although performance does taper off eventually.

Its also interesting to note the extreme variance in training performance the algorithm showed with only one iteration but 10 random seeds. As the number of iterations increased this variance plummeted as would be expected. This again reinforces how neccessary the random restarts are to the performance of this algorithm in scenarios with local optima.

```
[1] 0.06088407 0.41534612 0.02502085 0.12760634 0.41534612 0.40700584 0.34445371 0.41284404 0.17180984
0.40950792
```

Also, the computation time grew non-linearly from around 60 seconds at 10 seeds and 1 iteration to around 900 seconds at 10 seeds and 15 iterations.



Despite the improvements as the algorithm is allowed to run, the performance still does not quite approach the Back Propogation Neural Network or even approach a usbale model. In the plot above, the horizontal blue line is the 3 node Backpropogation Neural Network and acts as a baseline performance metric we hope to approach with one of these Random Local Search algorithms.

Simulated Annealing

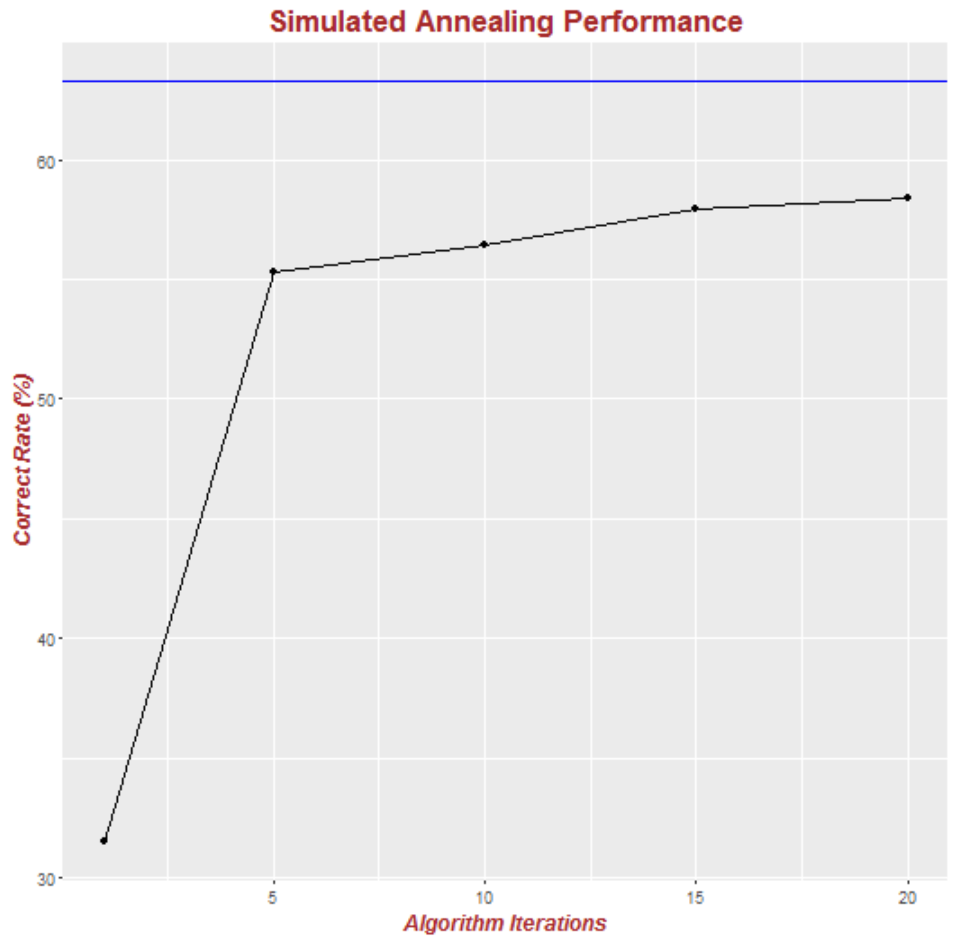
The Simualted Annealing implementation used the R package GenSA. A fitness function is required as an input, but the initial temperature was also varied to determine its effect. For this case the proposed weights were applied to a Neural Network and tested, with the resulting error rate acting as the score. The algorithm takes this score and, similar to the Random hill Climbing, searches the neighbors, but this time for both improvements and performance losses. Unlike the Random Hill implementation Simulated Annealing may randomly choose a less optimal path in hopes of finding a more optimal result on the other side. In order to compare implemnetations, the random weights of the Neural Network generated by Simualted Annealing were bounded at -30 and 30. Some additional findings:

- Similar to the Random Hill climbing, raising the potential bounded weights lowered the performance of the algorithm.
- Unfortunately, and somehwat surprisingly, attempting to raise the temperature parameter did not yield better results

Also similar to the Random Hill Climbing, there was an extreme variance in training performance with only one iteration but 10 random seeds. As the number of iterations increased this variance plummeted as would be expected since the algorithm was allowed to search further from its inital starting point.

```
[1] 0.03836530 0.04003336 0.01167640 0.31276063 0.11843203 0.11843203 0.01084237 0.01167640 0.33944954
0.11843203
```

Also, the computation time grew non-linearly from around 2 seconds at 10 seeds and 1 iteration to around 1520 seconds at 10 seeds and 20 iterations.



The Simualted Annealing and Random Hill Climbing implementations show similar traits, however, Simulated Annealing displayed an improved classification performance, even approaching the Back Propogation Neural Network. This makes some sense sense since Simulated Annealing is designed to do a good job of finding the global optima given a stochastic function with peaks and valleys. By starting off hot and finding different peaks it can eventually zero in on what is important. An extra Simualted Annealing run was performed since the algorithm appeared to still be improving. However, at 20 iterations, the graph still shows a plateau in performance. In the plot above, horizontal blue line is the 3 node Backpropogation Neural Network and acts as a baseline performance metric. Various ranges of temperature were used as a parameter in this algorithm, but interestingly it did not show any increased performance. The literature suggests higher temperatures could yield better results for more complex functions. Intuitively, this makes sense as it will allow the algorithm more opportunity to search, however, this was unfortunately not the case for this dataset. Settings from 100-1000000000 were used, but the higher temperatures did not show any improvements in performance.

Genetic Algorithm

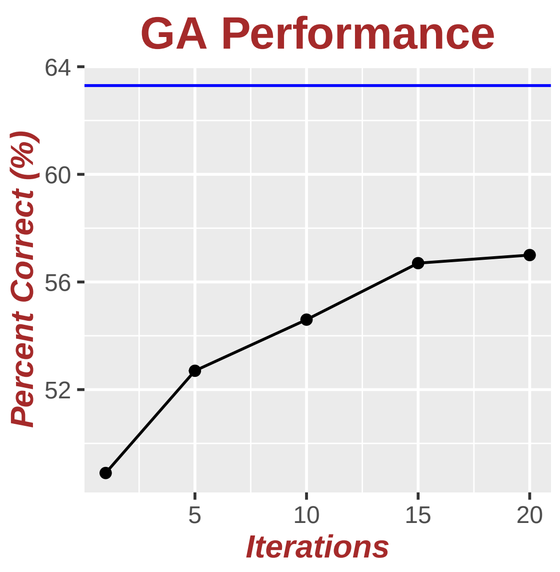
The Genetic Algorithm implementaiton used the R package GA and again a fitness function is required as an input. Just like with the Simulated Annealing implementation, the proposed weights were applied to a Neural Network and tested, with the resulting error rate the fitness score. The Genetic Algorithm implementation has a variety of tunable parameters such as pcrossover (the probability of crossover between pairs, typically set at 80%) and pmutation (the probability of mutation in a parent chromosome, typically set at 10%). Sadly, tweaking these parameters from their default did not yield any noticable improvement in the performance of the algorithm, however. The focus will instead continue to be performance of the algorithm as a function of the number of iterations it is allowed to run.

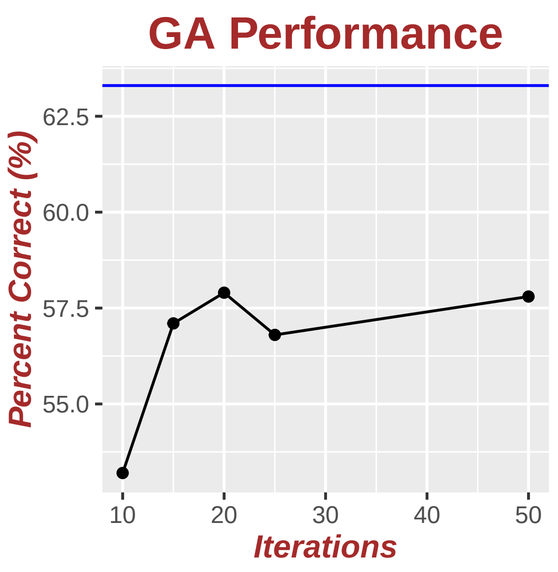
One notable difference in the GA implementation is the parrallel processing parameter baked into the library. This is, of course, only parrallel within each instantiation of the GA library and not the R script itself. That said the computing time was initially much longer with this algorithm as compared to the other two; from around 120 seconds at 10 seeds and 1 iteration. However, the processing time grew much more linearly to only around 240 seconds at 10 seeds and 15 iterations.

Perhaps a bit unexpected after the Simulated Annealing and Random Hill Climbing observations, but there was much less variance in classification performance at the initial 1 iteration and 10 seeds. This slight variance in performance between random seeds remained constant throughout the Genetic Algorithm runs. The algorithm begins with the one iteration and 10 random seeds with nearly 50% correctly classified. Given there are six classifier choices, this is somewhat surprising. The one iteration with its (likely) crossover instruction were enough to get the algorithm to over 40% correct in all ten random starting points. The seed value is much less critical to this algorithm relative to Random Hill Climbing in particular.

```
[1] 0.4895746 0.4345288 0.4637198 0.4386989 0.4386989 0.4386989 0.4412010 0.4386989 0.4386989 0.4470392
```

The Genetic Algorithm showed promise and also approached the Backpropogation algorithm, but ultimately topped out at under 60% correct, similar to Simulated Annealing. Since it was much more computationally effiecient in time, and since it showed itself as a possible competitor to the Backpropogation algorithm, additional tests were run at more iterations. Also, the Genetic Algorithm was modified to supply enough weights (222) to fit the original Backpropogation Neural Net with 12 hidden nodes (i.e. the optimal model originally identified in Assignment 1).





Even with additional iterations, all the way up to 50, the algorithm performance tops out at under 60% classified correctly. Tuning the pcrossover and pmutation settings did not yield any additioanl performance at these high iterations either.

Final Analysis on Fitting Neural Networks with Random Local Search

As seen in the table summary below, overall the Local Random Search Algorithms did not approach the Backpropogation Neural Network performance. Obviously, the iterations of the Backpropogation Neural Networks cannot be directly compared to the iterations of the Local Random Search, but the compute time and error rate can be. Among the Random Local Search algorithms, the Genetic Algorithm showed the most promise, both in terms of computation time and in terms of performance. Given the featureset of the dataset is quantitative, this is perhaps not very surprising, especially when considering the difficulty of comparing neighbors in the Random Hill Climbing and and Simulated Annealing implementations. In the end, there really isn't any tradeoff, the Backpropogation Neural Network is superior when applied to this dataset.

Neural Net Performance Table

Model	Iterations	Train Error	Compute Time (Wall Time)
Backprop. NN (3 Nodes)	~160	36.7%	< 60 sec.
Backprop. NN (12 Nodes)	~700	29.1%	< 260 sec.
GA NN (12 Nodes)	50	42.2%	~ 410 sec.
GA NN (3 Nodes)	15	43%	~ 240 sec.
SA NN (3 Nodes)	20	41.6%	~ 1520 sec.
RHC NN (3 Nodes)	15	52.3%	~ 900 sec.

Optimization Problems

For the second stage of Assignment 2 we are to create or borrow 3 optimization problems which can then be solved using Random Local Search algorithms. The following sections are organized as follows; the implementation is first described, then the optimization problems. Within each optimization problem, the application of each Random Local Search algorithm and its performance is presented, and finally summarized observations. This section will conlcude with overall findings and analysis.

Implementation Notes

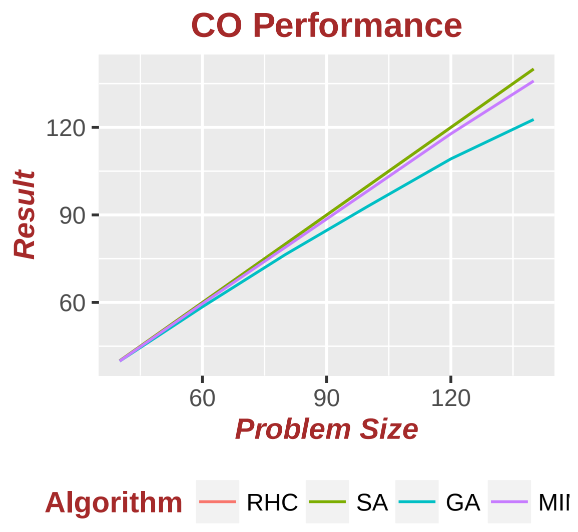
The ABAGAIL library is used as the algorithm implementation for this portion of the assignment and R was again used for the plots. Each algorithm was run a series of times (ranging from 40 to 140) with a series of parameters. In each run set, the average results, fitness function calls and time are captured for analysis and comparison.

Since the Genetic Algorithm has several tuning parameters, this implementation performed a series of nested for loops for each one, then captured the average results. A specific tuned Genentic Algorithm could easily be identified this way, however, this approach seeks to capture and compare the performance of the Genetic Algorithm across three disparate problems. Only a subset of the best results for each problem size was used in the final comparison. In this way we are able to compare the "tuned" models.

The MIMIC implementation took a similar approach to the Genetic Algorithm approach just described. Nested for loops ranged through possible parameters (specifically, the number of samples to generate and the percentage of samples to keep) and captured the average output. A subset of the best results, grouped by problem size, was captured and used to compare.

The Countones Problem

The Count Ones optimization problem description is as follows; we must decide on the majority of ones (or zeros) in the relevant attributes to determine the class given a string of bits. As can be seen below, when measuring these four implementations against each other with respect to classification performance the Simualted Annealing and RHC algorithms stand out as the optimal (its difficult to see in the plot, but they're actually overlapping each other). With the Simulated Annealing setting, we can norammly take this to mean the Temperature was set too low (for this optimization problem) since it essentially behaved like Random Hill Climbing. Tests at higher temperature settings yeilded the same result comparison, however. The Genetic Mutation did not fare as well and began deviating in performance almost from the start. We can attribute this to the problem itself; The Simulated Annealing and RHC, however, will continue until they hit an optima. That is, once they encounter the ideal string of ones, they will continue on until reaching the global optima. However, the Genetic Algorithm may, with some set likelihood, mutate or crossover children of a string it should have kept, basically getting in its own way. MIMIC performs well, but also falls short of Random Local Search and Simulated Annealing.



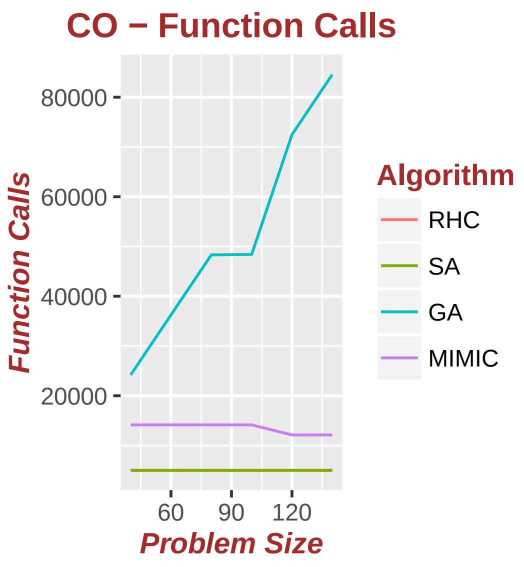
We can note a few takeaways from the Countones Computing Time and Evaluation Function calls comparisons below:

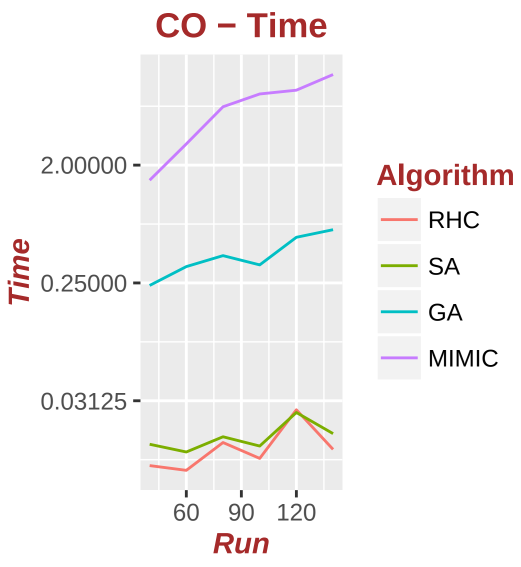
- The MIMIC algorithm performed worse in time efficiency due to the number of cycles within each iteration. This is due to the overhead and bookeeping within the algorithm itself in building a dependency tree and probability distribution with each iteration.

- Interestingly, the computing time took less for MIMIC than for the Genetic Algorithm implementation despite MIMIC being vastly more efficient in the number of evaluation function calls.
- conceptually, this makes some sense for the Genetic Algorithm, since it needs to evaluate every sample for the new population. As the problem size grows, the population size grows linearly, as does the number of fitness evaluations.
- Again, the Random Hill and Simulated Annealing performed identically in terms of evaluation functions. Simualted Annealing actaully performed a bit better in computing time, however.

I would have expected Random Hill Climbing to perform best on the countones problem when considering both model performance and computing time. Since there is one obvious global optima, Random Hill Climbing should find it fastest since there is no probablility of a misstep built into the algorithm as there is with Simulated Annealing and Genetic Algorithms. We can see Simulated Annealing actually matched Random

Hill Climbing though, indicaing the probaility of trying a less optimal neighbor never outweighed the more optimal option. This hypothesis was tested at temperatures all the way up to 1E10.

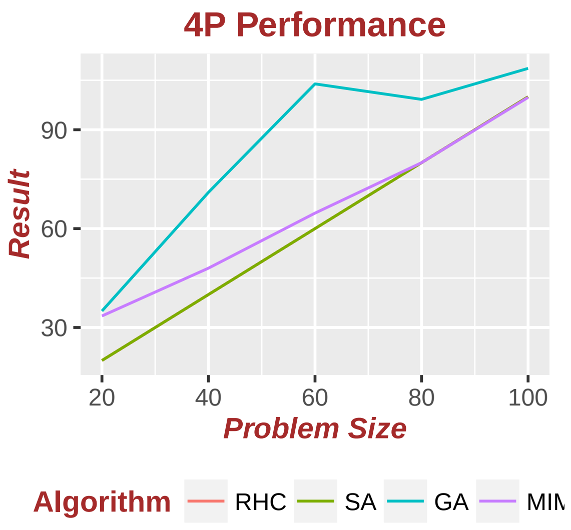




The Four Peaks Problem

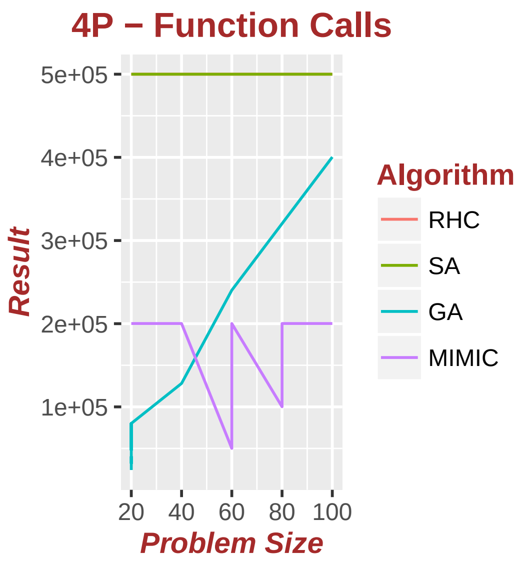
The Four Peaks optimization problem description is as follows; A vector X is input and made up of some number of binary items. The fitness function evaluates the binary string on whether it has achieved the maximum score and also whether the head or tail of the string is as large as possible. however, there are also two suboptimal local optima, designed to trap the algorithm. As the size of the string becomes grows, it becomes more and more difficult for the algorithm to find the global optima.

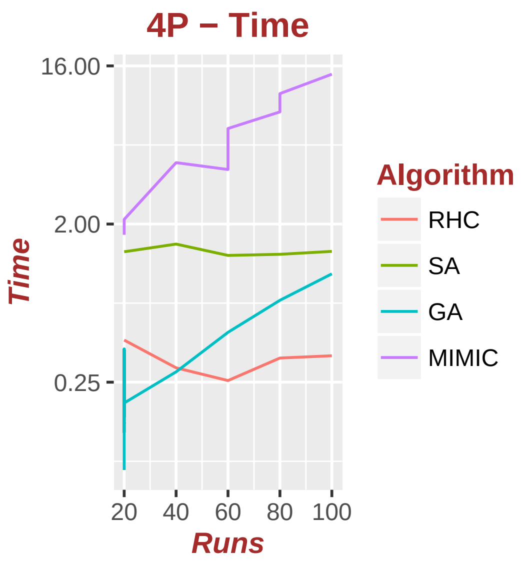
In the Four Peaks problem, the Genetic Algorithm shows best. As the algorithm identifies better "individuals" in its populations, we can infer they quickly begin to include the global optima and not just the local optima as the Random Hill climbing and Simulated Annealing are more succeptible to. MIMIC also performs well on this problem.



We can note a few takeaways from the Four Peaks Problem Computing Time and Evaluation Function calls comparisons below:

- Random Hill Climb and Simulated Annealing show a very similar profile to the Knapsack problem in terms of computing time and fitness function calls
- Being a more complex problem than Count Ones we expect Simulated Annealing to outperform Random Hill Climbing, particularly at higher initial temperatures. This does not appear to be the case, however.
- There again appears to be some noise in the MIMIC calls data, but we can see the Genetic Algorithm grow linearly with the Problem Size

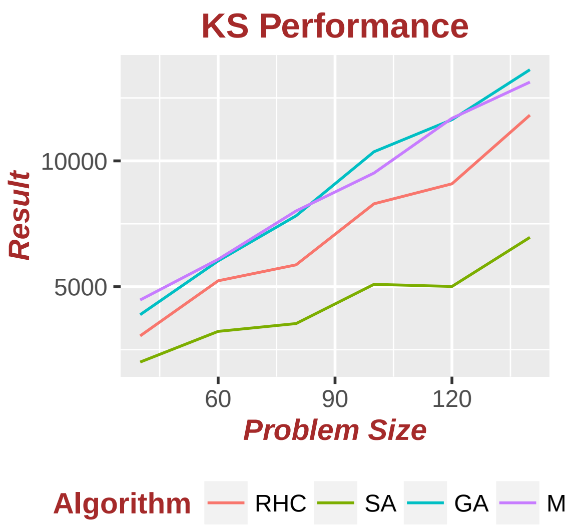




The Knapsack Problem

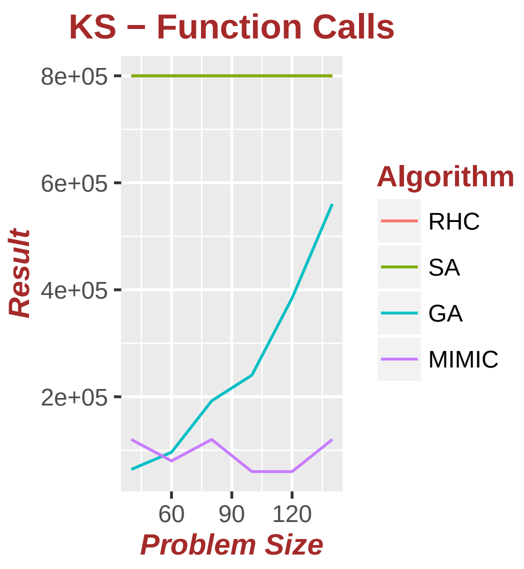
The Knapsack optimization problem description is as follows; a set of items is fed to the algorithm, presumably in a knapsack, with each item assigned a weight and a value. The objective is to determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. The problem is a combinatorial optimization problem thought to be NP complete with wide-ranging applications since many resource allocation problems can be reduced to it.

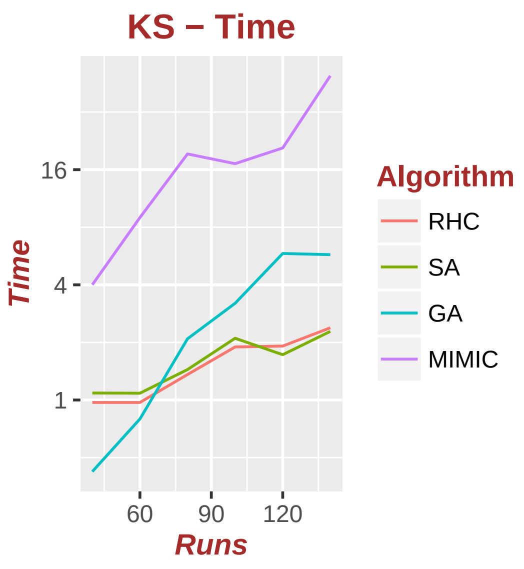
The performance comparison of these algorithms shift in the Knapsack Problem. Random Hill Climbing and Simulated Annealing don't lend themselves to the Knapsack Problem as well as they did to Count Ones. Since the Knapsack Problem is more complex and combinatorial, we can expect MIMIC and Genetic Algorithms to perform better. In identifying the right combination of items, both MIMIC and Genetic Algorithms build towards an optimal fit; Genetic Algorithm by replacing the least optimal with new, more fit "offspring" and MIMIC by narrowing down to the optimal population probability distribution. In fact, where Genetic Algorithms slightly outperformed MIMIC at lower problem sizes, MIMC began to outperform all others at the halfway mark, before Genetic Algorithms again pulled ahead at the end.



We can note a few takeaways from the Knapsack Problem Computing Time and Evaluation Function calls comparisons below. Consistent with the above observations, MIMIC takes the most time while making the fewest function calls:

- Random Hill Climbing finishes must faster despite many more fitness fucntion calls (again, the same amount as Simulated Annealing). We can again attribute this to the much smaller search space of Random Hill Climbing compared with MIMIC and Genetic Algorithm. Where Random Hill Climbing and Simulated Annealing are only evaluating their nearest neighbors, MIMIC and Genetic Algorithm are more computationally intensive.
- Even with MIMIC making the fewest fitness function calls, it still took the most time. This is again due to the overhead of each MIMIC iteration discussed above
- Simualted Annealing performed worse in computing time relative to Random Hill Climbing indicating it took longer to reach an optima since both algorithms follow similar steps. This seems to indicate the times when Simulated Annealing took the less optimal route it did not work out. That is, in searching for a better solution by proceeding through the proverbial valley, it did not come out the other side any better off.





We can also takeaway some insights into the problem itself from this exercise. A search agent can keep improving, but suddenly lose a lot of value by making one mistep in adding the wrong item. The knapsack can potentially lose value very easily and very quickly. This is why Simulated Annealing and Randomized Hill Climbing don't perform as well; there are a lot of dropoffs from a value of around 2000 to nothing.

Final Takeaways

We've seen the Random Local Search algorithms applied to both a Neural Network identified for the Wine Quality dataset and also applied to three optimization problems. Unfortunately, the algorithms did not outperform the Backpropogation Neural Network. From the optimization problem implementations, however, we can note a few important takeaways about these algorithms:

- As with the somewhat simplistic Countones problem, Random Hill Climbing and Simulated Annealing do well when a trend in the data leads to an obvious Global Optima. They are also performant since each iteration only makes some evaluation of its immediate neighbors.
- The need for random restarts in Random Hill Climbing was reinforced in the initial section of this assignment, where the algorithm was applied to the Wine Dataset
- MIMC did quite well with the more complex combinatorial Knapsack Problem. We can expect MIMIC to outperform simpler Local Search algorithms, such as Simulated Annealing, in these scenarios since it is building a more and more optimal distribution with each iteration even if the optimal solution is not apparent as in the Knapsack Problem. As was discussed in lecture (and above) though, MIMIC has its time complexity drawbacks.
- Genetic Algorithms also performed well and seemed to hit the balance between classification performance and time complexity. Genetic Algorithms also performed the best on the Four Peaks problem.
- The Simulated Annealing algorithm performed comparatively OK against the Wine Quality dataset, but didn't stand out otherwise. I would have expected some variance in performance from Random Hill Climbing at such higher temperatures but was not able to observe it through these tests. A particular hypothesis was Simulated Annealing should, on the whole, be the best algorithm when applied to the Four Peaks problem. With a high enough temperature, it should proceed through the peaks and local optima to eventually arrive at the global. It was Genetic Algorithms that stood out on that problem, however.

References and Citations

R Package nnet, Feed-Forward Neural Networks and Multinomial Log-Linear Models, https://cran.r-project.org/web/packages/nnet/nnet.pdf

R Package GenSA, Functions for Generalized Simulated Annealing, https://cran.r-project.org/web/packages/GenSA/GenSA.pdf

R Package GA, GA: A Package for Genetic Algorithms in R. Journal of Statistical Software, http://www.jstatsoft.org/v53/i04/

Random Hill R implementation, https://github.com/tyabonil/omscs.7641.ro/blob/master/my.ro.gen.R

ABAGAIL Machine Learning library, https://github.com/pushkar/ABAGAIL and https://github.com/chappers/CS7641-Machine-Learning/tree/master/Randomized%20Optimization

UCI Machine Learning Repository, Wine Dataset https://archive.ics.uci.edu/ml/datasets/Wine. Irvine, CA: University of California, School of Information and Computer Science.

UCI Machine Learning Repository, Car Evaluation Dataset https://archive.ics.uci.edu/ml/datasets/Car+Evaluation. Irvine, CA: University of California, School of Information and Computer Science.

Four Peaks Optimization Problem take from http://www.esprockets.com/papers/ML97.FINAL.fm.ps.pdf

Knapsack Optimization Problem taken from https://en.wikipedia.org/wiki/Knapsack_problem

Some code borrowed and tweaked from https://github.com/chappers/CS7641-Machine-Learning/tree/master/Randomized%20Optimization