**Abstract**

The immortal philosopher *Pokemon Theme* once said, ‘I want to be the very best, like no one ever was.’ While optimization predates this lyric, it does summarize the basic idea well: be the best. Randomized optimization methods seek to find the very best (global optima) for a given fitness function. This paper offers analysis of four contenders in aim of this pursuit: randomized hill climbing (RHC), simulated annealing (SA), genetic algorithm (GA) and MIMIC. Analysis on these four randomized optimization methods is presented in the context of training weights for a neural network and on three classic optimization problems (traveling salesman, continuous peaks and flip flop). Conclusions drawn on the respective strengths and weaknesses of each algorithm is discussed at the end.

# Random Optimization Algorithms, Problems and Methodology

All experiments were run using optimization methods from the ABAGAIL library. Optimzation problems were performed on discrete bitstrings with each method run 5 times to reduce the variance of performance from the ‘Random’ in Random Optimization.

## Random Hill Climbing (RHC)

*Parameters: NA*

Random hill climbing performs a simple process for optimization: start at a random point, evaluate the fitness function at neighboring points and if one is higher move there else stop. This is the simplest of the optimization methods. To reduce variance, multiple random initial points are tried.

***Simulated Annealing (SA)***

*Parameters: Cooling Exponent (CE)*

Similar to RHC, simulated annealing adds a further step when evaluating whether to move to a neighboring point. If the neighbor’s fitness value is lower than the current point, the optimization may still move depending on an acceptance function with four parameters: fitness value of the current point, fitness value of the neighboring point, temperature and cooling rate. If the distance between the points is close, the neighbor is more likely to be moved to. The parameter temperature is an effort to aid the algorithm in moving out of local optima. The temperature starts high and is cooled as the algorithm moves around. A high temperature is akin to a random walk, moving to neighbouring points every time, while a low temperature is closer to RHC in that it only moves when a neighboring point has a higher fitness function. As the number of steps increases the cooling rate decreases the temperature at a set rate. These parameters allow simulated annealing to become a type of adaptive hill climbing that is able to move out of local optima but settle on one optima as the temperature cools.

## Genetic Algorithm (GA)

*Parameters: Population, Mates, Mutations*

Genetic algorithms mimic the biological process of survival of the fittest reproduction in biology. Start with a population of points and apply some criteria to them, for example keeping the top half in terms of fitness value. Pair up the most fit mates (points) with the least fit of the surviving mates and apply some predetermined “mutation” or “crossover”, which is a combination of the two mates. The children points then replace the least fit mates from the population and the process is repeated. This process allows the algorithm to move in a direction that highlights the strengths of the top performing points while ignoring irrelevant points.

***MIMIC***

*Parameters: Samples, Keep, M*

MIMIC seeks to find the global optima while retaining information about the structure of the problem itself and does so probabilistically. Mimic starts with a distribution using all points and generates a distribution using dependency trees and some mutual information methods. Given some criteria (usually percentile threshold) the next distribution is generated using only the points above that threshold. This is repeated until a max is reached.

# Neural Network Training

## Madelon Data

### Instances: 5000 | Attributes: 440 | Data Types: Continuous (440) | Classes: 0, 1

The MADELON dataset is an artificial dataset created in 2003 for the NIPs conference as part of a feature selection challenge. The target class comes from a group of 32 clusters on the vertices of a five dimensional hypercube. Those points were randomly assigned a class (either 1 or -1). Additionally the five dimensions were transformed by linear combinations to form fifteen more features. To complicate the problem, 480 features of random noise were added to the dataset.

Of interest here is that the Madelon dataset presents a highly non-linear problem where the signal-to noise ratio is very low. 1% of the features are truly useful (the 5 dimensions) while 15 (3%) are superfluous albeit still informative. This leaves 96% as completely useless to learn from. To alleviate some of the imbalance in signal-to-noise ratio, sklearn's feature selection method SelectFromModel in tandem with a RandomForestClassifier was implemented. The feature selection was repeated four times with a threshold set to 'median', i.e. any feature deemed to be in the lower half of feature importance is dropped. In other words, the more important half of the features were kept with this repeated four times leaving 31 features for the algorithm to learn from. In the best case scenario, this would leave the 20 informative features and 11 noise features.

***NN Weights Training Experiment***

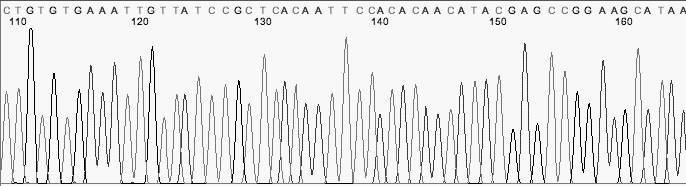
Each optimization algorithm (and backpropogation, the typical gradient based weight selection method) was used to train the weights on neural network with three hidden layers, each with 62 neurons and activated using Relu. These parameters were found to be optimal in previous supervised classification analysis. The training weights were tuned to minimize the mean squared error of the predictions and maximize accuracy. The Madelon set was split into a train/test/validation split of roughly 55/15/30. Results in terms of MSE, accuracy, algorithm iterations and time elapsed are examined.

# Optimization Problems

# Traveling Salesman

The Traveling Salesman is among the most famous optimization problems and a classic NP-hard problem The name comes from the setup of the problem: if a salesman is given a list of cities he must visit and the distances between all pairs of cities what is the shortest route possible visiting each city once and returning home (the initial city) at the end? This is a combinatorial optimization problem, as the potential solutions grow in a combinatorial matter as more cities are added. The nature of this problem also reflects structure in a good solution, as the distances are constant and once a city is visited it may not be chosen again. That is, just following the nearest neighbor of each city will likely not lead to a global optima. Most good solutions to the TSP problem involve moving between subclusters of cities close together and finding the optimal route between correctly identified subclusters. For this problem in ABAGAIL the bitstrings were implemented with N=100, max number of iterations run at 3000 and each algorithm parameter set was run 5 times to reduce variance of performance.

# Continuous Peaks

The Continuous Peaks problem examines how an algorithm performs when many local optima are present. This is similar to a wave function with many local optima but only one global optima:t

The difficulty here is falling into the many local optima present. With a bit string as an input space, this would look like many sequences with potential optima, such as 00001111111000011100111111111111111000. Optimization methods robust to local optima may perform better than others here. For this problem in ABAGAIL the bitstrings were implemented with N=100, max number of iterations run at 5000 and each algorithm parameter set was run 5 times to reduce variance of performance.

# Flip Flop

The flip flop problem is simple: for a given length bit string, find the correct sequence. Here structure matters in that each bit is independent of all others. Finding the optimal solution must preserve the subspaces (each singular bit) while ignoring locality of structure (groups of bits and their interrelationships are independent). Finding an optimal solution quickly must implement such a solution while a brute force solution would try all 2^N possibilities (where N is the length of the bit string). For this problem in ABAGAIL the bitstrings were implemented with N=1000, max number of iterations run at 3000 and each algorithm parameter set was run 5 times to reduce variance of performance.

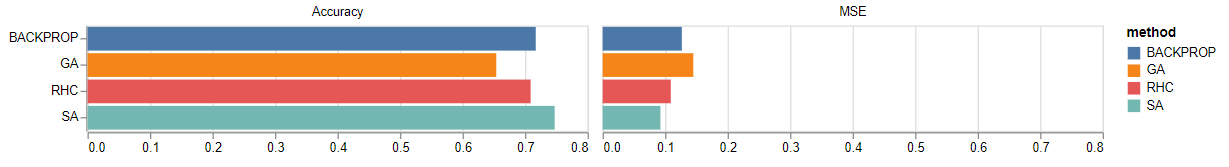
# Results

## NN Weights Training Experiment

Resulting scores on a holdout (validation) set were measured using classification accuracy and mean squared error (MSE):

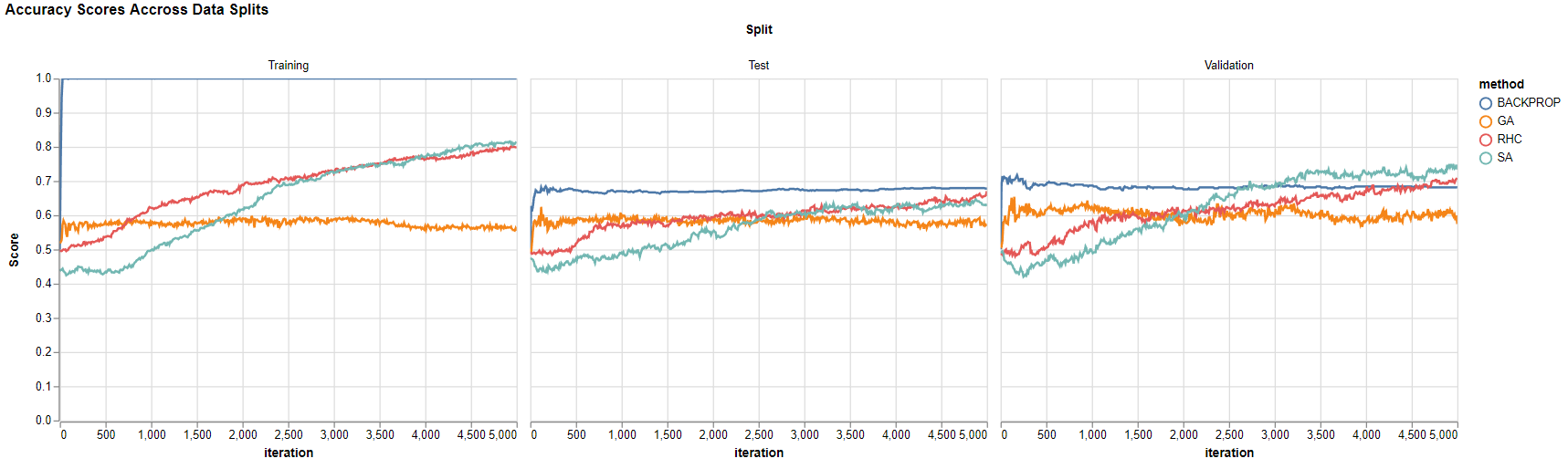
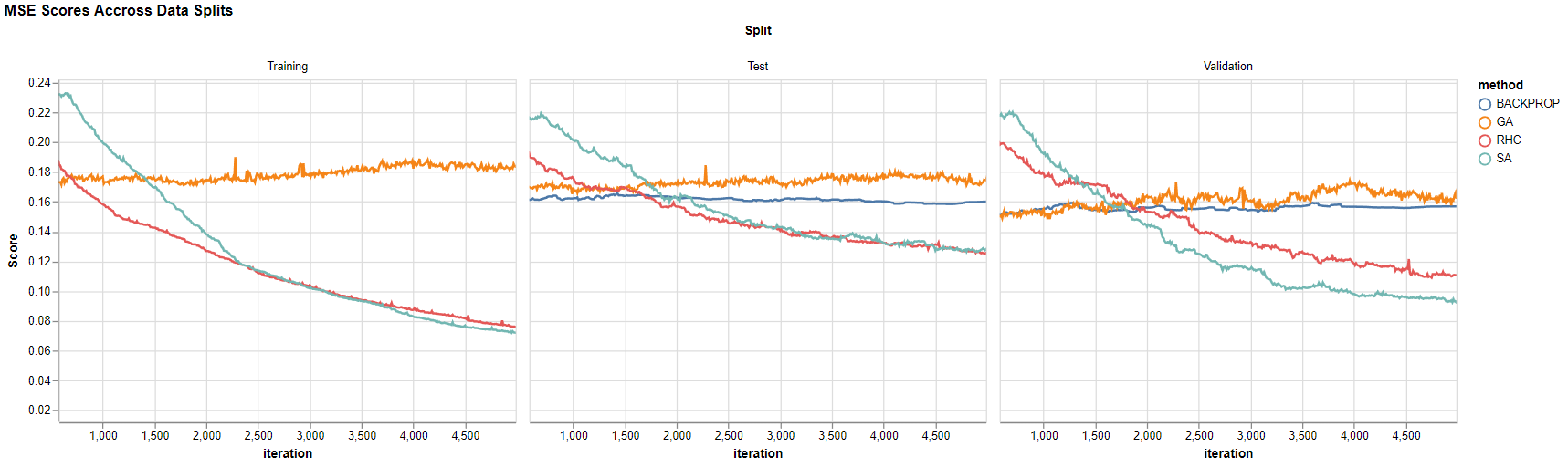
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Iterations | Time Elapse | Optimization Method | Metric | Score | Split |
| 4890 | 249.8347 | SA | Accuracy | 0.747253 | Validation |
| 180 | 29.3109 | BACKPROP | Accuracy | 0.717033 | Validation |
| 5000 | 270.8044 | RHC | Accuracy | 0.708791 | Validation |
| 150 | 93.03761 | GA | Accuracy | 0.653846 | Validation |
| 4990 | 254.9385 | SA | MSE | 0.092445 | Validation |
| 4750 | 257.2424 | RHC | MSE | 0.109042 | Validation |
| 20 | 3.296126 | BACKPROP | MSE | 0.126715 | Validation |
| 150 | 93.03761 | GA | MSE | 0.145067 | Validation |

# Represented Visually:



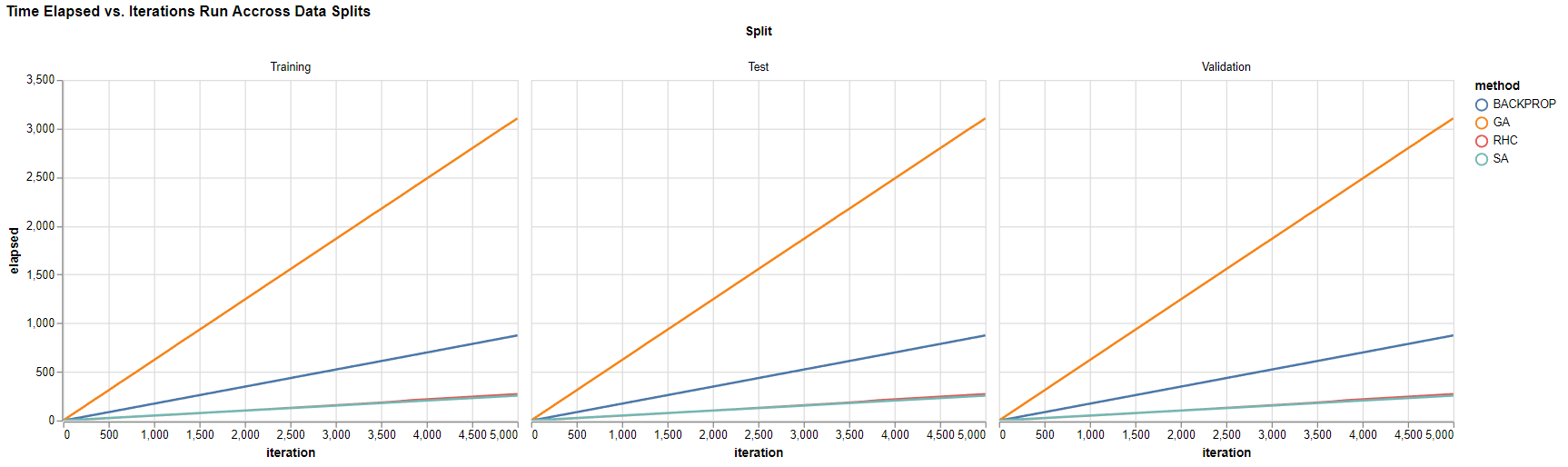
Judging by these results, simulated annealing performed the best overall on the validation sets, scoring slightly better than both BACKPROP and RHC for accuracy and MSE scores.

This is further evidenced when viewing the accuracy/MSE scores across data splits:

The first thing that jumps out is how backprop massively overfits the training sets for both metrics. It scores a maximum score for accuracy (1) and a very low MSE compared to the other algorithms (almost 0). In short, backprop massively overfits the training data. This is no doubt due to the massive expressive potential of an ANN with three large hidden layers. Backprop can minimize the error due to the 7,388,168 connections it is able to adjust with respect to minimizing training error (these weights come from the fully connected layers and inputs- 31 features in the input layer\*62 neurons in the first layer\*62\*62). Backprop can adjust quickly and with large magnitude. This is contrasted by the randomized optimization algorithms. The GA performed poorly throughout and did not improve much at all, however the RHC and SA made steady progress that is reflected across all splits. Slowly but surely the scores increased (downward for MSE and upward for accuracy) while eventually surpassing the backprop scores. This suggests that RHC/SA found, if not a global optima, at least a better local optima than backprop. Simulated annealing scored better than RHC after 2500 iterations which suggests that at that point there was a necessity to hop out of local optima, something it did better than both backprop and RHC.

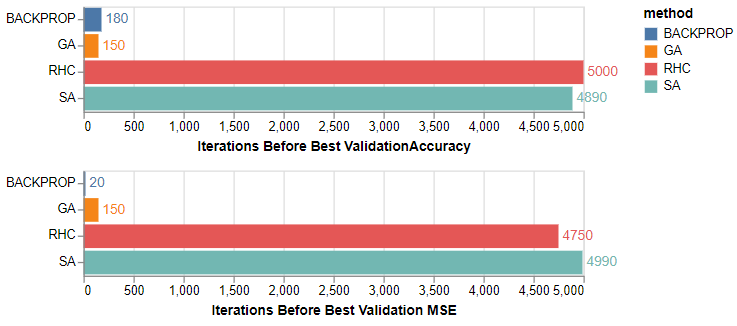
It makes sense that the GA would perform poorly whereas backprop/RHC/SA would perform similarity. GA can perform well where there is structure reflected by the criteria for evolving the weights, but in this case that structure was not found. It’s possible that the criteria could be changed so that the top 5th percentile would be chosen and in doing so this may have had the algorithm focused only on the true features of the Madelon dataset (5 out of 31) rather than optimize all training weights which include some redundant features and noise features. RHC/SA is similar to backprop but where backprop skips to the answer (the gradient which minimizes error in the maximal direction and taking a step), RHC/SA is much slower in this process. It looks at nearby neighbors, evaluates, and moves. This takes many more iterations to achieve the same level of error reduction but is less prone to overfitting like the backprop does with the training data. With enough data and iterations, it is possible that RHC/SA achieve a similar level of training scoring as backprop.

In terms of timing, all algorithms scale roughly linearly with the number of iterations:



GA took the most per iteration and scaled similar to O(n), while backprop/SA/RHC each scale closer to O(logn) (but still appears linear suggesting that it would not converge to logn, but perhaps be close in the optimistic case). The times didn’t change with the size of the data suggesting that the algorithms scale well with larger data sizes and most of the cost comes with the number of iterations required to run it in.

Since RHC/SA is more robust to overfitting and ultimately performed better, and additionally scaled better with iterations why is backprop the standard for training NNs? The question can be answered by this graph:

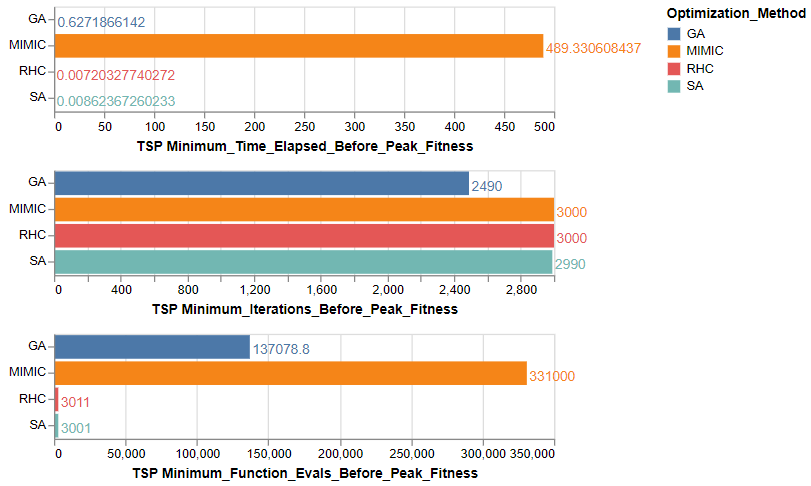
Although backprop doesn’t perform optimally it does do 90% of the job for far cheaper. Depending on how it is viewed backprop reached its optimal performance in roughly 1/25th to 1/50th as many iterations as RHC/SA for MSE and accuracy scores respectively. Applying this to computational cost expressed in time, this is a roughly an x80 speed up. While RHC/SA performed admirably on this problem it’s clear why backprop is implemented for practical reasons.

**Optimization Problems**

***Traveling Salesman***

|  |  |  |  |
| --- | --- | --- | --- |
| **Optimization\_Problem** | **Optimization\_Method** | **Parameters** | **Peak\_Fitness** |
| TSP | GA | 100\_50\_10 | 0.115696 |
| TSP | MIMIC | 100\_50\_0.1 | 0.042634 |
| TSP | RHC |  | 0.053812 |
| TSP | SA | 0.15 | 0.054405 |

# 

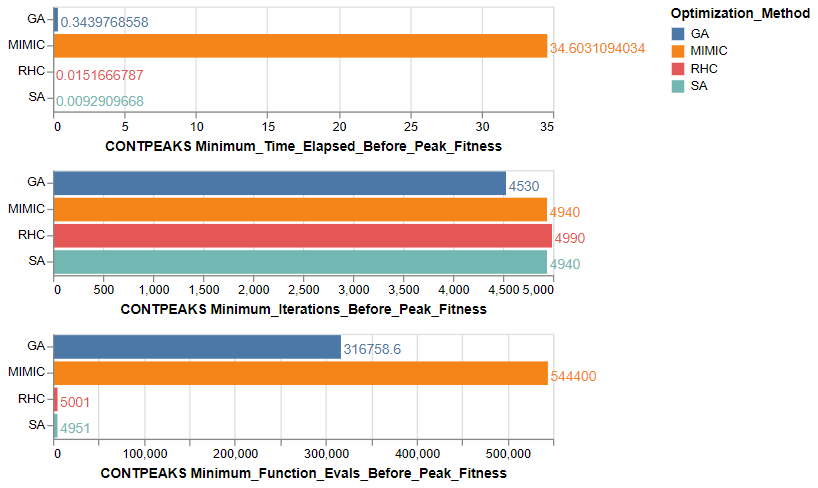
Of all the optimization problems examined, the traveling salesman is the most complex. In terms of fitness performance, the genetic algorithm far exceeded all others. In terms of complexity, measured in both wall time elapsed and function evaluations a common pattern emerged: MIMIC was far and away the most computationally expensive and complex while GA fell somewhere in the middle while RHC/SA were cheap overall:

This trend continues throughout the other problems. In terms of time elapsed MIMIC is nearly 500x greater than both GA/SA/RHC. For fitness function evaluations MIMIC is 100x more than RHC/SA while only about 2.5x that of GA. Here lies both the strength and weakness of MIMIC. As mentioned in the lectures, MIMIC concerns itself just as much with the journey towards the optima as it does the optima itself. For a problem like TSP the information contained in its sampled distributions may provide insight into the problem since the problem structure is of importance (the pairs of cities and their respective distances). Frankly, I am surprised that MIMIC was unable to perform better than RHC/SA but it’s possible the reason why it failed is the same for why GA performed well. With the traveling salesman problem, many optimal solutions come from correctly identifying sub structures within the data of cities that should be traveled to in clusters. This would suggest that the GA could perform well paired up cities relatively close to each other between generations and moved in that direction, which would be supported by the parameters selected for the best performing (top 10% kept for mating and top 10% mutated, i.e. only top cities used for future search area). An interesting note is that the fitness curve for GA is relatively turbulent, suggesting that at times pairs of cities were created as offspring that performed poorly (see the minor dips in the curve) but eventually the correct pairings were found. The GA also did not peak at the full 3000 iterations but found its optima at 2500. Perhaps for iterations 2500-3000 the GA was exploring a region of cities and had yet to find the correct pairings to exploit. With more iterations it is possible the fitness would continue to improve. The ability to explore many potential pairings and adjust accordingly is the key reason GA performed well while RHC/SA did not. RHC/SA only concerns itself with one point in the future and not the relationship between the points like GA does. For this reason, it’s understandable that RHC/SA may make progress in increasing fitness value but not find an optimal solution. The fact that RHC and performed nearly identically but not with great overall fitness would point to the fact that the fitness function is not smooth and didn’t have too many local optima (hence why SA barely outperformed RHC). SA’s parameter of a very small cooling rate would suggest it liberally moved around as it kept a high Temperature and only slowly cooled to settle closer to RHC. To improve the performance of MIMIC/RHC/SA would be difficult as the TSP is a hard problem to solve. Perhaps running SA with a low cooling rate thousands of times and looking for sub clusters of well performing cities would hint towards which clusters of cities to visit in tandem. For MIMIC focusing only on the best performing samples (keep top 5-10%) would similarly hone in on key clusters of cities to visit together. The difficulty after identifying these subclusters is the order in which to visit them, which is basically a simplified version of the TSP with less cities that are more distinct in distances. However, since TSP is a combinatorial problem, any reduction in inputs is a vast improvement.

***Continuous Peaks***

|  |  |  |  |
| --- | --- | --- | --- |
| **Optimization\_Problem** | **Optimization\_Method** | **Parameters** | **Peak\_Fitness** |
| CONTPEAKS | GA | 100\_50\_50 | 90.6 |
| CONTPEAKS | MIMIC | 100\_50\_0.7 | 79.6 |
| CONTPEAKS | RHC |  | 95.3 |
| CONTPEAKS | SA | 0.95 | 98.8 |

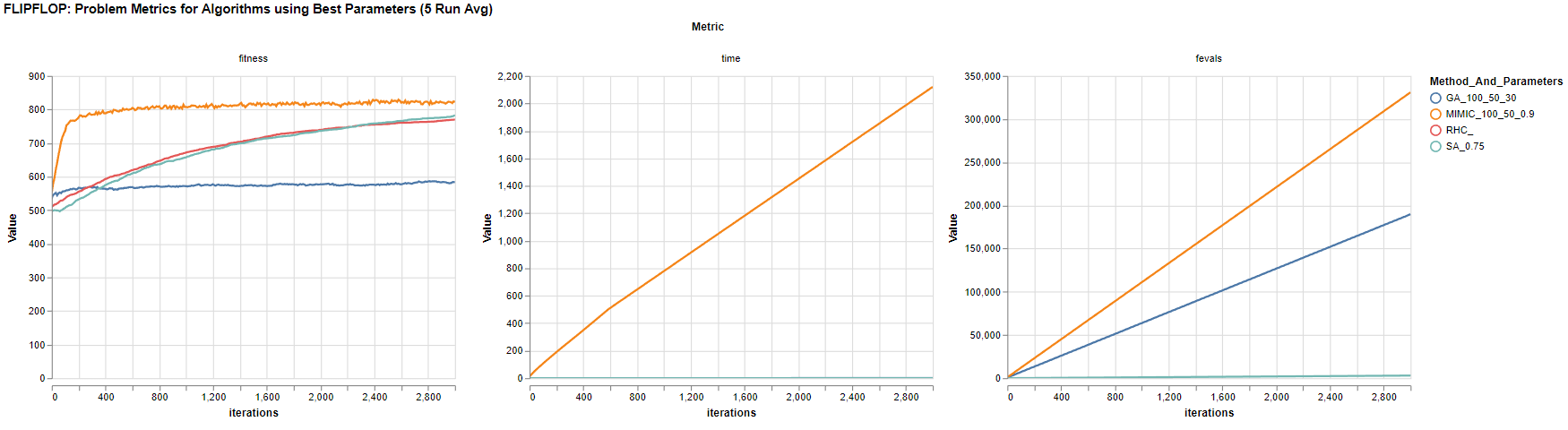
# 

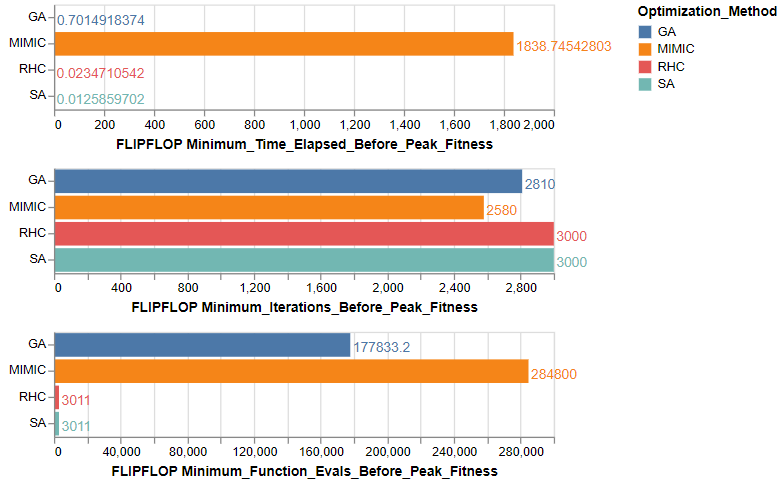
RHC/SA’s time to shine. It isn’t surprising that a method with ‘Hill Climbing’ in its name performs well on a ‘Peaks’ problem. RHC and SA outperformed all other algorithms in this domain, and once again did so in a fraction of the computational complexity, as seen in the curves above and graphs below:

RHC/SA found its optimal solution nearly instantly (<1 s) compared to the 35s it took for MIMIC. Once again GA split the difference with a quick time but in terms of function evaluations was in between RHC/SA and MIMIC. SA outperformed RHC by a slight margin with a large cooling rate (.95) which suggests that the true fitness may have some local optima that required ‘hopping’ out of with a high temperature but was smooth enough to not require many of these hops (thus decreasing the temperature quickly with a large cooling rate). Perhaps the most important aspect of the performance is that while GA and MIMIC appear to reach convergence by leveling out in their fitness/iterations curve, SA/RHC is still improving. With more iterations it is possible that RHC/SA would find the true global optima (if they have not already). The difference in those curves between GA/MIMIC and RHC/SA is interesting. MIMIC/GA make quick gains in fitness and it is not until after 70% of the iterations (3500) that RHC/SA passes GA in fitness value. The best hyperparameters selected for GA and MIMIC (number of mates/mutation for GA, m previous probability to retain for MIMIC) are both in the higher end of the ranges tested, suggesting that their best performance was found by keeping their search space general and wide. This contrasts their respective approaches: GA/MIMIC looks at the entire population for guidance towards the optima while RHC/SA only concerns itself with single samples (neighbors). This is akin to scanning the horizon vs. developing tunnel vision when choosing a direction to move. To possibly improve GA/MIMIC in this problem domain we could move to an extremely wide horizon and after so many iterations alter the parameters to hone in and develop tunnel vision. In other words, try to add the SA philosophy to these methods. Perhaps by having GA and MIMIC consider larger populations/mutations where applicable and after a certain number of iterations change to more restrictive parameters. This would allow them to move past local optima by heading the general correct direction before honing in on a single direction to move a la SA.

***Flip Flop***

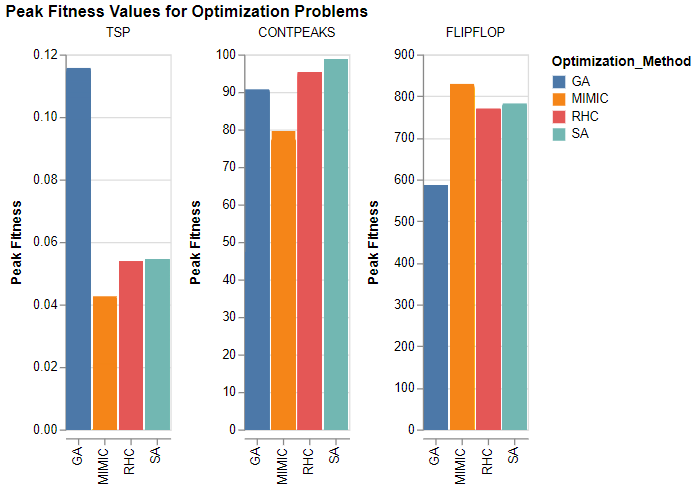
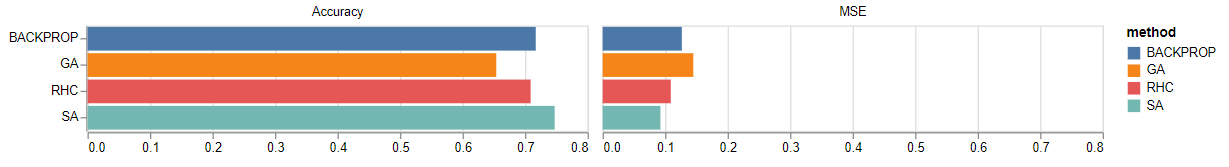
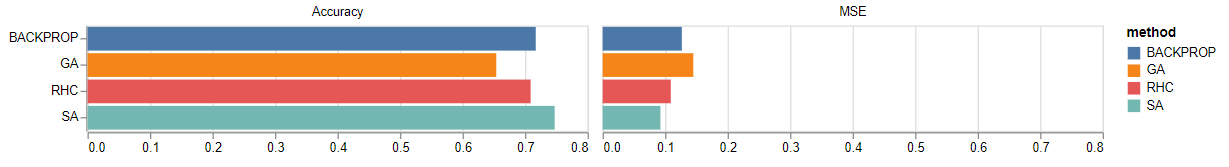
|  |  |  |  |
| --- | --- | --- | --- |
| **Optimization\_Problem** | **Optimization\_Method** | **Parameters** | **Peak\_Fitness** |
| FLIPFLOP | GA | 100\_50\_30 | 586.6 |
| FLIPFLOP | MIMIC | 100\_50\_0.9 | 829.4 |
| FLIPFLOP | RHC |  | 770.2 |
| FLIPFLOP | SA | 0.75 | 782.6 |

Finally, MIMIC’s time to shine. It appears that MIMIC was very well suited for the FLIPFLOP problem as it vastly outperformed GA while also topping RHC/SA. Not only did it outperform the others, but after roughly only 100 iterations had achieved a fitness that would outscore all other algorithms. The takeaway for MIMIC’s success: structure, structure, structure. As previously described, the FLIPFLOP problem is one designed so that each subspace (bit) is independent from all others. It’s not surprising that MIMIC with its use of dependency trees could decompose this problem simply: each bit is probabilistically independent of all others, and once this is established its rather simple to flip bits and see if the fitness went up or down. RHC/SA improved slowly, likely by changing bits and moving if it got more of the changes correct than incorrect (hence it’s smooth, incremental improvement). GA performed poorly since the bits were entirely independent of each other and any progress made would likely come from mating of bits that were already correct without a change mutated. The previous computational costs hold true with RHC/SA cheap, GA in the middle and MIMIC expensive. However, MIMIC reached its peak fitness before the full 3000 iterations:



# Additionally as was previously mentioned MIMIC achieved most of its gains in the first 200 iterations. If it stopped at that point it still would have put it above the peak of RHC/SA/GA and orders of magnitude less iterations (x15 less). This shows the particular type of problem MIMIC excels at: structured problems where conditional probability of that structure is key to finding an optimal solution. GA could have improved by implementing uniform crossover and flipping each bit independently of all others and keeping it the same if it improved fitness. SA/RHC would be tough to improve as it continually headed in the right direction, although increasing the cooling rate to maximum would reduce SA to become RHC which in this case is better suited for the problem. With enough iterations, it is likely SA/RHC may reach optimal fitness, but the upper bound is 2^N (N being the length of the string) options and SA/GA may not outperform that in the worst case scenario.

# Concluding Remarks



These graphs summarize the strengths of each optimization method. Like backprop, RHC/SA work well with smooth continuous fitness functions. For those with local optima SA works better than RHC and problems with more local optima require a slower cooling rate to allow for more ‘basin hopping’. Despite RHC/SA outperforming backprop the computational cheapness of backprop makes it the standard for NN training. For similar problems that may not be easily differentiable, RHC/SA may be a viable option. The Continuous Peaks further illustrated the strength of RHC/SA on smooth continuous functions. For complex problems where the relationship between inputs and the optimal solution may be complex and difficult to solve for, GA with some domain knowledge of the problem can be good options. This was evidenced by the complex Traveling Salesman problem. While MIMIC is much more computationally expensive to compute, given the right circumstances (e.g. when the inputs have conditional dependencies and understanding the structure of the entire problem is important to know), it can outperform other algorithms in far less iterations.