Name: Heng Chang Rong Kelvin
Course: CS7641
Assignment: 2

Randomized Optimization Analysis

Introduction

The goal of this assignment is to design 3 optimization problems to highlight the advantages of Simulated Annealing, Genetic Algorithm and Mutual-Information-Maximizing Input Clustering. The problems chosen are 8-Queens, Flip Flop and Knapsack and all implementations are done with ML Rose library.

Optimization Algorithms

In this section, we will touch on the different types of aforementioned optimization algorithms.

Randomized Hill-Climbing (RHC)

Randomized hill climbing is a randomized optimization technique which searches for a more optimal neighbour by moving in the gradient direction until it reaches a peak. Occasionally, the algorithm will randomize its original state to avoid falling into a local optimum. Increasing the number of restarts will allow a more thorough search of the state space, but result in a longer search time.

Simulated Annealing (SA)

Simulated Annealing is similar to RHC except that it makes a move probabilistically based on the temperature hyperparameter. A higher temperature increases the probability of exploring its neighbour states with lower fitnesses, whereas a lower temperature will tend to shift to only neighbours with high fitnesses.

Genetic Algorithm (GA)

Genetic Algorithm tackles an optimization problem through inspiration from natural evolution such as mutation and crossover. Genetic Algorithms usually initialize a random population and eliminate a subset of the population with insufficient fitness and does mutation and crossover for the rest of the population. However, GA does not scale with increasing complexity.

Mutual-Information-Maximizing Input Clustering (MIMIC)

MIMIC searches for optimal state through estimation of probability distributions. MIMIC tends to leverage on the known structure of a problem to do randomized search. This will allow the information about fitness generated by previous iterations to populate through time to generate better subsequent samples.

Methodology

For each problem and optimization algorithm, we conduct hyperparameters tuning to get optimal results. These runs are then averaged across 10 different runs to obtain the best fitness, number of fitness evaluations and time taken to run each time. Finally, we will compare each optimization algorithm to highlight the advantages for each problem.

For the optimization algorithms, we tuned the following hyperparameters for each problem:

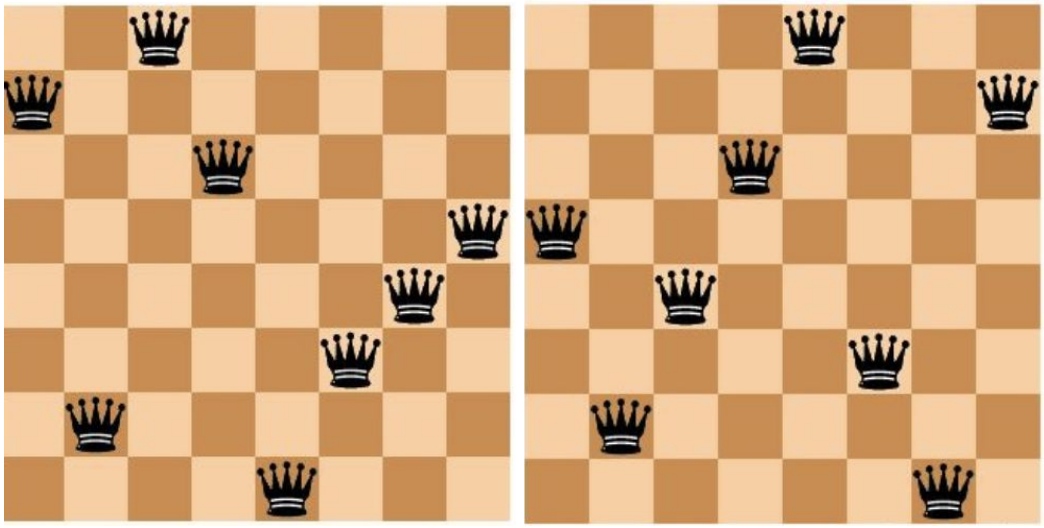
Algorithm	Parameters
Random Hill Climbing	Restarts: [10,50,100]
Simulated Annealing	Exponential constant: [0.001, 0.005, 0.01] Initial temperature: [0.1, 0.5, 1.0]
Genetic Algorithm	Population size: [50, 100, 200] Mutation probability: [0.2, 0.4, 0.6, 0.8]
Mutual-Information-Maximizing Input Clustering	Population size: [50, 100, 200] Keep percentage: [0.01, 0.05, 0.1, 0.5]

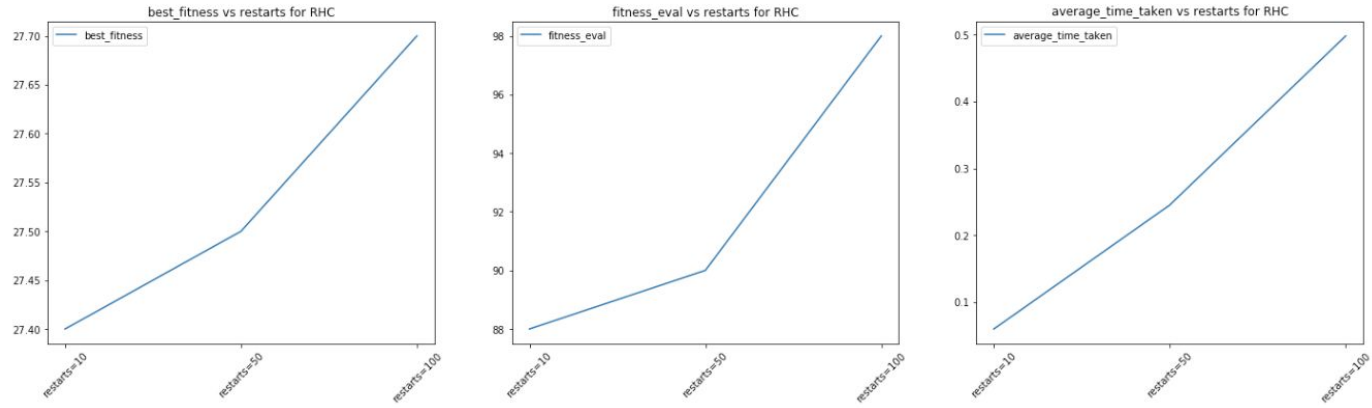
Problems

8-Queens

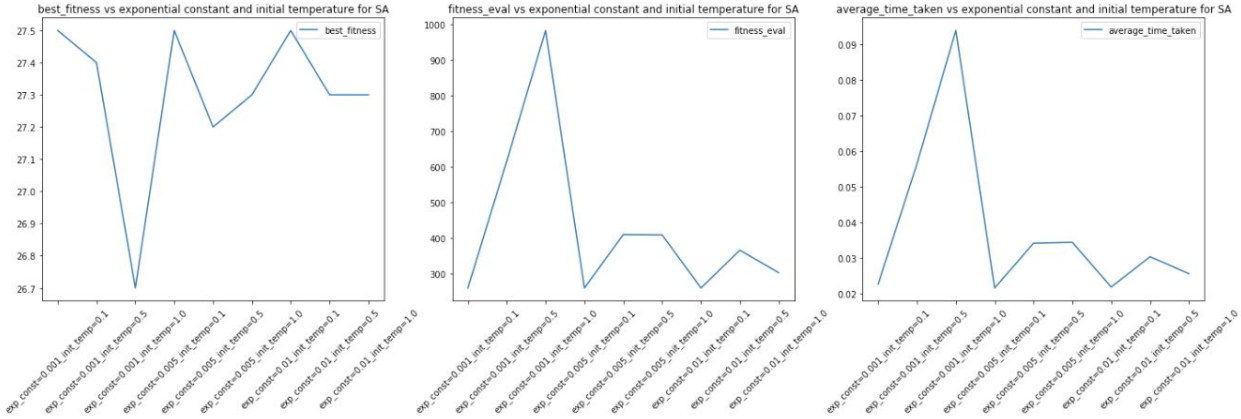
In the context of the 8-Queens problem, our goal is to find a state vector representing the positions of the Queens such that no pairs of queens can attack each other horizontally, vertically or diagonally in an 8*8 chessboard.

From the figure below, the chessboard on the left is suboptimal because the Queen on the extreme right can attack the other queens towards its left bottom diagonal. However, the chessboard on the right shows an example of optimality.

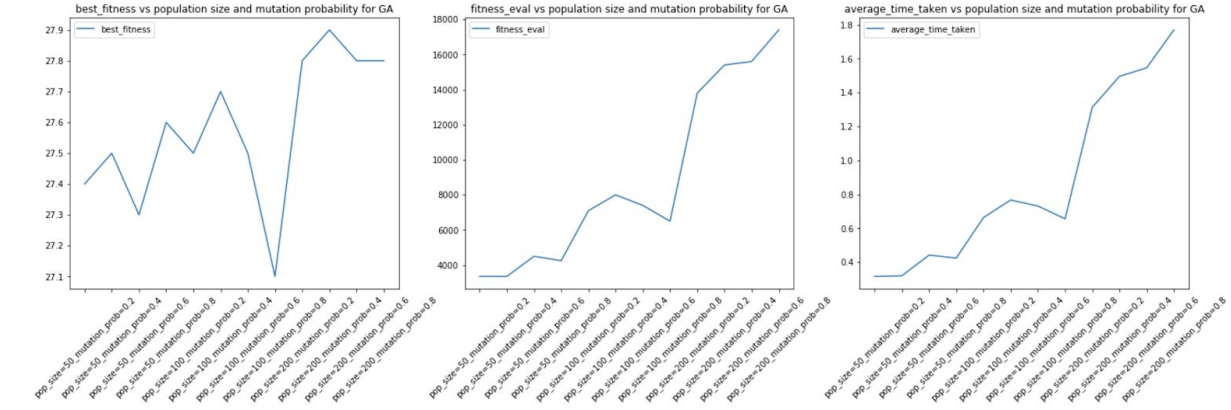




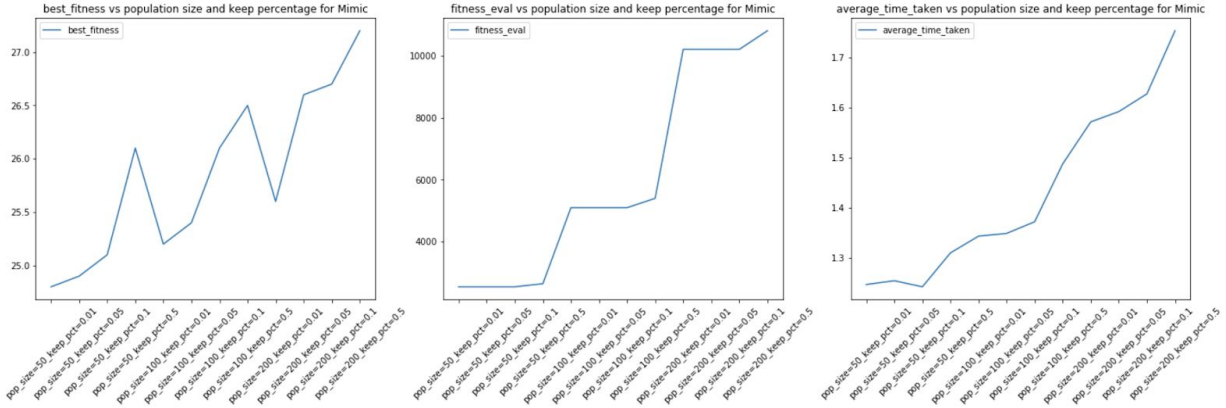
Referring to the chart of RHC above, we observed that best fitness, number of fitness evaluations and time taken for RHC algorithm to run increases with more restarts. With more restarts, the algorithm will allow us to explore more of the state space, but reverting back to a random original state each time, in the hope of finding a better global optimum.



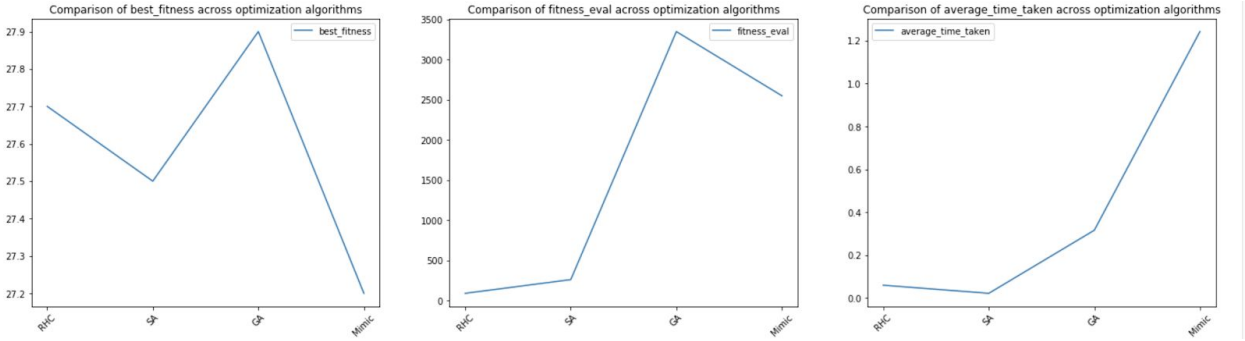
Referring to the chart of SA above, we observed that we don't see a clear pattern of best fitness by varying the parameters. This could be due to the fact that the random seed already results in good initial states. Intuitively, the fitness evaluations graph also shows the same pattern as time taken graph. Furthermore, we noticed that as initial temperature increases, the number of fitness evaluations increases due to more aggressive random explorations as well.



From the charts of GA, we observe that there is a general trend towards higher fitness, number of fitness evaluations and time taken. The trend of higher fitness is because we have more accurate average information when we increase the population size. Obviously the number of fitness evaluations and time taken will increase due to population size and mutation probability as well.



The charts of MIMIC are very similar to that of GA above. In general, the increasing population size definitely helps to propagate information about the calculated fitnesses across time.



From the charts above, we can tell that this problem was chosen to highlight the advantages of GA, which gives us a fitness of 27.9 out of 28.0 (8 choose 2), much higher than its peers. However, that also comes at a cost of high number of fitness evaluation and longer time taken.

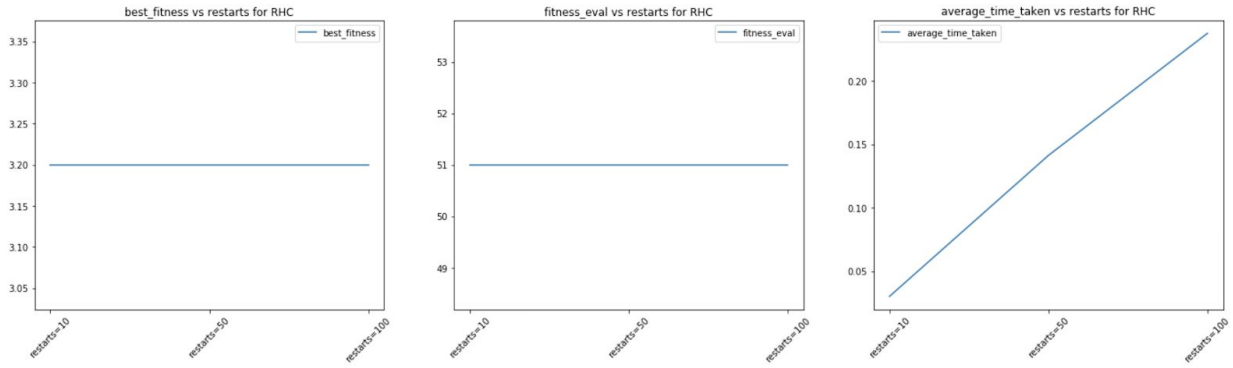
The reason that RHC and SA doesn't do that well as compared to GA is because they only consider neighbouring states as candidates for searching. resulting in lower overall fitness.
MIMIC also doesn't work well because it attempts to exploit the underlying structure of the 8-Queens problem to prevent exploration of previously identified sub-optimal solution space for future iterations. To summarize, RHC, SA and MIMIC does not exactly work well for 8-Queens because the solution space is not convex, i.e if we move the position of any Queen, that Queen may be in conflict with another Queen. GA, on the other hand allows us to crossover pairs of state space with good fitness, i.e two different states with only the same Queen conflicts.

Knapsack

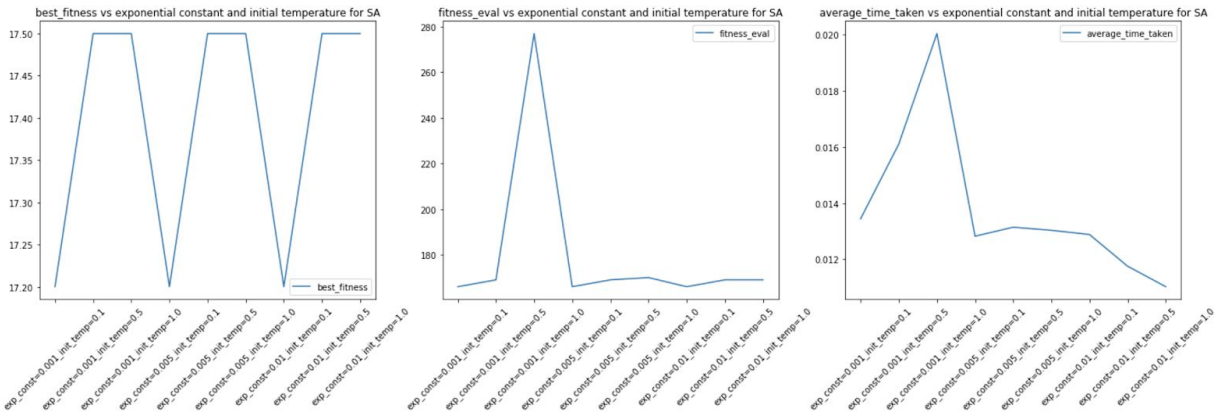
Knapsack is a NP-hard constrained combinatorial optimization problem with no analytical solution. Given K items of each type, $k_1, k_2, .., k_n$ with weights $w_1, w_2, .., w_n$, values $v_1, v_2, .., v_n$, and threshold W, we are supposed to find the number of each item to include in a backpacker's bag while maximizing total utility via maximizing the following fitness function:

$\sum_{i=1}^{n} k_i v_i$, subject to $\sum_{i=1}^{n} k_i w_i < W$ and $\sum_{i=1}^{n}$, and 0 otherwise.

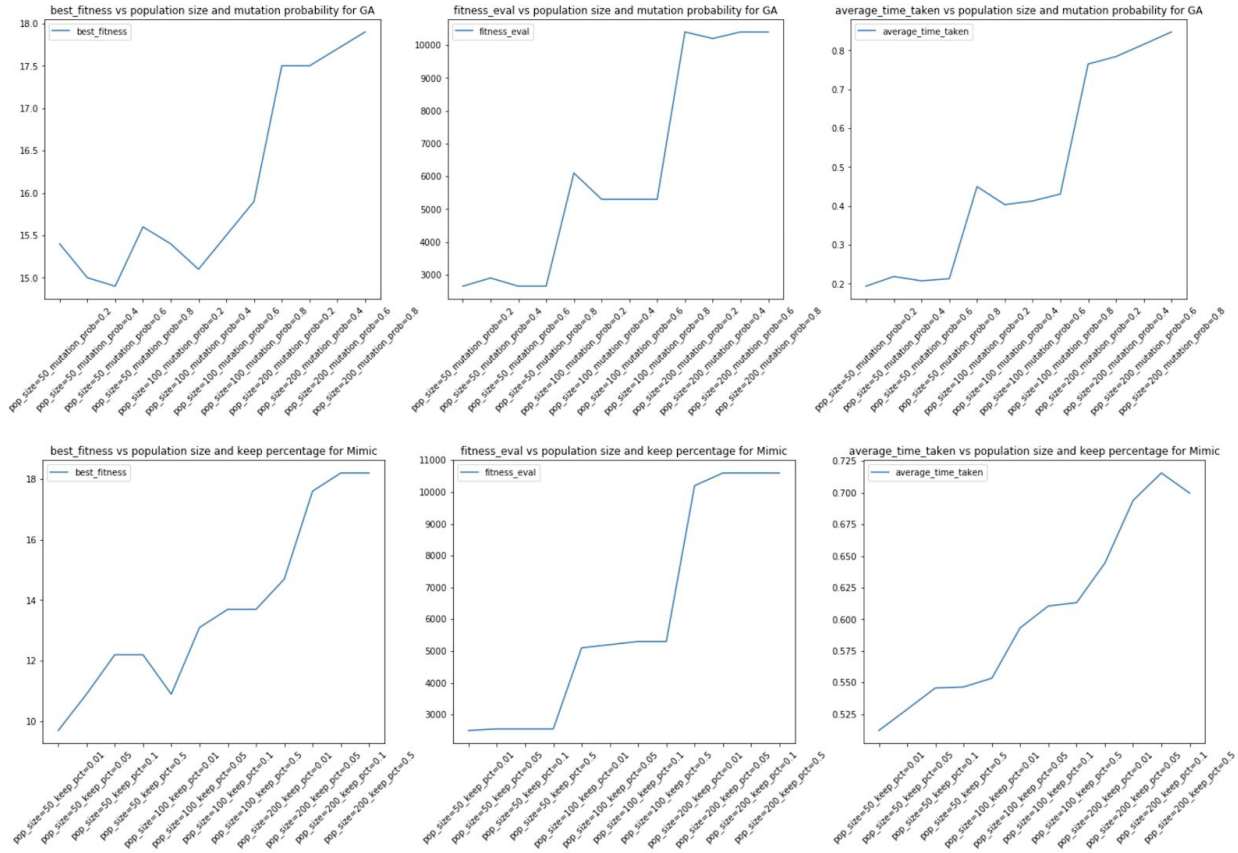
In our case, we use max_weight_pct = 0.6.



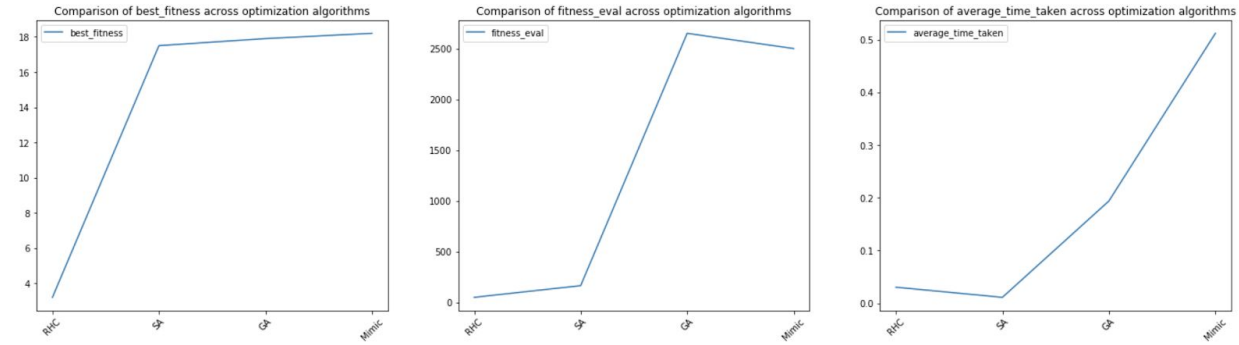
Referring to the charts above, RHC's best fitness and number of fitness evaluations doesn't seem to be affected much by the number of restarts. This is because there are simply too many combinations of items for knapsack, and we may end up in a bad local optimum without sufficiently many restarts.



Referring to the SA charts above, the only interesting thing to note is that the parameters exp_const=0.001, init_temp=1.0 have the same fitness as other parameters, but much higher number of fitness evaluations and time taken. Perhaps it spent more time exploring rather than optimizing with its aggressive explorations.



Since the GA and MIMIC charts above are similar to that observed in the 8-Queens problem. The explanation can be found in that section.

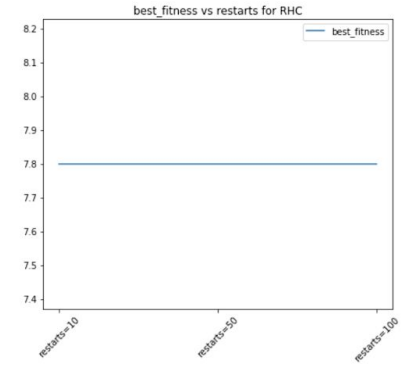


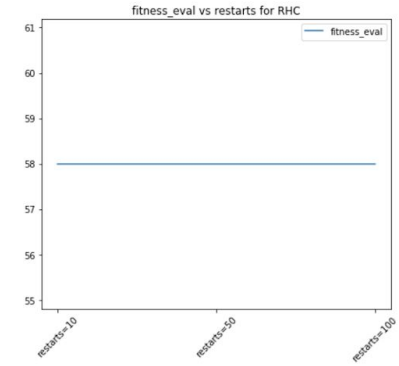
From the charts above, we can tell that this problem was chosen to highlight the advantages of MIMIC, which gives us a fitness of 18, slightly better than GA. SA and RHC performed poorly and did not converge.

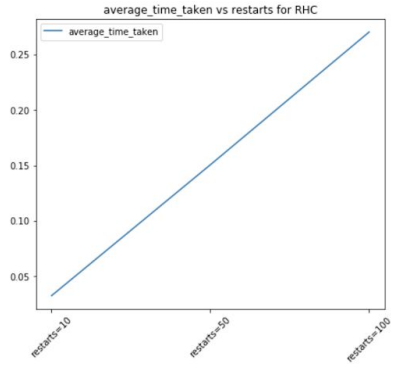
RHC and SA do not perform well on Knapsack because of their greedy nature. If we only look for a better neighbour, we will find that further iterations may violate the constraints. Since knapsack will benefit from retaining history and propagating the previously evaluated fitnesses to future iterations, MIMIC is the ideal choice for this problem.

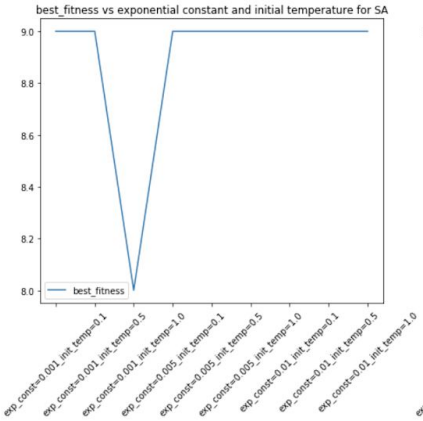
Flip Flop

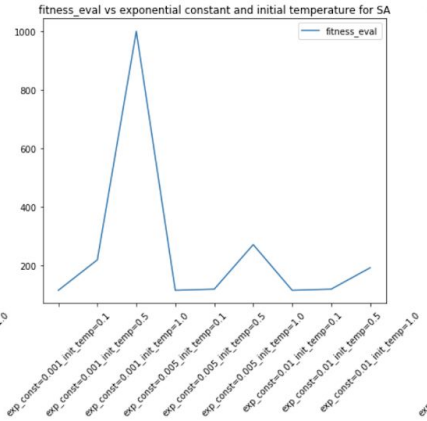
Flip Flop is a simple optimization problem which evaluates the fitness of a state vector x as the total number of pairs of consecutive elements of x, (x_i and x_{i+1}) where $x_{i+1} != x_{i+1}$. The optimal solution is always one less than the length of the state vector. For our experiment, we used a state vector of 10 binary bits elements.

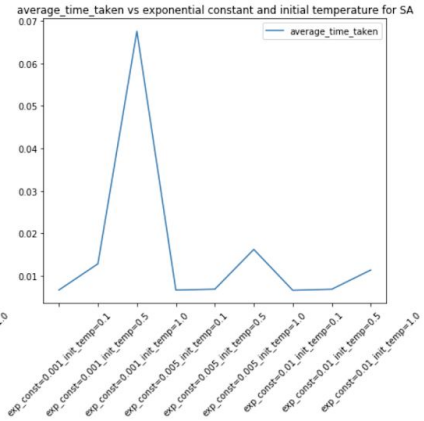


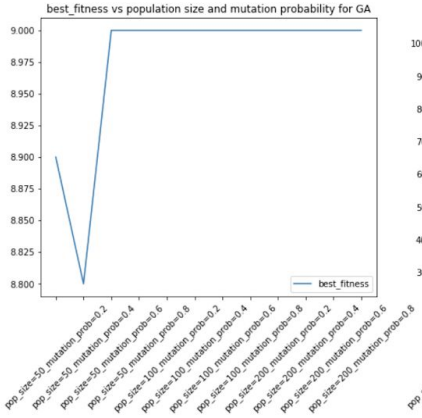


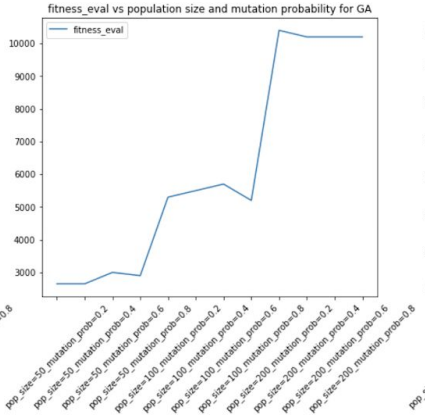


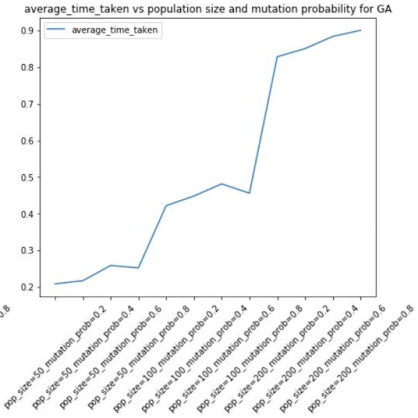


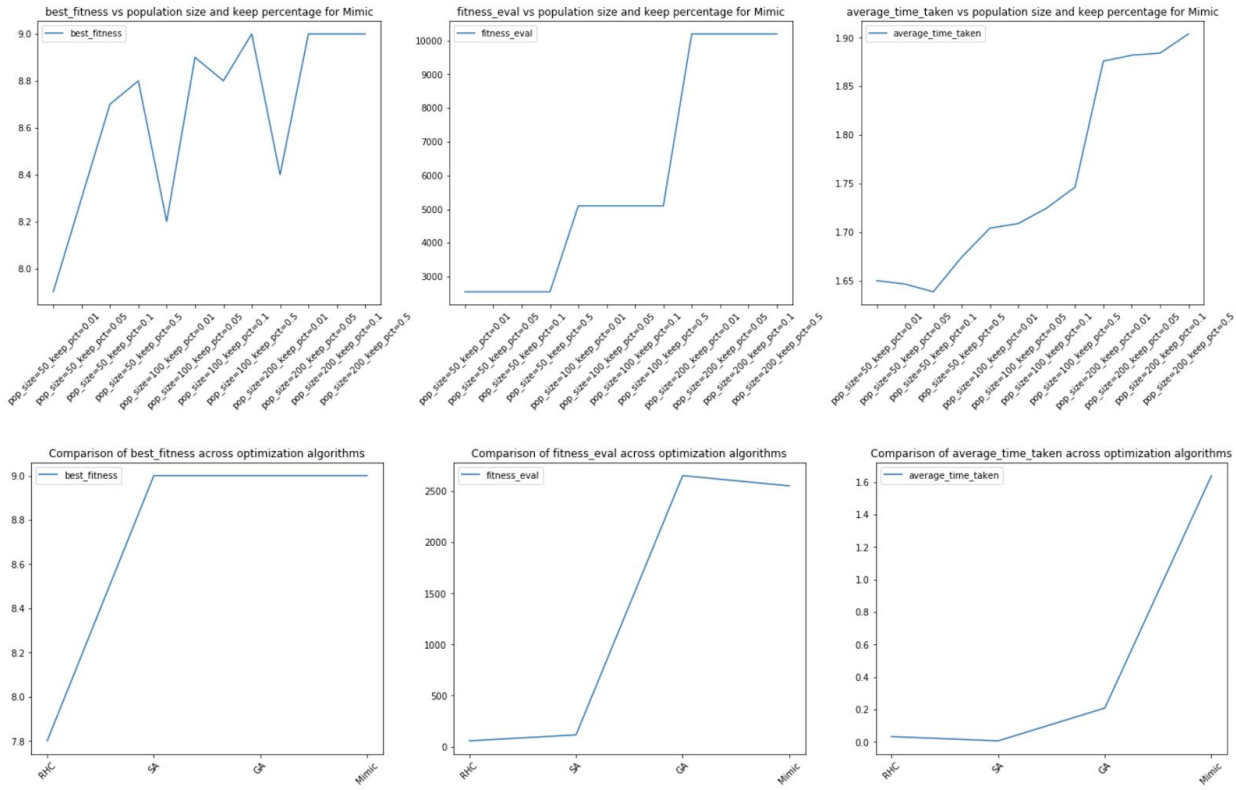












Since the trend in the charts are similar to the previous sections, we can refer to the explanations above other than a few observations. Without sufficient restarts, we notice that RHC only achieve a fitness of 7.8 out of 9.0. SA, GA and MIMIC are able to achieve optimal fitness with some hyperparameter tuning. SA and GA almost consistently achieve optimal fitness. However, this problem was chosen to highlight the advantages of SA, which achieves optimal fitness consistently with relatively few fitness evaluations and computation time.

Neural Network (NN)

Introduction

This goal of this section is to compare RHC, SA and GA with the Gradient Descent optimization method used in the contraceptives dataset of assignment 1.

Dataset

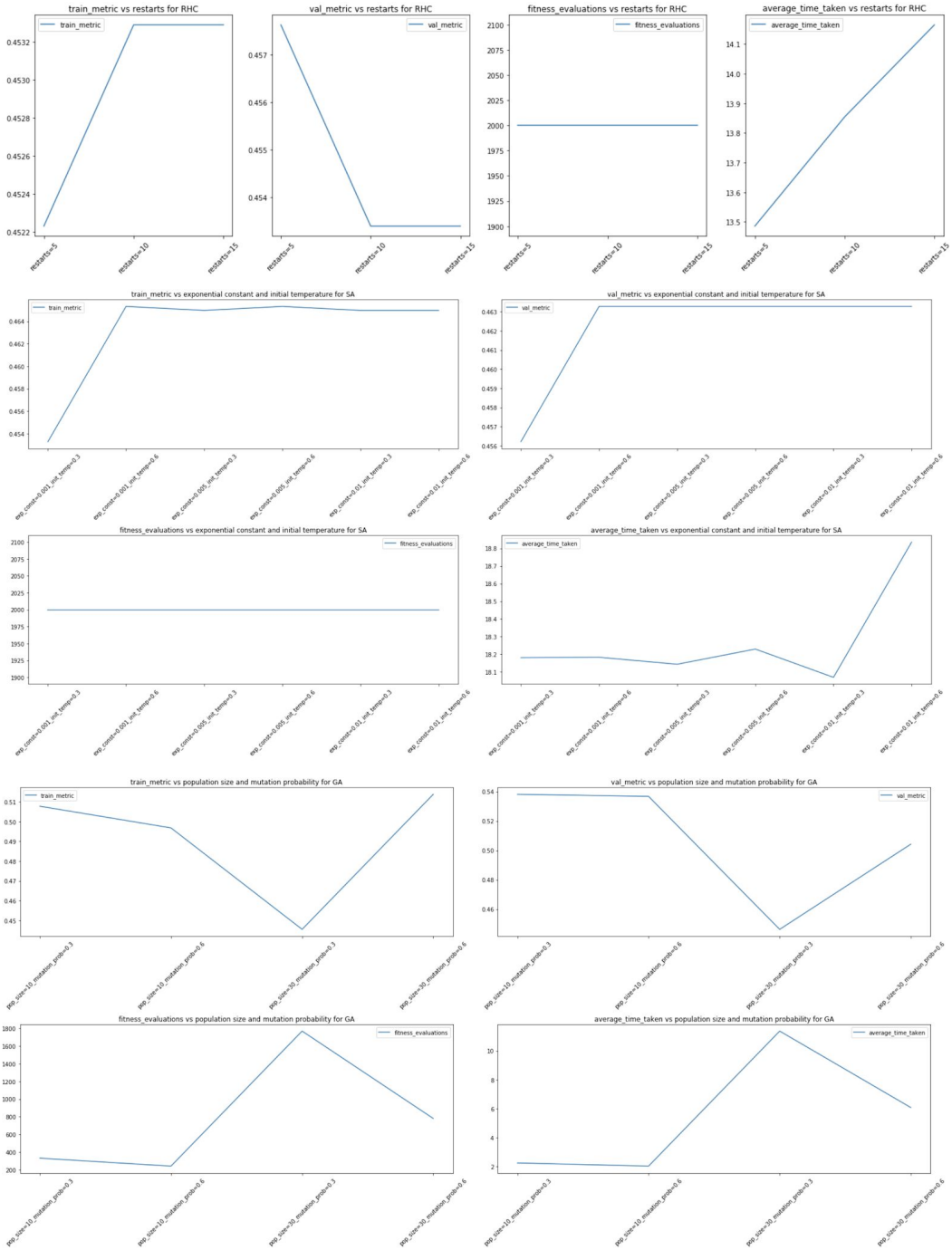
The contraceptives dataset helps to determine whether contraceptives are being used based on many attributes of the husband and wife. Modelling birth rates, sales of contraceptives and healthcare related applications are some of the interesting use cases from solving this dataset. The preprocessing done is very similar to that of assignment 1.

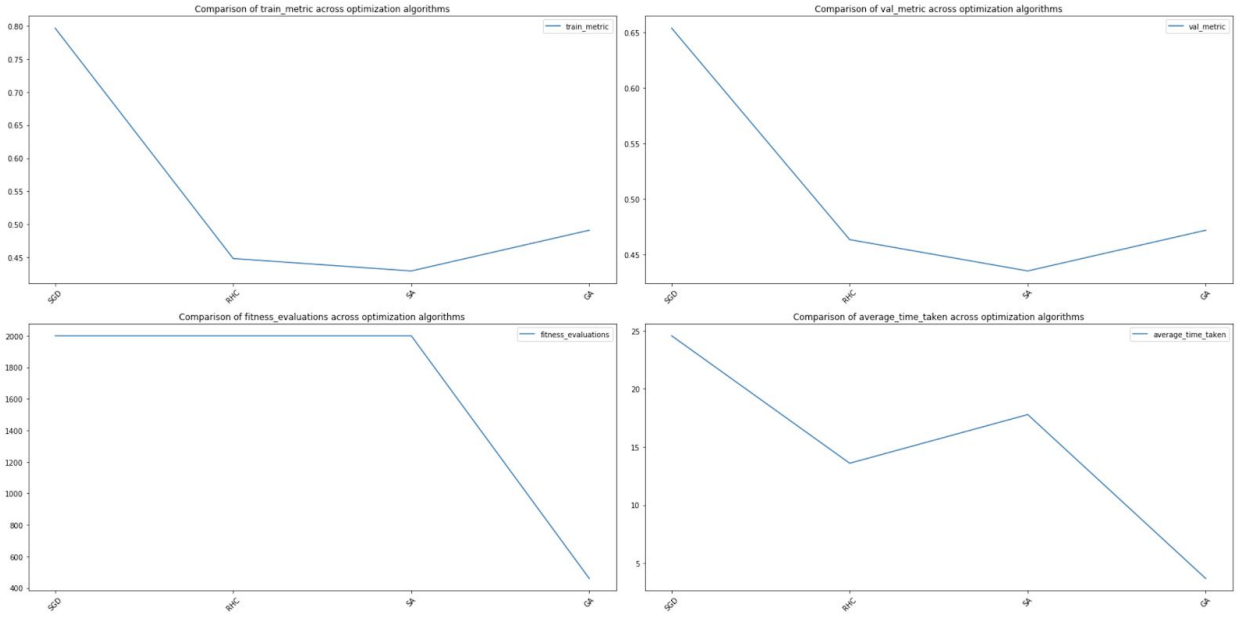
Methodology

For each optimization algorithm, we conduct hyperparameters tuning to get optimal results. The metric we have chosen is accuracy since the data set is well balanced. We also ensured that the NN architecture is consistent across each experiment. These runs are then averaged across 3 different runs to obtain the best fitness, number of fitness evaluations and time taken to run each time. The hyperparameters we tuned for each optimization algorithm are very similar to the previous sections.

Train Accuracy	Validation Accuracy	Number of fitness evaluations	Time taken (s)
81.7%	63.7%	2000	23.9

We choose to rerun the NN with Gradient Descent from assignment 1 for experimental completeness. The above table are the reported baseline results from Gradient Descent.





From the table above, we will use the Gradient Descent method as our baseline with validation accuracy of 63.7%, 2000 fitness evaluations and 24 seconds run time. From the validation accuracy chart above, Gradient Descent algorithm did result in high variance and bias. We also observed that the RHC, SA and GA did not perform better than Gradient Descent. With the same network architecture and common hyperparameter settings, it seems that RHC, SA and GA did not converge. Perhaps we can use different hyperparameters such as learning rate and activation function to allow convergence for RHC, SA and GA. However, that may not give the same ground for comparison. The reason why Gradient Descent performed so well may be due to the nature of its closed form solution, as opposed to the extreme probabilistic nature of RHC, SA and GA, which may require more iterations and different parameters to converge.

Conclusion

The assignment involved various random optimization on different problems. The analysis really challenged my initial understanding of optimization algorithms. In general, we observed that each optimization method has its own advantages in tackling different problems. The neural network optimization analysis is by far the most interesting. It shows that the industry standard of back-propagation via Gradient Descent works much better than RHC, SA and GA. Intuitively, that makes sense because Gradient Descent functions as a closed form solution given sufficiently huge batch size while RHC, SA and GA has a strong probabilistic nature.

References

[1] ML Rose: https://github.com/hiive/mlrose
[2] Mimic paper: https://www.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf