MACHINE LEARNING – OMSCS7641
SPRING 2022
RANDOMIZED OPTIMIZATION

PART 1

In this project, I study randomized optimization algorithms such as random hill climbing (RHL), simulated annealing (SA), genetic algorithms (GA) and mutual-information maximizing input clustering (MIMIC). I study the n-queens, 4-peaks and knapsnack problems. While RHL and SA are better at solving simpler concave problems (n-queens), generic algorithms are able to avoid the local maxima. Therefore, it works better for 4-peaks. MIMIC can benefit by identifying dependency trees such as the knapsack problem.

N-Queens Problem

N-queens aims to place the queens in a n*n board such that no queen hits another queen in any horizontal or diagonal direction.

Here, I optimize the simulated annealing parameters. Max_attemps is a parameter that determines the maximum number of attemts to find a better neighbour. I noticed that a low number for max_attempts runs fast but gives a lower fitness score. For example, for n_queens with 90 queens, max_attempts values of 1,3, 10 and 20 give 1500, 3400, 4000, and 4000 fitness scores; and 0.22, 0.75, 2.5 and 2.5 seconds computation time, respectively. As I did not see any improvement after max_attempts value of 10, I set this value as 10. max_iters is the parameter that determines the maximum number of iterations for convergence. I noticed it uses several hundreds of iterations for each problem size, hence I set max_iters to 1000. Initial random_state is fixed to 1 for reproducilibity. init_state is basically an integer array of [0,n] where n is the number of queens. For schedule, I tested geometric decay, arithmetic decay and exponential decay with their default values. As arithmetic decay is linear, it takes much longer for its temperature to cool down. Therefore, it takes more iterations and time to converge without any improvement in the fitness score. Geometric decay and exponential decay are very similar mathematically and they performed similar in the plots as well. However, geometric decay performed slightly better. Therefore, I chose geometric decay for schedule. I chose the default fitness function for Queens, which is a minimization problem. However, I plot the negative of the fitness scores in the plot to make it a maximization problem.

I had a similar approach with RHL where I made sure maximum number of iterations were not constrained for any problem size. I chose 10 restarts as there were no improvements above this value. The optimum values for GA were population size of 200, mutation probability of 0.1, and max_attempts of 10. For MIMIC, the optimum parameter are population size of 100, keep_pct of 0.2, max_attempts of 5. It didn't require many iterations. However, MIMIC gets very slow as the population size and maximum attempts increase. Therefore, I could not increase those values a lot.

Figure 1 shows the number of iterations, fitness and computational time vs the problem size for each algorithm. Problem is varied from n = [10,100] where n is the number of queens. As n-queens is a simpler problem, SA gives a good fitness score as shown in Figure 1 (top right). Even though SA requires many

more iterations, iterations are computed fast, and the total computational time for SA is the least. On the bottom right of Figure 1, we can see the fitness curve for four algorithms when n =60. We can see that SA goes to the best fitness value even though it requires more iterations.

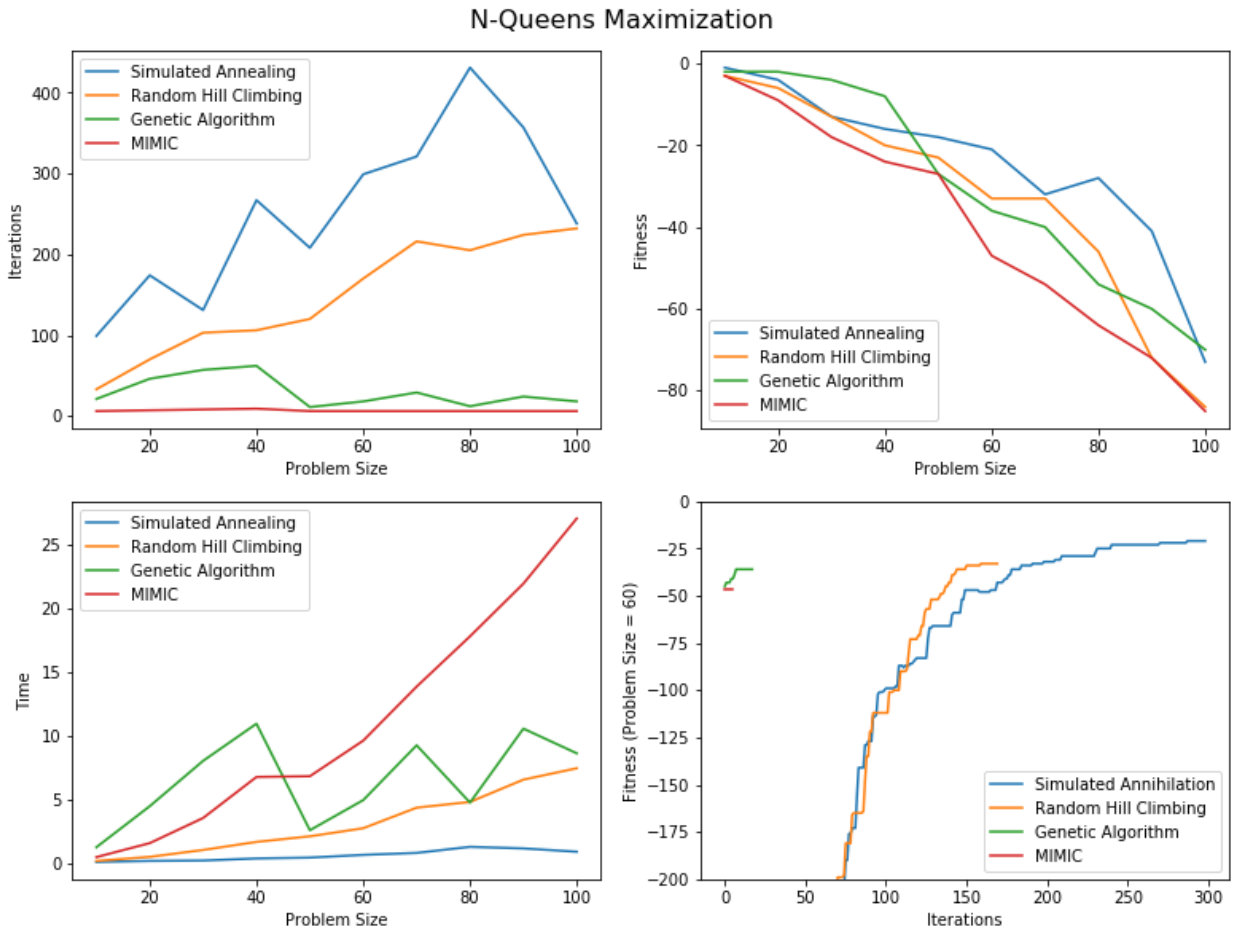

Figure 1. Top left: Iterations vs problem size. Top right: Fitness vs problem size. Bottom left: Computational time vs problem size. Bottom right: Fitness vs iterations when problem size is 60.

4-Peaks Problem

4 peaks problem has two local optima with wide range and two global optima close to the edges. SA and RHC tend to get stuck in the local optima. However, genetic algorithm mutates and crosses over in different dimensions at the same time in parallel. Therefore, it can experiment with different part of the problem space to find the global maximum.

For fitness function, I use the default fitness function of 4-peaks with t_pct of 0.15. The problem is a discrete optimization problem with max_val of 2 (as specified in the manual). For SA, I increased max_iters to 3000 as it wasn't converging very easily. max_attempts is kept at 10 to keep a balance between runtime and optimization. FOR RHL, 1000 iterations were enough with 10 restarts. For GA, I vary the poputation size between 100 and 300. At 100 population size, fitness score is low for larger problem set. Above 200, there is no improvement in fitness score. Therefore, the optimal parameters for GA are population size of 200, mutation_prob of 0.1, max_attempts of 10. The other parameters were also tuned with a similar

mindset. Larger population size and max_attempts will increase the likelihood of better fitness at the expense of computational time. For MIMIC, I lowered the population size to 50 as it was too costly. I also kept keep_pct at 0.1 and max_attempts at 5. MIMIC is not constrained with the number of iterations. For the problems, I fix the seed for reproducibility.

Figure 2 shows the iterations, fitness and computation time vs problem size. It can be seen that the fitness score of GA is much higher than other algorithms particularly at larger problem size. Computational time for GA is low. We can also see the fitness score vs iterations for problem set size 60 on the bottom right of Figure 2. We can see that GA converges rather quickly to much higher fitness score.

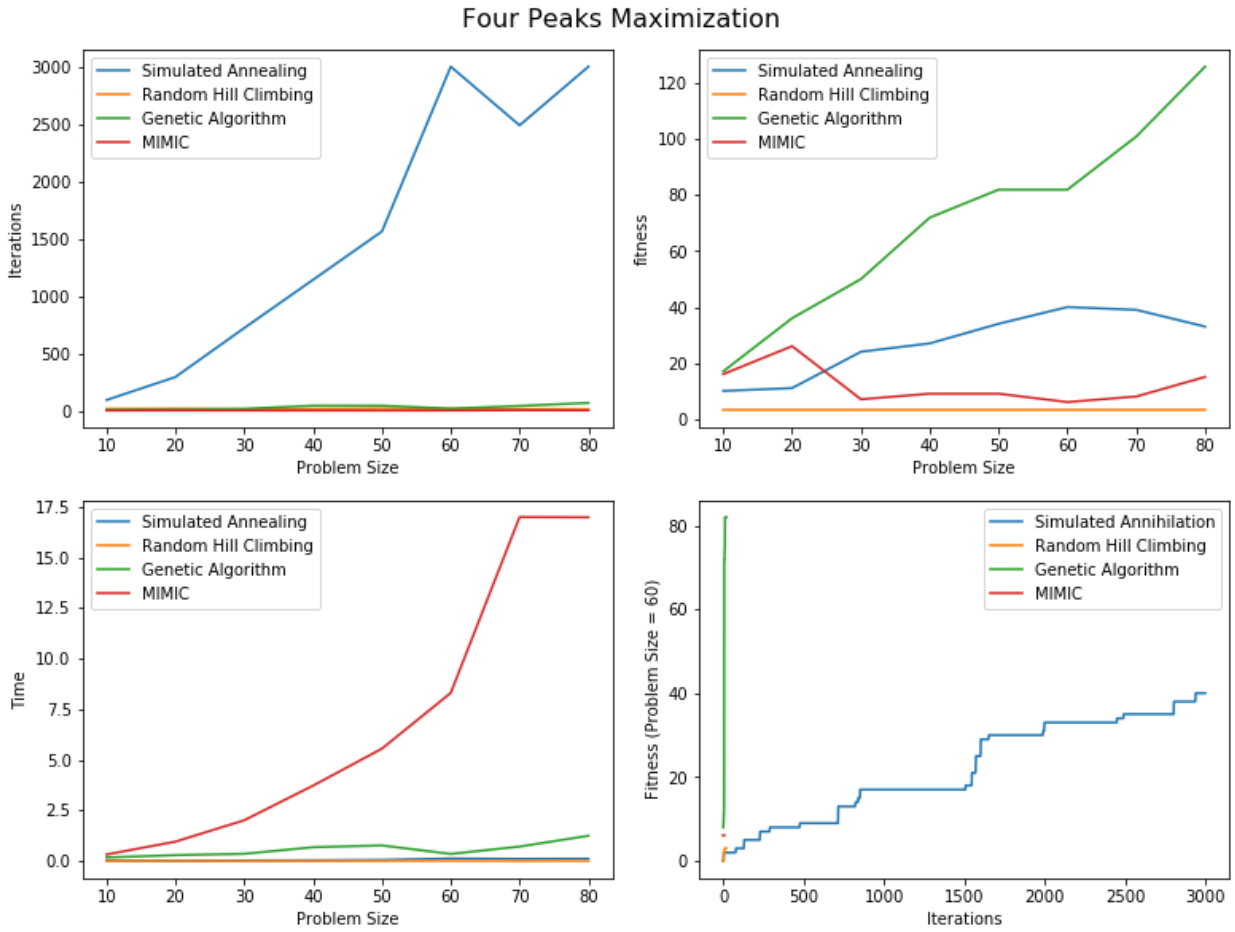

Figure 2. Four Peaks problem evaluation. Top left: Iterations vs problem size. Top right: Fitness vs problem size. Bottom left: Computational time vs problem size. Bottom right: Fitness vs iterations when problem size is 60.

Knapsack Problem

Knapsack problems aim to maximize the following function

$f(x) = \sum_{i=0}^{n-1} v_i\, x_i$, if $\sum_{i=0}^{n-1} w_i\, x_i \leq W$, and 0, otherwise

where v_i is the values, w_i is the weights, W is maximum Knapsack capacity, and n is the problem set size.

As this problem is more structured, MIMIC can take advantage of it by developing dependency tree. However, SA and RHC perform badly as the problem is too complicated for them.

I used the default fitness function of Knapsack by creating a list of n values and n weights. max_weight_pct is used to calculate the maximum knapsack capacity as a fraction of the total weights. The weights and values between 1 and 10. The problem is a maximization problem and the x values consist of bits.

Similar to what is described above, several parameters such as population size, max_attempts and keep_pct are varied for mimic to optimize the fitness results. The final values for MIMIC are population size of 500, keep_pct of 0.3, max_attempts of 10 and max_iters of 1000. The optimal values for GA are population size of 200, mutation probability of 0.1, max_attempts of 10. For SA, the optimal parameters are max_attempts of 10. For RHL, the optimal parameters are max_iters of 1000 and restarts of 10.

Figure 3 shows iterations, fitness, and computational time vs problem size for four algorithms. It can be seen that both GE and MIMIC show the best performance here. However, MIMIC converges in many fewer iterations. However, I should note that its computational time is higher. The bottom right of the figure shows that MIMIC converges to a very high fitness value in about 20 iterations.

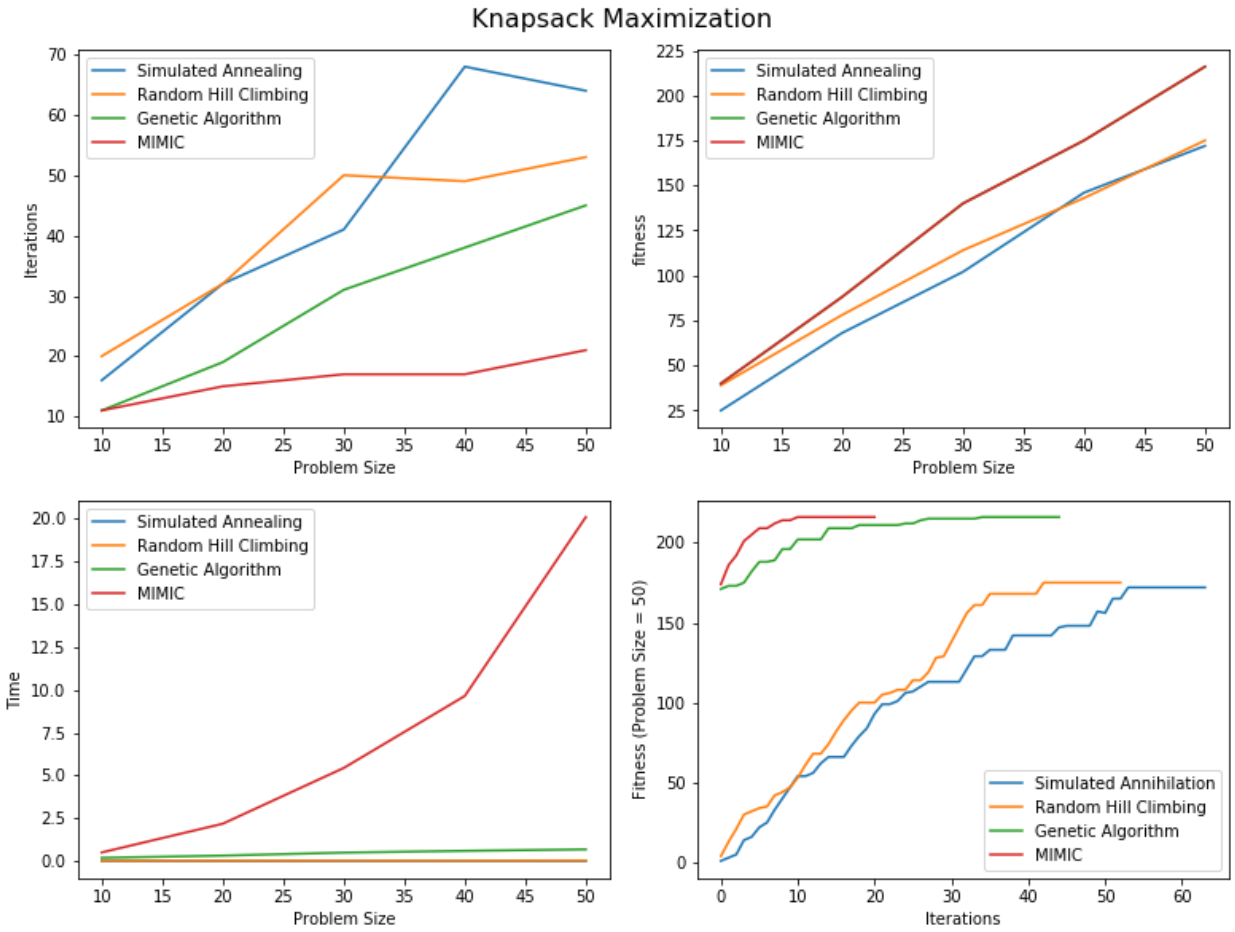

Figure 3. Knapsack problem evaluation. Top left: Iterations vs problem size. Top right: Fitness vs problem size. Bottom left: Computational time vs problem size. Bottom right: Fitness vs iterations when problem size is 50.

PART 2:

Here, I implement RHC, SA, GA and gradient descent (GD) on the water database that I had used in assignment 1. I copied the NN hyperparameters from assignment 1 which are 1 hidden layer, 15 neurons, tanh activation function, bias is true, classification problem, learning rate is 0.0001, early stopping is true, maximum attempts for validation is 100 and random state is 42. I do validation curve for RHC, SA and GA. I vary the restarts for RHC from 2 to 6. Figure 4 shows that training, validation and testing f-scores do not improve as a function of restarts. This is because RHC is not able to get out of the same local minimum even with several restarts. For future, more restarts can be tried. Figure 4 also shows that computational time increases as the number of restarts increases for RHC.

For SA, I vary the decaying schedule i.e. geometric, arithmetic and exponential. Again, as shown in Figure 4, there is no improvement between decaying schedules because SA is stuck in the local minimum and different schedules don't help. Computational time doesn't vary significantly.

For GA, I vary the population size from 100 to 300. 200 population size shows the best results and is the fastest. We could also try increasing the population size further to see if it helps.

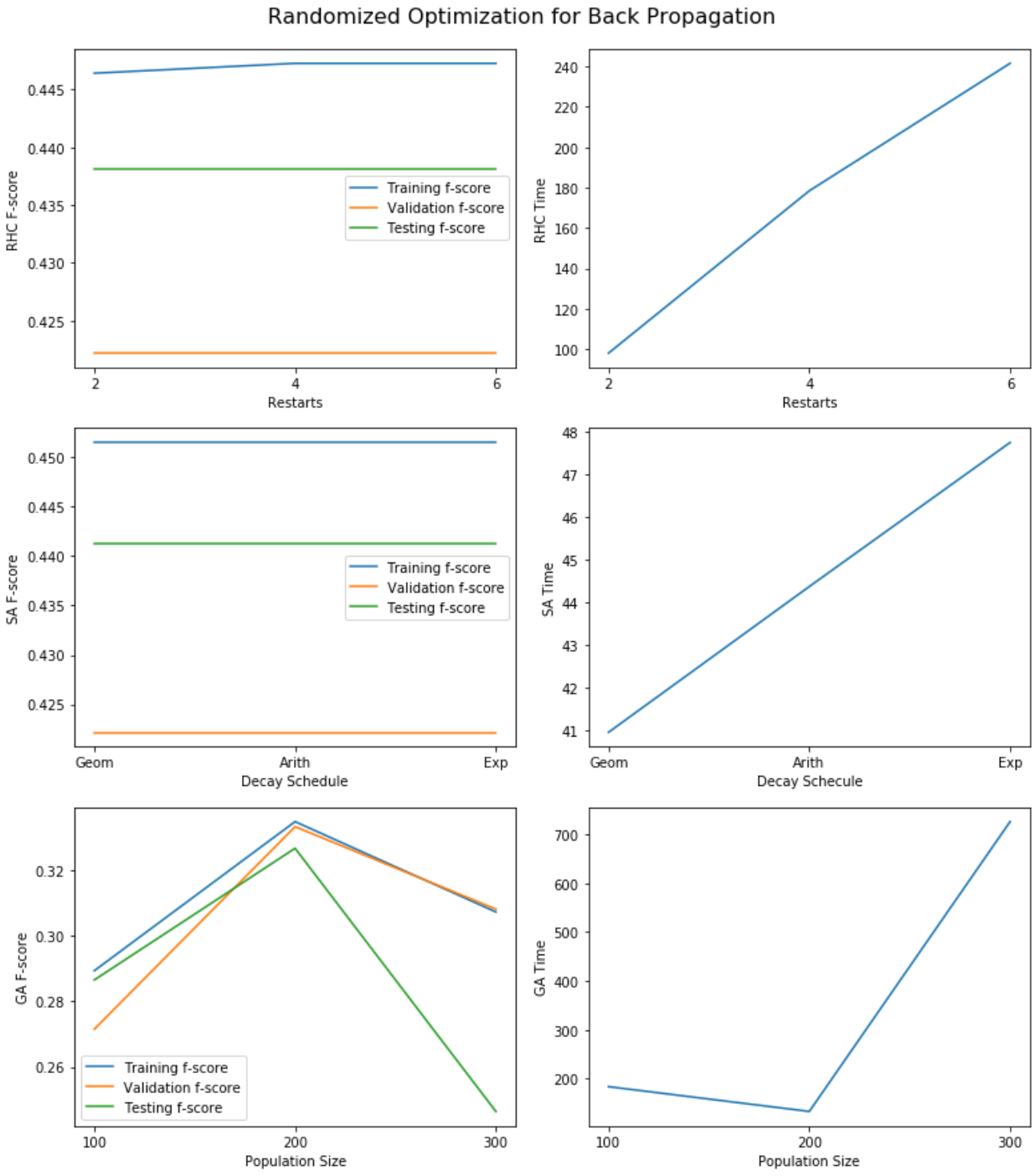

Figure 4. Top left: RHC F-scores vs restarts. Top right: RHC Time vs restarts. Middle left: SA F-scores vs restarts. Middle right: SA Time vs restarts. Bottom left: GA F-scores vs restarts. Bottom right: GA Time vs restarts.

I take the best for RHC, SA and GA and compare with GA. It can be seen in Figure 5 that gradient descent is better than the others in terms of training, validation and testing f-score, and computational time as well. This is expected because GD takes advantage of the derivative information where the others are more stochastic approached.

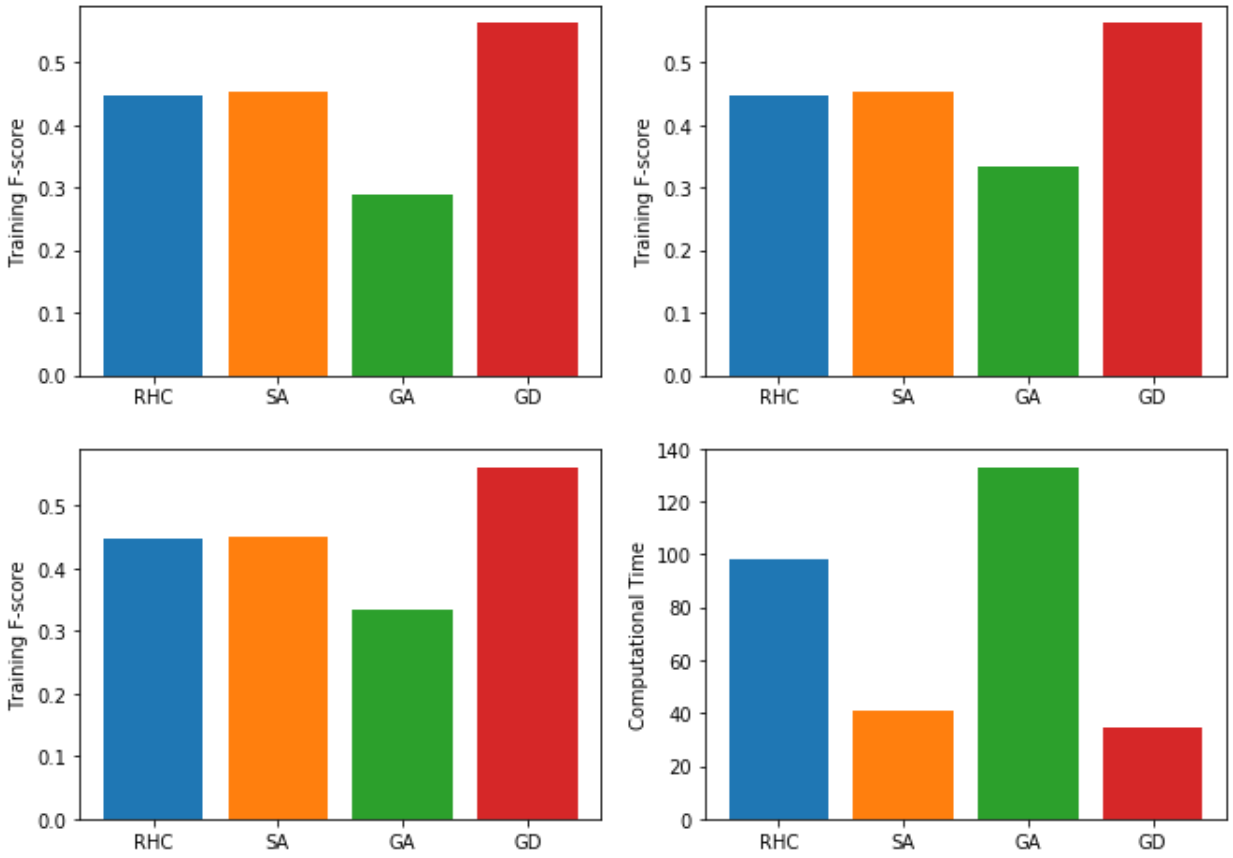

Figure 5. Top left: Training f-score vs algorithms. Top right: Validation f-score vs algorithms. Bottom left: Testing f-score vs algorithms. Bottom right: Computational time vs algorithms.