Project 2: Randomized Optimization
CS7641

Anita Rao
arao338@gatech.edu

1 RANDOMIZED OPTIMIZATION ALGORITHMS AND PROBLEMS

1.1 Optimization Algorithms

The four optimization algorithms that were evaluated were: (1) Randomized Hill Climbing, (2) Simulated Annealing, (3) Genetic Algorithms, and (4) MIMIC. These problems all aim to optimize (in our case, maximize) a fitness function, i.e. find the global maximum for "difficult" problems that may be discrete or non-differentiable. These algorithms were tested against three optimization problems that will be elaborated on in 1.2.

Randomized Hill Climbing (RHC) is an improvement on the Hill Climbing algorithm that steps around a local region in search of a maximum. Hill Climbing alone often falls into local maxima; RHC implements random restarts to the same problem, essentially re-solving the optimization problem from a new starting location. This increases the chance for a global maximum to be found while only scaling up compute time by a constant factor.

Simulated Annealing (SA) further improves on random restarts. This algorithm allows for some exploration, as opposed to solely "exploiting" to optimize the fitness function, to potentially discover the global maximum. The exploration is governed by a parameter T (simulating temperature), that decides whether jumping to a new point will occur or not. For greater T, the algorithm is more willing to move freely to explore other neighborhoods for a maximum; for smaller T, it becomes less free to explore and narrows in on, ideally, the global maximum.

Genetic Algorithms are useful for evaluating maxima in a multi-dimensional space. While local search (i.e. mutations) and number of iterations (i.e. generations) are still aspects of this algorithm, we now have the concept of *cross-over*, which allows us to combine attributes across neighborhoods to move closer to the global maximum.

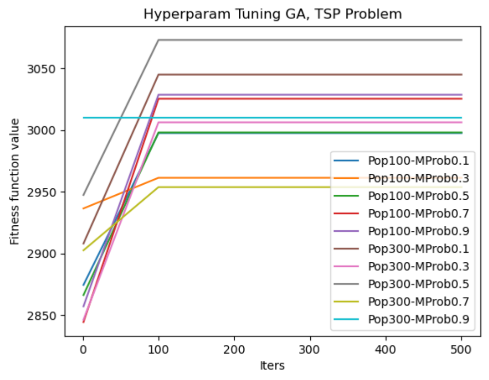
MIMIC addresses some shortcomings of the previous three algorithms, such as there being little information being passed from iteration to iteration (i.e. no structure) and an unclear definition of the underlying probability distribution guiding the algorithms. In MIMIC, we generate samples, retaining ones that have the best fit, and estimate new samples that fit the probability distribution.

1.2 Optimization Problems

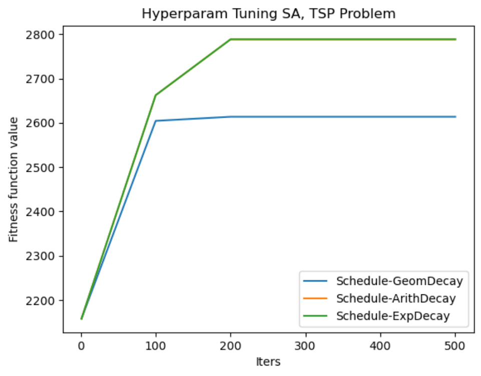
1.2.1 *Traveling Salesperson*

The Traveling Salesperson Problem (TSP) is an NP-hard problem for finding the optimal route between cities on a map. 100 random points were generated , and TSPOpt() from *mlrose* package was used to generate a fitness function. TSP showcases the weaknesses of algorithms that require a long number of iterations to converge, such as MIMIC, as TSP has a longer runtime on average than other optimization problems. It exercises the strengths of algorithms that work well for multi-dimensional spaces, such as GA, given that it aims to optimize distance in 2 dimensions.

Hyperparameter tuning in Figure 1 shows GA is optimized for {*pop_size*=300, *mutation_prob*=0.5}, SA for *schedule*=ArithDecay. The default parameters for MIMIC were used (*pop_size*=200, *keep_pct*=0.2). The parameter values for GA indicate that a larger population size with a 50% mutation probability yields optimal performance on the TSP problem's fitness function, perhaps due to the fact that more "points" in the problem space and increased crossover from the "fittest points" provide the best chance at reaching a global optimum.



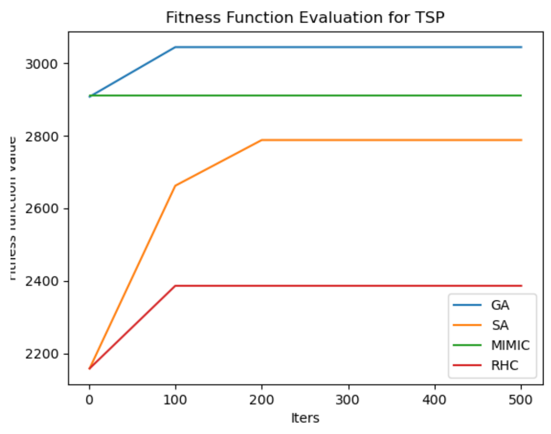
(a)



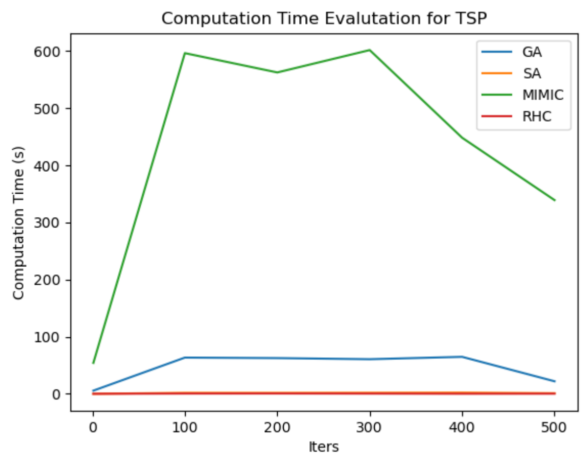
(b)

Figure 1—(a) Hyperparameter tuning for GA, TSP problem
(b) Hyperparameter tuning for SA, TSP problem

The algorithms were run for bit-string length=100, and a max of 500 iterations. Figure 2a shows that GA outperforms the other algorithms. It is interesting that MIMIC, which has the next highest fitness value, does not improve with more iterations indicating that we don't get much gain from maintaining a history over past iterations. SA performs better than RHC likely due to its ability to explore the problem space to a greater degree than RHC, which is strictly exploitative. While SA and RHC have low computation times according to Figure 2b, their fitness performance falls short of GA. GA finds a happy medium of highest fitness value, along with reasonable convergence time, making it the best algorithm for TSP.



(a)



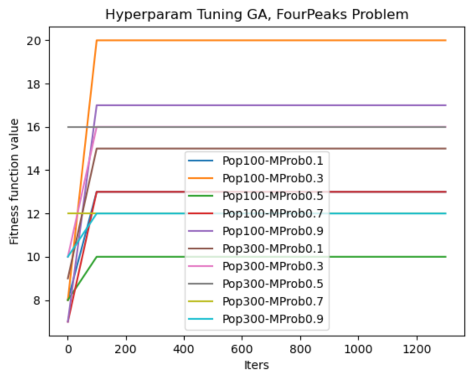
(b)

Figure 2—(a) Fitness Performance across Optimization Algorithms
(b) Compute Time Performance across Optimization Algorithms

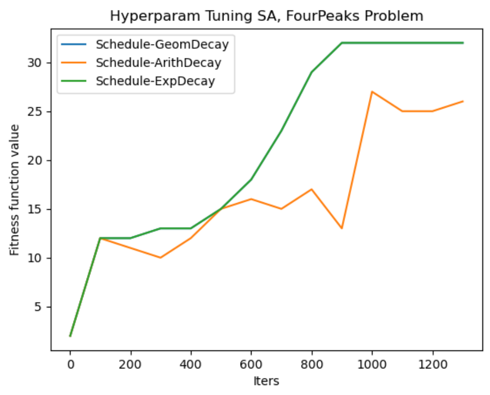
1.2.2 *Four Peaks*

The Four Peaks problem is interesting in that it includes two local maxima and two global maxima, expressing the weaknesses of algorithms that fall into a large basin of attraction around local maxima, such as RHC. It expresses the strengths of algorithms that get iteratively better at finding the global maximum as more time is spent exploring, such as SA. Due to the number of iterations required for exploration, algorithms with faster run times are preferred over heavy computational algorithms like MIMIC or GA.

Hyperparameter tuning in Figure 3 shows GA is optimized for *pop_size*=100, *mutation_prob*=0.3, SA for *schedule*=ExpDecay. The default parameters for MIMIC were used (*pop_size*=200, *keep_pct*=0.2). The exponential decay schedule outperforms the other schedules for determining temperature T in the SA algorithm; perhaps this is due to the fact that an exponential cooling of T narrows in on a maximum value faster (in fewer iterations) than the arithmetic schedule for the four peaks problem.



(a)

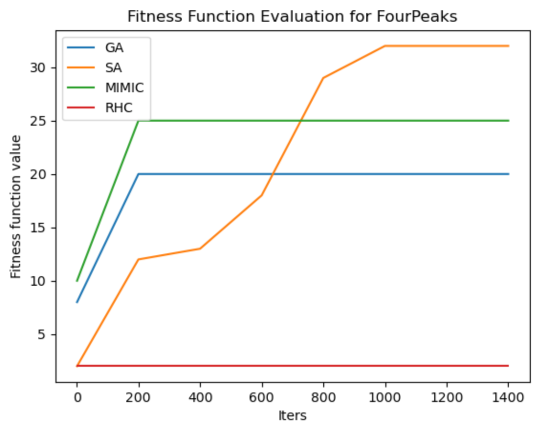


(b)

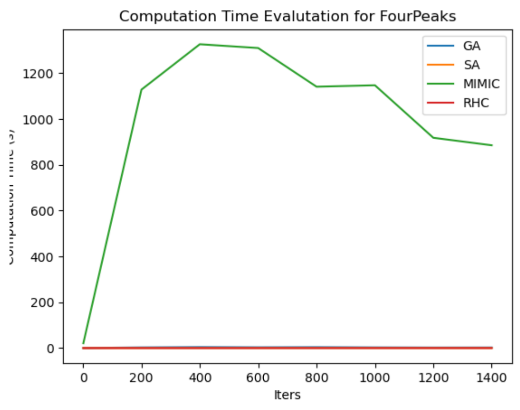
Figure 3—(a) Hyperparameter tuning for GA, Four Peaks problem

(b) Hyperparameter tuning for SA, Four Peaks problem

The algorithms were run for bit-string length=100, and a max of 1400 iterations. Figure 4a shows that SA outperformed the others in terms of fitness between 600-800 iterations. For less than 600 iterations, MIMIC and GA beat SA interestingly. This indicates that the four peaks problem initially benefits from maintaining some structure over iterations, or crossover/mutation. However, SA is able to find the global optimum after enough iterations to sufficiently explore the problem space. RHC performs poorly relative to the other algorithms, likely due to the fact that it gets stuck in the wide basins of attraction around the two local maxima. While SA takes more iterations to converge, its run-time is faster than other optimization algorithms, as seen in Figure 4b, making it the best algorithm for the Four Peaks Problem.



(a)



(b)

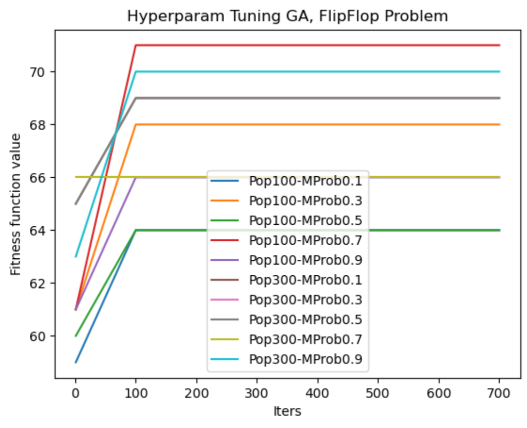
Figure 4—(a) Fitness Performance across Optimization Algorithms

(b) Compute Time Performance across Optimization Algorithms

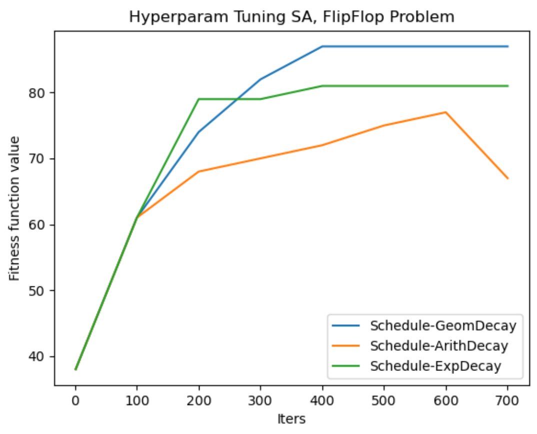
1.2.3 *Flip Flop*

The Flip Flop problem assigns greater fitness values to states with alternating bits (i.e. 101010). This problem expresses the strengths of algorithms that do well with structure and history, where some knowledge is maintained from iteration to iteration, like MIMIC. It expresses the weaknesses of algorithms that do not maintain structure, and therefore treat 101010 and 010101 as two different problems, like SA and RHC.

Hyperparameter tuning in Figure 5 shows that GA is optimized for *pop_size*=100, *mutation_prob*=0.7, SA for *schedule*=GeomDecay. Using the default parameters for MIMIC (*pop_size* = 200 and *keep_pct*=0.2), MIMIC outperformed the other algorithms quickly, so the hyperparameter tuning was performed after the algorithm comparison.



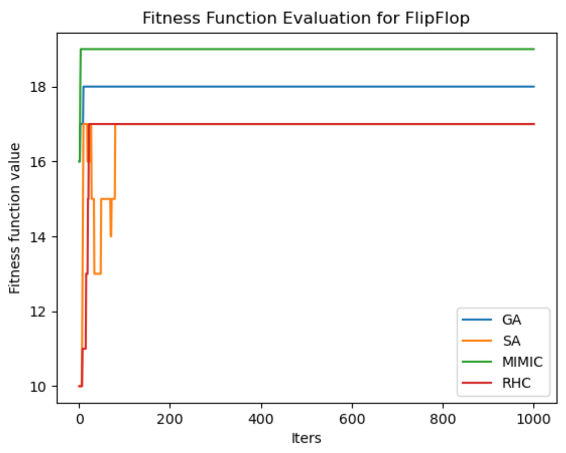
(a)

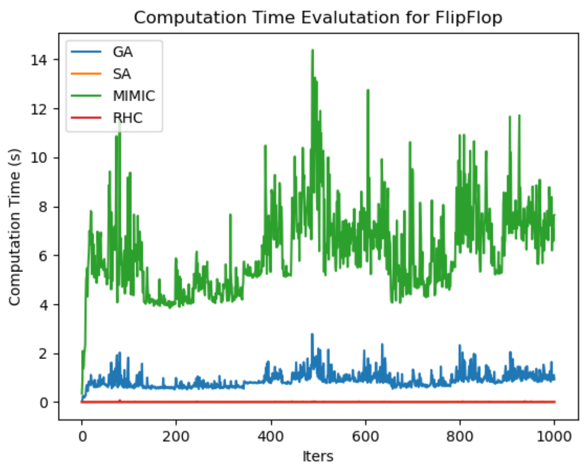


(b)

Figure 5—(a) Hyperparameter tuning for GA, Flip Flop problem
(b) Hyperparameter tuning for SA, Flip Flop problem

The hypertuned optimization algorithms were run for length=100, and 700 iterations. Figure 6 shows that MIMIC outperformed the others in terms of fitness in under 100 iterations. GA came second in fitness performance, signaling that mutation/crossover may be a beneficial technique for the Flip Flop problem to some degree. MIMIC likely outperforms GA due to the underlying structure that the problem exhibits (consecutively alternating bits). SA and RHC, while having fast compute times as seen in Figure 6b, cannot compare in terms of fitness to MIMIC. MIMIC exhibits the longest runtime of all the algorithms (also shown in Figure 6b), but converges quickly in terms of iterations. Ultimately, MIMIC is the best algorithm for these reasons.

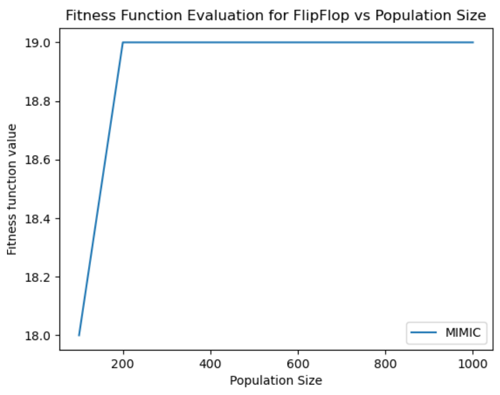


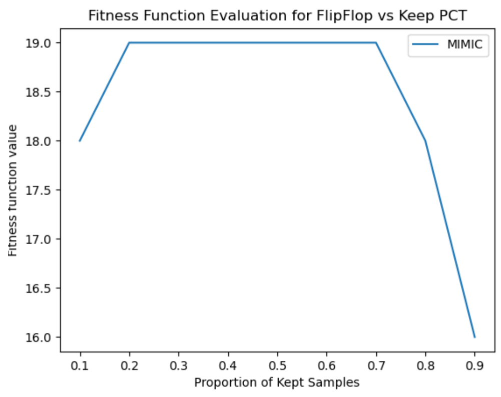


(a) *(b)*

Figure 6*—*(a) Fitness Performance across Optimization Algorithms
(b) Compute Time Performance across Optimization Algorithms

Hyperparameter tuning was performed on *pop_size* (size of population in MIMIC) and *keep_pct* (proportion of samples kept between iterations) to find an optimal setting for the MIMIC algorithm in Figure 7. Figure 7a shows that for population size less than 200, the fitness function is not optimized, likely due to the sample of points being too small to find a global maximum. Figure 7b shows that if the proportion of samples kept across iterations is too small or too large, we do not find the optimal fitness value, likely due to not trusting our data *enough* or trusting it *too much*, respectively.





(a) *(b)*

Figure 7*—*(a) Fitness Performance vs Population Size
(b) Fitness Performance vs Proportion of Kept Samples

1.3 Algorithm Improvement

For testing the algorithms on the three problems above, I noticed how the parameter tuning made a significant difference for GA and SA. Due to time constraints, MIMIC parameters were not tuned as extensively as GA and SA, and ultimately were chosen to remain as the default values provided in *mlrose*. It would be a good exercise to explore these parameter settings to understand how MIMIC performance compares to GA, SA, and RHC for the TSP and Four Peaks problems.

2 OPTIMAL WEIGHTS FOR NEURAL NETWORK

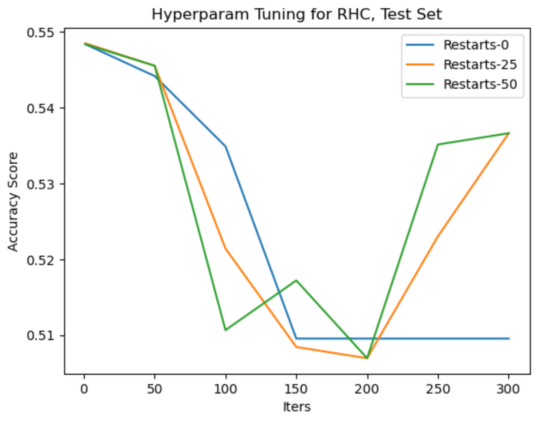
Using three of the optimization algorithms above, new weights were computed/optimized for the Multi-Layer Perceptron neural network that was implemented in Assignment 1. This model had 30 hidden layers, and used the ReLU activation function. The algorithms, (1) RHC, (2) SA, and (3) GA were then compared to the neural network whose weights were optimized using the backpropagation method. The algorithm comparison was done for the Flight Delay Dataset provided by the Jožef Stefan Institute (26969 samples), which considers airline departure and arrival information and classifies flights as “Delayed” or “Note Delayed.” See References for details.

2.1 Performance metrics

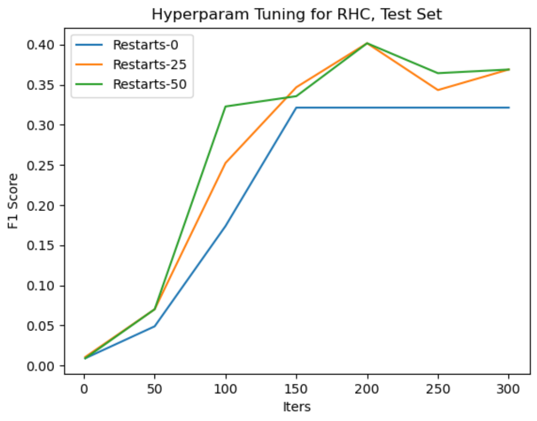
Accuracy and F1 Score were used as metrics to evaluate the performance of the neural network. Accuracy provides the number of correctly classified target values, while F1 score is a balance of precision and recall metrics, accounting for scenarios with a high penalty for False Positives and False Negatives, respectively. F1 score was chosen to evaluate the neural network performance on this dataset because False Positives (classifying a flight as “delayed” when it is *not*) and False Negatives (classifying a flight as “not delayed” when it *is*) are costly to airlines and customers.

2.2 RHC

Hyperparameter tuning was performed in Figure 8 for number of *restarts* in the RHC algorithm at 300 iterations, with accuracy results in Figure 8a and f1 score results in Figure 8b. Random restarts=50 produces a higher accuracy and f1 score, indicating that random restart helped to maximize the fitness function. The test set is used for evaluation (training set results were generated, but are not shown as they show very similar trends).



(a)

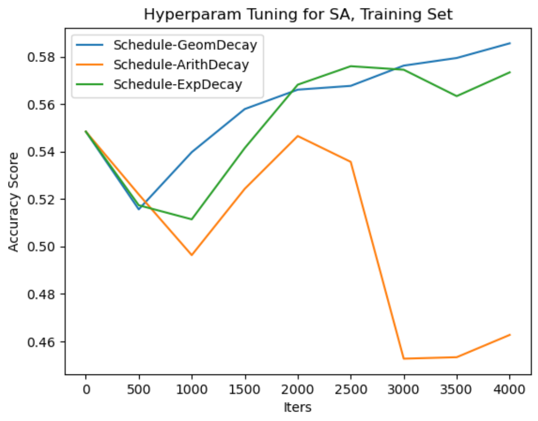


(b)

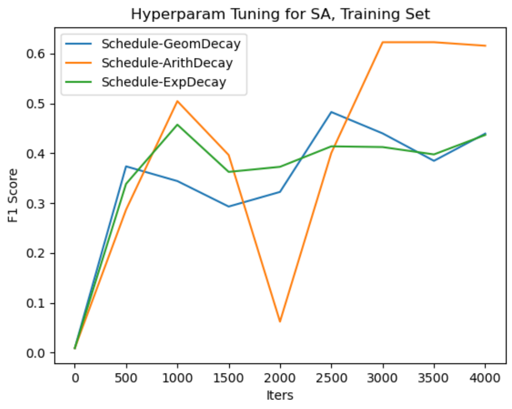
Figure 8*—*(a) Hyperparameter tuning for RHC, using accuracy
(b) Hyperparameter tuning for RHC, using f1 score

2.3 SA

Hyperparameter tuning in Figure 9 shows that SA is optimized for *schedule*=GeomDecay, which shows increasing Accuracy and F1 score as iterations increase. ArithDecay's rising F1 score but falling accuracy score implies precision is low (many false positives), while recall is high (few false negatives). This suggests that the classifications prefer the minority target value in the dataset (given that the dataset is slightly imbalanced). Similar to the plots above, the test set is used for evaluation (minor correction: titles should read "Hyperparam Tuning for SA, *Test* set").



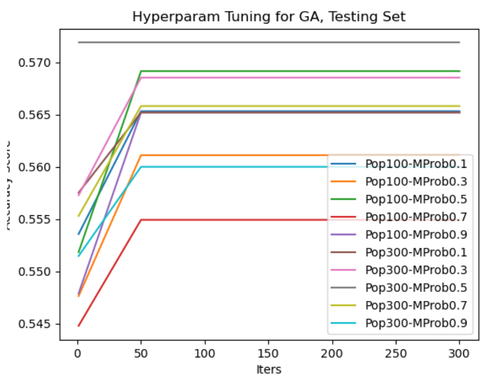
(a)



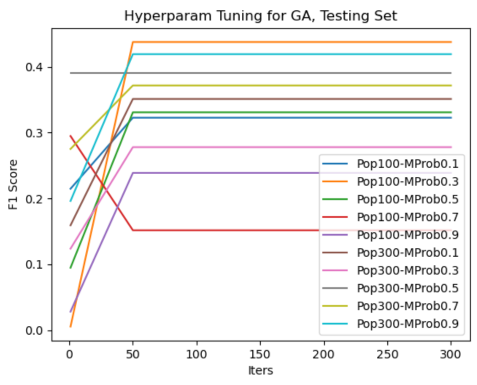
(b)

Figure 9*—*(a) Hyperparameter tuning for SA, using accuracy
(b) Hyperparameter tuning for RHC, using f1 score

2.4 GA



(a)

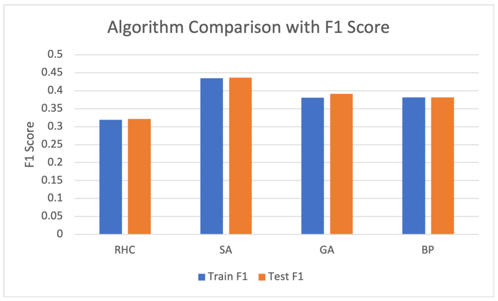


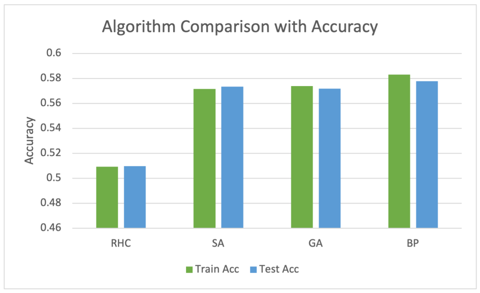
(b)

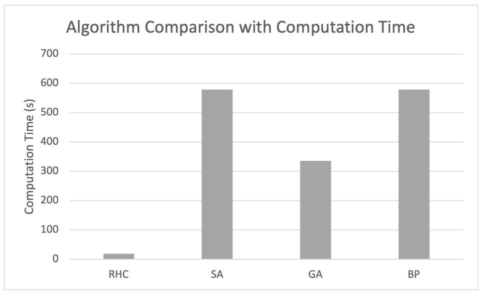
Figure 10—(a) Hyperparameter tuning for GA, using accuracy
(b) Hyperparameter tuning for GA, using f1 score

Hyperparameter tuning in Figure 10 shows that GA is optimized for *pop_size*=300, *mutation_prob*=0.5, over 300 iterations. This parameter selection yields the highest accuracy, while yielding one of the higher f1 scores compared to other combinations; hence the parameter values selected. Similar to the plots above, the test set is used for evaluation.

2.5 Comparing algorithms







(a) *(b)* *(c)*

Figure 11—(a) Algorithm Comparison, using F1 score on train/test set
(b) Algorithm Comparison, using Accuracy score on train/test set
(c) Algorithm Comparison, using Computation Time of model training

The four algorithms (RHC, SA, GA, and Backprop) were run for 4000 iterations, to allow SA to converge. A few observations can be made:

1. It is interesting to note that while Backprop beat the algorithms in terms of accuracy (slight), SA beat the other algorithms in terms of f1 score (by a larger margin). Given that SA and Backprop take approximately the same time run for 4000 iterations, SA emerges as the best algorithm for the Flight Delay dataset, over the Backprop method.
2. GA also exhibits a similar performance to Backprop, but does not surpass SA or Backprop in terms of accuracy or f1 score. It is noteworthy, however, that GA's runtime is a little more than half of SA's or Backprop's -- this creates a good argument for using GA if minimizing time/iterations is a factor in the classification process.
3. RHC is quick to complete, but does not perform well compared to the other algorithms.

2.6 Algorithm Improvement

All three of the algorithms that were compared against the backpropagation method could be optimized to a greater extent to obtain an even better performance:

1. The restarts parameter was evaluated for values of [0, 25, 50], showing more restarts resulted in higher performance. This parameter could have been swept further until the accuracy/f1 score converged relatively, although this would take more computation time (scaled linearly).
2. The schedule parameter for the SA algorithm was chosen to be GeomDecay, based on the accuracy and f1 score metrics. However, the input parameters to the schedule functions themselves were not tuned, and could yield very different results depending on their settings. This could reveal another schedule type to be better than GeomDecay, or perhaps the performance with GeomDecay could be further improved, ultimately increasing accuracy and f1 scores.
3. The parameters of the GA algorithm could be further tuned as well, as the only combinations that were looked at were the cartesian product of *pop_size*=[100, 300] and *mutation_prob*=[0.1, 0.3, 0.5, 0.7 ,0.9]. Perhaps for different values of *pop_size*, or *max_attempts* which was not goggled at all, the GA algorithm could be further improved and yield higher performance.

3 REFERENCES

1. Elena IKONOMOVSKA'S web page. (n.d.). Retrieved February 22, 2021, from http://kt.ijs.si/elena_ikonomovska/data.html