CS-7641 Machine Learning:
Assignment 2, Randomized Optimization

Pavel Ponomarev
pavponn@gatech.edu

1 INTRODUCTION

In this project, we [1] study the performance of four randomized optimization algorithms: Random Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA) and MIMIC. We first study performance of these methods when applied to three discrete optimization problems: *One-Max*, *N-Queens* and *Flip-Flop*. We then try to apply RHC, SA and GA in the machine learning setting to optimize the weights for a neural network and compare the results against ones obtained using Gradient Descent optimization.

2 PROBLEMS & DATASET OVERVIEW

The first chosen optimization problem is **One-Max**. Here, we need to find maximum of the following fitness function: $f(x) = \sum_{x_i \in x} x_i$, where x is a n bit vector. We generally expect algorithms like RHC and SA demonstrate good results and outperform more complex options like GA or MIMIC, given the simplicity of the problem. I.e., it lacks a complex structure, has one (global) maximum, and can be solved without exploration. In fact, maximizing the fitness function is the same as finding maximum for each element of the input vector. Another factor that might play a deciding role is computational complexity of the algorithms. In general, Random Hill Climbing and Simulated Annealing are cheap algorithms in terms of time per iteration, which also might be a plus for them.

Another discrete problem for randomized optimisation that we chose is **N-Queens**. The objective here is to minimize the number of attacking pairs among N queens on an N×N chess board. Represented as an N-dimensional vector x, where x_i denotes the row of the queen in column i. This problem is intriguing due to (i) the existence of multiple arrangements without any attacks, resulting in multiple global minima, and (ii) its complex, non-linear relationship between the state and fitness value (where fitness is determined by relations between elements of the input state vector). Both MIMIC and Genetic Algorithm are expected to perform well on this problem. MIMIC's potential lies in its ability to discern dependencies between positions of queens through the construction of dependency trees. On the other hand, GA's strength may lie in generating progressively better assignments for queen positions during the crossover/mutation operation. For visual comparison with other problems, we transform this into a maximization problem by negating the fitness function.

The third problem we explore for randomized discrete optimisation is **Flip-Flop**, which aims to maximize the number of sequential alternated bits (i.e., 01 or 10) in a bit vector. In this problem we expect to get the best result using MIMIC algorithm since the problem has a chain structure (special case of a dependency tree) that algorithm could exploit. In fact, this result was shown to be the case by the researcher, so it would be interesting to verify whether it holds. At

1 The work described in this report is completed fully independently by one sole author (Pavel Ponomarev). The pronounce "we" is used to follow academic writing style.

the same time, one could argue that problem structure is not that complex, so potentially other algorithms like SA or GA could demonstrate comparable results.

For Neural Network optimization, we utilize the **Wine Quality** dataset from Assignment 1. It's a multi-class dataset with uneven subsets for red and white wine: 1599 and 4898 samples, respectively. The class labels range from 3 to 9, with a majority falling between 5 to 7 in terms of quality.

3 TECHNOLOGIES

We utilized Python 3, leveraging its extensive library ecosystem. More specifically, we sourced implementations of the randomized optimization algorithms and chosen problems' fitness functions from `mlrose-hiive` [2] package.

4 EXPERIMENTAL METHODOLOGY

4.1 Discrete Randomized Optimization Problems

For each problem examined in Section 2, a standardized approach is followed. Initially, three distinct problem sizes (*small*, *medium*, and *large*) are chosen to gauge how algorithms respond to varying input dimensions.

We perform 10 independent runs with different random seeds for each algorithm, problem size, and set of hyperparameters. Each run is capped at 300 iterations, with a maximum of 50 attempts to find an improved state (convergence criteria). The results from these runs are averaged to represent the performance for a specific problem size, algorithm, and its hyperparameters. We start by comparing the best-performing runs (wrt. to the reached fitness value) across problem sizes and algorithms, focusing on metrics like Fitness %[3], Time, and Function Evaluations (FEvals). Then, we narrow our focus to a specific problem size, analyzing how performance metrics relate to algorithm iterations and examining the influence of individual hyperparameters.

The hyperparameters under consideration include initial temperature (SA), mutation rate (GA), population size (GA and MIMIC), and keep percentage (MIMIC).

4.2 Neural Network Weights Optimization

In this part, we optimize weights of a NN using four algorithms: Gradient Descent, Random Hill Climbing, Simulated Annealing, and Genetic Algorithm. Following Occam's Razor principle, we employ a modified neural network architecture tailored to the Wine Quality dataset compared to Assignment 1, featuring two hidden layers with 50 neurons each (down from 200). This model could still provide comparable results with increased resilience to overfitting. As an activation function, we still use ReLU. Unfortunately, there is no support for regularization in `mlrose-hive`.

The Wine Quality Dataset is divided into 3 parts: training dataset (70%), validation dataset (15%) and testing dataset (15%). Unfortunately, we could not directly integrate K-fold cross-validation into the experiments due to the lack of its support in `mlrose-hive`. Thus, we used Leave-p-out cross validation, where p is equal to 15%. As a performance metric, we use micro F1 score, as we did in the Assignment 1.

2 https://pypi.org/project/mlrose-hiive/

3 This metric represents how good is the found fitness function value compared to the size of the problem, i.e., "how close the found value is to the optimal value". Commonly just a ration of found and optimal values.

Each algorithm undergoes multiple runs with distinct hyperparameter combinations. We then identify optimal configurations for each using validation scores and compare their performance. Finally, we explore the impact of individual hyperparameters on algorithmic effectiveness. The considered hyperparameters are in line with those detailed in Section 4.1, with an added focus on restarts for RHC and learning rate for all algorithms. We limit number of attempts to find an imprvovement for randomized algorithms to 200 and total number of iteration to 500.

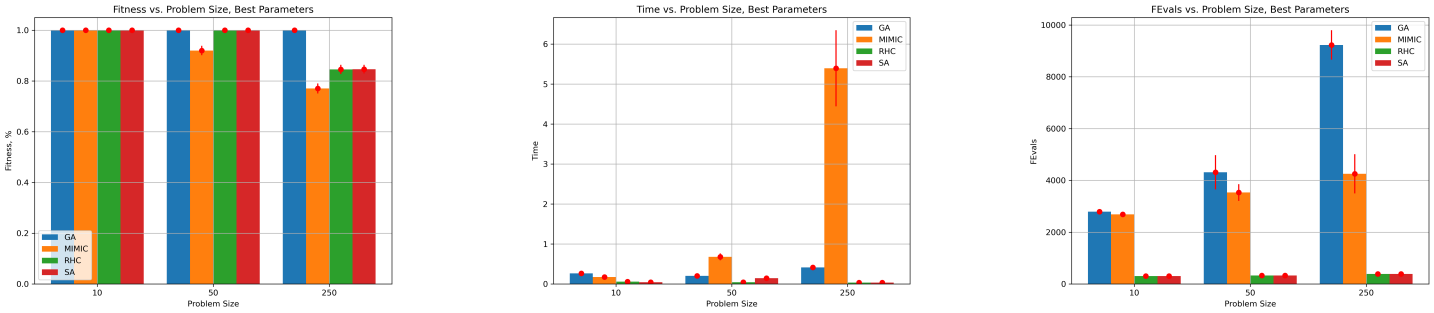
5 OPTIMIZATION PROBLEMS

Table 1 summaries the data about hyperparameters that we considered for different algorithms, as well as their best values for each optimisation problem and size of the problem we experimented with. The rest of this section is organised is follows: in Section 5.1,5.2 and 5.3 we overview results of best setups of hyperparameters for a specific problem. There, we first compare results against different problem sizes and then choosing one of them dive deeper analysing behavior of algorithms against iterations. Then, we analyse commonly noticed patterns of algorithms behaviour for different hyperparameters in Section 5.4. Finally, we conclude with suggestions of future possible improvements for the algorithm performance and analysis in Section 5.5.

Algorithm	**Hyperparameter Space**	**Best Hyperparameters**		
		One-Max	**N-Queens**	**Flip-Flop**
Random Hill Climbing	*restarts* = {*no_restarts*}	*restarts* = *no_restarts* (all sizes)	*restarts* = *no_restarts* (all sizes)	*restarts* = *no_restarts* (all sizes)
Simulated Annealing	*init_temp* = {0.1, 0.5, 1.0, 2, 5, 10}	*init_temp*= 0.1 `(all sizes)`	*init_temp* = 0.5 (size=50), 0.1 (size={10, 25});	*init_temp* = 0.1 (size=10), 0.5 (size={50, 250});
Genetic Algorithm	*mut_rate* = {0.05, 0.1, 0.2, 0.4, 0.6, 0.8} *pop_size* = {50, 250}	*mut_rate* = 0.05 (size={10,50}), 0.6 (size={250}); *pop_size* = 50 (all sizes)	*mut_rate* = 0.6 (size={10,50}), 0.8 (size={25}); *pop_size* = 50 (all sizes)	*mut_rate* = 0.6 (size=30), 0.4 (size=80), 0.8 (size=200); *pop_size* = 50 (all sizes)
MIMIC	*keep_pct* = {0.1, 0.2, 0.4, 0.5, 0.7}, *pop_size* = {50, 250}	*keep_pct* = 0.1 (size=10), 0.5 (size={50, 250}); *pop_size* = 50 (all sizes)	*keep_pct* = 0.5 (size=10), 0.7 (size={25, 50}); *pop_size* = 50 (all sizes)	*keep_pct* = 0.2 (size=30), 0.5 (size=80), 0.4 (size=200); *pop_size* = 250 (all sizes)

Table 1—Neural Network performance results based on optimization algorithm

5.1 One-Max



(a) Fitness, % ***(b)*** Time ***(c)*** Function Evaluations

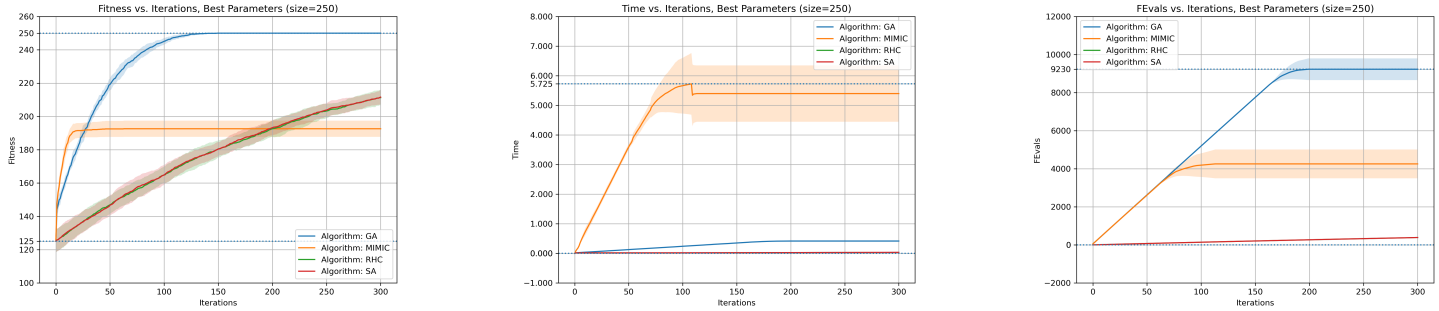
Figure 1—Comparative metrics of algorithms' best runs on different problem sizes for the One-Max problem

For the One-Max problem, Figure 1a indicates that GA, SA, and RHC achieve maximum fitness for problem sizes of 10 and 50. Surprisingly, for the larger problem size of 250, only GA achieves the best result, contrary to our initial expectations outlined in Section 2. This warrants further discussion in this subsection. It's noteworthy that MIMIC fails to reach the optimum for all problem sizes.

MIMIC proves to be the most time-consuming for problem sizes of 50 and 250 (Figure 1b). This is in line with expectations, as MIMIC's iterations are more computationally intensive. It constructs a dependency tree with a time

complexity of $O(n^2)$ on each iteration, where n is the population size. In contrast, other algorithms' computations are either constant (RHC, SA) or have a linear dependency on hyperparameters (GA). However, direct comparisons of algorithm runtimes may be nuanced due to factors like chosen hyperparameters and time until convergence. For instance, for a problem size of 10, MIMIC takes less time than GA, indicating a benefit in performance for small problem sizes. As expected, Random Hill Climbing and Simulated Annealing exhibit the lowest times.

In Figure 1c, we present the number of function evaluations by each algorithm. RHC and SA, as anticipated, demonstrate the lowest numbers. Both GA and MIMIC require significantly more iterations compared to RHC and SA, primarily due to the fact that the number of function evaluations for GA and MIMIC depends on the population size (which is 50 for both), resulting in a considerable difference. The reason, why MIMIC makes less function calls compared to GA is due to its faster convergences which essentially means that algorithm runs less iterations on average which we will see in the further paragraph.



(a) Fitness, %

(b) Time

(c) Function Evaluations

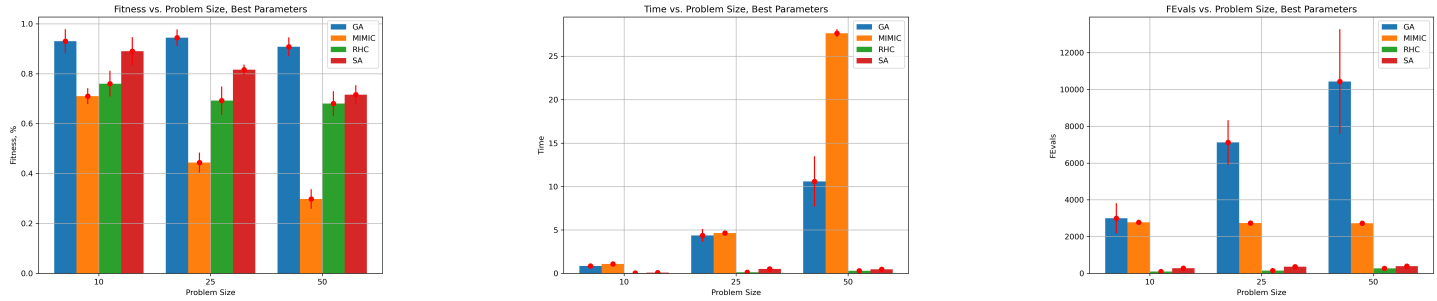
Figure 2—Comparative Metrics vs Iterations of all algorithms for the One-Max (size=250) Problem

Now, let's narrow our focus to a problem of size 250 and examine algorithm behavior versus iterations, as depicted in Figure 2. It's easy to see that both SA and RHC do not reach their maximum potential on this problem size, suggesting that allowing more iterations could enhance their performance. Indeed, Figure 2a suggests that algorithms are far from converging since fitness curve is not flat. Given their low time and number of function evaluations, which are also evident from Figures 2b and 2c, this might be the most practical approach, actually resulting in the most efficient solution. Thus, our initial belief was not far from truth, but should have been adjusted depending on number of maximum allowed iterations.

This data also confirms our other observation: MIMIC converges faster (wrt. iterations) than any other algorithm, however despite this and even not reaching maximum, it still takes more time than other algorithms.

5.2 N-Queens

The Genetic Algorithm demonstrates superior performance in optimizing the fitness function value in the N-Queens problem (Figure 3a). Conversely, MIMIC struggles with this problem, failing to find solutions even for the 10-Queens instance. This can hint us that the N-Queens problem is not necessarily a structural problem in "MIMIC's point of view" despite (i) problem having a graph-like representation (ii) typical brute-force solution being backtracking that is essentially traversing the tree of potential actions given current state, which is similar to dependency trees. In contrast,

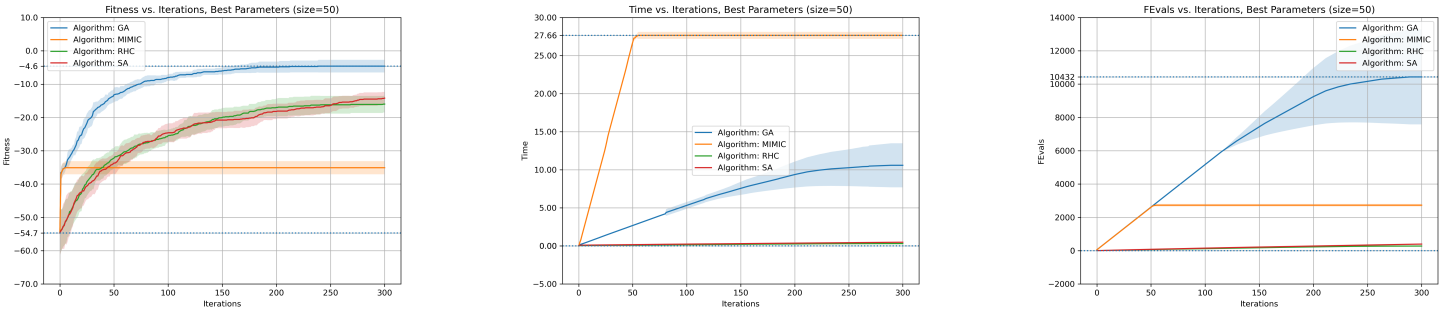


(a) Fitness, % ***(b)*** Time ***(c)*** Function Evaluations

Figure 3—Comparative metrics of algorithms' best runs on different problem sizes for the N-Queens problem

Simulated Annealing (SA) and Randomized Hill Climbing (RHC) exhibit relatively commendable performance. It is likely that further iterations of these algorithms could lead to enhanced solutions.

Figure 3b and Figure 3c show results akin to those observed for the One-Max problem in Section 5.1. Thus, everything said in the previous section should hold for this part, and we refrain from explicit discussion here to save space.



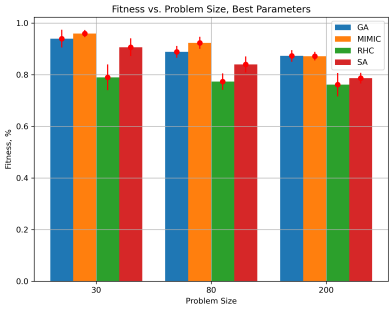
(a) Fitness, % ***(b)*** Time ***(c)*** Function Evaluations

Figure 4—Comparative Metrics vs Iterations of all algorithms for the N-Queens (size=50) Problem

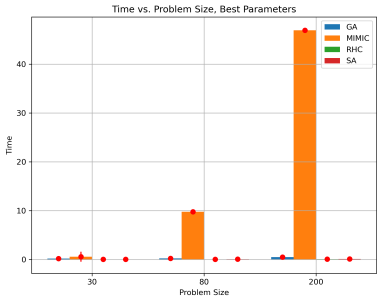
The results we see in the Figure 4 for the N-Queen problem resemble ones that we observes for the One-Max problem in Figure 2. However, there is a difference in the behaviour of Random Hill Climbing and Simulated Annealing algorithms compared to that case. In this problem, after looking at Figure 4a, we can notice that fitness curve resembles logarithmic function suggesting it is about to reach the plateau and converge to some value that is not a global maximum. This behaviour clearly articulate the difference in the complexity of One-Max and N-Queens problems: while in the former we could potentially just increase number of iterations to make the the algorithm find local maximum (while still maintaining low time), for the latter we will not be able to achieve similar result and despite increasing number of iterations these algorithms will still converge to non-optimal (and lower than GA) value of fitness function.

5.3 Flip-Flop

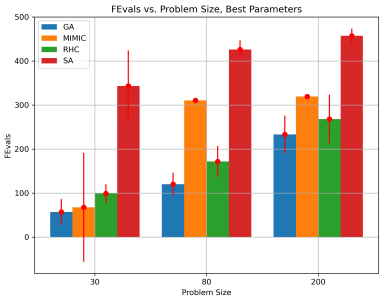
For the Flip-Flop algorithm, MIMIC consistently demonstrates the best performance across all problem sizes in terms of fitness value (Figure 5a), aligning with established research findings. This can be attributed, in part, to the chain-like structure of the problem that MIMIC adeptly models through decision trees. However, it's worth noting that other



(a) Fitness, %



(b) Time



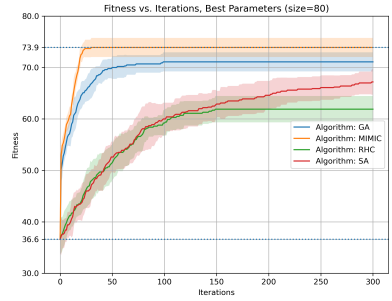
(c) Function Evaluations

Figure 5—Comparative metrics of algorithms' best runs on different problem sizes for the Flip-Flop problem

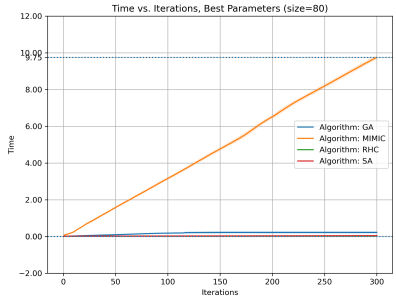
algorithms consistently yield high results, albeit slightly lower than MIMIC. Despite our belief that RHC, SA and GA will be confused by the fact that two completely opposite state vectors will have the same fitness function value, they tend to do relatively well and make themselves worth considering, especially due to some positive trade-offs wrt. time complexity.

While MIMIC does achieve superior results, it also requires significantly more time to converge compared to other algorithms (refer to Figure 5b), due to the reasons discussed in Section 5.1. Interestingly, in terms of time metric, GA closely rivals SA and RHC, indicating fast convergence.

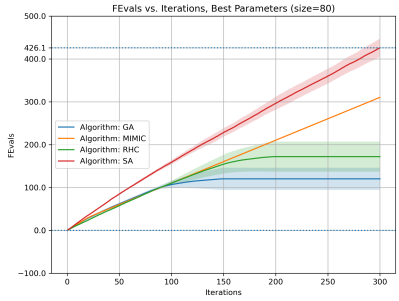
The data presented in Figure 5c proves particularly intriguing for this problem. Firstly, it's evident that MIMIC consistently executes a higher number of function evaluations than Genetic Algorithm, a deviation from prior observations. One possible explanation is that the optimal hyperparameters representing population size for MIMIC and GA differ: 250 and 50 respectively. Additionally, it's conceivable that GA converges more swiftly, leading to fewer function calls. Even more surprisingly, Simulated Annealing requires more function evaluations than both GA and MIMIC, despite the fact that these algorithms have population sizes of 50 and 250 respectively. This contrasts with previous expectations. This observation may suggest that both MIMIC and GA achieve relatively rapid convergence. We try to go deeper in the analysis for statistics against algorithm iterations.



(a) Fitness, %



(b) Time



(c) Function Evaluations

Figure 6—Comparative Metrics vs Iterations of all algorithms for the Flip-Flop (size=80) Problem

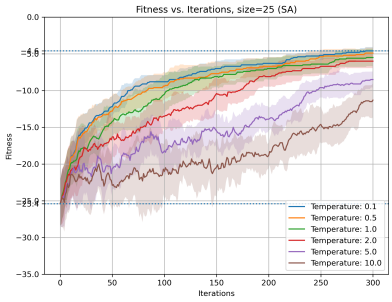
Using Figure 6, we evaluate the performance of the best algorithm setup in relation to the number of iterations. Our aim is to either confirm or refute our earlier hypothesis. Notably, the Genetic Algorithm exhibits the fastest

convergence, as evidenced by the early plateauing of time and function evaluations curves (Figure 6c and Figure 6b), although it may not necessarily reach the optimal value.

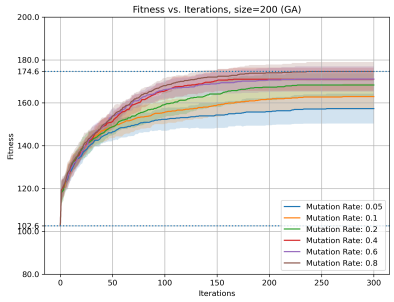
Next, we scrutinize the behavior of the MIMIC algorithm. Despite Figure 6a suggesting convergence after fewer than 25 iterations, a closer look at Figure 6b and Figure 6c reveals a continual increase in both time and function evaluations with each new iteration. This discrepancy could possibly be attributed to a scenario in which the algorithm rapidly converges to a distribution, P^{Θ_t}, generating a disproportionately low number of distinct samples due to the higher likelihood of a smaller set of samples. The algorithm then endeavors to produce $P^{\Theta_{t+1}}$ from P^{Θ_t}, incurring a computational cost of $O(n^2)$, which aligns with the observed data. However, the resulting distribution, $P^{\Theta_{t+1}}$, is highly skewed similarly to P^{Θ_t}. While this hypothetical situation could potentially elucidate the Fitness curve's apparent convergence, it does not account for the persistently low number of function calls in MIMIC and GA, especially in the initial iterations, where it should equate to the population size hyperparameter. We belive that the `mlrose-hiive` library may have an implementation issue related to the calculation of function calls for the Flip-Flop fitness function. Notably, the outcomes for RHC and SA align more closely with our expectations from theory.

5.4 Hyperparameters Analysis

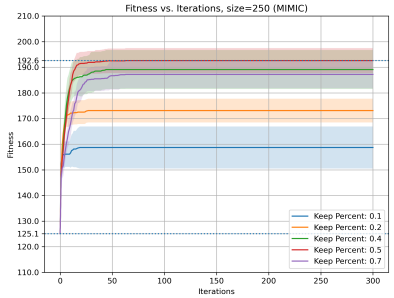
Let us quickly overview how main hyperparameters we worked with in randomization algorithms affect fitness function value. We generally notice the behaviour being relatively similar across problems, so in Figure 7 we only provide specific examples for each hyperparameter. We omit going into many details due to space limitation.



(a) Initial Temperature, Simulated Annealing (N-Queens, size=25)



(b) Mutation Rate, Genetic Algorithm (Flip-Flop, size=200)



(c) Keep Percent, MIMIC (One-Max, size=250)

Figure 7—Fitness curves with parameter split

In general, increasing initial temperature in SA method introduces higher variance and makes convergences slower (example in Figure 7a), this is a trade-off one should be willing to pay to increase chance of hitting global maximum on a function with multiple local optimums. For the problems we worked on, we noticed that it is almost always better to take average or high mutation rate in GA method (example in Figure 7b), besides the One-Max problem, where all mutation rates converged to approximately the same result. In MIMIC applications we noticed that increasing keep percentage parameter (nth-percentile) led to later convergence (wrt. interations) on all problems. Note that it does not necessarily guarantee that convergenes is getting better. The first and the last findings very well match the theory of these algorithms, while the last most likely is still problem specific.

5.5 Future Work

Future analysis could involve a broader range of hyperparameters for randomized optimization algorithms. For instance, experimenting with various decay types for the simulated annealing algorithm and scaling up population size for Genetic Algorithm and MIMIC would be of interest. Delving into the trade-off between exploration and exploitation, contingent on problem sizes, holds further intrigue. Running the experiments for more iterations could help us understand potential limits of convergence on bigger problems. Additionally for the Flip-Flop problem, an in-depth investigation is warranted to validate or refute our suspicion that the outcome of a low number of function calls without convergence in MIMIC may be attributed to an implementation issue within the utilized framework.

6 NEURAL NETWORK WEIGHTS OPTIMIZATION

6.1 Results

Algorithm	Micro F1 Score			Train Time	Best Setup
	Train	Val	Test		
Gradient Descent	0.5469	0.5463	0.5238	17.22s	`lr=10^-5`
Random Hill Climbing	0.4419	0.4172	0.4298	5.27s	`restarts=0, lr=10^-1`
Simulated Annealing	0.4419	0.4172	0.4298	11.00s	`sched_init=0.05, lr=10^-1`
Genetic Algorithm	0.5182	0.5375	0.500	117.18s	`mut_prob=0.1, pop_sz=25, lr=10^-4`

Table 2—Neural Network performance results based on optimization algorithm

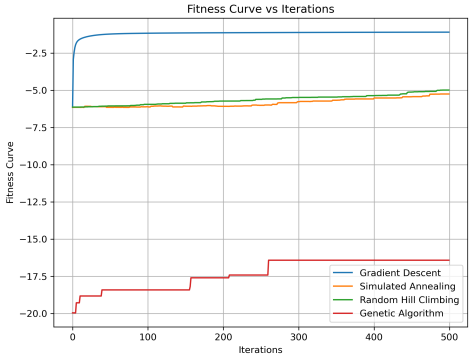


Figure 8—Fitness Curves

Table 2 summaries results of the performance of a fixed Neural Network described in Section 4 given four different algorithms for optimizing weights of a neural network. To its right, in Figure 8, we depict fitness curves of the NN's weights optimisation problem. The fitness curves are essentially negated loss curves. The best micro F1-score is yield by the Neural Network trained with the Gradient Descent algorithm. It achieved the highest scores across all three datasets. Notably, the model generalises very well to the unseen data showing F1-scores on validation and testing dataset comparable to the one calculated on training. This might indicate that we have low variance. Comparing these results to the best ones we obtained in an Assignment 1 on the Wine Quality Dataset (micro F1 score ≈ 0.63 on training dataset and ≈ 0.52 on testing), we could argue that there is still a chance to improve the model by conducting more thorough training without increasing variance reaching the balance point, however it is hard to say that the model is heavily underfitted. The change of the fitness/loss function also appears to be healthy. The possible improvements that we could do to decrease model's bias are regularization and changing vanilla gradient descent to Adam as we did in Assignment 1. From theoretical and practial experience these are usually simple ways to bring model performance up. However these are not accessible for us in the framework.

It is worth noting that high learning rates (i.e., $>= 10^{-3}$) produced very bad models that barely achieved micro F1-score of 0.01.

Next, looking at the performance of the models trained using Random Hill Climbing and Simulated Annealing we can see that the model yielded exactly the same result. Even more, while some of the hyperaprameters were able

to move the fitness curve up, none of the explored hyperparameters moved the resulting F1 scores lower or higher. Given the resulting micro F1 scores and before mentioned observation, we can conclude that the resulting models are biased (heavily underfitted), especially when comparing to the classifiers produced in Assignment 1. Given that fitness/loss curves with these algorithms barely moves, we conduct that these algorithms are not very well suitable for this task.

Last Neural Network was trained Genetic Algorithm. From Table 2 we can see that the produced model outperforms ones that were trained using RHC and SA algorithms. At the same time it does not reach the performance of the model produced by the model trained using Gradient Descent, which might mean suggest minor underfitting, but not enough to consider it serious. Indeed, the perfromance matches some of the classifiers we trained in Assignment 1 (i.e., Boosting).

Given that micro F1 scores for training, validation and testing sets are very similar and that validation score is even higher than training, we can conduct that model generalises well.

Now, let us consider the fitness curve of this model. First thing that we notice is that it has a much more different scale compared to the others. What is more important, while GA method shows better micro F1 scores than RHC and SA, its maximum fitness function value is lower. Given that fitness function is essentially a loss function (in our case log-loss), this is normal. Indeed, loss function might be considered as a "distance of an error", while micro F1 score is a number of errors. Thus, model trained using GA optimisation, tends to make less errors than models trained using RHC and SA, but if it makes an error in prediction it is a significant error (basically, this model is much more confident than others when it makes mistakes). This often happens when model's weights absolute values are relatively large that causes outputs to become larger and when output layer activation is applied (e.g., softmax or sigmoid) it results in probability predictions being very close to either 0 or 1. We looked at the total sum of absolute values of all algorithm's weights and it is indeed the case: this value for the model trained with GA is 8 times bigger compared to all the others,which means that on average the model weights absolutes are 8 times bigger as well. Ideally this type of problem should be avoided, especially to perform well in problems where we need to calculated probabilities rather than give actual predictions. This usually can be solved using regularization, but as was mentioned it was not available in the used framework. Another interesting feature of the training curve is the fact that it has a staircase like shape. Potentially, the stairs form when alorithm makes a success in mutation/crossover oepration and remains flat otherwise.

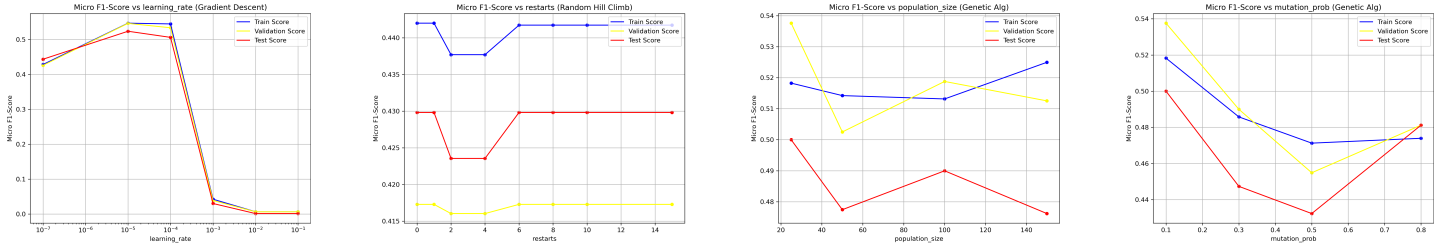
Finally, it is important to note the differences in training times. Gradient Descent required only 17.22 seconds to train the model, while other Genetic Algorithm took almost 10 times as long. While models trained using RHC and SA required even less time (5.27 and 11.00 seconds respectively), we do not want consider them at all due to a low performace.

Overall, none of the randomized optimization algorithms was able to surpass Gradient Descent and also none of them was able to provide a meaningfull trade-off in terms of time complexity.

6.2 Hyperparameters Analysis

Let us briefly examine how different values of hyperaparameters could have affected the performance of the algorithm and corresponding trained neural network. We plot validation curves for hyperparameters of different algorithms in Figure 9 (the other hyperparameters are assigned to their best values for a given algorithm). We omit plotting validation curve for initial temperature for SA and learning rate for RHC, SA, and GA since they don't produce any effect on the validation score with other parameters being fixed to their best values.

Notably, in Figure 9a we confirm that in Gradient Descent based model only values of learning rate less than 10^{-3} are reasonable to consider and that learning rate being equal to 10^{-5}. From Figure 9b, we conclude that change in the restarts hyperparameter of RHC does not affect the micro F1 score, besides maybe creating noise. While it is difficult to make definitive conclusions about the effect of population size on the micro F1 score as the graph seems to be very noise (Figure 9c), we can clearly see that lower mutation rate resulst in a higher micro F1 score (Figure 9d).



(a) Learning rate, Gradient Descent based model

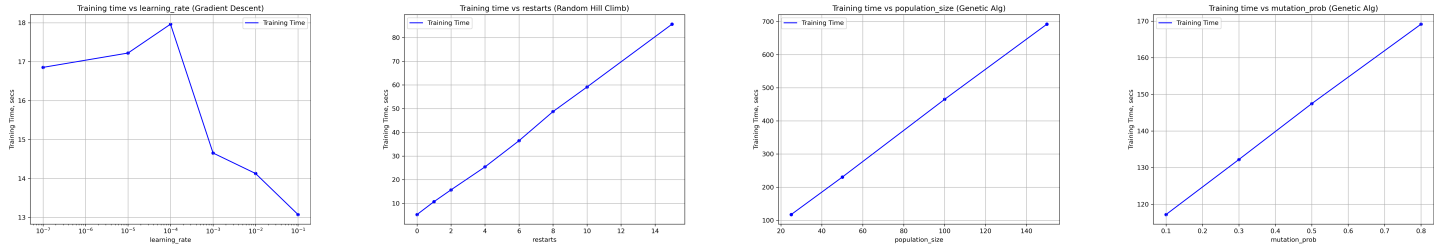
(b) Restarts, Random Hill Climbing based model

(c) Population size, Genetic Algorithm based model

(d) Mutation probability/rate, Genetic Algorithm based model

Figure 9—Validation curves

In Figure 10, we also overview effect of these parameters on the Neural Network's training time using corresponding methods. As expected, with increase of restarts in RHC and population size and mutation probability in GA, the training time grows linearly. For the Gradient Descent based model, high values of learning rate lead to faster training, however given that as we discussed above that model does not learn anything with these parameters, it is most likely due to an early termination on unsuccessful training (exceeded `max_attempts`).



(a) Learning rate, Gradient Descent based model

(b) Restarts, Random Hill Climbing based model

(c) Population size, Genetic Algorithm based model

(d) Mutation probability/rate, Genetic Algorithm based model

Figure 10—Time curves

7 CLOSING REMARKS

In this project, we examined four randomized optimization methods: Random Hill Climbing, Simulated Annealing, Genetic Algorithm, and MIMIC. We assessed their performance on discrete optimization problems (One-Max, N-

Queens, and Flip-Flop) and in the context of neural network weight optimization. Comparing these results to Gradient Descent, we uncovered differences in algorithm effectiveness across various problems. This gives us valuable practical experience guidance on the situations when each algorithm would yield the best results.