Assessment of Randomized Optimization Algorithms

Andreas Koutras

I. Introduction

This project examines the application of randomized optimization (RO) algorithms — Random Hill Climbing (RHC), Simulated Annealing (SA), and Genetic Algorithm (GA) — to different optimization tasks. The report is divided into two parts. In the first part, these RO algorithms are employed to find optimal solutions for two classic optimization problems: the One Max problem and the Travelling Salesperson Problem (TSP). In the second part, the algorithms are evaluated based on their performance in training and testing a neural network.

II. Part I: Discrete optimization problems

A. Selected problems and expectations

The performance of the RO algorithms was assessed on two optimization problems: the One Max problem and the Travelling Salesperson (TSP) problem. These problems were selected with the intent to highlight the relative strengths and weaknesses of the Simulated Annealing and Genetic Algorithms.

The One Max problem is a bit-string optimization task where the objective is to maximize the number of ones in the bit string. This problem has a single global maximum, represented by the bit string composed entirely of ones, and no local maxima. Therefore, for an N-bit string, the maximum fitness is N. The problem has no inherent structure, as the optimal value for each bit is independently 1. Therefore, it is expected to be straightforward for RHC and SA to reach the optimal solution. Although GA is also expected to find the optimal solution, its performance is more sensitive to population size, which may require more function evaluations, potentially leading to higher computational costs compared to RHC and SA.

The TSP is an optimization problem that aims to find the shortest route visiting all points in a set and returning to the starting point. The complexity of the problem increases with the number of points, N. GA is expected to perform well on this task, especially as problem size increases, since it constructs new solutions by combining previous states. If parent solutions are effective in optimizing different parts of the route, their offspring may inherit beneficial traits from both, leading to better overall solutions. In contrast, RHC and SA, which update solutions based on their neighbors, may struggle to find the optimal route as the problem size grows. To further highlight GA's capabilities, we also assessed the performance of a fourth RO algorithm, MIMIC, on the TSP. However, the TSP does not naturally exhibit the dependency structure that MIMIC relies on, so MIMIC is expected to face challenges in this context.

B. Experimental Methodology

The experiments were conducted using the implementation of the RO algorithms available in the mlrose-hiive library in Python. Since these algorithms rely on randomization, different random guesses can influence convergence. To account for this variability, each problem was repeated with 6 (or 5 for TSP) different random seeds to calculate the average performance and variance. The same set of random seeds was applied across all algorithms to ensure consistent comparisons. The performance of the RO algorithms was measured in terms of the fitness value, the number of evaluations of the fitness function (fevals), and the wall clock time, with respect to the algorithm iterations averaging the result over the different seeds.

To assess how problem size affects the performance of each algorithm, three different sizes were considered for each optimization problem: a small, medium, and large size. For the One Max problem, bit strings of length 5, 15, and 40 were chosen. For the more computationally challenging TSP problem, problem sizes of 5, 15, and 30 points were evaluated. For each problem size the topology of the TSP problem was created using the TSP generator of mlrose-hiive with a random seed of 100 and was kept the same across the runs with the multiple seeds in the solver level.

The hyperparameters of each RO algorithm were tuned using grid search, with separate tuning for each problem and problem size to demonstrate the algorithms' best capability in each case. Due to the limited computational resources, the search ranges were iteratively refined to ensure that the optimal values lay within the interior of the search space, using reasonably spaced search points. For RHC, the number of restarts was tuned, with values ranging from 0 to 10. In the case of SA, an exponential decay was used for the temperature of the probability function, and only the initial temperature (`Tinit`) was tuned, within a range of 0.001 to 10, while the residual temperature was fixed at 0.001, the default in mlrose-hiive. For GA, both population size (`pop_size`) and mutation rate (`mut_rate`) were adjusted. The population size was explored between 10 and 1500, and the mutation rate between 0.1 and 0.5. Similarly, for MIMIC, which was applied exclusively to the TSP problem, `pop_size` was tuned between 100 and 3000, and the keep percentage (`keep_perc`) between 0.1 and 0.7.

The optimal hyperparameter values were assumed to be those that produced the best fitness value in the shortest wall clock time, averaged over the multiple random seeds. Convergence was determined from the average fitness-vs.-iterations plots, with the iteration of convergence identified as the iteration at which the average fitness value stabilized without further improvement. In the implementation of mlrose-hiive, the RO algorithms terminate when either a specified maximum number of iterations is reached or a limit on consecutive iterations without fitness improvement (max. attempts) is exhausted, whichever occurs first. During our trials, these termination criteria were carefully adjusted for each problem to ensure sufficient iterations for convergence while avoiding unnecessary computational costs.

C. Results and discussion

1) The One Max problem

Table I summarizes the average fitness, function evaluations (fevals), and wall clock time at the point of convergence for the RHC, SA, and GA algorithms across the three problem sizes, along with their tuned hyperparameters. Fig. 1 compares the fitness, fevals, and wall clock time curves of the algorithms for the medium problem size of One Max. The point of convergence is marked on the curves. The following observations and conclusions can be drawn from the results.

As shown in Table I, the three algorithms successfully converged to the optimal fitness across all problem sizes. However, the impact of problem size on the number of iterations, function evaluations (fevals), and wall clock time varied between algorithms. RHC and SA were more efficient in terms of wall clock time for small and medium-sized problems, but GA outperformed them for the large problem.lthough GA consistently required fewer iterations to converge compared to RHC and SA, its larger population size increased the number of function evaluations, making it more computationally expensive for the small and medium problem. Our trials indicated that increasing the GA population size leads to more function evaluations but reduces the number of iterations. This is because a larger population allows GA to explore more candidate solutions and perform additional crossover and mutation operations in each iteration, increasing the likelihood of quickly discovering the optimal solution. For the large problem size, tuning showed that a population size of 100 offered an optimal balance between the number of iterations and function evaluations, resulting in the shortest wall clock time (0.434 s), outperforming SA, which had the second fastest time. For comparison, using a population size of 20 instead of 100 (with the same `mut_rate`=0.3) would increase the number of iterations by 3.2 times, reduce fevals by 36%, and extend the wall clock time by 51%. Conversely, with a population size of 200, the number of iterations would remain unchanged, but the fevals would double, and the wall clock time would increase by 50% compared to using a population size of 100.

For the SA algorithm, the grid search resulted in an initial temperature of 0.001 across all three problem sizes, which matches the residual temperature value used in mlrose-hiive. This means that the temperature function remains uniform throughout the iterations. A low temperature like this reduces randomness, favoring exploitation over exploration during solver iterations. Despite this small temperature, the behavior of SA differs from RHC. SA required fewer iterations for the medium and large problem sizes, leading to similar and, in the large problem, shorter wall clock times compared to RHC. However, RHC was more efficient for the small problem size. In other words, the limited randomness induced by the low temperature improves SA's efficiency in medium and large problem sizes relative to RHC but diminishes its performance in small problems. This occurs because the One Max problem has a single optimal solution, and in smaller problem sizes, any randomization tends to divert the solver away from the optimum rather than helping to find it.

TABLE I: Convergence iteration, average fitness, function evaluations, and wall clock time at convergence for the tuned algorithms across the One Max problem sizes. The values in parentheses represent the standard deviation for each metric.

(a) One Max small problem size (5 bits)

Algorithm	Hyper-parameters	Convergence Iteration	Fitness	Fevals	Wall clock time
RHC	restarts = 0	10	5.0 (0.0)	12.5 (0.55)	0.010 (0.001)
SA	Tinit = 0.001	17	5.0 (0.0)	19.5 (0.55)	0.022 (0.002)
GA	pop_size = 50 mut_rate = 0.2	1	5.0 (0.0)	102 (0.0)	0.025 (0.006)

(b) One Max medium problem size (15 bits)

Algorithm	Hyper-parameters	Convergence Iteration	Fitness	Fevals	Wall clock time
RHC	restarts = 0	81	15.0 (0.0)	87.83 (1.47)	0.112 (0.001)
SA	Tinit = 0.001	70	15.0 (0.0)	76.83 (1.47)	0.113 (0.012)
GA	pop_size = 50 mut_rate = 0.2	8	15.0 (0.0)	461.8 (1.17)	0.159 (0.017)

(c) One Max large problem size (40 bits)

Algorithm	Hyper-parameters	Convergence Iteration	Fitness	Fevals	Wall clock time
RHC	restarts = 0	247	40.0 (0.0)	253.8 (1.5)	0.551 (0.001)
SA	Tinit = 0.001	222	40.0 (0.0)	242.2 (2.1)	0.473 (0.024)
GA	pop_size = 100 mut_rate = 0.3	18	40.0 (0.0)	1927.3 (1.2)	0.434 (0.049)

It is noteworthy to mention that the RHC algorithm was used without any restarts, as our trials demonstrated that adding restarts only increased the total wall clock time. This is because the One Max problem lacks local optima, so the

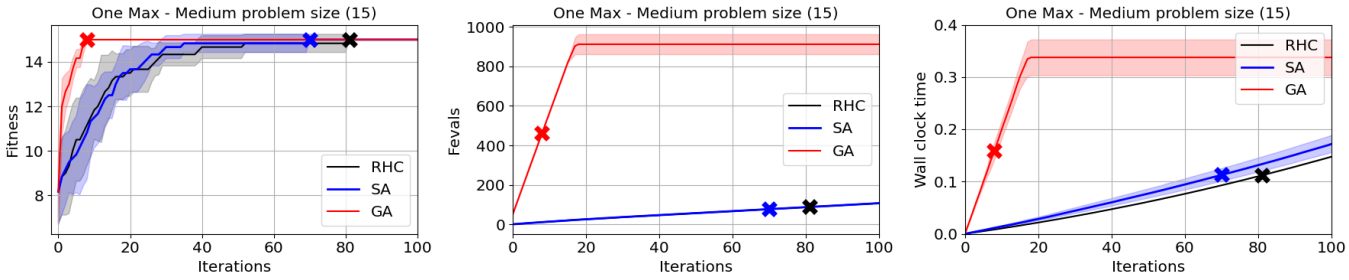

Fig. 1: Comparison of RHC, SA, and GA on the One Max problem (medium size) in terms of fitness, function evaluations, and wall clock time across iterations. X markers indicate the iteration at which each algorithm converged. The shaded areas represent one standard deviation above and below the average curve.

solution is not at risk of becoming trapped, even without restarts.

In summary, our findings for the One Max problem confirm that all three algorithms successfully converge to the optimal solution, as hypothesized. SA proved to be the most efficient for medium problem sizes, while GA demonstrated greater efficiency for larger problems, provided its population size and mutation rate were carefully tuned to balance the trade-off between the number of iterations and function evaluations.

2) The TSP problem

Table II summarizes the fitness, fevals, and wall clock time at the point of convergence of the RHC, SA, GA, and MIMIC algorithms along with their tuned hyperparameters for the three TSP problem sizes. Fig. 2 compares the corresponding fitness, fevals, and wall clock time curves for large TSP problem size. Although the TSP is inherently a minimization problem (finding the shortest path), the results are presented as a maximization problem by taking the negative of the fitness function to maintain consistency with the One Max problem. The following observations and conclusions can be drawn from these results.

In the small problem size, all four algorithms successfully converged to the actual solution. The RHC had the shortest wall clock time, as in the One Max small problem size, and was followed by the GA. The optimal `Tinit` for the SA remained at the minimum value of 0.001, as increasing the level of exploration (randomness) only led to more iterations without benefit, a pattern also observed in the One Max problem.

For the medium problem size, the algorithms no longer converged to the same fitness value, highlighting the increased complexity compared to the small problem size. GA achieved the best fitness, while SA, RHC, and MIMIC produced fitness values that were 5%, 6%, and 8% lower, respectively. Notably, GA's fitness is likely the theoretical optimum, as indicated by the zero standard deviation, meaning the GA consistently reached the same fitness across all random seeds. In terms of runtime, GA was only outpaced by RHC, which was three times faster.

For the large problem size, the superiority of GA becomes clear, achieving both the best fitness and the shortest wall clock

TABLE II: Convergence iteration, average fitness, function evaluations, and wall clock time at convergence for the tuned algorithms across the TSP problem sizes. The values in parentheses represent the standard deviation for each metric.

(a) TSP small problem size (5 points)

Algorithm	Hyper-parameters	Convergence Iteration	Fitness	Fevals	Wall clock time
RHC	restarts = 0	10	-361.0 (0.0)	11.6 (0.89)	0.0148 (0.005)
SA	Tinit = 0.001	17	-361.0 (0.0)	19.0 (1.0)	0.0239 (0.003)
GA	pop_size = 100 mut_rate = 0.2	1	- 361.0 (0.0)	202.0 (0.0)	0.0158 (0.003)
MIMIC	pop_size = 100 keep_perc = 0.5	1	-361.0 (0.0)	202.0 (0.0)	0.0457 (0.005)

(b) TSP medium problem size (15 points)

Algorithm	Hyper-parameters	Convergence Iteration	Fitness	Fevals	Wall clock time
RHC	restarts = 0	564	-878.1 (26.5)	587.6 (2.9)	1.92 (0.07)
SA	Tinit = 5.0	1026	-866.8 (46.9)	1055.2 (6.4)	83.4 (3.0)
GA	popu_size = 300 mut_rate = 0.4	43	-825.4 (0.0)	13257.6 (1.3)	5.93 (0.33)
MIMIC	pop_size = 700 keep_perc = 0.3	15	-892.6 (47.3)	11222.8 (1.9)	10.93 (0.39)

(c) TSP large problem size (30 points)

Algorithm	Hyper-parameters	Convergence Iteration	Fitness	Fevals	Wall clock time
RHC	restarts = 9	1876*	-1369.2 (73.3)	19118.4 (679.9)	1812.3 (81.1)
SA	Tinit = 5.0	1378	-1480.0 (133.8)	1428.4 (7.7)	121.1 (12.1)
GA	pop_size = 1000 mut_rate = 0.2	213	-1113.2 (13.0)	214266.4 (2.7)	56.8 (0.6)
MIMIC	pop_size = 3000 keep_perc = 0.4	64	-1525.3 (39.4)	195083.8 (4.8)	316.3 (12.0)

*The RHC iterations displayed correspond to the 9th restart run, while the function evaluations and total time reflect the cumulative values across all restarts.

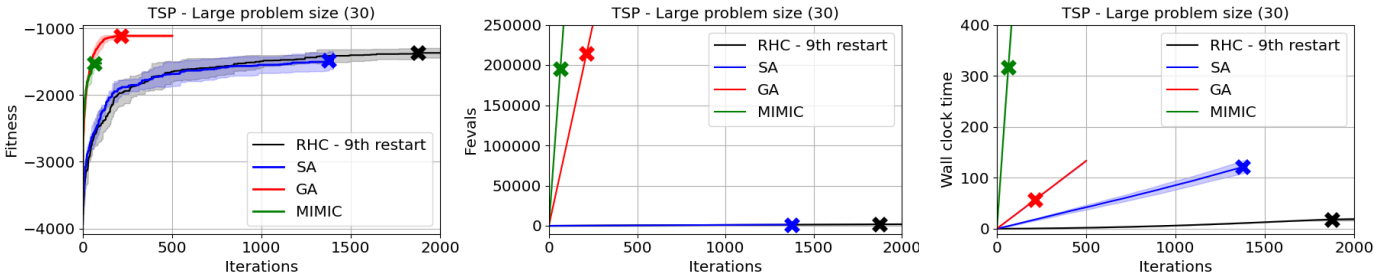

Fig. 2: Comparison of RHC, SA, GA, and MIMIC on the TSP problem (large size) in terms of fitness, function evaluations, and wall clock time across iterations. The results for RHC correspond to the 9th restart run. X markers indicate the iteration at which each algorithm converged.

time for convergence, as shown in Table IIc. The fitness gap between GA and the other algorithms is more pronounced than in the medium problem size, with RHC, SA, and MIMIC attaining a worst fitness by 23%, 33%, and 37%, respectively. These results strongly support our initial hypothesis that GA would perform best on the TSP problem, especially as the problem size increases.

By examining the hyperparameters of the tuned models, we observe a general trend: as the problem size increases, hyperparameters that promote greater exploration of the solution space become more effective. For example, in the large problem size, RHC improved its fitness by 13% when 9 restarts were used compared to no restarts, whereas restarts provided no benefit in smaller problem sizes. This suggests that larger problems are more likely to contain local optima that can trap the solution if insufficient randomization is applied. Similarly, the SA algorithm benefited from a higher initial temperature in the medium and large problem sizes, allowing for greater exploration and a better ability to overcome local optima. Both GA and MIMIC also performed better with larger population sizes as the problem size increased. However, no clear trend was observed for GA's mutation rate in relation to problem size based on these results.

The MIMIC algorithm exhibited the worst performance, achieving the lowest fitness in both the medium and large problem sizes, and the longest wall clock time in the small problem size. This outcome was anticipated, as outlined in our initial hypothesis, since the TSP problem cannot be effectively modeled by a dependency tree, which is central to the MIMIC method's formulation. An example of a more suitable optimization problem for demonstrating MIMIC's strength is the Flip Flop problem, where the goal is to maximize the number of bit alterations in a given bit string. In this case, the optimal solution depends on the bit sequence structure rather than individual bit values, since the first bit in the optimal alternating sequence can be either 0 or 1.

III. Part II: Application on a Neural Network

A. Dataset and expectations

To evaluate the capability of the RO algorithms in training a neural network (NN), we selected the "Vehicle Silhouettes" dataset from the UCI Machine Learning Repository for a classification task. The dataset contains 845 samples with 18 features representing various geometrical properties of vehicles, and the target contains four vehicle classes (1: bus, 2: opel, 3: saab, 4: van). In a prior project, a NN model trained using backpropagation and gradient descent (GD) in sklearn achieved ROC-AUC scores of 0.979 for training and 0.951 for testing on this dataset. Given the strong performance of GD, we anticipate that, with careful hyperparameter tuning, the RO algorithms will be able to at least match or approach GD's results. Additionally, the training wall clock time will be a key factor in selecting the most efficient algorithm.

In this project, for simplicity and compatibility with the mlrose-hiive neural network implementation, we converted the multi-class target to a binary classification task by assigning a label of 1 to the second vehicle class ("opel") and 0 to all others. This resulted in a dataset with 633 samples in Class 0 and 212 samples in Class 1, reflecting a 3:1 class ratio. The data was shuffled and split into a training set (70%) and a test set (30%) using stratified sampling to preserve the class proportions. To ensure that all features are on the same scale, the features were standardized to have zero mean and unit variance. This standardization was performed using the mean and variance computed from the training set, and the same transformation was subsequently applied to the test set. No other pre-processing was done on the data.

B. Experimental methodology

The mlrose-hiive NN optimization library, which has the RO algorithms asjusted for continous optimization problems, was used for training. The neural network was first trained using gradient descent with backprop to determine its architecture and establish a baseline for performance comparison. Once the architecture was set, the network was then trained using the RHC, SA, and GA algorithms to assess their effectiveness. To tune the hidden layers, number of nodes, and the algorithms' hyperparameters, 20% of the training set was reserved as a validation set, leaving 80% of the data for training. While K-fold cross-validation would have provided broader exposure to training data, we opted for a single validation set to prioritize computational efficiency. The hyperparameter tuning was done in terms of the training loss and the validation ROC-AUC score. Additionally, to account for the randomized

nature of the RO algorithms, each experiment was conducted using five different random seeds, consistent across all three algorithms. The performance metrics reported are the average values across the multiple seeds. Convergence was determined from the average fitness curve. The tuning process for each algorithm is detailed in the following section, and the final section presents a comparison of the performance of the resulting neural network models.

C. Hyperparameter tuning

1) Gradient descent and NN architecture

The architecture of the neural network, including the number of hidden layers and nodes, was determined using GD with backpropagation provided by the mlrose-hiive library. A network with a single hidden layer containing four nodes was found to yield the highest validation ROC-AUC score, and this architecture was adopted for all subsequent experiments. The learning rate was set to 0.0002 after evaluating the loss-vs-iteration curves for different values, as shown in Fig. 3. This learning rate achieved faster convergence compared to lower values, while avoiding the instabilities observed with larger rates. To ensure sufficient convergence, the maximum number of iterations was set to 40,000, based on the observed loss curve behavior. The corresponding loss value is 0.18 and the resulting ROC-AUC training and validation scores are 0.975 and 0.904 respectively.

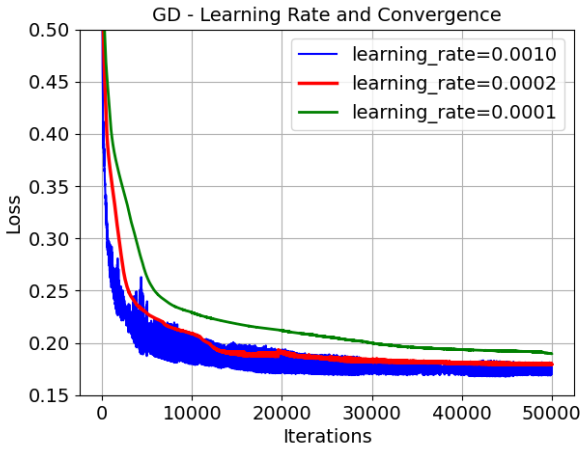

Fig. 3: Impact of the learning rate on the convergence behavior of the loss during gradient descent.

2) Random hill climbing

The behavior of RHC was first analyzed with respect to the step size and the maximum attempts (`max_attempts`) parameters. Choosing an appropriate step size is crucial for enabling the algorithm to approach the minimum loss solution without making it too small, which would unnecessarily prolong computation with minimal gains. Likewise, setting an effective `max_attempts` value aids in detecting convergence early, allowing the algorithm to terminate without performing redundant iterations. Figure 4 displays the loss curves for different step sizes, with no restarts and 400 `max_attempts`. A step size of 0.005 achieved nearly the same loss as 0.001 but with significantly fewer iterations, while larger step sizes led to higher final losses. Therefore, a step size of 0.005 would be the most efficient choice when no restarts are done and if minimizing the loss is the only goal. The `max_attempts` value of 400, identified through experimentation, was found to be sufficient for ensuring termination after convergence. For comparison, the dotted lines in Fig. 4 show the loss behavior when the max attempts termination criterion is effectively deactivated."

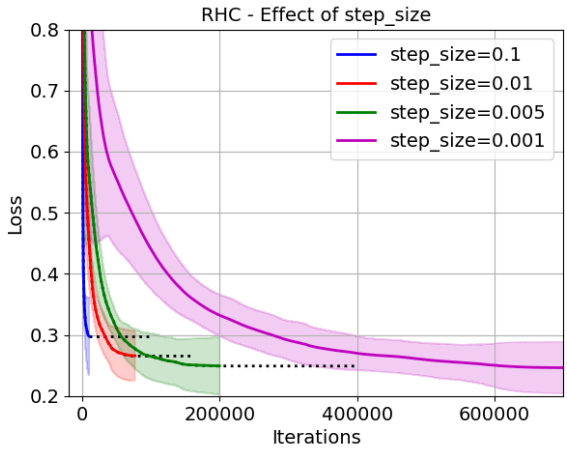

Fig. 4: Effect of step size on RHC convergence with a maximum of 400 attempts. Shaded areas indicate the range of two standard deviations above and below the mean, based on multiple random seed trials.

Next, the impact of restarts was investigated in combination with step size. The hypothesis being tested is that while larger step sizes may not reach the minimum loss without restarts, as shown in Fig. 4, incorporating restarts could enable the algorithm to achieve a better loss. This may also result in a shorter wall clock time compared to using the optimal step size of 0.005 without restarts.

Fig. 5 presents the average final loss after zero to 10 restarts for step sizes 0.1, 0.01, and 0.005. The corresponding total wall clock times are shown in Fig. 6. he largest step size exhibits more fluctuation in loss compared to the smallest step size, likely because the larger step overshoots the optimum to varying degrees depending on the restart position. In contrast, the smaller step size consistently approaches the minimum loss through incremental adjustments. However, when tuning a neural network, the focus should be on validation scores to ensure hyperparameter settings that lead to better generalization on unseen data.

As shown in Fig.7, there is a noticeable improvement in the ROC-AUC validation score for the 0.1 step size when three or more restarts are used, surpassing the scores of smaller step sizes. Furthermore, the wall clock time for step size 0.1 with three restarts is still shorter than that of smaller step sizes with zero restarts, as illustrated in Fig.6. Therefore, we selected the step size of 0.1 and three restarts as the final

configuration for RHC. The corresponding average fitness is 0.283 and the ROC-AUC training and validation scores are 0.932 and 0.897 respectively.

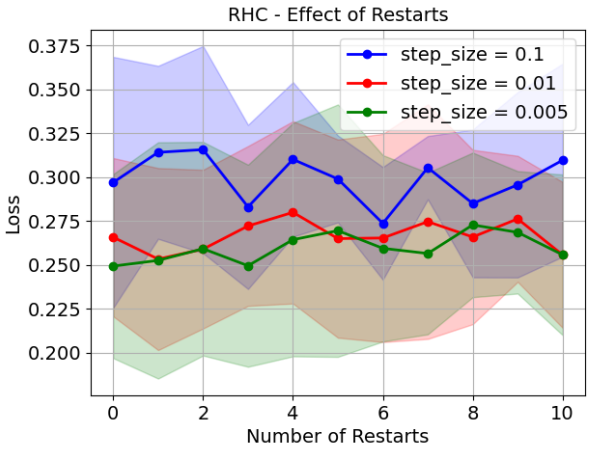

Fig. 5: Effect of number of restarts on the final loss of RHC depending on the step size.

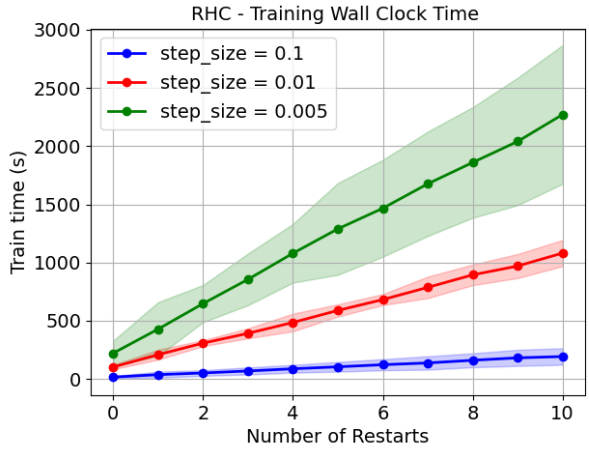

Fig. 6: Wall clock time for training with RHC with respect to restarts and step size.

3) Simulated annealing

For the SA algorithm, an exponential decay temperature schedule was used. The key hyperparameters tuned were the step size, initial temperature (`Tinit`), and the exponent constant (`exp_const`) for the decay schedule, while the residual temperature was kept at 0.001, the default value in mlrose-hiive. Fig. 8 presents the loss curves for different step sizes, using the default values of `Tinit`=1.0 and `exp_const`=0.005. The smallest step sizes (0.005 and 0.01) required an excessive number of iterations to converge, making them computationally expensive, while the largest step size (0.1) converged to a higher loss. Based on this, a step size of 0.01 was selected for training the neural network. Due to the noisy nature of the loss curves, using the maximum attempts

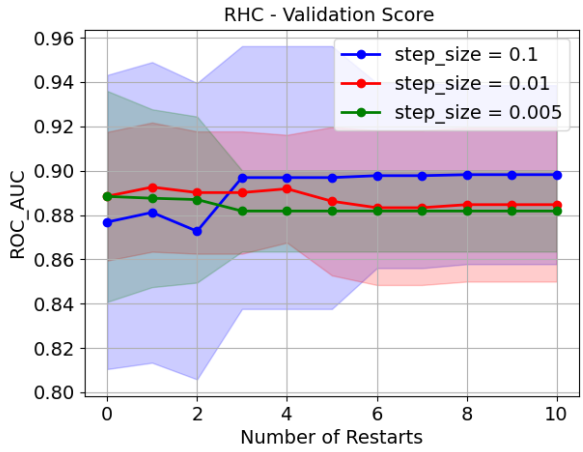

Fig. 7: Validation ROC-AUC scores as a function of the number of restarts and step size used in the model training with RHC.

as a termination criterion was impractical, so the maximum number of iterations was set to 100,000 for tuning and 400,000 for the final training.

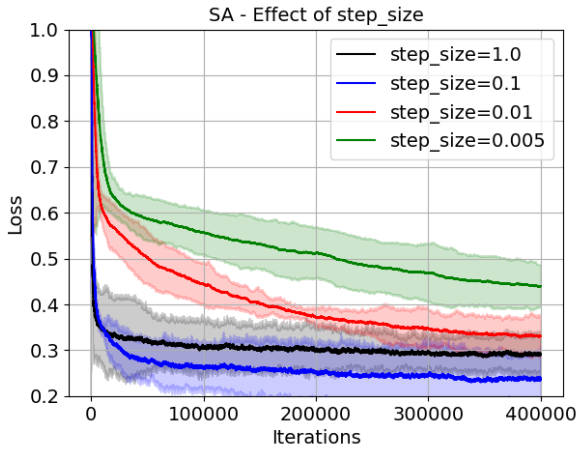

Fig. 8: SA loss curves for different steps sizes depicted for `Tinit`=1.0 and `exp_const`=0.005.

Next, the final loss and validation ROC-AUC score were evaluated with `Tinit` values ranging from 0.001 to 10.0, and `exp_const` set to 0.05, 0.005, and 0.0005. As shown in Fig. 9, no clear trend emerged from these combinations. For training the neural network, we selected the parameter combination that yielded the highest validation score, namely `Tinit`=1.0 and `exp_const`=0.0005. The resulting final loss is 0.259 and the training validation ROC-AUC scores are 0.94 and 0.90 respectively. The corresponding elapsed wall clock time for 100,000 training iterations was 219.9 s.

4) Genetic algorithm

The hyperparameters tuned for the GA algorithm were population size and mutation rate, as GA does not require

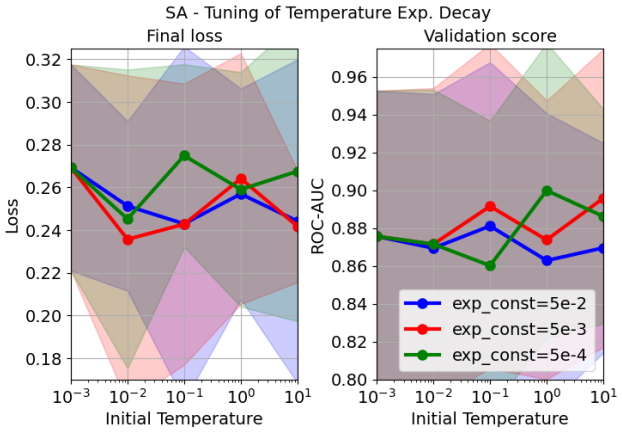

Fig. 9: SA final loss (left plot) and validation ROC-AUC score (right plot) with respect to the initial temperature and exponent constant of the temperature decay schedule.

a step size. We used a maximum attempt threshold of 400 as the termination criterion, which effectively detected convergence. As shown in the left plot of Fig. 10, increasing the population size reduced the loss, as a larger population allows the algorithm to evaluate more samples from the solution space and perform additional crossover operations. This trend is consistent with findings from the discrete optimization problems, where larger population sizes are beneficial for more complex problems. However, increasing the mutation rate had a detrimental effect on the loss, as shown in the right plot of Fig. 10. The additional randomness introduced by higher mutation rates appears to push the solution further from the optimal loss for this specific problem. This suggests that the impact of mutation rate is problem-dependent, since for discrete optimization problems, mutation rates between 0.2 and 0.4 were found to be effective. Regardless, the GA converges in significantly fewer iterations compared to GD, RHC, and SA (see Fig. 3, 4, 8, 10), which also aligns with its behavior in the discrete optimization problems. As shown in Fig. 11, the optimal hyperparameters for minimizing the loss and maximizing the validation ROC-AUC score were a population size of 700 and a mutation rate of 0.10. Using these settings, the final fitness was 0.342, with training and validation ROC-AUC scores of 0.897 and 0.879, respectively, and a total training time of 3,314 seconds.

D. Model training and comparison

Finding the optimal weights for a neural network involves selecting a model that not only fits the training data well but also generalizes effectively to unseen data, achieving a balance between bias and variance. To assess this, we will first compare the performance of the tuned models by examining the learning curves, focusing on the training and validation scores as a function of the number of training samples. Following this, we will evaluate the models' performance on the complete training set and test set to assess their overall effectiveness.

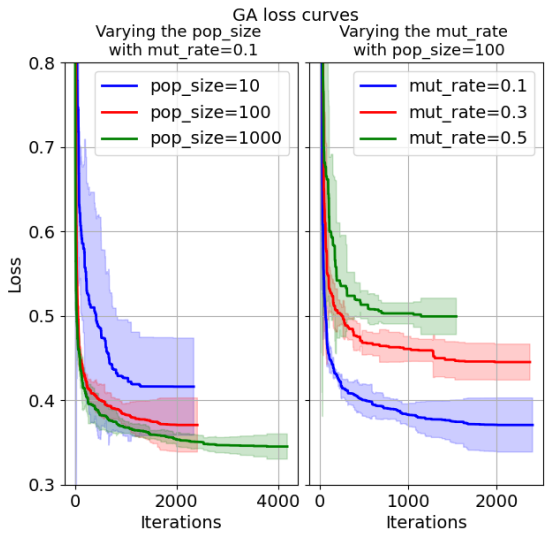

Fig. 10: Loss curves of GA for varying the population with `mut_rate`=0.1 (left plot) and varying the mutation rate with `pop_size`=100 (right plot). Max. attempts of 400 was used to detect convergence.

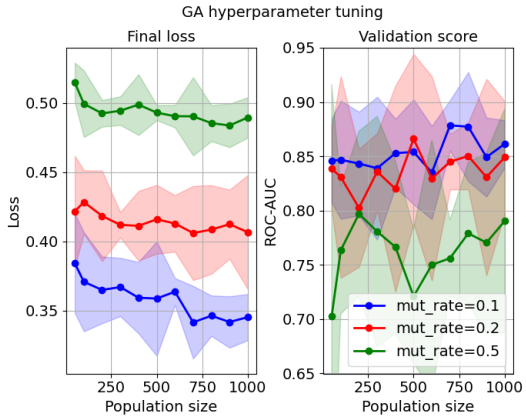

Fig. 11: GA final loss (left plot) and validation ROC-AUC score (right plot) with respect to the population size and mutation rate.

Fig. 12 shows the learning curves obtained from GD and the three RO algorithms. GD achieved the highest training and validation scores overall. RHC and SA performed similarly, with RHC yielding a slightly higher validation score and SA achieving a higher training score. Although GA produced the lowest training and validation scores, it exhibited less variance between these curves compared to the other models, making its performance more predictable for unseen data. Notably, GA overfitted the training data less when fewer samples were used, leading to better validation scores in smaller datasets. For instance, with 200 training samples, GA outperformed both RHC and SA in validation score and matched or slightly exceeded GD's validation score. Cross-validation with a sufficient number of folds would have enabled the model to learn

from training samples that are currently only included in the single validation set. This would likely reduce the models' variance avoiding the risk of overfitting to a particular data split.

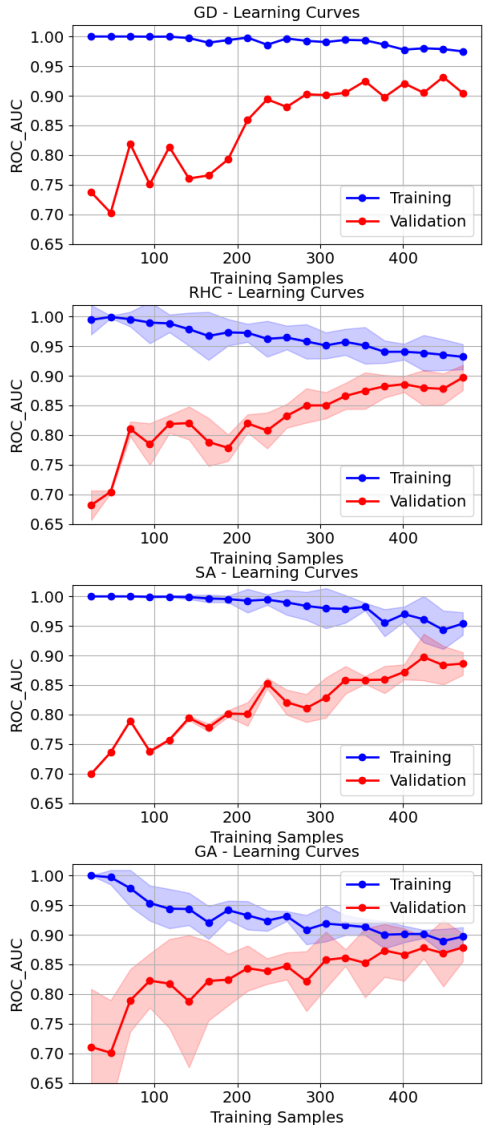

Fig. 12: Learning curves for GD, RHC, SA, and GA as a function of the number of training samples, evaluated using the full validation set.

Table III summarizes the results from the final training and testing of the models, using the complete training and test sets, along with the tuned hyperparameters for each solver. The results show that the RO algorithms do not offer no clear advantage over gradient descent (GD) in improving the NN's performance. GD achieved the highest test score, closely followed by SA, while GA produced the lowest scores across all metrics. The training scores and the loss function show the same trend, where GD achieved the lowest loss and GA the highest. In terms of training wall clock time, RHC was the fastest, completing training 30% faster than GD, even after 3 restarts. In contrast, SA took nearly nine times the time of GD to train the model, offsetting any potential advantage from its competitive test score. GA, although requiring fewer iterations to converge, was the least efficient overall due to the additional function evaluations needed for its population-based approach.

TABLE III: NN final training and testing metrics

Algo.	Hyper-parameters	Iterations	Train time (s)	Train loss	Train roc-auc	Test roc-auc
GD	learn_rate=2e-4 max_iter=4e4	40,000	105.9	0.184	0.974	0.908
RHC	step_size=0.1 restarts=3 max_attem=400	20,573	73.24 (22.03)	0.303 (0.013)	0.932 (0.014)	0.869 (0.023)
SA	Tinit=1.00 exp_cons=5e-4 max_iter=4e5	400,000	908.5 (18.33)	0.256 (0.025)	0.944 (0.012)	0.900 (0.027)
GA	pop_size = 700 mut_rate = 0.1 max_attem=400	4,863	3163.9 (896.4)	0.350 (0.020)	0.892 (0.016)	0.837 0.021)

*Values in parentheses are standard deviations. The RHC iterations displayed correspond to the 3rd restart run, while the total time reflect the cumulative values across all restarts.

IV. Conclusions

This report evaluated the performance of random optimization (RO) algorithms in two discrete optimization problems and in training a neural network (NN). In the One Max problem, the results were not far from our initial hypothesis. We anticipated that all algorithms would achieve the theoretical fitness value, which was indeed the case, and that the key distinction between algorithms would be in terms of function evaluations and wall clock time. However, we initially expected SA, that has no population evaluations, to be more efficient than GA across all problem sizes. This was true for the medium problem size, but after careful tuning, GA outperformed SA for the larger problem. In TSP, our expectation that GA would achieve the best fitness was confirmed by the results.

For the NN, we anticipated that the RO algorithms would produce results comparable to gradient descent (GD), yielding similar training and test scores by the least. However, the performance of the RO algorithms did not fully meet these expectations. SA was a close second after the GD having a test score difference of less than 1%. This suggests that with a more extensive parameter grid search, varying also the residual temperature, or by also examining a geometric decay temperature schedule, the SA could surpass the test score of GD. To improve GA's performance, we propose investigating population sizes larger than 1,000 and mutation rates lower than 0.10, as our analyses indicated a trend that GA's fitness improves as population size increases and mutation rate decreases. Additionally, implementing K-fold cross-validation for hyperparameter tuning would likely yield more reliable training metrics and validation scores, further enhancing the model's generalization capabilities.