CS7641 ML Randomized Optimization

Abanoub Abdelmalak
Computer Science, Machine Learning Department
Georgia Institute of Technology
abanoub_abdelmalak@gatech.edu

***Abstract*—This paper explores the application of randomized optimization (RO) algorithms, specifically Randomized Hill Climbing (RHC), Simulated Annealing (SA), and Genetic Algorithms (GA), on two classic optimization problems: the Flip Flop problem and the N-Queens problem. These problems are chosen to illustrate the strengths and weaknesses of each algorithm in different contexts—Flip Flop for binary string optimization and N-Queens for combinatorial constraint satisfaction. Further, we extend our investigation to evaluate the effectiveness of these algorithms in optimizing neural network hyperparameters using a stock market dataset. The study demonstrates the adaptability of RO algorithms to diverse problem domains and highlights their potential in enhancing neural network performance through effective hyperparameter tuning.**

***Index Terms*—Randomized Optimization, Genetic Algorithms, Simulated Annealing, and Neural Networks.**

I. Introduction

Machine learning (ML) is a rapidly evolving field, and there are numerous approaches to optimize the performance of the model to best fit the given problem. While deterministic approaches are undeniably useful and form the foundation of many algorithms, they often fall short when confronted with the chaotic and unpredictable nature of real-world environments. Deterministic methods can easily get trapped in local optima, especially if the initial starting point is not ideal.

In contrast, RO algorithms offer a robust alternative by leveraging randomness to explore the solution space more thoroughly. These algorithms are designed to avoid the pitfalls of local optima by probabilistically accepting suboptimal solutions and thus are capable of escaping from local traps. This characteristic is particularly beneficial in complex and high-dimensional spaces, where the search landscape is rugged and filled with numerous local optima.

The paper first introduces the RO algorithms that we are going to analyze. Then, we introduce the two classical optimization problems that have been selected to study the algorithms' performance. We also introduce the stocks market dataset that is being used to monitor the neural networks performance with these algorithms. Afterwords, we present the comparitive analytical results for their performance across these different problems. Finally, we summarize our findings in the conclusion section.

II. Optimization Problems Introduction

In this section, we introduce the optimization problems selected to study the influence of RO in different contexts, their data source, and the expected challenges to mitigate.

A. *Implementation and Dependencies*

Throughout the project, the optimization problems and algorithms were implemented using the *mlrose-hiive* library. This library provides a flexible and efficient framework for implementing and testing various optimization algorithms. The following features of *mlrose-hiive* were particularly beneficial:

- Comprehensive Algorithm Support: mlrose-hiive supports a wide range of optimization algorithms, including RHC, SA, and GA, making it ideal for comparative studies.
- Ease of Use: The library offers a straightforward API, simplifying the implementation of complex optimization problems and the execution of multiple algorithmic runs.
- Customizability: Users can easily adjust hyperparameters and experiment with different configurations, facilitating in-depth analysis of algorithmic performance.

This link refers to the documentation used to manage the challenges found along with manual code debugging inside the library.

B. *Flip Flop Problem (FFP)*

1) Introduction: The FFP is a binary optimization challenge involving finding a binary string that maximizes the number of consecutive differing bits ('01' or '10' transitions). It provides a valuable benchmark for evaluating optimization algorithms due to its simplicity and the clear analysis it allows of how well algorithms can explore the solution space and avoid local optima.

2) Optimization Objective: The primary objective in the FFP is to maximize the number of consecutive differing bits in a binary string. For a binary string of length N, the fitness function to be maximized is the count of '01' and '10' transitions within the string.

3) Challenges: The FFP presents unique challenges:

- *Local Optima:* Numerous local optima where small changes might not lead to immediate improvements.
- *Binary Space Exploration:* Requires robust strategies to find configurations with the maximum number of transitions.

4) Evaluation Criteria: To solve the FFP, an optimization algorithm must:

- Efficiently explore the binary search space.
- Navigate local optima to avoid suboptimal solutions.
- Balance exploration and exploitation to find the global optimum.

C. *N-Queens Problem (NQP)*

1) Introduction: The NQP involves placing N queens on an N×N chessboard so that no two queens threaten each other. This classic combinatorial optimization problem tests the efficiency of algorithms due to its complex constraint satisfaction nature.

2) Optimization Objective: The primary objective is to place queens without conflicts, ensuring no two queens share the same row, column, or diagonal. The complexity increases exponentially with N, making it challenging for simple deterministic methods.

3) Challenges: Solving the NQP requires efficient navigation of the large search space and effective constraint handling. Key challenges include:

- *Combinatorial Nature:* Exponential growth in possible placements with increasing N.
- *Constraint Satisfaction:* Managing row, column, and diagonal constraints simultaneously.

4) Evaluation Criteria: To solve the NQP, an optimization algorithm must:

- Efficiently explore the search space for valid placements.
- Resolve conflicts to ensure no two queens threaten each other.
- Balance exploration and exploitation to avoid local optima and find the global optimum.

D. *Neural Networks Dataset*

1) Introduction: Stock market prediction (SP) is a fascinating field in financial data analysis and has been a subject of ML algorithms study for a long time. This work has addressed this problem by analyzing data from 2009 to 2021 for stock symbol "JPM." Optimizing neural network weights is a critical task in ML, particularly for tasks such as stock market prediction. The problem involves adjusting the weights of a neural network to minimize the prediction error on a given dataset. Stock market data is typically complex, nonlinear, and volatile, making it a challenging domain for accurate prediction. The neural network must learn from historical data to predict future stock prices, requiring precise tuning of its parameters.

2) Data Source: The source for the database is here.

3) Dataset Features Details: The stock market relies on several technical indicators that signal the buy and sell signals. These indicators are used as features for a stock on a given day, and the corresponding target value is whether to buy or sell. Key indicators include:

- Moving Averages (MA): Shows the average price of a stock over a certain period.
- Bollinger Bands**:** Measures volatility.
- Relative Strength Index (RSI): Helps determine whether a stock is overbought or oversold.
- Volume Weighted Average Price (VWAP): Considers both the price and volume of trades, providing insight into the overall trend of a stock.
- Other Common Indicators: Include MACD (Moving Average Convergence Divergence), Stochastic Oscillator, and Fibonacci retracement levels.

The following features are used to calculate the buy/sell signal lines:

- Simple Moving Averages (SMA) with window referring to number of averaging days: SMA10, SMA20, and SMA50.
- Exponentially Weighted Moving Averages (EWMA) with window referring to number of averaging days: EWMA12, and EWMA26.
- Returns and Indicators: 26_day_return and 12_day_return, MACD, mad_histogram, and signal_line.

The generated buy/sell signal results are represented as:

- **1 :** Long the stock
- **-1:** Short the stock
- **0 :** Hold the position

In our analysis, we focus on how well the predictions of the ML performance can better predict future signals based on previous history compared to the signal generated from previously mentioned features.

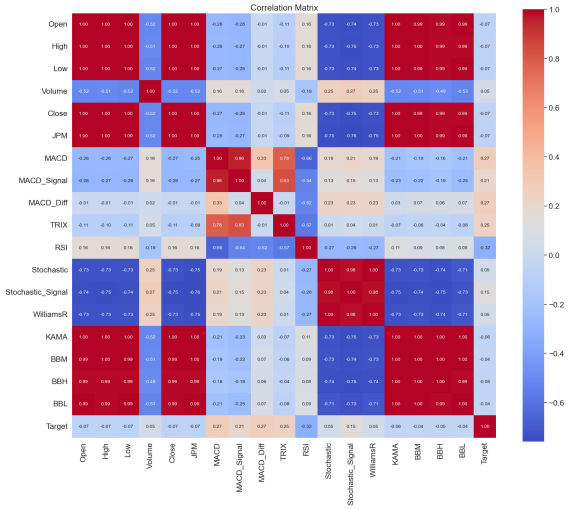


Fig. 1: SP-JPM Feature Heat Map

Figure 1 presents the correlation matrix between the features and the target. This heat map is a good indication of noise in the dataset. When analyzing the bottom row "Target" to the features columns, we see the feature "MACD" has the highest score tied with "MACD_Diff" in correlation with the "Target" column. This makes sense since some of these features are driven from each other and the final "Target" is heavily calculated from them as well while prioritizing the "MACD" signal for changing the stock position (i.e. "Target"). Looking at how the "MACD" feature is affected by other features, we can fire the chain reaction to see the correlation of the "MACD" signal with the "Trix" value and so on comparing the highest correlation values in columns and features. As we mentioned, these features are all driven from the "AdjClose" (i.e. "JPM" feature), therefore, the correlation between the "JPM" and all the other features is almost close to "1".

4) Optimization Objective: The primary objective in neural network weight optimization is to minimize the loss function, typically measured as the prediction error on the training and validation datasets. For stock market prediction, this involves:

- Reducing the mean squared error (MSE) between predicted and actual stock prices.
- Enhancing the model's ability to generalize from historical data to unseen future data.

5) Challenges: Optimizing neural network weights for stock market prediction presents several challenges:

- Non-convex Optimization: The loss surface of neural networks is highly non-convex, with numerous local minima, making it difficult to find the global minimum.
- High Dimensionality: Neural networks, especially deep ones, have a large number of weights, creating a high-dimensional search space.
- Data Volatility: Stock market data is highly volatile and noisy, requiring robust optimization techniques that can generalize well despite the noise.

6) Evaluation Criteria: The goal in optimizing neural network weights for stock market prediction is to achieve the best possible prediction accuracy, which involves:

- Performance: Evaluating the reduction in prediction error (e.g., MSE) on validation and test datasets.
- Convergence: Assessing how quickly and effectively the algorithm converges to a solution that minimizes the loss function.
- Generalization: Ensuring that the model performs well on unseen data, indicating its ability to generalize from the training data.

E. *Comparison Analysis*

*1) **Purpose: ML Perspective**:* The proposed problems highlight different aspects in the model adaptability to different real world applications and the chaotic environment. Each of these optimization problems shed the light on different areas that are commonly found in the neural network model (for example). The NQP highlights the ability to handle strict constraints and navigate large, intricate search spaces, reflecting the challenge of feature selection in ML models. NQP's combinatorial constraints are analogous to ensuring model parameters satisfy specific conditions, crucial in real-world ML applications. The FFP, a binary optimization task, emphasizes the need to effectively navigate local optima, akin to optimizing neural network weights where the loss surface is filled with many minima. FFP's simplicity allows for a clear evaluation of how algorithms perform in binary spaces, relevant for tasks such as binary feature selection and threshold optimization in ML. The Neural Network weight optimization for stock market prediction (SP-NN) tests an algorithm's capability to handle non-convex loss surfaces and high-dimensional weight spaces, reflecting the complexity and volatility inherent in real-world ML models. SP-NN also assesses the algorithm's ability to generalize from noisy, volatile data, a critical requirement for robust predictive modeling.

*2) **Data Size**:* Table I provides a summary of comparison for the data dimensions used throughout this project.

Problem	**Data Sizes**		
FFP	data_sizes=[40, 100, 500]		
NQP	data_sizes=[20, 50, 100]		
SP-NN-JPM	training_set=2159	testing_set=926	n_features=18

TABLE I: Data Sizes Comparison

*3) **Hypothesizes**:* Table II summarizes the comparison of the hypothesizes across two main aspects, the performance and challenges with respect to the optimization algorithms and the optimization problems used. Notice that the algorithm "MIMIC", is only used with FFP and NQP and was not applied with SP-NN problem. Same goes for the algorithm "GD" (Gradient Descent), since it is only used as a benchmark comparison for the SP-NN weight optimization.

III. Experiments Approach

*1) **Overview**:* This git repository is used throughout the project and has all the experiments compiled. In it, we split the analysis into two parts. Folder "part1_analysis", which has all the code development and the generated curves for FFP and QNP optimization analysis. Folder "part2_analysis" is where the learning extracted from part1 is applied on the SP-NN weight setting application. Next, we will introduce the optimization algorithms while highlighting their strengths, weaknesses, hyperparameters, and their relevance from an ML point of view.

A. *Randomized Hill Climbing (RHC)*

RHC starts with a random solution and iteratively makes small changes, accepting improvements. Random restarts help avoid local optima.

1) Strengths: Simple and easy to implement and efficient for smooth, unimodal landscapes.

2) Weaknesses: Prone to local optima without effective restarts. Less effective in high-dimensional, rugged landscapes.

3) Hyperparameters:

- *Max Attempts:* Max attempts to find an improvement before restarting.
- *Max Iterations:* Max iterations to run before stopping.

B. *Simulated Annealing (SA)*

SA starts with a high "temperature" allowing worse solutions. As temperature decreases, it becomes more conservative, accepting only improvements or small degradations.

1) Strengths: Escapes local optima through probabilistic acceptance. Effective for complex, multimodal landscapes.

2) Weaknesses: Requires careful tuning of the cooling schedule. Slower convergence due to acceptance of worse solutions.

3) Hyperparameters:

- *Initial Temperature:* Starting temperature.
- *Cooling Schedule:* Method to decrease temperature.
- *Max Attempts:* Max attempts at each temperature level.
- *Max Iterations:* Max iterations to run before stopping.

Algorithm	FFP		NQP		SP-NN	
	Performance	**Challenges**	**Performance**	**Challenges**	**Performance**	**Challenges**
RHC	Adequate for small sizes	Local optima	Struggles significantly	Combinatorial constraints	Reasonable for small networks	Local optima
SA	Robust performance	Cooling schedules	Strong performance	Balancing exploration	Robust for complex networks	Slower convergence
GA	Best performance	Computational cost	Performs well	Tuning operators	High-quality solutions	Computational cost
MIMIC	Very effective	Computational cost and model complexity	Excels in dependencies	Handling constraints	Superior configurations	Computational cost and model complexity
GD	NA	NA	NA	NA	Efficient with proper tuning	Learning rate sensitivity

TABLE II: Comparison of Optimization Algorithms Across the Different Problems

C. *Genetic Algorithms (GA)*

GAs maintain a population of solutions, evolving them through crossover and mutation over generations.

1) Strengths: Maintains diversity, exploring multiple solution regions. Effective for high-dimensional, multimodal problems.

2) Weaknesses: Computationally expensive due to population maintenance. Requires careful tuning of genetic operators.

3) Hyperparameters:

- *Population Size:* Number of solutions in the population.
- *Mutation Probability:* Probability of mutations in offspring.
- *Crossover Probability:* Probability of crossover between solutions.
- *Max Generations:* Max generations to run before stopping.

D. *Mutual Information Maximizing Input Clustering (MIMIC)*

MIMIC builds a probabilistic model of the solution space, sampling new solutions and refining the model based on the best solutions found.

1) Strengths: Captures dependencies between variables. Effective for complex, high-dimensional problems. Efficiently navigates large search spaces.

2) Weaknesses: Computationally expensive due to model building and updating. Model complexity increases with variable count.

3) Hyperparameters:

- *Population Size:* Number of solutions sampled each iteration.
- *Keep Percent:* Percentage of top solutions retained.
- *Max Iterations:* Max iterations to run before stopping.

E. *Summary*

Table III summarizes the hyperparameters tunning ranges that are used with respect to each optimization problem and the corresponding optimization algorithm.

Below general definitions are used:

- **Precision** quantifies the number of true positive predictions that belong to positive class. (Equation 1)

$$Precision = \frac{TP}{TP+FP} \tag{1}$$

Algorithm	FFP	QNP	SP-NN
RHC	restart_list=[10,2] max_iterations=[100, 1000]		restart_list=[20, 10, 5, 2]
SA	temperature_list= GeomDecay(init_temp=10.0, decay=0.95, min_temp=1)	temperature_list= GeomDecay(init_temp=10.0, decay=0.95, min_temp=1)	schedule=[GeomDecay(init_temp=10.0, decay=0.95, init_temp=0.1), ArithDecay(init_temp=10.0, decay=0.95, init_temp=0.1),]
GA	mutation_rates=[0.05, 0.1, 0.2, 0.5, 0.8, 1] population_sizes=[50, 100, 200, 500]		mutation_rates=[0.1, 0.3, 0.5] population_sizes=[100, 200, 300]
MIMIC	keep_percent_list=[0.2, 0.1] population_sizes=[100, 200, 500]	NA	NA
GD	NA		hidden_nodes_list = [[5, 5, 5, 5], [10, 10], [18, 10, 10, 10], [18, 100, 100, 100], [100, 100, 100],[20, 20],] activation_functions_list=["relu", "sigmoid", "tanh"]

TABLE III: Hyperparameters Tuning Range Summary

- **Recall** (also known as sensitivity or true positive rate) measures the proportion of actual positives that were predicted correctly. It takes into account false negatives, which are cases that should have been flagged for inclusion but weren't. (Equation 2)

$$Recall = \frac{TP}{TP+FN} \tag{2}$$

- **Accuracy** is a simple metric for evaluating classification models. It calculates the ratio of the number of correct predictions to the total number of predictions:

$$Accuracy = \frac{TP+TN+FP+FN}{TP+TN} \tag{3}$$

- **F1 Score** balances both precision and recall, providing a comprehensive evaluation of a model's performance. (Equation 4)

$$F1 = \frac{2\ .\ Precision \cdot Recall}{Precision\ +\ Recall} \tag{4}$$

- **Learning Curve (LC)** demonstrates if the data used is enough or do we need to add further data to have a more conclusive analysis.
- **Validation Curve (VC)** explains the relationship between the bias/variance and the model performance with respect to the model hyper-parameter value.

Where:

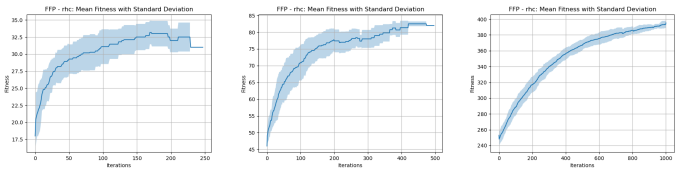
- TP = True Positives, TN = True Negatives
- FP = False Positives, FN = False Negatives

IV. Results and Discussion

Now, we examine the results of each algorithm separately across the different problems.

A. *RHC: Analysis*

Attached in figures 2 and 3 the trends of the fitness vs iterations while varying the restarts list by the values mentioned in the hyperparameters table.

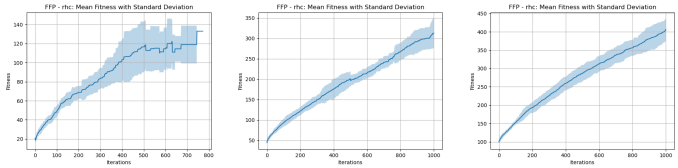


(a) (b) (c)

Fig. 2: RHC/FFP: Fitness vs Iterations across Data Sizes (a) 40 (b) 100 and (c) 500

1) FFP: Fitness vs Iterations: After reviewing the trends in the fitness curves, one can highlight that the performance of RHC algorithm on the FFP for data sizes 40, 100, and 500 shows a consistent pattern. For the smallest data size (40), the algorithm quickly improves fitness, reaching around 27.5 within 50 iterations and stabilizing at 32.5 after 200 iterations, with a relatively narrow standard deviation. For the medium data size (100), fitness increases rapidly to 65 within the first 100 iterations, gradually reaching 82 after 400 iterations, with increased variability. In the largest data size (500), the algorithm steadily improves fitness, reaching about 400 after 1000 iterations, with the widest standard deviation, reflecting greater variability due to the larger search space.

Overall, the RHC algorithm demonstrates effective scalability across different data sizes, quickly finding high-fitness solutions and then making incremental gains. The increasing standard deviation with larger data sizes indicates greater variability in results, suggesting that more complex search spaces pose additional challenges. Nonetheless, the algorithm consistently improves fitness values, highlighting its robustness and adaptability to various problem scales.



(a) (b) (c)

Fig. 3: RHC/QNP: Fitness vs Iterations across Data Sizes (a) 40 (b) 100 and (c) 500

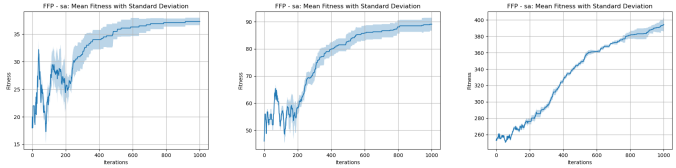
2) QNP: Fitness vs Iterations: For the NQP, the RHC algorithm's performance varies with data size. For a size of 20, fitness increases rapidly to 80 within 200 iterations and stabilizes around 120 after 600 iterations, with narrow initial variability. For size 50, fitness reaches 200 within 400 iterations and stabilizes around 320 after 1000 iterations, with increasing variability over time. For size 100, fitness steadily increases to 400 after 1000 iterations, showing consistent wide variability due to the larger search space. The larger the data size, the more iterations are needed for the algorithm to achieve high fitness values, indicating effective handling of complex search spaces.

3) Comparison:

- Performance Trends: Both problems show rapid initial fitness improvements and gradual stabilization. NQP has a more prolonged improvement period.
- Variability: Increases with data size for both problems. NQP exhibits higher variability, indicating sensitivity to search space exploration.
- Iterations Required: NQP generally requires more iterations to stabilize, suggesting additional challenges due to its constraint satisfaction nature.
-

B. SA: Analysis

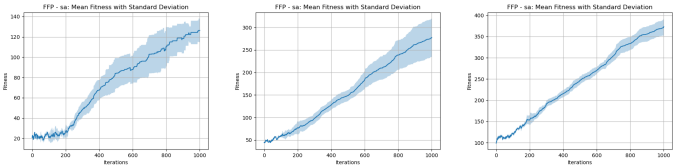
1) FFP: Fitness vs Iterations: For FFP, the SA algorithm's performance varies with data size. For a size of 40, the fitness shows significant initial volatility but stabilizes around 35 after approximately 400 iterations, with the standard deviation narrowing over time. For size 100, the fitness also experiences early volatility, reaching around 90 after 800 iterations, with variability decreasing as the algorithm converges. For size 500, the fitness increases steadily with minimal volatility, achieving about 400 after 1000 iterations, and the standard deviation remains relatively narrow. Overall, SA demonstrates effective exploration and convergence across all data sizes, with early instability giving way to consistent improvements and eventual stabilization in fitness values.



(a) (b) (c)

Fig. 4: SA/FFP: Fitness vs Iterations across Data Sizes (a) 40 (b) 100 and (c) 500

2) QNP: Fitness vs Iterations: For NQP, the SA algorithm's performance varies with data size. For a size of 20, the fitness shows a steady increase, stabilizing around 140 after 1000 iterations, with initial variability that decreases over time. For size 50, the fitness increases consistently, reaching approximately 300 after 1000 iterations, with relatively higher early variability that narrows as iterations progress. For size 100, the fitness steadily increases to about 400 after 1000 iterations, maintaining a consistent and narrower variability throughout. Overall, SA effectively explores and converges across all data sizes, demonstrating its ability to handle the complex constraint satisfaction inherent in NQP, with early variability giving way to steady fitness improvements and stabilization.



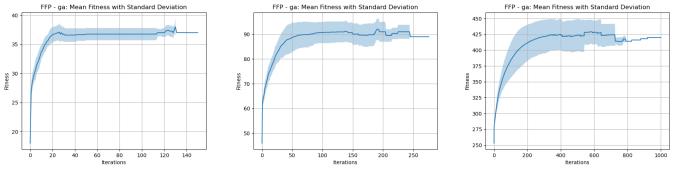
(a) (b) (c)

Fig. 5: SA/QNP: Fitness vs Iterations across Data Sizes (a) 40 (b) 100 and (c) 500

3) Comparison: The performance of the SA algorithm on the FFP and the QNP reflects the distinct nature of these problem domains. In FFP, SA demonstrates initial volatility followed by stabilization at different fitness values depending on data size, effectively navigating the binary search space filled with many local optima. The algorithm's probabilistic acceptance of worse solutions early on helps in avoiding premature traps, leading to gradual improvements and convergence to near-optimal solutions. In QNP, SA shows a steady increase in fitness across all data sizes, with higher final fitness values due to the complexity of placing queens on a chessboard without conflicts. The structured nature of QNP allows SA to leverage its cooling schedule to systematically reduce conflicts and improve placements. The final fitness values in both problems highlight SA's ability to balance exploration and exploitation, adapting its search strategy to the specific challenges and structures of each problem domain.

C. GA: Analysis

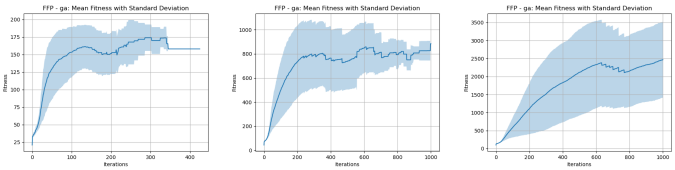
1) FFP: Fitness vs Iterations: For FFP, the GA shows rapid initial improvement in fitness across all data sizes, followed by stabilization. For a data size of 40, fitness quickly rises to around 35 within 20 iterations and stabilizes near 40, with minimal variability. For size 100, fitness increases rapidly to about 90 within 50 iterations and then stabilizes, with slight variability indicating consistent performance. For size 500, fitness climbs to around 425 within 200 iterations, stabilizing with some variability, reflecting the algorithm's effective handling of larger search spaces. The GA's population-based approach allows for diverse exploration early on, with crossover and mutation leading to quick convergence to high-quality solutions. The algorithm's ability to maintain and evolve a diverse set of solutions contributes to its robustness and efficiency in optimizing the FFP across varying data sizes.



(a) (b) (c)

Fig. 6: GA/FFP: Fitness vs Iterations across Data Sizes (a) 40 (b) 100 and (c) 500

2) QNP: Fitness vs Iterations: For the QNP, the GA shows rapid initial improvement in fitness across all data sizes, followed by stabilization. For a data size of 20, fitness quickly rises to around 150 within 100 iterations and stabilizes near 175, with moderate variability. For size 50, fitness increases rapidly to about 800 within 300 iterations and then stabilizes, with noticeable variability indicating consistent performance but with some fluctuations. For size 100, fitness climbs to around 2500 within 600 iterations, stabilizing with significant variability, reflecting the algorithm's effective handling of larger and more complex search spaces. The GA's population-based approach allows for diverse exploration early on, with crossover and mutation leading to quick convergence to high-quality solutions. The algorithm's ability to maintain and evolve a diverse set of solutions contributes to its robustness and efficiency in optimizing the QNP across varying data sizes.

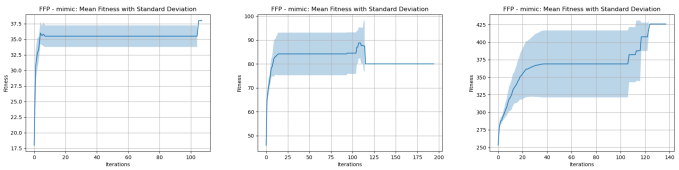


(a) (b) (c)

Fig. 7: GA/QNP: Fitness vs Iterations across Data Sizes (a) 40 (b) 100 and (c) 500

3) Comparison: The GA shows distinct performance on the FFP and the QNP. In FFP, GA achieves rapid fitness improvements with quick stabilization and minimal variability, indicating effective convergence and controlled impact from mutation and population changes. In contrast, QNP exhibits significant variability due to its combinatorial complexity. For smaller sizes, GA improves fitness rapidly but with moderate variability. For larger sizes, substantial fitness improvements come with high variability, reflecting the greater impact of genetic operations. Overall, GA handles both problems robustly, but QNP's complexity leads to more fluctuations, highlighting the sensitivity to mutation and population dynamics in constraint satisfaction tasks compared to binary optimization.

D. MIMIC: Analysis



(a) (b) (c)

Fig. 8: MIMIC/FFP: Fitness vs Iterations across Data Sizes (a) 40 (b) 100 and (c) 500

1) FFP: Fitness vs Iterations:

E. SP-NN: Analysis

In order to analyze the performance across multiple entries, several experiments have been conducted and summarized in the table IV.

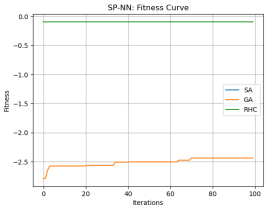


Fig. 9: Fitness vs Iterations across different algorithms

The figure 9 shows the fitness curves increasing in the negative log domain for both the SA and GA alike. It's interesting how they have similar performance with respect to the neural network weight optimization which we are going to discuss later. One important aspect noticed is that these curves might not to be very representable to the problem and the reason could be that we need to expand the search range beyond the setup selected.

1) Key Observations: **Training Performance:** Accuracy, precision, recall, and F1 score on the training set for RHC exhibit consistent values, suggesting a stable training process. Variability in precision and recall metrics indicates differing levels of true positive and false positive rates depending on the model configuration.

Testing Performance: There is a notable drop in performance metrics (accuracy, precision, recall, F1 score) on the testing set compared to the training set, which is common due to overfitting. Some configurations of RHC achieve better testing precision (up to 0.569) but struggle with recall, indicating they might be better at avoiding false positives but miss some true positives.

Algorithm Performance: RHC shows varied results with wall clock times ranging from around 7 to 31 seconds, reflecting the computational expense and the iterative nature of the algorithm. Despite achieving reasonable training metrics, the testing metrics suggest room for improvement in generalization capabilities.

Model Configuration Impact: Different neural network configurations (activation functions, hidden nodes, etc.) lead to varying performance outcomes. For instance, models using sigmoid activation show differing results compared to those using tanh, impacting both training and testing metrics.

2) Impact of Neural Network Hyperparameters:

- Hidden Layers and Nodes: Models with different configurations of hidden layers (e.g., [5, 5, 5]) and activation functions (sigmoid, tanh) show varying performance. Deeper networks (more

	accuracy_train	precision_train	recall_train	f1_train	accuracy_test	precision_test	recall_test	f1_test	wall_clock	pop_size	schedule
count	272.000000	272.000000	272.000000	272.000000	272.000000	272.000000	272.000000	272.000000	233.000000	3.0	15.000000
mean	0.471409	0.342452	0.471409	0.356185	0.293855	0.257507	0.293855	0.210000	316.574685	200.0	8.200000
std	0.126971	0.174554	0.126971	0.159355	0.134729	0.243382	0.134729	0.178247	485.100089	100.0	3.726354
min	0.279579	0.167476	0.279579	0.237683	0.111831	0.012506	0.111831	0.022497	3.580989	100.0	1.000000
25%	0.409238	0.209636	0.409238	0.271743	0.226904	0.051486	0.226904	0.083928	5.355864	150.0	10.000000
50%	0.457861	0.260874	0.457861	0.287594	0.226904	0.096436	0.226904	0.101572	112.952442	200.0	10.000000
75%	0.457861	0.393736	0.457861	0.316435	0.463533	0.483845	0.463533	0.455068	527.536080	250.0	10.000000
max	0.766613	0.764031	0.766613	0.762207	0.685575	0.716241	0.685575	0.653085	4996.483083	300.0	10.000000

TABLE IV: Experiments Summary Results

hidden layers) tend to require more iterations to converge but can capture more complex patterns.

- Activation Functions: Sigmoid and tanh functions impact the model's ability to converge and generalize. Tanh generally offers better performance due to its wider output range.
- Population Size (GA): Larger populations lead to more thorough exploration but increase computational expense and convergence time.
- Schedules (SA): Cooling schedules affect the convergence speed and final solution quality. A slower cooling schedule improves solution quality but increases computation time.

3) Convergence Behavior:

- RHC: Quick convergence with high sensitivity to initial conditions and restarts.
- SA: Slower convergence due to probabilistic acceptance, allowing for better exploration and avoiding local optima.
- GA: Convergence depends on population dynamics; initially fast but slows as the population evolves.
- GD: Fast convergence with appropriate learning rate settings, efficient for large-scale problems

4) Real-World Applicability:

- Best Algorithm: Gradient Descent (GD) generally offers the best performance for real-world applications due to its balance of efficiency and generalization. However, for problems with many local optima or non-smooth landscapes, SA and GA provide robust alternatives.
- Stock Market Data: The randomness inherent in stock market data necessitates algorithms that can handle volatility and avoid overfitting. SA and GA's ability to explore multiple solutions makes them suitable, while GD's efficiency is beneficial for large datasets.

5) Relationship Between Randomization and Results:

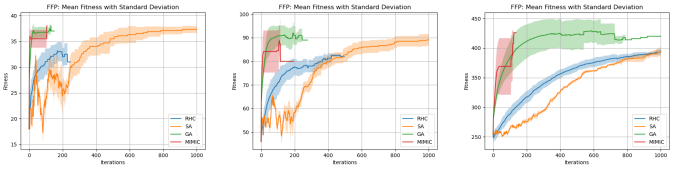
- RHC and SA: Both leverage randomization to explore the solution space, with SA's structured approach (cooling schedule) leading to better generalization.
- GA: Uses random mutations and crossovers to maintain diversity, effectively balancing exploration and exploitation.
- GD: Deterministic, but random initialization can influence convergence.

6) Summary: While there are some results where not reviewed in details, specially considering the hyperparameters, the conclusion found was mostly summarized in the table attached. Aside from that, each algorithm has strengths and weaknesses influenced by the neural network's hyperparameters and the problem's nature. GD is generally preferred for its efficiency, while SA and GA offer robustness for more complex, volatile datasets like stock market predictions. The role of randomization in RHC and GA highlights the need for diverse exploration to achieve high-quality, generalized solutions

V. Conclusion

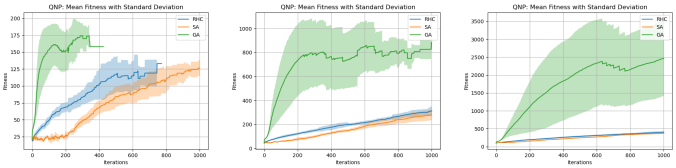
Finally, we discuss the comparison of these algorithms performance compared to each other.

RHC shows rapid initial improvement but struggles with variability and overfitting, leading to a significant drop in testing performance. This is evident in both the Flip Flop Problem (FFP) and the N-Queens Problem (QNP). Although RHC demonstrates quick convergence, its



(a) (b) (c)

Fig. 10: FFP Comparison: Fitness vs Iterations across Data Sizes (a) 40 (b) 100 and (c) 500



(a) (b) (c)

Fig. 11: QNP Comparison: Fitness vs Iterations across Data Sizes (a) 40 (b) 100 and (c) 500

sensitivity to initial conditions and restart strategies results in moderate variability. This makes RHC efficient for small-scale problems but less reliable for complex or large-scale problems due to its tendency to get trapped in local optima.

In contrast, SA exhibits initial volatility but stabilizes over time, effectively escaping local optima and showing consistent performance. This behavior is seen in both FFP and QNP, where SA demonstrates steady fitness improvements and robust performance across all data sizes. The algorithm's probabilistic acceptance of worse solutions early on helps it navigate complex search spaces effectively, making SA suitable for problems with many local optima.

GA shows rapid fitness improvement and quick stabilization, maintaining high accuracy and precision in FFP. In QNP, GA achieves substantial fitness improvements with higher variability due to mutation and population changes, but it ultimately converges to high-quality solutions. GA's population-based approach allows for diverse exploration early on, with crossover and mutation leading to quick convergence. This makes GA robust and effective for diverse problems, balancing exploration and exploitation through its genetic operations. GA is well-suited for high-dimensional and complex optimization tasks.

GD offers efficient convergence with high accuracy and consistent performance metrics, showing the best generalization among all algorithms. Its efficiency in handling large-scale, smooth optimization landscapes makes it the preferred choice for real-world applications involving large datasets and neural network training. GD's deterministic nature, coupled with appropriate learning rate settings, ensures fast convergence and robust performance.

The impact of neural network hyperparameters is significant across all algorithms. Deeper networks with more hidden layers require more iterations to converge but can capture more complex patterns. Activation functions like tanh generally offer better performance

compared to sigmoid due to their wider output range. In GA, larger population sizes enhance exploration but increase computational expense and convergence time. In SA, slower cooling schedules improve solution quality but extend computation time.

For real-world applicability, GD generally offers the best performance due to its efficiency and generalization. However, SA and GA provide robust alternatives for problems with many local optima or non-smooth landscapes. The inherent randomness in stock market data necessitates algorithms that can handle volatility and avoid overfitting. SA and GA's ability to explore multiple solutions makes them suitable, while GD's efficiency benefits large datasets.

The relationship between randomization and results is evident in RHC and SA, where both algorithms leverage randomization to explore the solution space, with SA's structured approach (cooling schedule) leading to better generalization. GA uses random mutations and crossovers to maintain diversity, effectively balancing exploration and exploitation. While GD is deterministic, its random initialization can influence convergence, highlighting the importance of initial conditions.

In summary, each algorithm demonstrates strengths and weaknesses influenced by the neural network's hyperparameters and the problem's nature. GD generally offers the best performance for real-world applications due to its efficiency and generalization. However, SA and GA provide robust alternatives for more complex, volatile datasets like stock market predictions. The role of randomization in RHC and GA emphasizes the need for diverse exploration to achieve high-quality, generalized solutions. This analysis validates the hypothesis that different optimization algorithms have unique advantages and constraints, making them suitable for varying problem domains and complexities.

VI. References

- Pahwa, N., et al. "Stock prediction using machine learning a review paper." International Journal of Computer Applications 163.5 (2017): 36-43.
- Z. K. Lawal, H. Yassin and R. Y. Zakari, "Stock Market Prediction using Supervised Machine Learning Techniques: An Overview," 2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE), Gold Coast, Australia, 2020, pp. 1-6, doi: 10.1109/CSDE50874.2020.9411609.