Enhancing Sudoku Solving with Guided Evolutionary Algorithm (GEA) through Neural Decision-Making

Ted Proctor - 700074708

May 2024

Abstract

This study investigates the integration of Neural Networks (NNs) with Evolutionary Algorithms (EAs) to solve Sudoku puzzles, an NP-complete problem. It explores the application of Recurrent Neural Networks (RNNs) and temporal data to enhance decision-making within EAs. The research compares traditional Sudoku-solving methods, such as brute-force, logical, and stochastic approaches, identifying their respective challenges and opportunities. By proposing a novel hybrid approach that combines the exploratory capabilities of EAs with the predictive strengths of NNs, this study aims to improve both the efficiency and accuracy of solving Sudoku puzzles. A new Guided Evolutionary Algorithm (GEA) framework is introduced, incorporating a NN decision-maker trained via reinforcement learning (RL) to dynamically adjust mutation strategies. Experimental results demonstrate that the GEA improves population diversity and solution quality across various Sudoku difficulties. While the findings are promising, they also highlight areas for further optimisation, such as refining the reward function and enhancing the robustness of the NN. The implications of this research extend beyond Sudoku, suggesting potential applications for solving other NP-complete problems. Future work will focus on optimising the NN parameters and adapting the model for Multi-Objective Evolutionary Algorithms (MOEAs) to optimise multiple conflicting objectives simultaneously.

Declaration

I certify that all material in this dissertation which is not my own work has been identified.

Contents

1 Introduction

Originating in Japan in 1984 and gaining global prominence in 2004 [2], Sudoku is a contemporary logic-based number puzzle that has captivated individuals worldwide. The objective is to fill a $9x9$ grid such that each row, column, and $3x3$ sub-grid (often referred to as "boxes" or "regions") contains all digits from 1 to 9 without repetition, as illustrated in Fig. 1. The puzzle initialises as a partially pre-filled grid, and the player's challenge is to logically deduce the remaining entries while adhering to the puzzle's constraints. Despite the traditional 9×9 grid receiving significant research focus, Sudoku's inherent scalability allows for generalisation to N^2xN^2 grids. In these larger variants, the player must ensure that each row, column, and NxN sub-grid contains all digits from 1 to N^2 without violating the no-repetition rule.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figure 1: Example of a Sudoku grid [1]

Human solvers typically employ an iterative process of logical deductions to tackle Sudoku puzzles. As placements are made, the solver continuously reassesses the grid's state, leveraging constraints imposed by existing entries to guide subsequent moves. This approach blends deductive reasoning with occasional trial-and-error tactics. The inherently analytical nature of these deductive methods renders them adaptable to computational solution strategies.

Sudoku is classified as an $\mathcal{NP}$-complete (Non-deterministic Polynomial time) problem, as proven by Yato *et al* [3]. An $\mathcal{NP}$-complete problem is a type of decision problem that, if solved in polynomial time, implies every problem in the $\mathcal{NP}$ class can also be solved in polynomial time. See figure 2 for a visual representation of their relationships. In computational complexity, a fundamental question is the relationship between $\mathcal{P}$ and $\mathcal{NP}$, where $\mathcal{P}$ is the set of problems solvable in polynomial time, and $\mathcal{NP}$ the set of problems solvable in non-deterministic polynomial time. If $\mathcal{P} = \mathcal{NP}$, then problems verifiable in polynomial time are also solvable in that time frame. If $\mathcal{P} \neq \mathcal{NP}$, some quickly verifiable problems remain computationally intractable [4]. It is currently unknown how $\mathcal{P}$ and $\mathcal{NP}$ are related, although the results of this research may shed some light on the issue. It is important to clarify that NP-complete problems always have a solution, although the solution is usually a binary response to whether an instance of the problem has a solution. The challenge lies in determining a solution for all possible instances efficiently

The study of $\mathcal{NP}$-complete problems holds significance as they serve as benchmarks for assessing the boundaries of efficient computation, providing insights into the limits of computational complexity.

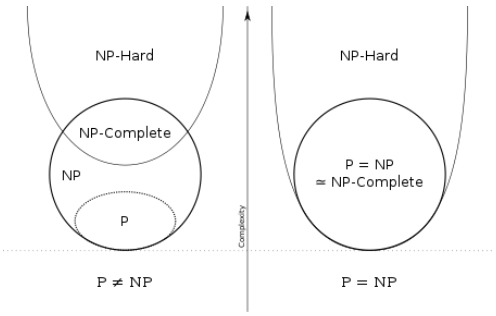


Figure 2: Euler diagram of the relationship between $\mathcal{P}$ and $\mathcal{NP}$ problems [5].

Although solving any $\mathcal{NP}$-complete problem represents a challenge, enhancements to current solution approaches can have far-reaching implications, propagating to improve myriad strategies for tackling the problem. Furthermore, the development of more efficient algorithms in any solution space can establish new benchmarks for computational performance and often result in significant reductions in resource utilisation, a valid consideration in light of the current ecological landscape. Notably, the principles defining an improved algorithm can often find applications in other domains, potentially yielding commercial benefits. The travelling salesman problem, another $\mathcal{NP}$-complete problem, exemplifies this, with advancements in algorithmic approaches producing enhancements in circuit design [6], drone path optimisation [7], and DNA sequencing [8]. While such alternative applications are often discovered, predicting them *a priori* remains challenging, and thus they will not be explored within the scope of this review.

Notwithstanding the potential for advancements in adjacent fields and improvements within the $\mathcal{NP}$-complete problem domain, Sudoku retains its widespread popularity. Developments in this area are likely to be well-received by enthusiasts, particularly if a stochastic approach unveils novel techniques. Moreover, research into solving Sudoku grids frequently yields insights into the generation of such grids, aiding in the delineation of the constraints that determine a grid's solvability.

The crux of the solution proposed in this paper revolves around enhancing the decision-making

process within an Evolutionary Algorithm (EA) framework (see 2.3). Departing from the conventional approach of using logic circuits to perform actions based on predefined events (e.g., n iterations elapsed, solutions converging to local optima, etc.), we introduce a Neural Network (NN) (**??**) trained to perform decision-making by considering the current problem state and leveraging prior knowledge. Given the nature of our data points, specialised NN architectures will be employed, such as Reinforcement Learning (RL) [9] and Recurrent Neural Networks (RNNs) [10]. These methods endow the NN with the capability to process sequential data and incorporate contextual or historical information, facilitating more informed decision-making.

With its focus on decision-making, this approach enables future incorporation of improved heuristics, crossover, and mutation methods. It could produce a model emulating human logical approaches via a decision-maker (DM) model applying logical techniques and evolutionary theory. Integrating these fields may produce robust hybrid models surpassing single-approach capacities, pushing boundaries of current AI and machine learning.

2 Related Literature

This section explores the principal computational theories applied to solving Sudoku, ranging from traditional to more contemporary techniques. There are a few main categories of approach to the Sudoku problem, the most popular of which are:

2.1 Brute-Force Methods

Brute-force techniques, such as the enumeration of all possible number combinations and the more refined backtracking approach [11], systematically explore the solution space to identify valid Sudoku grids. Backtracking, akin to depth-first search, incrementally inserts numbers while complying with Sudoku constraints, retracing steps when encountering contradictions. Another variant, the Dancing Links Algorithm developed by Knuth [12], addresses the exact cover problem pertinent to Sudoku by leveraging depth-first search principles. Although brute-force methods guarantee solutions, they are computationally expensive for more complex puzzles due to their exponential worst-case time complexity. These approaches do not provide insights into the NP-completeness of Sudoku, as they do not offer polynomial-time solutions. Though not the focus of this study, they serve as benchmarks in research.

2.2 Logical Approaches:

Logical strategies in Sudoku solving forego trial-and-error, instead leveraging rule-based techniques that emulate human approaches to problem-solving. These strategies range from straightforward methods like "naked singles" [13], where the value of a cell is unequivocally determined, to more intricate techniques such as "Finned X-Wing" [13]. These methods emphasise logic and constraint propagation [14] over brute-force search. Another interesting adaptation treats Sudoku as a "Boolean satisfiability problem" (SAT problem) [15], exploring whether a set of binary variable assignments can satisfy a given Boolean formula.

While logical approaches are generally more efficient than brute-force methods, they increase in complexity and susceptibility to errors as they advance. These techniques effectively narrow the search space, particularly in simpler puzzles. However, the most intricate Sudoku challenges may still require a blend of trial-and-error and algorithmic interventions. The specificity of these logical methods often limit their transferability to different problem-solving contexts. Nonetheless, when these strategies are integrated into broader heuristic-based Sudoku solvers, they can significantly boost the solver's performance.

2.3 Stochastic Methods:

Stochastic methods, such as Evolutionary Algorithms (EAs) [16][17] and Genetic Algorithms (GAs) [18], employ probabilistic techniques to solve Sudoku puzzles. These methods draw inspiration from Darwin's theory of evolution [19], attempting to mimic natural selection processes such as selection, mutation, and crossover to evolve populations of solutions. This is performed by iteratively refining these populations, and discarding suboptimal solutions in favour of more promising ones across generations. Particle Swarm Optimisation (PSO) [20] represents another stochastic approach, utilising behaviours observed in flocks of birds or schools of fish to navigate the solution space. Although these methods can deliver solutions or reasonable approximations faster than deterministic approaches, especially for complex puzzles, their performance varies significantly and often hinges on the specific configuration of the algorithm [3]. They often suffer from a lack of reliability regarding convergence due to their stochastic nature.

In pursuit of enhanced decision-making during the crossover and mutation phases of these algorithms, we have opted to investigate the following variations of EAs:

- **Genetic Algorithms:** A specific class of EAs, Genetic Algorithms excel in solving optimisation and search problems by continuously evolving a population of candidate solutions. Effective application requires careful problem-specific implementation, such as a suitable puzzle representation and an appropriate fitness function.
- **Evolutionary Strategies (ESs):** Focusing predominantly on mutation, ESs are tailored for optimising continuous, real-valued functions. Although this seems counterintuitive regarding with the discrete nature of Sudoku, inspiration from ESs can be drawn. Elements from ES can enhance single solution optimisation, augment local search capabilities, and assist in training decision-making models on optimal mutation strategies. A feature of ES is the adaptive mutation mechanism, where mutation parameters (e.g., mutation rate) adjust based on algorithmically defined conditions, promoting a dynamic balance between exploration and exploitation [21], another element that can be utilised.

2.4 Quantum approach:

Although a more novel and less studied approach, the advent of quantum computing opens new avenues of research for problems like Sudoku, such as:

- Quantum Superposition [22] and Parallelism [23]: Quantum computers can represent multiple states simultaneously through superposition. This capability allows them to handle a multitude of potential solutions at once, thus theoretically enabling them to solve problems like Sudoku more efficiently by evaluating many possibilities in parallel. This method could perhaps be integrated with stochastic optimisation algorithms to great effect.
- Quantum Algorithms: Algorithms such as Grover's algorithm [24], which provides a quadratic speedup for unstructured search problems, could potentially be adapted to the search space of Sudoku solutions. Although direct applications to Sudoku are not extensively documented in mainstream research, the principles could be relevant.

Due to simplicity and the abundance of prior research on the matter, the GA approach was selected as the most appropriate for our research. This enables us to compare the value of the DM with greater ease as more data is available for comparison, and for building an accurate model. The adaptive mutation technique will be considered as another feature to implement, especially if they can be tuned in accordance with decisions produced by the NN. Additionally, the ES technique of potentially mutating all candidates every generation will be employed, further utilising the potential of the DM. This approach might be less intuitive for discrete problems like Sudoku but could be adapted for solution refinement, once the solution tends to a local optima. GAs are composed of a number of steps: initialisation, fitness evaluation, mutation, crossover, selection, replacement, and termination. The following sections will detail these steps.

2.5 Similar approaches:

One such similar method, Evolutionary Reinforcement Learning (ERL) [25], delineates a similar approach, however, rather than using the RL model as a guide, it integrates the RL as gradient-based approach to help the algorithm converge. The paper states issues with stability and redundancy, along with learning efficiency; implying that their current methods may not be extracting all useful information from the problem.

Another study, detailed by M. Drugan [26], covers numerous intersections of RL techniques and EAs. Although none cover the novel method expressed in this paper, they provide insight into how these fields have been combined before, such as parameter tuning.

2.6 Drawbacks of EA approaches:

Despite the benefits of a stochastic approach, it does come with some drawbacks. These include but are not limited to the following:

Premature convergence [27]: Although not an issue for this specific implementation, this can result in not all solutions being shown. This problem may be exacerbated under MOEA conditions [28], resulting in a poor spread of solutions. This issue can be countered by appropriate selection of termination criteria. The integration of the DM with EAs should hopefully provide a faster convergence, with improved representation of solution pool. This can be achieved by altering the fitness function of the reinforcement element to prioritise exploration, or implementing an exploration-exploitation

trade-off gradient. One research [29] regarding self-adaptive mutations, delineates an approach where "self-adaptation means that the control parameters of the mutation distribution are evolved by the EA, rather than being predetermined by some exogenously given schedule". This is akin to our proposed methodology, despite differences in implementation. The study precludes that their approach leads to premature convergence, reinforcing the merit of our method when considering reducing computational load, but noting potential drawbacks, implying that similar approaches have certain use cases.

GA efficacy: The effectiveness of GAs is highly dependent on the correct parameter tuning, such as population size, mutation rate, and crossover rate [30]. This is highly dependent on the type of problem being tackled, and requires some degree of expert knowledge in order to optimise. This cannot be improved by the proposed methodology.

GA computational cost: The computational cost of GAs can be quite high, especially as the size of the problem increases. This is due to the need for evaluating fitness for a large population of solutions across multiple generations. Additionally, some problems (Aerospace design [31] and Power system design [32]) have long, more exhaustive fitness evaluations, thus necessitating the need for a faster convergence, reducing the total number of evaluations. This issue will theoretically be directly countered by the proposed methodology.

Local optima: GAs can easily become stuck in local optima, particularly in complex landscapes or multimodal problems [33]. This approach should be able to handle this problem, due to the reinforcement techniques used, allowing the model to update to the current conditions of the problem. This should result in the usage of more exploratory mutations (wider-scale application to the problem representation), as opposed to exploitative mutations.

3 Project Outline

3.1 Project Purpose:

This project involves developing a NN-guided GA Sudoku solver, utilising RL to enhance the decision-making in an EA. The focus is on using the NN to analyse the Sudoku grid and past problem states, optimising mutation strategies. This approach aims to tackle complex Sudoku puzzles more efficiently and accurately, showcasing the potential of NN integration with EAs in intricate problem-solving scenarios.

3.2 Aims:

- **Sudoku Puzzle Solving:** Develop an EA capable of solving or improving the fitness of a given Sudoku grid, demonstrating the algorithm's adaptability to various puzzle configurations.
- **Neural Network Decision Maker Implementation:** Implement a NN that functions as a decision-making tool within the EA, guiding the selection of mutations.
- **Performance Evaluation of NN-Enhanced EA:** Compare the performance of the NN-enhanced EA against a traditional EA, assessing improvements in solution quality and computational efficiency, as well as real time performance assessments.

3.3 Scope and Constraints:

Scope: The primary goal is to concentrate on the enhancement of GAs by implementing a NN DM. We will not be exploring the integration of this technique with other EAs, aiming instead to optimise capabilities of this GA within the context of Sudoku.

Constraints: Several constraints influence the potential success of this project. Sudoku puzzles typically have one or a limited set of correct solutions, which may limit the applicability of this approach to broader EA techniques such as MOEAs. Furthermore, there is a preference for initialising legal Sudoku grids along one axis, which simplifies the starting conditions but does not represent all possible scenarios. Although guided initialisation [34] is a technique often utilised, this methodology should be tested on problems which prefer random initialisation. Additionally, extensive training of the NN component is limited by the availability of high-performance computing facilities. This restriction impacts the scalability of training and refinement processes, thus preventing the upper limits of this approach being reached.

These carefully defined scope and constraints help ensure that the project remains focused and acknowledges practical limitations that might affect its progress and the feasibility of achieving its goals.

3.4 Assumptions:

For the project's success, certain conditions are assumed true. It is presumed that all Sudoku puzzles used for testing adhere to Sudoku's constraints. This is crucial as it ensures that the algorithms are tested against valid problems, reflecting their true efficacy. Another assumption is that the computational resources, such as processor speed and memory availability, are sufficient to perform all necessary calculations without performance bottlenecks, supporting the feasibility of running computationally intensive EAs within reasonable time frames. Additionally, it is assumed that the EA's performance is consistent across multiple runs with the same parameters, implying that the stochastic nature of these algorithms will tend towards the mean performance, thus ensuring reliability and reproducibility of results. There is an assumption that the results obtained from the selected sample of Sudoku puzzles can be generalised to other puzzles of similar types and difficulties, suggesting that the puzzles used in experiments represent a broad range of scenarios and that the algorithm's effectiveness is not limited to a specific subset. Finally, the framework's generalisability is assumed, suggesting that the techniques developed and tested on this problem will exhibit similar effectiveness when applied to different types of puzzles or optimisation instances. This forms the cornerstone of this research, and would indicate that the underlying principles and techniques are robust and versatile across various problem domains.

3.5 Functional requirements:

- **Sudoku Grid Interpretation:** The system must be capable of interpreting a given Sudoku grid, accurately processing and representing it for algorithmic analysis. This should be scalable.
- **Evolutionary algorithm - Genetic Algorithm:**
  - **Population Management:** Manage populations of candidate solutions efficiently.
  - **Crossover Mechanism and Mutation Mechanism:** Implement effective crossover and diverse mutation strategies (as specified in Table 1), enhancing exploration and preventing premature convergence.
  - **Selection and Replacement:** Integrate selection and replacement processes to maintain diversity in the population and ensure the best solutions are carried forward.
- **Neural Network Decision Maker:**
  - **DM components:** Combine a RL component with a Convolutional Neural Network (CNN) component in a fully connected neural layer. The RL should capture temporal patterns to promote diversity when the population stagnates, whilst the CNN will highlight important spacial features in the Sudoku puzzle.
  - **DM integration:** Integrate the NN DM within the EA to guide mutation decisions.
- **Performance Metrics:** Implement a system to measure key performance indicators such as solution quality and convergence rate.
- **Testing Framework:** Establish a framework to evaluate the system across Sudoku puzzles of varying difficulties, ensuring robustness and adaptability.
- **Hyperparameter Tuning:** Enable hyperparameter tuning for the NN to optimise its performance, including but not limited to learning rate, and network architecture.
- **Data Generation:** Procure a comprehensive data-set of Sudoku puzzles and solutions for training the NN, ensuring a wide range of difficulties and configurations.

3.6 Non-Functional Requirements:

- **Performance Requirements**:
  - Solving Speed: The objective is to improve the solving speed of Sudoku puzzles.
  - Scalability: The system should be capable of handling varying sizes and complexities of Sudoku puzzles, including the DM.
  - Training Time: Consideration must be given to the cost of training the NN, striking a balance between model accuracy and training time.
- **Reliability and Availability**:
  - Robustness: The system should handle erroneous or incomplete input, providing useful feedback or corrective measures.

  - Portability: The work should be replicable on any platform, provided the theories are clear.
  - Maintainability: Code should be documented for modifications and maintenance.
  - Compliance: The project should comply with software development standards and practices.

- **Usage Guidelines**: Derive usage examples for the proposed methodology, providing instances where it can be applied instead of standard methodologies.

3.7 Ethics:

The development and application of algorithms require commitment to transparency and integrity. This includes full disclosure about how algorithms are created, tested, and their results reported, ensuring all methodologies, data handling, and findings are presented accurately and without bias. Since Sudoku puzzles involve non-sensitive data, maintaining data privacy and security standards is not of utmost priority. The computational resources required for running EAs highlight the need for optimising algorithm efficiency and leveraging sustainable technologies to minimise environmental impacts. It is also vital to consider the potential misuse of similar algorithms in different contexts, which could lead to unintended consequences such as privacy violations or optimisation manipulations. Proactive measures, such as setting usage limitations and guidelines for ethical use, should be incorporated to address these risks.

3.8 Evaluation and Success Criteria:

We can evaluate success by considering our aims (3.2). We can assess the success of our approach through analytical and numerical comparisons to known methods and algorithms, as well as similar approaches, such as an EA without the NN decision-making component.

Success criteria:

- Sudoku Puzzle Solving: Test the EA model on some sample grids, assess the rate at which the puzzles are solved within a reasonable number of generations (e.g. 500 generations). This can be calculated with $\frac{\text{solutions found}}{\text{total puzzles}}$.
- Neural Network Decision Maker Implementation: We can validate the success of this element by ensuring that the model does not consistently prioritise a single mutation, under variable conditions. This can be performed by forcing conditions into the NN inputs, and testing to see if expected behaviour occurs (e.g. if the population has stagnated and no change in best solution has been found for a large number of iteration, then mutations which prioritise diversity should be chosen).
- Testing Across Different Difficulties: By manually checking the difficulty of test grids across some sample of humans, we can deduce the difficulty of a puzzle. Once this has been verified, we can evaluate the success of the EA on different difficulties, using a similar approach as detailed in the first bullet point.
- We can compare performance of our implementation to traditional EAs, and to established algorithms, using the following metrics:
  - Time Complexity: We can calculate the theoretical time complexity of this model, and compare it to other methods. A pertinent consideration is the time complexity associated with training the NN model.
  - Space Complexity: We can calculate the space complexity of the model, and compare it to other methods. Its important to consider the space complexity of the training set used for the NN, as well as the space complexity of the algorithm once the training process has been completed.
  - Actual Time: We can measure the exact amount of time a run of the program will take on a puzzle. We can average these values and compare to the average time performance of other algorithms, due to the stochastic nature of the approach. In order to ensure a fair test, these tests will have to run on the same machine under the same conditions, at least $\tilde{5}0$ times.
- Improve Solving Speed: The effectiveness of the method in this report compared to previous methods is gauged by its ability to consistently find valid solutions in fewer iterations or less time. The presence of stochastic elements necessitates a significant performance difference to attribute

improvements to the method rather than randomness. Averaging results over numerous runs will minimise error and help establish the mean number of iterations for solution discovery. An improvement margin of approximately 5% is deemed sufficient to indicate enhancement over prior methods, since this should overcome discrepancy due to the stochastic approach. Additionally, variance measurement will assess solution convergence consistency. Statistical analysis, such as T-tests, will be employed to evaluate performance differences.

4 Methodology

4.1 High-level Project Overview:

This project seeks to enhance the capabilities of EAs by integrating a NN based DM, specifically tailored for the task of solving Sudoku puzzles. The primary innovation lies in the application of RL to guide mutation selection processes within the EA framework, resulting in the formulation of a novel approach: the Guided Evolutionary Algorithm (GEA). By embedding a NN that operates on RL principles, the system is designed to dynamically adjust mutation strategies based on real-time feedback, optimising the evolutionary process to converge more quickly on valid Sudoku solutions.

The objective is to develop an EA that performs standard evolutionary operations such as selection, crossover, and mutation alongside incorporating advanced decision-making capabilities through a NN. This NN (trained via RL) will evaluate the current state of the problem and make informed decisions on mutation selection, attempting to accelerate convergence. Resultantly, the system aims to outperform traditional EAs in terms of speed (function evaluations), providing a robust model that can handle Sudoku puzzles of varying difficulty levels. This integration aims to advance AI problem-solving capabilities, demonstrating a novel methodology potentially extensible to other NP-complete problems and related domains. Fig. 3 elucidates the flow of the proposed method.

The implementation was decided to be performed in the Python [35] environment, due to the availability of extensive library support and plotting tools. The below figure provides visualisation of both a typical EA process and the GEA process.

4.2 Evolutionary Algorithm Implementation:

4.2.1 Problem Representation (EA)

Each Sudoku puzzle is represented as a NxN grid, where each cell contains a number from $1 - N$ or is left blank if no number has been assigned. In the context of the EA, these grids are encoded as two-dimensional arrays. Each sublist represents a row in the Sudoku grid, and each element within the sublist represents a cell in that row.

When applying genetic operations, the solution to a Sudoku puzzle (i.e. a filled grid) is encoded as an N^2-element list (flattened from the NxN grid). Each element corresponds to a cell in the Sudoku grid, reading from left to right and top to bottom. This linear representation simplifies the application of genetic operators, avoiding issues that may arise from having to handle list dimensionality. Additionally, the immutable cells (values native to the problem) are stored in a reference grid. This is used to prevent genetic operations altering the puzzle's native constraints.

4.2.2 Fitness Evaluation

As delineated by Jilg *et al* [36], the fitness of a Sudoku grid G, where $G_{i,j}$ signifies the value at the i-th row and j-th column, is the aggregate of repeated numbers in all rows, columns, and 3×3 sub-grids, defined as:

$$f(G) = \sum_{i=1}^{9} \text{Repeats}(G_{i,*}) + \sum_{j=1}^{9} \text{Repeats}(G_{*,j}) + \sum_{k=1}^{9} \text{Repeats}(\text{Sub-grid}_k) \tag{1}$$

The minimisation of this fitness function thereby corresponds to a reduction in rule violations within the grid, influencing the algorithm towards "correct" solutions. Additionally, as this fitness function measure "correctness", it facilitates uninhibited exploration of the solution space without being confined to incremental or local optimisations, reducing the tendency to converge to fitness-defined local optima.

4.2.3 Initialisation

In a partially filled Sudoku grid with provided immutable cells, solutions are generated by treating empty cells as variables to evaluate solution quality. Values assigned to these cells are optimised to

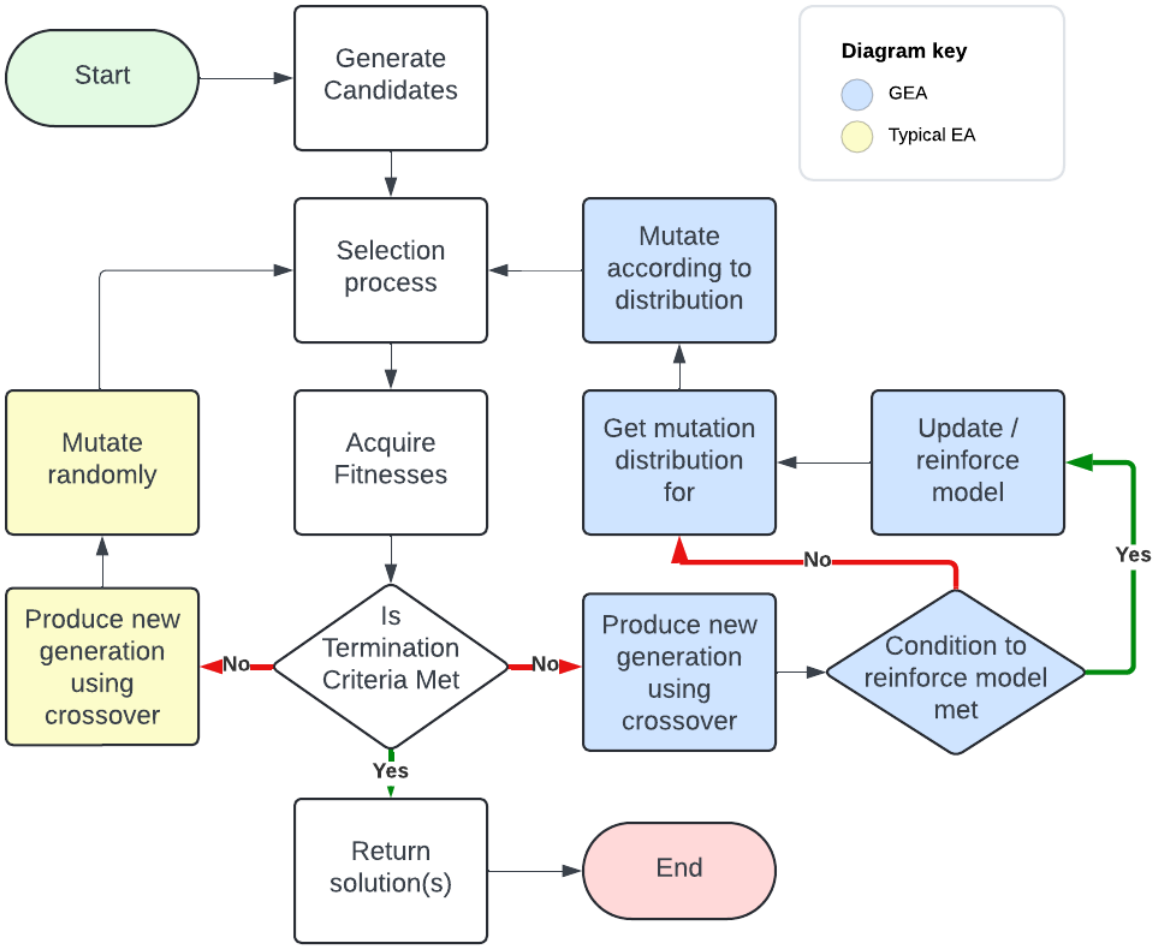


Figure 3: Algorithm flow - DM-Enhanced GEA (blue) and Typical EA (yellow) execution

meet the problem's constraints, expressed through the fitness evaluation, ensuring that a valid solution adheres to Sudoku rules throughout the grid, both in the provided and variable cells.

The initial population will be generated randomly. Two options are presented:

- **True Random Initialisation:** In this method, each empty cell c_{ij} in the Sudoku grid is filled with a random number from the set $S = \{1, 2, \ldots, N\}$, where N is the dimension of the grid. The formulation is given by:

$$c_{ij} = rand(S) \quad \forall i, j \in \{1, 2, \ldots, N\} \tag{2}$$

  where $rand(S)$ denotes a function that returns a random number from the set S.

- **Guided Random Initialisation:** In this approach, each empty cell in a row or column is filled such that no repeats occur in that row or column. This can be expressed as:

$$c_{ij} = \begin{cases} rand(S \setminus R_i), & \text{for row-wise filling} \\ rand(S \setminus C_j), & \text{for column-wise filling} \end{cases} \quad \forall i, j \in \{1, 2, \ldots, N\} \tag{3}$$

  Here, $rand(S \setminus (R_i \text{ or } C_j))$ is a function that returns a random number from the set S excluding the numbers already present in the i-th row (R_i) or j-th column (C_j).

The True Random initialisation (Eq. 2) promotes diversity via an unbiased initial population, facilitating early-stage exploration of the solution space. This provides ample mutation possibilities, further diversifying the genetic pool. However, this approach disregards Sudoku's no-repeat constraints for rows, columns, and 3x3 sub-grids, yielding more solutions with poor fitness and necessitating increased computational effort for error correction and constraint handling, compromising efficiency.

In contrast, the Guided Random method (Eq. 3) generates initially viable solutions by adhering to the no-repeat constraints from inception, precluding repetitions in rows or columns. Nonetheless,

this strategy restricts initial population diversity and limits mutation options, as all mutations must satisfy Sudoku's intrinsic constraints along the guided axis, or risk futility.

A comparative analysis of the two initialisation methods described will be undertaken to determine their impact on the GEA's efficacy. This comparison will focus on measuring the impact of each initialisation method on the convergence speed, and computational efficiency of the algorithm. By evaluating the performance of these methods in diverse puzzle configurations, the most effective strategy for initialising Sudoku grids within our evolutionary framework can be identified. Hopefully, this should help define guidelines of usage for the GEA approach when tackling different problems.

4.2.4 Termination

The termination of the algorithm is contingent upon two criteria: the **identification of a viable solution**, or a **predefined timeout** (based on iterations or real time) in the case that no solution is found in reasonable time. While certain Sudoku grids may possess multiple solutions, our approach necessitates identifying only one viable solution. Investigating all potential solutions for a given grid, involving extensive comparisons to brute-force methods, falls outside the current research scope and is recommended for future research.

4.2.5 Crossover

Crossover, an essential genetic operator in GAs, merges genetic data from two parents to produce offspring, mimicking biological reproduction. This mechanism can unveil more solutions within the algorithm's search space, with the potential for fitter solutions being found. This section particularly focuses on geometric crossover, which merges parent solutions geometrically, typically by a linear combination of vector components, ensuring that offspring remain within viable solution boundaries [37].

Cyclic crossover maintains gene positions by swapping genes cyclically between parents [38]. Partially Mapped Crossover (PMX) exchanges segments between parents, preserving uniqueness and order [39], and is generally applied to individual columns or rows.

Cyclic crossover was selected as more appropriate for this project due to preservation of permutations. This ensures that all genetic material from the parents is transferred without repetition or omission, resulting in improved structural integrity. This can be particularly advantageous in maintaining the logical structure necessary for solving more complex Sudoku puzzles in reasonable time. When the proposed methodology is applied to other problems, the optimal crossover technique will likely require expert guidance.

4.2.6 Selection

Selection in EAs involves choosing individuals from a population for crossover, and several methods are explored to facilitate this process. Roulette wheel selection [40] selects individuals based on a probability proportional to their fitness relative to the total population fitness, providing higher fitness individuals a higher chance of being chosen. Another common method is tournament selection [40], where a subset of individuals is randomly selected, and the fittest among them is chosen. Additionally, rank-based selection is used, where individuals are ranked according to their fitness, and selection is based on this ranking. This method helps to reduce the bias towards the fittest individuals, promoting a more diverse genetic pool.

By conducting tests on various selection methods, we can identify which process results in the best performance. The different selection procedures should promote variable levels of diversity, presumably impacting the performance of the GEA. A diversity measure will be used to compare the selection methods.

4.2.7 Replacement

Replacement in genetic algorithms is essential for balancing the exploration of new solutions with the refinement of existing ones. It plays a critical role in preserving genetic diversity and preventing premature convergence on local optima. A prevalent strategy in this context is elitism [41], which involves retaining the best solutions for the next generation whilst introducing new candidates. This ensures the preservation of superior traits but may limit genetic diversity and pose a risk of premature convergence [42]. This technique is employed because it often improves algorithm performance, supporting the crux of the experiment. It will be used by comparative algorithms too, so that we can ensure that any performance enhancements are not attributed solely to it. Preservation of the least fit candidates is also incorporated. This strategy enhances the exploration of the search space once a local optimum is reached, facilitating the escape from such optima.

4.2.8 Mutation

Mutation serves as a key mechanism in EAs by introducing random variations to the genetic components of individuals within a population. This process not only enhances genetic diversity but also facilitates exploration of the solution space, thereby helping to prevent premature convergence on sub-optimal solutions. This premise posits the focal point of the research, since the GEAs objective is to optimise this step, using the DM to select the most appropriate mutation at the current instance.

Various mutation methods, as detailed in Table 1, are designed with varying balances of impact (the proportion of alleles altered) and tendency (if the mutation promotes diversity or convergence). The DM process selects the most appropriate mutation given the current state of the problem. Whilst the implementation of these mutations may vary, the underlying principle of having a spread of exploratory and exploitative mutations remains constant. We also considered if a mutation was naturally constrained, specifically if it preserves the constraints of the Sudoku puzzle on whatever axis the mutation occurred upon. Naturally constrained mutations will often help convergence, whereas non-constrained mutations should promote diversity. The following selection was considered an adequate spread to fulfil the above conditions:

Table 1: Mutation operations and their characteristics

Mutation	**Description**	**Limitations**	**Naturally Constrained?**	**Potential Impact**	**Tendency**
Swap	Exchange contents of two random variable cells, confined to row, column or sub-grid, extended from Jilg *et al* [36].	Very slow on complex grids.	Yes	Low	Diversity
Flip	Flips the order of variable cells in a row, column, or sub-grid.	Counter-intuitive when problem approaches completion.	Yes	Medium	Diversity
Regenerate X	Regenerates X (where $1 \leq X \leq N$) variable cells, where $C_{ij} \in S$.	Can break constraints.	No	High	Diversity
Smart regen X	Regenerates X variable cells, where $C_{ij} \in (S \cup X_{removed})$, where $X_{removed}$ are the values being replaced.	Can become easily trapped in local optima.	Yes	High	Both, primarily diversity
Filtered Mutation [43]	Mutations are confined to sub-grids. The mutation, a swap, occurs only if it does not degrade the solution quality for the second point. This process repeats until a feasible swap is found, ensuring single cell quality improvement.	Increased computational complexity.	Yes	Low	Convergence
Normalise	If row/column has repeated values, switch one at random to a value that does not appear in the list.	requires grid to break constraints.	Yes	Low	Convergence

This set of mutations was deemed reasonable for the problem at hand due to the spread of impact and tendency observed throughout. No mutations with high potential impact and a tendency to converge were implemented, as in this case they would require considerable problem specific knowledge or logic, a characteristic that would reduce the integrity of the GEAs experimental build, since any improvement in performance could instead be attributed to said mutations. Some mutations with problem-specific logic were included, but these were kept simple and intuitive. The majority of mutations included are generic operations that could be applied or adapted for most genetic representations of solutions. This should improve the reproducibility of the GEA across other problems, and ensure that performance increases are attributed to the framework, as opposed to the mutations themselves.

4.3 Decision Maker Implementation:

The DM's primary objective is to enhance the efficiency and effectiveness of the system by making informed choices. It is a component designed to facilitate intelligent decision-making within the system. Its function is to select the best course of action based on current conditions and learned strategies, navigating the system towards optimal outcomes. A NN was decided as the optimal approach for such a component.

4.3.1 Problem Representation (NN)

Effectively representing Sudoku puzzles is critical for the successful application of NN-based solutions. Sudoku grids are encoded as NxN matrices where each cell contains a digit from $1-N$. This numerical representation is directly amenable to processing via CNNs 4.3.2, which can recognise patterns and features relevant to the game's rules.

Each Sudoku instance is also associated with a set of scalar features that provide additional context for the NN. These features include the current fitness of the grid, amongst others. These Scalar inputs are crucial for providing metadata about the Sudoku instance, informing the DM on the progress of the algorithm.

The NN's architecture processes these two inputs independently yet concurrently (as seen in Figure 4). The CNN branch processes the grid matrices to capture spatial dependencies and patterns [44], essential for recognising the underlying issues with the current solution. In parallel, a dense network branch [45] [46] processes the scalar features, allowing the DM to integrate temporal and performance-related information into the decision-making process. These factors are concatenated, allowing the model to gain a comprehensive understanding of both the current state of the puzzle and its evolutionary progress. This integrated approach allows the model to predict the most beneficial mutations at any given point.

4.3.2 Neural Network Architecture:

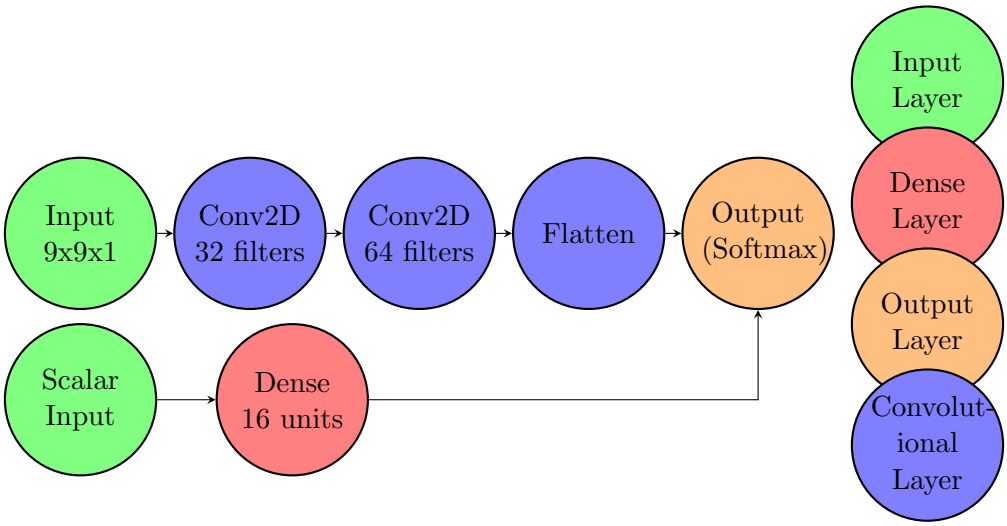


Figure 4: Neural Network Architecture

Convolutional Neural Network: The CNN elements form the core of the model designed for processing Sudoku grids. Used for extracting spatial and structural patterns from the grid, these elements are essential for solution optimisation. The CNN architecture employed in this model consists of several layers, each designed to capture varying levels of abstraction from the input grid:

- **Input Layer:** The input layer receives the NxN Sudoku grid, reshaped into a $NxNx1$ tensor to accommodate the format required by CNN layers.
- **First Convolutional Layer:** Applies 32 filters of size 3x3 to detect basic patterns, like lines and shapes, indicating numerical alignments and rule compliance in the Sudoku grid.
- **Second Convolutional Layer:** Uses 64 filters of size 3x3 to deepen analysis and capture complex interactions within the grid, enhancing the detection of intricate structures.
- **Flattening Layer:** Converts the output from the second convolutional layer into a one-dimensional vector, preparing it for integration with scalar features and further processing in

the dense layers.

Function and Impact These CNN layers transform raw grid data into a format that represents individual cells but also their relational dynamics within the grid. This enhances pattern recognition, allowing the model to effectively interpret common and rare grid patterns through an increase in the complexity of feature detection. Additionally, the CNN layers condense the spatial information into a compact form, maintaining essential features while reducing data dimensionality, a feature that is crucial for efficient processing in later stages. The output from the CNN is optimised for integration with scalar features, ensuring a worthy contribution to the final decision-making process in the model.

Scalar Features Scalar features in the NN are utilised to incorporate context that complements the spatial pattern recognition provided by the CNN. These features, which are derived from both the state of the Sudoku puzzle and the metrics of the EA, are used to enhance the model's predictive accuracy. The model leverages the following scalar features:

- **Grid Fitness:** This feature measures the quality of a Sudoku grid based on the number of constraint violations present. It measures how close the current solution is to a valid solution, indicating which moves lead to a valid solution.
- **Current Generation:** The generation count of the evolutionary process is used to inform the DM of the progression in the solution process. This temporal context helps in adapting the mutation strategies based on how long the algorithm has been running.
- **Generations Since Last Best Fitness:** This tracks the number of generations since a new best solution (lower fitness) was found. It is useful for detecting stagnation in the EA, intended to prompt the network to increase exploration to escape local optima.
- **Previous Mutation:** Despite not being directly conducive to the optimisation process, this input allows for an additional degree of context for the DM. This insight can inform the decision-making process, enabling the DM to adjust strategies based on previous outcomes and potentially anticipate the effectiveness of certain mutations, or mutation sequences, in the current context.

Integration These scalar inputs are later processed through a final layer, integrating them with the features extracted by the CNN layers. The integration is performed before the final decision-making layer of the network, ensuring that both spatial and contextual insights are considered in the mutation decision process. This holistic approach refines the model's responsiveness to immediate inputs but also strategically aligns it with the overall solution trajectory, enhancing the effectiveness of the GEA.

4.3.3 Reinforcement Learning Element

RL is incorporated into our NN to refine the decision-making process. This process is employed to ensure dynamic puzzle responses, and to optimise the selection of mutations during different evolutionary steps of the algorithm.

The RL component interacts with the NN by evaluating the mutation outcomes once applied to a given grid. Each mutation is treated as an action undertaken by the RL agent, with the state being the current configuration of the Sudoku puzzle. Actions are evaluated based on their success in progressing towards a solution, and this feedback is used to train the RL model, enhancing its ability to predict beneficial mutations in the future. Choices that enhance the diversity of the population will be rewarded, especially when the fittest candidate has stagnated. See more in 4.3.4

Integrating RL leverages its strength in learning optimal actions through trial and error. This is particularly valuable in complex puzzle environments like Sudoku, where the efficacy of specific mutations can vary depending on the current state of the puzzle. RL helps the model adapt to these dynamics, promoting a more intelligent exploration of the solution space and avoiding less productive or idempotent paths.

Model Outputs The DM outputs a set of probabilities, each corresponding to a potential mutation type within the evolutionary process. These probabilities are generated by the final output layer of the model, which utilises a softmax activation function [47]. This output layer evaluates the combined features processed from both the CNN, considering both spacial dependencies and metadata. Each probability in the output vector represents the likelihood that a specific mutation will lead to an optimal or improved grid configuration, guiding the EA in selecting the most effective mutations to apply.

4.3.4 Reward Function

The efficacy of the reinforcement learning (RL) algorithm is contingent upon the reward function, which quantifies the success of mutations. This function assesses each mutation, using the improvement in fitness of candidate and by the enrichment it provides to the population diversity, thereby counteracting early convergence.

Reward Mechanism The reward function is designed to integrate immediate and long-term evolutionary success metrics, guiding the RL agent towards a balanced exploration and exploitation strategy. It is formally expressed as:

$$\text{Reward} = (\Delta F \cdot I) + \lambda \cdot \Delta D \cdot (I - 1) \tag{4}$$

where:

- $\Delta F = \textbf{current\_fitness} - \textbf{previous\_fitness} - 0.5$: Represents the change in fitness due to the mutation, adjusted for gradual performance degradation. A negative value (improvement in fitness) results in a positive reward, and vice versa. The constant 0.5 penalises stagnation, ensuring that only mutations that yield improvements are rewarded.
- $\Delta D = \textbf{current\_diversity} - \textbf{previous\_diversity}$: Measures the change in genetic diversity from one generation to the next, calculated using the entropy [48] of the population's fitness distribution. This term rewards increases in diversity, fostering exploration of the genetic landscape.
- λ: A scaling factor that adjusts the influence of the diversity change on the reward, facilitating fine-tuning of the exploration-exploitation balance.
- $I = \textbf{iterations\_since\_best\_fitness}$: Denotes the number of generations elapsed since the last peak fitness achievement. This increases the weight of the diversity term when the algorithm has not found a better solution recently, promoting diversity to break out of stagnation.

This formulation of the reward function dynamically balances the immediate benefits of fitness improvements against the strategic advantages of maintaining genetic diversity within the population. It incentives direct enhancements to the current solutions alongside encouraging exploration necessary to discover superior solutions.

Integration with the Model Rewards calculated from the fitness evaluation inform the model's learning, altering its policies to optimise decision-making. This integration is essential for the convergence of the reinforcement learning process, aligning short-term actions with the long-term goal of consistently reducing rule violations and progressing toward the solution.

4.4 Dependencies:

Python [35] was chosen for implementation due to its readability and extensive library support. Libraries such as NumPy [49] and Pandas [50] are utilised for mathematical operations and data handling. For the neural network implementation, Tensorflow [51] is employed, with Scikit-learn [52] used for pre-processing and statistical analysis. In-built libraries are regularly used, and can be surveyed in the codebase.

4.5 Training Procedure:

The training procedure is designed to refine the decision-making capabilities of our model dynamically. Here's an overview of the key steps involved in training the model to solve Sudoku puzzles:

Data Preparation Data is extracted from the evolutionary process on a generational basis, where each puzzle instance includes the grid and associated scalar data. These grids are reshaped to 9x9x1 tensors, and the scalar inputs are formatted to accompany the grid data.

Training Loop The model's training procedure involves updating parameters during runtime. After every decision is made, the reward for the decision is stored, and at the end of each episode (generation) the model is updated. For each episode:

- The model predicts mutation actions based on the current grid state and its scalar features.
- Actions are executed, and the environment updates the puzzle's state and returns a reward based on the effectiveness of the mutation.

- A loss is calculated using the negative log probability of the selected action, scaled by the received reward, encouraging the model to prefer actions that lead to positive outcomes.
- The model's weights are updated via backpropagation [53] to minimise this loss, refining the model's strategy over successive training iterations.

The effectiveness of the training is monitored by evaluating the cumulative rewards and the model's ability to improve its predictions over time, aiming to enhance its guidance for the GEA in achieving valid solutions more quickly.

Choice of Optimiser The Adam optimiser [54] was selected for training the model due to its efficiency and effectiveness in handling sparse gradients and adapting the learning rate during training. Known for combining the advantages of AdaGrad [55] and RMSProp [56], Adam facilitates faster convergence and is well-suited for complex problems like Sudoku, where the model must adjust quickly to dynamic changes in puzzle configurations.

Load Balancing Updating the model too frequently (e.g., after each evaluation) can be computationally expensive and inefficient due to the high overhead and potential for rapid, unnecessary changes in the model's parameters. Conversely, updating too infrequently might delay the integration of valuable insights gained from new data.

Updating the model at the end of each generation is an effective compromise that leverages batch learning advantages while keeping the model responsive to new data. Maintaining this balance improves the stability of the model, as well as improving computational efficiency, since fewer changes are applied to the model, thus reducing calculations. Additionally, updating the model on a generational basis ensures that the stochastic nature of an EA will be less of a factor when optimising the model.

4.6 Optimisation and Efficiency:

The project's performance could be further optimised through parallelisation and the strategic use of computational frameworks tailored to specific hardware. Parallel processing could significantly expedite the EA and NN operations, allowing for faster evaluations and training. Additionally, aligning computational framework that matches the hardware, such as PyTorch [57] for AMD cards and TensorFlow [51] for NVIDIA GPUs, will further enhance processing efficiency due to inbuilt functionality. These optimisations were not implemented in the current project scope but could be considered for future works.

4.7 Development Process

An Agile, sprint-based methodology was employed, with weekly/bi-weekly iterations punctuated by reviews. Three phases structured the process: development, testing/validation, and documentation. As a solo effort, flexibility was essential to address NN implementation complexity, amongst other obstacles. Despite lacking formal integration pipelines, the iterative approach enabled frequent refinement of the algorithm. Each phase built upon prior insights to meet objectives and handle emergent challenges. Documentation during the reporting phase synthesised findings and methods for academic evaluation.

4.8 Deliverables

The following are expected deliverables from the course of this project. **Research Report:** A detailed account of the study, methodologies, and outcomes. **Codebase:** The fully annotated source code, available for academic use and further development. **Documentation:** Includes technical specifications and operational guidelines to support understanding and application of the research findings. These deliverables are designed to ensure transparency, reproducibility, and utility in the academic community.

5 Results

This section presents the findings of the GEA applied to varying complexities of Sudoku puzzles. By evaluating the performance across different problem difficulties — ranging from easy to very hard — the efficacy of various configurations of crossover and selection methods within the GEA framework are explored, allowing for the production of more in-depth usage guidelines. The analysis uses quantitative assessments of fitness, convergence rates, and diversity metrics to measure quality. Additionally, we

delve into hyperparameter tuning, the cost-effectiveness of our approach, and its robustness against atypical cases.

The quality of our solutions are measured using their fitness, a direct measure of how many Sudoku constraint violations they contain. This allows direct comparisons of quality for a run of the GEA. To measure diversity, we use the entropy of a population, which measures the amount of randomness or disorder in a system, and is often used to quantify the diversity or information.

The Mutation rate was kept constant at 0.4 for all experiments, along with the population size of 100 and the grid size. Despite the infrastructure being build for scalable grids, all experiments focused on the typical Sudoku grid, as further experimentation left the scope of the project. Rudimentary experimentation determined that a population size of 100 was appropriate, although further works could improve on this.

5.1 Problem Set and Grading

By categorising our test problems, we can develop a more nuanced understanding of the GEA's performance. "Easy" problems feature $15 - 25$ empty cells, "medium" problems have $26 - 40$ empty cells, "hard" problems are void in $41 - 55$ cells, and "very hard" problems have 56+ empty cells. "Very hard" problems will not be used in every comparative analysis, but will be examined in our final results. Our test set is limited to three instances of each grading, due to the limited scope of the project. If greater computational resources were available, a larger set could have been adopted.

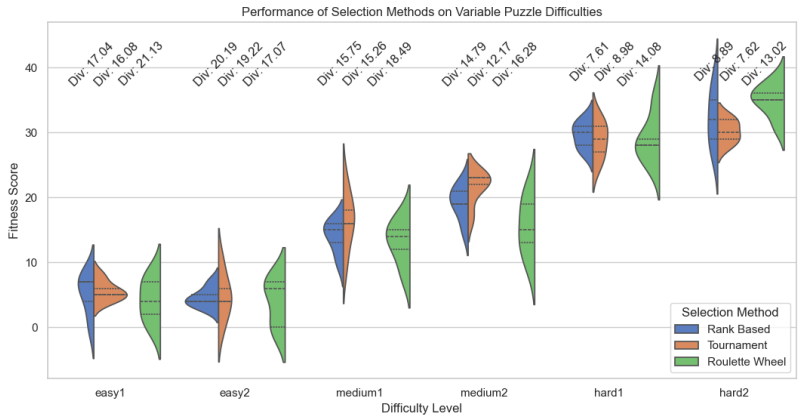


Figure 5: Violin chart comparison of **Selection methods** during GEA runtime, aggregated over 10 runs. Performance displayed on 6 distinct problems.

5.2 Crossover and Selection Methods Comparison

In this section, we analyse and compare the impacts of different crossover and selection methods on the performance of the GEA and EA. Focusing on the fitness of the fittest solution as the primary evaluation metric, we aim to identify optimal configurations for different scenarios. Secondary considerations include diversity and convergence rates, guiding the selection of methods that either promote quick convergence to suboptimal solutions or support more thorough exploration of the solution space. This comparison highlights the strengths and limitations of each approach, enabling the production of usage guidelines in other environments.

We achieve this by isolating the independent variable (selection or crossover method) and aggregating their performance over numerous runs. This method is applied to various problems for a comprehensive overview. By ensuring consistent parameters, we can enhance the fairness of comparisons and the validity of conclusions. We tested six sample problems, with two each from categories ranging from easy to hard. An automatic termination was set at 300 generations, and the results were aggregated over 10 runs. Although more instances per problem would be preferable, the execution time was a limiting factor, reducing the overall veracity of these tests. We decided that the most important metric is final solution fitness, with diversity and convergence as secondary considerations.

As seen in Fig. 5, the selection methods "Rank-based," "Tournament," and "Roulette Wheel" perform comparably. Roulette Wheel selection typically excels in diversity, except for the puzzle "easy2," where all methods perform decently. The dotted lines in each violin segment represent the spread of best fitness achieved, calculated using standard deviations. Tournament selection often shows consistent results with tightly coupled deviations, while Rank-based selection has a smaller overall spread. Roulette Wheel selection often outperforms the others in average fittest solution and best instance, despite poor performance on "hard2." Overall, Roulette Wheel selection is the best, likely due to its less harsh selection criteria, and was adopted for our GEA instance. While its diversity

benefit may be less useful in multi-objective problems, it seems advantageous for this single-objective problem.

As delineated by Fig. 6, "PMX" and "Cyclic" perform similarly on easy problems, with PMX even surpassing cyclic in terms of solution spread. However, cyclic excels on harder problems, consistently outperforming other crossover methods in average solution quality and best solution found. "Alternate Rows" crossover was consistently outperformed on all metrics by at least one of the other two methods, so can be discarded from consideration. While cyclic hampers diversity somewhat, typically scoring middling to low, overall solution quality was deemed more valuable. Therefore, cyclic crossover will be adopted for the GEA.

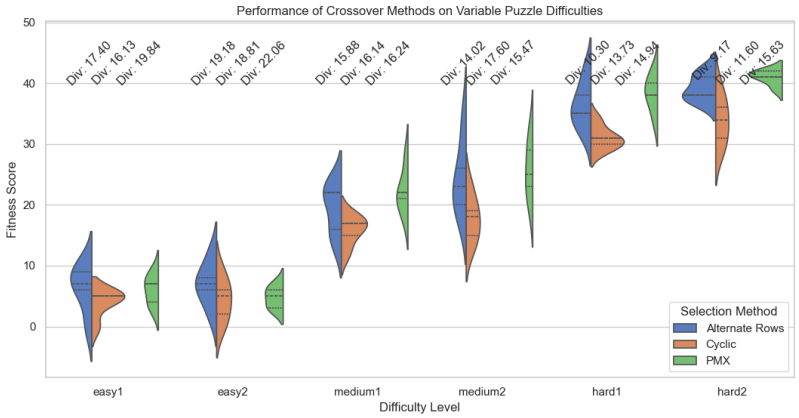


Figure 6: Violin chart comparison of **Crossover methods** during GEA runtime, aggregated over 10 runs. Performance displayed on 6 distinct problems.

5.3 Hyperparameter tuning

We need to ensure that the DM is optimised. By varying the internal architecture of the NN, we can achieve improved performance. In this test, we vary the number of hidden layers after the CNN and scalar input branch have concatenated, before the softmax output layer. This further optimises the GEA configuration for this problem and helps us develop usage guidelines, although the internal architecture of the GEA's DM will realistically vary significantly from problem to problem.

No. of layers	**0**	**1**	**2**	**3**
Easy	f_a:8.2 ± 3.6 D_a: 17.1	f_a:5.3 ± 2.9 D_a: 19.9	f_a:5.1 ± 2.0 D: 18.5	f_a:5.0 ± 2.7 D_a: 23.0
Medium	f_a:24.1 ± 7.3 D_a: 12.5	f_a:18.3 ± 4.9 D_a: 19.1	f_a:18.1 ± 3.1 D_a: 14.0	f_a:18.3 ± 5.3 D_a: 16.2
Hard	f_a:41.6 ± 10.3 D_a: 8.8	f_a:25.6 ± 5.2 D_a: 15.3	f_a:25.8 ± 6.1 D_a: 14.9	f_a:24.5 ± 5.4 D_a: 13.6

Table 2: Table displaying performance under variation of DM model. Different quantities of hidden layers shown against puzzle sets

We measure the average fitness of the configuration, f_a, and consider the standard deviations from this average, $f_a\pm$. We also monitor average diversity D_a across the instances. By isolating the model's architecture as the independent variable and keeping all other variables consistent, we can improve the integrity of the test. Each puzzle, from the categories of easy, medium, and hard, was tested, with results aggregated over 5 runs. More rigorous testing was desired but limited by scope. This process was repeated with various model architectures, increasing the number of hidden layers in each set. A termination of 200 generations was set. Table 2 displays the results of this experiment:

Zero hidden layers perform poorly across all metrics and should be discarded. For all other permutations, performance becomes more comparable. Although average fitness typically improves with more hidden layers, the variation is insignificant until hard problems are tested, where three layers average over one fitness point higher. Conversely, one hidden layer consistently scores well in diversity, only being outperformed on easy puzzles by the three-layer network, likely due to overfitting from other models.

One important consideration is the additional time required for backpropagation with more layers. When updates are frequent, this becomes increasingly expensive. Due to variability in local machine performance and the scientific standard of requiring 50+ runs for temporal analysis, comprehensive timing analysis was outside the project's scope. Despite this, more layers notably increased execution time. Therefore, one hidden layer was considered optimal, offering a slight performance degradation in exchange for faster execution and enhanced diversity.

5.4 Learning Rate

Another consideration when implementing the DM is the learning rate, which represents how often the reinforcement learning element updates on a generational basis. This balances DM optimisation and execution time, as updating and retraining the network is costly. By varying the update interval and testing the consequences, we can determine a trade-off between performance and solution quality.

Analysing average fitness, fitness spread (shaded area), and diversity helps determine when the performance-quality trade-off is no longer worthwhile. This experiment, run over 150 generations, aggregated five times, and tested across all puzzle categories, provides the necessary insight. Results are in Fig 5, with the update interval as the independent variable.

One instance converged before the timeout; otherwise, all tests took the full 200 generations, showing no distinct difference in convergence rates. Puzzle suites display inverse proportionality between update interval and solution quality, degrading slowly.

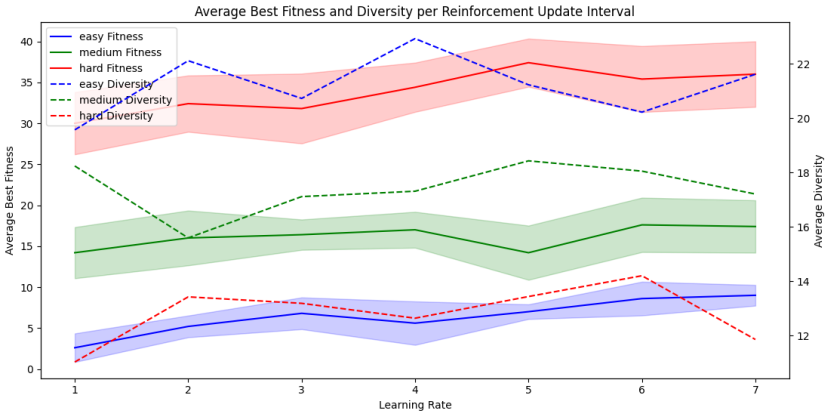


Figure 7: Bar chart of aggregated performance for the GEA with variable learning rates (interval between NN reinforcements). Tested across varying difficulties. Displays fitness and diversity.

Harder puzzles are harder to keep diverse and optimise. Update intervals of four or less seem reasonable, with three suffering slightly in performance. The optimal is one, but many rates are acceptable depending on the problem. Note that diversity suffers on harder problems as the learning rate increases, so a more frequent rate is preferred if diversity is a priority. To minimise the learning rate, a rate of three was adopted. Despite slight performance drops on easy puzzles, this could be due to algorithm stochasticity. Fitness spread up to a rate of three is consistently low except for hard tests. This update interval is now adopted for subsequent experiments, alongside previously decided design choices.

5.5 Cost-Benefit Analysis of Training

The variable section of our experimental setup is the mutation phase. Due to our combination of ES strategy and EA theory, the standard approaches mutation phase has a complexity of $G \cdot N \cdot M_c$, where G is the number of generations, N is the population size, and M_c is the mutation complexity. Despite the variation from the mutation pool (1), both instances of the algorithm pull from it, therefore it can be justified that the sum complexity from applying the mutations will be similar. The GEA approaches complexity can be delineated as $G \cdot N \cdot (M_c + P)$, where P is the complexity of making a prediction using the model. This is usually a low cost, since linear vector mathematics can be processed quickly. There is also an associated training cost to the DM, which can be expressed as: $\frac{G \cdot C_{bp}}{T_{interval}} + (G \cdot N \cdot f_{reward})$, where C_{bp} is the cost of performing back-propagation to update the model, $T_{interval}$ is the interval between model reinforcement, and f_{reward} is the complexity of the reward function. Again, f_{reward} is composed of linear arithmetic, and can be executed quickly.

By examining the two configurations side by side, we can deduce that the additional costs of the GEA are $G \cdot N \cdot (P + f_{reward}) + \frac{G \cdot C_{bp}}{T_{interval}}$. For this to be valuable, the difference in generations ΔG make up for these additional calculations. When applied to other problems with more intensive processes during other evolutionary phases, such as fitness evaluations, the reduction in G becomes more important.

When applied to this problem, there are not particularly high returns, however, there is great potential in fields such as aerospace engineering [58] and Energy systems [59], where fitness evaluations are often very complex, requiring fluid dynamic models or similarly complicated representations.

5.6 Final comparison

This experiment compares the quality of the GEA against a standard EA. To ensure fairness, all parameters were kept constant except for the mutation selection method. A termination criterion of

150 generations was set, with data aggregated over ten runs for each of the three problem instances under each problem grading. The compound Fig. 12 displays the results of the experiment:

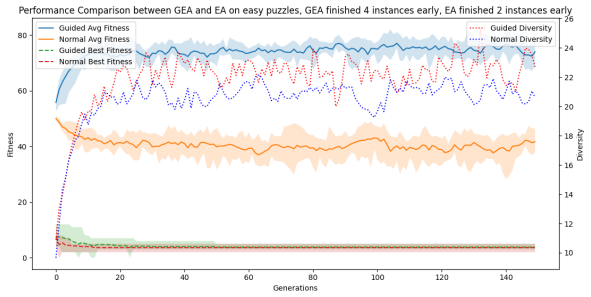


Figure 8: Easy

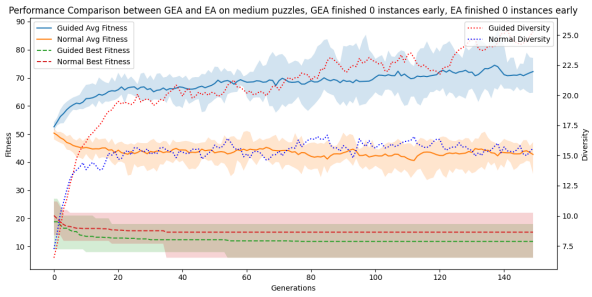


Figure 9: Medium

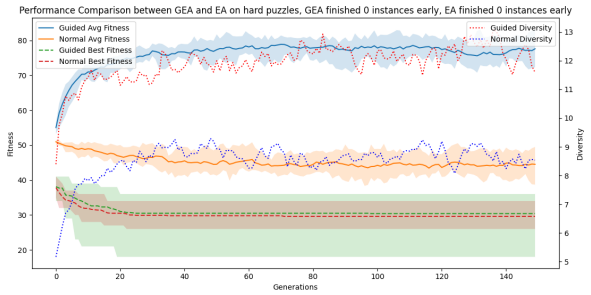


Figure 10: Hard

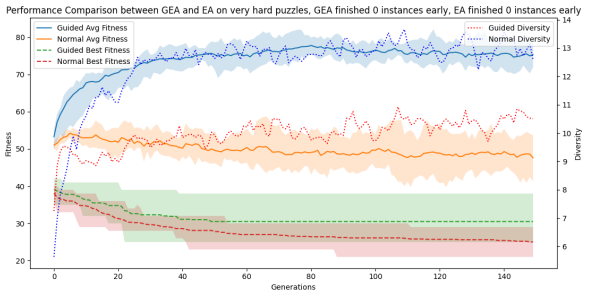


Figure 11: Very Hard

Figure 12: Comparing EA and GEA performance - using best fitness, average fitness and diversity (entropy)

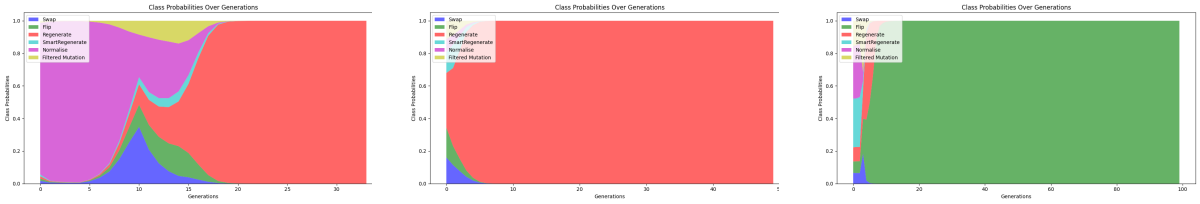
Easy puzzle suite (Fig. 8): The GEA completed more puzzles (4) than the EA (2), although the EA's rapid completion of one puzzle is likely due to random chance. Both methods maintained similarly diverse populations, but the GEA showed higher average diversity, suggesting a stronger emphasis on exploration. **Medium puzzle suite (Fig. 9):** The results were more distinct in the medium puzzle suite. The GEA showed a marked difference in exploration, evident in both average fitness and diversity, with diversity continually rising. Despite the average best fitness being better for the EA, it briefly outperformed the GEA, which could be attributed to stochasticity. **Hard puzzle suite (Fig. 10):** In the hard puzzle suite, the GEA had slightly lower average best fitness performance but better best case performance, with a significantly fitter solution being found. The difference in diversity remained evident, as shown by the average fitness and entropy. **Very hard puzzle suite (Fig. 11):** The EA outperformed the GEA in terms of the best solution. While the EA steadily improved its best solution, the GEA appeared to hit local optima and then prioritised exploration. Diversity remained higher for the GEA in both metrics. This suggests that the GEA might benefit from being adaptive to problem difficulty or incorporating a mechanism to maintain optimisation focus.

In summary, the GEA demonstrated greater diversity and higher average fitness across all puzzle suites, implying that it tends to favour exploration. This is likely due to the nature of the reward function. While the GEA typically provides slight improvements in convergence (measured by generations until the best solution is found), overall quality of the best solution, and diversity, the benefits in terms of quality and convergence are minor, almost negligible. The GEA in most instances provides the best (or shared best) solutions, although not with the degree of consistency that would indicate a significant upgrade. Overall, this suggests the GEA can be particularly useful for problems requiring high diversity, such as those with highly multi-modal landscapes.

5.7 Mutation chance over generations

To investigate the GEA's propensity for getting stuck in local optima, an analysis of mutation distributions was performed. Each generation, the predicted distribution for the fittest solution was returned. This couldn't be extended to the entire population due to variation in class distribution

output. These tests validate that the CNN and scalar inputs are meaningful, but they show the DM tends towards one class output (see Fig. 13).



(a) (b) (c)

Figure 13: Mutation class distribution over generations

To address this, an additional penalty term was introduced for single-class outputs, but it made no difference, suggesting issues with exploding or vanishing gradients [60]. Gradient clipping [61] was applied but was ineffective, indicating the problem lies with the reward function. Despite this, Fig. 13 (a) shows an effective run. The model favoured convergence until around generation 10, then shifted to prioritising diversity, eventually finding an optimal solution. Notably, (a) still had single-class output issues but managed to overcome the local optima.

5.8 Robustness and Stability Analysis

By studying edge cases, one can identify limitations, allowing for the identification of specific conditions under which the algorithms might fail or under-perform. This can also improve robustness. By understanding how the algorithms behave in extreme situations, enhancements can be made. We investigated the following edge cases: **Empty column:** Solved consistently in the first generation. **Empty subgrid:** Solved consistently in the first generation. **Empty row:** Often took more generations to complete, suggesting mutations are skewed against row resolution. **Empty grid:** The reliance on diversity helped, allowing frequent improvements in solution quality, but no instances were solved within 200 generations.

5.9 Comparison to Aims

To evaluate the success of this project, we compare the outcomes to the stated aims. Completing these aims demonstrates progress in applying NN-enhanced EAs.
Sudoku Puzzle Solving: This aim was met. The developed EA solved or relatively improved the fitness of various Sudoku grids, demonstrating adaptability and effectiveness across different configurations, although encountered difficulty against harder grids.
Neural Network Decision Maker Implementation: This aim was achieved, but improvements are possible. The NN guided mutation selection within the EA, though further refinement could enhance performance, particularly in optimising the reward function.
Performance Evaluation of NN-Enhanced EA: This aim was met, though more intensive testing is needed for thorough validation. The GEA showed improved diversity compared to a traditional EA, with slight increases in convergence rates and overall solution quality, especially in maintaining diversity.

Overall, the project met its aims with a reasonable degree of success, showcasing the potential of integrating NN with EAs.

6 Conclusion

The project objectives were successfully met. The EA developed effectively solved or improved the fitness of Sudoku puzzles, demonstrating adaptability across various configurations. The integration of a NN within the EA was implemented, guiding mutation selection and enhancing performance. Performance metric and testing were implemented, although the hyperparameters of the GEA could be improved upon.

Further experimentation with the number of mutations in the decision maker should be pursued, potentially by future academic efforts. Additionally, reworking the mutation strategies may improve performance, particularly for row-solving tasks. The GEA indicates that it would prove useful in scenarios requiring high diversity, such as highly multi-modal landscapes. It could be argued that a highly exploratory approach to mutation and crossover may yield similar benefits. However, the current reward function appears to overly favour diversity, indicating a need for reevaluation and

potential reworking to achieve a more balanced optimisation. Overall, the project demonstrated the potential of integrating NNs with EAs for complex problem-solving, while also identifying areas for further enhancement. Finally, the described techniques could be highly useful when applied to problems with very long function evaluations, aiding a faster convergence.

6.1 Usage Guidelines

To guide the implementation of GEAs in various contexts, this section provides recommendations based on the observed performance of different configurations. These guidelines highlight the strengths of selection methods, crossover techniques, and learning rates.
Promoting Exploratory Moves: For multi-modal landscapes, use Roulette Wheel selection and cyclic crossover. A higher learning rate enhances exploration, helping avoid local optima.
Rapid Convergence: For quicker convergence, use Tournament selection with PMX crossover. A moderate learning rate should ensure stability and faster convergence.
Achieving Optimal Fitness Values: For high fitness values, use Rank-based selection and cyclic crossover. Maximise learning rate for a temporal cost.
Additional Considerations:
Mutation Rate: Maintain at 0.4 for balance, employ ES techniques for single-objective optimisation.
Population Size: 100 is effective; increase for complex problems with bigger genomes.

These guidelines provide a starting point for implementing GEAs in diverse problem settings, ensuring that the chosen configuration aligns with the desired outcomes.

6.2 Difficulties and Reflections

Several challenges were encountered during the project. Breaking out of local optima proved difficult, and optimising parameters was an ongoing task. The model frequently experienced issues with exploding or vanishing gradients, leading to fixation on a single classification. Additionally, the algorithm sometimes adopted suboptimal diversity mutations. The reward function, which favoured diversity, should prioritise fitness more to enhance overall performance. Despite these challenges, the project provided valuable insights and highlighted areas for future improvement.

Although not tested in the scope of this report, scalability of grids was inbuilt into this testing framework, and can be explored by further researchers.

6.3 Summary of Findings

Overall, the integration of NN and RL elements in the GEA showed potential but requires further optimisation. The GEA improved diversity and fitness across various problem difficulties but struggled with local optima and single-class output issues. Key recommendations include refining the reward function and optimising hyperparameters like the NN architecture and learning rate for better performance.

6.4 Future Work

To enhance the GEA, future research will focus on advanced optimisation techniques and expanding its applicability.

- **Weighted Initialisation:** Introduce weighted initialisation [62] to bias the algorithm towards either convergence or exploration, depending on the landscape.
- **Hyperparameter Optimisation:** Implement grid search [63] or Bayesian optimisation [64] for fine-tuning NN parameters, ensuring the most effective configuration.
- **Parallelisation:** Develop parallel processing techniques to improve computational efficiency and handle larger populations.
- **NN-Guided Crossover:** Explore using NN's to guide crossover points, improving solution quality by indicating optimal crossover locations.
- **Integration with MOEAs:** Adapt the model for Multi-Objective Evolutionary Algorithms to optimise multiple conflicting objectives simultaneously.
- **Bayesian Neural Networks:** Integrate Bayesian Neural Networks [65] to incorporate uncertainty in the decision-making process, enhancing outcome likelihood estimation.
- **Expanding Applicability:** Test and validate the model across diverse problems to refine its adaptability and effectiveness in various settings.

References

[1] A Moraglio. *Example of a Sudoku puzzle*. https://www.researchgate.net/figure/Example-of-Sudoku-puzzle_fig1_220739836. Accessed on: 2023-11-20. 2007.

[2] *The History of Sudoku*. Accessed on: 2023-11-01. URL: https://sudoku.com/how-to-play/the-history-of-sudoku/.

[3] Takayuki Yato and Takahiro Seta. "Complexity and completeness of finding another solution and its application to puzzles". In: *IEICE transactions on fundamentals of electronics, communications and computer sciences* 86.5 (2003), pp. 1052–1060.

[4] Larry Hardesty. *Explained: P vs. np*. URL: https://news.mit.edu/2009/explainer-pnp.

[5] Behnam Esfahbod. *Euler diagram for P, NP, NP-complete, and NP-hard set of problems*. https://commons.wikimedia.org/wiki/File:P_np_np-complete_np-hard.svg. Accessed: 01/05/2024. 2007.

[6] Ekrem Duman and I Or. "Precedence constrained TSP arising in printed circuit board assembly". In: *International Journal of Production Research* 42.1 (2004), pp. 67–78.

[7] Quang Minh Ha et al. "On the min-cost traveling salesman problem with drone". In: *Transportation Research Part C: Emerging Technologies* 86 (2018), pp. 597–621.

[8] Ajit Narayanan, Spiridon Zorbalas, et al. "DNA algorithms for computing shortest paths". In: *Proceedings of genetic programming* 718 (1998), p. 723.

[9] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[10] Larry R Medsker and LC Jain. "Recurrent neural networks". In: *Design and Applications* 5.64-67 (2001), p. 2.

[11] Dhanya Job and Varghese Paul. "Recursive backtracking for solving 9* 9 Sudoku puzzle". In: *Bonfring International Journal of Data Mining* 6.1 (2016), pp. 7–9.

[12] Donald E Knuth. "Dancing links". In: *arXiv preprint cs/0011047* (2000).

[13] Bernhard Hobiger. *Welcome to hodoku, a sudoku generator/solver/trainer/analyzer*. URL: https://hodoku.sourceforge.net/en/index.php.

[14] Helmut Simonis. "Sudoku as a constraint problem". In: *CP Workshop on modeling and reformulating Constraint Satisfaction Problems*. Vol. 12. Citeseer. 2005, pp. 13–27.

[15] Inês Lynce and Joël Ouaknine. "Sudoku as a SAT Problem". In: *AI&M*. 2006. URL: https://api.semanticscholar.org/CorpusID:10853506.

[16] Nils Aall Barricelli et al. "Esempi numerici di processi di evoluzione". In: *Methodos* 6.21-22 (1954). Translated and summarised by ChatGPT, pp. 45–68.

[17] Thomas Bäck and Hans-Paul Schwefel. "An overview of evolutionary algorithms for parameter optimization". In: *Evolutionary computation* 1.1 (1993), pp. 1–23.

[18] John H Holland. "Genetic algorithms". In: *Scientific american* 267.1 (1992), pp. 66–73.

[19] Charles Darwin and William F Bynum. *The origin of species by means of natural selection: or, the preservation of favored races in the struggle for life*. AL Burt New York, 2009.

[20] James Kennedy and Russell Eberhart. "Particle swarm optimization". In: *Proceedings of ICNN'95-international conference on neural networks*. Vol. 4. IEEE. 1995, pp. 1942–1948.

[21] Kay Chen Tan et al. "Balancing exploration and exploitation with adaptive variation for evolutionary multi-objective optimization". In: *European Journal of Operational Research* 197.2 (2009), pp. 701–713.

[22] Jonathan R Friedman et al. "Quantum superposition of distinct macroscopic states". In: *nature* 406.6791 (2000), pp. 43–46.

[23] Gerasimos G Rigatos and Spyros G Tzafestas. "Parallelization of a fuzzy control algorithm using quantum computation". In: *IEEE Transactions on Fuzzy Systems* 10.4 (2002), pp. 451–460.

[24] Lov K Grover. "A framework for fast quantum mechanical algorithms". In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 1998, pp. 53–62.

[25] Shauharda Khadka and Kagan Tumer. "Evolutionary reinforcement learning". In: *arXiv preprint arXiv:1805.07917* 223 (2018).

[26] Madalina M. Drugan. "Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms". In: *Swarm and Evolutionary Computation* 44 (2019), pp. 228–246. ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2018.03.011. URL: https://www.sciencedirect.com/science/article/pii/S2210650217302766.

[27] Charlie Vanaret et al. "Preventing premature convergence and proving the optimality in evolutionary algorithms". In: *Artificial Evolution: 11th International Conference, Evolution Artificielle, EA 2013, Bordeaux, France, October 21-23, 2013. Revised Selected Papers 11*. Springer. 2014, pp. 29–40.

[28] Chiang-Heng Chien, Wei-Yen Wang, and Chen-Chien Hsu. "Multi-objective evolutionary approach to prevent premature convergence in Monte Carlo localization". In: *Applied Soft Computing* 50 (2017), pp. 260–279.

[29] G. Rudolph. "Self-adaptive mutations may lead to premature convergence". In: *IEEE Transactions on Evolutionary Computation* 5.4 (2001), pp. 410–414. DOI: 10.1109/4235.942534.

[30] Maria Angelova and Tania Pencheva. "Tuning genetic algorithm parameters to improve convergence time". In: *International Journal of Chemical Engineering* 2011 (2011).

[31] Abu Bakar et al. "Design of low altitude long endurance solar-powered UAV using genetic algorithm". In: *Aerospace* 8.8 (2021), p. 228.

[32] D Pralhaddas Kothari. "Power system optimization". In: *2012 2nd National conference on computational intelligence and signal processing (CISP)*. IEEE. 2012, pp. 18–21.

[33] Jeffrey Horn and David E Goldberg. "Genetic algorithm difficulty and the modality of fitness landscapes". In: *Foundations of genetic algorithms*. Vol. 3. Elsevier, 1995, pp. 243–269.

[34] Tuan Hao Hoang, Nguyen Xuan Hoai, and RI Mc Kay. "Does it matter where you start? A comparison of two initialisation strategies for grammar guided genetic programming". In: (2004).

[35] *Python 3.12.0 Documentation*. Accessed on: 2023-11-20. URL: https://docs.python.org/3/.

[36] J. Jilg and J. Carter. "Sudoku evolution". In: *2009 International IEEE Consumer Electronics Society's Games Innovations Conference*. 2009, pp. 173–185. DOI: 10.1109/ICEGIC.2009.5293614.

[37] A. Moraglio, J. Togelius, and S. Lucas. "Product Geometric Crossover for the Sudoku Puzzle". In: *2006 IEEE International Conference on Evolutionary Computation*. 2006, pp. 470–476. DOI: 10.1109/CEC.2006.1688347.

[38] Padmavathi Kora and Priyanka Yadlapalli. "Crossover operators in genetic algorithms: A review". In: *International Journal of Computer Applications* 162.10 (2017).

[39] David E Goldberg and Robert Lingle. "Alleles, loci, and the traveling salesman problem". In: *Proceedings of the first international conference on genetic algorithms and their applications*. Psychology Press. 2014, pp. 154–159.

[40] Peter JB Hancock. "An empirical comparison of selection methods in evolutionary algorithms". In: *AISB workshop on evolutionary computing*. Springer. 1994, pp. 80–94.

[41] Chang Wook Ahn and Rudrapatna S Ramakrishna. "Elitism-based compact genetic algorithms". In: *IEEE Transactions on Evolutionary Computation* 7.4 (2003), pp. 367–385.

[42] Carlos Segura et al. "The importance of diversity in the application of evolutionary algorithms to the Sudoku problem". In: *2016 IEEE Congress on Evolutionary Computation (CEC)*. 2016, pp. 919–926. DOI: 10.1109/CEC.2016.7743888.

[43] Zhiwen Wang, Toshiyuki Yasuda, and Kazuhiro Ohkura. "An evolutionary approach to sudoku puzzles with filtered mutations". In: *2015 IEEE Congress on Evolutionary Computation (CEC)*. 2015, pp. 1732–1737. DOI: 10.1109/CEC.2015.7257096.

[44] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network". In: *2017 international conference on engineering and technology (ICET)*. Ieee. 2017, pp. 1–6.

[45] Song Han et al. "Dsd: Dense-sparse-dense training for deep neural networks". In: *arXiv preprint arXiv:1607.04381* (2016).

[46] VL Helen Josephine, AP Nirmala, and Vijaya Lakshmi Alluri. "Impact of hidden dense layers in convolutional neural network to enhance performance of classification model". In: *IOP Conference Series: Materials Science and Engineering*. Vol. 1131. 1. IOP Publishing. 2021, p. 012007.

[47] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. "Activation functions in neural networks". In: *Towards Data Sci* 6.12 (2017), pp. 310–316.

[48] Gregory E Sims et al. "Alignment-free genome comparison with feature frequency profiles (FFP) and optimal resolutions". In: *Proceedings of the National Academy of Sciences* 106.8 (2009), pp. 2677–2682.

[49] *Numpy Documentation*. Accessed on: 2023-11-20. URL: `https://numpy.org/doc/`.

[50] *Pandas documentation - pandas 2.1.3 documentation*. Accessed on: 2023-11-20. URL: `https://pandas.pydata.org/docs/`.

[51] *TensorFlow documentation*. Accessed on: 2023-11-20. URL: `https://www.tensorflow.org/api_docs`.

[52] *Scikit-learn documentation*. Accessed on: 2023-11-20. URL: `https://scikit-learn.org/stable/`.

[53] Robert Hecht-Nielsen. "Theory of the backpropagation neural network". In: *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[54] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[55] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[56] Geoffrey Hinton. *Lecture 6e - Neural Networks for Machine Learning*. Coursera. Available: `[URLofthespecificlectureorcourse]`. 2012.

[57] *PyTorch documentation - PyTorch 2.1 documentation*. Accessed on: 2023-11-20. URL: `https://pytorch.org/docs/stable/index.html`.

[58] Edmondo Minisci and Massimiliano Vasile. "Robust design of a re-entry unmanned space vehicle by multi-fidelity evolution control". In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. 2011, pp. 689–696.

[59] Ajit C Pillai et al. "Optimisation of offshore wind farms using a genetic algorithm". In: *International Journal of Offshore and Polar Engineering* 26.03 (2016), pp. 225–234.

[60] Boris Hanin. "Which neural net architectures give rise to exploding and vanishing gradients?" In: *Advances in neural information processing systems* 31 (2018).

[61] Jingzhao Zhang et al. "Why gradient clipping accelerates training: A theoretical justification for adaptivity". In: *arXiv preprint arXiv:1905.11881* (2019).

[62] Siddharth Krishna Kumar. "On weight initialization in deep neural networks". In: *arXiv preprint arXiv:1704.08863* (2017).

[63] Petro Liashchynskyi and Pavlo Liashchynskyi. "Grid search, random search, genetic algorithm: a big comparison for NAS". In: *arXiv preprint arXiv:1912.06059* (2019).

[64] Kirthevasan Kandasamy et al. "Neural architecture search with bayesian optimisation and optimal transport". In: *Advances in neural information processing systems* 31 (2018).

[65] Pavel Izmailov et al. "What are Bayesian neural network posteriors really like?" In: *International conference on machine learning*. PMLR. 2021, pp. 4629–4640.