

## Highlights

### Solving nonograms using Neural Networks

José María Buades Rubio, Antoni Jaume-i-Capó, David López González, Gabriel Moyà Alcover

- We analyze and compared different techniques to solve nonograms.
- We combined the Heuristic algorithm with neural networks.
- We generate a public dataset of nonograms to train the neural networks.

# Solving nonograms using Neural Networks

José María Buades Rubio<sup>a</sup>, Antoni Jaume-i-Capó<sup>a,b</sup>, David López González,  
Gabriel Moyà Alcover<sup>a,b,\*</sup>

<sup>a</sup>*Computer Graphics and Vision and AI Group (UGiVIA), Research Institute of Health Sciences (IUNICS), Departament de Matemàtiques i Informàtica, Universitat de les Illes Balears, Ctra Valldemossa Km 7.5, Palma, 07122, Illes Balears, Spain*

<sup>b</sup>*Laboratory for Artificial Intelligence Applications (LAIA@UIB), Universitat de les Illes Balears, Ctra Valldemossa Km 7.5, Palma, 07122, Illes Balears, Spain*

---

\*Corresponding author

Email addresses: [josemaria.buades@uib.es](mailto:josemaria.buades@uib.es) (José María Buades Rubio ), [antonи.jaume@uib.es](mailto:antonи.jaume@uib.es) (Antoni Jaume-i-Capó), [lopezdavidg.com@outlook.com](mailto:lopezdavidg.com@outlook.com) (David López González), [gabriel.moya@uib.es](mailto:gabriel.moya@uib.es) (Gabriel Moyà Alcover )

---

## Abstract

Nonograms are logic puzzles in which cells in a grid must be colored or left blank according to the numbers that are located in its headers. In this study, we analyze different techniques to solve this type of logical problem using an Heuristic Algorithm, Genetic Algorithm, and Heuristic Algorithm with Neural Network. Furthermore, we generate a public dataset to train the neural networks. We published this dataset and the code of the algorithms. Combination of the heuristic algorithm with a neural network obtained the best results. From state of the art review, no previous works used neural network to solve nonograms, nor combined a network with other algorithms to accelerate the resolution process.

*Keywords:* Nonograms, Artificial Intelligence, Nonogram solver, Neural networks, Depth first search, Genetic algorithms

---

## 1. Introduction

A nonogram, which is also known as a Picross or Hanjie, is a Japanese logic puzzle in which cells in a grid must be colored or left blank according to a set of numbers that is located at the side of the board, also known as row and column headers, to reconstruct a binary image. Each header indicates the number of cells that must be marked in a row inside the board to construct a block. If there is more than one number in the same row or column header, at least one empty cell must exist between them. Puzzles of an arbitrary size can be defined as rectangular or square. The cells of a nonogram are defined by two states: filled (| ■ |) and empty (| x |).

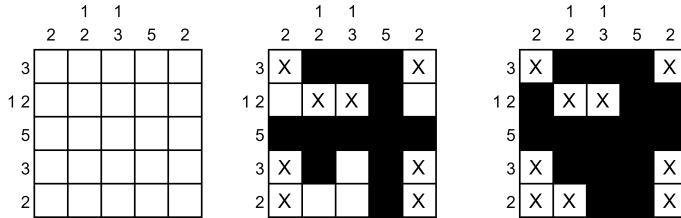


Figure 1: Examples of different nonogram states: unsolved, partially solved, and solved. The black cells are considered as filled, whereas those with a cross are empty.

Figure 1 depicts the three stages of nonogram resolution: unsolved, partially solved, and solved. Note that this type of problem falls into the category of NP completeness [1, 2, 3]; thus, a solution cannot be obtained in polynomial time. Moreover, certain nonograms do not have a single solution, and all solutions that are compatible with the constraints defined by their headers are valid. An example of the situation is illustrated in Figure 2.

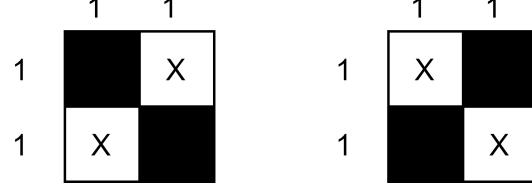


Figure 2: Nonogram with multiple solutions.

### 1.1. Problem specification

A nonogram is a board that is defined by a matrix of size  $n \times m$  with  $n, m \geq 1$ . As stated previously, each cell has two possible states: filled and empty, which are represented by the values  $\{0, 1\}$ . We define  $\Omega_{n \times m}$  as the space of possible values for boards of size  $n \times m$ :  $\Omega_{n \times m} : \{0, 1\}^{n \times m}$ . For example, boards of size  $5 \times 5$  have 25 cells; consequently, there are  $2^{25}$  boards of this size.

In this study, we only consider squared boards; therefore, for all nonograms,  $n = m$ . This implies that the rows and columns of these squared boards have the same possible codifications. We use  $\Omega_n$  instead of  $\Omega_{n \times n}$  to simplify the notation. Given a board of size  $n$ , we define  $C_n$  as the space of possible headers, and denote the encoding space of the boards as  $|\Phi|$ , where  $|C_n|^{2n}$ .

0	X   X   X   X   X	3	XXXXX   X   X
1	X   X   X   X	3	X   XXX   X
1	X   X   X     X	3	X   X   XXX
1	X   X     X   X	2 1	X   X   X
1	X     X   X   X	2 1	X   X   X   X
1	X   X   X   X	2 1	X   X   X   X
2	XXX   X   X   X	1 2	X   X   X
2	X   XXX   X	1 2	X   X   X
2	X   X   XXX   X	1 2	X   X   X   X
2	X   X   X   X	1 1 1	X   X   X
1 1	X   X   X   X	4	XXXX   X
1 1	X   X   X	4	X   XXX
1 1	X   X     X   X	3 1	X   XXX
1 1	X   X   X     X	2 2	X   X   X
1 1	X   X   X   X	1 3	X   X   X   X
1 1	X   X   X   X	5	XXXXX

Figure 3: Example of all existing configurations in a five-column nonogram. The numbers represent the row encodings.

If  $n = 5$  the set of possible header encodings is:  $C_5 = \{0, 1, 2, 3, 4, 5, 11, 12, 21, 13, 31, 111\}$ , where  $|C_5| = 12$ . Therefore,  $|\Phi_n| = 12^{10}$ , being  $|\Phi_5| > |\Omega_5|$ . This is because not all elements of  $\Phi_5$  are valid, as they do not represent an element of  $\Omega_5$ , this happens for any  $n$ . Having  $|\Phi_n| > |\Omega_n|$  for all  $n$ , following the same example, for  $n = 5$  a column/row may have the codifications depicted in Figure 3.

We denote  $\Psi_n$  as the subset of elements of  $\Phi_n$  such that they are represented by a possible board in  $\Omega_n$ . This implies that  $\Psi_n$  contains only valid encodings. The encoding space  $\Psi_n$  has fewer elements than  $\Omega_n$  ( $|\Psi_n| < |\Omega_n|$ ). Table 1 displays the relationship between  $|\Omega|$  and  $|\Psi|$  for  $n \in [1, 5]$ .

Size ( $n \times n$ )	$ \Omega_n $	$ \Psi_n $	Ratio
$1 \times 1$	2	2	1
$2 \times 2$	16	15	0.938
$3 \times 3$	512	445	0.869
$4 \times 4$	65536	58196	0.888
$5 \times 5$	33554432	28781820	0.858

Table 1: For each square board sized 1 to 5, we show the number of possible boards ( $|\Omega_n|$ ), number of different valid encodings ( $|\Psi_n|$ ), and ratio  $|\Psi_n|/|\Omega_n|$ .

The coding of a board is an application  $f$ :

$$f : \Omega_n \rightarrow \Psi_n.$$

The solution of a nonogram involves a decoding process. As a step in this process, given an element  $\psi \in \Psi_n$ , determine an element  $\omega \in \Omega_n$  that verifies  $f(\omega) = \psi$ . There is not always a single solution; for every  $n$  there is an element  $\psi \in \Psi_n$  with all row and column headers equal to 1, such that it has as the solution  $n!$  elements of  $\Omega_n$ .

A solver must perform the decoding process returning a compatible board with headers within the shortest amount of time. We measure the performance of different algorithms.

### 1.2. State of the art

In the literature, we identified five main strategies for solving nonograms efficiently: heuristic, depth-first search (DFS), genetic algorithms (GAs), and reinforcement learning (RL).

#### 1.2.1. Heuristic algorithms

Regarding heuristic algorithms, Salcedo-Sanz *et al.* [4] designed a set of ad-hoc heuristics. In particular, they proposed a combinatorial ad-hoc heuristic based on trying feasible combinations of solutions in each row and column of the puzzle that may fail in solving large puzzle and a logic ad-hoc heuristic, that starts with a pre-processing step, where they filled trivial cells, and is based on the calculation of the feasible right-most and left-most solution for a

given line, where right-most stands for the feasible solution which has the first filled square of each condition most in the right, that solved all the 19 nonograms that they used to validate their proposal in less than half a second. Batenburg and Kosters [5] proposed a reasoning framework that can be used to determine the value of certain pixels in a nonogram; by iterating their procedure, starting from an empty grid, it is possible to solve the nonogram completely. Another rules-based approach was described in [6], and Khan implemented a new integer linear programming formulation, that is an optimization algorithm where the variables are integer values and the objective function and equations are linear.

### 1.2.2. Depth-first search algorithms

Concerning DFS, researchers [7, 8, 9] combined DFS algorithms and the use of logical rules. The key concept is to obtain as much information as possible by applying logical rules to detect cells which can be determined immediately at first and then applying backtracking to search for the puzzle solution. Yu *et al.* [8] improved the search process using information that was obtained from a two sets of logical rules, first set contain 5 rules and the second set contain 3 rules to refine the results obtained first set, then they applied a backtracking algorithm. Jing *et al.* [7] proposed a solution based on the fact that most of Japanese puzzle are compact and contiguous, they defined a set of 11 rules divided in three main parts and then they applied the branch-and-bound algorithm to improve the search process. Finally, Stefani *et al.* [9] applied rule-based techniques that consist of simple boxes, simple spaces, forcing, and contradiction then the best-first search is used to solve the puzzle.

Wikeckowski *et al.* [10] modified the DFS method, they also used a soft computing method based on permutation generation and used both to solve nonograms. The two proposed algorithms were analyzed to obtain a solution, the number of iterations, and the time required to obtain the final state. They concluded that, in contrast to the soft computing method, the DFS algorithm guaranteed correct solutions regardless of the level of difficulty of the nonogram.

### 1.2.3. Genetic algorithms

Respecting GA, Tsai [11] proposed a Taguchi-based GA that was effectively applied to solve nonograms, and tested their approach on large nonograms (from  $15 \times 15$  to  $30 \times 25$ ), managing to solve more than 50% of the nonograms. Bobko and Grzywacz [12] presented the concept of using classical GAs as a tool for solving nonograms. They obtained good results with nonograms of sizes  $4 \times 4$  and  $5 \times 5$ , but in the case of  $10 \times 10$ , with a chromosome length of 100, none of their proposals yielded the expected result. Soto *et al.* [13] developed a GA that uses a new generation of the initial population and experimentally applied it to  $5 \times 5$  boards.

Habes and Hasan [14] proposed an adaptation of the particle filter, called Particle Swarm Optimization which is a population based stochastic optimization method, to solve nonograms and performed an experiment using only three boards with different sizes.

Chen and Lin [15] proposed to add a new parameter in the dynamic programming approach to solve nonograms to finish maximal painting earlier without significant overhead expense. Their approach was designed to solve nonograms of size  $5 \times 5$  under tournament constraints.

#### 1.2.4. Reinforcement learning algorithms

Finally, as for RL ,Shultz *et al.* [16] applied RL to solve this problem. They found that the RL solvers learned near-optimal solutions that outperformed a heuristic solver based on general rules, but they only applied their solution to solve  $5 \times 5$  boards because of the computational time required for larger puzzles.

#### 1.3. Research objective

Our main objective was to determine whether a neural network can improve the performance of two well-known algorithms (DFS and GA) to solve nonograms. The hypothesis was neural networks can help to solve nonogram efficiently From literature review, no previous works used a neural network to solve nonograms, nor combined a network with other algorithms to accelerate the resolution process. For this reason, in this study, we developed two new approaches. First, we used a combination of the DFS and a neural network. Second, we combined a GA and a neural network. Furthermore, we developed a dataset to evaluate the proposed solutions. This dataset and the algorithm code were published.

#### 1.4. Outline

The remainder of this paper is organized as follows: In Section (2), we explain the methods that we developed to solve nonograms. Section (3) presents the experimental setup, including a description of the dataset that we created, the metrics used, and the experiments that were performed. In Section (4), we outline the obtained results and discuss the performance of the proposed methods. Finally, Section (5) summarizes the conclusions and presents the strengths and shortcomings of the proposed system.

## 2. Methods

In this section, we explain the networks that we designed, as well as the algorithms that we used for solving the boards. These well-known algorithms were modified to include the networks as a support for the decision-making process.

#### 2.1. Network design

We employed a fully connected (FC) neural network, which is an artificial neural network that is composed solely of fully connected layers. The input to the network was a vector of size  $LM \cdot width + LM \cdot height$ , where  $width = height = n$ ;  $LM = \lfloor (n+1)/2 \rfloor$ , and  $n$  was the board size. We constructed this vector by concatenating all headers. To make them equal in size, we added

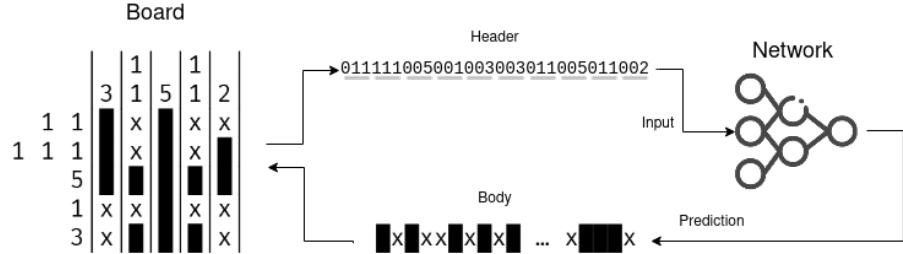


Figure 4: Given the nonogram on the left, we codify its headers and construct a one-dimensional vector. This vector is the network input. As a result, we obtain a vector of size  $width \cdot height$  that we can easily transform into a nonogram board.

leading zeros to each header until its size was equal to  $LM$  (see Figure 4 for details).

The output of the network was a vector of size  $width \cdot height$  and each element of this vector had two possible states: empty or filled. This vector was converted into a board shape of size  $(width, height)$  by applying a simple transformation. See Figure 4 for further details.

Owing to the characteristics of the designed networks, they require an input of a fixed size; thus, a model that is trained for a  $5 \times 5$  board will not be useful for solving  $10 \times 10$  or  $15 \times 15$  boards.

## 2.2. Reflections

As stated previously, we can represent nonograms in two spaces: the board space  $\Omega_n$  and header space  $\Psi_n$ . The space  $\Omega$  represents all existing boards. For example, for a  $5 \times 5$  board using two states, there are  $2^{25}$  possibilities. The space  $\Psi_n$  represents all valid header combinations; in the case of a  $5 \times 5$  board, the number is 28781820. Moreover, several elements of the space  $\Psi$  can represent different boards; that is, the encoding operation is not bijective, as illustrated in Figure 2.

To improve the network performance, we searched for bijective functions in the  $\Psi$  space, which implied a bijection in the  $\Omega$  space. Reflections are functions that fulfill this requirement. By applying these reflections, we could improve the network results; if the network learned one of the reflections, it could make a correct prediction. We defined eight bijective functions that form a noncommutative group, which can be generated using the following three functions and identity ( $id$ ): swapping rows and columns ( $g$ : diagonal reflection, where  $g \circ g = id$ ), reversing the row order ( $f_v$ : vertical reflection, where  $f_v \circ f_v = id$ ), and reversing the column order ( $f_h$ : horizontal reflection, where  $f_h \circ f_h = id$ ).

The vertical and horizontal reflections produce header realignment and provoke a swap of the header values. For example, if a row has header  $1\ 2$  when horizontal reflection is applied, it becomes  $2\ 1$ . The eight bijective functions are depicted in Figure 5 and explained below.

1. **rc** → Rows first, columns second. Identity function  $id$ .
2. **cr** → Columns first, rows second. Diagonal reflection  $g$ . Interchange rows and columns.
3. **rC** → Rows first, then columns in reverse order. Horizontal reflection  $f_h$ . Invert column order and invert row numbers.
4. **Rc** → Rows first in reverse order, then columns. Vertical reflection  $f_v$ . Invert row order and invert row numbers.
5. **cR** → Columns first, then rows in reverse order. Composed of  $f_h \circ g = g \circ f_v$ .
6. **Cr** → Columns first in reverse order, then rows. Composed of  $f_v \circ g = g \circ f_h$ .
7. **RC** → Rows first in reverse order, then columns in reverse order. Composed of horizontal and vertical reflection  $f_v \circ f_h = f_h \circ f_v$ .
8. **CR** → Columns first in reverse order, then rows in reverse order. Composed of  $f_v \circ f_h \circ g = f_h \circ f_v \circ g$ .

### 2.3. Algorithms

In this section, we describe the different approaches that we designed and implemented to solve nonograms. The first is a heuristic algorithm that we used as a baseline. We implemented a modification to introduce the neural network, as described in Section 2.1, as a support for the decision-making process and the reflections as an improvement. Furthermore, we introduced the concept of intuition, which means weighting the solution path by the similarity of the state of the board with the neural network proposal. We also describe the partial and full erase ideas that are used to accelerate the solution process. Finally, we propose a GA that also uses a neural network with the same objective, which is used in the state of the art. We were interested in evaluating their performance when we added a neural network to the decision process.

#### 2.3.1. Heuristic algorithm ( $H$ )

The first algorithm is composed of heuristic rules and a DFS. The heuristic is based on the concept of trivial cells, which can be filled using the information that is available from the puzzle headers and previous board state. We used the set rules found in [8] to design this algorithm.

Given a row or column, we establish the trivial cells by calculating all possible combinations given the header information. Subsequently, we obtain combinations that are compatible with the state of the row or column. At each step, we analyze the combinations that are compatible with the header to determine if there is a cell with the same value in all configurations to update the cell value. An example of this situation is shown in Figure 6. This process is repeated until the nonogram is solved or until it is not possible to obtain further trivial cells.

Once we fill all of the trivial cells, the second process, namely DFS, is implemented, whereby the path selection is determined by the number of possible combinations of the row or column in ascending order. At each search

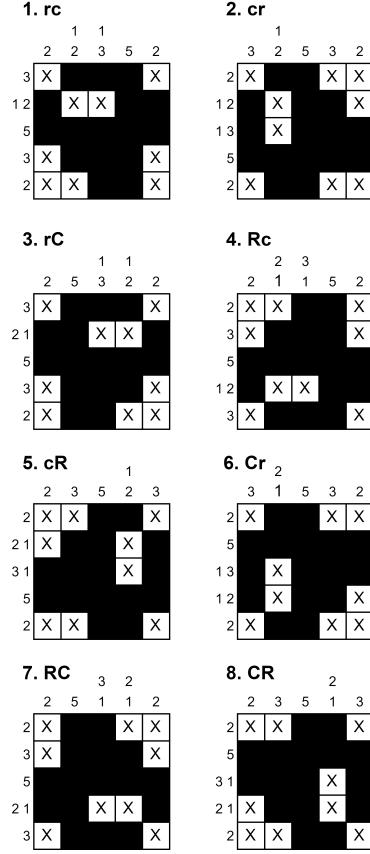


Figure 5: Visual description of the reflections on a  $5 \times 5$  board.



Figure 6: Example of the process that we followed to determine a trivial cell. To solve a row with header 3, there are three possible solutions, and the cell in the center is marked in all of them. We can conclude that this cell must be filled.

step, we repeat the trivial cell-filling process before performing a new search. The heuristic algorithm is summarized in algorithm 1 and code is available at <https://gitlab.com/DLG-05/musi-tfm-nonograma>.

---

**Algorithm 1** Heuristic (H)

---

```
1: function SOLVE(board)
2:   if Is_Full(board) and Is_Solution(board) then
3:     return True
4:   else if Is_Full(board) then
5:     return False
6:   end if
7:   comb, trivial_cells  $\leftarrow$  get_combinations(board)
8:   if trivial_cells  $\neq \emptyset$  then
9:     t2  $\leftarrow$  Fill_trivial_cells(board)
10:    return Solve(t2)
11:   else
12:     for c in Sorted(comb) do
13:       t2  $\leftarrow$  Apply_combination(c, board)
14:       if Solve(t2) then
15:         return True
16:       end if
17:     end for
18:   end if
19:   return False
20: end function
```

---

### 2.3.2. Network, heuristic algorithm and intuition (NeHI)

We modify the heuristic algorithm to integrate the neural network. This algorithm has two steps: First, we predict the board with the neural network and check if a correct solution is obtained

Second, if a solution is not obtained in the previous step, we start the DFS search using *intuition* with an empty board. This concept is applied at each step of the search, and we weight each path according to the similarity to the neural network prediction. This is the number of differences between the path and network prediction. The algorithm visits them according to their scores in ascending order; in the case of a tie, we select a path randomly.

These modifications are summarized in Algorithms 2 and 3, and code is available at <https://gitlab.com/DLG-05/musi-tfm-nonograma>.

---

**Algorithm 2** NeHI - Part 1

---

```
1: function MAIN(board)
2:   prediction  $\leftarrow$  Get_network_prediction(board)
3:   if Is_Solution(prediction) then
4:     return True
5:   else
6:     return Solve_AI(board)
7:   end if
8: end function
```

---

### 2.3.3. Heuristic algorithm using a network, with partial erase, full erase and intuition (NeHPF/NeHPFI)

We designed this third version of the heuristic algorithm to improve the results of the previous algorithm when network prediction is not a valid solution.

---

**Algorithm 3** NeHI - Part 2

---

```
1: function SOLVE_AI(board)
2:   if Is_Full(board) and Is_Solution(board) then
3:     return True
4:   else if Is_Full(board) then
5:     return False
6:   end if
7:   comb, trivial_cells  $\leftarrow$  get_combinations(board)
8:   if trivial_cells  $\neq \emptyset$  then
9:     t2  $\leftarrow$  Fill_trivial_cells(board)
10:    return Solve_AI(t2)
11:   else
12:     for c in Weighted(comb) do
13:       t2  $\leftarrow$  Apply_combination(c, board)
14:       if Solve_AI(t2) then
15:         return True
16:       end if
17:     end for
18:   end if
19: end function
```

---

Following the same procedure as that described in Section 2.3.2, we start by predicting the nonogram using a neural network. Then, if we do not obtain a valid solution, we analyze the network result to find the rows and columns that are not correctly predicted and those that are not consistent with the header, and we erase these. This process is known as partial erase. We then start the heuristic algorithm using the board state after the partial erase. The intuition is a hyperparameter of this algorithm.

Following the previous process, if we do not find a correct solution, we start the heuristic algorithm with an empty board without using the network. We refer to this process as full erase.

Using this new version of the algorithm, we accelerate the solution process for most nonograms. If the board that is predicted by the network contains few errors, the algorithm can reach a final state (solution or not) in a small number of iterations, thereby avoiding the performance of the entire heuristic process to obtain a valid solution. The pseudo-code of the algorithm is presented in 4 and code is available at <https://gitlab.com/DLG-05/musi-tfm-nonograma>.

#### 2.3.4. Heuristic algorithm using a network trained with all reflections (Ne8HI/Ne8HPF/Ne8HPFI)

Based on the preliminary experiments, the network could not solve all boards, but it could solve one of its eight reflections. We design this algorithm to improve the network prediction results.

This version includes the same algorithms as those in Sections 2.3.2 and 2.3.3. However, the networks are trained with the eight possible reflections of each nonogram of the training set, and if the network can solve any of the eight reflections, it can solve the proposed nonogram. Once one of the reflections is solved, the original board can be obtained by reversing it. It is important to note

---

**Algorithm 4** NeHPF

---

```
1: function MAIN(board, intuition)
2:   prediction  $\leftarrow$  Get_prediction(board)
3:   if Is_Solution(prediction) then
4:     return True
5:   else if intuition then
6:     t2  $\leftarrow$  Delete_wrong_rows_columns(prediction)
7:     if algorithm_AI(t2) then
8:       return True
9:     else
10:      return Solve_AI(board)
11:    end if
12:   else
13:     return Solve(board)
14:   end if
15: end function
```

---

that the eight reflections are bijective functions in the header and board spaces. Code available at <https://gitlab.com/DLG-05/musi-tfm-nonograma>.

### 2.3.5. Genetic Algorithm (GA)

Finally, we develop a GA to solve nonograms. The mutations consist of filling, emptying, or moving filled cells. A cell is filled if the number of filled cells in the current nonogram is smaller than that in the target nonogram. We empty a cell if there were more filled cells than those in the target nonogram. Finally, if the number of filled cells is equal to that in the target nonogram, the cell is moved. The number of mutations is determined randomly between 0 and 4, and the locations are randomly selected following a uniform distribution.

We generate the initial population of this algorithm by predicting a board using a neural network, and then we apply mutations to it. We define the following fitness function to evaluate each board:

$$\text{Fitness} = ncc + ncr + (\text{width} \cdot \text{height}) \cdot \frac{nmc}{ncmb} \quad (1)$$

where  $ncc$  denotes the number of correct columns,  $ncr$  denotes the number of correct rows,  $nmc$  denotes the number of marked cells, and  $ncmb$  denotes the number of cells that must be marked to complete the board. The fitness function is based on the concepts presented in [13].

The pseudo-code of the algorithm is presented in 5 and code is available at <https://gitlab.com/DLG-05/musi-tfm-nonograma>.

## 3. Experimental setup

In this section we describe the dataset, the experimental environment, the experiments and the metrics.

---

**Algorithm 5** GA

---

```
1: function GENETIC(board)
2:   while True do
3:     Total_weight  $\leftarrow$  get_total_weight()
4:     for i in range(len(individuals)) do
5:       w  $\leftarrow$  random_uniform(0, 1)  $\cdot$  Total_weight
6:       x  $\leftarrow$  get_individuals(w)
7:       y  $\leftarrow$  get_mutation(x)
8:       if Is_Solution(y) then
9:         return True
10:      end if
11:    end for
12:    change_population()
13:  end while
14: end function
```

---

### 3.1. Dataset

To train the neural networks, we required a large number of nonograms with their corresponding solutions. To the best of our knowledge, there is no public dataset that fulfills our needs; therefore, we generated our own dataset.

The dataset contained samples of boards of sizes  $5 \times 5$ ,  $10 \times 10$  and  $15 \times 15$ . We generated all possible boards of size  $5 \times 5$ ; that is,  $2^{5 \times 5} \simeq 3.36e07$  boards. It was not possible to generate all boards of sizes  $10 \times 10$  and  $15 \times 15$ , because the number thereof was very high:  $2^{10 \times 10} \simeq 1.27e30$  and  $2^{15 \times 15} \simeq 5.39e67$  boards. For these sizes, we employed the following strategy: First, we used the MAME Icons dataset [17] and Icons50 dataset [18], and applied the average image hash [19] to detect and remove duplicated images efficiently. Second, we scaled the images to the desired board size ( $10 \times 10$  and  $15 \times 15$  pixels). Third, we applied the following four transformations to generate the first set of nonograms: a Canny filter to detect edges, a binary threshold with a threshold value equal to 128, Otsu's threshold, and an inverted Otsu's threshold of the original image. An example of this transformation is depicted in Figure 7. After applying this process, a set of 76,368 boards was available for each size.

To complete the dataset, we generated a set of random nonograms because during several preliminary tests, we found that the size of the dataset was not sufficiently large to train the neural networks. We used two different techniques: we applied uniform noise between 0 and 1 to an empty board with a threshold of 0.5, and we drew a random number of simple geometric figures (lines, rectangles, and circles) with random sizes and positions on an image. One-third of this second dataset was generated by uniform noise and the remaining two-thirds by drawing simple figures. The dataset is available at: <https://github.com/josebambo/NonoDataset>.

Finally, our dataset was composed of 33,554,432 samples of boards of size  $5 \times 5$ , and after joining the two sets, we obtained a large dataset that was composed of 76,368 boards generated from images and 300,000 random boards of size  $10 \times 10$ , and 76,368 boards generated from images and 600,000 random boards of size  $15 \times 15$ .

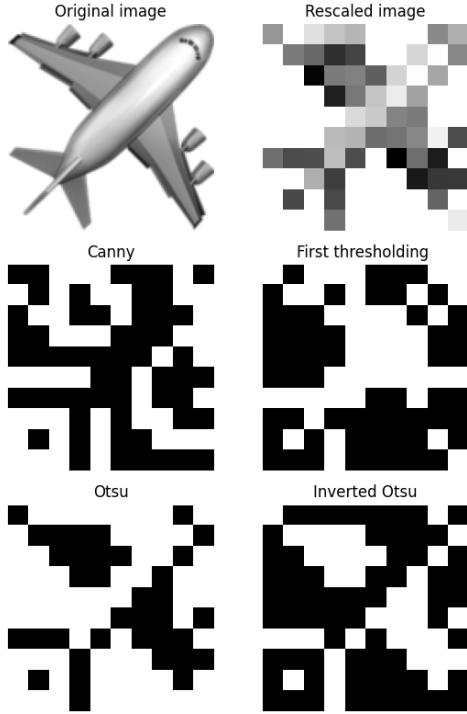


Figure 7: Examples of the different methods that we used to transform an image into a nonogram. The white color denotes a filled cell.

### 3.2. Experimental environment

We used TensorFlow and Keras on different machines with a NVIDIA K40, NVIDIA RTX2060, and NVIDIA RTX3090 to train the neural network models.

### 3.3. Experiments

#### 3.3.1. Experiment 1: Neural network training

The first part of the experiment consisted of evaluating the performance of the different neural networks that we designed using the dataset that we described previously. The results of this experiment were used to determine the best networks, which we used in the subsequent algorithms.

To perform this experiment, we designed a set of FC networks using three to eight hidden layers. The activation function for the intermediate layers was a ReLU and the function for the output layer was a sigmoid. We used the Adam [20] optimizer with different learning rates.

To train the neural networks, we used all boards of size  $5 \times 5$ , and we used the same dataset to evaluate the network performance. We performed this simple experiment to verify the capabilities of the designed networks. To create the

training and test sets for the boards of sizes  $10 \times 10$  and  $15 \times 15$ , we divided the data generated from the images into 80% for the training set and 20% for the test set. Subsequently, we added all nonograms generated by the random process to the training dataset.

Eventually, we had 33,554,432 boards for the training and test sets of boards of size  $5 \times 5$ ; 361,094 samples for the training set and 15,274 samples for the test set of boards of size  $10 \times 10$ ; and 661,094 boards for the training set and 15,274 for the test set of boards of size  $15 \times 15$ .

### 3.3.2. Experiment 2: Heuristic algorithms

We designed this experiment to evaluate the heuristic algorithm, which we used as a baseline, and its three variations that used a neural network. We tested them with and without reflections, and when enabling and disabling the intuition mechanism. Table 3.3.2 summarizes the different parts of the experiment.

For each board of size  $10 \times 10$ , we tested the algorithms described in Table 3.3.2 12 times using the validation dataset to obtain valid statistical measurements. For the boards of size  $15 \times 15$ , we used only a subset of 50 samples from the validation set.

Using this subset, we estimated the time required to solve the entire validation dataset.

Experiment	Acronym
Heuristic algorithm	H
Network, heuristic algorithm, and intuition	NeHI
Network with reflections, heuristic algorithm, and intuition	Ne8HI
Network, heuristic algorithm with partial and full erase	NeHPF
Network with reflections, heuristic algorithm with partial and full erase	Ne8HPF
Network, heuristic algorithm with partial and full erase, and intuition	NeHPFI
Network with reflections, heuristic algorithm with partial and full erase, and intuition	Ne8HPFI

Table 2: Experiments performed on the heuristic algorithm and its modifications. We tested all possible combinations of the reflections and intuition hyperparameters.

### 3.3.3. Experiment 3: Genetic Algorithm

Finally, we evaluated the GA with boards of size  $10 \times 10$  using the following set of parameters: 100 iterations and a population of 1000 individuals. We did not evaluate this algorithm with boards of size  $15 \times 15$  because a significant amount of time was required to obtain a result.

### 3.4. Metrics

From the early empirical results, we verified that the number of erroneous cells on the boards that were predicted by the tested neural networks followed a discrete Weibull distribution.

Given a neural network, let  $F$  be the random variable that represents the number of erroneous cells that are predicted on a board by a given network. Then, we have:

$$P\{F \leq f\} = 1 - e^{-(\frac{f+1}{\alpha})^\beta}, \quad (2)$$

where  $\alpha$  and  $\beta$  are the scale and shape parameters respectively, and  $f \in \{0, 1, 2, \dots, n \times n\}$  is the number of erroneous cells. We can compare two neural networks by comparing the parameters  $\alpha$  and  $\beta$  of the Weibull distribution that best fit the experimentation results.

The shape value ( $\beta$ ) of the neural networks with the best results was less than one, which implies that its mode was 0. Thus, from the neural networks with mode 0, we selected that with the smallest scale ( $\alpha$ ), which matched the neural network with a higher number of full correct boards (0 errors).

We used time-based metrics: the mean, standard deviation, and median of the time to solve a nonogram to evaluate the performance of the algorithms.

## 4. Results and discussion

In this section, we describe and discuss the results obtained from the three experiments.

### 4.1. Experiment 1: Neural networks

We correctly predicted 77.06% of boards of size  $5 \times 5$ . The best model architecture was composed of three layers of sizes 2048, 1024, and 256. After each layer, we added a dropout rate of 5% to avoid overfitting during the training phase, the learning factor was set to 0.001, and the error function was the binary cross-entropy as each cell can have only two values. The Weibull distribution parameters were as follows: shape ( $\beta$ ) = 0.5091745 and scale ( $\alpha$ ) = 0.1102024.

For boards of sizes  $10 \times 10$  and  $15 \times 15$ , the best model architecture was composed of five dense layers of sizes 2048, 1024, 1024, 1024, and 512. After each layer, we added a *dropout* of 5%. The learning factor was 0.0001 and the error function was *binary cross-entropy*. The Weibull parameters were as follows: shape ( $\beta$ ) = 0.420651 and scale ( $\alpha$ ) = 3.401227 for the  $10 \times 10$  boards, and shape ( $\beta$ ) = 0.6219632 and scale ( $\alpha$ ) = 18.54031 for the  $15 \times 15$  boards. By applying this model, we correctly predicted 27.25% of boards of size  $10 \times 10$  and 5.87% of boards of size  $15 \times 15$ .

The histogram of the number of errors on each board using the networks described above is depicted in Figure 8.

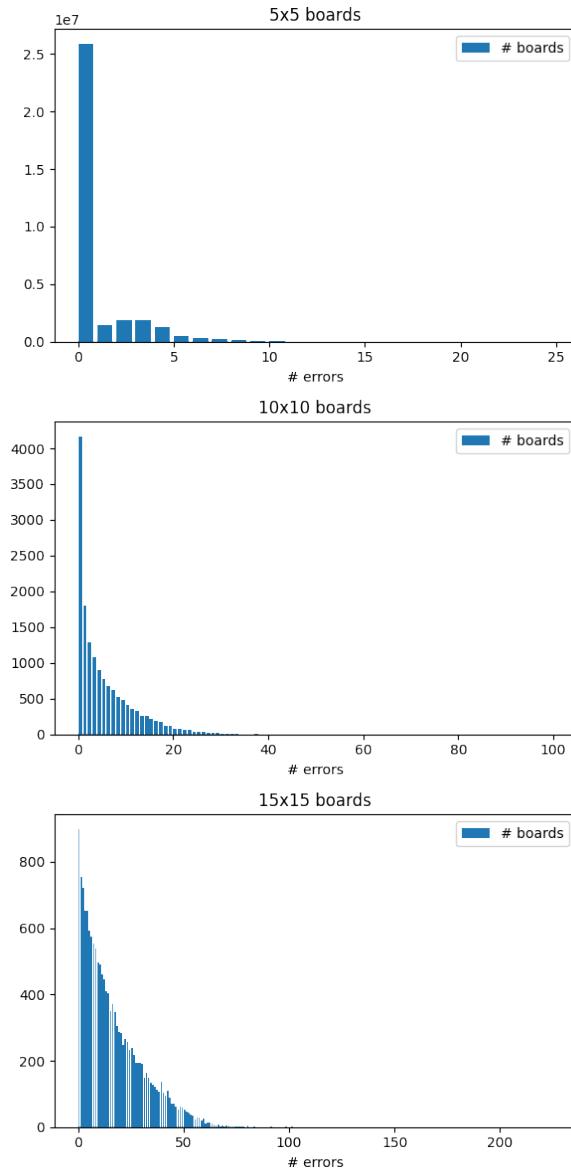


Figure 8: Histograms of number of errors of best network for each board size. We considered an error to have occurred when a position was incorrectly predicted.

To evaluate the performance of the neural networks using reflections, we calculated the number of correct predictions of boards of size  $10 \times 10$  when these were applied. The success rate increased from 27.25% to 39.26%. Figure

<b>Algorithm</b>	<b>Mean</b>	<b>Std</b>	<b>Median</b>
H	1.2003	3.4236	0.2042
NeHI	1.0428	3.1065	0.1526
Ne8HI	0.9811	3.0475	0.1003
NeHPF	0.7936	2.5936	0.0858
Ne8HPF	0.7630	2.6768	0.0520
NeHPFI	0.7891	2.6290	0.0904
Ne8HPFI	<b>0.7254</b>	<b>2.5853</b>	<b>0.0519</b>

Table 3: Statistical measures of the execution time, in seconds, of the algorithms with boards of size  $10 \times 10$ . Algorithms that contain an 8 in their name are the same algorithms as those in the line above, but they use reflections.

<b>Algorithm</b>	<b>Mean</b>	<b>Std</b>	<b>Median</b>
H	11.0972	14.3556	6
NeHI	9.2355	12.8242	6
Ne8HI	8.5279	12.7462	6
NeHPF	7.6012	12.8066	4
Ne8HPF	6.9063	12.9157	<b>3</b>
NeHPFI	7.4203	<b>12.6373</b>	4
Ne8HPFI	<b>6.6978</b>	12.7493	<b>3</b>

Table 4: Statistical measures of the number of iterations required to solve boards of size  $10 \times 10$ . Algorithms that contain an 8 in their name are the same algorithms as those in the line above, but they use reflections.

9 indicates an improvement in the network accuracy when we used reflections, according to the histogram of the number of errors in each board.

#### 4.2. Experiment 2: Heuristic algorithms

Once we obtained the results of the neural networks, we selected the best models for each board size and evaluated the performance of the proposed algorithms using boards of size  $10 \times 10$  and  $15 \times 15$ .

After executing all experiments summarized in Table 3.3.2 12 times, as a first result, all algorithms solved all boards. To analyze the results in detail, we present a statistical summary of the execution time in Table 4.2 and the number of iterations required to solve the boards in Table 4.2.

To analyze the time required for each variation of the heuristic algorithm in depth, Figure 10 depicts a histogram that describes the execution times of the algorithms during the first two seconds as they concentrated almost all information. Following the same concept, Table 4.2 presents the percentiles for each algorithm.

It can be observed that the use of a neural network implied that the minimum time to solve a nonogram increased (from 0.001 to 0.0045 s) when we compare

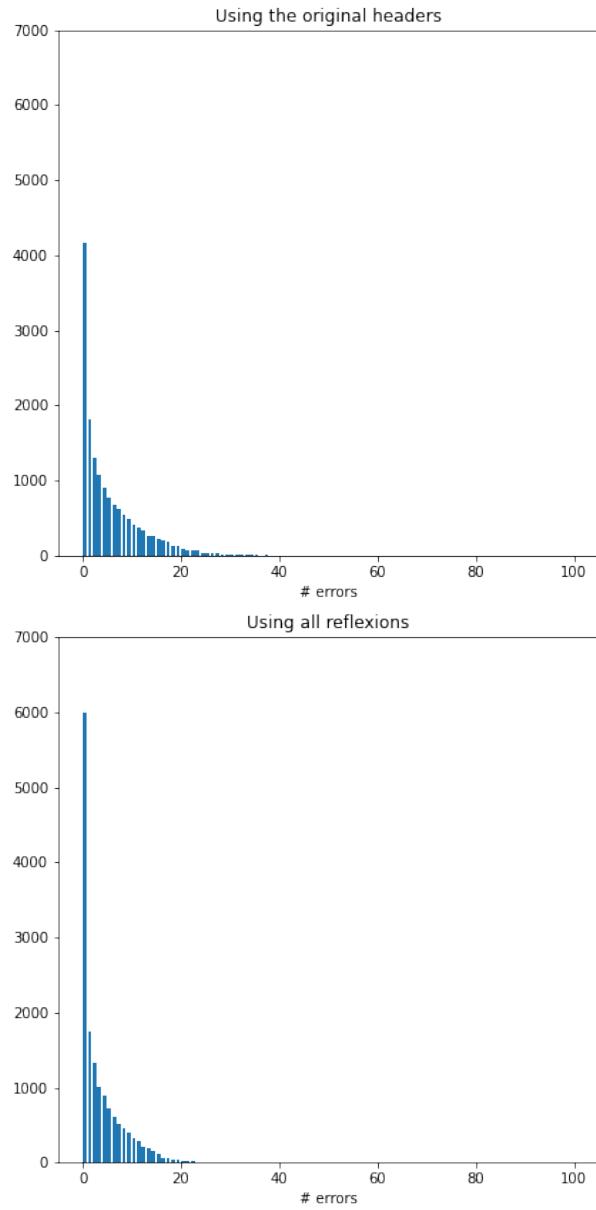


Figure 9: Comparison of the performance (number of incorrectly predicted cells) of the best network for boards of size  $10 \times 10$  when we used reflections. We can observe that the first bins in the histogram that correspond to the prediction of the networks with reflections are higher than those of the network without reflections, indicating better performance.

the solutions that used networks with the heuristic algorithm (H). However, this fact was irrelevant on more than 90% of the boards (percentile 0.1). According to the Ne8Hi, Ne8HPF, and Ne8HPFI columns for the algorithms that used a network and the reflections, solving a board using the reflections reduced the time to solve a nonogram in 70% of the cases (percentile 0.3). When we analyzed the time of the algorithms that used the partial erase technique, we observed that time improvements were achieved: comparing the NeHI and NeHPFI columns, starting at the 0.1 percentile, we obtained better results in 90% of the boards, and when observing the results of Ne8HI and Ne8PFI, we obtained better results in 60% of the boards.

Finally, we determined that applying the intuition technique without using reflections implied a time penalty (see NeHPF and NeHPFI columns). However, using reflections (see Ne8HPF and Ne8HPFI columns) slightly improved the results in all cases, except in the worst case, where the use of intuition provoked a penalty of 4 seconds.

P	H	NeHi	Ne8Hi	NeHPF	Ne8HPF	NeHPFI	Ne8HPFI
0.0	<b>0.0001</b>	0.0045	0.0060	0.0043	0.0062	0.0045	0.0060
0.1	0.0284	0.0050	0.0069	<b>0.0046</b>	0.0070	<b>0.0049</b>	0.0070
0.2	0.0533	<b>0.0052</b>	0.0071	<b>0.0047</b>	0.0072	<b>0.0050</b>	0.0072
0.3	0.0843	0.0334	<b>0.0073</b>	0.0182	<b>0.0074</b>	0.0189	<b>0.0074</b>
0.4	0.1299	0.0852	0.0230	0.0469	0.0139	0.0485	0.0137
0.5	0.2042	0.1526	0.1003	0.0858	0.0520	0.0904	0.0519
0.6	0.3298	0.2707	0.2187	0.1613	0.1123	0.1689	0.1123
0.7	0.5895	0.5182	0.4545	0.3043	0.2504	0.3169	0.2492
0.8	1.1843	1.0972	1.0360	0.6488	0.5551	0.6661	0.5529
0.9	2.9784	2.7175	2.6103	1.8512	1.7439	1.8789	1.6834
1.0	145.0626	151.3491	152.5022	52.0141	60.0945	66.7343	64.3024

Table 5: Percentiles of the execution time in seconds of the algorithms with boards of  $10 \times 10$ . The algorithms that contain an 8 in their name are the same algorithms as those in the line above, but they use reflections.

The time to solve a nonogram did not follow a normal distribution in any of the proposed methods; therefore, we could not compare them by comparing their mean times. Thus, we analyzed the data using the Wilcoxon nonparametric test. To perform this test, we used the paired data Wilcoxon test with the difference in time between the two algorithms. We defined a p-value threshold of 0.05 to assume that the methods that we compared were significantly different. Table 4.2 summarizes the results.

As a first test, we compared the heuristic algorithm (H) and the algorithm with a network and partial and full erase using intuition (NeHPFI), and obtained a p-value near 0, so we could assume that the NeHPFI algorithm was superior. Next, we compared NeHPFI and NeHPF, similar to NeHPF, without using intuition, and we concluded that in this case (p-value near 0), the use

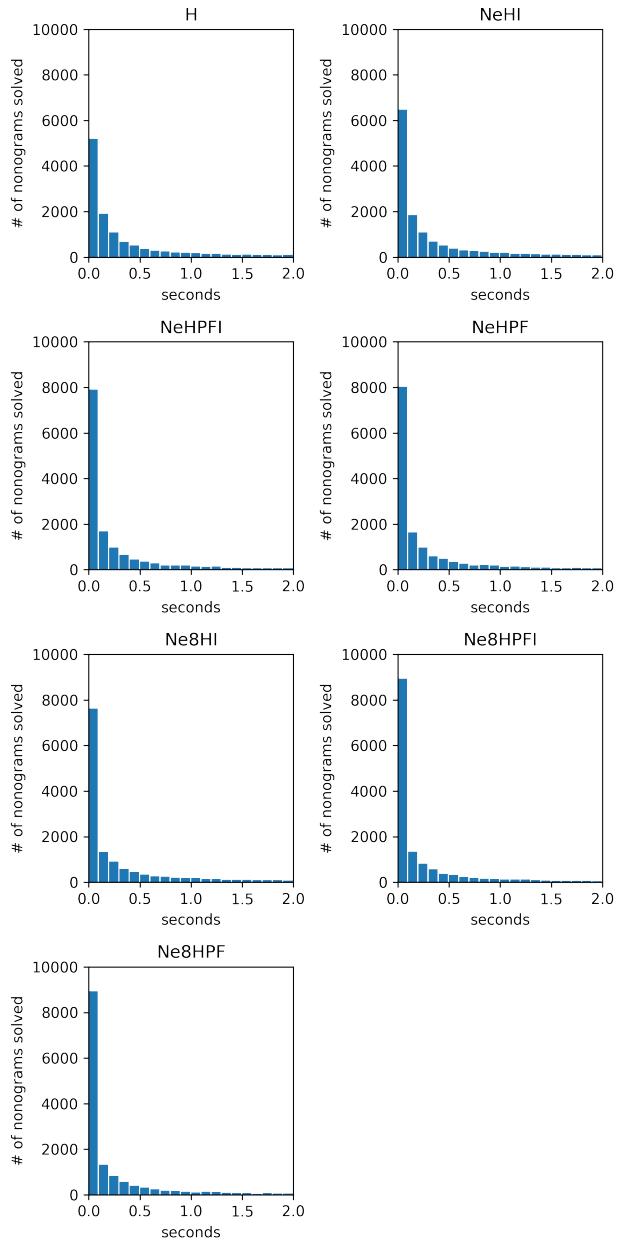


Figure 10: Histograms depicting the mean time required to solve a board of size  $10 \times 10$  using the different algorithms. We only show the first two seconds as these concentrated almost all information.

of intuition led to the worst results. We can observe that in all comparisons between the algorithms using predictions with and without reflections, respectively (Ne8HI vs. NeHI, Ne8HPFI vs. NeHPF, and Ne8HPF vs. NeHPF), the use of this technique led to better results. Finally, we compared the Ne8HPFI and NeHPF algorithms, the two best among the previous comparisons. As the p-value was near 0, we concluded that Ne8HPFI was the best algorithm, as it solved nonograms in less time than the other algorithms.

<b>Study</b>	<b>W</b>	<b>p-value</b>
H vs <b>NeHPFI</b>	18461386	$\approx 0$
NeHPFI vs <b>NeHPF</b>	98581813	$\approx 0$
<b>Ne8HI</b> vs NeHI	37076704	$\approx 0$
<b>Ne8HPFI</b> vs NeHPFI	41639500	$3.03e^{-206}$
<b>Ne8HPF</b> vs NeHPF	53285564	$1.09e^{-20}$
<b>Ne8HPFI</b> vs Ne8HPF	52985562	$5.48e^{-23}$

Table 6: Wilcoxon contrast using paired data, with best algorithm in bold. Algorithms that contain an 8 in their name are the same as those in the line above, but they use reflections.

Regarding the results of boards of size  $15 \times 15$ , Table 4.2 shows a summary of the execution of the 50 boards that we used as a test set with algorithms without reflections: H, NeHI, NeHPF, and NeHPFI. Table 4.2 outlines the total execution time for all 50 boards and an estimation of the time required to solve all boards using the entire validation set. The execution of the four algorithms required 2 days, 16 hours, and 46 minutes, and if this time was extrapolated to the execution of the entire dataset, the execution would require 2 years, 94 days, and 6 hours.

<b>Algorithm</b>	<b>Mean</b>	<b>Std</b>	<b>Median</b>
H	1313.6846	6535.3968	5.4766
NeHI	1669.1414	7212.9861	5.4736
NeHPFI	1147.2674	5955.4429	<b>2.8779</b>
NeHPF	<b>532.4148</b>	<b>1733.1702</b>	3.0873

Table 7: Statistical measurements of the execution time, in seconds, of the algorithms with boards of size  $15 \times 15$  on 50 images of the test set.

#### 4.3. Experiment 3: Genetic Algorithm

We executed the genetic algorithm (GA) once on the test set of boards of size  $10 \times 10$ . The execution required 2 days, 18 hours, and 20 minutes, and it only managed to solve 15% of the boards, without taking into account the boards that the network predicted correctly. Using the data obtained from this experiment, we can conclude that the GA yielded worse results than the heuristic algorithm,

<b>Algorithm</b>	<b>Total</b>	<b>Estimated</b>
H	18:14:45	232 days 05:40:20
NeHI	23:10:58	295 days 01:47:47
NeHPFI	15:56:04	202 days 19:36:03
<b>NeHPF</b>	<b>07:23:41</b>	94 days 02:55:04

Table 8: Total execution time on 50 images of the test set and estimated execution time for the entire test set of the heuristic algorithms with boards of size  $15 \times 15$ .

as it required a large amount of time to solve a nonogram and it did not have a guaranteed solution. This may be owing to the randomness of the algorithm or the fact that the *fitness* function was not sufficient for the dimensionality of the solution space.

#### 4.4. Results summary

From the results obtained regarding the number of iterations and time required for each algorithm, we can conclude that even if the neural network cannot solve all boards, it can provide a good starting point to solve a nonogram that saves considerable computation time. The combination of the heuristic algorithm and neural networks outperformed the heuristic algorithm, DFS, and GA.

### 5. Conclusions

In this study, we have presented an alternative to traditional algorithms designed to solve nonograms by adding neural networks to the decision process. Using the neural networks, it was possible to predict 27.25% of boards of size  $10 \times 10$  and 5.87% of boards of size  $15 \times 15$  correctly.

The best solution was the combination of the heuristic algorithm with a neural network using reflections with partial and full erase, and intuition (Ne8HPFI). The GA exhibited the worst performance for the proposed task because it could not solve the entire set of boards.

We also created a dataset <https://github.com/josebambu/NonoDataset>, which we have opened to the scientific community, to train and test the different networks and algorithms. The project code is available at <https://gitlab.com/DLG-05/musi-tfm-nonograma>.

#### 5.1. Further work

Despite testing different fully connected network architectures to determine which was the best for predicting nonograms, we would like to evaluate other types of networks or other nonogram representations.

Furthermore, we can design new *fitness* functions to determine whether the conclusions that we obtained in our experiment with the GA are useful for improving its results.

Likewise, for the variant of the algorithm that uses partial erase, a system could be proposed that would determine whether it is profitable to carry out partial deletion or to progress directly to complete deletion, thereby avoiding entering an invalid path and improving the overall system performance.

### Acknowledgments

This work has a grant by project EXPLAINable Artificial INtelligence systems for health and well-beING (EXPLAINING) (PID2019-104829RA-I00 / MCIN / AEI / 10.13039/501100011033).

### Declarations

**Conflict of Interests** The authors declare that they have no conflict of interest.  
**Data and Code Availability** The datasets and the code generated during and/or used during the current study are available in next url: <https://github.com/josebambu/NonoDataset>. The project code is available at <https://gitlab.com/DLG-05/musi-tfm-nonograma>

### References

- [1] N. Ueda, T. Nagao, NP-completeness Results for NONOGRAM via Parsimonious Reductions, Tech. rep. (1996).
- [2] J. V. Rijn, Playing Games: The complexity of Klondike, Mahjong, Nonograms and Animal Chess, 2012, pp. 33–41.  
URL <https://theses.liacs.nl/398>
- [3] J. Benton, R. Snow, N. Wallach, A combinatorial problem associated with nonograms, Linear algebra and its applications 412 (1) (2006) 30–38.
- [4] S. Salcedo-Sanz, E. G. Ortiz-Garcia, A. M. Pérez-Bellido, A. Portilla-Figueras, X. Yao, Solving japanese puzzles with heuristics, in: 2007 IEEE Symposium on Computational Intelligence and Games, IEEE, 2007, pp. 224–231.
- [5] K. J. Batenburg, W. A. Kosters, Solving nonograms by combining relaxations, Pattern Recognition 42 (8) (2009) 1672–1683.
- [6] K. A. Khan, Solving nonograms using integer programming without coloring, IEEE Transactions on Games (2020).
- [7] M. Q. Jing, C. H. Yu, H. L. Lee, L. H. Chen, Solving japanese puzzles with logical rules and depth first search algorithm, Proceedings of the 2009 International Conference on Machine Learning and Cybernetics 5 (July) (2009) 2962–2967. doi:10.1109/ICMLC.2009.5212614.

- [8] C. H. Yu, H. L. Lee, L. H. Chen, An efficient algorithm for solving nonograms, *Applied Intelligence* 35 (1) (2011) 18–31. doi:10.1007/s10489-009-0200-0.
- [9] D. Stefani, A. Aribowo, K. V. I. Saputra, S. Lukas, Solving pixel puzzle using rule-based techniques and best first search, in: International Conference on Engineering and Technology Development (ICETD), 2012.
- [10] J. Więckowski, A. Shekhovtsov, Algorithms effectiveness comparison in solving nonogram boards, *Procedia Computer Science* 192 (2021) 1885–1893.
- [11] J.-T. Tsai, Solving japanese nonograms by taguchi-based genetic algorithm, *Applied Intelligence* 37 (3) (2012) 405–419.
- [12] A. Bobko, T. Grzywacz, Solving nonograms using genetic algorithms, in: 2016 17th International Conference Computational Problems of Electrical Engineering (CPEE), IEEE, 2016, pp. 1–4.
- [13] R. Soto, B. Crawford, C. Galleguillos, E. Olguín, Solving nonogram using genetic algorithms, in: 2016 11th Iberian Conference on Information Systems and Technologies (CISTI), 2016, pp. 1–4. doi:10.1109/CISTI.2016.7521597.  
URL <https://ieeexplore.ieee.org/document/7521597>
- [14] H. Alkhraisat, H. Rashaideh, Dynamic Inertia Weight Particle Swarm Optimization for Solving Nonogram Puzzles, *International Journal of Advanced Computer Science and Applications* 7 (10) (2016) 277–280. doi:10.14569/ijacsa.2016.071037.
- [15] Y.-C. Chen, S.-S. Lin, A fast nonogram solver that won the taai 2017 and icga 2018 tournaments, *ICGA Journal* 41 (1) (2019) 2–14.
- [16] R. Thomas, F. Dandurand, Q. C. H. V. Canada, J. J. Lussier, Solving nonogram puzzles by reinforcement learning, *Proceedings of the Annual Meeting of the Cognitive Science Society* (2012).  
URL <https://escholarship.org/uc/item/13t021xk>
- [17] A. Paradossi, MAME MAMu's Icons, <https://www.progettosnaps.net/icons/>.
- [18] D. Hendrycks, T. Dietterich, Benchmarking Neural Network Robustness to Common Corruptions and Perturbations, *Proceedings of the International Conference on Learning Representations* (2019).
- [19] N. Krawetz, Looks Like It, <http://www.hackerfactor.com/blog/index.php?archives/432-Looks-Like-It.html>, (Última consulta 27/06/2021) (2011).
- [20] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).