American University of Armenia

College of Science and Engineering

Artificial Intelligence Project

Hakob Janesian, Gagik Khalafyan, Apig Aramian

Summer, 2022

Contents

Abstract

Nonograms are very popular Japanese logic puzzles, whose name varies from country to country, (Hanjie, Paint by Numbers, Griddlers). The goal is to fill cells of a grid in a way that contiguous
blocks satisfy the constraints of placement stated in the problem[1].

In this report Nonogram puzzles are considered as a Constraint Satisfaction Problem (CSP), which resulted in more thorough research on the topic of CSP. In addition, other solutions differing from the given approach are going to be reviewed in this paper. Next, the problem formulation and solving methods are also given. After all, the results of the solved puzzles are presented with the corresponding runtimes.

Keywords: nongram, integer linear programming, genetic algorithms, depth first search, constraint satisfaction problem, inference, backtracking, forward-checking.

Introduction

In 2012 two Japanese professors from the Tokyo Institute of Technology, Tadaaki Nagao and Nobuhisa Ueda proved that solving a nonogram puzzle is an NP-complete problem[2]. In order to solve this puzzle, the cells in the matrix should either be colored black or white, and the decision should be made based on a description that shows the lengths of the successive black segments for each column and row. Additionally, one of the constraints is that there must be at least one white cell between each pair of consecutive black segments. The hidden image will become visible after the puzzle has been finished and completed. Thus, several examples of uncolored nonograms were sampled, and the primary objective of the project was to solve them.

Structure of project paper: the paper consists of three main sections. First section contains literature reviews of other existing solutions for nonograms. The next section contains the description of the main algorithm used in this project to solve the puzzles. And the last section overviews the achieved results. Finally, further work is discussed and bibliography of the used literature is provided.

Literature review

The most well known approaches to solving uncolored nonograms are the iterative search, integer linear programming (ILP), genetic algorithm (GA), and depth first search.

1) Depth first search[3]

When using a depth first search to solve a black and white nonogram, all feasible options for the set of blocks on each line are tried. The most crucial step in performing a depth first search is conducting an analysis of all the different possibilities present in a specific row or column, or possibly the entire image, and then using that information to decide which cells need to be colored black and which cells cannot be colored black. It is crucial to make every effort to determine as many cell values as possible by logical analysis, although this may not always be achievable. In the case when the algorithm is stuck, it can be speculated on particular cell values and back away from our choice if it causes a contradiction. The runtime complexity of such a solution for uncolored nonograms is O(N * M * 2^(N * M)). Although the evaluation function and the mechanism used to generate the state space during the depth first search are both high speed, the overall performance of the approach is subpar. The reason for this is a large number of states that must be verified.

2) Integer linear programming (ILP) and iterative search algorithm[1]

In 2009 Luís Mingote and Francisco Azevedo, who are Portuguese Ph.D. professors at the NOVA University of Lisbon, published their work ‘Solving Colored Nonograms.’ In that research paper, it is mentioned that in 2001 Robert A. Bosch, an American recreational mathematician, author, and professor of mathematics at Oberlin Collegiate Institute, designed the integer linear programming (ILP) method specifically for black and white nonograms. Thus he offered a new universal solution for the nonogram puzzles. So L. Mingote and F. Azevedo proposed the idea of a hybrid model between well performing iterative approaches and integer linear programming. The iterative approach was effective and practical in rapidly filling many grid cells using basic inferences on the rows and columns cues. So professors hoped that combining the integer linear programming model with the iterative approach might speed up the process and use the new theoretical conclusions. To create this, the professors recommended using the ILP only after many cells had been filled using the iterative method, therefore greatly simplifying the model by replacing a large number of variables with constants.

3) Genetic algorithm (GA)[4]

In 2004, Professor Wouter Wiggers from the University of Twente developed a novel genetic algorithm (GA) for solving nonograms. The algorithm was excessively slow, and there was a strong chance of being trapped in local optimums while it was being executed. The three main operators of a genetic algorithm are mutation, selection, and crossover. The genetic algorithm performs an evaluation each time the fitness function calculates the fitness of a chromosome. Wiggers has conducted a study into a variety of scenarios for the crossover and mutation rate of the genetic algorithm. The situation in which he obtained the best outcome was one in which the crossover rate was 60%, and the mutation rate was 5%. He is of the opinion that a mutation rate of 5% is relatively high for a genetic algorithm, but this was the only option to prevent being stuck in the local optima of the nonogram. In the genetic algorithm, he chose to work with a population size of 100 chromosomes since the algorithm's performance did not noticeably change if the population size was more than 100 chromosomes. Because the genetic algorithm is stochastic, the performance was evaluated by taking the average of the results from the first three times it was successfully run. Thus, the genetic algorithm outperforms the depth first search on larger, uncolored nonograms (for example, a puzzle with the size of 10x10). Contrarily, depth first search performs better than the genetic algorithm when it comes to small nonograms. This is due to the fact that a genetic algorithm begins with a population that has 100 chromosomes and that the algorithm first processes each member of the population before determining whether or not a solution has been discovered. Therefore, the article by W. Wigger demonstrates that a genetic algorithm may perform quite well compared to an algorithm that is more robust and predictable. It is fascinating to see how quickly genetic algorithms may generate a solution that is almost correct.

Numerics

Background

Our group did the coding part of the project in Python. We used famous python libraries such as Pandas, Numpy, and Matplotlib. We solved the nonograms as a constraint satisfaction problem (CSP). The variables are the descriptive cells from the left panel of the grid showing the sizes of the required blocks that should be positioned in the rows. The first part of the constraints is formulated from the same panel to place the blocks and fit them in a row correctly, leaving enough space for the other blocks in the same row. Next, the second part of the constraints is formulated from the top panel again for correctly positioning the blocks in the columns. The domain of each variable is going to include all the possible starting positions for the block it represents. Initially, all the row`s cells of the corresponding variable are included in its domain, but later the domain size is reduced by inference, and then the final algorithm is applied.

Solution

The first step of the algorithm removes the values from the domains which violate the row constraints. The first values which will be removed are the ones that are not able to simply be fitted in the row. The positioning of the rest of the variables in the same row is also considered to fit all of them. Thus, we get all the permissible values for each variable to allow the rest of the blocks to fit in the row. Next, the backtracking algorithm begins, and forward checking is applied to make the searching process faster for each value assignment.

Algorithm

As we stated, the first step is the enforcement of the row constraints. To do this, firstly, we remove the values from each variable`s domain, which leads the block to get out of the grid. After that, a traversal is applied on both ends of the row, which helps us to understand the number of blocks needed, which will be placed before and after the currently observed variable.

Next, their lengths are summed for each direction each time, also adding 1 for neighboring blocks for keeping the gaps, and in this way, the invalid cases when other blocks can not be fitted are eliminated. Thus, considering the fact that some cutting was already applied from the right side to fit the variable itself, after the aforementioned calculations, the domain is again being cut from left and right by the recently obtained value.

At the point when no other values can be eliminated from the domain, the backtracking algorithm starts. The first recursive call is done with the top-left variable from the left panel, and in this way, variables are picked from left to right, row by row, until reaching the variable located at the bottom right corner.

During all calls, consistency checks are applied for current assignments to ensure that constraints defined by column indicators are not violated. Consequently, the searching process continues. Because of starting from the top row, partially constructed blocks are quickly identified, and backtracking is applied sooner instead of going too deep; thus, this method is beneficial in light of speed. In addition, following the assignment of each value, the domains of the other variables in the row are updated with the assistance of forward checking. This avoids the possibility that two blocks will cover the same cell in the output. Finally, after going through the process of backtracking, an assignment that is complete and consistent is provided for all of the variables; alternatively, all of the variables are left unassigned, which means that the given nonogram problem does not have a solution.

Results

The algorithm was tested for puzzles of different sizes, and the correct solution was found for all, with only differing execution times. Following crosswords with corresponding running times were tested:

1) Name of the nonogram: 'The initial sample.'

Size: 4 x 6,

Number of variables: 7,

Execution time: 0.1 seconds

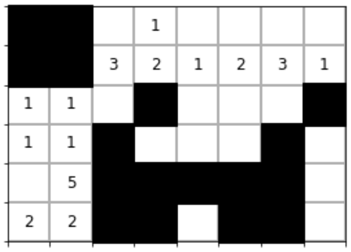

Figure 1. 'The initial sample' nonogram

2) Name of the nonogram: 'The face',

Size: 10 x 10,

Number of variables: 23,

Execution time: 0.5 seconds

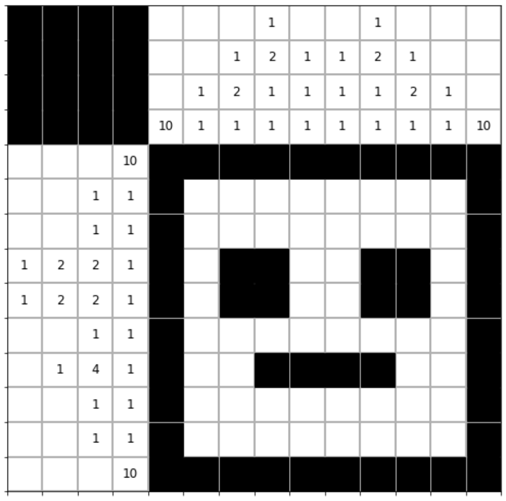

Figure 2. 'The face' nonogram

3) Name of the nonogram: 'Sun',

Size: 15 x 15,

Number of variables: 40,

Execution time: 0.8 seconds

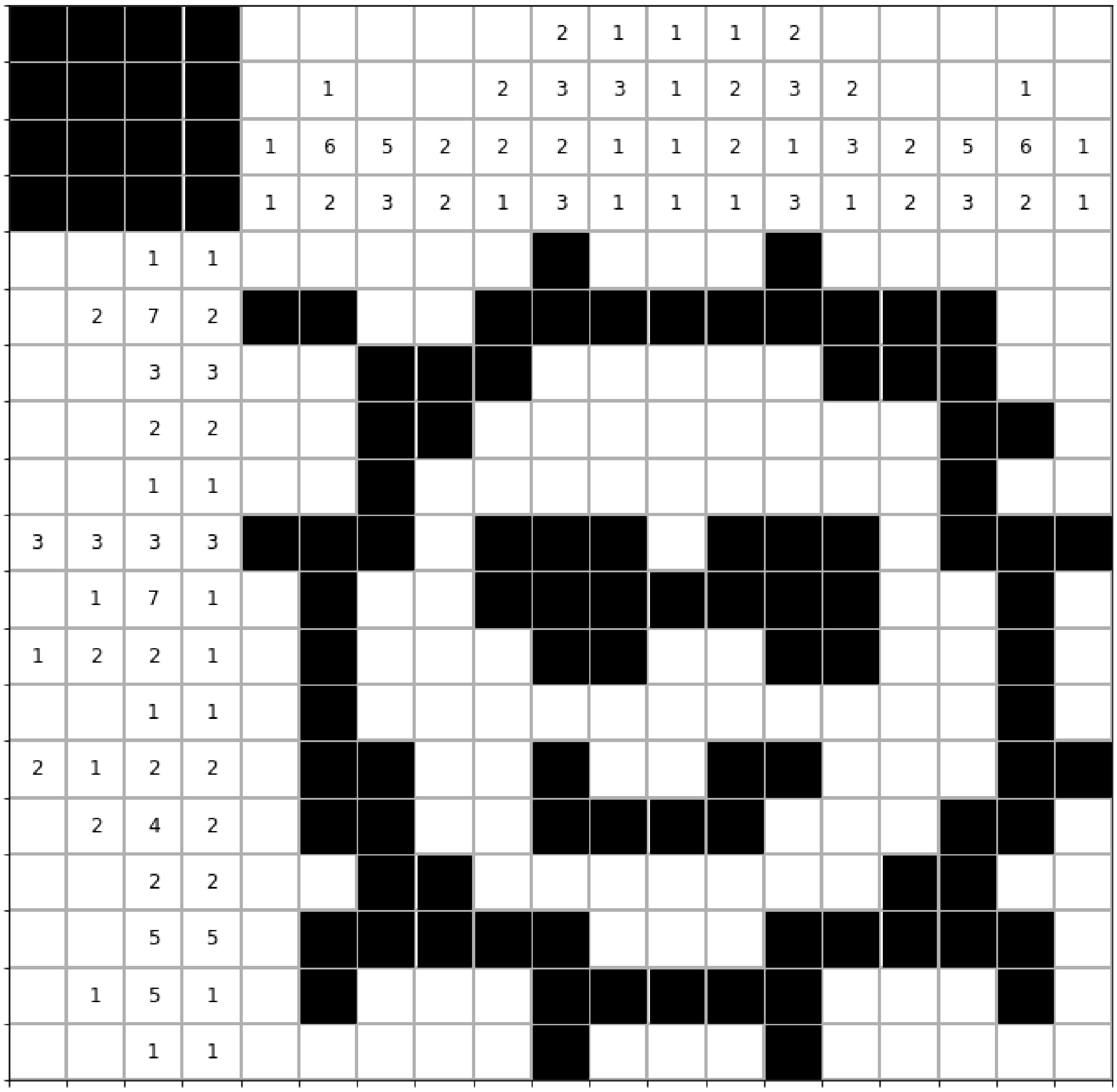

Figure 3. 'Sun' nonogram

4) Name of the nonogram: 'Car',

Size: 28 x 27,

Number of variables: 105,

Execution time: 5.8 seconds

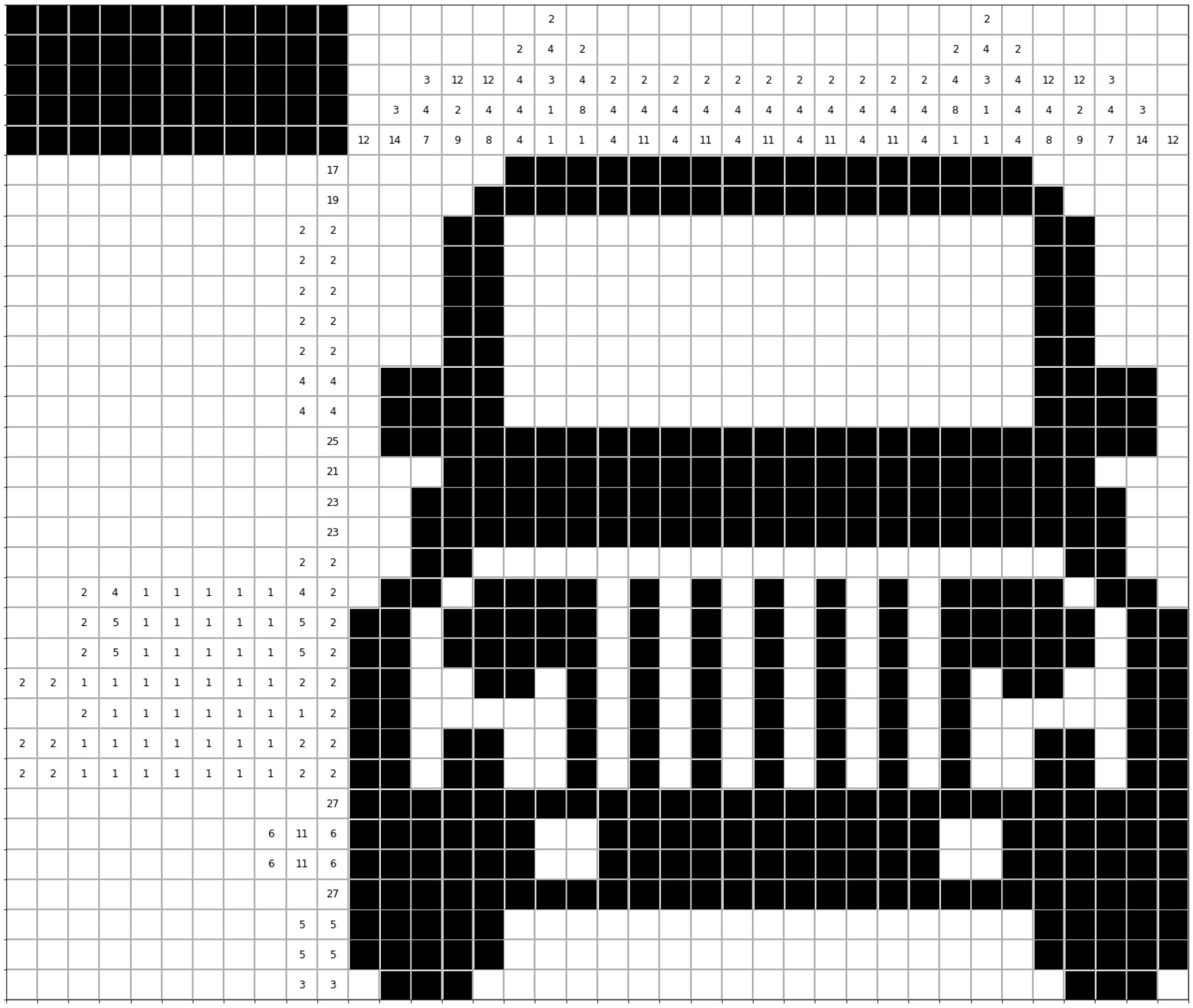

Figure 4. 'Car' nonogram

5) Name of the nonogram: 'Kettle',

Size: 30 x 35,

Number of variables: 77,

Execution time: 1.2 minutes

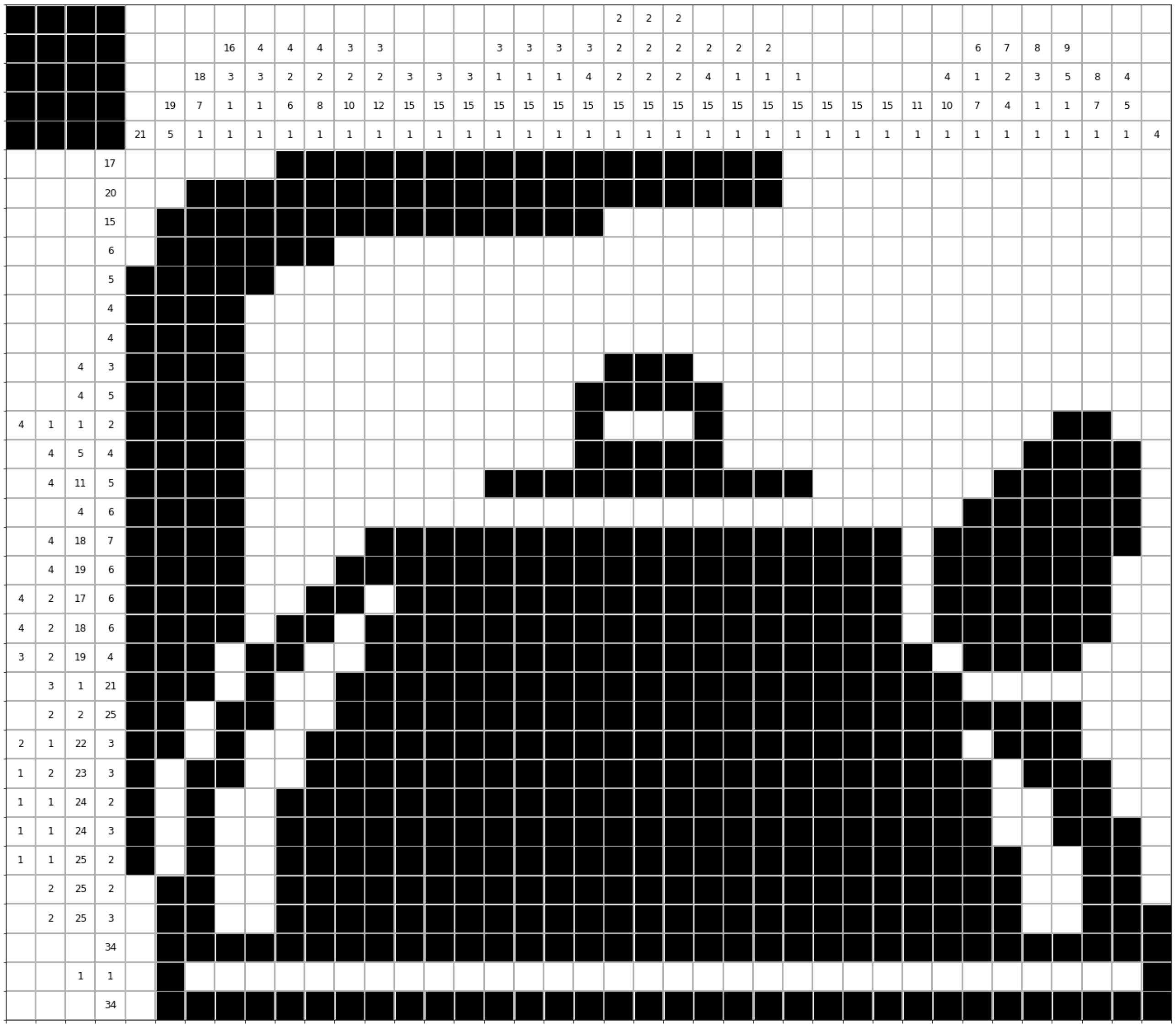

Figure 5. 'Kettle' nonogram

Conclusion

It is clear that small crosswords are being solved in milliseconds, but as expected, when the size of the crossword grows, more time is required to solve it. The most crucial thing is that the algorithm is able to solve any configuration of the crossword. The increase in execution time for bigger crosswords could also be acceptable unless we observed the difference between the execution times of the fourth and fifth crosswords. The fifth nonogram contains fewer variables and has a slightly greater size compared to the fourth nonogram, but there is a huge gap between their execution times. The fifth nonogram`s execution time is 70 seconds, and the fourth`s executes in 5.8 seconds. Then the difference is 64.2 seconds. Thus it can be deduced that execution time is the only issue, and still, some optimization is required for the algorithm to become more efficient during the inference and search process. Thus, the main goal is accomplished, and the algorithm is able to solve uncolored nonograms of any size and difficulty.

Further work

We saw that the algorithm is fully proficient in light of finding the correct solution to the puzzle, and the only work to be done further is the reduction of the time complexity of the algorithm that we used to solve the nonograms. In the future, we can consider some modifications of backtracking algorithms. For instance, backtracking algorithms may be modified and given heuristics to solve various nonograms more efficiently. One excellent method that lessens the undesirable tendency of backtracking in order to rediscover the same dead end is the backjumping. Avoiding the same dead ends by constantly instantiating the preceding variable may be made more efficient by firstly identifying the problem variable and then immediately reinstantiating it. Conflict sets are the basis for backtracking's method of identifying a culprit variable. In addition to that, there is Gaschnig's backjumping. This method of backjumping involves recording some information while the next step in the backtracking process is being generated. This information is then used to establish which variable caused the dead end. An approach to marking is utilized by the algorithm, in which each variable keeps a reference of the most recent predecessor whose values were discovered to be incompatible with any of the variable's values. Thus the aforementioned modifications help to solve some nonograms more efficiently.

Bibliography

[1] Luís Pedro Canas Ferreira Mingote, ‘Solving Colored Nonograms’, 2009, https://run.unl.pt/bitstream/10362/2388/1/Mingote_2009.pdf

[2] Nobuhisa Ueda and Tadaaki Nagao, ‘NP-completeness Results for NONOGRAM via Parsimonious Reductions’, December 2002, https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.5277&rep=rep1&type=pdf

[3] R.A. Oosterman, Prof.dr. J. Top, Prof. dr. W.H. Hesselink ‘Complexity and solvability of Nonogram puzzles’, April 2017, https://fse.studenttheses.ub.rug.nl/15287/1/Master_Educatie_2017_RAOosterman.pdf

[4] Wouter Wiggers, ‘A comparison of a genetic algorithm and a depth first search algorithm applied to Japanese nonograms’, 2004, https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.62.9443&rep=rep1&type=pdf