Solving Nonogram Using Genetic Algorithms

林揚益
National Yang Ming Chiao Tung University
Hsinchu, ROC
yylin233@gmail.com

劉沛凡
National Yang Ming Chiao Tung University
Hsinchu, ROC
ben900926.cs09@nycu.edu.tw

陳清海
National Yang Ming Chiao Tung University
Hsinchu, ROC
tecs1003@gmail.com

温柏萱
National Yang Ming Chiao Tung University
Hsinchu, ROC
alison.cs09@nycu.edu.tw

ABSTRACT

The Japanese Nonogram puzzle is a well-known NP-complete problem and falls under constrained combinatorial optimization. Genetic algorithms are a common approach to such problems, but they often result in vast search spaces and provide imprecise guidance toward solutions. We conducted experiments with two genetic algorithms to compare and analyze their efficiency and ability to converge to a correct solution.

CCS CONCEPTS

• **Computing methodologies → Optimization algorithms**.

KEYWORDS

Genetic Algorithms, Japanese nonograms

ACM Reference Format:
林揚益, 劉沛凡, 陳清海, and 温柏萱. 2018. Solving Nonogram Using Genetic Algorithms. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

A Nonogram is a logic puzzle featuring a grid with numbered clues along its rows and columns. These clues indicate the number of groups of consecutive filled squares and the size of each group in a specific row or column. The goal is to determine the placement of filled and empty squares, revealing a hidden picture or pattern within the grid. The challenge involves correctly interpreting these clues and filling the grid such that all clues are satisfied simultaneously, without contradictions[9].

Logic reasoning may be highly efficient for solving small-scale Japanese nonograms [5, 9], but scaling it up for larger puzzles is challenging. Therefore, the Japanese nonogram problem, a popular NP-complete problem, is often targeted by genetic algorithms [12, 11], known for solving combinatorial, design, and optimization problems [3, 8, 6, 4]. Numerous studies have attempted to resolve this issue [12, 1, 2].

An important principle in solving Japanese nonogram problems is avoiding guesses, as a single error can cascade throughout the entire grid and compromise a potentially viable solution.

In our work, we employ two methods: the vanilla method, designed as a baseline, and the Intelligent Genetic Algorithm (IGA) method, adapted from [10] with the encoding method from [9].

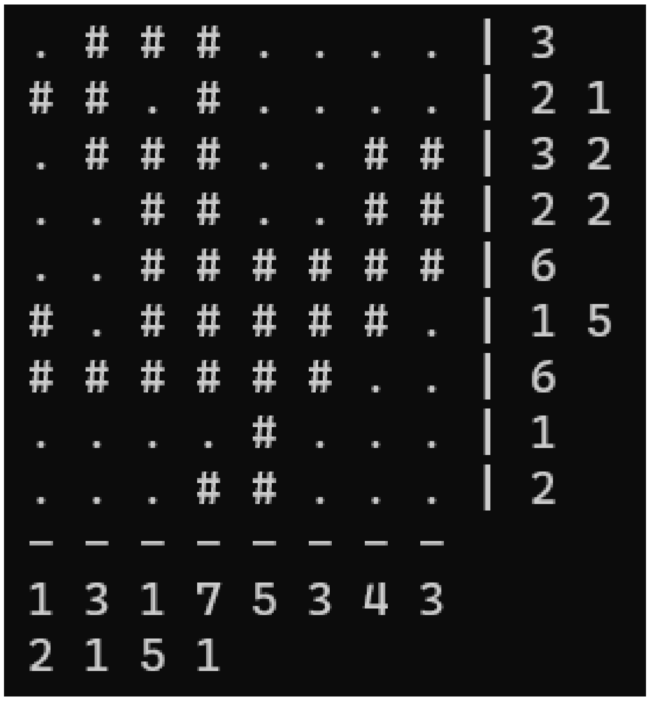
In the vanilla method, we use a grid representation for the genotype and define the fitness function by summarizing the grid points that meet the constraints, adding penalties for any violations. In the IGA method, we adopt the effective condense encoding from [9], which is considered a constraint-fitting representation. Such genotype ensures the chromosome to be a feasible solution in all rows or columns. The customized fitness function takes the order of filled grids into account, which is better at identifying the quality of potential solutions. The adoption of these new representations and fitness functions, which maintain validity after mutations and crossovers, has shown success in the results. Also, increasing the number of generations had also contributed to the success in evolving to a fine solution.

2 METHODS

2.1 Vanilla Method

The vanilla method serves as a baseline to compare with other results, thus we adopted the simplest, most straight forward methods for each components in genetic algorithms. We use the grid representation as its genotype representation, the chromosome is a potential solution grid. For selection, tournament selection is adopted with bitflip mutation and two point crossover. The setting can be summarized as the following:

Genotype	Binary grid representation
Chromosome	A potential solution grid
Selection	Tournament selection
Mutation	Bit-flip mutation
Crossover	Two-point crossover
Fitness	Add points if the constraints are met, add penalty if the constraints are violated.



```
. # # # . . . . | 3
# # . # . . . . | 2 1
. # # # . . # # | 3 2
. . # # . . # # | 2 2
. . # # # # # # | 6
# . # # # # # . | 1 5
# # # # # # . . | 6
. . . . # . . . | 1
. . . # # . . . | 2
- - - - - - - -

1 3 1 7 5 3 4 3
2 1 5 1
```

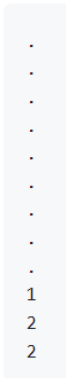
Figure 1: Example board configuration

2.2 Intelligent Genetic Algorithm

To represent a Nonogram board configuration, a simple approach is to use a "0" bit for a blank cell and a "1" bit for an occupied cell. However, this method has been deemed impractical for performing genetic algorithms as seen in the baseline. Crossover and mutation operations are likely to compromise the solution's validity. The original naive method fails to ensure validity, thereby diminishing its effectiveness as a genetic algorithm.

To address this issue, we explored an approach beyond the naive genetic solution for representing each Nonogram candidate board configuration. We referred to a column-order genetic solution proposed by Manlio Morini [7], which offers a more effective way to maintain the integrity of the solutions throughout the genetic process.

2.2.1 Genome. The pattern can be encoded in the column-order based on the amount of given column clues. For instance, the example board configuration shown as figure 1, can be encoded as: {1, 2, 0, 2, 0, 0, 0, 0, 4, 4, 2, 2}. Each number in the sequence represents the minimal distance to the first valid position. Take column 1 as an example, there are two clues in the column: 1 and 2. The first clue "1" indicates the first block has one unit of distance to the first valid position (i.e. row 1 column 1), so it is located in row 2; and the second clue "2" indicates the second block has two unit of distance to the next valid position (since the first block has taken row 2, the next position will be row 4), so it is located in row 6. Formally, we encode the board as a sequence of k integers ranging from 0 to the total number of rows, where k is the total amount of column clues. Notice that it is possible to produce individuals with invalid board configuration. Therefore, in order to eliminate invalid individuals,



```
.
.
.
.
.
.
.
.
.
1
2
2
```

Figure 2: Example Column

we need to perform a simple mapping. In the tutorial, the author provided modulo-based mapping to solve this problem efficiently. The details will be covered in the decoding 2.2.2. This can create a separation between the genotype space (list of integers) and the phenotype (the real positions of the blocks).

2.2.2 Decoding the genotype. For each block, we have three attributes to be taken into consideration:

(1) the first allowed starting position which depends on already placed spots (start)
(2) the number of allowed positions considering the already placed blocks and the remaining blocks (allowed)
(3) where to placed the current block (placed): placed = start + encoded distance % allowed.

Based on the example in [7], let's consider an example column 2 with the genomes $1, 7, 6$.

First, consider the block with the clue "1". We should assume that the remaining blocks are placed as far apart as possible. In this case, the "start" will be at row 0, and the allowed positions for block 1 will include the range from row 1 to 3 (considering that block 2 occupies rows 5 to 6 and block 3 occupies rows 8 to 9). Therefore, the position for block 1 is calculated as $0 + 1\%3 = 1$, corresponding to row 2.

Next, consider the second block with the clue "7". Since block 1 is placed in row 2, the next available starting row is row 4 (index 3). Using similar reasoning, the allowed positions for block 2 become 2. Thus, the position for block 2 is calculated as: $3 + 7\%2 = 4$. Using this encoding has the following advantages:

(1) Allow the possibility of swapping row and column clues
(2) Preserve validity after crossover and mutation operations
(3) Few numbers needed to represent the whole board
(4) Reduce the search space of our solution

2.2.3 Fitness Function. Observing the aforementioned cases and rules, it is evident that each representation can be correlated with a valid configuration. The performance is contingent on the row clues and, as such, necessitates evaluation through a customized fitness function.

This fitness function assesses performance by calculating the delta value between the row clues of each candidate and those in the answer configuration. The formula for the fitness function can

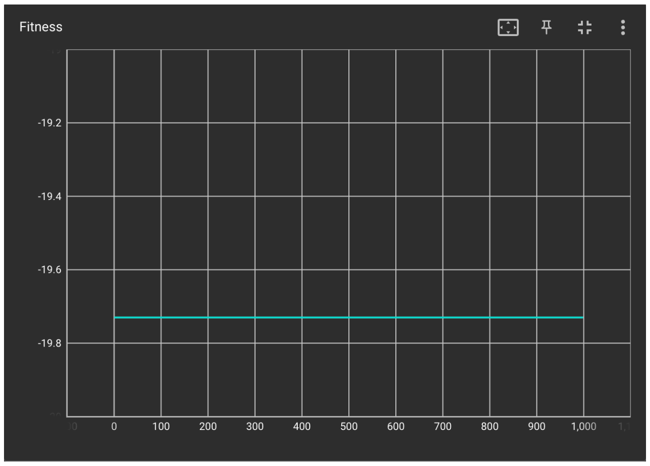


Figure 3: Baseline Fitness

be expressed as follows:

$$Fitness = row\_count * col\_count - \sum_{i}^{n} |R_i - r_i|$$

Where R_i represents each row clue component in the answer configuration, and r_i represents that in the candidate configuration.

For instance, consider a candidate configuration where row 1 has a block occupying 3 spots (row clue: [3]), while in the answer configuration, the row clue for row 1 is [3, 1]. In this case, the total delta value for row 1 would be $|3 - 3| + |0 - 1| = 1$.

Since the genetic algorithm seeks a higher value, the delta is subtracted from a positive value.

3 EXPERIMENTS

3.1 Vanilla Method

For the baseline experiment, a 10×10 grid solution is randomly generated, with 10% of the cells being filled. The specific configuration is as follows:

3.2 Intelligent Genetic Algorithm

In this experiment, the challenge presented in 1 is addressed. An intelligent genetic representation, coupled with a corresponding fitness function, is employed. To better align the genetic process with the genomes, modified crossover and mutation operations are introduced. The selection method utilized in this section is roulette selection.

The crossover operation is tailored to the specificities of column clues. The probability value determines the occurrence of this operation. Contrary to the conventional method of selecting a random single point for crossover, our revised approach exchanges entire row components without segmentation. For instance, consider the encoded representation [1,2], [3], [4] with three column clues, where the first column comprises two clues. A typical single-point crossover might split these two clues if the random point is 1. In contrast, our adapted crossover only permits exchanges of complete components.

The revised mutation operation involves assigning a new random value to each row, triggered when the corresponding probability surpasses a predetermined threshold.

The experimental procedure adheres to the standard evolutionary computation framework, commencing with the generation of a random initial population, followed by roulette selection, then the modified crossover, and culminating in mutation. The process concludes upon meeting predefined criteria.

3.3 Adaptability of Models

In addition to efficiency, the ”adaptability” of GA-based models is of interest. We postulate that a model's ”adaptability” is indicated by the ”minimal population required for problem resolution.”

With this concept, we evaluate the adaptability of our models under various settings. A testing protocol is established to assess model adaptability. Each test consists of ten rounds, where the model is tasked with solving a randomly generated problem. A model is considered successful in a round if it resolves the problem within a set number of generations, fixed at 150 in our experiments. If a model succeeds in at least three rounds, it is deemed to have passed the test.

Here, ”adaptability” is defined as the smallest population size required to pass a test. We focus on parent selection methods in our examination framework. The methods under comparison are the roulette and tournament methods. Moreover, experiments are conducted with varying board sizes to determine the influence of problem scale on adaptability. We hypothesize that models become less adaptable with larger-scale problems.

Given limited computational resources, pinpointing the exact boundary of the adaptable population size is challenging. Initial explorations revealed that adaptability values typically fall within a narrow range. Consequently, binary search is employed to expedite the boundary-finding process. Although there is a risk of inaccuracy due to system randomness, repeated experiments consistently yield similar results.

4 RESULTS & ANALYSIS

4.1 Vanilla Method

Observations indicate that the vanilla method lacks efficacy in evolving towards a feasible solution, often struggling to evolve at all. This suggests that the method is overly simplistic for complex problems like Japanese nonograms.

Parameter	Value
Fill rate	10%
Generations	1000
Mutation rate	10%

4.2 Intelligent Genetic Algorithm

The performance was evaluated using various parameter settings, experimenting with both roulette and tournament selection methods. Roulette selection tends to maintain a larger search space, continually exploring possibilities by recombining potentially optimal solutions with other genomes. In contrast, tournament selection rapidly converges to a locally optimized solution. The following

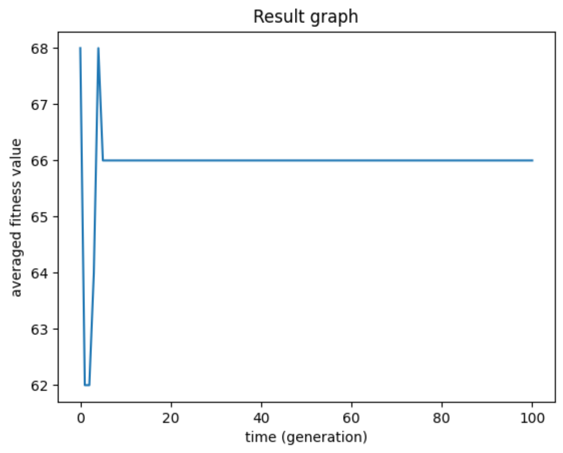


Figure 4: Tournament Selection

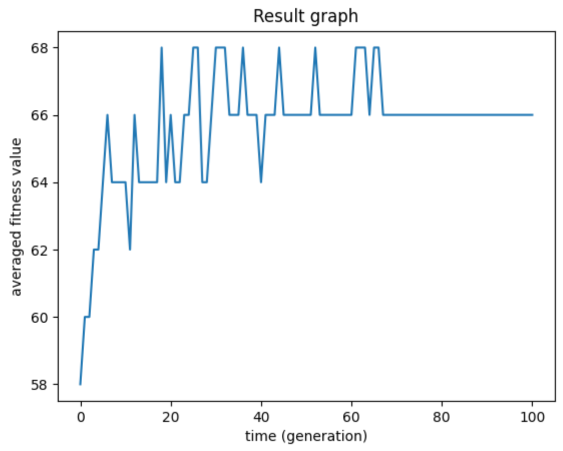


Figure 5: Roulette Selection

figures illustrate the performance of tournament 4 and roulette 5 selections.

Crossover and mutation were performed with probabilities of 0.9 and 0.05, respectively. The primary factors considered were population size and maximum generations. Generally, the size of the population is crucial in identifying the most promising solutions. A specific number of maximum generations is required to reach the convergence point.

In subsequent figures, the performance is compared using combinations of population size and maximum generation: (10, 5000) 6, (500, 100) 7, and (1000, 50) 8. A sufficiently large population can yield candidates with adequate performance; however, an insufficient population size, even with enough generations, does not guarantee acceptable results.

Remarkably, the optimal candidate was identified with the setting (500, 100), as depicted in figure 9, where only one block is misplaced.

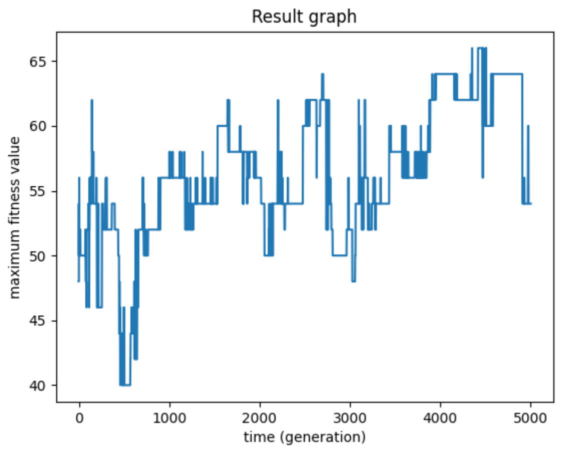


Figure 6: PS=10, MG=5000

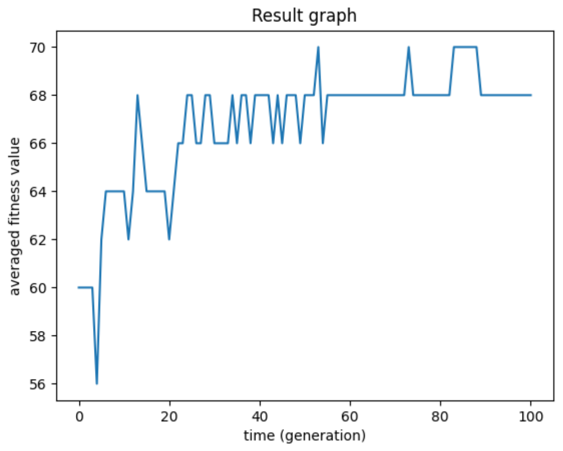


Figure 7: PS=500, MG=100

4.3 Adaptability of Models

As shown in 10, it was observed that adaptability values increase with board size and exhibit a rapid escalation after a period of gradual growth, aligning with our initial hypothesis.

Comparing the two parent selection methods, it was noted that the adaptability values for both models are comparably low for smaller-scale problems. However, as the problem scale increases, the adaptability value for the model employing the roulette method escalates more rapidly than that using the tournament method. Consequently, the tournament method is deemed more suitable for our model in solving the nonogram problem.

5 CONCLUSION

In the experiments conducted, selecting an appropriate genotype representation with favorable characteristics constitutes the initial step towards solving the problem. Furthermore, the size of the

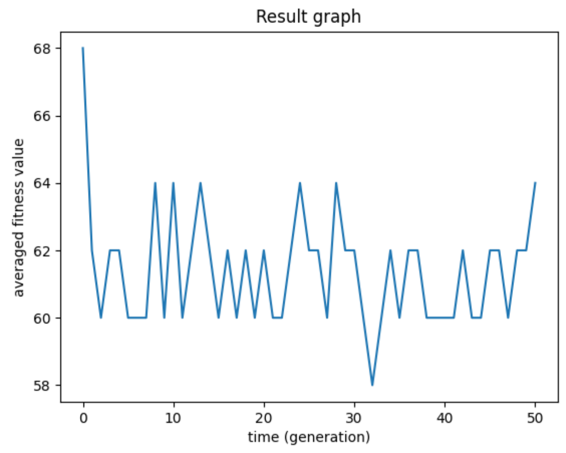
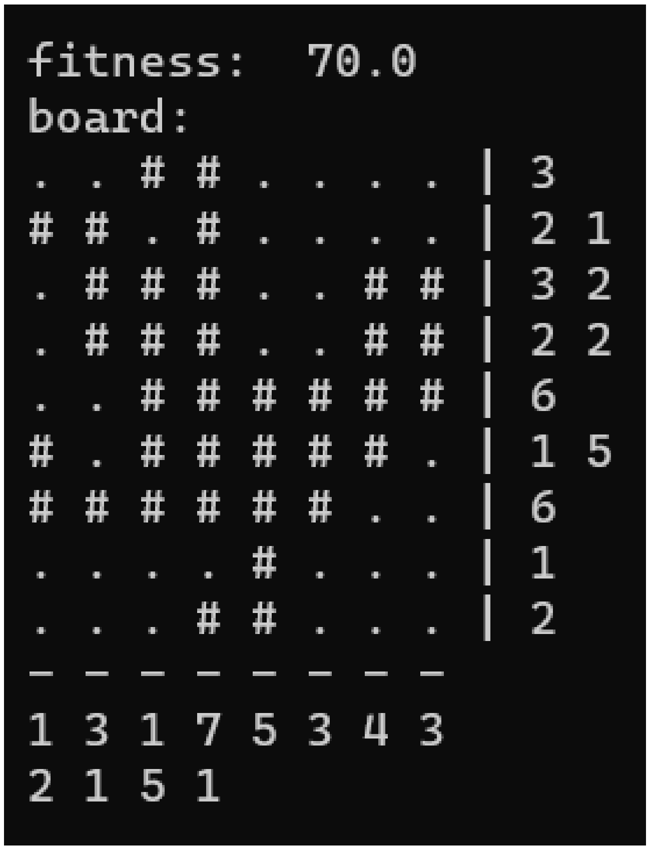


Figure 8: PS=1000, MG=50



```
fitness:  70.0
board:
. . # # . . . . | 3
# # . # . . . . | 2 1
. # # # . . # # | 3 2
. # # # . . # # | 2 2
. . # # # # # # | 6
# . # # # # # . | 1 5
# # # # # # . . | 6
. . . . # . . . | 1
. . . # # . . . | 2
— — — — — — — —
1 3 1 7 5 3 4 3
2 1 5 1
```

Figure 9: The Best Candidate with Population Size=500, Max Generation=100

population and the duration of the evolutionary process are critical factors in determining the emergence of convergence. Regarding selection methods, tournament selection proves superior to

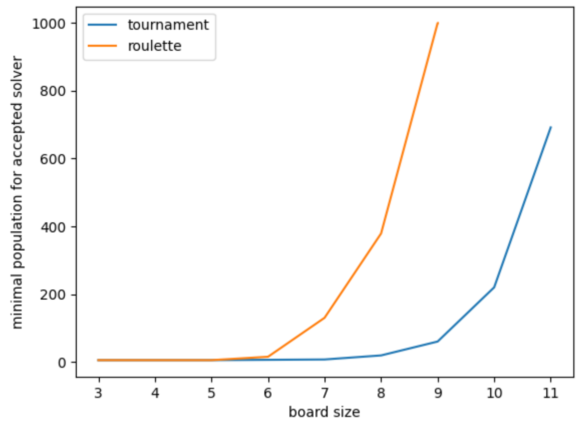


Figure 10: Adaptability of Models Under Different Conditions

the roulette method, particularly as the scale of the grid increases, thereby demonstrating enhanced adaptability to the environment.

REFERENCES

[1] Kees Joost Batenburg and Walter A Kosters. 2009. Solving nonograms by combining relaxations. *Pattern Recognition*, 42, 8, 1672–1683.

[2] Yen-Chi Chen and Shun-Shii Lin. 2019. A fast nonogram solver that won the taai 2017 and icga 2018 tournaments. *ICGA Journal*, 41, 1, 2–14.

[3] Tomasz Garbolino and Gregor Papa. 2010. Genetic algorithm for test pattern generator design: automatic evolution of circuits. *Applied Intelligence*, 32, 193–204.

[4] José Fernando Gonçalves, Jorge José de Magalhães Mendes, and Maurício GC Resende. 2005. A hybrid genetic algorithm for the job shop scheduling problem. *European journal of operational research*, 167, 1, 77–95.

[5] Min-Quan Jing, Chiung-Hsueh Yu, Hui-Lung Lee, and Ling-Hwei Chen. 2009. Solving japanese puzzles with logical rules and depth first search algorithm. In *2009 International Conference on Machine Learning and Cybernetics*. Vol. 5. IEEE, 2962–2967.

[6] Emin Erkan Korkmaz. 2010. Multi-objective genetic algorithms for grouping problems. *Applied Intelligence*, 33, 179–192.

[7] Morinim. 2023. Nonogram tutorial - vita wiki. Accessed: 2023-04-01. (2023). https://github.com/morinim/vita/wiki/nonogram_tutorial.

[8] Tadahiko Murata, Hisao Ishibuchi, et al. 1995. Moga: multi-objective genetic algorithms. In *IEEE international conference on evolutionary computation*. Vol. 1. IEEE Piscataway, 289–294.

[9] Jinn-Tsong Tsai. 2012. Solving japanese nonograms by taguchi-based genetic algorithm. *Applied Intelligence*, 37, 405–419.

[10] Jinn-Tsong Tsai, Ping-Yi Chou, and Jia-Cen Fang. 2011. Learning intelligent genetic algorithms using japanese nonograms. *IEEE Transactions on Education*, 55, 2, 164–168.

[11] Nobuhisa Ueda and Tadaaki Nagao. 1996. Np-completeness results for nonogram via parsimonious reductions. *preprint*.

[12] Chiung-Hsueh Yu, Hui-Lung Lee, and Ling-Hwei Chen. 2011. An efficient algorithm for solving nonograms. *Applied Intelligence*, 35, 18–31.