Appl Intell (2011) 35:18–31
DOI 10.1007/s10489-009-0200-0

An efficient algorithm for solving nonograms

Chiung-Hsueh Yu · Hui-Lung Lee · Ling-Hwei Chen

Published online: 13 November 2009

Abstract Nonogram is one of logical games popular in Japan and Netherlands. Solving nonogram is a NP-complete problem. There are some related papers proposed. Some use genetic algorithm (GA), but the solution may get stuck in local optima. Some use depth first search (DFS) algorithm, the execution speed is very slow. In this paper, we propose a puzzle solving algorithm to treat these problems. Based on the fact that most of nonograms are compact and contiguous, some logical rules are deduced to paint some cells. Then, we use the chronological backtracking algorithm to solve those undetermined cells and logical rules to improve the search efficiently. Experimental results show that our algorithm can solve nonograms successfully, and the processing speed is significantly faster than that of DFS. Moreover, our method can determine that a nonogram has no solution.

Keywords Japanese puzzle · Nonogram · Depth first search · Chronological backtracking

1 Introduction

Nonogram, also known as Japanese puzzle, is one of logical games popular in Japan and Netherlands. The question "Is this puzzle solvable?" is a NP-complete problem [1, 2]. Some related papers [3, 4] solved this problem by non-logical algorithms, and the execution speed is slow. In the following, we will give a brief review.

1.1 Nonograms

Figure 1(a) shows a simple nonogram. The positive integers in the top of a column or left of a row stand for the lengths of black runs in the column or row respectively. The goal is to paint cells to form a picture that satisfies the following constraints:

1. Each cell must be colored (black) or left empty (white).
2. If a row or column has k numbers: $s_1, s_2, \ldots, s_k$, then it must contain k black runs—the first (leftmost for rows/topmost for columns) black run with length s_1, the second black run with length s_2, and so on.
3. There should be at least one empty cell between two consecutive black runs.

It is evident that the puzzle in Fig. 1(a) has a unique solution shown in Fig. 1(b). However, the puzzle in Fig. 2 has two solutions, Fig. 2(b) is the first solution, Fig. 2(c) is the second solution, and the puzzle in Fig. 3 has no solution (i.e. no corresponding picture). Hence, there may be none, exact one, or more than one solution for given integral numbers. Note that a puzzle solution can be considered as a black-white picture. Here, we use ■ as a colored (black) cell, □ as an empty (white) cell, and ▨ as an unknown cell (i.e. an undetermined cell).

1.2 Constraint satisfaction problem

Formally speaking, the constraint satisfaction problem (CSP) is composed of a finite set of variables, $X_1, X_2,$

C.-H. Yu · H.-L. Lee · L.-H. Chen (✉)
Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, ROC
e-mail: lhchen@cc.nctu.edu.tw

C.-H. Yu
e-mail: Sharon@debut.cis.nctu.edu.tw

H.-L. Lee
e-mail: huilung@debut.cis.nctu.edu.tw

$\ldots, X_n$, each of X_i is associated with a domain D_i of possible values, and a set of constraints, $C_1, C_2, \ldots, C_n$ [5]. The solution of CSP is to find a legal assignment that any variable will be assigned a value without violating any constraint. In fact, a nonogram can be modeled as a CSP. Each cell in a nonogram is a variable with domain: black, white or grey. The chronological backtracking (CB) is a depth first search based algorithm and commonly used for solving CSP. The algorithm chooses one variable at a time and backtracks to the last decision when it becomes unable to proceed [5]. For obtaining the solution efficiently, CB algorithm with look ahead strategy will be applied to reduce the search space and detect illegal assignment.

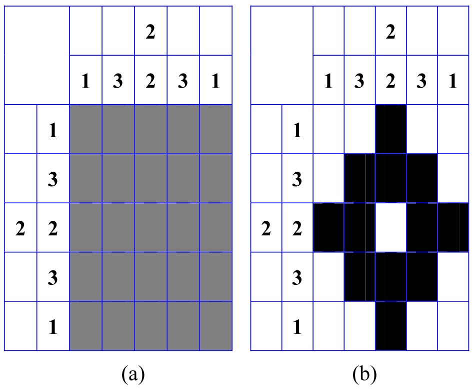


Fig. 1 Nonogram. (**a**) A simple puzzle. (**b**) The solution of (**a**)

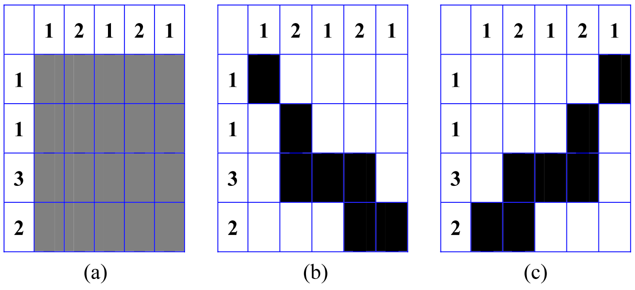


Fig. 2 A puzzle with two solutions. (**a**) A puzzle example. (**b**) The first solution. (**c**) The second solution

1.3 Previous works

In 2003, Batenburg [3] proposed an evolutionary algorithm for discrete tomography (DT). DT is concerned with the reconstruction of a discrete image from its projections [4]. Nonograms can be considered as a special form of the DT problem. Figure 4 shows an example to explain the difference between them. Batenburg and Kosters [4] modified the evolutionary algorithm to solve nonogram. Since the evolutionary algorithm in [3] will converge to a local optimum, the obtained solution may be incorrect.

In 2004, Wiggers [6] proposed a genetic algorithm (GA) and a depth first search (DFS) algorithm to solve nonograms. He also compared the performance of these two algorithms. For a puzzle of small size, DFS algorithm is faster than GA; otherwise, GA is faster. However, both methods are slow, and the GA algorithm may get stuck in local optima. In the following, we will give a brief description for the DFS algorithm.

1.4 Depth first search (DFS)

In DFS, all possible solutions of each row are generated. Figure 5 gives an example to illustrate the DFS algorithm. Figure 5(a) shows a nonogram. Figure 5(b) shows the corresponding DFS tree of Fig. 5(a), $(r.i_1, \ldots, r.i_j, \ldots, r.i_k)$ in each node stands for the first black run of row r starting at position i_1, and the jth black run of row r starting at position i_j. Figure 5(c) shows all possible solutions in the tree. Each possible solution corresponds to a path from root to a leaf node, find all paths using DFS and then use the column information (the numbers in the top of each column) of the puzzle to verify each possible solution. A possible solution will be considered as a true solution, if it satisfies all columns' restrictions. Through the verification process, we

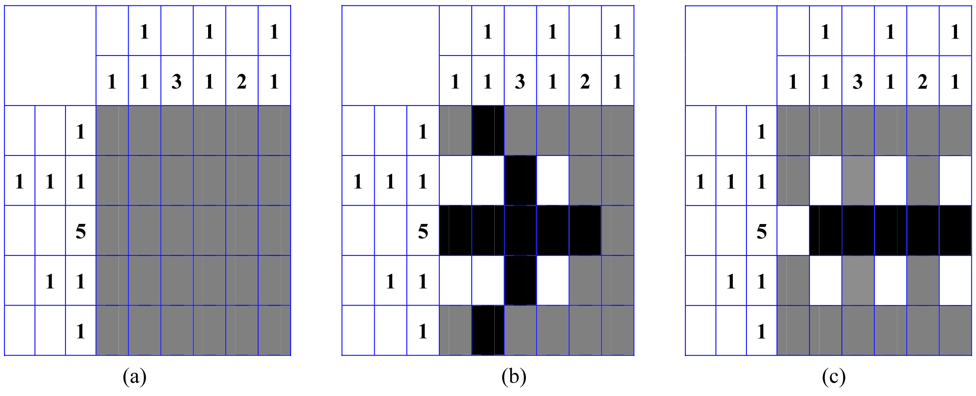


Fig. 3 A puzzle with no solution. (**a**) A puzzle example. (**b**) If the first five cells are black in row 3, there is no solution in rows 2 and 4. (**c**) If the last five cells are black in row 3, there is no solution in row 4

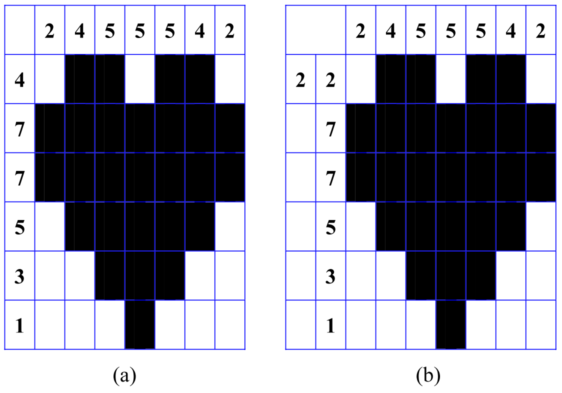


Fig. 4 DT problem and nonogram. (**a**) DT problem. (**b**) Nonogram

can see that only the fourth possible solution in Fig. 5(c) is the true one.

Note that both of the evolutionary algorithm presented in [4] and the genetic algorithm (GA) described in [6] will converge to a local optimum, so sometimes the obtained solution may get stuck in local optima. Although the DFS algorithm proposed in [6] can always find a correct solution, it takes much longer time than GA when solving large puzzles. To solve these problems, in this paper, we consider the nonogram as a CSP problem. The CB algorithm will be applied to search for solution. In order to speed up the search, some logical rules (LR) are considered as CSP filters to determine as many unknown cells as possible in a nonogram. On the other hand, LRs are used for detecting unsatisfiable cases and reducing the search space.

The remainder of this paper is organized as follows. The proposed algorithm is presented in Sect. 2. In Sect. 3, several experimental results are shown. Finally, conclusions are made.

2 Proposed method

In a general nonogram game, we usually paint those cells which can be determined immediately at first. Then, the rest of undetermined cells will be solved by guess. Based on this fact, we propose a method to solve nonograms automatically. At first, a puzzle will be solved by logical rules until all logical rules can not be applied. Then, the proposed method will use CB to visit a possible solution (an internal node) in a row and the column information is used to do verification. All logic rules will be immediately applied first before we use CB to visit the next node. The proposed method will recurrently run these steps until we obtain a solution (a leaf node).

2.1 The first phase: logical rules (LR)

One may use some rules [7] to solve nonograms. In this phase, eleven rules with a new concept of range of a black run are proposed. These rules can be divided into three main parts. The first part is used to determine which cells should be colored or left empty, the second part is used to refine the ranges of black runs, and the third is used to determine which cells should be colored or left empty and to refine the ranges of black runs.

Note that in [7], the rules of "基礎常識" and "一般常識" are similar to our rules. Some are composed of our several rules. Some are the same as ours. Our proposed rules 1.3, 1.4, 2.1, 2.3, 3.1, 3.3-2, and 3.3-3 are new idea. And our rules do not include the rules of "高等常識" in [7] since the rules of "高等常識" provide an undetermined guess. In our method, our rules are used for reducing the search space. Hence, the rules must provide a determined answer to eliminate some cases.

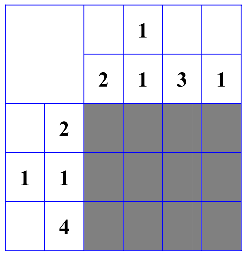
In the beginning, all cells in a puzzle are considered as unknown. Then each rule is applied in each row and then in each column. The total eleven rules are executed sequentially and iteratively. In some iterations, some unknown cells will be determined. However, in some iterations, maybe only the ranges of some black runs are refined. Thus, if no unknown cell is determined and no black run's range is changed, we will stop using logical rules and start to apply CB algorithm.

Since all rules are applied to each row and column, we only take a row as an example to explain the proposed algorithm.

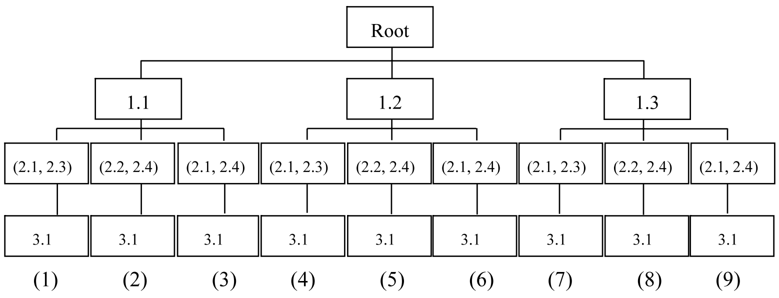
Preliminary for run range

The position where a black run may be placed plays an important role. An idea about the range (r_{js}, r_{je}) of a black run j is proposed, where r_{js} stands for the left-most possible starting position of run j, and r_{je} stands for the right-most possible ending position. That is, black run j can only be placed between r_{js} and r_{je}. If the range of each black run is estimated more precisely, we can solve puzzle more quickly. Note that for each black run, we must reserve some cells for the former black runs and the later ones. Figure 6 gives an example to illustrate the idea. Figure 6(a) shows a special row with three black runs of lengths 1, 3, and 2, respectively. Figure 6(b) shows the left-most possible solution

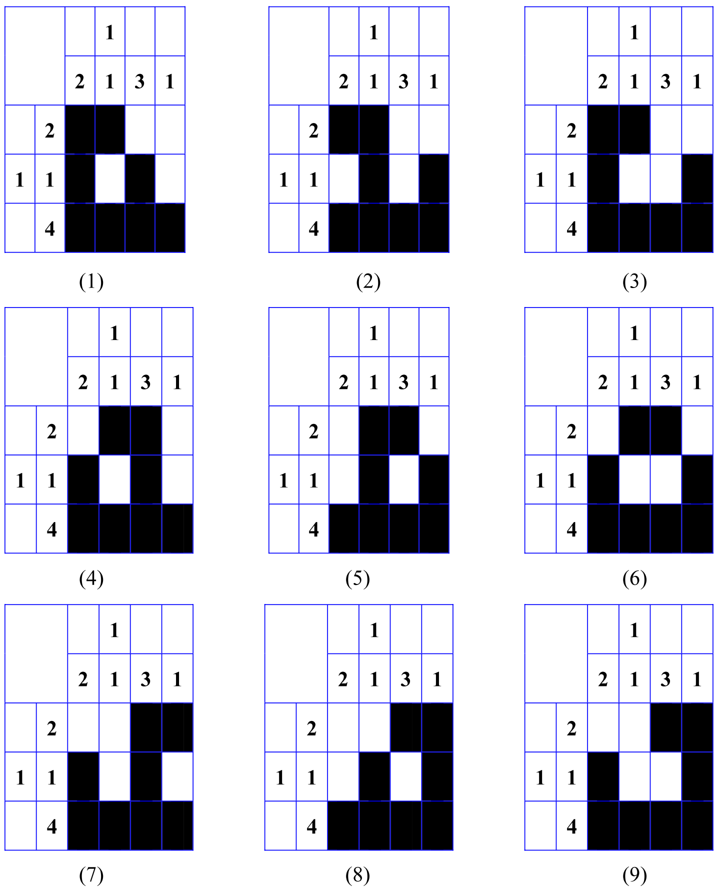
Fig. 5 An example to illustrate the DFS algorithm for solving nonogram. Only solution (4) is correct



(a) A puzzle problem.



(b) The DFS tree of (a).



(c) Nine corresponding solutions in (b).

and the left-most possible starting position of each run. Figure 6(c) shows the right-most possible solution and the right most possible ending position of each run.

Thus, in the beginning, the initial range of a black run in a row is set between its left-most possible position and its right-most possible position. In the following, we will give

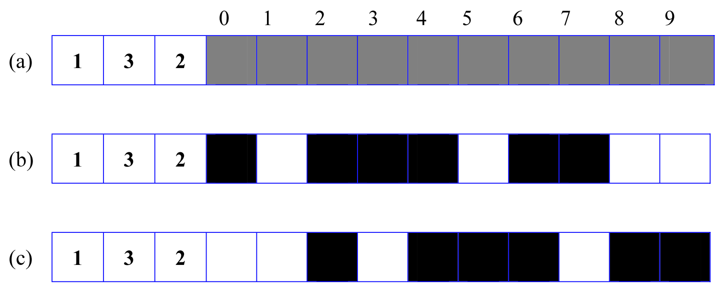


Fig. 6 An illustration of an initial black run range. (**a**) A special row of a puzzle. (**b**) The left-most possible solution of (**a**). (**c**) The right-most possible solution of (**a**)

a formal formula to calculate the initial run range of each black run.

Initial run range estimation

Let the size of each row with k black runs be n and the cells in the row with index $(0, \ldots, n-1)$, we can use the following formula to determine the initial range of each black run.

$$\begin{aligned} r_{1s} &= 0, \\ r_{js} &= \sum_{i=1}^{j-1}(LB_i + 1), \quad \forall j = 2, \ldots, k \\ r_{je} &= (n-1) - \sum_{i=j+1}^{k}(LB_i + 1), \quad \forall j = 1, \ldots, k-1 \\ r_{ke} &= n - 1 \end{aligned} \tag{1}$$

where LB_i is the length of black run i.

Using formula (1), we can get the initial ranges of the three black runs in Fig. 6(a), which are (0, 2), (2, 6), and (6, 9), respectively.

Rules in Part I

There are five rules in this part, all are used to determine which cells to be colored or left empty.

Rule 1.1

For each black run, those cells in the intersection of all the possible solutions of the black run must be colored. In fact, the intersection of all possible solutions is the intersection of the left-most possible solution of the black run and the right-most possible solution of the black run. It is obvious that the intersection exists when the length of the black run's range is less than two times the actual length of the black run. Figure 7 shows an example. Consequently, we provide Rule 1.1 to paint the cells sure to be colored. In the rule, c_i stand for the cell with index i.

Rule 1.1 For each black run j, cell c_i will be colored when $r_{js} + u \leq i \leq r_{je} - u$, where $u = (r_{je} - r_{js} + 1) - LB_j$

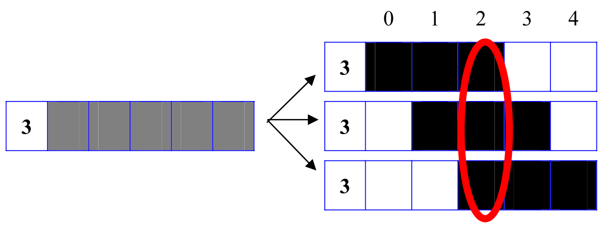


Fig. 7 An example of Rule 1.1

Rule 1.2

When a cell does not belong to the run range of any black run, the cell should be left empty. Rule 1.2 is provided to do this work.

Rule 1.2 For each cell c_i, it will be left empty, if one of the following three conditions is satisfied

(1) $0 \leq i < r_{1s}$,
(2) $r_{ke} < i < n$,
(3) $r_{je} < i < r_{(j+1)s}$ for some j, $1 \leq j < k$.

Rule 1.3

For each black run j, when the first cell $c_{r_{js}}$ of its run range is colored and covered by other black runs, if the lengths of those covering black runs are all one, cell $c_{r_{js}-1}$ should be left empty. Similarly, when the last cell $c_{r_{je}}$ is colored and covered by other black runs, if the lengths of those covering black runs are all one, cell $c_{r_{je}+1}$ should be left empty. We provide Rule 1.3 to determine whether cells $c_{r_{js}-1}$ and $c_{r_{je}+1}$ should be left empty. Taking Fig. 8 as an example, the colored cell $c_{r_{js}}$ in Fig. 8(a) is the starting cell of the range of the last black run with length 3. It is also covered by the third black run with length 1. Thus, it must be the third black run with length 1 (see Fig. 8(b)) or the head cell of the last black run (see Fig. 8(c)). No matter what case, the cell $c_{r_{js}-1}$ should be left empty.

Rule 1.3 For each black run j, $j = 1, \ldots, k$

(1) If the lengths of all black run i covering $c_{r_{js}}$ with $i \neq j$ are all one, cell $c_{r_{js}-1}$ will be left empty.

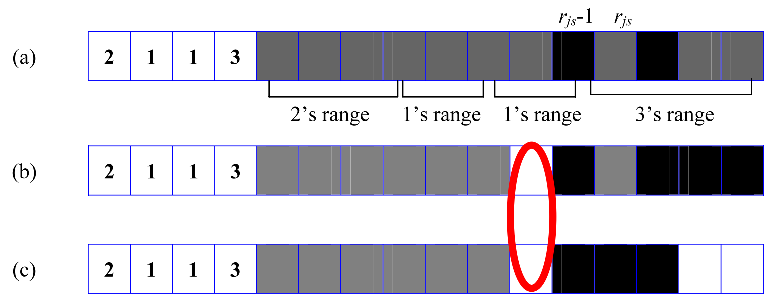


Fig. 8 An example of Rule 1.3. (**a**) One row in a puzzle with a partial painting result. (**b**) The cell $c_{r_{js}}$ belongs to the third black run with length one. (**c**) The cell $c_{r_{js}}$ is the head cell of the last black run

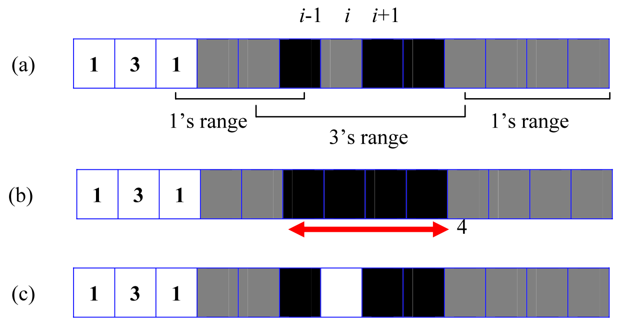


Fig. 9 An example of Rule 1.4. (**a**) One partial painting row of a puzzle with max $L = 3$. (**b**) The new black segment length after coloring c_i is $4 > 3$. (**c**) The cell c_i should be left empty

(2) If the lengths of all black run i covering $c_{r_{je}}$ with $i \neq j$ are all one, cell $c_{r_{je}+1}$ will be left empty.

Rule 1.4

There may be some short black segments in a row. If two consecutive black segments with an unknown cell between them (see Fig. 9(a)) are combined into a new black segment (see Fig. 9(b)) with length larger than the maximal length max L of all black runs containing part of this new segment, the unknown cell should be left empty (see Fig. 9(c)). Rule 1.4 is provided to deal with this situation.

Rule 1.4 For any three consecutive cells c_{i-1}, c_i, and $c_{i+1}, i = 1, \ldots, n-2$.

Let max L be the maximal length of all black runs containing the three cells.

Assumption: cells c_{i-1} and c_{i+1} are black, cell c_i is unknown. If we color c_i and find that the length of the new black segment containing c_i is larger than max L, c_i should be left empty.

Rule 1.5

Some empty cells like walls may obstruct the expansion of some black segments, we can use this property to color more cells. Figure 10 gives an example. In this figure, we do not know c_i belonging to which black run. However, an empty cell c_{i-2} obstructs the black segment containing c_i to expand to the left side of c_{i-2}. Hence, no matter c_i belongs to the run with length 3 or the run with length 4, cell c_{i+1} should be colored.

On the other hand, for a black segment covered by a series of black runs, which have the same length and overlapping ranges, if the length of the black segment equals to the length of those black covering runs, the two cells next to the two ends of the black segment are set as empty (see Fig. 11). Rule 1.5 is proposed to deal with the above two situations.

Rule 1.5 For any two consecutive cells c_{i-1} and $c_i, i = 1, \ldots, n-1$.

Constraint: cell c_{i-1} must be empty or unknown, and cell c_i must be black.

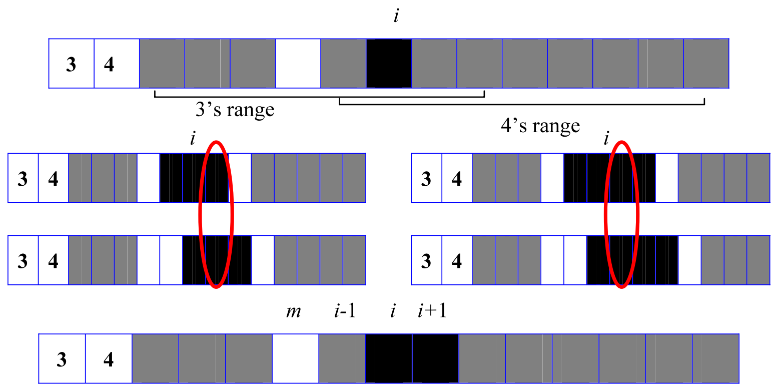


Fig. 10 An example of Rule 1.5. *Top*: one partial painting row of a puzzle. *Middle*: all possible solutions for c_i. *Bottom*: the result of applying Rule 1.5 to top figure

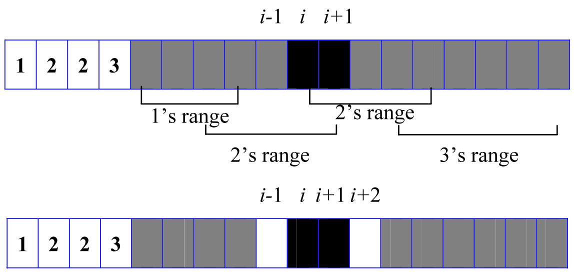


Fig. 11 An example of Rule 1.5. *Top*: all black runs containing c_i and c_{i+1} have the same length 2. *Bottom*: c_{i-1} and c_{i+2} are left empty after applying Rule 1.5 to top figure

1. Let $\min L$ be the minimal length of all black runs covering c_i.
2. Find an empty cell c_m closest to $c_i, m \in [i - \min L + 1, i - 1]$. If c_m exists, color each cell c_p with $i + 1 \le p \le m + \min L$.
3. Find an empty cell c_n closest to $c_i, n \in [i + 1, i + \min L - 1]$. If c_n exists, color each cell c_p with $n - \min L \le p \le i - 1$.
4. If all black runs covering c_i have the same length as that of the block segment containing c_i.
   (1) Let s and e be the start and end indices of the black segment containing c_i
   (2) Leave cells c_{s-1} and c_{e+1} empty.

Rules in Part II

This part contains three rules, which are designed to refine the ranges of black runs.

Rule 2.1

For two consecutive black runs j and $j + 1$, the start (end) point of run j should be in front of the start (end) point of run $j + 1$. Based on this property, Rule 2.1 is provided to update the range of each black run j with $r_{js} \le r_{(j-1)s}$ or $r_{je} \ge r_{(j+1)e}$.

Rule 2.1 For each black run j, set

$$\begin{cases} r_{js} = (r_{(j-1)s} + LB_{j-1} + 1), & \text{if } r_{js} \le r_{(j-1)_s} \\ r_{je} = (r_{(j+1)e} - LB_{j+1} - 1), & \text{if } r_{je} \ge r_{(j+1)e} \end{cases}$$

Rule 2.2

There should have at least one empty cell between two consecutive black runs, so we should update the range of black run j if cell $c_{r_{js}-1}$ or $c_{r_{je}+1}$ is colored. Rule 2.2 is proposed to treat this situation.

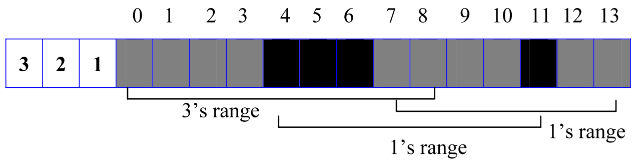


Fig. 12 An example of Rule 2.3

Rule 2.2 For each black run j, set

$$\begin{cases} r_{js} = (r_{js} + 1), & \text{if cell } c_{r_{js}-1} \text{ is colored} \\ r_{je} = (r_{je} - 1), & \text{if cell } c_{r_{je}+1} \text{ is colored} \end{cases}$$

Rule 2.3

In the range of a black run j, maybe one or more than one black segment exist. Some black segments may have lengths larger than LB_j, but some not. For each black segment with length larger than LB_j, if we can determine that it belongs to the former black runs of run j or the later ones, we can update the range of black run j. Rule 2.3 is provided to do this work. Figure 12 gives an example, the original range of the second black run is (4, 11). In (4, 11), the length of the first black segment, FS, is 3 which is larger than the length (2) of the second black run, FS belongs to the first black run. The range of the second black run can be updated to (8, 11).

Rule 2.3 For each black run j, find out all black segments in (r_{js}, r_{je}). Denote the set of these black segments by B.

For each black segment i in B with start point i_s and end point i_e. If $(i_e - i_s + 1)$ is larger than LB_j, set

$$\begin{cases} r_{js} = (i_e + 2), & \text{if black segment } i \text{ only belongs to the former black runs of run } j \\ r_{je} = (i_s - 2), & \text{if black segment } i \text{ only belongs to the later black runs of run } j \end{cases}$$

Rules in Part III

This part is composed of three rules. The purpose of each rule is not only to determine which cells should be colored or left empty but also to refine the ranges of some black runs.

Rule 3.1

When several colored cells (see Fig. 13(a)) belonging to the same black run are scattered, all unknown cells among them should be colored to form a new black segment, and the run range can also be updated (see Fig. 13(b)). Rule 3.1 is presented to treat this situation.

Rule 3.1 For each black run j, find the first colored cell c_m after $r_{(j-1)e}$ and the last colored cell c_n before $r_{(j+1)s}$ color

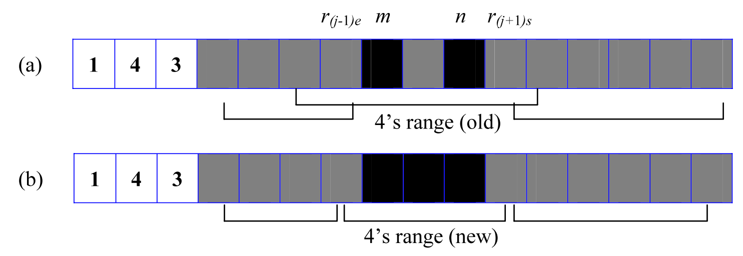


Fig. 13 An example of Rule 3.1. (**a**) One partial painting row of a puzzle. (**b**) The result of applying Rule 3.1 to (**a**)

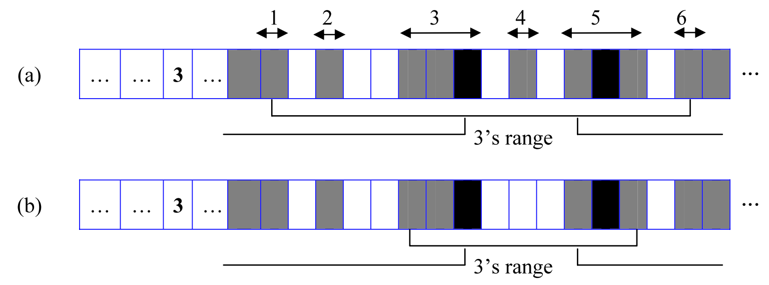


Fig. 14 An example of Rule 3.2. (**a**) One partial painting row of a puzzle. (**b**) The result of applying Rule 3.2 to (**a**)

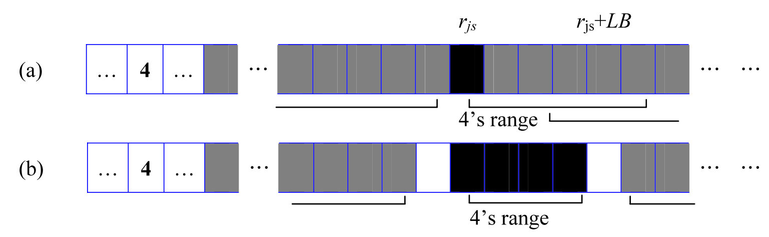


Fig. 15 An example of Rule 3.3-1. (**a**) One partial painting row of a puzzle. (**b**) The result of applying Rule 3.3-1 to (**a**)

all cells between c_m and c_n, and set

$$\begin{cases} r_{js} = (m-u) \\ r_{je} = (n+u) \end{cases}$$

where $u = LB_j - (n-m+1)$

Rule 3.2

As shown in Fig. 14(a), some empty cells may be scattered over the range of black run j, so there will be several segments bounded by these empty cells. The lengths of some segments may be less than LB_j, these segments can be skipped and the run range can be updated. Rule 3.2 is proposed to solve this situation. Figure 14(a) shows six segments in 3's range. The first two and the last segments with lengths less than 3 will be skipped and the run range can be adjusted. Since the length of the forth segment is less than 3 and is covered only by one black run with length 3, it should be empty. Figure 14(b) shows the result of applying Rule 3.2 to Fig. 14(a).

Rule 3.2 For each black run j, find out all segments bounded by empty cells in (r_{js}, r_{je}). Denote the number of these segments to be b and index them as $0, 1, \ldots, b-1$.

Step 1. Set $i = 0$.

Step 2. If the length of segment i is less than LB_j, $i = i+1$ and go to step 2. Otherwise, set r_{js} = the start index of segment i, stop and go to step 3.

Step 3. Set $i = b-1$.

Step 4. If the length of segment i is less than LB_j, $i = i-1$ and go to step 4. Otherwise, set r_{je} = the end index of segment i, stop and go to step 5.

Step 5. If there still remain some segments with lengths less than LB_j, for each of this kind of segments, if the segment does not belong to other black runs, all cells in this segment should be left empty

Rule 3.3

This rule is designed for solving the situations that the range of black run j do not overlap the range of black run $j-1$ or $j+1$. First, if black run j does not overlap the range of black run $j-1$, consider the following three cases:

Case 1: Cell $c_{r_{js}}$ is black (see Fig. 15(a)).

We can finish this black run (see Fig. 15(b)). Rule 3.3-1 is provided to treat this situation.

Rule 3.3-1 For each black run j with $C_{r_{js}}$ colored and its range not overlapping the range of black run $j-1$,

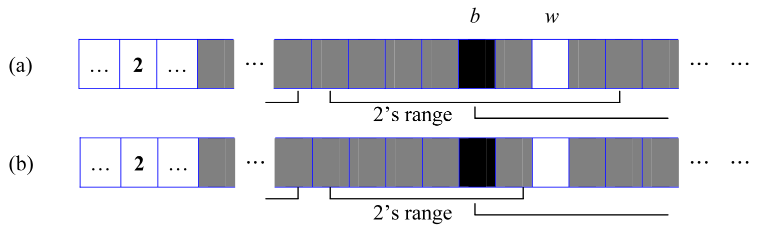


Fig. 16 An example of Rule 3.3-2. (**a**) One partial painting row of a puzzle. (**b**) The result of applying Rule 3.3-2 to (**a**)

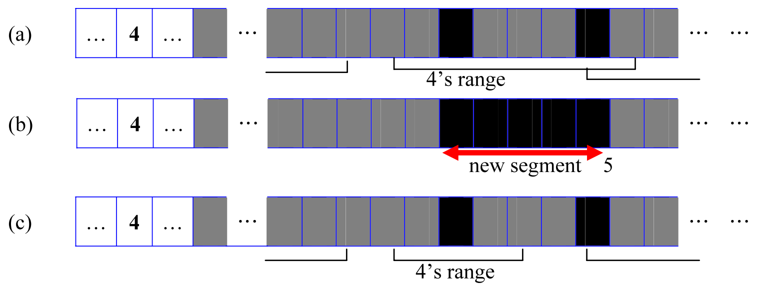


Fig. 17 An example of Rule 3.3-3. (**a**) One partial painting row of a puzzle. (**b**) The result of merging two black segments. (**c**) The result of applying Rule 3.3-3 to (**a**)

(1) Color cell C_i, where $r_{js}+1 \le i \le r_{js}+LB_j-1$ and leave cell $C_{r_{js}-1}$ and $C_{r_{js}+LB_i}$ empty
(2) Set $r_{je}=(r_{js}+LB_j-1)$
(3) If the range of black run $j+1$ overlaps the range of black run j, set $r_{(j+1)s}=(r_{je}+2)$
(4) If $r_{(j-1)e}=r_{js-1}, r_{(j-1)e}=r_{js-2}$

Case 2: An empty cell c_w appears after a black cell c_b (see Fig. 16(a)).

It should be true that each cell after c_w will not belong to black run j. Rule 3.3-2 is provided to refine the range of black run j as shown in Fig. 16(b).

Rule 3.3-2 For each black run j with its range not overlapping the range of black run $j-1$, if an empty cell c_w appears after a black cell c_b with c_w and c_b in the range of black run j. Set $r_{je}=w-1$

Case 3: There is more than one black segment in the range of black run j (see Fig. 17(a)).

In (r_{js}, r_{je}), find the first and second black segments. If the length of the new run after merging these two black segments by coloring those cells between these two segments is larger than LB_j, then these two segments should not belong to the same run. Otherwise, keep checking the length of the new run after merging the first and third black segments. The process will be repeated until all black segments in (r_{js}, r_{je}) have been checked or a black segment i is found such that the length of the new run after merging the first black segment and black segment i is larger than LB_j. Rule 3.3-3 is provided to deal with this situation. Figure 17 gives an example. In Fig. 17(a), the black run with length 4 covers two black segments. The length after merging these two black segments is 5 (see Fig. 17(b)), which is larger than 4, thus we can update the run range as shown in Fig. 17(c).

Rule 3.3-3 For each black run j with its range not overlapping the range of black run $j-1$, if there is more than one black segment in the range of the black run j, find out all black segments in (r_{js}, r_{je}).

Denote the number of these black segments to be b and index them as $0, 1, \ldots, b-1$.

Step 1. Set $i=0$
Step 2. Find the first black cell c_s in black segment i
Step 3. Set $m=i+1$
Step 4. If $m <$ b, find the first black cell c_t and the end black cell c_e in black segment m,

If $(e-s+1) > LB_j$, stop and set $r_{je}=t-2$. Otherwise, $m=m+1$ and go to step 4.

If the range of black run j does not overlap the range of black run $j+1$, we can use the similar way to treat this situation.

2.2 The second phase: chronological backtracking with LR filter

The chronological backtracking (CB) is a depth first search based algorithm [5] and commonly used for solving CSP. CB is an exhaustive search, so it can find the puzzle solution eventually. Since the CB is time-consuming, here we use the LR to raise the processing speed.

One thing should be mentioned at first, we use row information to build a DFS tree, and the column information is used immediately to do verification when a row is used to create a layer. It means that every layer of the DFS tree is composed of row information, and all nodes in each layer

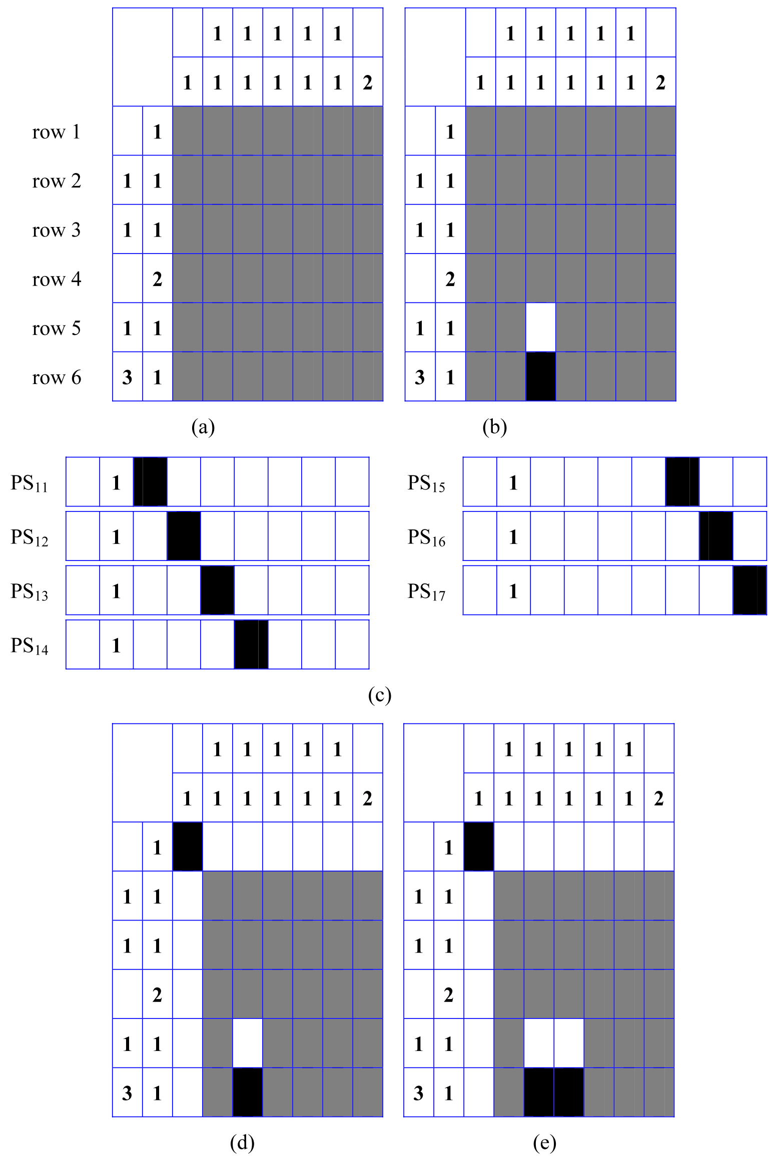


Fig. 18 An example of chronological backtracking with LR filter. (**a**) A given puzzle. (**b**) The result after applying LRs to (**a**). (**c**) Seven possible solutions for row 1. (**d**) The result after visiting node PS$_{11}$ and doing column verification. (**e**) The result after applying LRs to (**d**). (**f**) Ten possible solutions for row 2. (**g**) The result after visiting node PS$_{21}$ and doing column verification. (**h**) The result after applying LRs to (**g**). (**i**) The result after visiting the next node (the first possible solution in row 3) of the DFS tree and doing column verification. (**j**) The result after applying LRs to (**i**)

are the possible solutions (PS) for the corresponding row. Figure 18 gives an example for CB with LR filter.

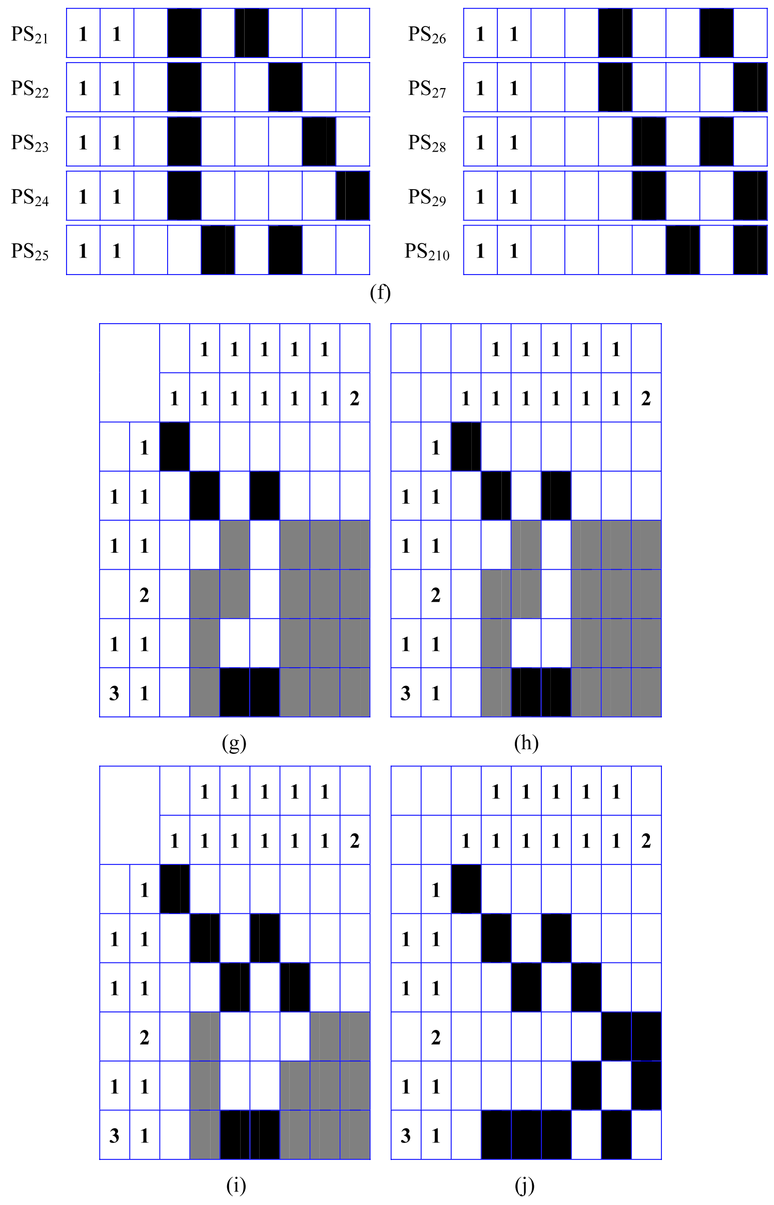
First, we run the LRs to the nonogram of Fig. 18(a) until LRs can not be applied. We obtain the result as shown in Fig. 18(b). There are seven possible solutions for row 1 (see Fig. 18(c)). Next, we visit the first possible solution PS$_{11}$ of row 1 in layer 1 of the DFS tree. Then the column information is used to do verification. We will obtain the result as shown in Fig. 18(d). Subsequently, we apply the LRs to Fig. 18(d) and the result is shown as Fig. 18(e). There are ten possible solutions for row 2 (see Fig. 18(f)). Then, we visit the first possible solution PS$_{21}$ of row 2 in layer 2 of the DFS tree. We use the column information to do verification. We obtain the result as shown in Fig. 18(g). Subsequently, we apply the LRs to Fig. 18(g) and the result is shown as Fig. 18(h). We recurrently apply the above procedure to the remaining rows until the last row is processed. The final result is shown in Fig. 18(j).

3 Experimental results

To do experiment, we collect 264 puzzles. Most of them come from [4, 6–8] and few are created by us. A PC (CPU: AMD Athlon 2600+ 1.92 GHz) is used to run the proposed method and the methods GA and DFS provided in [6].

Because most nonograms are meaningful, they can be solved quickly and completely only by logical rules. The execution time is less than 1 second. If a puzzle cannot be solved deterministically using logical rules, the unknown part left can be solved successfully by CB with LR filter. Figure 19 shows some test image. Using the methods provided in [6], Figs. 19(a), (b) and (d) take more than 20

Fig. 18 (*continued*)



minutes to get the solutions, even an incorrect solution, but using our proposed logical rules, only about 0.1 second is needed to solve them correctly. Using the proposed method, Figs. 19(c) and (f), which are random images (50% black), needs 13.1 and 23.4 seconds respectively to get the solutions, the speed is slow. The reason is that when black cells in a puzzle scatter everywhere, the LRs will fail to solve this kind of puzzles in the first time to apply LRs. Furthermore, in CB with LR filter, since there are many short black runs in each row, this produces many possible solutions in each row and we should check all of them. However, using the GA or DFS provided in [6], more than 2 days is needed, our method is still faster.

On the other hand, Fig. 19(e) is solved successfully within 4.82 minutes using the proposed method. The reason is that there are many short black runs in a row, that is, many possible solutions should be checked in the DFS tree.

Figure 20 from [4] is a puzzle with two solutions. After applying our algorithm, all solutions are found. Figure 21 shows a puzzle with no solution, using GA [6] will get a local optimal answer (see Fig. 21(b)). However, our method can detect it quickly. We use a scheme that when one cell is determined as a colored cell, but it has been left empty, the puzzle is considered to have no solution. Similarly, if one cell is determined as an empty cell, but it has been colored, the puzzle will also be considered to have no solution.

Finally, Fig. 22 shows some other test images and Table 1 shows the comparison of the experimental results using the methods proposed in [6] and our proposed algorithm.

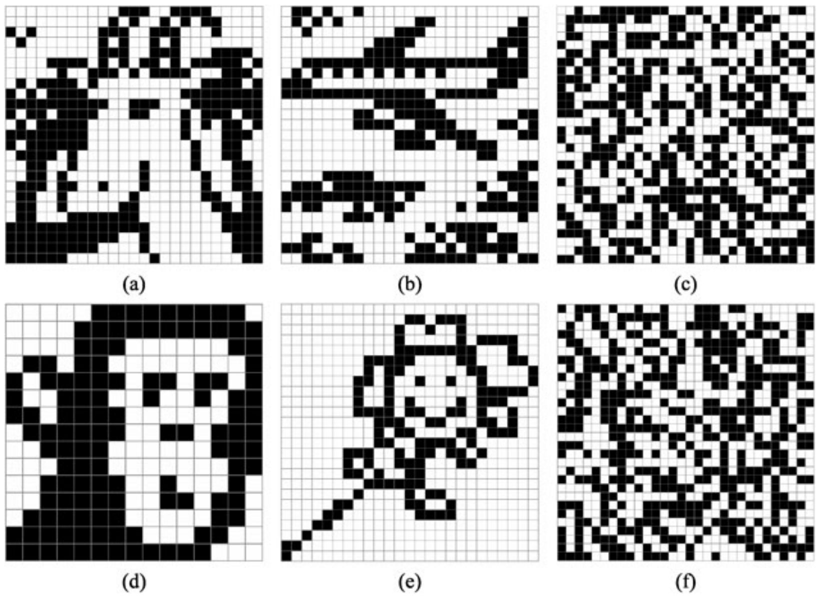


Fig. 19 Test images. (**a**) Sheep (25 × 25). (**b**) Airplane (25 × 25). (**c**) Random_1 (30 × 30). (**d**) Monkey (15 × 15). (**e**) Sunflower (25 × 25). (**f**) Random_2 (30 × 30)

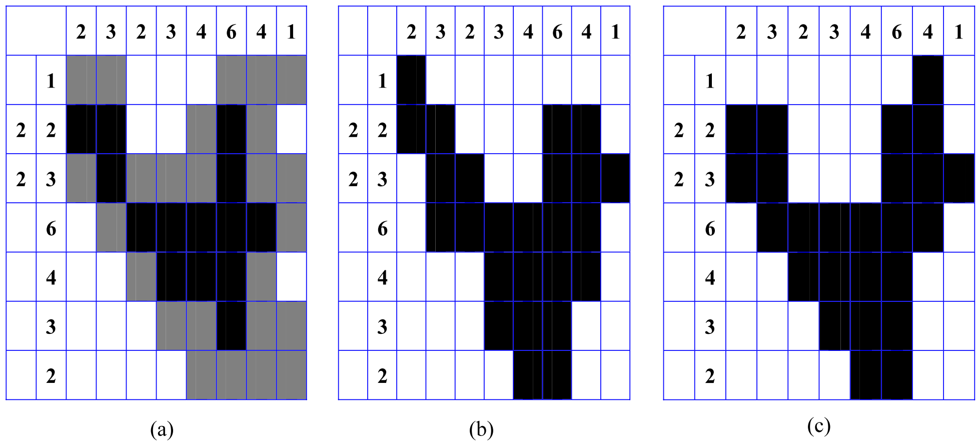


Fig. 20 A 7 × 8 puzzle with two solutions. (**a**) The result after applying logical rules. (**b**) The first solution. (**c**) The second solution

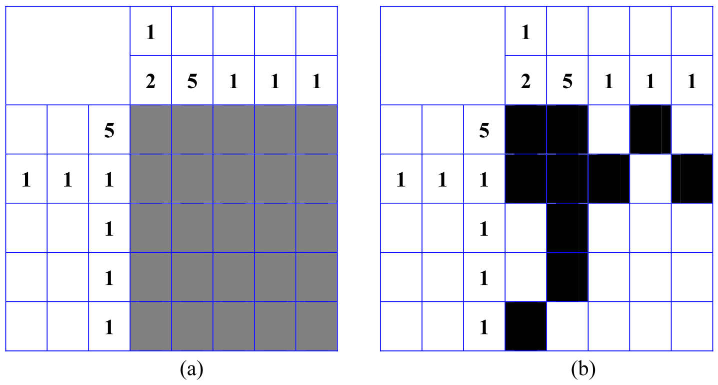


Fig. 21 A puzzle with no solution. (**a**) The original puzzle. (**b**) A local optimal answer obtained using GA

Fail denotes the case where the executing time is more than 24 hour. From Table 1, we can see that over 93% puzzles take more than 1 minute using GA or DFS. However, applying our proposed method, over 98% puzzles need less than 0.6 seconds. All puzzles in our database are solved successfully and correctly. The detail results are shown in Table 1.

Note that our decision heuristic used in the search is lexicographical (i.e. selects sequentially row by row starting from the first). We have also done other experiments using the most conflicting heuristic of starting with the row with

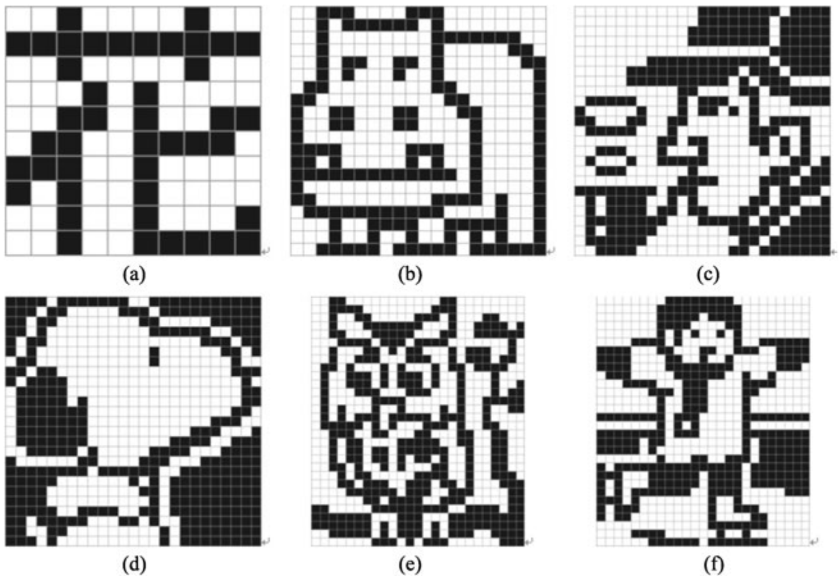


Fig. 22 Test images. (**a**) Flower_word (10 × 10). (**b**) Hippo (20 × 20). (**c**) Formosa (25 × 25). (**d**) Snoopy (25 × 25). (**e**) Owl (30 × 25). (**f**) Skating (30 × 25)

Table 1 The comparison of the experimental results among methods in [6] and our algorithm

Puzzle size: Number of puzzles (264 puzzles)		Execution time		
		GA	DFS	Our method
5×5:1, 5×6:1		Local optimal answer >1 min	No solution <0.07 sec	No solution <0.032 sec
10×10:2, 30×40:1		Local optimal answer >1 min	No solution >1 min	
(above five puzzles have no solution)				
≦6×6:7		<6 sec	<0.7 sec	<0.048 sec
6×6–10×10:9		30 sec:1, >1 min:8	<1 min:7, >1 min:2	<0.048 sec
10×10–15×15:33 15×15–25×25:111 ≧25×25:99	Random_1	Fail	Fail	13.1 sec
	Sunflower	>1 hr	>1 hr	4.82 min
	Random_2	>1 hr	>1 hr	23.4 sec
	Others	>1 min	>1 min	Owl: 0.25 sec Skating: 0.359 sec Others: <0.517 sec

tighter constraints (i.e. higher numbers). The result shows that the search with the most conflicting heuristic does not perform better than with the lexicographical heuristic. In our method, rows are processed sequentially, once the column information is used to do verification, some cells in the next row according to the column information may be determined; this will reduce the size of the DFS tree. For example, in Fig. 23, if cell i in the first row and cell j in the second row are colored, we can certainly determine that cell k in the third row should be colored. If we select the row with the higher number, then row 6 and row 4 will be processed first. Since these two do not adjoin, the column information can not be used to determine the cells in rows 3, 5, or 7. In Fig. 23, if cell o and m are colored, we can not determine cell n.

4 Conclusions and discussion

In this paper, we have proposed a fast method to solve nonograms. The method first applies the LRs to solve nonograms iteratively until all LRs can not be applied. Most nonograms can be solved by only applying LRs. If it can not be solved completely, the proposed method will use CB with LR filter to solve those unknown cells.

The experimental results show that our method can solve those puzzles with compact black patterns quickly. For those puzzles with random black patterns, the method can also raise the speed of DFS using the pruning scheme. Furthermore, our method can always provide correct solutions and is better than the methods provided in [6].

Since the DT problem can be considered as a general form of nonograms, extending our method to solve the DT

Fig. 23 An example to illustrate the search with the most conflict heuristic and the lexicographical heuristic

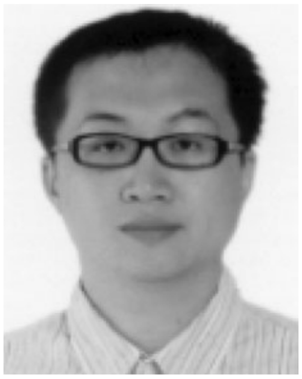
			3	
			5	
1		**…**	*i*	
2		**…**	*j*	
2		**…**	*k*	
4		**…**	*m*	
3		**…**	*n*	
5		**…**	*o*	
⋮		**…**		
2		**…**		

problem is our future work. At first, a DT problem can be decomposed into many kinds of corresponding nonograms. Then these nonograms are considered as the input of our method, and our method will provide all of the correct solutions. Moreover, our method can determine that a nonogram has no solution. The number of the corresponding nonograms is very large, and most nonograms have no solution and take a lot of processing time, the key challenge of our future work is to reduce the number of nonograms corresponding to a given DT.

References

1. Ueda N, Nagao T (1996) NP-completeness results for NONOGRAM via parsimonious reductions. Technical report TR96-0008, Department of Computer Science, Tokyo Institute of Technology, May 1996
2. McPhail BP (2005) Light up is NP-complete. Feb 2005. http://www.reed.edu/~mcphailb/lightup.pdf
3. Batenburg KJ (2003) An evolutionary algorithm for discrete tomography. Master thesis in computer science, University of Leiden, The Netherlands
4. Batenburg KJ, Kosters WA (2004) A discrete tomography approach to Japanese puzzles. Proceedings of BNAIC, pp 243–250
5. Tsang EPK (1993) Foundations of constraint satisfaction. Academic Press, London
6. Wiggers WA (2004) A comparison of a genetic algorithm and a depth first search algorithm applied to Japanese nonograms. Twente student conference on IT, Jun 2004
7. http://www.pro.or.jp/~fuji/java/puzzle/nonogram/knowhow.html
8. Database of Japanese puzzles. http://hattori.m78.com/puzzle/picture/java/stage_01/index.html and http://www.books.com.tw/exep/prod/booksfile.php?item=0010317755

Chiung-Hsueh Yu was born in Hsinchu, Taiwan, Republic of China on February 26, 1982. She received the B.S. degree and M.S. degree in Computer Science from National Chiao Tung University, Hsinchu, Taiwan in 2004 and 2007, respectively. Her research interests include image processing and algorithm. Currently, she is an engineer in Generalplus Technology Inc. in Hsinchu, Taiwan. She takes charge of maintaining and writing the algorithm of image processing and signal processing.



Hui-Lung Lee was born in Taoyuan, Taiwan, ROC, on June 16, 1976. He received the M.S. degree in Computer Science and Information Engineering from Fu Jen Catholic University in 2003. Currently, he is a Ph.D. candidate of the Department of Computer Science at National Chiao Tung University. His major research interests include information hiding and watermarking.

Ling-Hwei Chen was born in Changhua, Taiwan, in 1954. She received the B.S. degree in Mathematics and the M.S. degree in Applied Mathematics from National Tsing Hua University, Hsinchu, Taiwan in 1975 and 1977, respectively, and the Ph.D. degree in Computer Engineering from National Chiao Tung University, Hsinchu, Taiwan in 1987.

From August 1977 to April 1979 she worked as a research assistant in the Chung-Shan Institute of Science and Technology, Taoyan, Taiwan, From May 1979 to February 1981 she worked as a research associate in the Electronic Research and Service Organization, Industry Technology Research Institute, Hsinchu, Taiwan. From March 1981 to August 1983 she worked as an engineer in the Institute of Information Industry, Taipei, Taiwan. She is now a Professor of the Department of Computer Science at the National Chiao Tung University. Her current research interests include image processing, pattern recognition, Multimedia compression, content-based retrieval and Multimedia Steganography.