SOLVING JAPANESE PUZZLES WITH LOGICAL RULES AND DEPTH FIRST SEARCH ALGORITHM

MIN-QUAN JING, CHIUNG-HSUEH YU, HUI-LUNG LEE, LING-HWEI CHEN

Department of Computer Science, National Chiao Tung University Hsinchu, Taiwan, R.O.C.
E-MAIL: ching@debut.cis.nctu.edu.tw, sharon@debut.cis.nctu.edu.tw, huilung@debut.cis.nctu.edu.tw, lhchen@cc.nctu.edu.tw

Abstract:

Japanese puzzle is one of logical games popular in Japan and Netherlands. Solving Japanese puzzle is a NP-complete problem. There are some related papers proposed. Some use genetic algorithm (GA), but the solution may be wrong. Some use depth first search (DFS) algorithm, which is an exhaustive search, 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 Japanese puzzle are compact and contiguous, some logical rules are deduced to paint some cells. Then, the DFS algorithm with the "branch and bound" scheme, which is used to do early termination for those impossible paths, is used to solve those undetermined cells. Experimental results show that our algorithm can solve Japanese puzzles successfully, and the processing speed is significantly faster than that of DFS.

Keywords:

Japanese puzzle; Nonogram; Depth first search; Branch and bound

1. Introduction

Japanese puzzle, also known as nonogram, 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 this paper, we firstly apply some logical rules to solve most part of a puzzle, and then DFS with branch and bound scheme is used to solve the remaining part.

Fig. 1(a) shows a simple Japanese puzzle and Fig. 1(b) is the solution of Fig. 1(a). Ignoring the numbers, the solution can be considered as a black-white (1-and-0) picture. Here, we use ■ as a colored (black) cell, □ as an empty (white) cell, and 🟪 as an unknown cell (i.e. an undetermined cell). The positive integers in the top rows and left columns give the lengths of black runs in the corresponding row and column 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, …, 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.

Figure 1(a) is a puzzle which has a unique solution (see Fig. 1(b)). Some puzzle may be no, exact one, or more than one solution for given integral numbers.

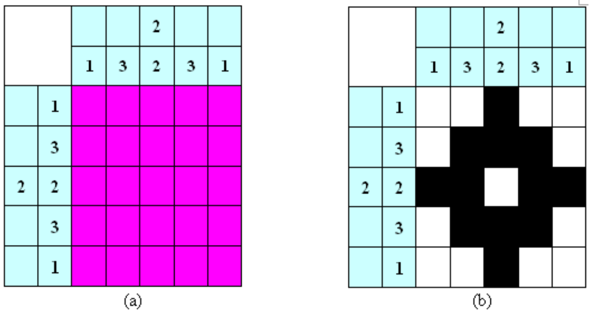

Figure 1. Japanese puzzle. (a) A simple puzzle. (b) The solution of (a).

2. Previous Works

In 2003, Batenburg [3] described an evolutionary algorithm for discrete tomography (DT) and then Batenburg and Kosters [4] provided a method to solve Japanese puzzle. By modifying the fitness function in [3], the evolutionary algorithm can be used to solve Japanese puzzle. Since the evolutionary algorithm in [3] will converge to a local optimum, the obtained solution may be incorrect. In 2004, Wiggers [5] proposed a genetic algorithm (GA) and a depth first search (DFS) algorithm to solve Japanese puzzles. 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. In this paper, we apply some logical rules (LR) to determine the unknown cells in a Japanese puzzle as many as possible, and then the DFS algorithm with “branch and bound” is used to solve those remaining unknown cells.

3. Proposed Method

In general Japanese puzzle 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 will propose a method to solve Japanese puzzles automatically. The method contains two phases. In the first phase, some logical rules are deduced and used to determine some cells; in the second phase, the DFS algorithm will be applied to solve those remaining unknown cells. Furthermore, the “branch and bound” technique is used to accelerate the searching speed of the general DFS.

3.1. The first phase: Logical rules (LR)

One may use some rules [6] to solve Japanese puzzles. In this phase, eleven rules with a new concept of range of a black run (“range” will be explained later) are proposed. These rules can be divided into three main parts. The first part is to determine which cells should be colored or left empty, the second part is to refine the ranges of black runs, and the third is not only to determine which cells should be colored or left empty but also to refine the ranges of black runs. Each rule will be applied in each row and then in each column. The total rules are executed sequentially and iteratively.

In the beginning, all cells in the puzzle are considered as unknown. In some iterations, some unknown cells will be determined as colored cells or empty ones. However, in some iterations, maybe only the ranges of some possible black runs are refined. That is, there are not always some unknown cells determined in each iteration. Thus, if no unknown cell is determined and no black run’s range is changed, we will stop using logical rules because there will be no changes in later iterations.

Note that, the rules applied in a row are the same as those applied in a column, so we only take a row as an example to explain our algorithm.

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 starting position of the run, and r_{je} stands for the right-most ending position of the run. That is, black run j can only be placed between r_{js} and r_{je}. If the range of each black run is precisely estimated, these range information can help us solve puzzles quickly. In the beginning, the initial range of a black run in a row is set between the left-most possible position and the right-most possible position. For each black run, it must reserve some cells for the former black runs and the later ones.

Initial run range estimating

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

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

where LB_i is the length of black run i.

Rules in Part I

There are five rules in this part, all of them are used to determine which cells should be colored (black) or left empty (white).

Rule 1.1

For each black run, some cells must be colored if all the possible solutions of the black run have the intersection. Actually, the intersection of all possible solutions is also the intersection of the left-most case of the black run and the right-most case 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 a black run.

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

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
For each cell c_i, it will be left empty,
if one of the following three conditions is satisfied
(1) $0 \le i < r_{1s}$,
(2) $r_{ke} < i < n$,
(3) $r_{je} < i < r_{(j+1)s}$ for some $j, 1 \le j < k$.

Rule 1.3

For each black run j, when the first cell $c_{r_{js}}$ of its range is colored, we will check $c_{r_{js}}$ covered by what other black runs. If the lengths of those covering black runs are all one, the cell $c_{r_{js}\text{-}1}$ should be left empty. In the similar way, when the last cell $c_{r_{je}}$ of its range is colored, we will check $c_{r_{je}}$ covered by what other black runs. If the lengths of those covering black runs are all one, the cell $c_{r_{je}+1}$ should be left empty. We provide Rule 1.3 to determine whether cells $c_{r_{js}\text{-}1}$ and $c_{r_{je}+1}$ 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 $c_{r_{js}\text{-}1}$ will 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 are combined into a new black segment with length larger than the maximal length *maxL* of all black runs containing part of this new segment, the unknown cell should be left empty.

Rule 1.4
For any three consecutive cells $c_{i\text{-}1}$, c_i, and c_{i+1}, $i = 1, \ldots, n\text{-}2$
Constraint: cells $c_{i\text{-}1}$ and c_{i+1} must be black, and cell c_i must be unknown
1. Let *maxL* be the maximal length of all black runs containing the three cells.
2. If we color c_i and find that the length of the new black segment containing c_i is larger than *maxL*, c_i should be left empty.
3. If we color c_i and find that the length of the new black segment containing c_i is larger than *maxL*, 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. On the other hand, for a black segment covered by series of black runs, which have the same length but have 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.

Rule 1.5
For any two consecutive cells $c_{i\text{-}1}$ and c_i, $i = 1, \ldots, n\text{-}1$
Constraint: cell $c_{i\text{-}1}$ must be empty or unknown, and cell c_i must be black
1. Let *minL* be the minimal length of all black runs covering c_i.
2. Find an empty cell c_m closest to c_i, $m \in (i\text{-}minL + 1, i\text{-}1)$
   If c_m exists, color each cell c_p with $i + 1 \leq p \leq m + minL$.
3. Find an empty cell c_n closest to c_i, $n \in (i + 1, \ i + minL\text{-}1)$
   If c_n exists, color each cell c_p with $n\text{-}minL \leq p \leq i\text{-}1$.
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 $c_{s\text{-}1}$ 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 $r_{js} \leq r_{(j-1)s}$ or $r_{je} \geq r_{(j+1)e}$.

Rule 2.1
For each black run j, set

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

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 the cell $c_{r_{js}\text{-}1}$ or $c_{r_{je}+1}$ is colored.

Rule 2.2
For each black run j, set

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

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

For each black run j, find out all black segments in (r_{js}, r_{je})
We denote the class of these black segments by B.
For each black segment i in B with start point i_s and end index i_e,
If (i_e-i_s+1) is larger than LB_j, set

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

Rules in Part III

This part is also 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 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.

Rule 3.1

For each black run j, find the first colored cell c_m after $r_{(j\text{-}1)e}$ and the last colored cell c_n before $r_{(j+1)s}$
color all cells between c_m and c_n, and set

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

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

Rule 3.2

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

For each black run j, find out all segments bounded by empty cells in (r_{js}, r_{je}).
We denote the number of these segments to be b and index them as 0, 2, …, 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 in $R$,
        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, for the situation "black run j not overlap the range of black run j-1", we consider the following three cases:

Case1: The cell $c_{r_{js}}$ is black

Because the black run j does not overlap the black run j-1, when the head cell of black run j has been colored, we can finish this black run. 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,

(1) Color cell c_i, where $r_{js} + 1 \le i \le r_{js} + LB_j$-1
and leave cell $c_{r_{js}+LB_j}$ 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
$r_{(j+1)s} = (r_{js} + 2)$

Case2: An empty cell c_w appears after a black cell c_b

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.

Rule 3.3-2

For each black run j with its range not overlapping the range of black run j - 1,
Constraint: 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

Case3: There is more than one black segment in the range of black run j

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.

Rule 3.3-3

For each black run j with its range not overlapping the range of black run $j-1$

Constraint: 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}).

We denote the number of these black segments to be b and index them as 0, 2, …, $b-1$.

```
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 $c_e$ in black segment $m$,
        If $(e\text{-}s+1) > LB_j$, stop and set $r_{je} = e\text{-}2$.
        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.

3.2. The second phase: DFS with branch and bound

After the first phase, a puzzle is not always solved completely. If some cells in the puzzle are still unknown, we will enter the second phase. Depth first search (DFS) is an exhaustive search, thus it will find out the solution of puzzle eventually. For this reason, we use DFS method to solve the unsolved puzzle. Since the general DFS is time-consuming, we will provide a "branch and bound" scheme to improve the processing speed.

One thing should be mentioned at first, we choose row information to build a DFS tree and use the column information to do verification as the method used in [3]. It means that every layer of the DFS tree is composed of row information and all nodes of each layer are the possible solutions (PS) for the corresponding row. Fig. 2 gives an example to do explanation. Fig. 2(a) is a given puzzle and Figs. 2(b) and (c) shows all possible solutions for row 1 and row 2, respectively. There are three possible solutions for row 1, two possible solutions for row 2, two possible solutions for row 3, and one possible solution for row 4. Fig. 2(d) shows the corresponding DFS tree of Fig.2(a).

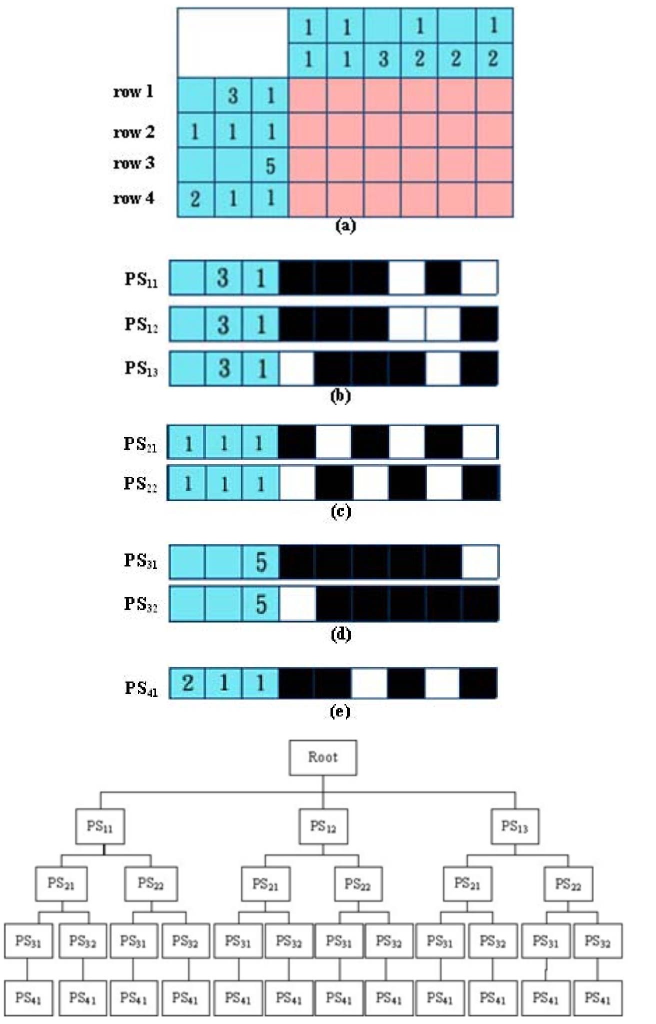

Figure. 2 An example of DFS. (a) A given puzzle. (b) Three possible solutions for row 1. (c) Two possible solutions of row 2. (d) Two possible solutions for row 3. (e) One possible solution for row 4. (f) The DFS tree of (a).

First, we try the first possible solution, PS$_{11}$, of row 1 and use column information to deduce that some cells in row 2 should be colored or left empty. Then we check all possible solutions of row 2 to determine which ones still are possible. If there is no possible solution in row 2, we skip all descendent nodes of PS$_{11}$ and then we try the next possible solution, PS$_{12}$, of row 1 and the same result is obtained. Thus, we proceed to try the last possible solution, PS$_{13}$, is still a possible solution of row 1 and find that the first possible solution, PS_{12}, of row 2. We then use the column information to deduce the cells in row 3. The similar process is applied, and PS_{31} is found not to be a possible one and its descendant is skipped. Then PS_{32} is found the possible solution. Finally, PS_{41} is checked and an answer is found. The above deduction and skipping process is called branch and bound scheme.

4. Experimental Results

In our database, there are about 260 puzzles. Most of them come from [4-5, 7] and few are created by us. A PC (CPU: AMD Athlon 2600+ 1.92GHz) and a NB (CPU: Intel Pentium M 2.00GHz) are used to run the proposed method.

Fig. 3 is some test images: Figs. 3(a) – (c) come from [4], Fig. 3(d) comes from [5], and Figs. 3(e) and (f) are created by us. Table 1 is the comparison of the experimental results between surveyed paper and our algorithm.

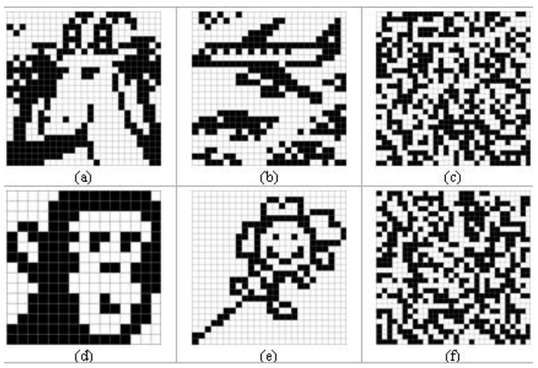

Figure 3. Test images. (a) Sheep (25x25). (b) Airplane (25x25). (c) Random_1 (30x30). (d) Monkey (15x15). (e) Sunflower (25x25). (f) Random_2 (30x30).

Table 1. The comparison of the experimental results between surveyed paper and our algorithm.

Puzzle size: Number (264 puzzles totally)		Execution Time		
		GA	DFS	Our Method
5x5: 1 5x6: 1		wrong answer > 1 min.	no solution < 0.07 sec.	no solution < 0.1 sec.
10x10: 2, 30x40: 1		wrong answer > 1 min.	no solution > 1 min.	
(above five puzzles have no solution)				
≦6x6: 7		0.5 sec. ～ 5 sec.	0.0 sec. ～ 0.6 sec.	0.0 sec. ～ 0.1 sec.
6x6 ～ 10x10: 9		about 30 sec.: 1 >1 min.: 8	<1 min.: 7 >1 min.: 2	about 0.1 sec.
10x10 ～ 15x15: 33 15x15 ～ 25x25: 110 ≧25x25: 100	Random_1	more than 2 days	more than 2 days	about 36 hr.
	Sunflower Random_2	> 1 hr.	> 1 hr.	about 20 min.
	Others	> 1 min.	> 1 min.	Owl: 2 sec. Skating: 16 sec. Others: about 0.1 sec.

5. Conclusions

In this paper, we have proposed a fast method to solve Japanese puzzles. The method contains two phases, the first phase uses logical rules to solve cells as many as possible. In the second phase, based on column information, DFS with branch and bound scheme is used to solve those unknown cells remained after the first phase. 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 branch and bound scheme. Furthermore, our method always provides correct solutions.

Acknowledgements

This work is supported in part by National Science Council (NSC 97-2221-E-009-137-.).

References

[1] N. Ueda and T. Nagao, “NP-completeness Results for NONOGRAM via Parsimonious Reductions,” Technical Report TR96-0008, Department of Computer Science, Tokyo Institute of Technology, May 1996.

[2] B. P. McPhail, “Light Up is NP-Complete,” Feb. 2005. URL: http://www.reed. edu/~mcphailb/lightup.pdf.

[3] K. J. Batenburg, “An Evolutionary Algorithm for Discrete Tomography,” *M.Sc. thesis in Computer Science*, University of Leiden, The Netherlands, 2003.

[4] K. J. Batenburg and W. A. Kosters, “A Discrete Tomography Approach to Japanese Puzzles,” *Proceedings of BNAIC*, pp. 243-250, 2004.

[5] W. A. Wiggers, “A Comparison of a Genetic Algorithm and a Depth First Search Algorithm Applied to Japanese Nonograms,” *Twente Student Conference on IT*, Jun. 2004.

[6] URL: http://www.pro.or.jp/~fuji/java/puzzle/nonogram/knowhow.html.

[7] Database of Japanese puzzles. URL: http://hattori.m78.com/puzzle/picture/java /stage_01/index.html and http://www.books.com.tw/exep/prod/booksfile.php ?it em=0010317755.