Available online at www.sciencedirect.com
ScienceDirect
Computers & Graphics 31 (2007) 750–760
COMPUTERS & GRAPHICS
www.elsevier.com/locate/cag

Technical Section

Automated generation and visualization of picture-logic puzzles

Emilio G. Ortiz-García[a], Sancho Salcedo-Sanz[a,*,1], Jose M. Leiva-Murillo[b], Angel M. Pérez-Bellido[a], José A. Portilla-Figueras[a]

[a]Department of Signal Theory and Communications, Universidad de Alcalá, 28805 Alcalá de Henares, Madrid, Spain
[b]Department of Signal Theory and Communications, Universidad Carlos III de Madrid, Madrid, Spain

Received 8 January 2007; received in revised form 17 July 2007; accepted 16 August 2007

Abstract

A picture-logic puzzle is a game that takes the form of an $N \times M$ grid, with numbers situated on the left of its rows and on the top of its columns, which give the clues for solving the puzzle. These puzzles have gained popularity in the last years all over the world, and there are companies involved in the commercialization of products related to them, mainly magazines, on-line puzzles via the web and games for mobile phones. The main problem with selling picture-logic puzzles is the need to generate a large number of picture-logic puzzles per month, or even per week, so automating the generation process of such puzzles is an important task. In this paper we propose algorithms for the automated generation of picture-logic puzzles (both black and white and color puzzles), starting from any digital RGB color image. We also present a visualization interface which provides several useful tools for finishing the puzzle generation process, like the possibility of changing colors and displaying the process of puzzle solution. We show the performance of the algorithms for generating picture-logic puzzles with several examples.

Keywords: Picture-logic puzzles; Puzzles solvers; Graphic interfaces; Automated puzzles generation; Algorithms; Heuristics

1. Introduction

Puzzles, board games and computer games in general have been studied for many years in the Computer Science, Mathematics and Artificial Intelligence fields, including in image processing [1–7]. Nowadays, the logic type puzzles (Sudoku, Picture-logic (Nonograms), Kakuro, Divide-by-squares, etc.) are quickly spreading over the world, and there are many companies involved in the process of commercializing these puzzles [8–10], including the most important newspapers of the world. In addition, puzzles and logic games in general are also sold to be played as games for mobile telephones or as web pages on the Internet, in a market with a turnover of millions of dollars per year.

The work we present here is related to one of this logic-type puzzles, specifically the one known as a *Picture-logic puzzle*, which has become popular in the last few years. There are several companies [8,9] that publish magazines and sell mobile phone games based on picture-logic puzzles. In general, these companies need an automated processes to generate hundreds or even thousands of puzzles per week. This generation process is, however, a difficult task, since it applies methods from image processing, optimization and heuristics.

1.1. Picture-logic puzzles

A picture-logic puzzle takes the form of an $N \times M$ grid, some of the cells of which are to be colored to form an image. In the grid there are numbers situated on the left of its rows and on the top of its columns. There are two types of picture-logic puzzles, black and white (B/W) and color puzzles. In B/W picture-logic puzzles [12–19], the numbers in the rows and columns represent, in the order they appear, how many cells in the corresponding row or column of the grid must be filled. If there are two or more numbers, each filled group of cells must be separated by at

*Corresponding author. Tel.: +34 91 885 6731; fax: +34 91 885 6699.
E-mail address: sancho.salcedo@uah.es (S. Salcedo-Sanz).
[1]This work has been partially supported by Universidad de Alcalá, through Projects CAM-UAH2005/019 and UAH PI2005/078.

0097-8493/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2007.08.006

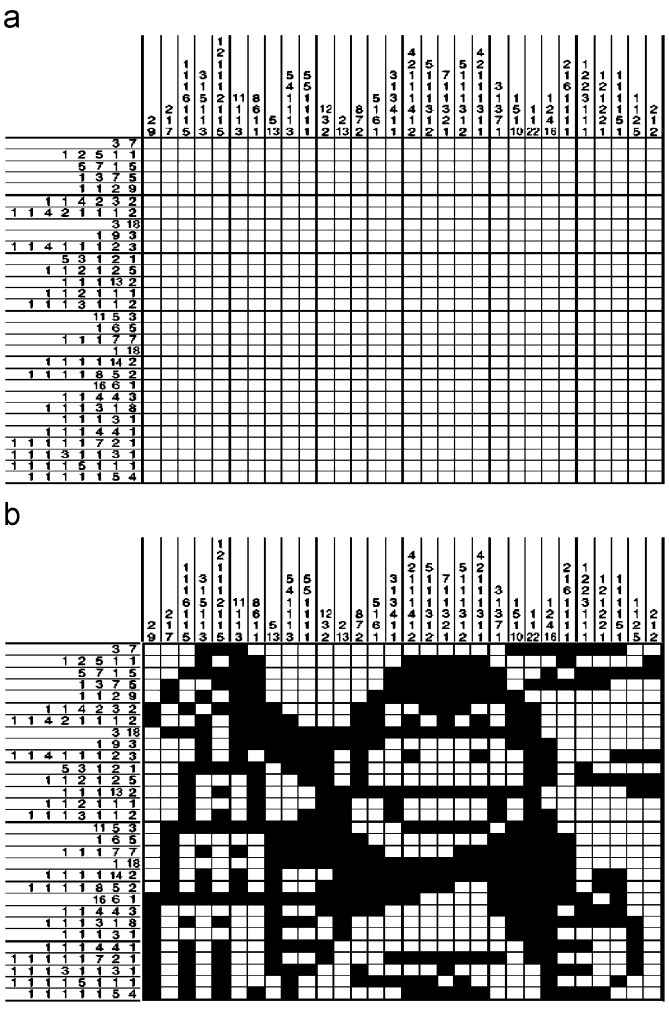

Fig. 1. Example of a B/W picture-logic puzzle; (a) grid with conditions; (b) solution.

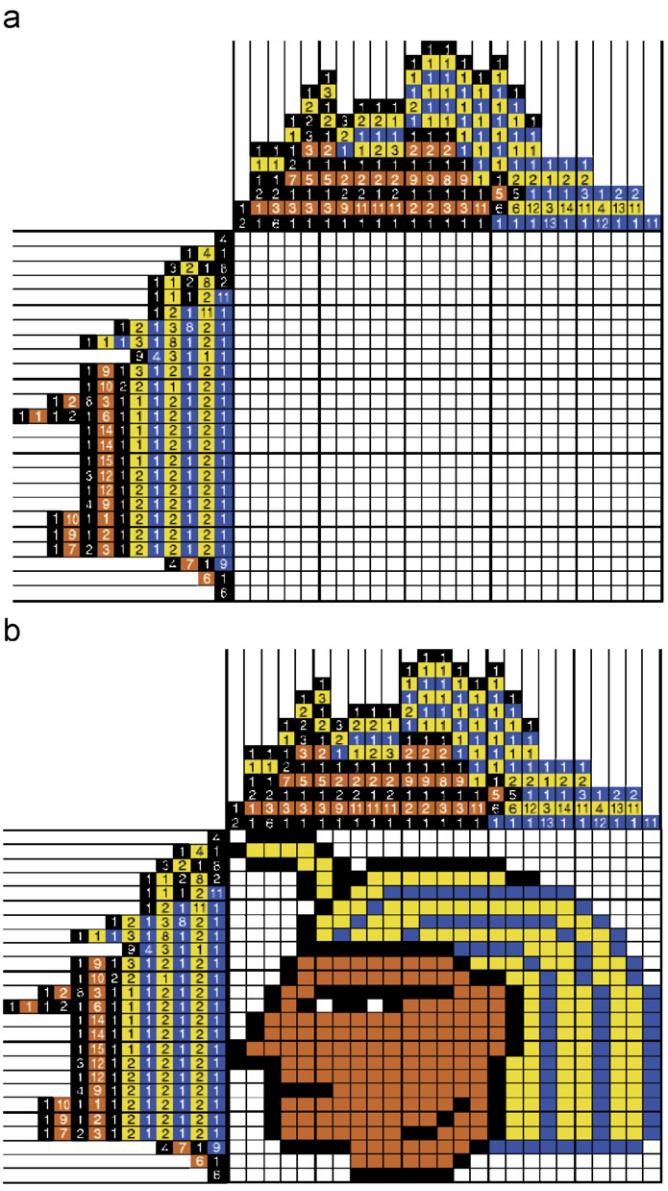

Fig. 2. Example of a color picture-logic puzzle; (a) grid with conditions; (b) solution.

least one blank. For example, Fig. 1(b) shows the solution to the picture-logic puzzle displayed in Fig. 1(a). Note that in the first row, 3 and then 7 cells must be filled, whereas in the second row 1, 2, 5, 1 and 1 cells should be filled, with at least one blank cell between them. Columns in the puzzle follow the same rules. When all the requirements in rows and columns are satisfied, the puzzle is solved, and a nice picture can be seen, in this case the picture of King-Kong on top of the Empire State Building in New York. Color picture-logic puzzles [8,20] follow similar rules to B/W puzzles, but taking into account that cells of the same color must be separated by at least one blank cell, and cells of different color may or may not be separated by blank cells. Fig. 2 shows an example of a color picture-logic puzzle (a), and its solution in (b).

Picture-logic puzzles were independently created by Non Ishida, a graphics editor, and Tetsuya Nishio, a professional puzzler, in 1987. Picture-logic puzzles are also known as *Japanese puzzles*. Soon after that, the puzzles started appearing in several Japanese puzzle magazines and gaining popularity, until 1990, when the newspaper *The Sunday Telegraph* started publishing them on a weekly basis. In the United Kingdom the name given to this kind of puzzles was *Nonograms* [11]. In the last few years the popularity of these puzzles has increased a lot. There are several companies which publish magazines and web pages exclusively devoted to picture-logic puzzles [8–10], in countries such as Spain, Germany, The Netherlands, Italy or Finland.

1.2. Paper objectives and structure

In this paper we propose algorithms for the automated generation of picture-logic puzzles, both B/W and color puzzles. In all cases a good solver is needed to generate the puzzles. Initial RGB images are processed prior to using the solver as a puzzle generator. Then, a threshold to decide the percentage of B/W cells in the puzzle is applied,

and a logic heuristic to solve the puzzle is used to check if a solvable puzzle has been obtained. The process of creating a color puzzle is more complex, and involves several procedures as image quantification, color reduction or using a mechanism to control the number of white pixels. Several heuristic algorithms are involved in all the processes, including a genetic algorithm (GA) [24] for the color reduction process. This paper also presents a visualization interface, which allows us to solve and display the solution of picture-logic puzzles, modifying the colors, and showing the process of solving the puzzle, in a dynamic way.

The rest of the paper is structured as follows: the next section presents the main existing algorithms for solving picture-logic puzzles. Section 3 describes the computer algorithms proposed in this paper for generating picture-logic puzzles, both for B/W and color puzzles. Section 4 shows the performance of our algorithms with several RGB image examples. Section 5 gives some final remarks.

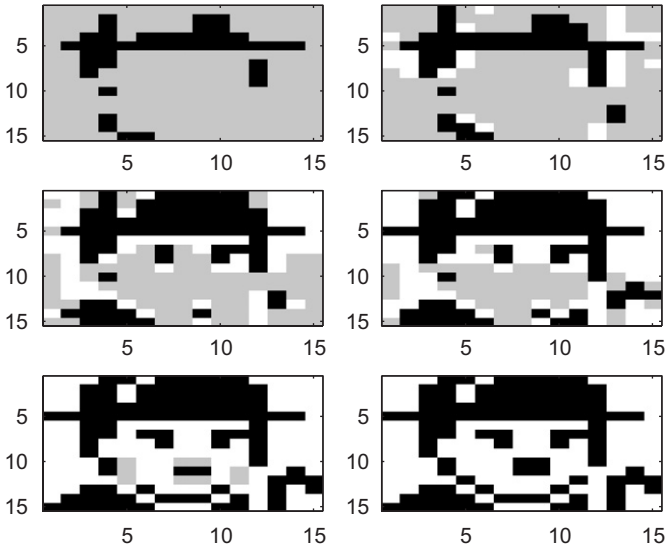

Fig. 3. Evolution of the solution given by the logic ad hoc heuristic in the benchmark puzzle "charlie", from [8].

2. Picture-logic puzzles solvers

Picture-logic puzzles are designed to be solved by people through the use of logical thinking. These puzzles must be unique-solution puzzles, so, if the reasoning of the solver is right, he will finally get the solution of the puzzle. The main idea in solving one of these puzzles is to discover cells in the grid which must be filled, or must be left blank, using the logic of the problem. There are several hints for solving picture-logic puzzles by hand explained in [14], which is a nice way to familiarize oneself with the puzzles and to understand the logic behind them. Some of these hints can be easily implemented in an algorithmic form, and the majority of these are implemented in the existing solvers.

Several picture-logic puzzles solvers can be found in the literature, the majority are for B/W puzzles. For example, in [15] a solver based on constraint programming is proposed for B/W puzzles. This solver is able to solve B/W puzzles very fast, but there is no color version of the algorithm available. In [16] another solver for B/W puzzles can be found. This solver includes a graphical interface where the user can check how the puzzle is being solved by the algorithm. There are also several attempts to use GAs to solve picture-logic puzzles. The first such attempt can be found in [17], where a solver based on a GA with operators based on discrete tomography was presented. Another paper dealing with GA for solving picture-logic puzzles can be found in [18], where a comparison between a GA and a depth first search algorithm for solving B/W puzzles is presented. The performance of a hybrid GA for picture-logic puzzles, and its application in education has been recently explored in [19]. In general, GAs are able to find solutions to small or medium size picture-logic puzzles, but with larger puzzles they fail to find a solution. In addition, the computational cost of GAs is much higher than that of other algorithms considered. Finally, in [20] a solver based on an ad hoc algorithm both for B/W and color puzzles has been proposed. The main properties of the solver isthat it is very fast, and uses similar procedures to solve B/W and color puzzles. As a small example of how this solver works, Figs. 3 and 4 show the evolution of this ad hoc algorithm in two picture-logic puzzles downloaded from [21]. The ad hoc algorithm works with a set of *unknowns* (cells which are not yet fixed as being filled or left blank are marked in gray in the figures), and progresses by fixing cells to be filled or blank, by means of a set of logic procedures (see Ref. [20] for details on this solver). This algorithm has been used as the solver in our paper.

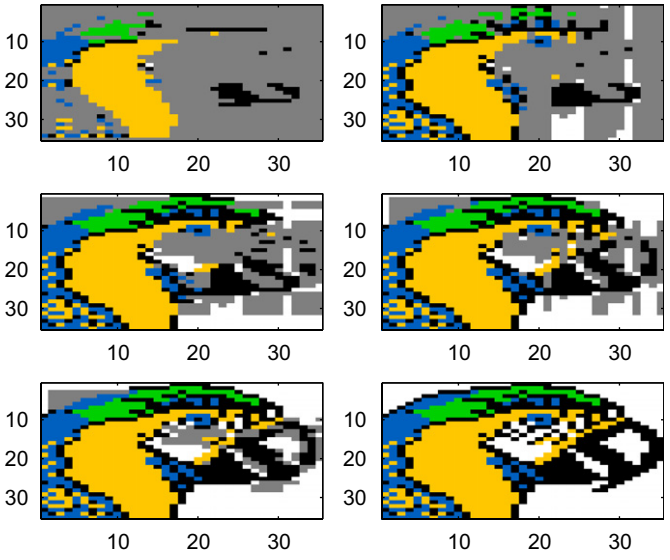

Fig. 4. Evolution of the solution given by the color logic ad hoc heuristic in the benchmark puzzle "parrot", from [8].

3. Algorithms for generating picture-logic puzzles from RGB images

This section presents our approaches for generating picture-logic puzzles starting from RGB digital images. First, we present algorithms for obtaining B/W picture-logic puzzles. Then, we present algorithms for the case of color puzzles.

3.1. Algorithms for B/W puzzles

The procedure to generate B/W puzzles is summarized in Fig. 5. Starting from an RGB color picture, the first step is its conversion to a grayscale image. This is achieved by a simple transformation into hue-saturation-luminance (HSL) values and by retaining the luminance value.

Once we have a grayscale image, a resizing must be carried out on the image to adjust it to the width and height constraints of the design of the puzzle. Basically we establish a priori either the width or the height of the puzzle, and then adjust the other parameter to preserve the image proportions. The resizing operation requires an interpolation criterion to map the pixel values in the original image to those in the resized image. In our case, we have made use of a simple nearest-neighbor interpolation [22]. The performance of the nearest-neighbor algorithm is not as good as a bilinear of bicubic interpolation algorithms when increasing the size of the image. In our case, we are reducing the size so the nearest-neighbor algorithm provides a computationally efficient yet good interpolation method.

After that, an optional 3×3 median filtering is carried out on the resized grayscale image. The median filtering is used in image processing to remove noise with certain non-gaussian statistics. Unlike low-pass filtering, median filtering is edge-preserving, which is important in order to maintain the sharpness of an image, and so its recognizability [23]. In our case, we apply it in order to get a certain color uniformity throughout the image. Thus, median filtering provides us with a *watercolor* effect that reduces the presence of blank pixels, which is the main source of insolvability in puzzles.

Finally, we apply a solver-guided binarization to convert the grayscale image into a B/W image. Basically, a threshold is applied so that pixels with a gray value above the threshold are set as white, while those below it are set to black. The threshold is tuned between a minimum gray value at which no fully black rows or columns exist, and the maximum at which no white rows or columns appear.

This B/W image is then used to codify the logic puzzle, which is the input of the solver. The output of the solver can be either a zero, if the puzzle has been correctly and completely solved, or a positive number indicating the number of ambiguous cells that cannot be solved. The tuning of the threshold allows us to obtain a set of solvable puzzles, ordered by their percentage of black cells. This percentage is extremely important in the puzzle design process, since it gives us a measure of:

- The effort required from the (human) user (the higher the number of black cells, the longer the time it takes to solve the puzzle, even when it is easy).
- The difficulty of the puzzle (a high percentage of black cells makes the puzzle easier because the clues are more straightforward).
- The recognizability of the picture: an appropriate percentage of black cells allows the user to recognize the content of the picture once the puzzle is finished.

Since some of these design criteria may be contradictory, a tradeoff among the constraints must be reached that guarantees recognizability, with a reasonable level of user effort and difficulty.

3.2. Algorithms for color puzzles

Algorithms for creating color picture-logic puzzles are slightly different than those for B/W puzzles. In this case the use of thresholds cannot be applied to generate the puzzle. Instead of using thresholds, a better strategy is to quantify the image. The procedure to generate color picture-logic puzzles is summarized in Fig. 6. In the following subsections we will describe the whole generation process, first we describe the process for color reduction and filtering of the image, and then we describe how to use the solver to discard non-solvable puzzles and obtain the final solution.

3.2.1. Color reduction and filtering

The first step is to resize the image to the size of the puzzle. This step is common to both B/W and color puzzle generation procedures. Each of the RGB pixels in the resized image can be expressed as a 3D point inside a $1 \times 1 \times 1$ cube. Since the puzzle consists of a finite number of

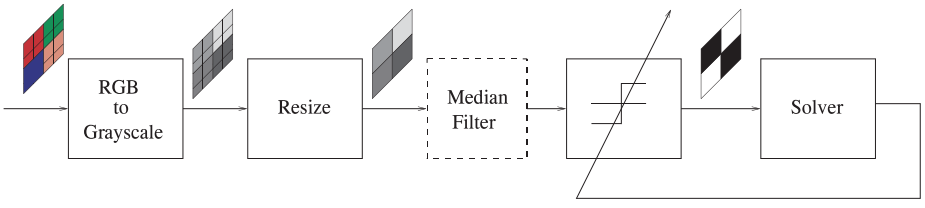

Fig. 5. Scheme of the B/W puzzle generation.

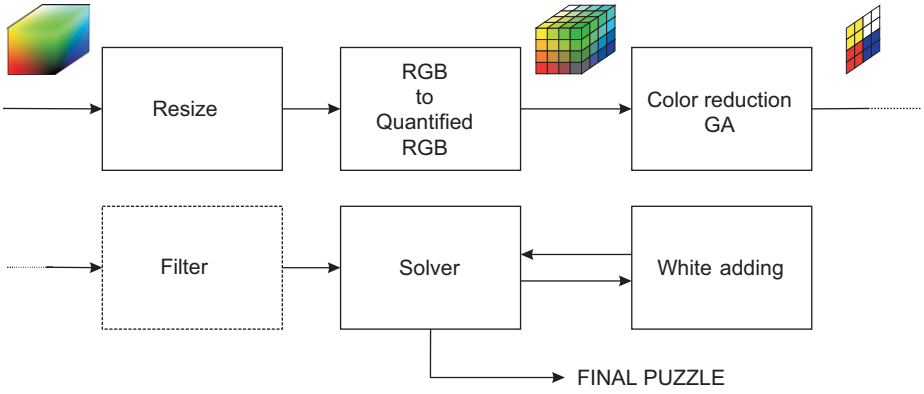

Fig. 6. Scheme of the color puzzle generation.

Fig. 7. Example of a line (row or column) in a color puzzle with no white color cell included. The solution of the line is straightforward.

colors (typically from three to five colors), this RGB cube must be discretized. This discretization (or color clustering) must make the final colors as representative and similar to the original ones as possible. We apply an intermediate discretization to a number of colors which is lower than the original one but higher than the final one. The objective of carrying out this discretization process is that an RGB color image may have a huge number of colors, and we wish to reduce those to a smaller set, more treatable for our purposes. We first divide the RGB cube in smaller cubes (micro-cubes), in a non-uniform way, and we associate the mean value of the colors inside it to each micro-cube. A special case is the micro-cube closest to the white color, which we associate with the color white instead of with the mean of the micro-cube. Usually a subdivision of the RGB cube in 64 micro-cubes (four micro-cubes in each direction (4^3)) is enough to obtain an image quite similar to the original one, but much more treatable to generate a picture-logic puzzle. In order to form the puzzle, we must choose C colors out of the 64 of our quantified image.

In our case there is an extra constraint: the white color must always be present in the image. This is because a picture-logic puzzle without any white cell can be solved in a straightforward way. Fig. 7 shows an example of a line (row or column) of a puzzle which does not contain any white color cell. This example shows that the white color cells are what makes the puzzle challenging in the case of color picture-logic puzzles.

We make a first attempt to include the white color in the puzzle by means of assigning the micro-cube closest to the white color directly to the color white (not to its mean color value). As was explained above, this process is able to introduce some white color in the puzzle depending on the colors of the original RGB image. We call the image obtained with this process the *quantified RGB image*.

In order to reduce the 64 colors[2] to C, we propose to solve a *p-median* problem [25], i.e. choose the C micro-cubes out of the 64 initial ones to be *medians*, such that a given cost is minimized. Note that the rest of the colors corresponding to the 64-C micro-cubes will be converted to the color of their closest median. In our case, the cost function to choose our C medians, is the Euclidean distance between the C colors associated with the micro-cubes and any other micro-cube in the image, multiplied by the number of pixels inside each micro-cube. After this process, we obtain an image we call the *color reduction image*. This color reduction can be tackled in a number of ways, but we have applied a GA (previously described in [25,28]) to solve the problem. The GA chooses the most representative set of colors of the image, and the puzzles resulting from their application are similar. Other approaches can be used to do this process, for example the ones described in [26,27]. Different approaches to obtain color reduction inRGB images can also be found in [29].

In general, the color reduction image obtained by means of the above process may not have any (or very few) white pixels. However, white cells are essential to control the difficulty of color picture-logic puzzles (puzzles without white cells are solvable in a straightforward way, as mentioned before). Then, an algorithm must be implemented in order to guarantee a percentage of white pixels exist in the image. We have called this process the *white inclusion procedure*.

[2]Note that 64 colors are the worst case, where each micro-cube has a contribution to the image. In the case that a micro-cube does not contribute to the color of the image, we do not take it into account, and we have less than 64 initial colors.

3.2.2. The white inclusion procedure

This procedure includes white pixels in the image by using a threshold method similar to the one described in the B/W puzzle generation section. Starting from the original image, we convert it to a grayscale image. A threshold is set and the pixels over the threshold are turned into white pixels. The rest of the pixels are assigned to the equivalent colors in the color reduction image. The threshold is modified until a pre-set percentage of the image is in the white color.

We complement this white color inclusion based on threshold with an easy way of including white pixels in some edges of the image: We locate puzzle conditions (numbers on the left-hand side or top of the grid) larger than a previously fixed value γ, i.e.

$$c_k \in W \quad \text{iff } c_k \geqslant \gamma, \tag{1}$$

where W stands for the set of conditions larger than γ, and c_k represents each of the conditions (numbers on the left-hand side or top of the grid).

We locate then the image pixels associated with the last cell in the conditions in W, and set them to white color. Note that using this process, the conditions in W are reduced by one unit:

$$\text{if } c_k \in W \rightarrow c_k = c_k - 1 \tag{2}$$

and also other conditions in the orthogonal row or column are modified accordingly.

As an example, consider Fig. 8. In Fig. 8(a), we show a color line (row or column) of a picture-logic puzzle. Let us consider $\gamma = 5$ for example. Then, only the first condition $c_k = 7$ belongs to W. We decrease this condition from 7 to 6, and fix the corresponding pixel in the image to white, which provides us the result shown in Fig. 8(b).

3.3. Application of a solver to obtain the final puzzle

After the color reduction and filtering process, we have a treatable image as input to the solver. This algorithm will try to solve the puzzle, and in the case where this is not possible, the algorithm produces an output indicating which cells do not have a final assigned color. The corresponding pixels in the image are then fixed to be white, and the solver is run again. Note that we also have the possibility of fixing the unsolved pixels to a given color different from white. This would also produce solvable puzzles, but our tests have shown that the conversion to white pixels gives good enough puzzles. This process is repeated until every cell in the puzzle has a color assigned to it.

4. Experimental part: examples

In this section we provide several examples of picture-logic puzzle generation. The next subsection discusses examples of the generation of B/W puzzles, and Section 4.2 deals with examples of the generation of color puzzles. Finally, Section 4.3 presents an interface for the visualization and modification of picture-logic puzzles.

4.1. Examples on the automated generation of B/W puzzles

First, we show several experiments that illustrate the sequence of operations for generating B/W puzzles (displayed in Fig. 5). We also stress the importance of the parameters involved in the automated generation of the puzzles. Three experiments are proposed: first, we discuss the validity of using median filtering; second, we perform the generation of puzzles with different sizes from the same picture; finally, we analyze the importance of the threshold tuning.

4.1.1. Median filtering

As explained in Section 3.1, a median filter helps us to obtain solvable puzzles from a picture that otherwise would not yield any.

Here we describe two experiments: In the first, the median filter is used to obtain solvable puzzles by median-filtering. In the second, we show a case in which the use of a median filter is undesirable.

The first picture consists of the famous Van Gogh self-portrait. We have constrained the width to be equal to 40 cells, so that the height is automatically set to 65 to preserve the proportion. The result of the different steps of the puzzle generation are shown in Fig. 9. It is obvious from the figure that the puzzle generated without filtering has a better resolution and provides a more detailed B/W image. However, note that in this case the puzzle is not solvable (see Fig. 9(e)). In fact, none of the puzzles generated by the tuning of the threshold from the non-filtered image were solvable. On the other hand, the puzzle generated from the median-filtered picture is solvable, although it is not as recognizable as the previous one.

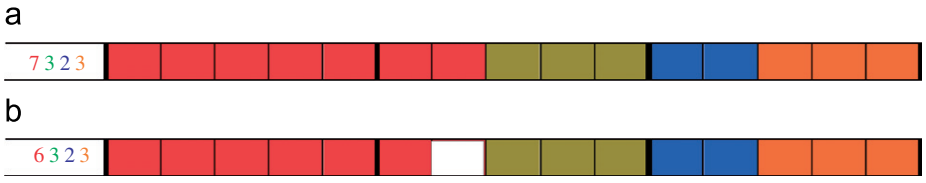

Fig. 8. (a) Example of a line (row or column) in a color puzzle with no white color cell included; (b) example of a white pixel inclusion by means of decreasing the first puzzle condition by one unit.

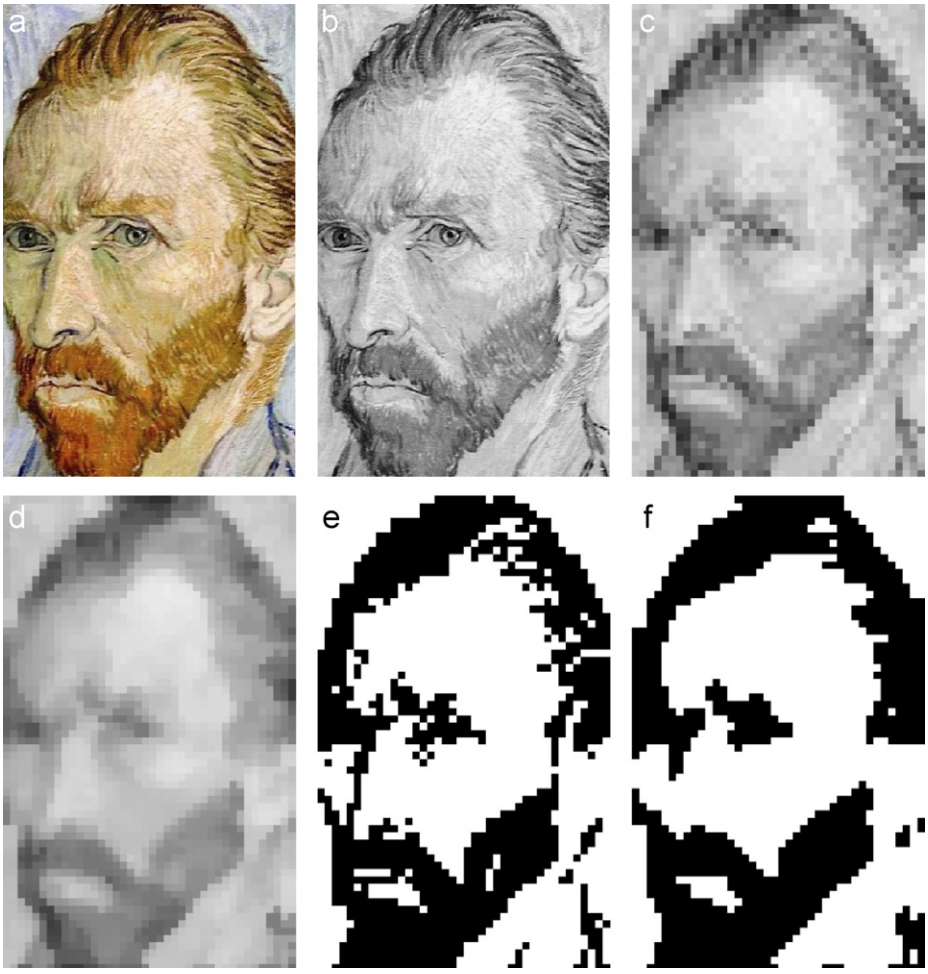

Fig. 9. Example of a positive use of median filtering. (a) Original picture; (b) grayscale picture; (c) resized picture; (d) median-filtered picture; (e) (unsolvable) puzzle generated by a binarization of figure (c); (f) solvable puzzle generated by a binarization of picture (d).

The second picture corresponds to a photograph of the actor Tom Cruise. The picture, together with the intermediate steps in the puzzle generation are depicted in Fig. 10. In this case, both generated puzzles are solvable. Now, we prefer to use the non-filtered puzzle because it allows us to obtain a more detailed puzzle.

4.1.2. Tuning the threshold

Next, some pictures are presented to show the effect of tuning the threshold when generating a puzzle. A caricature of Woody Allen is now used to generate the puzzles. Since the caricature is already a grayscale image, the first step of the scheme is not applied. The percentage of black color varies between 35.55% and 52.25%. As shown in Fig. 11, the threshold must be chosen to maximize the recognizability of the picture in addition to the solvability criteria.

Note that the recognizability-based criterion is subjective. The threshold that provides a recognizable B/W images with the lowest number of black pixels is preferable, unless an excessively difficult puzzle is generated. In that case, the solver algorithm can help us to measure the difficulty level by means of the number of iterations needed to reach the solution.

4.1.3. Different size puzzles from the same picture

We again make use of Tom Cruise's picture to show the effect of generating puzzles of different sizes. In Fig. 12, three puzzles with 30, 40 and 50 cells wide are displayed. The height is set accordingly to 35, 50 and 60 cells, respectively, to preserve the proportion. The images stress the intuitive fact that a larger size allows for a better resolution. When generating a puzzle, a tradeoff must be reached between the effort required by the human user (which is proportional to the size) and the recognizability of the generated picture.

The apparent distortion of the proportions of the 30×35 puzzle is due to the fact that it is a common practice to force both the width and the height of the puzzle to be a multiple of 5. This is done to make it easy for the user to count cells, but at small sizes this rounding may lead to a perceivable distortion.

The computational efficiency of our algorithm can be gauged from the fact that all the B/W puzzles presented in

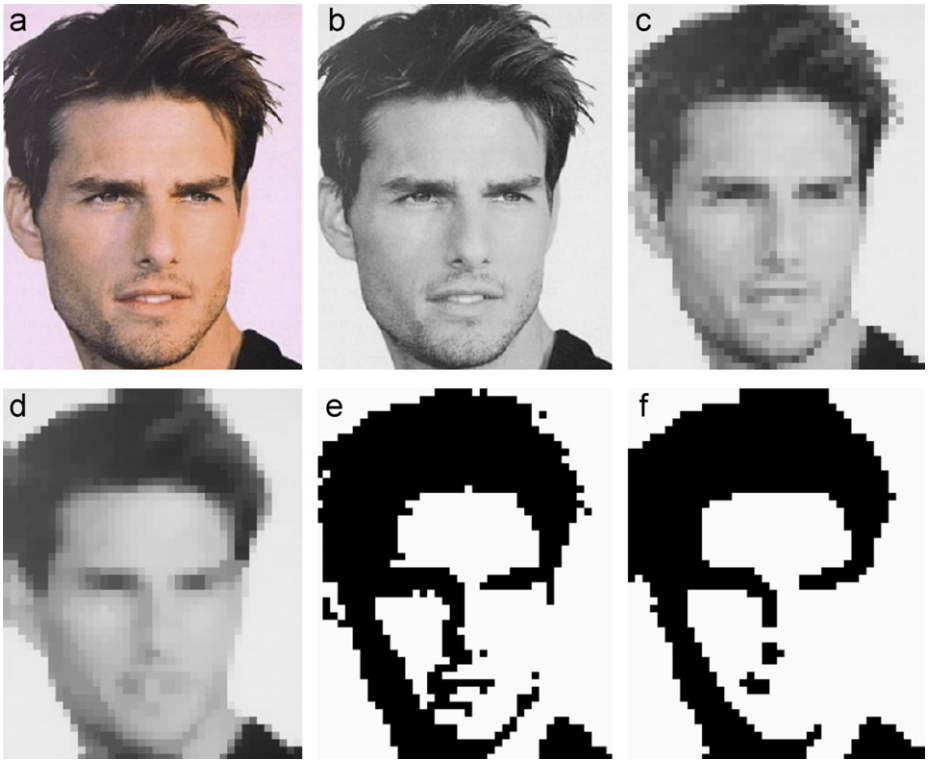

Fig. 10. Example of a negative use of median filtering. (a) Original picture; (b) grayscale picture; (c) resized picture; (d) median-filtered picture; (e) solvable puzzle generated by a binarization of figure (c); (f) solvable puzzle generated by a binarization of picture (d).

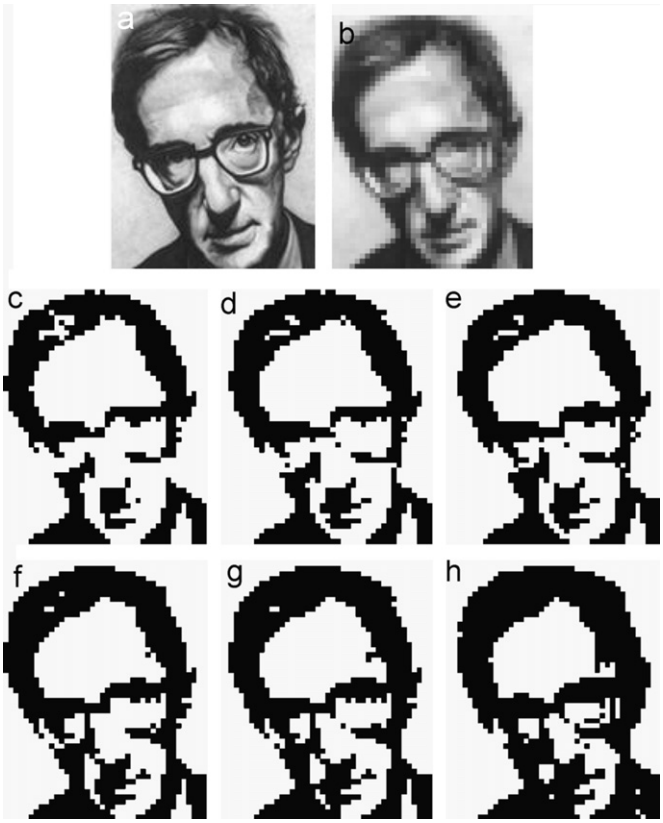

Fig. 11. Example of the effect of tuning the threshold. (a) Original, grayscale picture; (b) resized picture; (c)–(g) puzzles generated by binarization with threshold values 93, 101, 108, 120, 126 and 151, respectively.

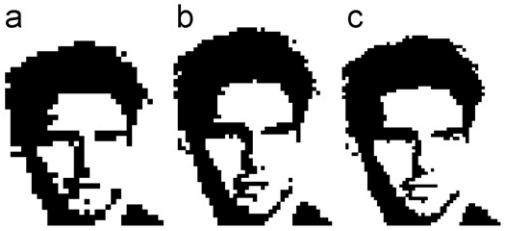

Fig. 12. Examples of puzzles with different sizes from the same picture: (a) 30 × 35 cells; (b) 40 × 50 cells; (c) 50 × 60 cells.

this paper were processed in less than 30 s by an implementation of our algorithm running on a Pentium IV 2.5 GHz processor.

4.2. Examples on the automated generation of color puzzles

We show several experiments that illustrate the generation of color picture-logic puzzles scheme in Fig. 6. Two experiments are presented: first, we show a case of an image where the percentage of white pixels is already over the desired one (we have set the minimum percentage of white pixels to be 15% of the image). Fig. 13(a) shows an initial RGB image of the actress Scarlett Johansson. Fig. 13(b) shows a resized image, and Fig. 13(c) displays the quantified RGB image. A maximum of 64 colors have been used to generate this image. The next step is to reduce the 64 colors of the quantified image to a maximum of five colors, to obtain the color reduction image. As was explained in Section 3.2.1, this process implies the solving

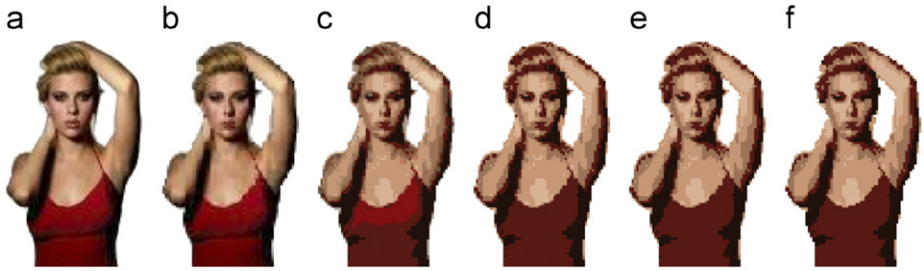

Fig. 13. Examples of picture-logic generation in an image with white pixels. (a) Original RGB image; (b) resized image; (c) quantified image; (d) color reduction image; (e) image after using the *white pixels inclusion procedure*; (f) final picture-logic puzzle.

Fig. 14. Examples of picture-logic generation in an image without white pixels. (a) Original RGB image; (b) quantified image; (c) color reduction image; (d) final picture-logic puzzle using the *white pixels inclusion procedure*.

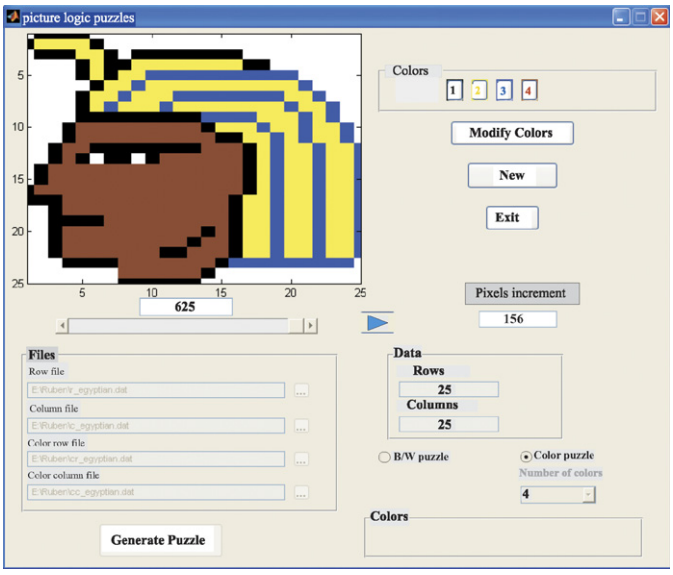

Fig. 15. Picture-logic puzzle of Fig. 2 solved and displayed in the visualization interface.

of a p-median problem. As was explained before, we have used a GA [25] with a local search to fix the number of colors to p. The resulting color reduction image is displayed in Fig. 13(d). Note that this image has already a percentage of white color pixels over 15% of the image. Thus the filter for adding white pixels described in Section 3.2 does not affect the image, as can be seen in Fig. 13(e). Finally, this image is passed to the solver, which produces the picture-logic puzzle displayed in Fig. 13(f).

The second example shows the case when the initial RGB image does not have white pixels. We have chosen an image of the actress Emmy Rossum, shown in Fig. 14(a). In this case, Figs. 14(b) and (c) show the quantified and color reduction images, respectively. Note that now the color reduction image does not have enough white pixels, so the filter to add white to the image has an important role to play in this case. Fig. 14(d) shows the final picture-logic puzzle obtained after the application of the *white pixels inclusion procedure* and the solver. It is easy to see that the white inclusion procedure works by turning to white the brightest pixels in the image.

The computational efficiency of our algorithm can be gauged from the fact that all the color puzzles presented in this paper were processed in less than 45 s by an implementation of our algorithm running on a Pentium IV 2.5 GHz processor. In general, the color puzzle generator is slower than the B/W puzzle generator working on the same image.

4.3. A picture-logic puzzle visualization interface

Once the picture-logic puzzle is created, there are still two main important points to solve: first, if we are considering color puzzles, it would be interesting to choose the final colors for the puzzle, since perhaps different colors from the starting picture may be required. Also, it would be interesting to be able to follow the process that a given solver uses to solve a picture-logic puzzle, since it could be an indicator of the puzzle's complexity.

In order to do these two final steps, it is necessary to have a visualization interface for the display of the puzzles. Though there are several visualization interfaces and toolkits described in the literature [30,31], there is no specific one for picture-logic puzzles. In this paper we present such a specific visualization interface for picture-logic puzzles. Our visualization interface has been

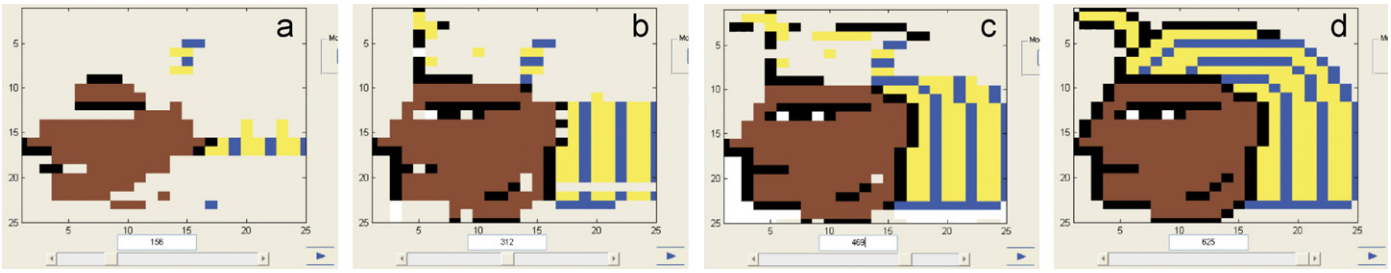

Fig. 16. Example of the "play" function of the visualization interface. Each sub-figure shows 156 new cells fixed by the solver.

programmed in Matlab (specifically, Matlab 7.0 has been used). It is able to solve and display B/W and color picture-logic puzzles. It allows us to change colors in the puzzle and it is also able to display the solution process of the solver. The latter provides an indication of the complexity of the puzzle for a human solver.

Once the type of puzzle has been selected (color or B/W), the visualization interface runs the solver described in [20], and the result of this solver is displayed, as shown in Fig. 15. In this case, the picture-logic puzzle given in Fig. 2 has been solved. The user can now modify the colors of the puzzle, by clicking in the corresponding color at the right-hand side of the picture.

Our visualization interface is able to display the process by which the algorithm solves the puzzle. We call this the "play" function (blue triangle in Fig. 15). The user can set how many cells are filled in each iteration of the play function (cell "pixels increment" in Fig. 15). Fig. 16 shows the process of solving the Egyptian picture-logic puzzle of Fig. 2, with 156 cells showed in each iteration.

With this visualization tool we are able to set the final puzzle's colors and have an estimation of the difficulty of the puzzle through the "play" function. The last step is to generate the final puzzle's grid, which can be done from this visualization tool using a different procedure also implemented in Matlab. This last step finishes the process of the automated creation of a picture-logic puzzle.

5. Conclusions

This paper presents a sequence of operations to generate picture-logic puzzles from RGB color images. We have proposed algorithms for generating B/W and color puzzles. Both approaches use a fast solver, previously used in the literature. Algorithms for B/W puzzles are based on image binarization and the use of thresholds for fixing the number of B/W cells. The process of color picture-logic puzzles generation is more complicated, and involves quantification, color reduction and a mechanism for adding white pixels to the image. A visualization interface tool for these puzzles has also been presented in this paper. The algorithms proposed in this paper can generate puzzles quite similar to the original pictures, and can be used in order to generate a very large number of picture-logic puzzles, needed for online gaming on the web, mobile telephone games or puzzle-devoted magazines.

Acknowledgments

The authors would like to thank professor J.A. Jorge, and the anonymous reviewers for their interesting comments, hints and assistance in the revision of this paper. The authors would also like to thank *Conceptis Puzzles* and *Pléyades Ediciones* for granting the permission to reproduce their copyrighted puzzles. The authors really appreciate the support and help of Dave Green, from *Conceptis Puzzles*.

References

[1] Laird JE. Using a computer game to develop advanced AI. IEEE Computer 2001;34(7):70–5.
[2] Van den Herik H, Iida H, editors. Games in AI research. Maastricht: Universiteit Maastricht; 2000.
[3] Vaccaro J, Guest C. Planning an endgame move set for the game RISK: a comparison of search algorithms. IEEE Transactions on Evolutionary Computation 2005;9(6):641–52.
[4] Jefferson C, Miguel A, Miguel I, Armagan Tarim S. Modelling and solving English Peg Solitaire. Computers & Operations Research 2006;33(10):2935–59.
[5] Smith K. Dynamic programming and board games: a survey. European Journal of Operational Research 2007;176(3):1299–318.
[6] Lucas S, Kendall G. Evolutionary computation and games. IEEE Computational Intelligence Magazine 2006;1(1):10–8.
[7] Khoo A, Zubek R. Applying inexpensive AI techniques to computer games. IEEE Intelligent Systems 2002;17(4):48–53.
[8] Conceptis Puzzles Inc., ⟨http://www.conceptispuzzles.com⟩
[9] Catchy Software Inc., ⟨http://www.catchysoft.com/jcwd.html⟩.
[10] Nikoli Software Inc., ⟨http://www.nikoli.co.jp/en/⟩
[11] Ishida N. Sunday Telegraph book of nonograms. Pan Publishers; 1993.
[12] Benton J, Snow R, Wallach N. A combinatorial problem associated with nonograms. Linear Algebra and its Applications 2006;412(1):30–8.
[13] Ueda N, Nagao T. NP-completeness results for nonograms via parsimonious reductions. Internal Report, Computer Science Department, University of Tokyo, 1996.
[14] Dorant M. A beginner's guide to solving picture forming logic puzzles ⟨http://www.conceptispuzzles.com\products\picapix\solving_a_puzzle.htm⟩.
[15] Duncan G. Puzzle solving. B.Sc. Degree Final Project Report, Computer Science Department, University of York, 1999.
[16] ⟨http://www.comp.lancs.ac.uk\computing\users\ss\nonogram\index.html⟩.

[17] Batenburg B, Kosters W. A discrete tomography approach to Japanese puzzles. In: Proceedings of the Belgian-Dutch conference on artificial intelligence, 2004. p. 243–50.
[18] Wiggers W. A comparison of a genetic algorithm and a depth first search algorithm applied to Japanese nonograms. In: Proceedings of the first Twente student conference on IT, 2004. p. 1–6.
[19] Salcedo-Sanz S, Portilla-Figueras J, Bellido APérez, Ortiz-García E, Yao X. Teaching advanced features of evolutionary algorithms using Japanese puzzles. IEEE Transactions on Education 2007;50(2):151–5.
[20] Salcedo-Sanz S, Ortiz-García E, Pérez.Bellido A, Portilla-Figueras J, Yao X. Solving Japanese puzzles with heuristics. In: IEEE symposium on computational intelligence and games, Honolulu, USA, April 2007.
[21] ⟨http://www.conceptispuzzles.com\products\picapix\puzzle_samples.htm⟩.
[22] Bracewell R. Two-dimensional imaging. Englewood Cliffs, NJ: Prentice-Hall; 1995.
[23] Lim J. Two-dimensional signal and image processing. Englewood Cliffs, NJ: Prentice-Hall; 1990.
[24] Goldberg D. Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley; 1989.
[25] Alp O, Erkut E, Drezner Z. An efficient genetic algorithm for the *p*-median problem. Annals of Operations Research 2003;122:21–42.
[26] Senne E, Lorena L, Pereira M. A branch-and-price approach to *p*-median location problems. Computers & Operations Research 2005;32(6):1655–64.
[27] Rosing K, Hodgson M. Heuristic concentration for the *p*-median: an example demonstrating how and why it works. Computers & Operations Research 2002;29(10):1317–30.
[28] Salcedo-Sanz S, Camps-Valls G, Pérez-Cruz F, Sepúlveda-Sanchis J, Bousoño-Calzón C. Enhancing genetic feature selection through restricted search and Walsh analysis. IEEE Transactions on Systems, Man and Cybernetics Part C 2004;34(4):398–406.
[29] Atsalakis A, Papamarkos N. Color reduction and estimation of the number of dominant colors by using a self-growing and self-organized neural GAs. Engineering Applications of Artificial Intelligence 2006;19(7):769–86.
[30] Ullrich T, Fellner D. AlgoViz—a computer graphics algorithm visualization toolkit. In: Proceedings of World conference on educational multimedia, hypermedia and telecommunications, Chesapeake, USA, 2004. p. 941–8.
[31] Dogrusoz U, Feng Q, Madden B, Doorley M, Frick A. Graph visualization toolkits. IEEE Computer Graphics and Applications 2002;22(1):30–7.