# Puzzle Generators and Symmetric Puzzle Layout

**Kozo Sugiyama[1], Ryo Osawa[1] and Seok-Hee Hong[2]**

[1] School of Knowledge Science, Japan Advanced Institute of Science and Technology, Asahidai 1-1, Tatsunokuchi, Nomi, Ishikawa, 923-1292, Japan

{Sugi, R-osawa}@jaist.ac.jp

[2] National ICT Australia; School of Information Technologies, The University of Sydney, Sydney, NSW 2006, Australia

shhong@it.usyd.edu.au

## Abstract

We present two abstract models of puzzles, *permutation* puzzles and *cyclic* puzzles, which can be modelled as *puzzle graphs*. Based on these models, we implement two *puzzle generators* and produce various layouts of the puzzles using graph drawing algorithms. Using these puzzle generators we can create new puzzles. Further, we can create new user interfaces of a puzzle by applying different layout algorithms. We implement a method for constructing symmetric layouts of puzzles, as symmetry is the most important aesthetic criteria for the puzzle layout.

*Keywords*: Puzzle generator, Symmetric layout

## 1 Introduction

Puzzles have sophisticated logical structures, beautiful shapes, and attractive user interfaces. To invent new puzzles and to design new interfaces of puzzles, we have carried out a systematic approach called '*divergence via abstraction*' (Maeda, Sugiyama and Mase 2002a, 2002b; Sugiyama, Hong and Maeda 2003; Sugiyama, Maeda and Mizumoto 2003; Sugiyama, Maeda, Osawa and Mizumoto 2005). The basic idea of the approach is illustrated in Figure 1. Here existing popular puzzles are abstracted and converted into other media such as graphs, blocks, sounds, and robots, while preserving their logic. Using this approach, we can create new puzzles and new interfaces for puzzles. Our approach closely relates to the creativity theory called '*Equivalent Transforming Thinking'* proposed by Ichikawa (Ichikawa 1970).

In this paper, we are mainly concerned with conversions of puzzles into *graphs*. More specifically, we consider two classes of puzzles, *permutation puzzles* (for example, *Rubik's cube*) and *cyclic puzzles* (for example, *Lightsout*). We analyse the operations of the puzzles and derive two abstract models that can be modelled as *puzzle graphs*. Based on these models, we implement two *puzzle generators* and produce various layouts of the puzzles using various graph drawing algorithms. Using these puzzle generators, we can parametrically change the levels of difficulty of the puzzles and create new puzzles. Further by applying various graph drawing algorithms, we can create new user interfaces for a puzzle. Hence, the puzzle layout is an interesting application for graph drawing. The puzzle layout should be very beautiful and attractive to the users. Further, the layout should operate as an interactive puzzle for the users. We apply various standard graph drawing methods (Sugiyama 2002) to produce puzzle layouts including the spring embedder algorithm, orthogonal drawing and visibility representations. Further, we observed that symmetry is one of the most important aesthetic criteria in the puzzle layout problem, as the puzzle graph is highly symmetric inherently.

In the next section, we present two abstract models of puzzles, which can be modelled as graphs. Based on the model, we present two puzzle generators and various layouts of the puzzles in Section 3. Then we present a method for constructing symmetric layouts of puzzles in Section 4. Section 5 concludes.

## 2 Abstract Models of Puzzles

### 2.1 Permutation Puzzles

Rubik's cube is one of the most popular puzzles and there are several variations such as the Megalinx and the Pyraminx (see Figure 2). They differ in shapes and the number of elements, but have similar structure. These are classified as *permutation puzzles* because the operations on the puzzles can be characterized as permutations between elements. Group theoretic descriptions and analysis for such puzzles can be found in (Turner and Gold 1982, Joyner 2002).

Rubik's cube consists of 3×3×3 blocks, also called as the $3^3$ Rubik's cube ($3^3$RC). For simplicity, we consider the $2^3$ Rubik's cube ($2^3$RC) that consists of 2×2×2 blocks. Figure 3(a) shows the $2^3$RC. The elements on the surface of the cube are numbered from 1 to 24. The $2^3$RC has 12 operations: 90° clockwise (or anti-clockwise) rotations of four blocks around the positive (or negative) direction of each axis. However, there are redundancies between the operations and it is sufficient to consider only three operations $x^+$, $y^+$ and $z^+$, where $x^+$ represents 90° clockwise rotation of the four blocks around the positive $x$-axis. We define the *operational redundancy* as a ratio between the numbers of sufficient and possible operations. For example, for the case of the $2^3$ RC, it is 9/12.

Operation $x^+$ can be denoted as *(4·4·4)-type expression*: a sequence of three permutations, each of length 4, where

each permutation is cyclic. If we insert the elements of the last permutation into the elements of the second permutations in the (4·4·4)-type expressions, we have $(4\cdot8^2)$-type expressions. Similarly, we can define $(12^3)$-type expression, as in Figure 3(b).

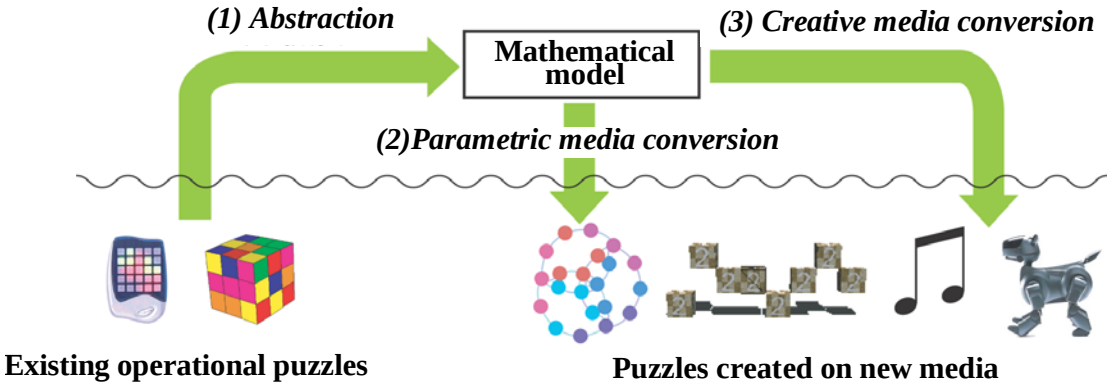
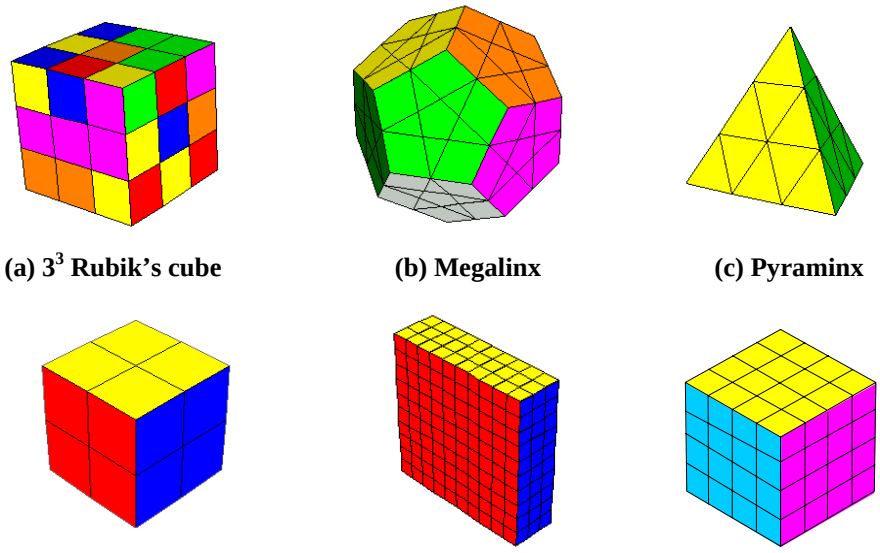


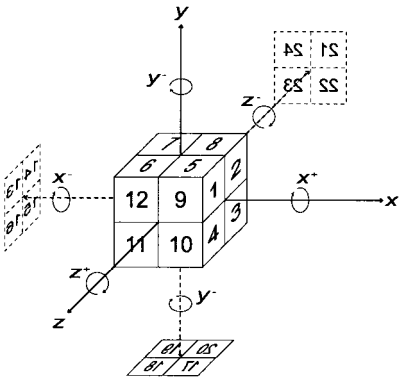

**Figure 1: "Divergence via abstraction" approach**



**(a) $3^3$ Rubik's cube** **(b) Megalinx** **(c) Pyraminx**

**(d) Other variations**

**Figiure 2: Examples of the permutation puzzles**



**(a)**

**Types of expressions**

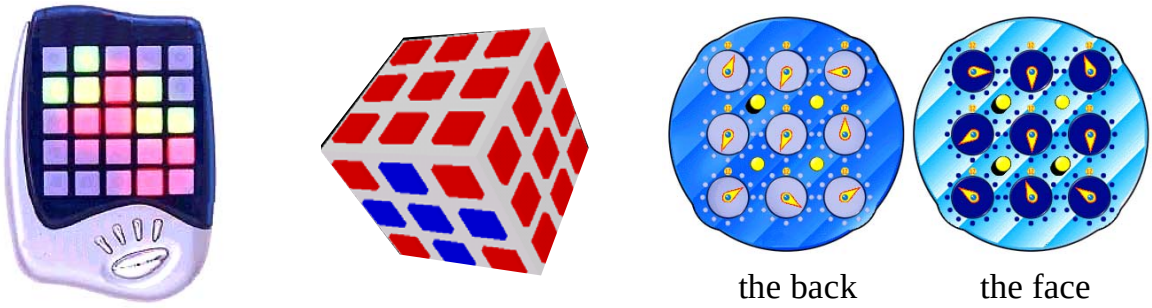
$$x^+ = (1\ 2\ 3\ 4)(5\ 21\ 20\ 10)(8\ 22\ 17\ 9) \quad \textbf{(4·4·4)}$$
$$= (1\ 2\ 3\ 4)(5\ 8\ 21\ 22\ 20\ 17\ 10\ 9)^2 \quad \textbf{(4·8²)}$$
$$= (1\ 5\ 8\ 2\ 21\ 22\ 3\ 20\ 17\ 4\ 10\ 9)^3 \quad \textbf{(12³)}$$

$$y^+ = (5\ 6\ 7\ 8)(2\ 9\ 13\ 24)(1\ 12\ 14\ 21) \quad \textbf{(4·4·4)}$$
$$= (5\ 6\ 7\ 8)(2\ 1\ 9\ 12\ 13\ 14\ 24\ 21)^2 \quad \textbf{(4·8²)}$$
$$= (2\ 1\ 5\ 9\ 12\ 6\ 13\ 14\ 7\ 24\ 21\ 8)^3 \quad \textbf{(12³)}$$

$$z^+ = (9\ 10\ 11\ 12)(1\ 17\ 16\ 6)(4\ 18\ 13\ 5) \quad \textbf{(4·4·4)}$$
$$= (9\ 10\ 11\ 12)(1\ 4\ 17\ 18\ 16\ 13\ 6\ 5)^2 \quad \textbf{(4·8²)}$$
$$= (9\ 1\ 4\ 10\ 17\ 18\ 11\ 16\ 13\ 12\ 6\ 5)^3 \quad \textbf{(12³)}$$

**(b)**

**Figure 3: Numbering on the $2^3$RC and types of expressions of operations**




**(a) Lightsout** **(b) Lightsout cube** **(c) Rubik's clock**

**Figure 4: Example of cyclic puzzles**

Based on the analysis of the $2^3$RC (similar analysis can be done for the cases of the Megalinx and the Pyraminx), we can derive an abstract model $M_p$ of permutation puzzles as follows:

$$M_p = (X, C, \varphi, R, \boldsymbol{s}, \boldsymbol{t}) \quad \text{where} \qquad (1)$$

- $X = \{1, 2,\ldots, n\}$: a set of $n$ elements;
- $\varphi : X \rightarrow C = \{c_1, c_2,\ldots, c_p\}$: a colour mapping. $C$ is a set of $p$ colours;
- $R = \{r_1, r_2, \ldots, r_m\}$: a set of $m$ operations, and

$$r_i = p_{i1}^{q_{i1}} p_{i2}^{q_{i2}} \ldots p_{ik_i}^{q_{ik_i}} \quad (i = 1,2,\ldots,m); \qquad (2)$$

- $\boldsymbol{s}$: an initial state obtained by randomly repeating operations from the goal state;
- $\boldsymbol{t}$: the goal state that is a sequence of the numbers labelled to the elements.

In (2), each permutation $p_{ij}$ corresponds to a cycle and therefore a set of operations can be expressed by a set of cycles, where each cycle represents a permutation. Thus we can define a *puzzle graph*, which consists of a set of cycles.

## 2.2 Cyclic Puzzles

We can classify the puzzles such as the Lightsout, the Lightsout cube, and Rubik's clock in Figure 4 as *cyclic puzzles*. This is because when we repeat the same operation on the puzzle, the state of the puzzle changes cyclically and returns to the original state after a fixed number of operations.

For example, Figure 4(a) shows the Lightsout which has 5×5 buttons with lights and Figure 4(b) shows the Lightsout cube which has 3×3×6 buttons with lights. The goal of the puzzle is to turn off all the lights or change all the colours of the lights to the same one. Pressing one button toggles the lights of 5 buttons; the one selected and its four neighbours.

The method for solving the Lightsout puzzle has been studied in (Aoki 1997); by pressing each button twice, the ON/OFF state returns to the original state. A solution can be determined whether each button is pressed once or not. Therefore, the Lightsout puzzle can be formulated as follows: for a given graph $G=(V, E)$, we define the matrix $A=[a_{ij}]$ as

$$a_{ij} = \begin{cases} 1 & (i = j \ \text{ or } \ e = (v_i, v_j) \in E) \\ 0 & (e = (v_i, v_j) \notin E) \end{cases} \qquad (3)$$

The matrix $A$ represents the toggling rules of the buttons. We denote the solution vector as $\boldsymbol{x} = \{x_1, x_2,\ldots, x_n\}$ where $x_i = 1$ (pushed) or $x_i = 0$ (not pushed). Further we denote an input vector as $\boldsymbol{b} = \{b_1, b_2,\ldots, b_n\}$ which represents an initial state of the buttons. To obtain a solution, we define an equation

$$A\boldsymbol{x} = \boldsymbol{b} \qquad \text{where} \qquad (4)$$

$$b_i = a_{1i}x_1 \oplus a_{2i}x_2 \oplus \cdots \oplus a_{25i}x_{25}, \ \ i = 1,\ldots,25. \qquad (5)$$

The equation $A\boldsymbol{x}=\boldsymbol{b}$ has a solution if and only if $r(A)=r([A\ \boldsymbol{b}])$ (Aoki 1997).

We denote the current state of the buttons as $\boldsymbol{y} = \{y_1, y_2,\ldots, y_n\}$. If $push(v_i)$ is performed, then the state $\boldsymbol{y}$ is replaced by

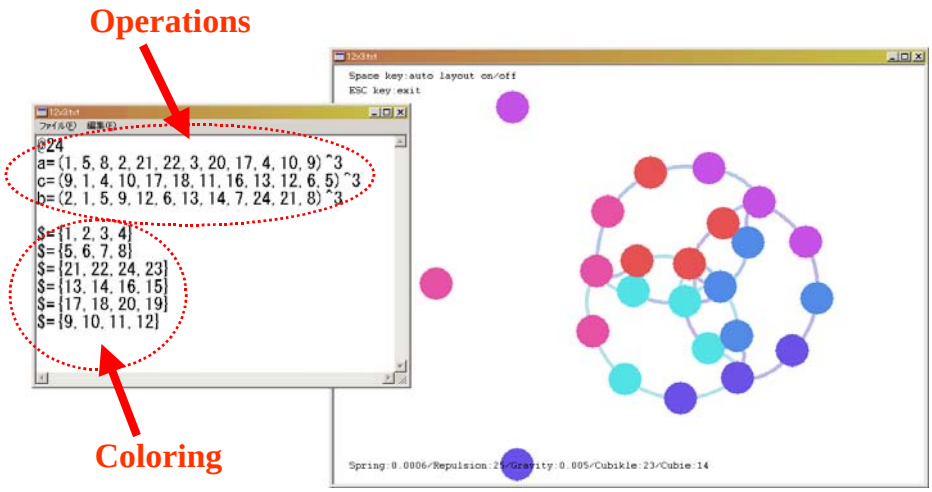
$$y_j \leftarrow y_j \oplus a_{ij} \,(\text{mod}2), \ j = 1, \ldots, 25. \qquad (6)$$

Figure 4(c) shows Rubik's clock, which has a more complicated structure. Each clock has 12 states, and ON/OFF states of four switches change toggling rules between clocks. There are 16 different sets of rules and three types of effects: positive, negative, and no effect. However, the basic structures of these three puzzles are similar. It has been shown that the operational redundancy of Rubik's clock is 112/128 (Maeda, Sugiyama, and Mase 2002b) while that of the Lightsout is 2/25 (Aoki 1997).

Based on the analysis, we can define an abstract model $M_c$ of cyclic puzzles as

$$M_c = (\boldsymbol{y}, q, A, R, \boldsymbol{b}, \boldsymbol{t}) \qquad \text{where} \qquad (7)$$

- $\boldsymbol{y} = [y_1, y_2,\ldots, y_m]$, $y_i \in \{1, 2,\ldots, q\}$: a vector of element states;




**(a)** **(b)**

**Figure 5: User interface of a permutation puzzle generator**

- $A=\{A_1, A_2, \ldots, A_p\}$: a set of $n$ x $m$ adjacency matrices;
  $R=\{r_1, r_2, \ldots, r_n\}$: a set of operations where
  $$r_i : y_j \leftarrow y_j \oplus a_{ij} (\text{mod } q), \quad i = 1, \ldots, n, \; j = 1, \ldots, m. \qquad (8)$$
- ***b:*** an initial state (or input pattern)
- ***t*** : the goal state.

## 3 The Puzzle Generators and Layouts of Puzzles

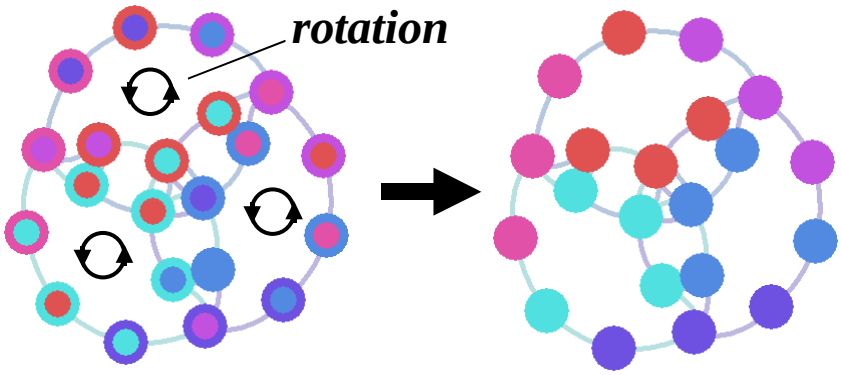
### 3.1 Permutation Puzzle Generator

Based on the model in Section 2, we implement a puzzle generator that allows a user to define their own puzzle. Figure 5 shows a user interface of the permutation puzzle generator for the case of the $(12^3)$-type expression of the $2^3$RC. Using the interface, puzzles can be defined in a text-based dialog box. To define a new puzzle, the user needs to input the number of elements, expressions for the operations, and a colour mapping. An example of defining a puzzle is shown in Figure 5(a).

We use a spring algorithm (Eades 1984) for the layout. However, we observe that it often fails to achieve a good symmetric layout. Thus we allow the user to interact with the drawing to improve the layout. An example of the layout is shown in Figure 5(b). To operate the layout as a puzzle, the user can simply use drag and drop to move an element or to rotate a cycle. Then the cycle that contains the element and the other cycles that are included in the operation rotate together (see Figure 6).

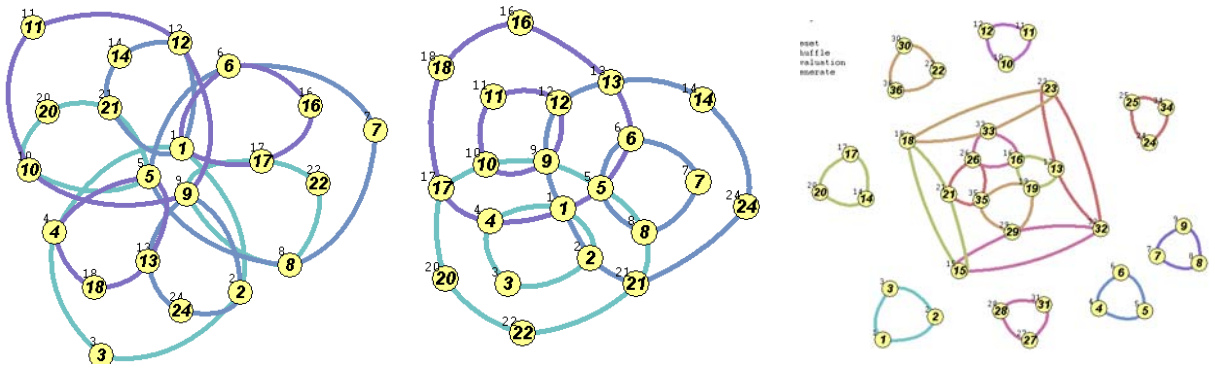
Figure 7(a) shows the layouts of different expressions, $(4{\cdot}4{\cdot}4)$ and $(4{\cdot}8^2)$, of the $2^3$RC. Figure 7(b) shows a layout of a puzzle graph that models the Pyraminx. Note that the graph is disconnected. Further, it can be shown that the longest expression of the $n^3$ Rubik's cube can be modeled as a planar graph for any $n \geq 2$.

Permutation puzzles such as Rubik's cube are sometimes too difficult to solve. Using our puzzle generator, we can create new puzzles with different level of difficulty. Figure 8 shows examples of the new puzzles with small size.




**(a)** **(b)**

**Figure 6: States of permutation puzzle $2^3$RC/$(12^3)$: (a) start (b) goal**



**(a)** **(b)**

**Figure 7: Layouts of $2^3$ Rubik's Cube (a) Pyraminx (b)**

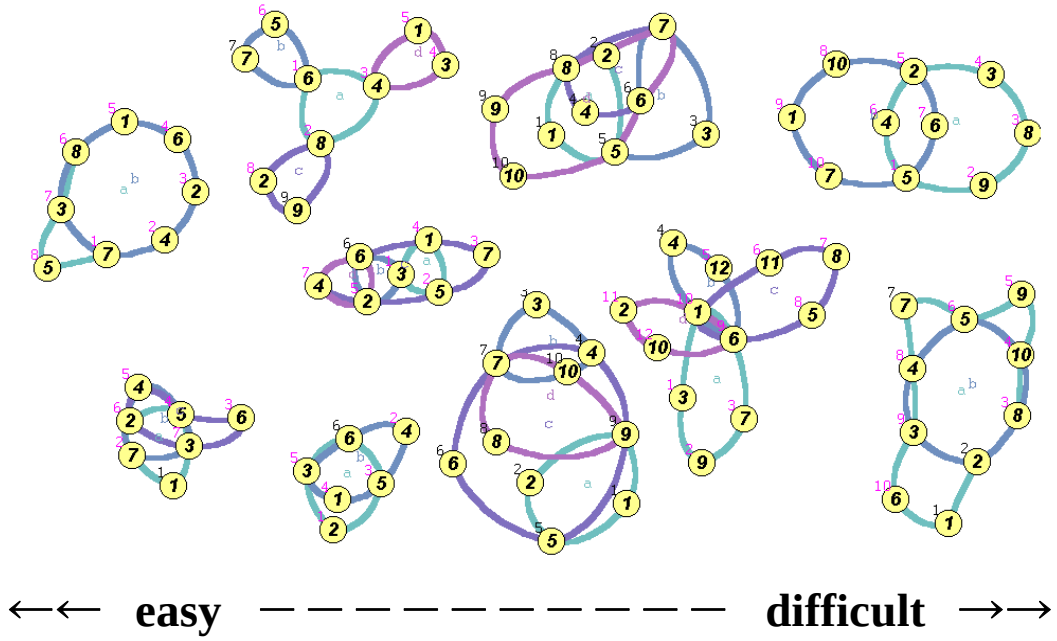
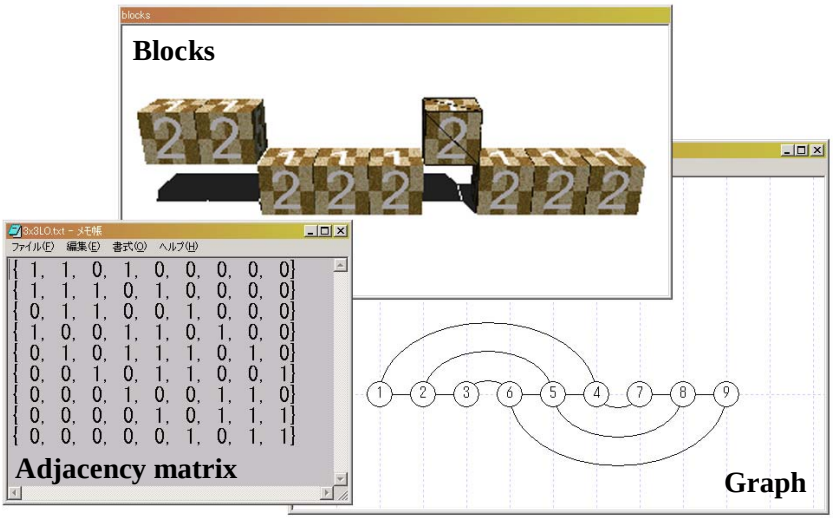



**Figure 8: New permutation puzzles and their layouts**

### 3.2 Cyclic Puzzle Generator

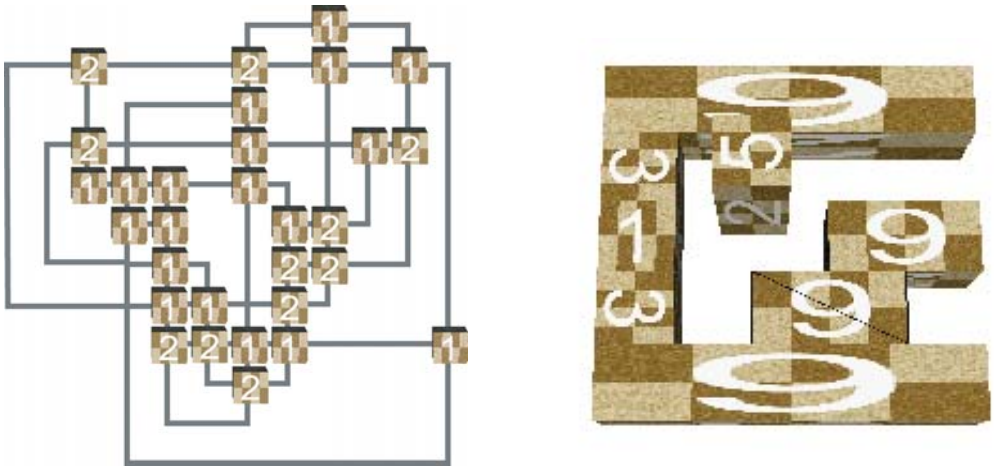
Figure 9 shows a user interface for the cyclic puzzle generator. A puzzle (i.e. toggling rules) is defined in Figure 9(a) and the rules are illustrated using L-mapping (Ozawa 1980) in Figure 9(b). An 'up and down box puzzle' is shown in the middle. Figure 9(b) shows variations of the layouts of cyclic puzzles which correspond to *n*-colour cyclic puzzles.

We further applied various graph drawing algorithms to achieve a variety of layouts for the cyclic puzzle. Figure 10(a) shows an orthogonal layout and Figure 10(b) shows a visibility representation (Otten and van Wijk 1978).




**(a)** **(b)**

**Figure 9: User interface of a cyclic puzzle generator**



**(a)** **(b)**

**Figure 10: (a) Orthogonal layout (b) visibility representations of cyclic puzzles**

## 4 Symmetric Layout of Puzzles

In this section, we describe a method for constructing symmetric layouts of puzzles. Our aim is to produce symmetric layouts of the puzzle graphs as in Figures 6 and 7 automatically without interaction. Thus, the main problem can be defined as follows.

**Symmetric Puzzle Layout Problem**

*Input: a puzzle graph G.*
*Output: a drawing D of G displaying maximum number of symmetries.*

In general the problem is NP-hard; however, heuristics have been proposed (Lipton, North and Sandberg 1985; de Fraysseix 1999; Carr and Kocay 1999). Further there is a linear time algorithm for outerplanar graphs (Manning and Atallah 1992) and planar graphs (Hong, McKay, and Eades 2002; Hong and Eades 2005). Unfortunately, these methods are quite complicated, as they involve implementation of many other complicated algorithms (Hong, McKay, and Eades 2002; Hong and Eades 2005). Therefore, we design our method based on the Tutte algorithm (i.e. barycenter method (Tutte 1963)), which can be implemented in $O(n^{1.5})$ time at best (Hara and Kaya 1984; Osawa 2004). Thus the main algorithm can be described as follows:

**Symmetric Puzzle Layout Algorithm**

1. Preprocess the puzzle graph *G* into a triconnected planar graph *G'*.
2. Choose the outer face and a regular polygon for the outer face.
3. Apply the Tutte algorithm (Tutte 1963) to produce a symmetric layout *D'* of *G'*.
4. Postprocess the layout *D'* to produce a layout *D* of *G*.

5. Go to Step 2 and repeat with different outer face for different layout, until the user is satisfied.

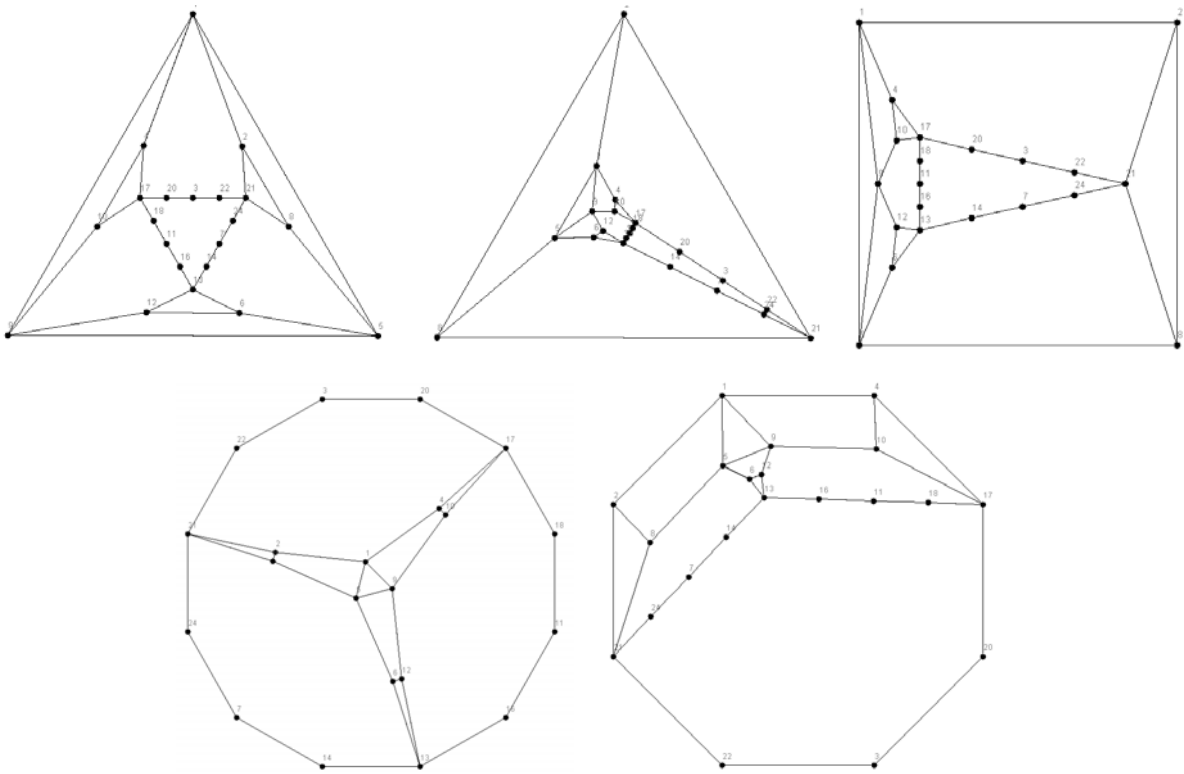


**Figure 11: Symmetric layouts of $2^3RC/12^3$**

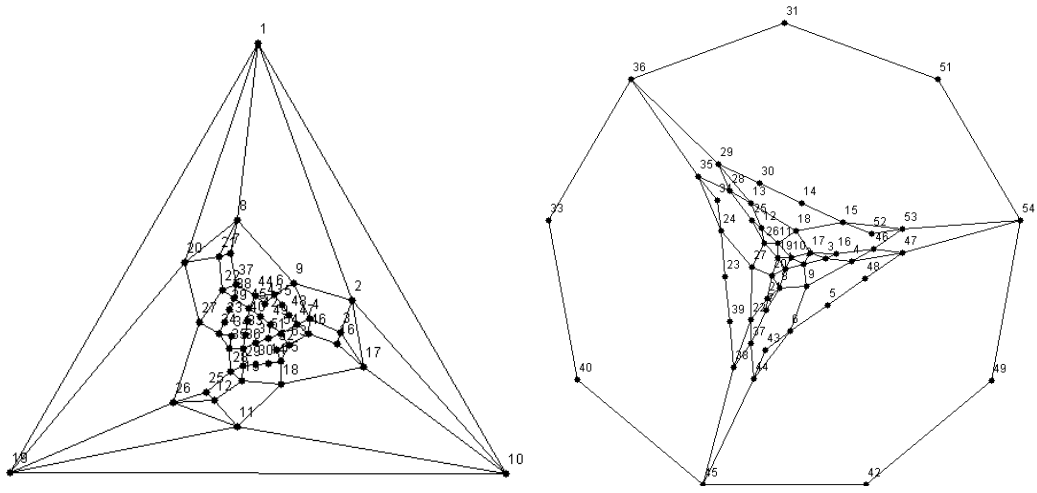
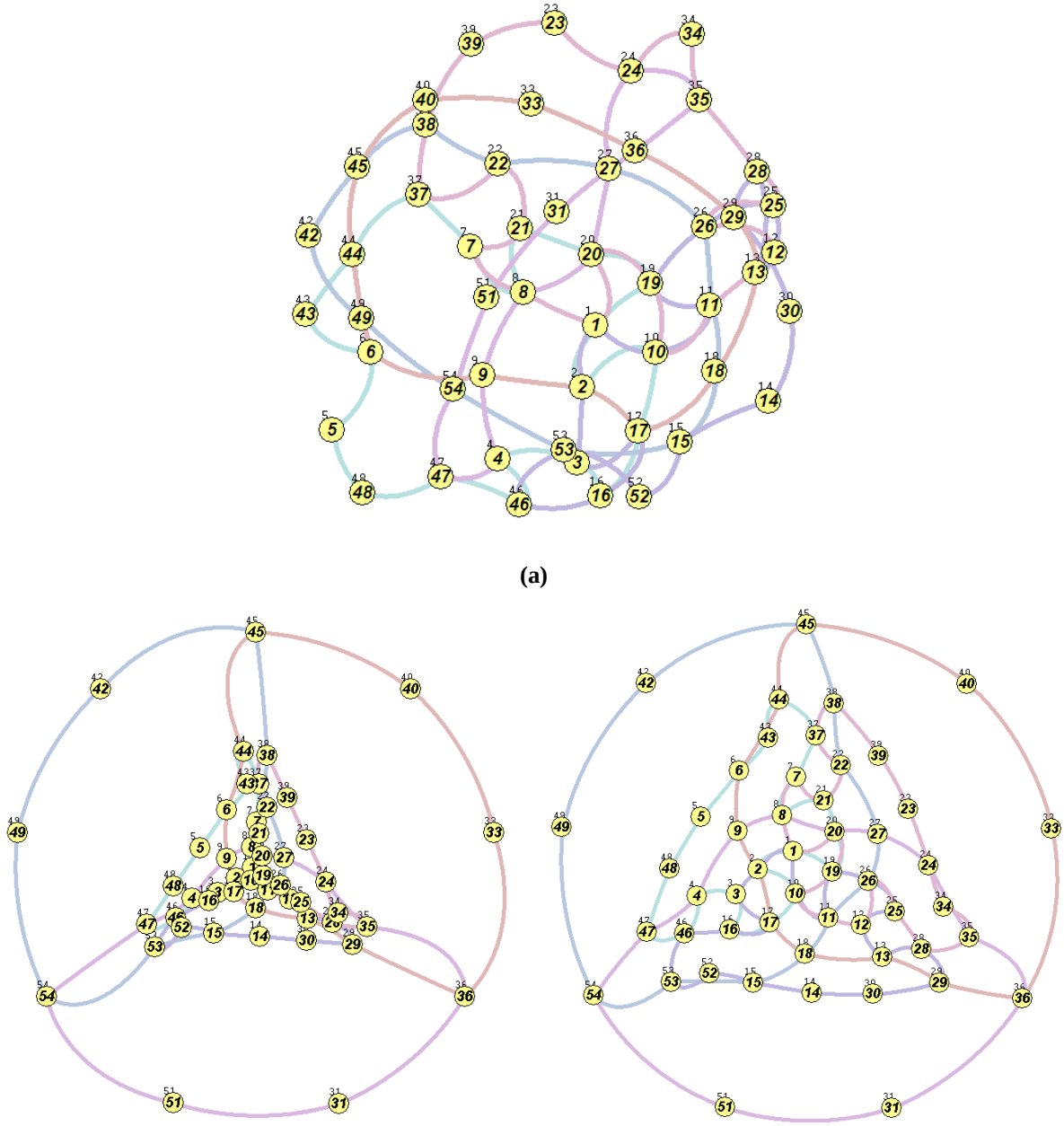


**Figure 12: Symmetric layouts of $3^3RC/20^5 \cdot 12^3$**

In Step 1, we preprocess the given puzzle graph $G$ to increase connectivity. Triconnectivity is required for the Tutte algorithm, but in general a puzzle graph is not triconnected. Thus we use a kind of augmentation method to increase connectivity. Suppose that a given puzzle graph $G$ is biconnected. Then we transform $G$ into triconnnected planar graph $G$' by deleting vertices of degree two and then treat multiple edges into a single edge. Step 2 is to choose the outer face of $G$' for the input of the Tutte algorithm.

For Step 3, to compute a symmetric layout $D$' of $G$', we used a library provided by *yFiles* (http:// www.yworks.de/index.htm).

In Step 4, we postprocess the layout $D$' to produce a layout $D$ for the puzzle graph $G$.



**(a)**

**(b)** **(c)**

**Figure 13: Various layouts for the $3^3RC/20^5{\cdot}12^3$**

Finally, we implement an interactive method to change the layout of the puzzle graph, by choosing the different outer face for the Tutte algorithm. Thus the user can choose his or her favourite symmetric layout for the puzzle interactively. We further implement the morphing between two different layouts with a slow animation, trying to preserve user's mental map.

Figure 11 shows various symmetric layouts of $2^3RC/12^3$ and Figure12 shows two symmetric layouts of $3^3RC/20^5{\cdot}12^3$.

Figure 13 shows various layouts of the permutation puzzle $3^3RC/20^5{\cdot}12^3$. Figure 13(a) is obtained by using the spring algorithm, Figure 13(b) is obtained by our algorithm, and Figure 13(c) is the final drawing improved by the puzzle browser. Sometimes we have a dense set of vertices in the middle of the layout as in Figure 12(b), due to the use of Tutte algorithm. We improved such layout by using the spring algorithm of the puzzle browser as in Figure 13(c).

## 5 Conclusion

We present a new application for graph drawing, the puzzle layout. We define abstract models for permutation and cyclic puzzles which can be modelled as puzzle graphs. Based on these models, we implement two puzzle generators and produce various layouts of the puzzles using various graph drawing algorithms. Using the puzzle generators, we create new puzzles. Moreover by applying

various graph drawing algorithms, we create new user interfaces for a puzzle with different attractions. We further implement a symmetric drawing algorithm for puzzles.

In this paper it is assumed that both planarity and symmetry are important aesthetics for the puzzle layout problem. However, further investigation for aesthetics of puzzle layout need to be carried out in the future.

Our ultimate goal is to develop a software that implements the puzzle generators and various layouts, so that the users can define their own puzzles and then communicate the puzzles with small communication devices, such as PDA or mobile phones.

## 6 References


Aoki, S. (1997): A location problem with variables on boolean field, B. Eng. Thesis in Information Engineering, Toyohashi University of Technology.

Carr, H. and Kocay, W. (1999): An algorithm for drawing a graph symmetrically, *Bulletin of the Institute of Combinatorics and its Applications*, 27:19-25.

Eades, P. (1984): A heuristic for graph drawing, *Congressus Numerantium*, 42:149-160.

de Fraysseix, H. (1999): An heuristic for graph symmetry detection, *Proc. of Graph Drawing 99*, Lecture Notes in Computer Science 1731:276-285, Springer Verlag.

Hara, Y. and Kaya, Y. (1984): Automatic graph construction techniques for systems with strongly connected components, *Proc. of Conf. of the Society of Instruments and Control Engineers*, 20(6):506-513 (in Japanese).

Hong, S. and Eades, P. (2005): A linear time algorithm for constructing maximally symmetric straight-line drawings of planar graphs, *Proc. of Graph Drawing 2004*, to appear.

Hong, S., McKay, B. and Eades, P. (2002): Symmetric drawings of triconnected planar graphs, *Proc. of SODA 2002*, 356-365.

Ichikawa, K. (1970): *The science of creativity*, Japan Broadcast Publishing Co., Ltd.

Joyner, D. (2002): *Adventures in Group Theory: Rubik's Cube, Merlin's Machine, and Other Mathematical Toys*, Johns Hopkins Univ. Press.

Lipton, R. North, S. and Sandberg, J. (1985): A method for drawing graphs, *Proc. ACM Symp. on Computational Geometry*, 153-160.

Maeda, A., Sugiyama, K. and Mase, K., (2002a): Media Conversion of permutation puzzles and development of permutation puzzles generators, *IPSJ SIG Notes Human Interface*, 101:33-40 (in Japanese).

Maeda, A., Sugiyama, K. and Mase, K. (2002b): Media conversion of cyclic puzzles and development of cyclic puzzle generators, *IPSJ SIG Notes Human Interface*, 101:40-47 (in Japanese).

Manning, J. and Atallah, M. J. (1992): Fast detection and display of symmetry in outerplanar graphs, *Discrete Applied Mathematics*, 39:13-35.

Otten, R. and van Wijk, J. G. (1978): Graph representations in interactive layout design, *Proc. IEEE Int Symp. Circuits and Systems*, New York, 914-918.

Osawa, R. (2004): Research and development on the layout problem of puzzle graph, Master Thesis, School of Knowledge Science, Japan Advanced Institute of Science and Technology.

Ozawa, T. (1980): Planarity testing for IC layout with constraints for pin order and congestion between pins, *IEEE Conf. Record of the 14th Asilomar Conf. On Circuit, Systems and Computers*, 188-192.

Sugiyama, K. (2002): *Graph Drawing and Applications for Software and Knowledge Engineers*, World Scientific.

Sugiyama, K., Hong, S. and Maeda, A. (2003): The puzzle layout problem, *Proc. of Graph Drawing*, Lecture Notes in Computer Science 2912:500-501, Springer Verlag.

Sugiyama, K., Maeda, A. and Mizumoto, A. (2003): 'Divergence via Abstraction': Practices for inventing new puzzles, *Proc. of The 4th International Conference on Systems Science and Systems Engineering*, 39-44.

Sugiyama, K., Maeda, A., Osawa, R., and Mizumoto A. (2005): Creating new puzzles : Practices of 'abstraction and conversion' strategy, *Journal of the Japan Creativity Society* (to appear).

Tuner, E. C. and Gold, K. F. (1982): Rubik's Groups, *American Mathematical Monthly*, 92(9).