The Puzzle Conversion and Layout Problem

Kozo Sugiyama[1], Seok-Hee Hong[2] and Atsuhiko Maeda[3]

1 School of Knowledge Science, Japan Advanced Institute of Science and Technology
2 School of Information Technology, University of Sydney
3 NTT Network Innovation Labs, NTT Corporation

July 1, 2003

KS-RR-2003-002

The Puzzle Conversion and Layout Problem

Kozo Sugiyama[1], Seok-Hee Hong[2], and Atsuhiko Maeda[3]

[1] School of Knowledge Science, Japan Advanced Institute of Science and Technology, Asahidai 1-1, Tatsunokuchi, Nomi, Ishikawa, 923-1292, Japan
Sugi@jaist.ac.jp
[2] School of Information Technologies, University of Sydney, NSW 2006, Australia
Shhong@it.usyd.edu.au
[3] NTT Network Innovation Labs, NTT Corporation, Hikarinooka 1-1, Yokosuka, Kanagawa, 239-0847, Japan
Maeda.atsuhiko@lab.ntt.co.jp

Abstract. We address the new problem of *puzzle conversion and layout* as a new application of graph drawing. We present two abstract models of puzzles, *permutation* puzzles and *cyclic* puzzles, which can be modeled 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 by applying different layout algorithms, we can create *new user interfaces* of a puzzle with different attractions. Finally, we discuss a method for constructing symmetric layouts of puzzles, as symmetry is the most important aesthetic criteria for the puzzle layout.

1 Introduction

We present a new, interesting application of graph drawing, *puzzle conversion and layout*. Puzzles have sophisticated logical structures, beautiful shapes, and attractive user interfaces. To make the world of puzzles much richer and to design new interfaces of puzzles, we have carried out a systematic approach called '*media conversion*' [1,2]. The basic idea of the approach is illustrated in Fig. 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.

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, the Rubik's cube) and *cyclic puzzles* (for example, the Lightsout). We analyze the operations of the puzzles and derive two abstract models which can be modeled 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 with different attractions. 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 a puzzle interactively by the users. We applied various standard graph drawing methods in [3, 4] to produce puzzle layouts. These include the spring embedder algorithm, orthogonal drawing and visibility representations. However, we observed that symmetry is the far 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 have the property that a graph representation can effectively be derived from them. Based on the model, we present two puzzle generators and various layouts of the puzzles in Section 3. Then we discuss a method for constructing symmetric layouts of puzzles in Section 4. Section 5 concludes.

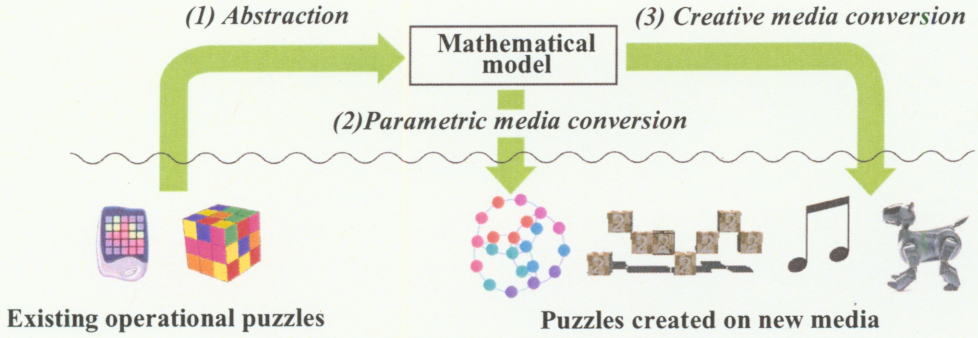
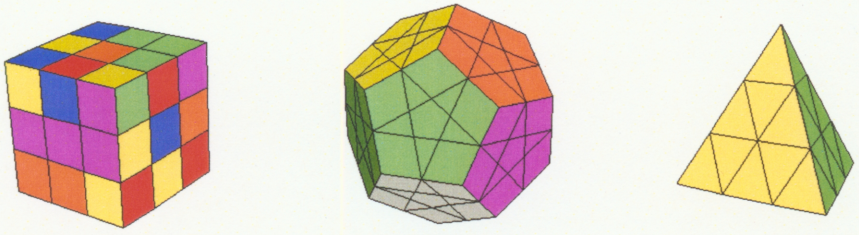


Fig. 1. Media conversion approach.

2 Abstract Models of Puzzles

2.1 Permutation Puzzles

The Rubik's cube is one of the most popular puzzles and there are several variations such as the Megalinx and the Pyraminx (see Fig. 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 [12,13].



(a) (b) (c)

Fig. 2. Examples of the permutation puzzles: (a) Rubik's cube (b) Megalinx and (c) Pyraminx.

Numbering and Color Mapping. The Rubik's cube consists of 3×3×3 blocks, also called as the 3^3 Rubik's cube. For simplicity, we consider the 2^3 Rubik's cube, which consists of 2×2×2 blocks. Fig. 3 shows the 2^3 Rubik's cube. The elements in the surface of the cube are numbered from 1 to 24. We use this numbering for the purpose of analysis. Note that only 6 different colors are enough to color the surface of the cube. The *color mapping* is one of the important factors characterizing the puzzle.

Operational Redundancy. The 2^3 Rubik's cube 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* of the 2^3 Rubik's cube as 9/12. Redundancy is another important factor for characterizing puzzles.

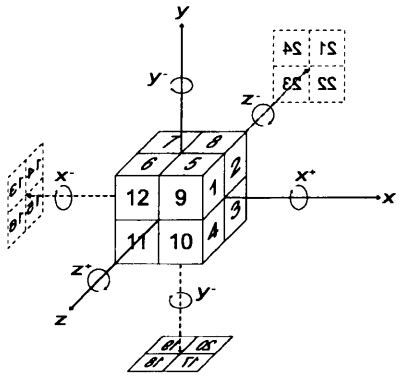


Fig. 3. The 2^3 Rubik's cube and a numbering.

Operation x^+ can be denoted as the following *expression*: a sequence of three permutations, each of length 4.

$$x^+ = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 2 & 3 \end{pmatrix}\begin{pmatrix} 5 & 21 & 20 & 10 \\ 10 & 5 & 21 & 20 \end{pmatrix}\begin{pmatrix} 8 & 22 & 17 & 9 \\ 9 & 8 & 22 & 17 \end{pmatrix} \qquad (1)$$

In (1), each permutation is cyclic. Hence x^+, y^+ and z^+ can be simply rewritten as

$$\begin{aligned} x^+ &= (1\ 2\ 3\ 4)(5\ 21\ 20\ 10)(8\ 22\ 17\ 9) \\ y^+ &= (5\ 6\ 7\ 8)(1\ 12\ 14\ 21)(2\ 9\ 13\ 24) \\ z^+ &= (9\ 10\ 11\ 12)(1\ 17\ 16\ 6)(4\ 18\ 13\ 5) \end{aligned} \qquad (2)$$

Types of Expressions. We define the expressions in (2) as (4·4·4)-type expressions, as each operation consists of three permutations of length 4. If we insert the elements of the last permutation into the elements of the second permutation in expressions (2), we have the following expressions.

$$\begin{aligned} x^+ &= (1\ 2\ 3\ 4)(5\ 8\ 21\ 22\ 20\ 17\ 10\ 9)^2 \\ y^+ &= (5\ 6\ 7\ 8)(2\ 1\ 9\ 12\ 13\ 14\ 24\ 21)^2 \\ z^+ &= (9\ 10\ 11\ 12)(1\ 4\ 17\ 18\ 16\ 13\ 6\ 5)^2 \end{aligned} \qquad (3)$$

We define these expressions as $(4{\cdot}8^2)$-type expressions. Similarly, (12^3)-type expressions can be defined as follows.

$$\begin{aligned} x^+ &= (1\ 5\ 6\ 2\ 21\ 22\ 3\ 20\ 17\ 4\ 10\ 9)^3 \\ y^+ &= (2\ 1\ 5\ 9\ 12\ 6\ 13\ 14\ 7\ 24\ 21\ 8)^3 \\ z^+ &= (9\ 1\ 4\ 10\ 17\ 18\ 11\ 16\ 13\ 12\ 6\ 5)^3 \end{aligned} \qquad (4)$$

Abstract Model. Based on the analysis of the 2^3 Rubik's cube (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:

$$M_p = (X, C, \varphi, R, s, t) \qquad \text{where} \qquad (5)$$

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

$$r_i = p_{i1}^{q_{i1}} p_{i2}^{q_{i2}} \ldots p_{ik_i}^{q_{ik_i}} \quad (i = 1,2,\ldots,m); \qquad (6)$$

- ***s***: an initial state obtained by randomly repeating operations from the goal state;
- ***t***: the goal state that is a sequence of the numbers labeled to the elements.

In (6), each permutation p_{ij} corresponds to a cycle and therefore a set of operations can be expressed by a set of cycles, as illustrated in Fig. 4. Here, each cycle represents a permutation. Thus we can define a *puzzle graph*, which consists of a set of cycles.

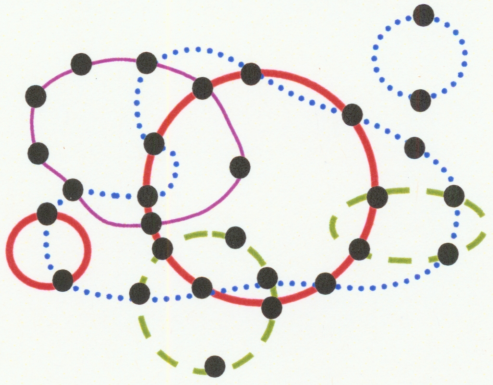


Fig. 4. An illustration of a puzzle graph.

2.2 Cyclic Puzzles

We can classify the puzzles such as the Lightsout, the Lightsout cube, and the Rubik's clock in Fig. 5 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.

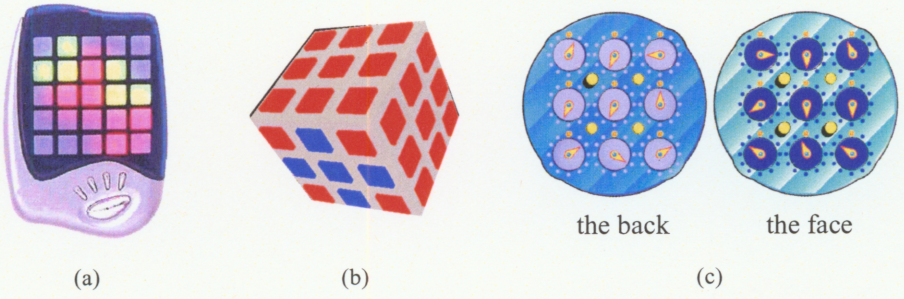


Fig. 5. Example of cyclic puzzles: (a) Lightsout, (b) Lightsout cube, and (c) Rubik's clock.

For example, Fig. 5(a) shows the Lightsout which has 5×5 buttons with lights and Fig. 5(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 colors of the lights to the same one. Pressing one button toggles the lights of 5 buttons; the one selected and its four neighbors.

The method for solving the Lightsout puzzle has been studied in [5]; 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

$$a_{ij} = \begin{cases} 1 & (i = j \text{ or } e = (v_i, v_j) \in E) \\ 0 & (e = (v_i, v_j) \notin E) \end{cases} \tag{7}$$

The matrix A represents the toggling rules in regards to 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

$$A\boldsymbol{x}=\boldsymbol{b} \qquad \text{where} \qquad (8)$$

$$b_i = a_{1i}x_1 \oplus a_{2i}x_2 \oplus \cdots \oplus a_{25i}x_{25},\ i=1,\ldots,25\,. \qquad (9)$$

The equation $A\boldsymbol{x}=\boldsymbol{b}$ has a solution if and only if $r(A)=r([A\ \boldsymbol{b}])$[5].

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

$$y_j \leftarrow\ y_j \oplus a_{ij} (\text{mod}2),\quad j=1,\ldots,25. \qquad (10)$$

Fig. 5(c) shows the Rubik's clock, which has more complicated structure. Each clock has 12 states, and ON/OFF states of four switches changing 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 the Rubik's clock is 112/128 [2] while that of the Lightsout is 2/25 [7].

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

$$M_c = (\boldsymbol{y}, q, A, R, \boldsymbol{b}, \boldsymbol{t}) \qquad \text{where} \qquad (11)$$

- $\boldsymbol{y}=[y_1, y_2,\ldots, y_m], y_i \in \{1, 2,\ldots, q\}$: a vector of element states;
- $A=\{A_1, A_2,\ldots, A_p\}$: a set of n x m adjacency matrices;
- $R=\{r_1, r_2,\ldots, r_n\}$: a set of operations where

$$r_i : y_j \leftarrow\ y_j \oplus a_{ij} (\text{mod}\ q),\quad i=1,\ldots,n,\quad j=1,\ldots,m. \qquad (12)$$

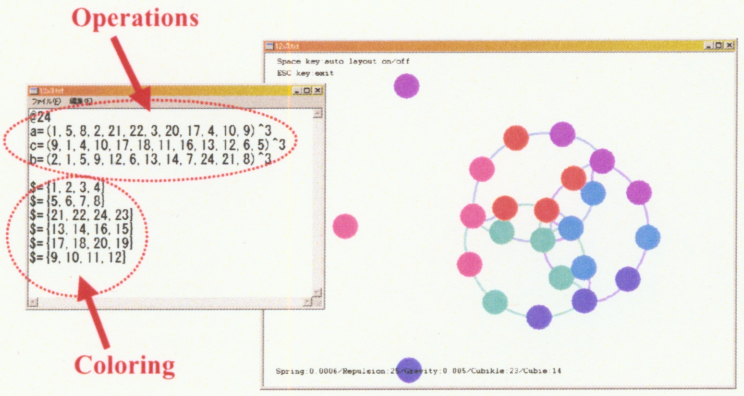
- $\boldsymbol{b}$: an initial state (or input pattern)
- $\boldsymbol{t}$: the goal state.

3 The Puzzle Generator and the Layouts of Puzzles

3.1 Permutation Puzzles

Permutation Puzzle Generator. Based on the model, we implement a puzzle generator which allows a user to define their own puzzle. Fig. 6 shows a user interface of the permutation puzzle generator. 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 color mapping. An example of defining a puzzle is shown in Fig. 6(a).

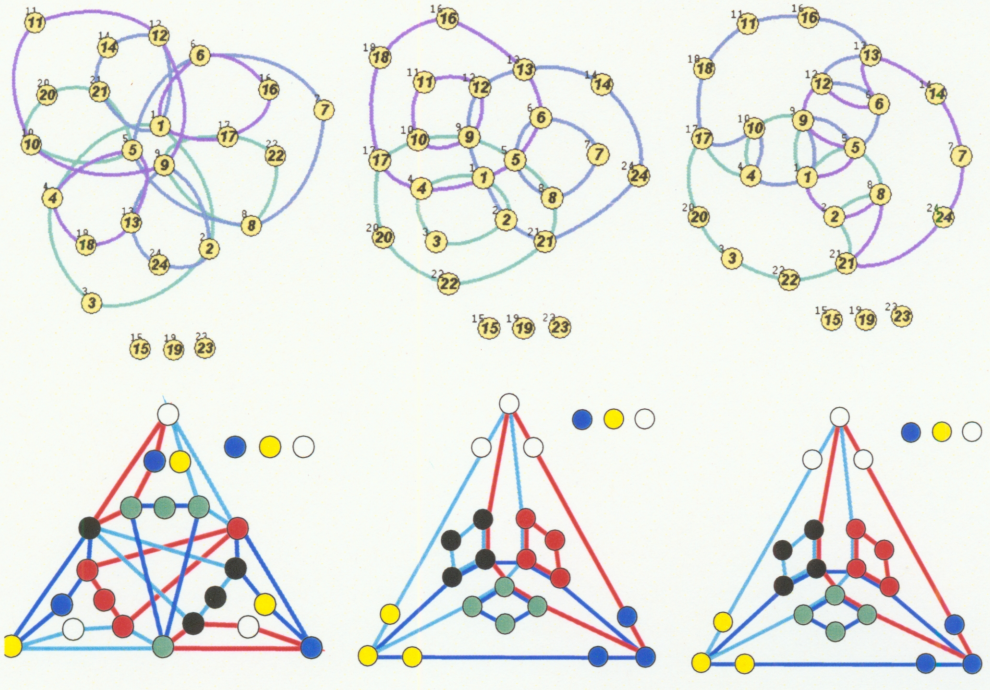
We use a spring algorithm for the layout. However, we observe that it sometimes fails to achieve a good symmetric layout. Thus we allow the user to interact with the drawing to improve the layout. An example of the layout is shown in Fig. 6(b). To operate the layout as a puzzle, the user can simply use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



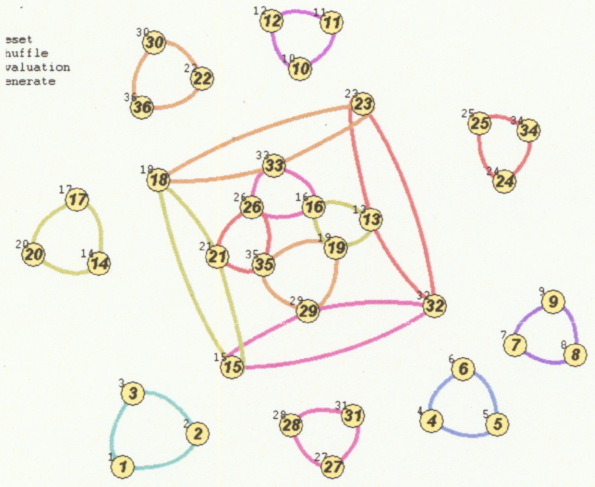
(a) (b)

Fig. 6. User interface of a permutation puzzle generator and its layout.

Layouts of the 2^3 Rubik's Cube and the Pyraminx. Fig. 7(a) shows the layouts of three different expressions of the 2^3 Rubik's Cube. In Fig. 7(a), the upper three layouts are produced by a spring algorithm and the lower three layouts are drawn manually. Note that all the layouts are symmetric and the graphs corresponding to the $(4{\cdot}8^2)$-type and the (12^3)-type expressions are planar. Fig. 7(b) shows a layout of a puzzle graph which models the Pyraminx. Note that the graph is disconnected.



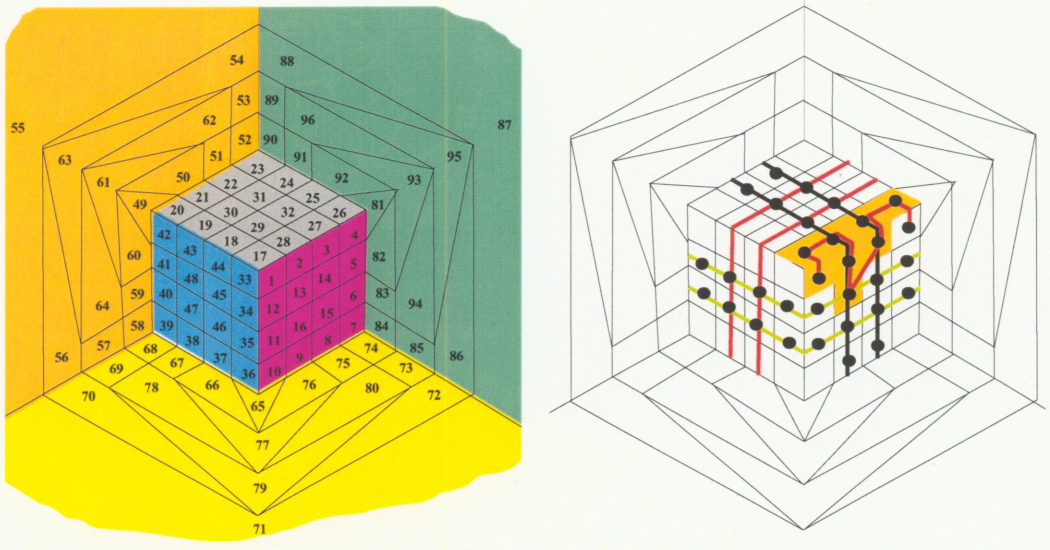
(a)



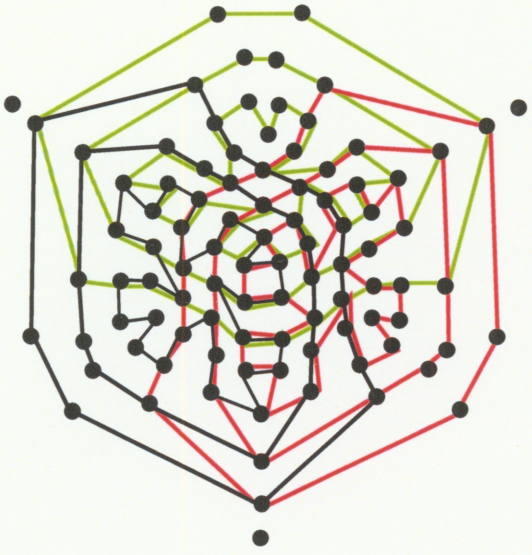
(b)

Fig. 7. Layouts of (a) 2^3 Rubik's Cube and (b) Pyraminx.

Layouts of the n^3 Rubik's Cube. Note that the longest expression of the n^3 Rubik's cube can be modeled as a planar graph for any $n \geq 2$. Fig. 8(a) shows a tiling of the 4^3 Rubik's cube and Fig. 8(b) shows how to route the tiling without edge crossings. Fig. 8(c) shows the corresponding drawing. This method can be applied to the case of any n in a similar way.



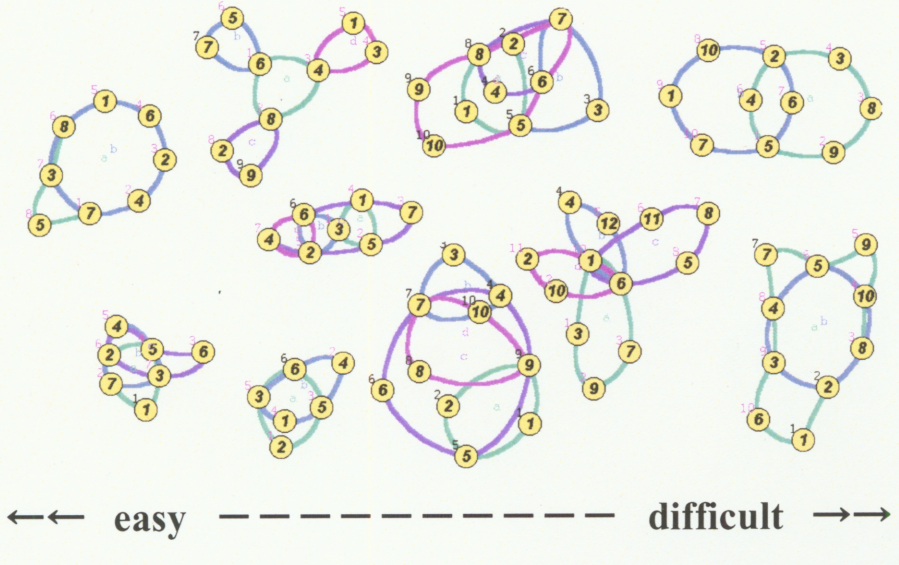
(a) (b)



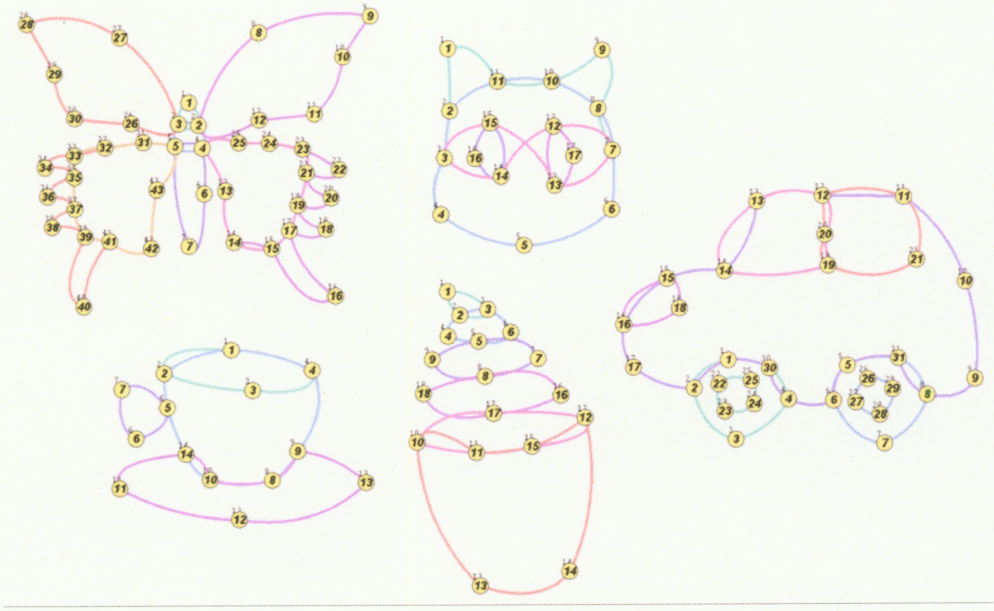
(c)

Fig. 8. (a) Tiling, (b) routing, and (c) drawing of 4^3 Rubik's cube.

New Permutation Puzzles and More Layouts. Permutation puzzles such as the Rubik's cube are sometimes too difficult to solve. Using the puzzle generator, we can create new puzzles with different levels of difficulty. Fig. 9(a) shows examples of the new puzzles with small size. Note that sometimes hand drawn drawings can be more amusing and attractive for the users as shown in Fig. 9(b).



(a)

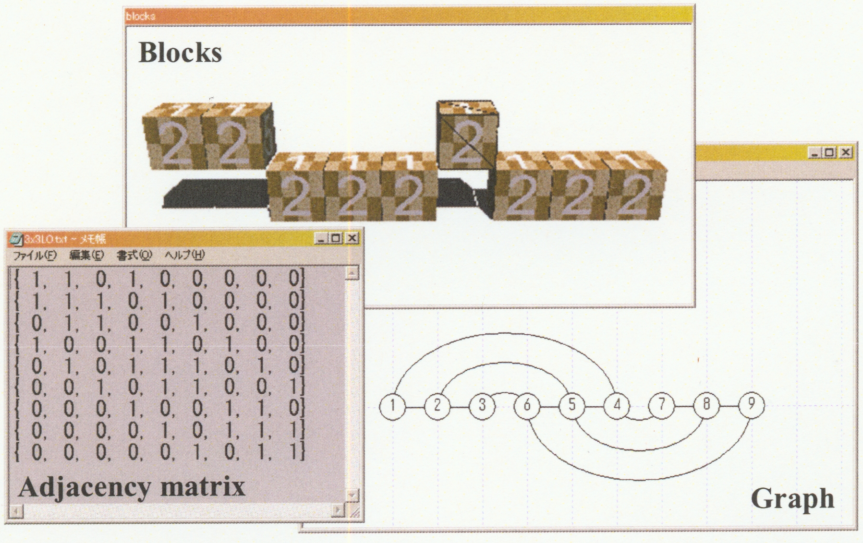


(b)

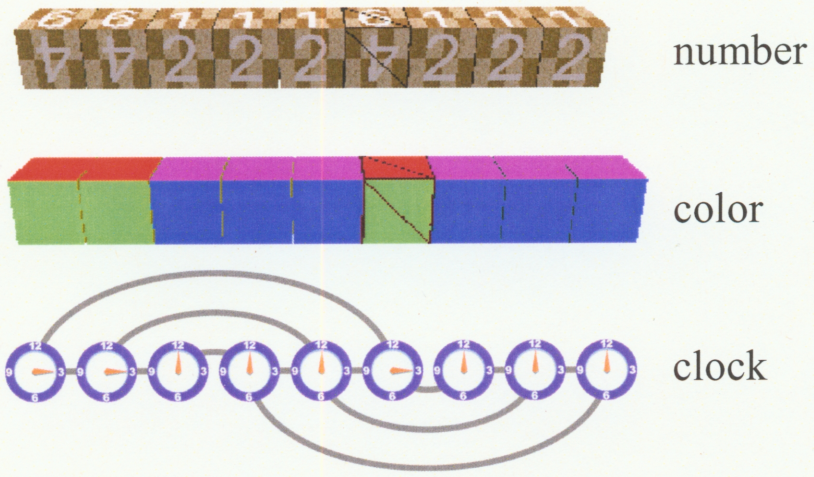
Fig. 9. New permutation puzzles and their layouts.

3.2 Cyclic Puzzles

Cyclic Puzzle Generator. Fig. 10(a) shows a user interface for the cyclic puzzle generator. A puzzle (i.e. toggling rules) is defined in the lower-left window and the rules are illustrated using L-mapping in the lower-right window. An 'up and down box puzzle' is shown in the upper window. Fig. 10(b) shows variations of the layouts of cyclic puzzles which correspond to n-color cyclic puzzles.



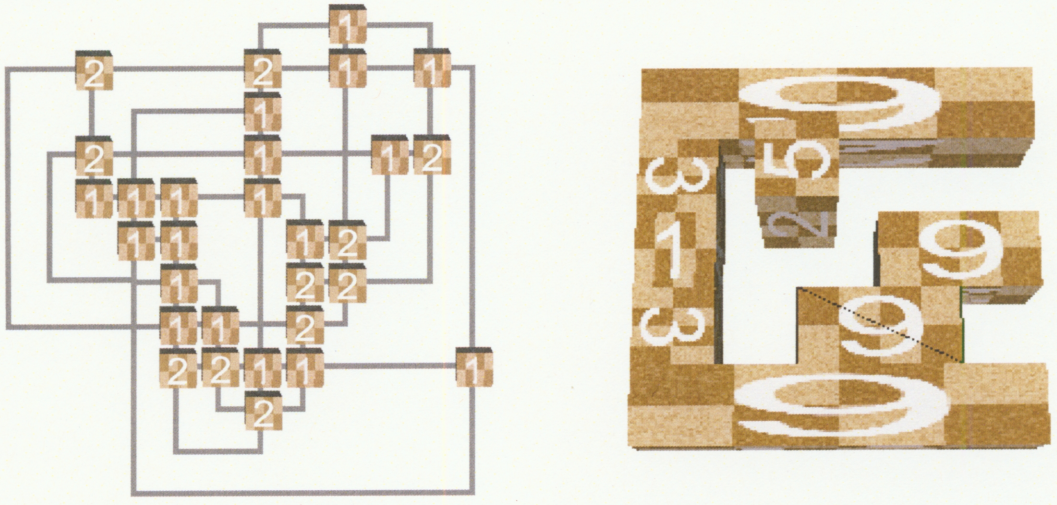
(a)



(b)

Fig. 10. (a) User interface and (b) variations of layouts of cyclic puzzles.

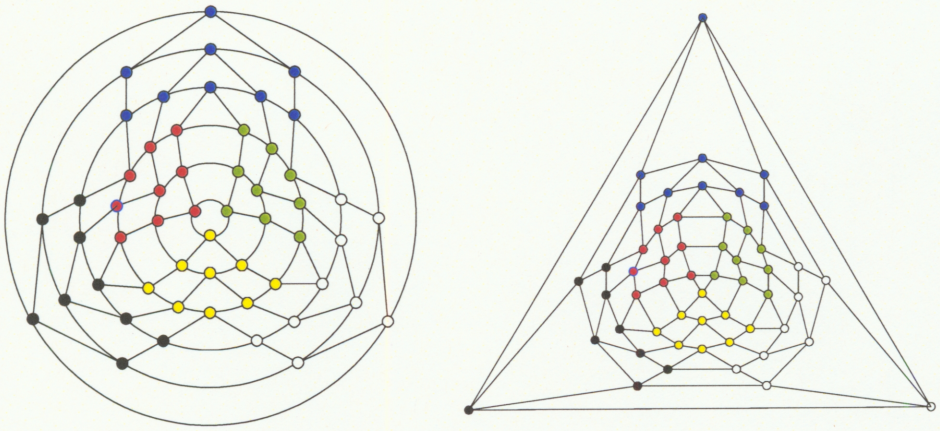
More Layouts of Cyclic Puzzles. We applied various graph drawing methods to achieve a variety of layouts for the puzzle. Fig. 11(a) shows an orthogonal layout and Fig. 11(b) shows a visibility representation.



(a) (b)

Fig. 11. (a) orthogonal layout and (b) visibility representations of cyclic puzzles.

Layouts of Lightsout cube. Fig. 12 shows two symmetric layouts of the Lightsout cube. Fig. 12(a) uses the concept of concentric circles, and Fig. 12(b) uses straight-line. Both are produced manually.



(a) (b)

Fig. 12. Layouts of the Lightsout cube.

4. Symmetric Layout Algorithm for 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 Fig. 7 and Fig. 12 *automatically*. The main problem can be formally defined as follows.

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

The problem of drawing graphs with a maximum number of symmetries can be formulated as a problem of finding a *geometric automorphism group* of a maximum size. It has been proven that the problem is NP-hard. There are many algorithms available to construct a symmetric drawing of a graph for different classes of graphs and different motivations. However, to achieve the maximum number of symmetries, we choose the algorithm by Hong et al. for planar graphs [7,8,9] and the algorithm by Abelson et al. for general graphs [6]. The main algorithm can be described as follows.

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

If *G* is disconnected
then use the algorithm by Hong and Eades [9]
elseif *G* is planar
then if *G* is triconnected, then use the algorithm by Hong, McKay and Eades [7]
if *G* is biconnected, then use the algorithm by Hong and Eades [8]
else use the algorithm by Abelson, Hong and Taylor [6]

Note that the algorithm for the disconnected case [9] was developed for planar graphs. However the same algorithm can be applied to general graphs as long as we want to minimize the number of edge crossings. In fact, the algorithm uses each algorithm in [7,8] as subroutines.

To construct a symmetric drawing, we need two steps. The first step is to find the symmetries of a graph. The second step is to display the symmetries. All the algorithms in [6,7,8,9] provide both a symmetry finding algorithm for step 1 and a symmetric drawing algorithm for step 2. Two algorithms

for planar graphs [7,8] and the disconnected case [9] can be implemented in linear time. The implementation of the symmetry finding algorithm in [8] is rather complicated as it involves many other algorithms such as computing isomorphism of planar graphs and construction of the SPQR tree. However, the output of the symmetry finding algorithm fixes the plane embedding of the graph, which shows the maximum number of symmetries. Given an embedding, the drawing algorithm constructs a drawing displaying given symmetries.

The method for general graphs [6] was implemented using MAGMA and the experimental results show that, in practice, it runs very fast. For example, generally it finds the maximum symmetries of a graph with up to 50 vertices within a second. However, it does not give an embedding as an output. The output of the symmetry finding algorithm is a set of orbits under the geometric automorphism group. An orbit is a subset of the vertex set V of G. If two vertices u and v belong to the same orbit, then there is a geometric automorphism which maps u to v. The drawing algorithm implemented by Abelson et al [6] simply draws each orbit as a concentric circle, as in Fig. 12(a). However, the relative ordering of the orbits is not decided, as the embedding is not given. Hence we need another algorithm to decide the ordering of the concentric circles, each representing an orbit, to minimize the number of edge crossings. Unfortunately, this problem is also NP-hard [10]. However, a heuristic is given in [10], and it is under implementation.

We now discuss a variation of the algorithm, an algorithm for the permutation puzzles. In this case, the puzzle graph can be defined as a set of cycles. In general, the size of the puzzle graph is small (the number of vertices is around 30) and not so dense, but not necessarily planar. If the size becomes large, then we can use the following speed up method.

Firstly, observe that there are many degree two vertices in the set of cycles. In terms of finding symmetries and deciding embeddings, the degree two vertices do not contribute. The most important vertices are intersection points, i.e. the vertices which are overlapped by more than one cycle. This motivates the following algorithm. The main idea is to delete all the degree two vertices and then define a simplified graph G' of G. Then we use the algorithms in [6,7,9] to draw G' and then finally reinsert all the degree two vertices. This approach has an advantage in terms of running time and implementation. That is, we don't need to implement the algorithm in [8].

Algorithm Symmetric_Permutation_Puzzle_Layout
Input: a puzzle graph G consists of a set of cycles.
Output: a drawing D of G displaying the maximum number of symmetries.

1. Delete all degree two vertices resulting in a graph G'.
2. Label the edges and vertices of G' with proper color, so that its automorphism preserve the color. This includes the case of multiple edges or self-loops.
3. If G' is disconnected
then use the algorithm by Hong and Eades [9].
elseif G' is planar
then use the algorithm by Hong, McKay and Eades [7] to draw G'.
else use the algorithm by Abelson, Hong and Taylor [6] to draw G'.
4. Reinsert all deleted degree two vertices to construct a drawing of G.

The refinement of the algorithm needs a little modification of the existing algorithm. For example, we need to modify the symmetry finding algorithm to work for the labeled graphs. This is an easy extension and can be done easily for the algorithm for the triconnected planar graphs [7] and the general graphs [6]. The main advantage of the algorithm is that G' is either disconnected or triconnected. Hence the algorithm requires only the implementation of the triconnected case [7]. It is a slight adaptation of the existing algorithm to preserve colors of edges and vertices.

Another variation is the use of other drawing algorithms, the barycenter algorithm by Tutte [3], or the algorithm by Carr and Kocay [11]. Both algorithms require more input. We can use the output of the symmetry finding algorithm to fix the outer face of the triconnected planar graphs for the Tutte algorithm. The time complexity is superlinear [3], but gives a straight-line convex symmetric drawing

as in Fig. 12(b). The algorithm of Carr and Kocay needs a geometric automorphism as an input, and then finds a largest cycle. It is implemented in Group & Graphs [11].

Another variation is to combine the two methods. That is to use symmetry finding algorithm to fix the embedding and then run the spring algorithm to preserve the embedding to produce nice layout.

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 modeled 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 discuss a symmetric drawing algorithm for puzzles.

Our current work includes the implementation of the symmetric drawing algorithm for the puzzle layout problem. However, our ultimate goal is to develop a software which 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.

References

1. Maeda, A., Sugiyama, K. and Mase, K., Media Conversion of Permutation Puzzles and Development of Permutation Puzzles Generators, *IPSJ SIG Notes Human Interface*, No. 101, 33-40. (in Japanese)
2. Maeda, A., Sugiyama, K. and Mase, K., Media Conversion of Cyclic Puzzles and Development of Cyclic Puzzle Generators, *IPSJ SIG Notes Human Interface*, No. 101, 40-47. (in Japanese)
3. Di Battista, G., Eades, P., Tamassia, R. and Tollis, I., *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, 1999.
4. Sugiyama, K., *Graph Drawing and Applications for Software and Knowledge Engineers*, World Scientific, 2002.
5. Aoki, S., *A Location Problem with Variables on Boolean Field*, B. Eng. Thesis in Information Engineering, Toyohashi University of Technology, 1997.
6. Abelson, D., Hong, S. and Taylor, D. E., A Group-Theoretic Method for Drawing Graphs Symmetrically, *Proc. of Graph Drawing 2002*, LNCS, Springer Verlag, pp. 86-97, 2002.
7. Hong, S., McKay, B. and Eades, P., Symmetric Drawings of Triconnected Planar Graphs, *Proc. of SODA 2002*, pp. 356-365, 2002.
8. Hong, S. and Eades, P., Drawing Planar Graphs Symmetrically II: Biconnected Graphs, Technical Report CS-IVG-2001-01, School of IT, The University of Sydney, 2001.
9. Hong, S. and Eades, P., Drawing Planar Graphs Symmetrically III: Disconnected Graphs, Technical Report CS-IVG-2001-03, School of IT, The University of Sydney, 2001.
10. Buchheim, C. and Hong, S., Crossing Minimization for Symmetries, *Proc. of ISAAC 2002*, Lecture Notes in Computer Science, Springer Verlag, pp. 563-574, 2002.
11. Carr, H. and Kocay, W., An Algorithm for Drawing a Graph Symmetrically, *Bulletin of the Institute of Combinatorics and its Applications*, 27, pp. 19-25, 1999.
12. Tuner, E. C. and Gold, K. F., Rubik's Groups, *American Mathematical Monthly*, Vol. 92, No.9, 1982.
13. Joyner, D., *Adventures in Group Theory: Rubik's Cube, Merlin's Machine, and Other Mathematical Toys*, Johns Hopkins Univ. Press, 2002.