Proceedings of Bridges 2013: Mathematics, Music, Art, Architecture, Culture

From Path-Segment Tiles to Loops and Labyrinths

Robert Bosch, Sarah Fries, Mäneka Puligandla, and Karen Ressler
Dept. of Mathematics, Oberlin College, Oberlin, OH 44074
(bobb@cs.oberlin.edu)

Abstract

We explain how to use integer programs to assemble a collection of path-segment tiles—squares decorated with path segments—on a grid so that the path segments join together to form a single closed loop or a single open path through the centers of the squares of the grid. We describe how to optimize (maximize or minimize) the number of bends (90-degree turns) in the loop or path. We show how to force loops to be symmetric and to encourage paths to be as close to symmetric as possible. We conclude by displaying laser-cut and 3D-printed artwork designed with our integer programs.

1 Introduction

Integer programs are mathematical optimization problems in which the objective is to optimize (in some cases, to maximize, and in others, to minimize) a linear function of integer-valued variables subject to one or more linear constraints (equations or inequalities) on those variables [10]. Since the 1960s, integer programs have been applied with great frequency and success to problems in the areas of logistics, manufacturing, and scheduling, and in these applications, the objective is usually to maximize profit or minimize cost [7]. For the last 12 years, Bosch and colleagues have conducted investigations into how integer programs can be employed in the construction of visual artwork [1-6,9]. In the present article, we continue these explorations.

We focus on using integer programs to assemble a collection of *path-segment tiles* (displayed in Figure 1) on a square grid so that the tiles' path-segment decorations (drawn in black) join together to form a closed loop (a Hamiltonian cycle) or a labyrinth (a Hamiltonian path) through the centers of the squares of the grid. We strive to produce aesthetically pleasing loops and labyrinths. To this end, we employ methods for optimizing (maximizing or minimizing) the number of bends (90-degree turns) in the loop or labyrinth, for forcing loops to be symmetric, and for encouraging labyrinths to be as close to symmetric as possible.

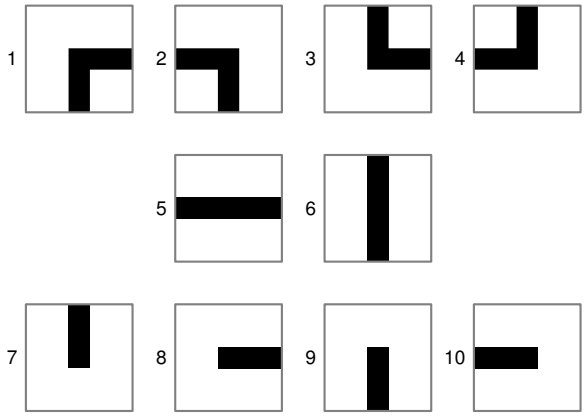

Figure 1: Path-segment tiles.

2 Designing Loops with Path-Segment Tiles

In this section, we describe how to use integer programs to arrange copies of the path-segment tiles into a loop (a Hamiltonian cycle) on an $m \times n$ board (grid of squares). We require that a tile be placed in each square of the board, and we strive to maximize (or minimize) the number of bends (90-degree turns).

2.1 Variables

We set $x_{t,i,j}$ equal to 1 if we place a copy of tile t in square (i, j), the row-i, column-j square of our $m \times n$ board, and we set $x_{t,i,j}$ equal to 0 if we don't do this. Note that there are $6mn$ variables. (For loops, we can restrict ourselves to tiles 1 through 6, so $1 \leq t \leq 6$, $1 \leq i \leq m$, $1 \leq j \leq n$.)

2.2 The Core Constraints

To force ourselves to place precisely one tile in square (i, j) of our board, we impose the constraint

$$\sum_{t=1}^{6} x_{t,i,j} = 1. \tag{1}$$

We need mn such constraints—one for each square.

To force ourselves to place "compatible" tiles in square (i, j) and its right hand neighbor $(i, j+1)$, we impose the constraints

$$x_{1,i,j} + x_{3,i,j} + x_{5,i,j} = x_{2,i,j+1} + x_{4,i,j+1} + x_{5,i,j+1} \tag{2}$$

and

$$x_{2,i,j} + x_{4,i,j} + x_{6,i,j} = x_{1,i,j+1} + x_{3,i,j+1} + x_{6,i,j+1}. \tag{3}$$

Constraint (2) states that the number of tiles placed in square (i, j) that have the path segment exiting the right side of square (i, j) must be equal the number of tiles placed in square $(i, j+1)$ that have the path segment exiting the left side of square $(i, j+1)$. Constraint (3) states that the number of tiles placed in square (i, j) that have *no* path segment exiting the right side of square (i, j) must be equal the number of tiles placed in square $(i, j+1)$ that have *no* path segment exiting the left side of square $(i, j+1)$. We need $m(n-1)$ constraints of each type—one for each square that has a right hand neighbor.

To force ourselves to place compatible tiles in (i, j) and its lower neighbor $(i+1, j)$, we impose the constraints

$$x_{1,i,j} + x_{2,i,j} + x_{6,i,j} = x_{3,i+1,j} + x_{4,i+1,j} + x_{6,i+1,j} \tag{4}$$

and

$$x_{3,i,j} + x_{4,i,j} + x_{5,i,j} = x_{1,i,j+1} + x_{2,i,j+1} + x_{5,i,j+1}. \tag{5}$$

We need $(m-1)n$ constraints of each type—one for each square that has a lower neighbor.

To keep the path from leaving the board, we simply set certain variables equal to 0 and then remove them from the model. By setting $x_{3,1,j} = x_{4,1,j} = x_{6,1,j} = 0$ and $x_{1,m,j} = x_{2,m,j} = x_{6,m,j} = 0$ for each $1 \leq j \leq n$, we prevent the path from crossing either the top edge or the bottom edge of the board. By setting $x_{2,i,1} = x_{4,i,1} = x_{5,i,1} = 0$ and $x_{1,i,n} = x_{3,i,n} = x_{5,i,n} = 0$ for each $1 \leq i \leq m$, we prevent the path from crossing either the left edge or the right edge of the board.

2.3 Objective Function

If we want to find a loop that has as many (or as few) bends as possible, we maximize (or minimize)

$$\sum_{t=1}^{4} \sum_{i=1}^{m} \sum_{j=1}^{n} x_{t,i,j}. \tag{6}$$

2.4 Sub-loop Elimination Constraints

If we maximize (6) subject to the core constraints, we will end up with a solution similar to the one displayed in Figure 2.

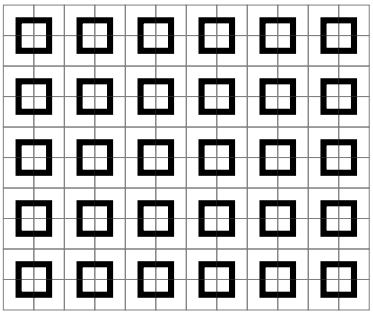

Figure 2: 30 sub-loops on a 10×12 board.

This solution satisfies all of the core constraints, but it it is composed of sub-loops. To eliminate sub-loops, we adapt Dantzig, Fulkerson, and Johnson's approach to solving instances of the Traveling Salesman Problem (TSP) [8]. In the TSP, when we encounter subtours, we add linear inequalities to the model to eliminate them. Here, to eliminate the top left sub-loop, we add the linear inequality

$$x_{1,1,1} + x_{2,1,2} + x_{3,2,1} + x_{4,2,2} \leq 2 \tag{7}$$

to our model. This constraint states that we may use no more than two of the four tile placements that make up the top left sub-loop. Note that if we were to use three of the four tile placements, the compatibility constraints (constraints (2) through (5)) would force us to use all four of them.

To eliminate other sub-loops, we include constraints similar to constraint (7). As in the case of the TSP, we add our sub-loop elimination constraints as they are needed.

2.5 Symmetry Constraints

If we want our loop to be symmetric, we can include constraints that force in the desired symmetry. For example, if we want our 10×12 loop to have 180-degree rotational symmetry, we can include

$$\begin{aligned} &x_{1,i,j} = x_{4,11-i,13-j}, \qquad x_{2,i,j} = x_{3,11-i,13-j}, \qquad x_{3,i,j} = x_{2,11-i,13-j}, \\ &x_{4,i,j} = x_{1,11-i,13-j}, \qquad x_{5,i,j} = x_{5,11-i,13-j}, \qquad x_{6,i,j} = x_{6,11-i,13-j} \end{aligned}$$

for all $1 \leq i \leq 5$ and all $1 \leq j \leq 12$.

2.6 A Gallery of Loops

To date, we have used our integer programming model to create thousands of loop designs. Figure 3 displays three designs that have 180-degree rotational symmetry. Loop 3*a* has 108 bends, the greatest number possible for a 10×12 loop with 180-degree rotational symmetry. There are seven other 10×12 loops that have 108 bends and 180-degree rotational symmetry. Once we have an optimal solution, we can find additional optimal solutions (and eventually suboptimal solutions) by including sub-loop elimination constraints to eliminate the solutions we have already found.

Loop 3*c* has only 20 bends, the least number possible for a 10×12 loop with 180-degree rotational symmetry. There are nine other 10×12 loops with 180-degree rotational symmetry that have only 20 bends.

Loop 3*b* has 68 bends. To produce this design, we employed special constraints to force in the interior staircase structure.

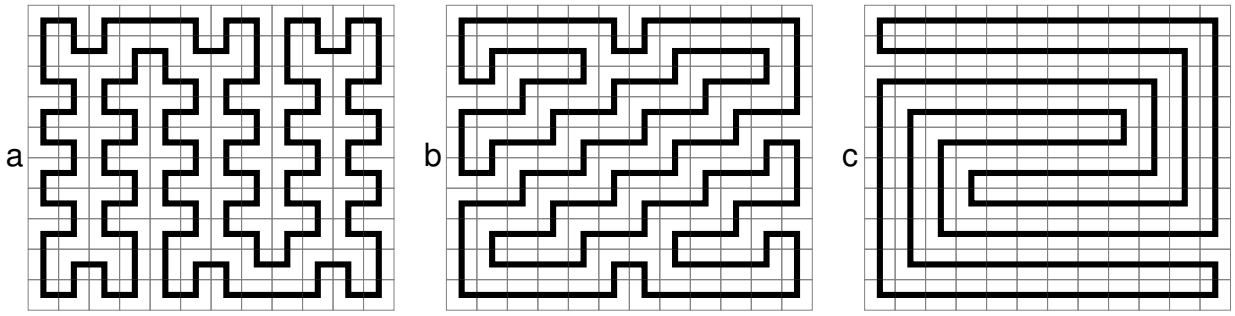

Figure 3: Three 10×12 loops that have 180-degree rotational symmetry.

Figure 4 displays six designs that have horizontal mirror symmetry, vertical mirror symmetry, and 180-degree rotational symmetry. By repeatedly adding sub-loop elimination constraints to eliminate all previously obtained solutions, we discovered that there are 2179 such loops. Loop 4*a* is one of 24 loops that have 100 bends, the greatest number possible for a 10×12 loop with these symmetries. Loop $4f$ is one of eight loops that have only 36 bends, the least number possible for a 10×12 loop with these symmetries. Loops 4*b*, 4*c*, 4*d*, and 4*e* have 84, 76, 60, and 52 bends, respectively. They are some of our favorite designs.

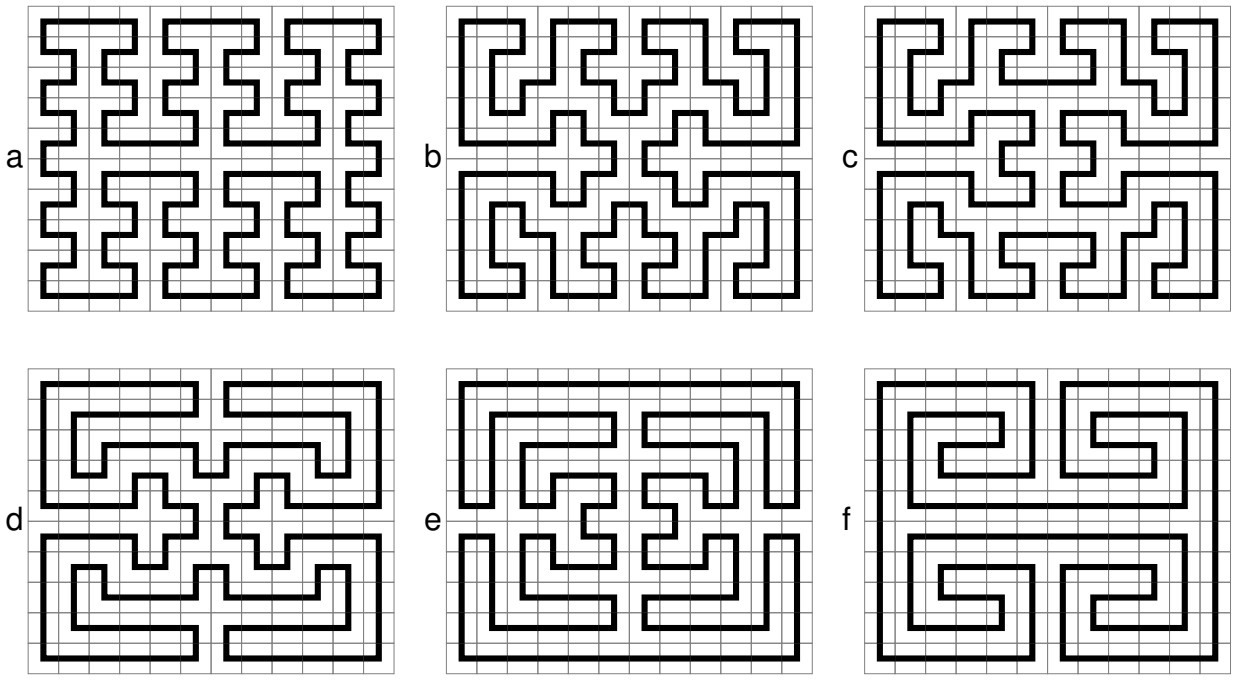

Figure 4: Six 10×12 loops that have horizontal mirror symmetry, vertical mirror symmetry, and 180-degree rotational symmetry.

2.7 Loops in Physical Form

To get some sense of what it would be like to travel through these loops, Robert Bosch made laser-cut versions out of hardboard and MDF. The laser-cut loops can be placed atop a BRIO Labyrinth game, as shown in the left half of Figure 5. The player can then turn the game's knobs to tilt the board and maneuver a steel ball through the loop.

Figure 5: A laser-cut version of Loop 4*e* (left) and a loop made out of 3D-printed tiles (right).

To be able to construct *all* loops (and labyrinths) that can be constructed from path-segment tiles, Robert and Derek Bosch designed a set of 3D-printed tiles shown in right half of Figure 5. On one side of each tile, there is a path segment (not drawn in black, but inset into the tile) that has a 90-degree bend. On the opposite side, there is a path segment (again, inset into the tile) that goes straight across.

3 Designing Labyrinths with Path-Segment Tiles

In this section, we provide a sketch of how to modify the integer program described in the previous section so that it can be used to arrange copies of the path-segment tiles into a labyrinth (a Hamiltonian path) that starts on a specified square on the edge of the board and ends on a specified square in the interior.

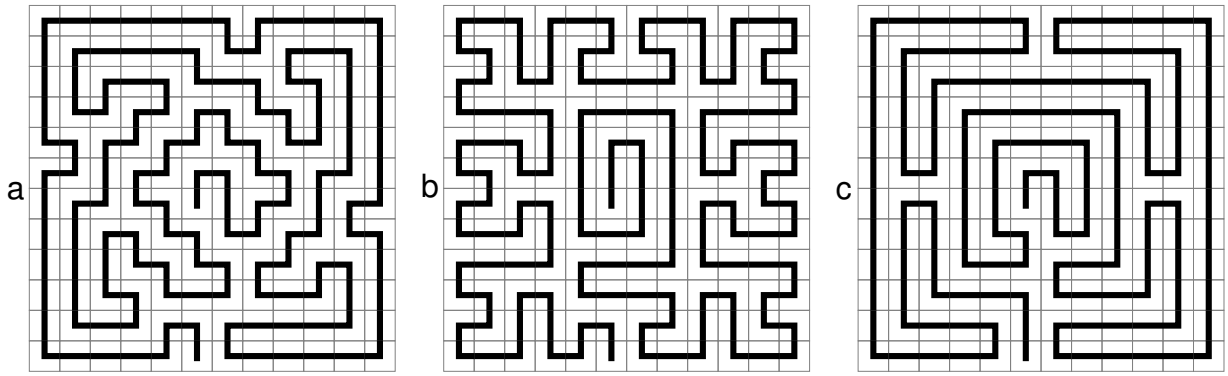

Figure 6: Three 12×12 labyrinths.

For the three 12×12 labyrinth designs displayed in Figure 6, square $(12,6)$ is the start square and square $(7,6)$ is the end square. If we want the path to leave the start square and enter the end square through these squares' upper neighbors, we set $x_{7,12,6} = x_{7,7,6} = 1$ and $x_{t,i,j} = 0$ for all $8 \leq t \leq 10$. Other than making some small modifications to the instances of constraints (2) through (5) that involve squares $(12,6)$ and $(7,6)$, we are able to leave the core constraints alone. As in the loop case, we add sub-loop elimination constraints as needed.

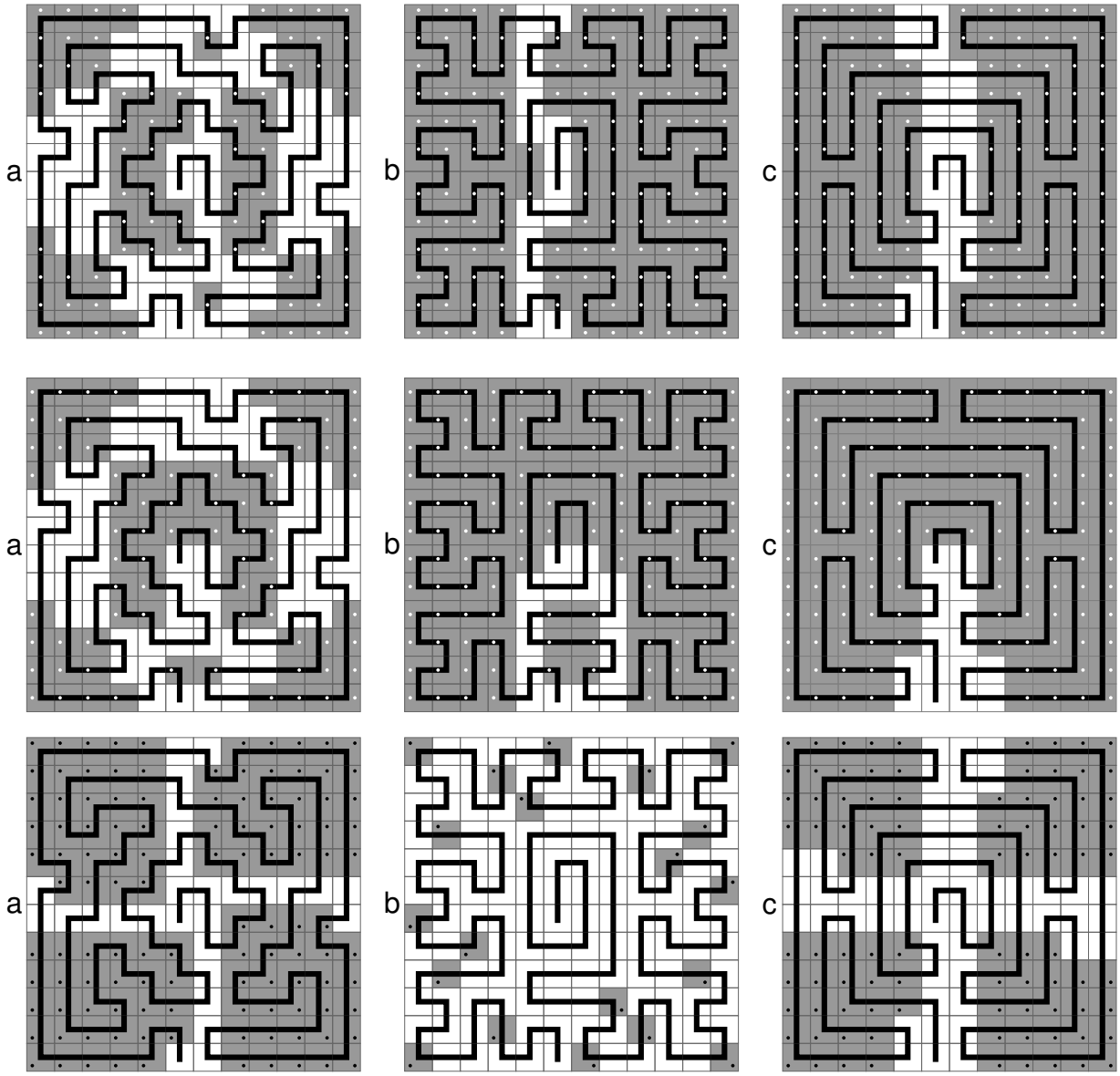

Figure 7: Three 12×12 labyrinths with (top) all instances of horizontal mirror symmetry shaded and marked with white dots, (middle) all instances of vertical mirror symmetry shaded and marked with white dots, and (bottom) all instances of 90-degree rotational symmetry shaded and marked with black dots.

As it is impossible for a labyrinth with our chosen start and end squares to be perfectly symmetric, we do not include symmetry constraints. Instead of maximizing or minimizing the number of tiles with 90-degree bends, we make it our goal to design labyrinths that will be as close as possible to being symmetric.

To measure closeness to horizontal mirror symmetry, we introduce a binary variable $h_{t,i,j}$ for each tile t with $1 \leq t \leq 6$ and each square (i,j) with $1 \leq i \leq m/2$ and $1 \leq j \leq n$ (each square in the top half of the board). For each path-segment tile t, we let $H(t)$ denote the image of tile t in a horizontal mirror. Note that $H(1) = 3$, $H(2) = 4$, $H(3) = 1$, $H(4) = 2$, $H(5) = 5$, and $H(6) = 6$. We then constrain the variable $h_{t,i,j}$ as follows:

$$\begin{aligned} x_{t,i,j} + x_{H(t),m+1-i,j} &\leq 1 + h_{t,i,j}, & (8)\\ h_{t,i,j} &\leq x_{t,i,j}, & (9)\\ h_{t,i,j} &\leq x_{H(t),m+1-i,j}. & (10) \end{aligned}$$

Inequality (8) forces $h_{t,i,j}$ to equal 1 when both $x_{t,i,j}$ and $x_{H(t),m+1-i,j}$ equal 1. Inequalities (9) and (10) force $x_{t,i,j}$ and $x_{H(t),m+1-i,j}$ to equal 1 when $h_{t,i,j}$ equals 1. In other words, inequalities (8), (9), and (10) state that the variable $h_{t,i,j}$ equals 1 if and only if tile t and its image $H(t)$ are involved in an *instance of horizontal mirror symmetry* that takes place in squares (i,j) and $(m+1-i,j)$. In the top row of Figure 7, the squares that have been shaded and marked with white dots are precisely those squares that house instances of horizontal mirror symmetry. Labyrinth *b* has 128 such instances, while Labyrinth *c* has 120 instances, and Labyrinth *a* has only 74 instances.

To count instances of vertical mirror symmetry, we introduce a binary variable $v_{t,i,j}$ for each tile t with $1 \leq t \leq 6$ and each square (i,j) with $1 \leq i \leq m$ and $1 \leq j \leq n/2$ (each square in the left half of th board). For each t, we let $V(t)$ denote the image of tile t in a vertical mirror. Note that $V(1) = 2$, $V(2) = 1$, $V(3) = 4$, $V(4) = 3$, $V(5) = 5$, and $V(6)$. We then constrain the variable $v_{t,i,j}$ with inequalities similar to (8), (9), and (10). The middle row of Figure 7 shows the locations of instances of vertical mirror symmetry. Labyrinths *b* and *c* are the best in terms of vertical mirror symmetry, with 128 instances, while Labyrinth *a* is far behind with only 76 instances.

To count instances of 90-degree rotational symmetry, we introduce a binary variable $r_{t,i,j}$ for each tile t with $1 \leq t \leq 6$ and each square (i,j) with $1 \leq i \leq n/2$ and $1 \leq j \leq n/2$ (each square in the top left quadrant of the board). Here we are assuming that $m = n$. For each t, we let $R(t)$ denote the tile produced when tile t is rotated 90 degrees clockwise about its center. Note that $R(1) = 2$, $R(2) = 4$, $R(3) = 1$, $R(4) = 3$, $R(5) = 6$, and $R(6) = 5$. We then constrain the variable $r_{t,i,j}$ as follows:

$$\begin{aligned} x_{t,i,j} + x_{R(t),j,n+1-i} + x_{R(R(t)),n+1-i,n+1-j} + x_{R(R(R(t))),n+1-j,i} &\leq 3 + r_{t,i,j},\\ r_{t,i,j} &\leq x_{t,i,j},\\ r_{t,i,j} &\leq x_{R(t),j,n+1-i},\\ r_{t,i,j} &\leq x_{R(R(t)),n+1-i,n+1-j},\\ r_{t,i,j} &\leq x_{R(R(R(t))),n+1-j,i}. \end{aligned}$$

These inequalities state that the variable $r_{t,i,j}$ equals 1 if and only if tile t and its rotations $R(t)$, $R(R(t))$, and $R(R(R(t)))$ are involved in an instance of 90-degree rotational symmetry that takes place in squares (i,j), $(j,n+1-i)$, $(n+1-i,n+1-j)$, and $(n+1-j,i)$ (square (i,j) and the three squares it rotates into). The bottom row of Figure 7 shows the instances of 90-degree rotational symmetry. Here, Labyrinth *a* is the best, with 116 instances. Labyrinth *c* has 92 instances, and Labyrinth *b* has only 20 instances.

We produced Labyrinth *a* by maximizing the sum of the $r_{t,i,j}$ variables. By repeatedly adding sub-loop elimination constraints to eliminate all previously obtained solutions, we discovered that Labyrinth *a* is one of 112 12×12 labyrinths that have 116 instances of 90-degree rotational symmetry, the highest number possible.

To produce Labyrinth *b*, we maximized the sum of $h_{t,i,j}$ variables *and* the $v_{t,i,j}$ variables. Labyrinth *b* is one of more than four hundred 12×12 labyrinths than have a total horizontal/vertical (h/v) score of 256, the highest number possible. We did not find all such labyrinths. We suspect that the number is very large.

To produce Labyrinth *c*, we maximized the sum of the $h_{t,i,j}$ variables, the $v_{t,i,j}$ variables, and the $r_{t,i,j}$ variables. Labyrinth *c* has a total horizontal/vertical/rotational (*h*/*v*/*r*) score of 340, the highest number possible. Only one other 12×12 labyrinth has a total *h*/*v*/*r* score of 340, and like Labyrinth *a* it closely resembles a classical Chartres-like labyrinth. Labyrinth *b* has a total *h*/*v*/*r* score of 276, and labyrinth *a* has a total *h*/*v*/*r* score of 266.

4 Conclusion

We have shown that integer programs can be used to arrange a supply of path-segment tiles into loops (Hamiltonian cycles) and labyrinths (Hamiltonian paths). When designing loops with our model, the user can include constraints that force in desired symmetries and can use the objective function to find the loops that have the most bends (or the least bends). When designing labyrinths, the user cannot force in symmetries, but can strive for closeness to symmetry via an objective function that counts instances of symmetries.

In other words, we have shown that integer programs can be used for searching through the set of all loops and labyrinths. We can reduce the size of the sets by including constraints that eliminate unwanted elements (non-symmetric loops, for example). And through our choice of an objective function, we can focus our search in a desired direction (for example, towards the "bendy" portion of the set of loops or towards the "almost rotationally symmetric" portion of the set of labyrinths).

We have not shown, nor have we attempted to show, that integer programs are the best tool (or *a* best tool) for this task.

5 Acknowledgments

We thank the anonymous reviewers for their constructive feedback.

References

[1] R.A. Bosch, "Constructing domino portraits," in *Tribute to a Mathemagician,* ed. B. Cipra et al., A.K. Peters, 2004, 251-256.

[2] R. Bosch. Opt art. *Math Horizons*, February 2006, 6-9.

[3] R. Bosch. Edge-constrained tile mosaics. In *Bridges Donostia: mathematical connections in art, music, and science*, pages 351-360, 2007.

[4] R. Bosch. Connecting the dots: the ins and outs of TSP Art. In *Bridges Leeuwarden: mathematical connections in art, music, and science*, pages 235-242, 2008.

[5] R. Bosch. Simple-closed-curve sculptures of knots and links. *Journal of Mathematics and the Arts.* 4(2):57-71, 2010.

[6] R. Bosch and A. Pike. Map-colored mosaics. In *Bridges Banff: mathematical connections in art, music, and science*, pages 139-146, 2009.

[7] D.-S. Chen, R.G. Batson, and Y. Dang. *Applied Integer Programming: Modeling and Solution,* Wiley, 2010.

[8] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2:393-410, 1954.

[9] C.S. Kaplan and R. Bosch. Operations research in the visual arts. In *Wiley Encyclopedia of Operations Research and Management Science*, Wiley, 2010.

[10] L. Wolsey, *Integer Programming*, Wiley-Interscience, 1998.