Railway Mazes: From Picture to Solution

Antonín Slavík, Charles University, Prague, Czech Republic; slavik@karlin.mff.cuni.cz
Stan Wagon, Macalester College, St. Paul, Minnesota, USA; wagon@macalester.edu

1. Introduction

Sir Roger Penrose, pursuing an idea first investigated by his father, Lionel Penrose, presented several examples of railway mazes in [1]. One of them, along with some other puzzles, is carved in stone on the Luppitt Millennium Bench in England [2]. A railway maze, such as the example in Figure 1, uses curves to represent tracks and asks for a route from *S* to *F*. But at each junction the train must follow the direction of the track: there can be no reversal of direction (though each track supports movement in either direction and sometimes a certain loop can lead to a reversal of direction). These mazes can be difficult to solve by hand so we wanted to see how computation could be used.

Conceptually an algorithmic solution is simple: use a breadth-first search on the maze, allowing only legal moves at each step. But we wanted to develop a method that would start with just a picture of maze, in jpeg form, say, and find the solution from that. If we succeeded, we could then investigate further to see if there were any solutions ("cooks") other than the one that the Penroses intended. We use *Mathematica* since it has good image processing capabilities in addition to its programming strength. The heart of the problem is determining, when at a junction point, the possible legal continuations.

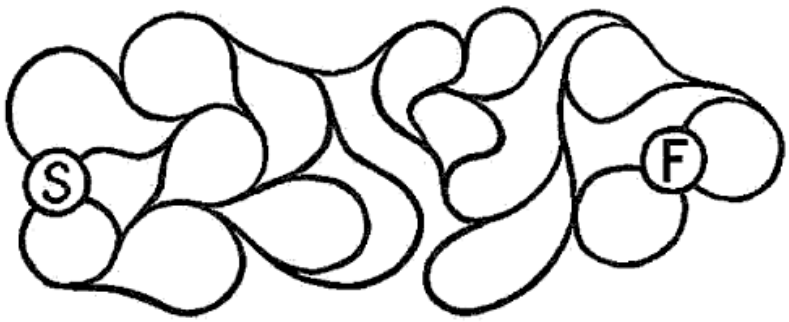

Figure 1. A simple railway maze that is relatively easy to solve by hand.

2. An Algorithm

Consider the maze in Figure 1, where one starts at any spot on the circle around the start, *S*, and aims for the finish at *F*. To repeat, the rules require that at each junction one must follow the curves as a train would. This example is not hard to solve by hand, since there is a bridge at center top that is the only connection from the left side to the right. The first steps of a general algorithm are:

1. binarize the image, making each pixel pure white or pure black;

2. flip black and white for convenience because white corresponds to 1.

These two steps can be carried out by the *Mathematica* commands `Binarize` and `ColorNegate`. This leads to the image in Figure 2.

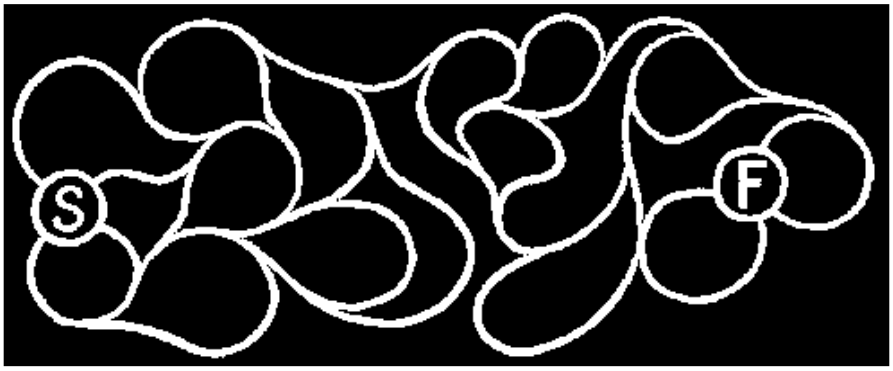

Figure 2. Some initial massaging simplifies the maze.

Next we want to thin this to a small, discrete set of white points. A direct approach using *Mathematica*'s `Thinning` command leaves small loops at certain junctions (evident at right in Fig. 3). To eliminate these we just blur the image, which turns some pixels near the boundary gray, and then apply an additional binarization and then some thinning. The result is quite good as it reduces the maze to a sequence of points (Fig. 4, left). The small triangle at upper right in Figure 4 is not a problem, as will be explained shortly. We will not show *Mathematica* code here, but a notebook with all the details is available at [3].

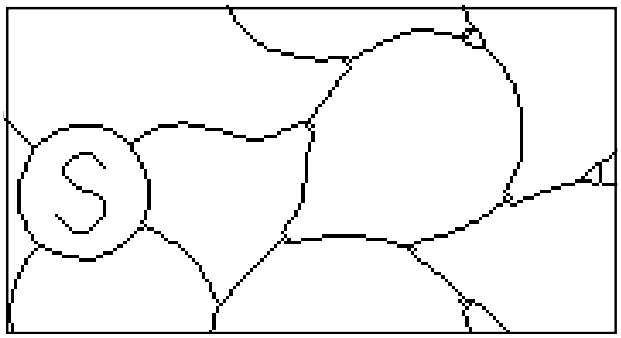

Figure 3. Simple thinning leaves troublesome loops.

After the blurring and thinning steps, we must determine the coordinates of the pixels in the image and then, for each pixel, the set of neighboring pixels, so as to get an undirected graph. Figure 4, left, shows the result after the thinning, while the image at right is a closeup with segments indicating the resulting graph structure. The first step is complicated by the fact that a nontraditional coordinate system is used in images, but the solution is simply to apply the transformation $(x, y) \mapsto (y, Y - x)$, where Y is the number of pixels in the vertical dimension.

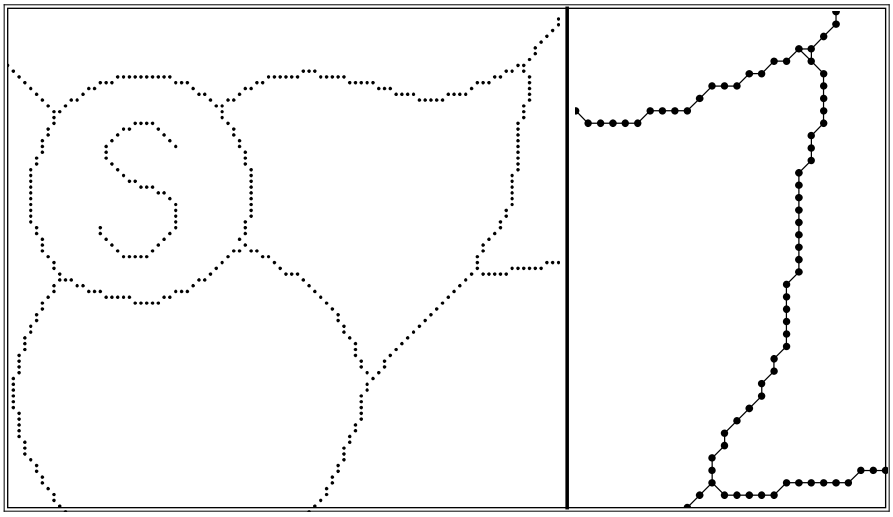

Figure 4. Discretization and thinning leads to a set of points that will allow an algorithmic approach to the maze solution.

Having an actual graph is not all that helpful, since the crux of the problem is that the legal moves from a point depend on how the point was reached. For example, if one is at a pixel at the center of a wye (a Y-junction) having reached that point via the stem, then there are two legal moves, but if one has arrived along one of the wye's upper branches, one can move only onto the stem. Similarly at points that are not junctions: if one has moved from *A* to *B*, one cannot return to *A* but must leave *B* without changing direction.

One approach is to add direction to each edge in the graph, along with additional edges that cause the rules to be enforced; for example, away from the junctions one can duplicate edges, with one track directed one way and the other track the opposite. But this gets excessively complicated in cases where the junctions are not simple wyes. So instead we take a more straightforward approach that uses a queue to set up a breadth-first search in the standard way, but with a separate algorithm that, at each step, uses the history to determine the legal continuation moves.

At a degree-two vertex, there is only one legal move (continue forward). Finding possible moves at a junction (a vertex with degree three or more) is harder. The basic idea is to measure the turning angle between the incoming direction and all possible outgoing directions. Since the train cannot make sharp turns, the legal moves are only those where the corresponding angle does not exceed a certain threshold. Our curves are made up of discrete points, so we have no access to the true tangent to the maze's curves. Instead we use secants to approximate the tangents. To get reasonable secants we must skip nearby points; experimentation led us to jump 7 points forward and 8 points backward to get segments that allow good angle measurements (Fig. 5). After some trial and error, we found that accepting outgoing directions with turning angle no greater than 70° is a reasonable choice.

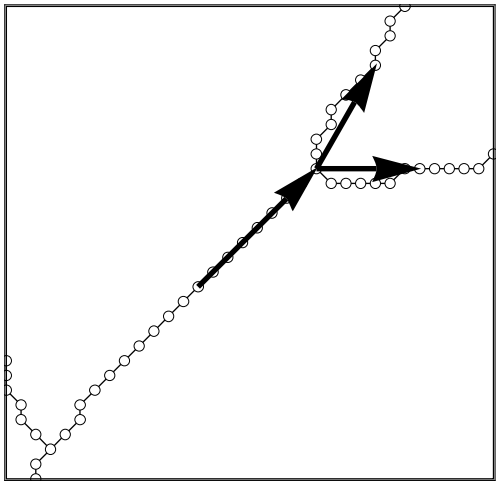

Figure 5. A junction with two outgoing directions and the corresponding secant vectors that are used for angle measurements.

Another technical point arises from the fact that the neighbors of a point are chosen from the eight possible neighboring pixels. Thus one can see situations such as that in Figure 6. As we go from pixel 1 to 2 we arrive at a point having degree four. Neglecting the incoming direction, there are three possible continuations: to 3, 4, or 6. But it would be illegal to go to 3, then 4, and then back to 2, as that would reverse direction. Therefore upon reaching a junction such as 2 we skip the neighboring pixels having degree three or more (3, 4, and 6 in the figure) and consider only moves to pixels having degree two (5 and 7). Only the latter are allowed as possible continuations. This approach also means that configurations such as the small triangle that shows up at the upper right of Figure 4 cause no problems.

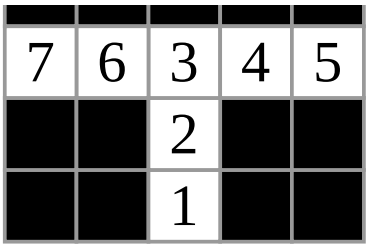

Figure 6. A wye junction with pixel 2 connected to three pixels (3, 4, and 6) having degree three. These must be ignored when determining legal continuations.

With the coordinate data of the points and a robust method for determining legal moves in hand, it is straightforward to set up a queue-based program to carry out a breadth-first search to explore the maze. This yields the shortest path from start to finish in terms of the number of points visited. This quickly solves the maze; the solution is shown in Figure 7.

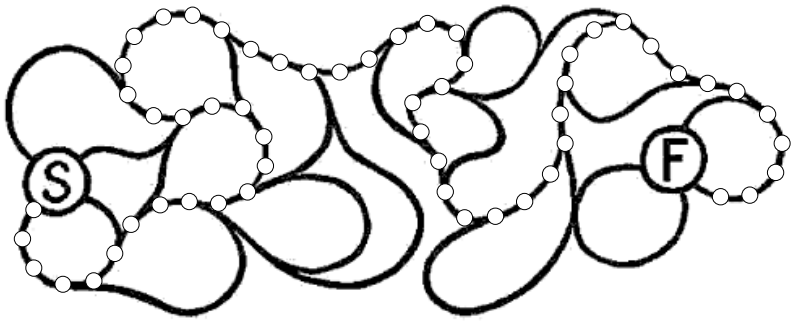

Figure 7. The original maze and a solution found by a breadth-first search.

3. Harder Mazes

The approach of §2 works well on the example of Figure 1, but the maze is quite simple. The method works with essentially no change on some complicated mazes, but in one case some further manual intervention was needed. Figures 8–10 show three mazes from [1] that are all challenging to solve by hand. Our basic method works fine on the first two (solutions in Figs. 11 and 12). But the third maze presents some new difficulties.

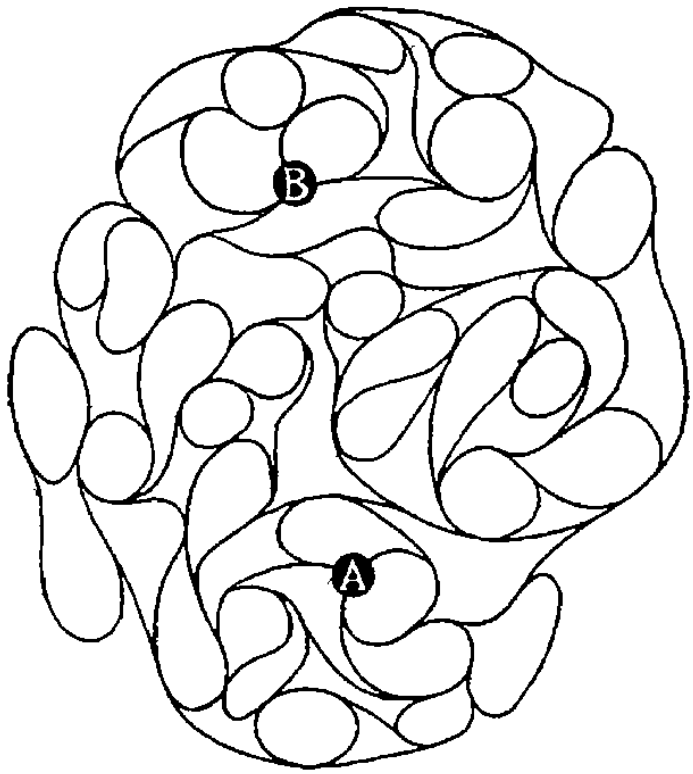

Figure 8. An elegant railway maze designed by Lionel and Roger Penrose in 1958.

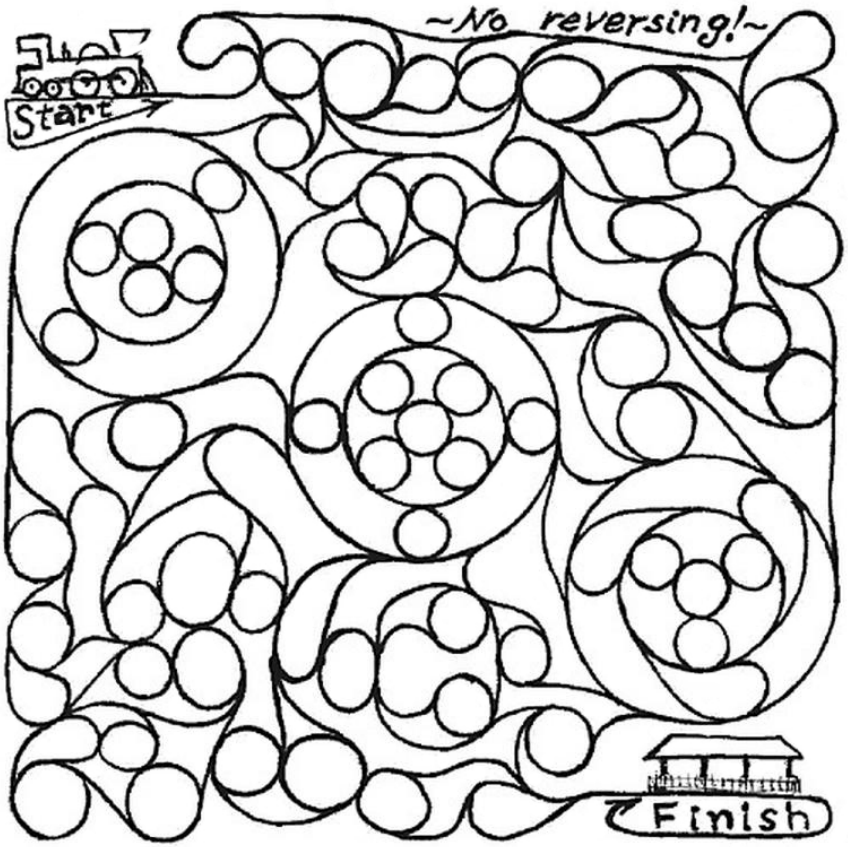

Figure 9. Sir Roger's maze that is etched in stone in Luppitt [2].

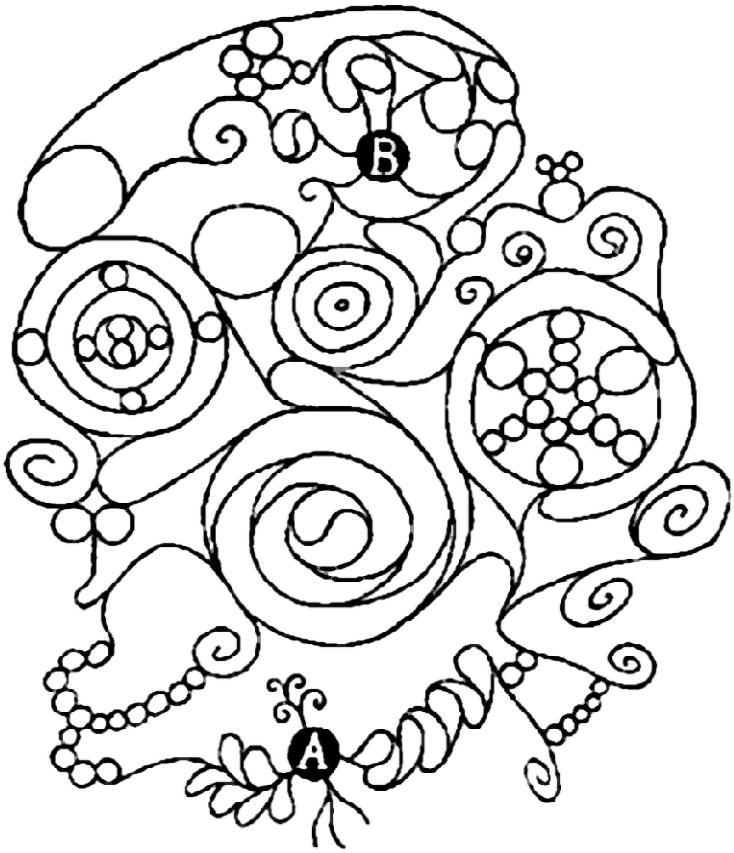

Figure 10. Another maze by the Penroses from 1958.

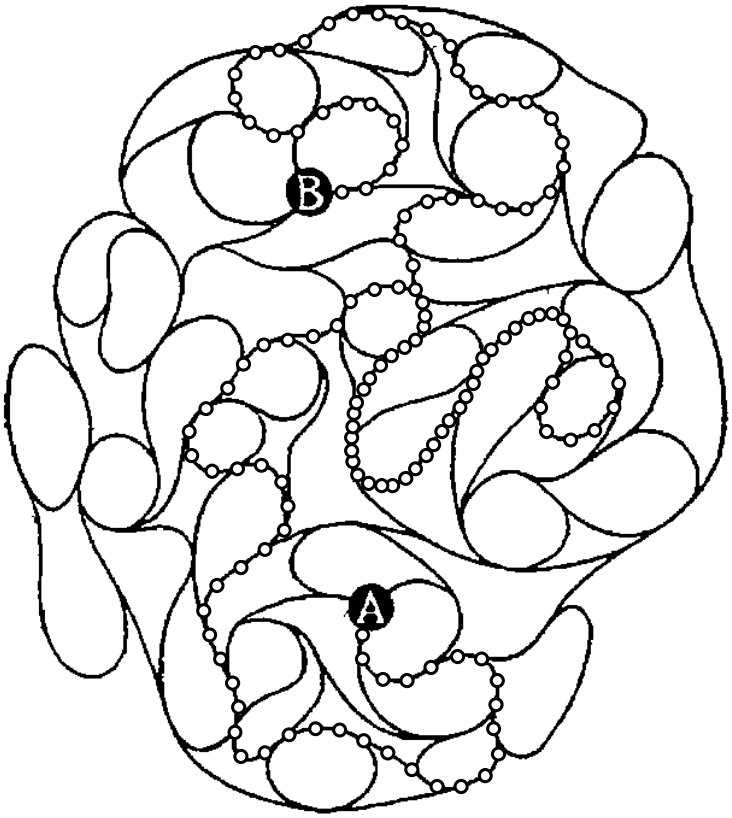

Figure 11. Solution to the maze of Figure 8, found by our basic algorithm.

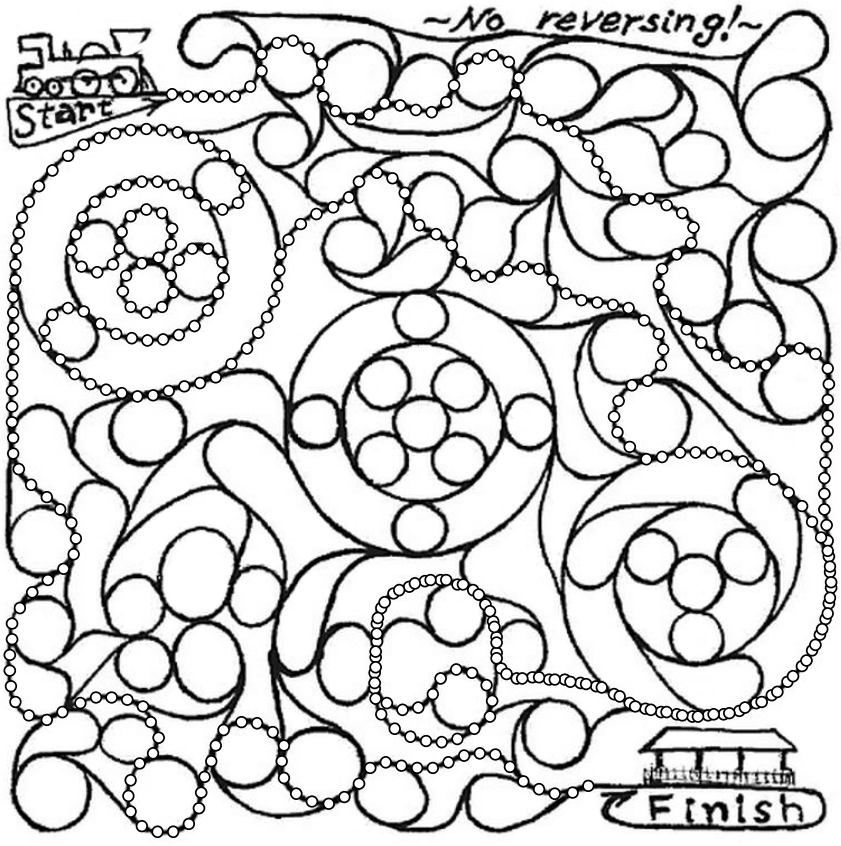

Figure 12. Solution to the maze of Figure 9, found by our basic algorithm.

To solve the maze in Figure 10, we first try our threshold angle of 70° but find that that leads to an incorrect solution; see Fig. 13, which shows two illegal turns; there are more elsewhere in the maze.

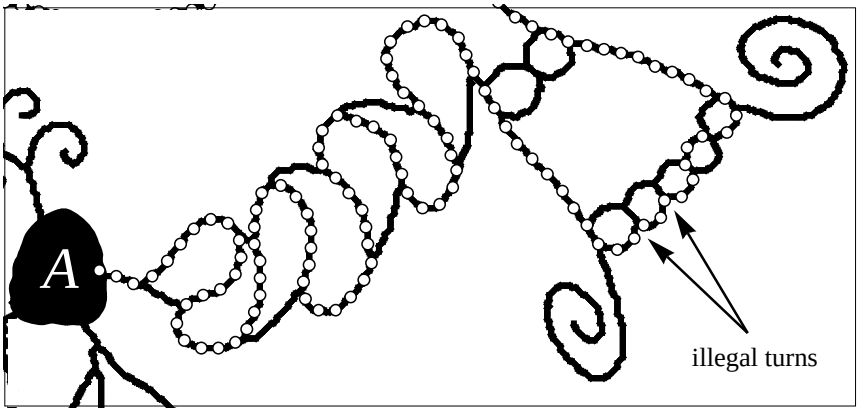

Figure 13. Using the 70° threshold for continuing a route leads to two illegal turns at lower right.

The problem is the fuzziness of the image in places. While one can use a photo editing program to clean up and sharpen the image near the illegal turns, we wanted to pursue our main idea of using programming as much as possible. Thus we took some manual intervention and stated that certain moves are forbidden when determining possible continuations. After a few iterations, the solution shown in Figure 14 was found.

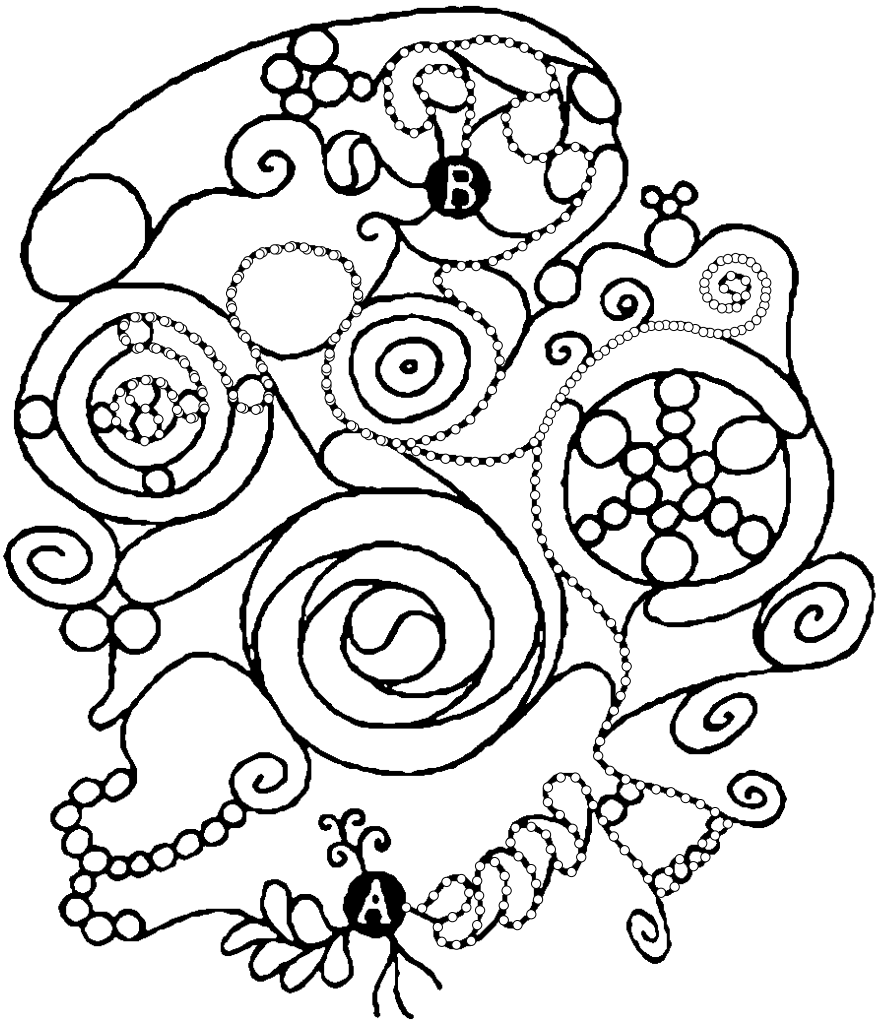

Figure 14. A correct solution to a whimsical railway maze. There are two places where the path traverses a section in both directions.

Having an automated process allows us to check for cooks. That is, starting with the expected unique solution, we can delete from the maze an edge from that solution and see if any new solutions arise. We did this for the three mazes presented here. For the first two we found only solutions that differed in inessential ways from the expected ones: one always has a choice of direction when traversing a direction-reversing loop. However, for the maze of Figure 14 there are many ways, and they are somewhat different, of traversing the leftmost of the two direction-reversing loops (see Fig. 15 for one such). Thus unlike the other two mazes, this one is not quite so pure. The solution in Figure 14 is the shortest.

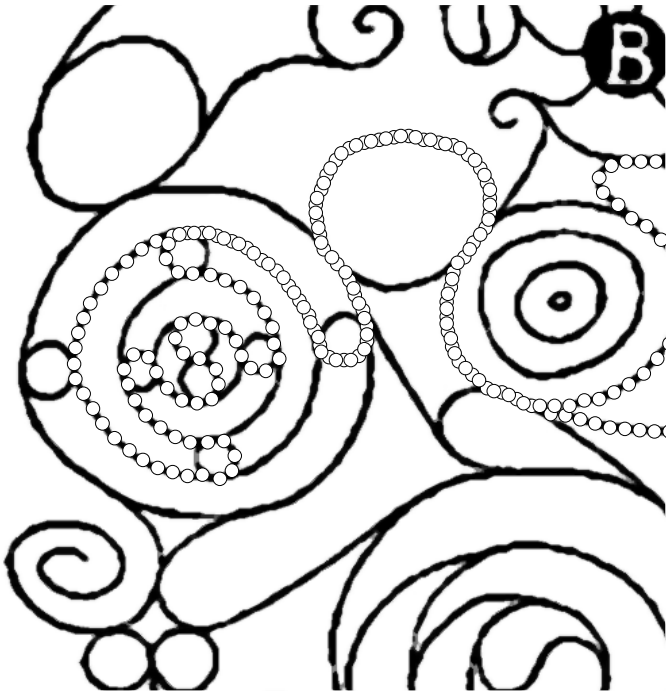

Figure 15. There are several different ways of reversing direction using the configuration of circles at upper left. Here is one of them.

A possible project for the interested reader is to develop an algorithm that would take a scanned image and automatically perform the necessary sharpening to eliminate the sort of issue that causes problems in the maze of Figure 10. Another project is to write a program that will automatically generate interesting railway mazes with unique solutions. And there is always the more general problem of going from a picture to a solution for other types of planar mazes.

Acknowledgment. The authors are grateful to Joan Hutchinson for valuable input related to a solving method using a transformation to a directed graph.

References

1. R. Penrose, Railway mazes, in *A Lifetime of Puzzles*, E. D. Demaine, M. L. Demaine, T. Rodgers (eds.), AK Peters, Wellesley, Mass., 2008, 133–148.

2. The Millenium Monument in Luppitt, Devon, England; http://puzzlemuseum.com/luppitt/lmb02.htm

3. Electronic supplement (*Mathematica* file) to this article; http://stanwagon.com/public/RailwayMaze.nb