Optimizing Our Way Out of Minor Mazes

Evan Pritchard
MATH 4400

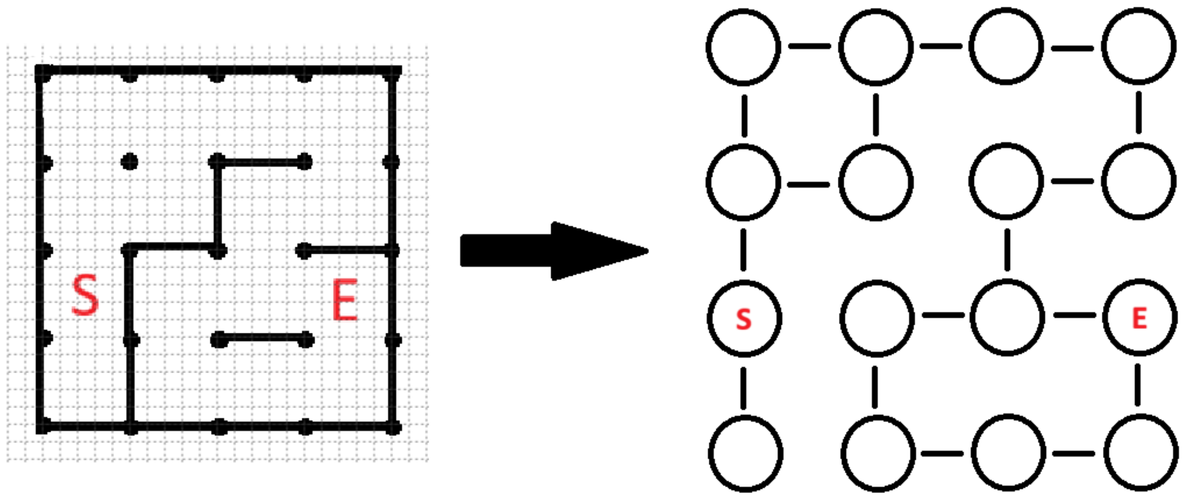
December 22, 2025

1. Maze Solving Methods

Mazes have been around since the beginning of recorded history. Labyrinthian lairs of winding walls and path after path of turnbacks and dead-ends, their structures and their solutions have mystified humans since their inception (Knill). A pleasureful puzzle for most, the ability to solve a maze requires complex problem solving skills, one not easily automated. And with maze sizes ranging from the minuscule to the massive, not all of them can be quickly and efficiently analyzed by hand. There are some commonly implemented algorithms for solving mazes such as the Breadth First Search and the Floodfill Algorithm, but there is ample room for improvement. Neither of these maze solving algorithms is ideal in all cases (Sadik, 56). With the boom in modern Artificial Intelligence, the need to effectively and efficiently automate many tasks is great. Being able to successfully navigate a space and find the most efficient route is no different; even an incremental improvement in computational complexity could have massive effects. As Dell'Aversana points out, mazes can be used to model many multivariate optimization problems themselves!

The first simple algorithms for solving a maze likely included simply tracing the wall with one's hand, either the right or the left, but never changing hands and never allowing their hand to leave the wall. But if the start or end is not guaranteed to be on the edges of our maze, this algorithm likely fails to find any solution. If the traveler has their hand on a wall that is not connected to a wall that passes the exit then they may travel in a loop, endlessly lost in the maze.

If we can see the maze prior to traversing it, we have some advantages to solving it. We can see where the paths lead, and can identify when we've crossed back over our own path, something no shortest path will ever allow. We can cross off the dead-ends and potentially find paths that are shorter than others. We can analyze the maze. And as mathematicians, we can analyze it mathematically. We can turn it into an undirected graph, allowing us to see it as an organization of nodes and the paths that allow our passage to adjacent nodes.



We can look at each element of our maze as a binary value. Walls are either present or absent and nodes in our maze are either on the solution path or they are not. But a problem persists. We have an enormous amount of mazes that could be created. Some of them are solvable, many are not. And we cannot possibly even consider solving them all. For even a small maze, like our 4×4 example above, each of our 12 horizontal paths could have a wall or not. A wall in our undirected graph is represented by the absence of a path between corresponding nodes in the graph. The same is true for the 12 vertical paths between nodes in this 4×4 maze. Consider that our start position could be in any of our 16 nodes, and that our end position can occupy any of the remaining 15 nodes, for a total of $2^{12} \times 2^{12} \times 16 \times 15 = 4,026,531,840$ different possible combinations of this size. Many of these graphs become disconnected between the start and end nodes, and there is no solution to these mazes. Others are spanning trees, these are mazes where every node is connected through some number of steps and there are no loops present. For any start and end positions in a spanning tree maze, there is exactly one solution, and it is the only solution for that maze. Our 4×4 example has over $100,000$ spanning trees with exactly one solution for each combination of start and end nodes. For a 10×10 maze, there are over 5×10^{42} different possible mazes of this spanning tree type with exactly one solution, just for a single fixed start and end point (Knill). Clearly, solving by hand even a small fraction of the possible mazes we could encounter is a fruitless task. And when mazes are sufficiently large, even solving a single maze without computational assistance may be unreasonably difficult.

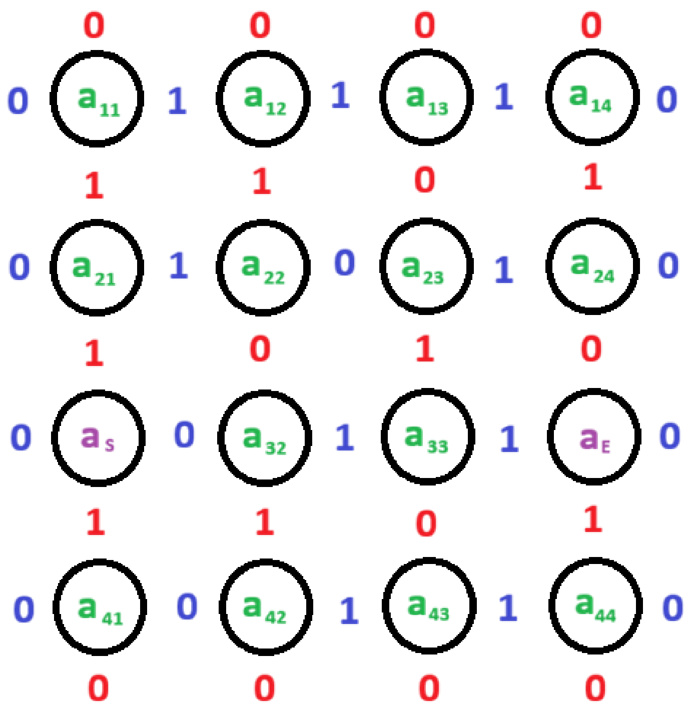
With the advent of computation, it became possible to more quickly parse the maze, even if it is very large, and search for the exit. Most successful maze solving processes rely on graph theory (Sadik, 53-54). Even if computers can help us when it comes to solving larger mazes, we run into other challenges. Far more frequently than spanning tree mazes occur, we find mazes where there are loops around portions of our maze. These conditions lead to some of the most challenging mazes to analyze. Loops threaten to confuse a computer into running around the same nodes endlessly, looking for an exit that it will never encounter. Multiple paths leave room for the first path found to be one that is wasteful and costly in the wrong applications.

2. Modeling a Maze for Optimization

If we are to think creatively about modeling mazes, then we can represent them mathematically and apply the techniques of Linear Optimization (Staab, 2.1.2). In order to properly apply linear optimization to our maze problem, we must be able to express it mathematically. There are several ways to do this, but the one chosen works well for our problem. We start with a two-dimensional maze of size $m \times n$, where m is the number of rows and n is the number of columns. Each cell has potential walls above and below, as well as to the left and to the right. Note that this leaves us with $n+1$ potential walls in the horizontal direction and $m+1$ possible walls in the vertical direction. To model this, we can take our maze and create two wall matrices. Let:

$$X: \; x_{ij} = \begin{cases} 0 & \text{if there is a wall to the left of the} \\ & \text{cell in the } i^{th} \text{ row and } j^{th} \text{ column.} \\ 1 & \text{if path the to the left is open.} \end{cases} \quad \text{and} \quad Y: \; y_{ij} = \begin{cases} 0 & \text{if there is a wall above the cell} \\ & \text{in the } i^{th} \text{ row and } j^{th} \text{ column.} \\ 1 & \text{if path above is open.} \end{cases}$$

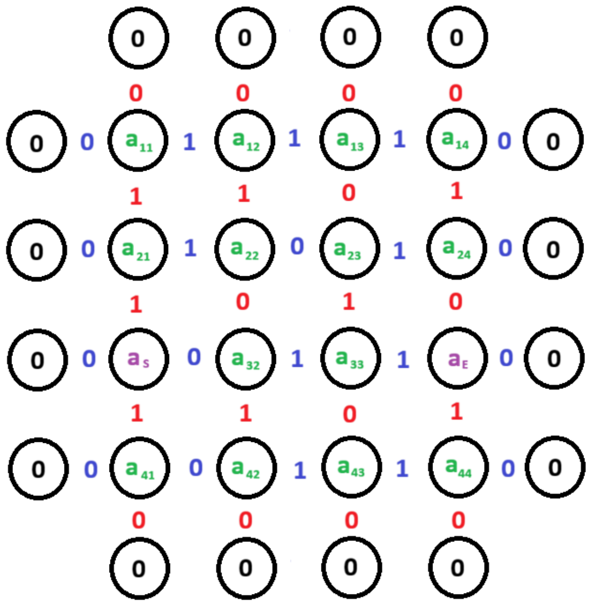
Let us call a cell accessible if there is no wall between an adjacent cell and that cell, and inaccessible if there is a wall. Also, so that we cannot simply just leave the maze, and make it to the exit in less steps than it would be possible within the maze, we should make sure that all of our $x_{i0} = 0$ and $x_{i(n+1)} = 0$, and similarly $y_{0j} = 0$ and $y_{(m+1)j} = 0$. These wall matrices will act as the coefficients in our LOP that denote accessibility of the cell adjacent to a cell in question.


0 0 0 0
0 a_{11} 1 a_{12} 1 a_{13} 1 a_{14} 0
1 1 0 1
0 a_{21} 1 a_{22} 0 a_{23} 1 a_{24} 0
1 0 1 0
0 a_S 0 a_{32} 1 a_{33} 1 a_E 0
1 1 0 1
0 a_{41} 0 a_{42} 1 a_{43} 1 a_{44} 0
0 0 0 0

So that we can solve for our shortest path, we must model our maze as a mathematical system. The first choice we must make is what kind of problem we are trying to build and which variables we will use. To choose the proper problem, we must think of what we are trying to find. Since we not only want to find a possible path, but also the shortest path, we will begin with a minimization problem. We will add up all of the cells that are on the path, and disregard the cells that are not on the path. To achieve this mathematically, we can see that the following formula and assignment of variable values will do the job:

$$\text{Minimize: } \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} \qquad a_{ij} = \begin{cases} 1 & \text{if the cell in the } i^{th} \text{ row and } j^{th} \text{ column is on the path.} \\ 0 & \text{if not.} \end{cases}$$

Now that we have formulated a minimization problem, we need to find a way to constrain which cells are on the path or not. But before we start worrying about our constraints, recognize that we have three different kinds of cells we would need to come up with constraints for; the corners, the edges, and the inner cells would all need to have their neighbors' status as on the path or not summed up in a different way. And the corners and the edges each have four different sets of orientations to account for. Instead of worrying about each of these cases individually, we can simply standardize our cells so that we can work on them each the same way. This can be achieved by buffering our maze with an extra row of cells on the top and bottom of the maze and an extra column of cells on the left and right. The values in these cells does not really matter, the rationale for which will be made clear shortly. So, let us buffer the edges of our maze with an extra row and column on each side of the maze, and constrain them to be equal to zero;



$$a_{i0} = a_{i(n+1)} = a_{0j} = a_{(m+1)j} = 0 \text{ for } 0 \leq i \leq m+1 \text{ and } 0 \leq j \leq n+1.$$

Now each cell in our maze has exactly four adjacent neighbors, and each cell can be treated the same as far as the constraints applied to them. Another route would have been defining the constraints differently for the different cells depending on their location in the maze relative to the outer walls. As the mazes get larger, the chosen solution of buffering the edges of the maze may prove to be computationally irresponsible, and may be revisited in future iterations of this project.

Our first two constraints are the most straight forward. First, let us define our start node to be define $a_{\hat{i}\hat{j}}$ to be the starting point and $a_{\tilde{i},\tilde{j}}$ to be the end point. Our start and our end cells must be on the path. So,

$$a_{\hat{i}\hat{j}} = 1, \quad \text{and} \quad a_{\tilde{i}\tilde{j}} = 1.$$

In order for our maze to have a solution, at least one cell next to the start needs to be on the path and accessible. In order for our path to be optimal, no more than one accessible cell on the path can be next to the start cell. If two or more were on the path and accessible from the start cell, the optimal solution would leave all but one of those cells off of the path, instead leaving the start cell by the remaining accessible cell in favor of the shortest path. Since the start must have at least one accessible neighbor on the path and can also have at most one accessible neighbor on the path, we know that our start node is constrained to have exactly one accessible neighbor on the path. Similarly, the exit also has exactly one accessible neighbor for the same reason.

$$a_{(\hat{i}-1)\hat{j}} + a_{\hat{i}(\hat{j}-1)} + a_{\hat{i}(\hat{j}+1)} + a_{(\hat{i}+1)\hat{j}} = 1$$
$$a_{(\tilde{i}-1)\tilde{j}} + a_{\tilde{i}(\tilde{j}-1)} + a_{\tilde{i}(\tilde{j}+1)} + a_{(\tilde{i}+1)\tilde{j}} = 1$$

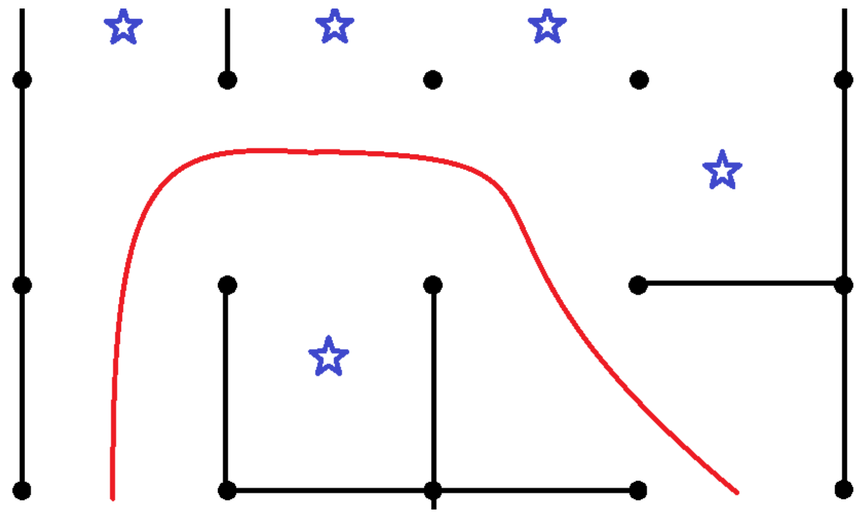
Once we get more than one step away from the start or end nodes, we must begin to think about other possible orientations of maze sections when building our constraints. Consider that for each a_{ij} on the path,

$$x_{ij} \cdot a_{i(j-1)} + y_{ij} \cdot a_{(i-1)j} + x_{i(j+1)} \cdot a_{i(j+1)} + y_{(i+1)j} \cdot a_{(i+1)j} = 2.$$

That is, there are exactly two accessible neighbors on the path for any cell on the path. If there was a third accessible neighbor on the path, then we would cut out a portion of the path, exit out the third accessible neighbor that we did not enter or exit from before, and exclude one of the accessible neighbors from the path. If a cell has four accessible cells all on the path, it means our solution passed through the cell at least twice, and our solution has a loop in it and no chance of being optimal. But if a neighbor is not accessible, i.e. there is a wall between them, then the zero value of the wall will exclude the neighbor on the path on the other side of the wall from our calculation. So each cell on the path will have exactly two accessible neighbors also on the path.

When we begin to look at cells of our maze that are not on our path, there are a couple of cases that are possible. A cell not on the path can clearly have no accessible neighbors on the path. The obvious example is one of a cell that is removed from the path by at least two cells, and who therefore has no adjacent neighbors on the path either. Another case would be a cell that is adjacent to, but not accessible from the path. In this case, the cell on the path does not influence the value of the cell in question.

A cell not on the path could also have one accessible neighbor on the path. An example is a dead end that branches from the optimal path. A cell adjacent to and accessible from the path that leads to a dead end will have an accessible neighbor on one side and not on any of the other three. In fact, many accessible and adjaent cells will be passed up by the optimal path, leaving them with one neighbor on the path and accessible. You can see in the example below, that there are 5 cells not on the path with one adjacent neighbor on the path, which have been marked with $\star$.



Perhaps less obviously are the cells that can be not on the path that have two accessible neighbors on the path. The clearest example to the author is one where there are four cells all open to each other in a square that the optimal path will pass through. The optimal path will pass through three out of four of these cells, but the fourth one will be left off of the path with two accessible neighbors on the path. An example of this can be found at the beginning of the Appendix.

And trivially, since all cell values are binary, every cell's sum of accessible neighbors on the path will be nonnegative. This constraint seemed unnecessary, but in the 4x4 example found in the Appendix, the single open cell on the path below the start caused the entire sum of all nodes in the matrix to be zero. It seems that a contradiction was created on that particular arrangement of conditions in the maze. More thoughtful investigation is needed to find if and why this constraint is truly necessary to the LOP.

All of these conditions combined, and all we can say for certain is that for each cell in the maze, the sum of its neighbors minus two times the cell's value (0 if not on path, 1 if on path) is less than or equal to zero and that the sum of all of its neighbors is greater than or equal to zero. This helps us to come up with two constraints for each cell in the maze not already explored:

$$x_{ij} \cdot a_{i(j-1)} + y_{ij} \cdot a_{(i-1)j} + x_{i(j+1)} \cdot a_{i(j+1)} + y_{(i+1)j} \cdot a_{(i+1)j} - 2 \cdot a_{ij} \leq 0.$$
$$x_{ij} \cdot a_{i(j-1)} + y_{ij} \cdot a_{(i-1)j} + x_{i(j+1)} \cdot a_{i(j+1)} + y_{(i+1)j} \cdot a_{(i+1)j} \geq 0.$$

And when we put it all together, we find that our Linear Optimization Problem for mazes of size mxn with walls denoted by matrices X and Y and with start cell $a_{\hat{i}\hat{j}}$ and end cell $a_{\tilde{i}\tilde{j}} is$:

3. Primal Feasibility and Dual Uniqueness

Using techniques found in Staab's book on Linear Optimization, we can put this LOP into standard form and build our simplex tableau from the coefficients found from this form. An example of a 3x3 maze primal tableau can be found in the Appendix. All of our cells are binary, and therefore nonnegative. Since there are no free variables, we may look past this initial step of Phase 0 of the simplex method. But before we can start to work towards a feasible solution, we need to bring some standard variables in the rows with no slack variables into the basis. This step is crucial to finding the certificate and verifying the dual problem later on. With no free variables to give preference to in our example in the Appendix, we should bring x_4, x_7, x_8, and x_9 into the basis. This completes phase zero of the simplex method.

Phase 1 begins with the question, "Is this feasible?" By looking at our objective row, we can see that since there are negative values in the far right column representing our vector $\vec{b}$, and our slack variables are also constrained to be nonnegative, that it is not. In Phase 1 of the simplex method, we will bring negative restricted variables into the basis, while being careful not to let any of the variables that we just brought in during Phase 0 to leave the basis during Phase 1. By finding the row with the largest negative value in the $\vec{b}$ column and selecting the basic variable associated with this row to leave the basis, we can pivot x_j into the basis, where j is the column with the leftmost negative in the row with the most negative $\vec{b}$ value. In our 3x3 example in the Appendix, it takes a couple of pivots to get there, but this task is easily automated by current computational tools, and the algorithm can be automated or followed by hand for small problems. In these mazes, it is common to run into issues where there is no negative value left in a row with a negative value left in the

Phase 2 asks if our solution is optimal? We can determine this by making sure there are no negative values in the objective row. If there is a negative value in the objective row, then our solution is not optimal. If there is a negative number in the objective row, let us choose the leftmost negative number in the objetive, and calculate the ratio of each value in the $\vec{b}$ column in the same row as each value in the column j in question. The row with the lowest nonnegative $\vec{b}$ ratio will denote the index of the leaving variable, and the column j will determine the index of the entering variable. We do this again and again until either no nonnegative $\vec{b}$ ratio exists, a negative appears in the $\vec{b}$ column, or all negatives are removed from the objective row. If there are no nonnegative $\vec{b}$ ratios, then the problem is unbounded. This solution is not reasonable for a maze bounded on all sides by walls, and is not likely to be encountered. One way to become unbounded would be to be stuck in an infinite loop, but the structure of our minimization problem would abandon a loop for a shorter path, if it exists. And if that shorter path does not exist then it will be infeasible, as discussed earlier.

Once we have an optimal solution from our primal problem, we can utilize the simplex tableau to obtain the certificate. We can check the certificate quickly to determine the uniqueness of our optimal solution. Let us explore our 3x3 example some more for clarity on the topic. From our solved simplex tableau in the appendix, we can see that our primal solution $\vec{x}\,'$ has the form:

$$\vec{x}\,' \;=\; (0,+,+,+,+,+,+,+,+ \mid +,0,0,0,0,0,0,0,0,0,0,0,0).$$

So, $\vec{y}'$ should be of form:

$$\vec{y}\,' = (0,*,*,*,*,*,*,*,*,*,*,*,* \mid *,0,0,0,0,0,0,0,0).$$

And from our objective row of our solved primal tableau, we can pluck off the parts of $y\,'$ to find that it has the form:

$$\vec{y}\,' = (0,0,*,*,0,*,*,0,0,0,0,0,0 \mid 0,0,0,0,0,0,0,0,0).$$

If our solution were unique, the forms of $\vec{y}\,'$ and $\vec{y}$ would match exactly. From our conflicting forms of $\vec{y}\,'$, we can see that our forms do not violate each other. This is another indication of optimality. But also, we could pivot our simplex tableau and retain a valid form of $\vec{y}\,'$. This tells us our solution is not unique. By pivoting our simplex tableau to let x_{10} leave the basis and x_{11} enter. Our new optimal solution gives $\vec{y}\,'$ the form:

$$\vec{y}\,' = (*,0,*,*,0,*,*,*,*,*,*,*,* \mid 0,0,0,0,*,0,0,0,0)$$

which is still a valid form for our dual certificate.

While this checking of the dual certificate is helpful in finding alternate solutions in small examples, the scale of larger problems may render this practice impractical. When utilizing software to solve these problems, we can add a random perturbation to our objective in order to weigh each cell slightly differently than the others. This practice shifts the optimal solution off of preferred paths in order to minimize the minuscule randomness introduced to find alternate solutions. It is important to keep the perturbations small, or the randomness may overcome the actual optimal solution to an indistinguishable degree. As an example, random perturbation was performed on the 10x10 example in the appendix, and both solutions can be found in binary matrix form.

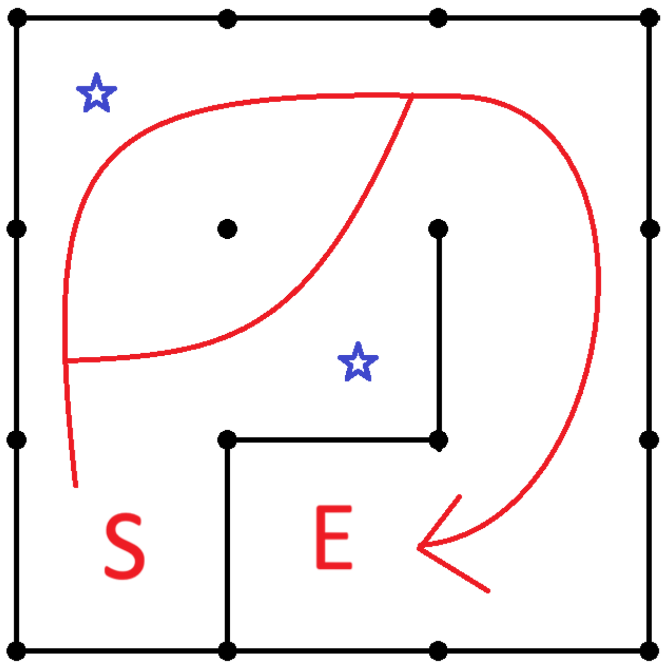
Our LOP has given us the values of each node in our maze. Remember that we have $m \times n$ variables. The first n of them are our values for each node in the first row. variables $n+1$ through $2n$ give us the values of the nodes in the second row; variables $2n+1$ through $3n$ are our third row, and so on for each row. Once put into the proper rows to match the shape of our maze, we may simply overlay our variable solutions of our LOP onto our maze, and the cells with 1's in them are part of our optimal solution. An example for a 10×10 matrix solution can be found at the end of the Appendix.

4. Findings and Further Questions

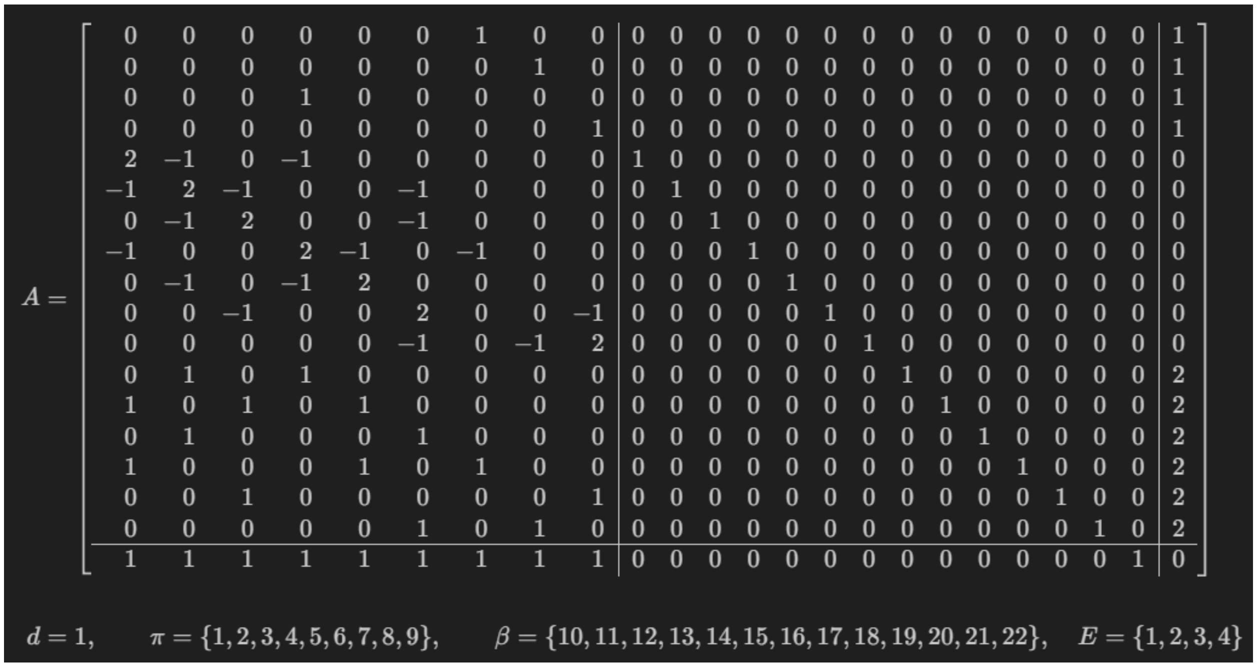
Thinking about our maze as a graph and modeling it as a collection of binary constants and variables allowed us to apply linear optimization not only to the task of checking our maze to see if it has a solution, but also for finding the shortest path. We also were able to identify the uniqueness of our problem, or lack thereof. This application of linear optimization seems well-suited for solving this problem, which is of interest across disciplines. There are still some problems that need to be addressed. The process of inputting the values that describe the maze is laborious and time-intensive. This task could be automated using software such as OpenCV to translate an image to its binary representation. Moving this reasoning to a three dimensional maze seems to be the next logical step, and one of great importance to AI automation. And while this method of maze solving seems to be effective, considerations should be made to how it scales computationally when compared to the more traditional and established algorithms utilized in solving mazes.

5. Appendix

3x3 Maze:



Simplex Tableau:



$$A = \left[\begin{array}{ccccccccc|cccccccccccccc|c}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
2 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 2 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 2 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 2 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & -1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 2 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 2 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 2 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 \\
\hline
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}\right]$$

$d = 1, \quad \pi = \{1,2,3,4,5,6,7,8,9\}, \quad \beta = \{10,11,12,13,14,15,16,17,18,19,20,21,22\}, \quad E = \{1,2,3,4\}$

Phase 0 Result:

$$A=\left[\begin{array}{ccccccccc|cccccccccccccc|c}
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
2&-1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
-1&2&-1&0&0&-1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&-1&2&0&0&-1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
-1&0&0&0&-1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&-1\\
0&-1&0&0&2&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1\\
0&0&-1&0&0&2&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&-1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&-1\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1\\
1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&2\\
0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&2\\
1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1\\
\hline
1&1&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&-4
\end{array}\right]$$

$d=1,\qquad \pi=\{1,2,3,5,6\},\qquad \beta=\{4,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22\},\qquad E=\{1,2,3,4\}$

Phase 1 Result:

$$A=\left[\begin{array}{ccccccccc|cccccccccccccc|c}
0&0&0&0&0&0&2&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2\\
0&0&0&0&0&0&0&2&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2\\
0&0&0&2&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2\\
0&0&0&0&0&0&0&0&2&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2\\
0&2&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2&0&0&0&0&0&0&2\\
0&0&2&0&0&0&0&0&0&0&0&1&0&0&0&-1&1&0&0&0&0&0&0&2\\
0&0&0&0&2&0&0&0&0&0&2&1&-2&0&0&-3&-3&0&0&0&0&0&0&2\\
2&0&0&0&0&0&0&0&0&0&-2&-1&0&0&0&3&3&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&-4&-2&4&2&0&6&8&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&2&3&1&0&0&0&0&0&0&0\\
0&0&0&0&0&2&0&0&0&0&0&0&0&0&0&-2&0&0&0&0&0&0&0&2\\
0&0&0&0&0&0&0&0&0&2&4&2&0&0&0&-6&-4&0&0&0&0&0&0&4\\
0&0&0&0&0&0&0&0&0&0&0&-1&2&0&0&1&-1&2&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2&-2&0&2&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&2&0&0&0&0&0&0&2&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&-1&0&0&0&1&-1&0&0&0&2&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2&0&0&0&0&0&2&0&0\\
\hline
0&0&0&0&0&0&0&0&0&0&0&-1&2&0&0&3&-3&0&0&0&0&0&2&-16
\end{array}\right]$$

$d=2,\qquad \pi=\{11,12,13,16,17\},\qquad \beta=\{1,2,3,4,5,6,7,8,9,10,14,15,18,19,20,21,22\},\qquad E=\{1,2,3,4\}$

Phase 2 Result:

$$A = \left[\begin{array}{ccccccccc|cccccccccccccc|c}
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&1&0&0&0&0&0&0&0&0&0&-1&0&0&-2&-3&0&0&0&0&0&0&0&1\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&-1&-2&0&0&0&0&0&0&0&1\\
0&0&0&0&1&0&0&0&0&0&1&2&-1&0&3&3&0&0&0&0&0&0&0&1\\
1&0&0&0&0&0&0&0&0&0&-1&-2&0&0&-3&-3&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&-2&-5&2&1&-8&-9&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&2&3&1&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&-1&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&1&2&3&0&0&4&3&0&0&0&0&0&0&0&2\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&2&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&2&4&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&2&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0\\
\hline
0&0&0&0&0&0&0&0&0&0&0&1&1&0&3&6&0&0&0&0&0&0&1&-8
\end{array}\right]$$

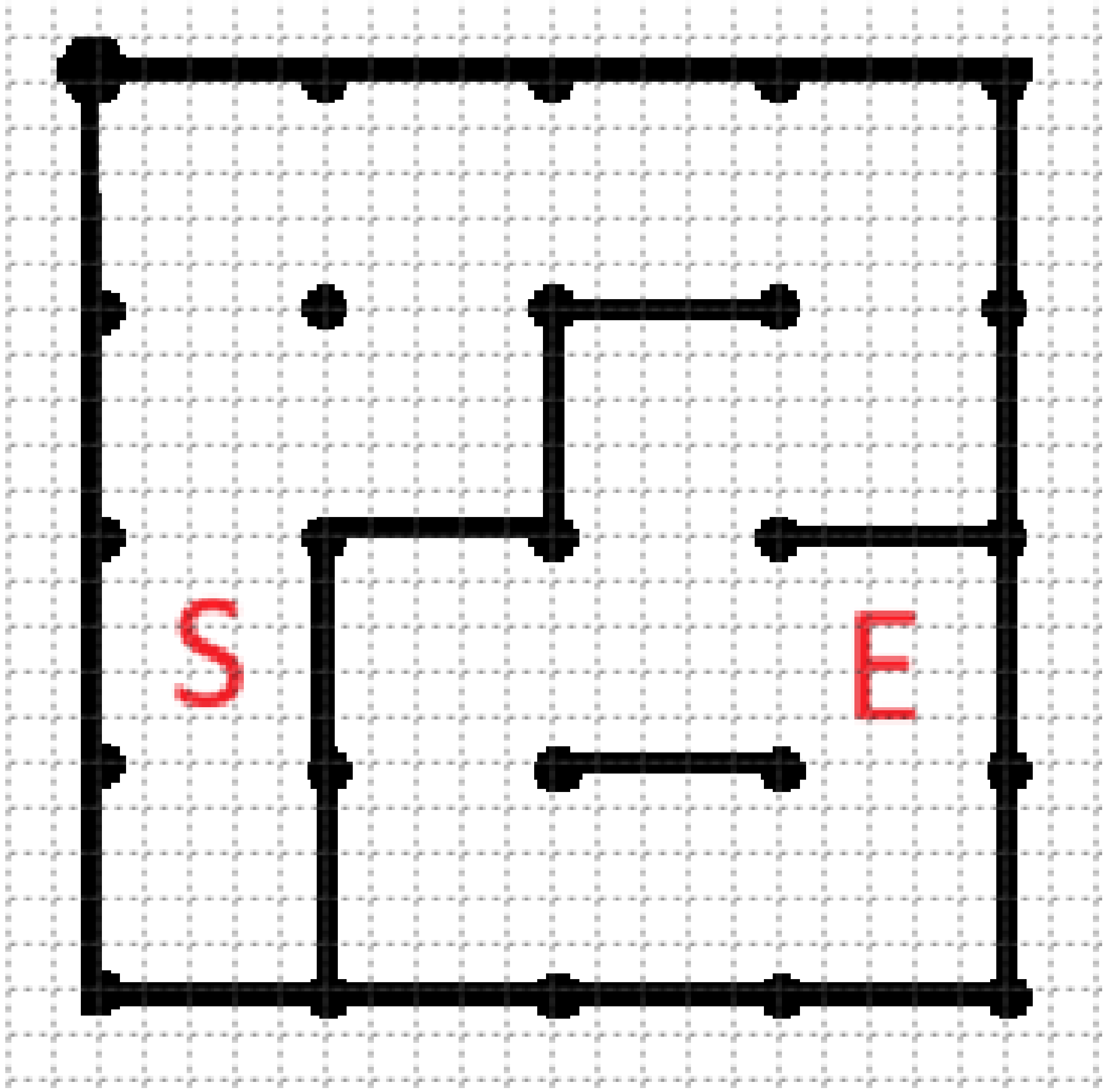
$d = 1, \qquad \pi = \{11, 12, 13, 15, 16\}, \qquad \beta = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14, 17, 18, 19, 20, 21, 22\}, \quad E = \{1, 2, 3, 4\}$

Optimal Tableau Second Solution:

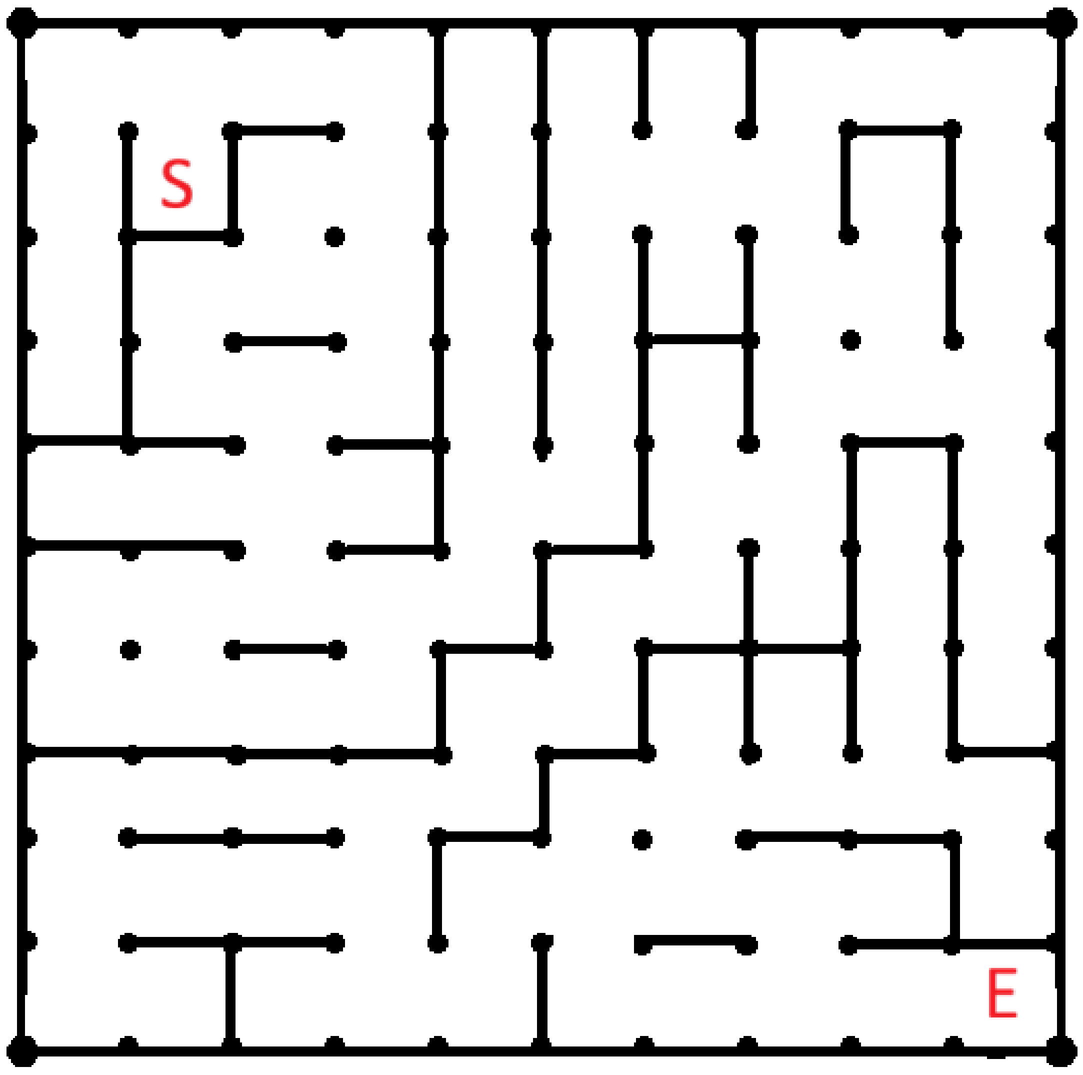
$$A = \left[\begin{array}{ccccccccc|cccccccccccccc|c}
0&0&0&0&0&0&2&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2\\
0&0&0&0&0&0&0&2&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2\\
0&0&0&2&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2\\
0&0&0&0&0&0&0&0&2&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2\\
0&2&0&0&0&0&0&0&0&0&0&-2&0&0&-4&-6&0&0&0&0&0&0&0&2\\
0&0&2&0&0&0&0&0&0&0&0&0&0&0&-2&-4&0&0&0&0&0&0&0&2\\
0&0&0&0&2&0&0&0&0&-1&0&1&-2&0&2&3&0&0&0&0&0&0&0&0\\
2&0&0&0&0&0&0&0&0&1&0&-1&0&0&-2&-3&0&0&0&0&0&0&0&2\\
0&0&0&0&0&0&0&0&0&2&0&-4&4&2&-8&-12&0&0&0&0&0&0&0&4\\
0&0&0&0&0&0&0&0&0&0&0&2&0&0&4&6&2&0&0&0&0&0&0&0\\
0&0&0&0&0&2&0&0&0&0&0&0&0&0&0&-2&0&0&0&0&0&0&0&2\\
0&0&0&0&0&0&0&0&0&1&2&3&0&0&4&3&0&0&0&0&0&0&0&2\\
0&0&0&0&0&0&0&0&0&0&0&0&2&0&2&4&0&2&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&2&0&0&4&8&0&0&2&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&2&0&0&0&0&0&0&2&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&2&4&0&0&0&0&2&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2&0&0&0&0&0&2&0&0\\
\hline
0&0&0&0&0&0&0&0&0&0&0&2&2&0&6&12&0&0&0&0&0&0&2&-16
\end{array}\right]$$

$d = 2, \qquad \pi = \{10, 12, 13, 15, 16\}, \qquad \beta = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 14, 17, 18, 19, 20, 21, 22\}, \quad E = \{1, 2, 3, 4\}$

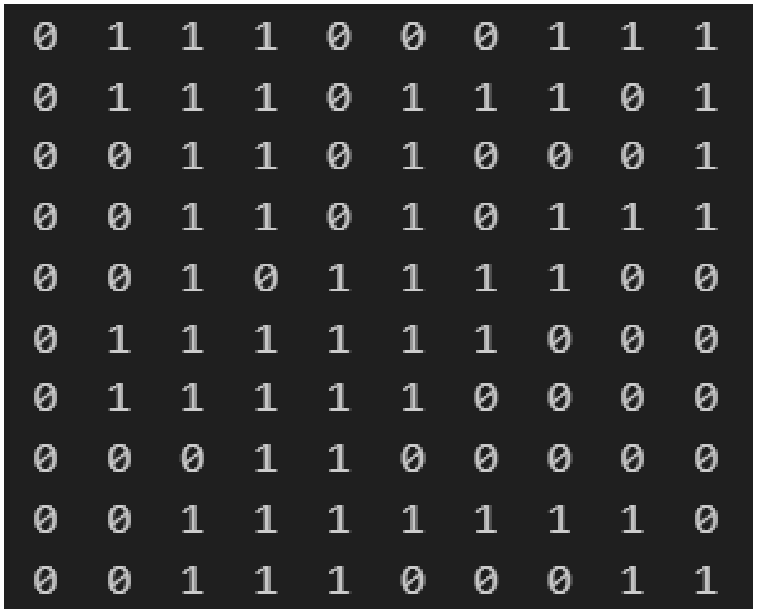
4x4 Maze:



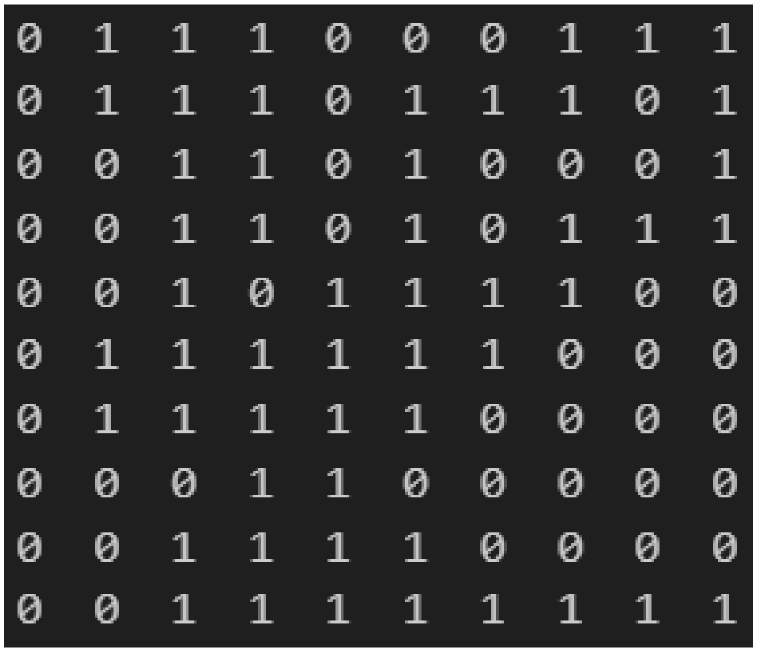
10x10 Maze:



10x10 Maze Optimal Solutions:

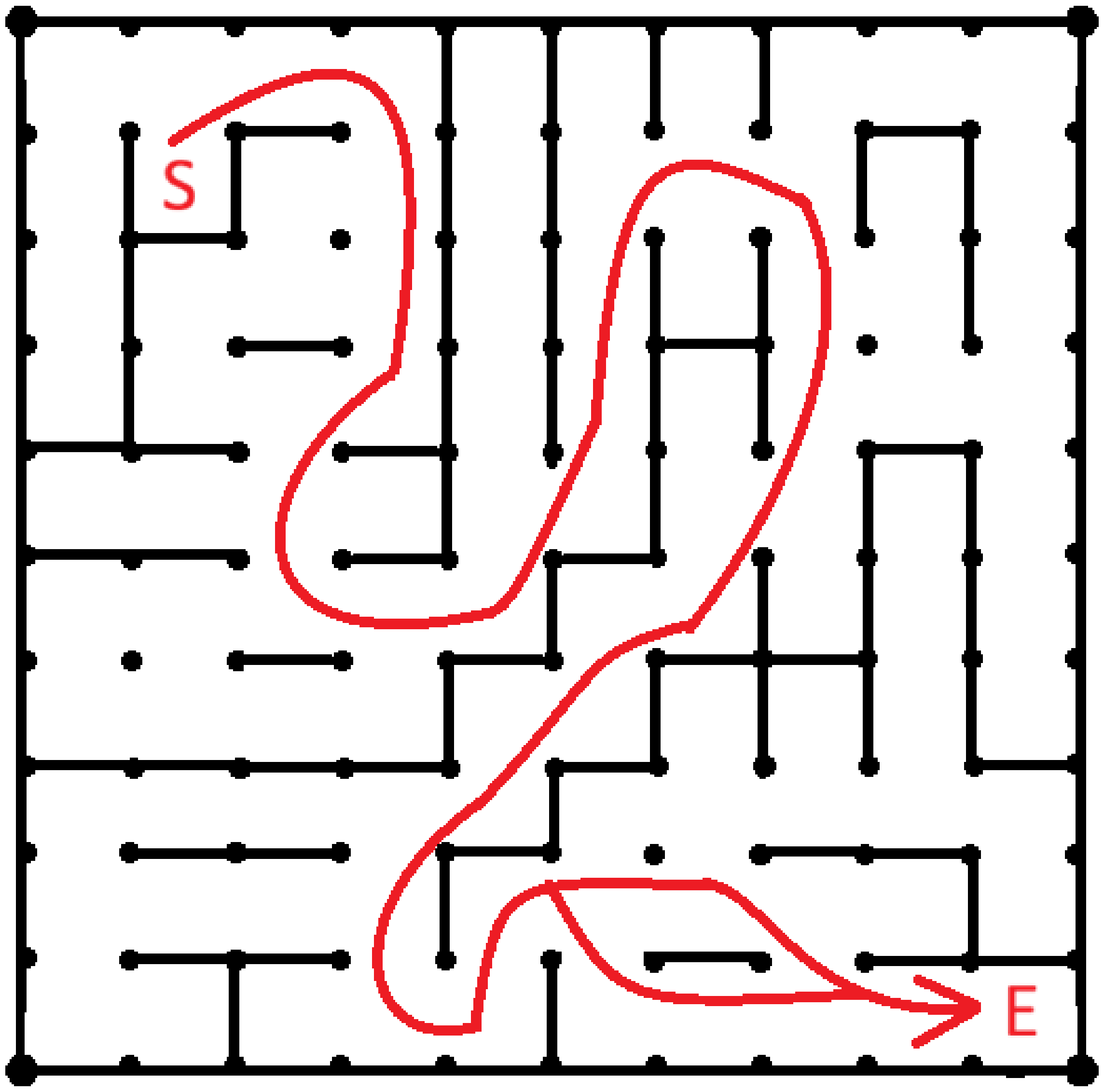


```
0  1  1  1  0  0  0  1  1  1
0  1  1  1  0  1  1  1  0  1
0  0  1  1  0  1  0  0  0  1
0  0  1  1  0  1  0  1  1  1
0  0  1  0  1  1  1  1  0  0
0  1  1  1  1  1  1  0  0  0
0  1  1  1  1  1  0  0  0  0
0  0  0  1  1  0  0  0  0  0
0  0  1  1  1  1  1  1  1  0
0  0  1  1  1  0  0  0  1  1
```



```
0  1  1  1  0  0  0  1  1  1
0  1  1  1  0  1  1  1  0  1
0  0  1  1  0  1  0  0  0  1
0  0  1  1  0  1  0  1  1  1
0  0  1  0  1  1  1  1  0  0
0  1  1  1  1  1  1  0  0  0
0  1  1  1  1  1  0  0  0  0
0  0  0  1  1  0  0  0  0  0
0  0  1  1  1  1  0  0  0  0
0  0  1  1  1  1  1  1  1  1
```

10x10 Maze Solution:



6. Works Cited

Dell'Aversana, P. (2022, June 1). Analogies between maze solving and optimization problems. ResearchGate. https://doi.org/10.13140/RG.2.2.21128.49928

Knill, O. (2022). Lecture 1: The mathematics of mazes. Harvard University. https://people.math.harvard.edu/ knill/teaching/mathe320_2022/handouts/00-worksheet.pdf

Sadik, A. M. J., Dhali, M. A., Farid, H. M. A. B., Rashid, T. U., & Syeed, A. (2010). A comprehensive and comparative study of maze-solving techniques by implementing graph theory. In 2010 International Conference on Artificial Intelligence and Computational Intelligence (pp. 52-56). IEEE. https://doi.org/10.1109/AICI.2010.18

Staab, P. (2025). Linear optimization: Modeling and solving linear problems with Julia and WebCAS. Fitchburg State University. https://pstaabp.github.io/linear-optimization-book/frontmatter.html