Playing With Mazes

David B. Suits
Department of Philosophy
Rochester Institute of Technology
Rochester NY 14623

david.suits@rit.edu

Copyright © 1994 David B. Suits

3. Solving Mazes

Once a maze is known to be connected, there are probably countless methods of automating a solution. I shall present some simple methods and some complex methods.

3.1 Random Walk

Beginning in the start cell, a random walk from cell to cell could eventually lead to the goal cell. If directions are chosen pseudo-randomly such that the distribution of choices in the long run covers all possibilities, then of course the goal cell will eventually be reached. Obviously, though, this method is better thought of as a non-method.

3.2 The Left- (or Right-) Hand Walk

Starting at the start cell, keep your left (or right) hand touching a wall. Walk. Eventually (with certain kinds of mazes) you will reach the goal cell. The maze in figure 13 is amenable to this solution, but the maze in figure 15 is not. The reason is that the maze in figure 15 has multiple paths between some cells (it is ***multiply connected***). Suppose cell 5 is the start cell and cell 9 is the goal cell. A left-hand walk starting in cell 5 and facing east (or a right-hand walk starting in cell 5 facing west) will consist of an infinite loop traversing cells 5, 6, 3, 2, 1 and 4. Cells 7, 8 and 9 will never be visited. A left-hand walk beginning at cell 5 but facing west, or a right-hand walk beginning in cell 5 facing east, will, however, find a solution.

An improvement on this method is to keep track of cells visited. If you return to a visited cell, there is a loop, and to break out of it, switch from a left- (right-) hand walk to a right- (left-) hand walk. This method will solve the maze in figure 15.

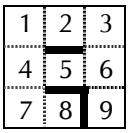

Figure 15. A multiply connected maze which cannot necessarily be solved by a left-hand or right-hand walk.

But both the simple and improved versions of this method will fail to solve mazes where the goal cell is a "King's chamber", i.e., a cell none of chose corners is part of a wall. Such a maze is shown in figure 16.

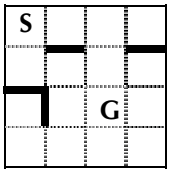

Figure 16. The left-hand or right-hand walk does not guarantee a solution to a maze which the goal cell is a "King's Chamber".

3.3 Least Recently Used Walk

A simple (but not speedy) algorithm which guarantees the solution of any maze extends the idea of marking the cells in order to take note of revisits. But in this case it is the doors, and not the cells themselves, which are marked, and the goal cell is (eventually) discovered by exiting whatever

cell you are in through the least recently used door.

1. Mark all doors with "0".
2. Set $i \leftarrow 0$.
3. If the present cell is the goal cell, stop.
4. $i \leftarrow i + 1$.
5. Choose the door with the lowest number. (If more than one door has the same lowest number, randomly choose between them.)
6. Mark the door as *i*, and exit through that door.
7. In the neighboring cell, mark the door through which you came as *i*.
8. Go to step 3.

In case the maze is so large that an automated solution requires numbers larger than the representative power of the machine being used, it may help to realize that it is not necessary to increment *i* in the manner given in the algorithm above. Rather, *i* need be set only to one more than the highest number in the present cell. So for step 7 we could substitute:

7′. In the neighboring cell, set *i* to the highest number in the cell + 1, and mark the door through which you came as *i*.

Even smaller numbers can be maintained by any simple scheme (not to be detailed here) which always renumbers the doors of a cell just entered from 0 (least recently used) incrementally to most recently used (i.e., the door you just came through). I mention this possibility only because such simplifying measures were necessary for a version of this algorithm which I implemented on a small, programmable calculator (a TI-58, circa 1977).

The main virtue of the Least Recently Used algorithm is that its memory requirements are minimal. The main vice of the algorithm is that since many cells might be visited many times, its time requirements can be huge. Another virtue of the method is that once a path from the start to the goal has been achieved, a path back to the start cell is available by going through doors with the highest value (not counting the door through which you entered the present cell). But another vice of the algorithm is that it does not guarantee to find the *shortest* path to the goal cell; indeed, it does not necessarily find all paths to the goal cell, so that even if you were able to pick the shortest of the paths traversed, there might be even shorter paths not yet investigated.

Pruning algorithms may be added to this (and to most any other) solution algorithm. For example, a cul-de-sac, once discovered, might be specially marked (say, with a negative number) so that it is henceforth no longer used. By this means, a tunnel (a string of one or more cells with exactly two doors) which leads into a cul-de-sac will itself be incrementally closed off. That having been done, a solution path can be more efficient. Sometimes a loop can be discovered and marked so that it will not be traversed again. But this requires significantly more memory resources (and a little bit more time en route) to manage, and so I will not discuss such methods here. (See, for example, Allen 1979.)

3.4 Breadth-First Search

The breadth-first search algorithm labels cells (not their doors), searching from the start cell to all its immediate neighbors. If the goal cell is not found, the search is performed outward to the neighbors of each of the neighbors of the start cell; and so on until the goal cell is found. The algorithm keeps track of which cells are immediate neighbors of the start cell, and which cells are neighbors of those immediate neighbors, etc., by labeling each set of cells (neighbors, then neighbors of neighbors, and so on) with higher and higher numbers.

1. Label the start cell as 0.
2. $i \leftarrow 0$.
3. For each cell labeled *i*, label all unlabeled adjacent cells with $i + 1$. (If there are no such adjacent cells, stop; the maze is unconnected.)
4. If any of the newly labeled cells is the goal cell, stop; a solution path has been found.
5. $i \leftarrow i + 1$.
6. Go to step 3.

The results of a breadth-first search are shown in figure 17. Notice that a solution path has been found. (It is possible that in a multiply connected maze multiple solution paths will be marked out.) Unfortunately, such a search is best

performed in parallel; for a poor traveler trying to find his way through the maze, the breadth first search method would involve so much backtracking as to make the algorithm very tedious (although perhaps not as bad as the Least Recently Used Algorithm). Moreover, additional memory is required to keep track of what "level" of the search one is on, and, for each given cell at the previous level, whether its immediate neighbors have been labeled at the new level. In addition, the cells, once having been labeled, do not provide a very efficient means of traveling that same path again without going through another breadth-first search. The labeling does, however, provide an excellent means of returning to the start cell from the goal cell: simply move to the neighbor cell with the lowest number. As a bonus, the number in a cell also tells you how many cells remain between your present position and the start cell. (Since breadth-first search might mark out multiple solution paths, there could also be multiple return paths.)

The breadth-first search method, if started at the goal cell in search of the start cell, will provide a clear (and shortest) path from the start to the goal. (Just move to that neighbor with the lower number.) And the label in a cell will indicate the distance to the goal. Of course, retracing one's path will involve as much backtracking as finding the goal cell did in the original version.

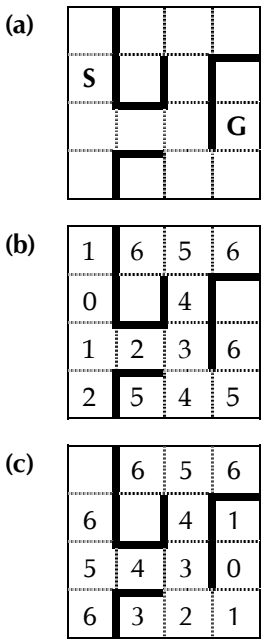

Figure 17. (a) A 4×4 maze with start and goal cells. (b) The final labelling generated by breadth-first search. (c) A breadth-first search started from the goal cell back to the start cell.

3.5 Tremaux's Algorithm

Tremaux's algorithm (see Even, 1979, pp. 53f) is a depth-first search used to visit all cells, terminating in the original cell after having examined all doors once in each direction. (The Hopcroft-Tarjan version of depth first search is described in Even 1979, p. 56. Variations on Tremaux's algorithm are given in Even 1979, pp. 66f.)

1. Make all doors unmarked.
2. Choose some cell to be *p*, the present cell.
3. If there are no unmarked doors, go to step 7.
4. Choose any unmarked door, mark it "E", and go through to the neighboring cell.
5. If this neighbor has any marked doors (i.e., if the cell has been visited before) mark the door back to *p* with "E", go back to *p*, and go to step 3.
6. This neighbor has not been visited before. Mark the door back to *p* with "R", assign this cell to *p*, and go to step 3.
7. If there is no door marked "R", stop; we are back in the start cell, and the whole maze has been visited.
8. Go through the door marked "R", assign this new cell to *p*, and go to step 3.

Figure 18 shows the progress of this algorithm. If it is necessary only to find *some* path to the goal cell (rather than to visit every cell), then the following step could be substituted for step 6:

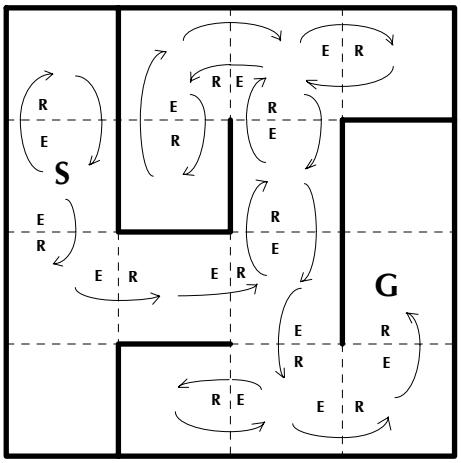

Figure 18. Tremaux's algorithm at work.

6'. Mark the door back to *p* with "R" and assign this cell to *p*. If *p* is the goal cell, stop. Otherwise, go to step 3.

Tremaux's algorithm will find the goal cell (provided, of course, that the maze is connected), and in addition it will provide a clear return path to the start cell. It also has the advantage of marking each door only once, unlike the Least Recently Used algorithm. However, it does not provide any means of repeating the solution path, except by once again solving the maze. But this can easily be remedied. During the retracing, use a counter, initialized to zero, and incremented by one each time you follow the "R" marks to a neighboring cell; replace the "E" on the door through which you just came with the value of the counter. Now a permanent path is given from the start to the goal simply by following the numbered doors. What's more, the numbers on the doors will indicate the number of steps necessary to get to the goal cell. Unfortunately, if there are any "cycles" in the maze — if, that is, it is multiply connected — the solution path is not guaranteed to be the shortest.

Another variation on the original algorithm is to include a test for the goal cell, at which time, instead of halting or simply retracing our steps (even with a counter), we continue the search, this time with the help of a counter. New cells found are incrementally marked, and during backtracking we decrement the counter. We will have to revisit some cells in order to mark them with counters. But when the process is complete — when we are back in the start cell after having visited all cells (some twice) — then each cell will have exactly one door marked with a number indicating the number of steps (through that door) from the start to the goal. (I suspect that this path will also be the shortest path, but a proof of that is, as they say, left up to the reader.)

3.6 The Cheese Algorithm

All the previous methods of solving a maze relied on a maze traveler's own discoveries during movement though the maze. I suggested earlier in the discussion of the breadth-first search algorithm that if the search were reversed so that investigation proceeded outward from the goal cell, rather than outward from the start cell, then the maze traveler would have a clearly marked path to the goal cell. But how, in terms of a traveler in a maze, are we to conceive of such a backwards search from the goal? If we already knew where the goal cell was, we wouldn't have to search at all.

Suppose, however, we present the maze problem as a simulation of two events taking place simultaneously: the first is the traveler, trying to find a solution path as usual. The second is some kind of information from the goal cell being broadcast outwards. Imagine, then, that in the goal cell is a piece of cheese — perhaps Limburger cheese — the odor of which gradually permeates the entire maze. Naturally, the strength of the odor in any given cell will be proportional to the distance of that cell from the cheese itself. The maze traveler (which we might as well concede is a rat) need only stay put in the start cell (or, if the rat is smart enough, it might initiate one of the earlier search algorithms) until it smells the cheese, at which time it follows the rule: proceed in the direction of strongest cheese odor. And as cells are gone through, they may be marked so that retracing the path back to the start cell is possible, and so that the path from the start to the goal may be followed at any future time without again having to search.

The strength of the cheese smell can be simulated by attaching a number (say between 0 and 1) to each cell. Let us call such a number the *Total Aroma Rating*, or *TAR*. ("TAR" is "RAT" going backwards.) The cheese cell will have a constant TAR of 1, and all other cells will begin with a TAR of 0. After each unit of simulated time, the strength of the odor in a given cell will be the result of the aroma the cell already has, minus what it loses to all its immediate neighbors, plus whatever it gains from its neighbors. For the sake of simplicity, we may say that the TAR of a cell will be equal to the average of its own TAR plus its neighbors' TARs.

Although spreading TAR is similar to a breadth-first search from the goal to the start, there are some differences. TARs change, whereas in the breadth-first search, a cell once labeled retains that label. And if there are cycles in the maze, odor will spread along two paths and then tend to disturb each other as they join up later. The question is, will the rat be able to find the cheese simply by following increasing TARs?

The answer is, almost obviously, Yes. But instead of constructing a formal proof, a simulation was employed. Figure 19 shows the 9 by 9 cell test maze after a number of iterations as the TAR activation spreads. A number of such runs were made, with the expected results.

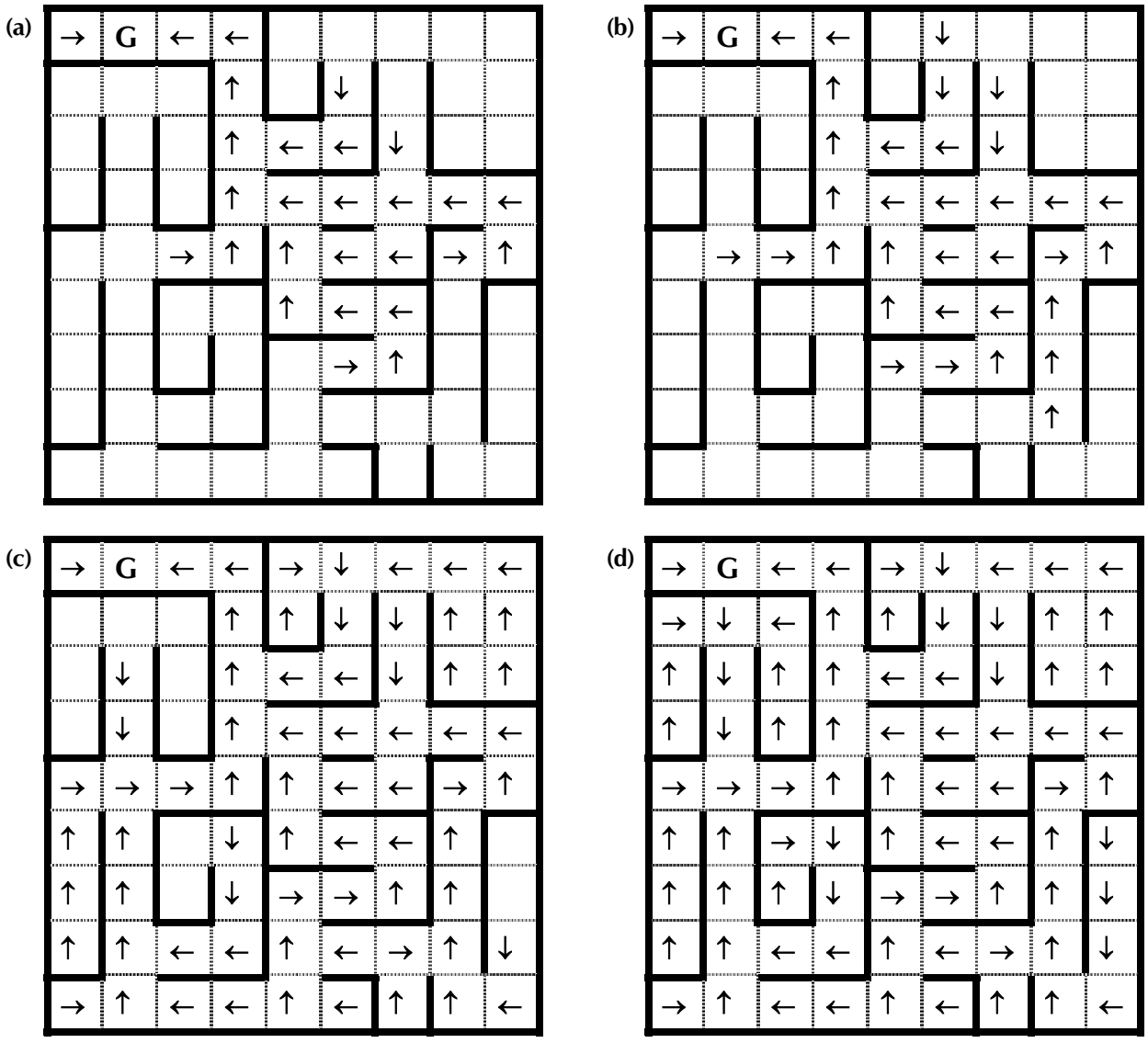

Figure 19. TAR spreading throughout a maze. The cheese is in the goal cell (G). Although the TAR for each cell changed over time, the direction of greatest TAR for each cell changed only rarely. (Arrows indicate, for each cell, the direction of greatest aroma.)