GRESHAM

COLLEGE

GAMES, GRAPHS AND GASKETS

A Lecture by

PROFESSOR IAN STEWART MA PhD FIMA CMath
Gresham Professor of Geometry

9 October 1996

GRESHAM COLLEGE

Policy & Objectives

An independently funded educational institution, Gresham College exists

- to continue the free public lectures which have been given for 400 years, and to reinterpret the 'new learning' of Sir Thomas Gresham's day in contemporary terms;

- to engage in study, teaching and research, particularly in those disciplines represented by the Gresham Professors;

- to foster academic consideration of contemporary problems;

- to challenge those who live or work in the City of London to engage in intellectual debate on those subjects in which the City has a proper concern; and to provide a window on the City for learned societies, both national and international.

Gresham College, Barnard's Inn Hall, Holborn, London EC1N 2HH
Tel: 020 7831 0575 Fax: 020 7831 5208
e-mail: enquiries@gresham.ac.uk

Gresham Geometry Lecture
9 October 1996

Games, Graphs, and Gaskets

Mazes are a common form of puzzle. The aim of this lecture is to show that the same principles that can be used to solve mazes can also be used to solve 'logical mazes' --- mathematical problems of many kinds.

One way to solve a maze is to look at a map and block off all the dead ends. But what if you don't have a map? Well, you *can* try the left-hand-on-wall trick: when you first enter the labyrinth --- or any maze, come to that --- you put your left hand on the wall, and keep it there. That guarantees that eventually you'll find your way back to the start; but not necessarily that you will find your way to the desired goal in the maze. (You won't get lost, at least.) What happens is that you traverse one complete connected system of walls, and if that system isn't connected to the goal, it gets you nowhere useful.

There is a more abstract formulation of the whole problem. The most important thing about a maze is how the junctions connect up. The lengths of the passages don't affect the path you take to get out, just how long it takes. So threading a maze is a topological problem. We can represent the topological essentials by a *graph*: its nodes correspond to the junctions in the maze, its edges correspond to the tunnels. The problem of getting out of a maze --- or of finding a particular place within it --- thus become that of *traversing a graph* from one node to another.

There is a key theorem that governs the possibility of doing this. *Two nodes can be joined by a continuous path if and only if they lie in the same connected component of the graph.* A *connected component* is the set of all nodes that can be reached from a given one by a continuous path. So what the therorem says is that two nodes can be joined by a continuous path if and only if there exists a continuous path that joins them. This may sound trivial, and it is, but it points to an important concept: connectivity. However, it's useless for solving the problem.

A more constructive approach is to devise a maze-threading *algorithm*. The word comes from the arab mathematician Muhammad ibn Musa abu Abdallah al-Khorezmi al-Madjusi al-Qutrubilli. 'Al-Khorezmi' became 'al-Gorizmi', then 'algorism', and finally 'algorithm'. It's used to describe a specific procedure, a computer program. The first general maze-threading algorithm was invented around 1892 by M. Trémaux. It was rediscovered nearly a century later by J.Hopcroft and R.Tarjan in the context of graph theory, which is just the same as mazes, really. They named it *Depth First Search* or the DFS algorithm: it goes like this.

Depth First Search

This visits all nodes in the same connected component as the starting node: in particular it can if desired be terminated when it hits a particular 'finishing' node.

- Begin at any chosen node.
- Visit any adjacent node that has not yet been visited.
- Repeat this as far as possible.
- If all adjacent nodes have been visted already, backtrack through the sequence of nodes that have been visted until you find one that is adjacent to an unvisited node: then visit that one.
- Delete any edge that has been backtracked.
- Repeat until you return to the starting node and there are no unvisited nodes adjacent to it.
- Then you have visited *all* nodes in the connected component of the graph that contains the starting node.

Depth First Search is especially appropriate for threading mazes, because it is possible to program the algorithm so that it can be carried out without having a map of the maze. It involves only *local* rules at nodes, plus a record of nodes and edges already used. You can *explore* the graph and traverse it as you go. The name is fairly appropriate: the idea is to give top priority to pushing deeper into the maze. The algorithm is quite efficient: the number of steps is at most twice the number of edges in the graph.

Wolf, Goat, Cabbage

Many puzzles are really maze-threading problems in disguise: the objective is to thread a *logical* maze. For example, recall the famous old problem of a farmer who has to cross a river. He has with him a wolf, a goat, and a cabbage. The boat can hold only the farmer plus one item of 'produce'; but he can't leave the wolf alone with the goat, because the goat will get eaten, and he can't leave the goat alone with the cabbage, because the cabbage will get eaten. What does he do? This puzzle is usually attributed to the mediaeval mathematician Alcuin (735-804). It is certainly quite ancient, and appears in Ozanam's *Récréations Mathématiques et Physiques* of 1694.

The important thing is which side of the river each of the three items is on. We can represent the position of a single item by the digits 0 and 1, using 0 to represent *this* side of the river and 1 to represent the far side. Thus the configuration of all three items is represented by a triple (w,g,c) in three-dimensional wolf-goat-cabbage space. For example (w,g,c) = (1,0,1) represents w = 1, g = 0, c = 1; that is, the wolf on the far side, the goat on this side, and the cabbage on the far side.

How many configurations are there? Well, each coordinate w, g, or c can take one of the two values 0 or 1. Thus there are 2x2x2 = 8 possibilities. What's more, they have a beautiful geometric structure: they are the eight vertices of a unit cube in wolf-goat-cabbage space (**Fig.1**).

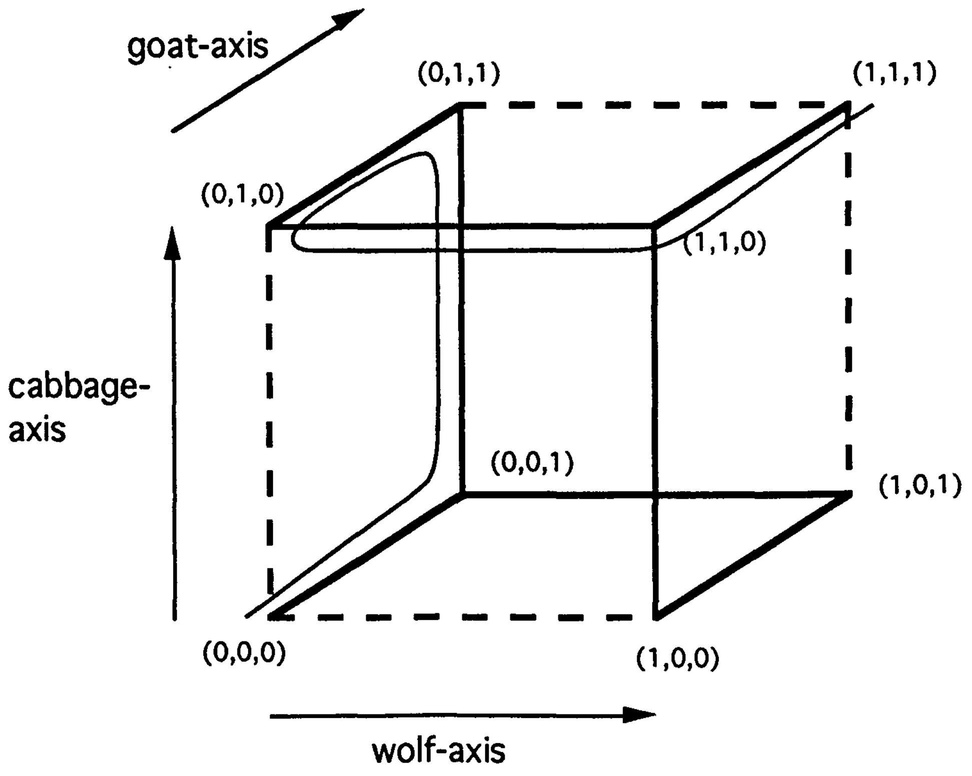

Fig.1 One solution. Find the other.

We may move only a single item at a time; that is, we may traverse only the edges of the cube. But some edges are forbidden. For example, the edge from (0,0,0) to (1,0,0) corresponds to taking the wolf across the river on its own. But this leaves goat and cabbage unchaperoned, so we would shortly be greeted by a fat goat and no cabbage. In fact these gastronomic constraints rule out exactly *four* edges, which are drawn in grey. The rest, representing permissible moves,will be coloured black.

The problem thus geometrized becomes: can we start at (0,0,0) --- all items on *this* side --- and get to (1,1,1) --- all items on the other side --- passing only along green edges of the cube? And of course the answer is 'yes'. Indeed, from a topological viewpoint, we can lay the edges out flat, and the solution stares us in the face. There are two solutions, in fact, and *only* two ifwe avoid unnecessary repetitions. They differ only by a wolf/cabbage symmetry operation.

How to Get Across With Your Produce Intact

Solution 1

(0,0,0) Start
(0,1,0) Take goat over
(0,1,1) (Return and) take cabbage over
(0,0,1) Bring back goat
(1,0,1) Take wolf over
(1,1,1) (Return and) take goat over.

Solution 2

(0,0,0) Start
(0,1,0) Take goat over
(1,1,0) (Return and) take wolf over
(1,0,0) Bring back goat
(1,0,1) Take cabbage over
(1,1,1) (Return and) take goat over.

Notice what we did to represent the puzzle geometrically. Each possible position in the puzzle can be thought of as a node in a graph, and each move from one position to another as an edge joining the corresponding nodes. You can also use depth first search instead of visual inspection. Here's the general procedure when you do not have a 'map' of the graph, but can only construct it edge by edge as you go.

Using Depth First Search to Solve Puzzles.

Many puzzles involve the movement of people, animals, or objects from a given position to a selected finishing position, subject to various rules. If the number of possible positions is finite, then Depth First Search can be employed. It takes the following form:

- Whenever you first come to a new position (including the initial position at the start of the puzzle) list all possible positions that can be reached from it.
- From the starting position, make any move that leads to a "new" position (one that has not yet occurred), at random.
- Repeat this as far as possible.
- If all possible moves lead to "old" positions, backtrack through the sequence of moves that have been made until you find a position from which it is possible to move to a "new" position. Then make that move.
- Henceforth ignore any move that has been backtracked.
- Repeat until you either reach the desired position, or return to the starting position with no moves available, in which case the puzzle is impossible.

We'll apply this to the wolf-goat-cabbage puzzle. We represent the participants by F = farmer, W = wolf, G = goat, C = cabbage, and positions by symbols such as [FG || WC] where the symbol || ("Styx") represents the river. Here farmer and goat are on the left bank, wolf and cabbage on the right.

step	current position	possible moves	move made	comments
1	[FWGC ‖ -]	[WC ‖ FG]	[WC ‖ FG]	starting position
2	[WC ‖ FG]	[FWGC ‖ -]		old position
		[FWC ‖ G]	[FWC ‖ G]	forced by algorithm
3	[FWC ‖ G]	[WC ‖ FG]		old position
		[W ‖ FGC]	[W ‖ FGC]	random choice of available moves
		[C ‖ FWG]		other possible choice
4	[W ‖ FGC]	[FWC ‖ G]		old position
		[FWG ‖ C]	[FWG ‖ C]	forced by algorithm
5	[FWG ‖ C]	[W ‖ FGC]		old position
		[G ‖ FWC]	[G ‖ FWC]	forced by algorithm
6	[G ‖ FWC]	[FWG ‖ C]		old position
		[FCG ‖ W]	[FCG ‖ W]	random choice (bad move but it
		[FG ‖ WC]		shows how the algorithm copes)
7	[FCG ‖ W]	[G ‖ FWC]		old position
		[C ‖ FWG]	[C ‖ FWG]	forced by algorithm
8	[C ‖ FWG]	[FWC ‖ G]		old position }backtrack required!
		[FCG ‖ W]		old position
9	[FCG ‖ W]	see step 7		backtrack to step 7
10	[G ‖ FWC]	see step 6		backtrack to step 6
			[FG ‖ WC]	alternative move from position 6
11	[FG ‖ WC]	[G ‖ FWC]		old position
		[- ‖ FWGC]	[- ‖ FWGC]	forced by algorithm
12	[- ‖ FWGC]			finish

The algorithm does not require the graph to be drawn in advance: instead sections are explored as needed. The solution takes a "wrong turn" at step 6 but the algorithm successfully corrects this by backtracking.

This is a powerful way to solve all sorts of puzzles that involve moving objects around. Just as for mazes, the rules of the algorithm are local, and require only a record of past moves.

Another traditional puzzle leads to a graph of considerable beauty. The *Tower of Hanoi* was marketed in 1883 by the great French recreational mathematician Edouard Lucas (under the pseudonym M.Claus). In 1884, in *La Nature,* M. De Parville described it in romantic terms:

> In the great temple at Benares, beneath the dome which marks the centre of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four discs of pure gold, the largest disc resting on the brass plate, and the others getting smaller and smaller up to the top one. This is the Tower of Bramah. Day and night unceasingly the priests transfer the discs from one diamond needle to another according to the fixed and immutable laws of Bramah, which require that the priest on duty must not move more than one disc at a time and that he must place this disc on a needle so that there is no smaller disc below it. When the sixty-four discs shall have been thus transferred from the needle on which at the creation God placed them to one of the other needles, tower, temple, and Brahmins alike will crumble into dust, and with a thunderclap the world will vanish.

The Tower of Hanoi is similar to the Tower of Brahma but with eight (or sometimes fewer) discs. It is an old friend of recreational mathematicians, and it may seem that little new can be said about it. But, as we shall see, the graphical approach leads to a delightful surprise.

For definiteness, consider *3-disc Hanoi*, that is, the Tower of Hanoi with just three discs (**Fig.2**). To construct the graph, we must first find a way to represent all possible positions, then work out the legal moves between them, and finally draw up the graph. I'll describe what I actually did, because to begin with it's not obvious how to proceed --- and then we'll observe, with twenty-twenty hindsight, that there is a much cleverer method.

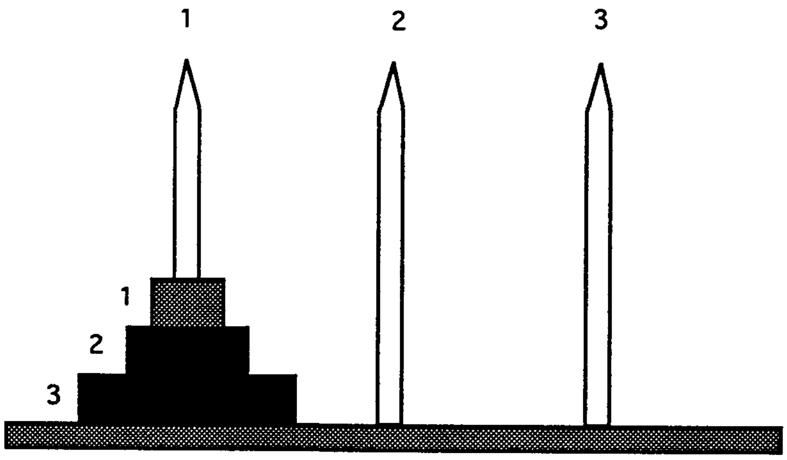

Fig.2. 3-disc Hanoi

How can we represent a position? Number the three discs as 1,2,3, with 1 being the smallest and 3 the largest. Number the needles 1,2,3 from left to right. Suppose that we know on which of the three needles each disc is: for example disc 1 is on needle 2, disc 2 on needle 1, and disc 3 on needle 2. Then we have completely determined the position, because the rules imply that disc 3 must be *underneath* disc 1. We can encode this information in the sequence 212, the three digits in turn representing the needles for discs 1, 2, and 3. Therefore each position in 3-disc Hanoi corresponds to a sequence of three digits, each being 1, 2, or 3.

It follows that there are precisely 3x3x3 = 27 different positions in 3-disc Hanoi. But what are the permitted moves?

The smallest disc on a given needle must be at the top. It thus corresponds to the *first* appearance of the number of that needle in the sequence. If we move that disc, we must move it to the top of the pile on some other needle, that is, we change the number so that it becomes the *first* appearance of some other number.

For example, in the position 212 above, suppose we wish to move disc 1. This is on needle 2, and corresponds to the first occurrence of 2 in the sequence. Suppose we change this first 2 to 1. Then this is (trivially!) the first occurrence of the digit 1; so the move from 212 to 112 is legal. So is 212 to 312 because the first occurrence of 3 is in the first place in the sequence.

We may also move disc 2, because the first occurrence of the symbol 1 is in the second place in the sequence. But we cannot change it to 2, because 2 already appears earlier, in the first place. A change to 3 is, however, legal. So we may change 212 to 232 (but *not* to 222).

Finally disc 3 cannot be moved, because the third digit in the sequence is a 2, and this is *not* the first occurrence of a 2.

To sum up: from position 212 we can make legal moves to 112, 312, and 232, and only these.

We can list all 27 positions and all possible moves by following the above rules: the result is:

The legal moves in 3-disc Hanoi

start here...	finish on any of these...		
111	211	311	
112	212	312	113
113	213	313	112
121	221	321	131
122	222	322	132
123	223	323	133
131	231	331	121
132	232	332	122
133	233	333	123
211	111	311	231
212	112	312	232
213	113	313	233
221	121	321	223
222	122	322	
223	123	323	221
231	131	331	211
232	132	232	212
233	133	333	213
311	111	211	321
312	112	212	322
313	113	213	323
321	121	221	311
322	122	222	312
323	123	223	313
331	131	231	332
332	132	232	331
333	133	233	

All but three positions give exactly three legal moves, but the other three positions give only two legal moves. Why?

The next task requires care and accuracy, but little thought. Draw 27 dots on a piece of paper, label them with the 27 positions, and draw lines to represent the legal moves. A bit of thought, rearranging the vertices and edges to avoid overlaps, leads to (**Fig.3**).

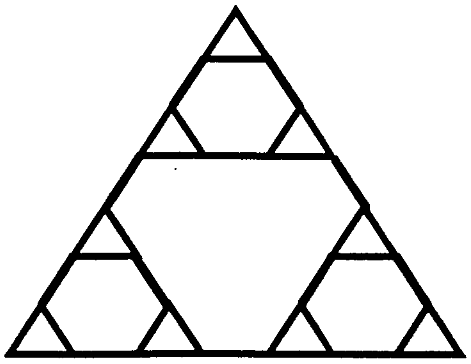

Fig.3. Graph of 3-disc Hanoi.

Something that pretty can't be coincidence!

But before we investigate *why* the graph has such a regular form, let's observe that it answers the original question. To transfer all three discs from needle 1 (position 111) to needle 2 (position 222) we merely run down the left-hand edge, making the moves

$$111 \to 211 \to 231 \to 331 \to 332 \to 132 \to 122 \to 222$$

Indeed, by consulting the graph, it is clear that we can get from *any* position to any other --- and it is also clear what the quickest route is.

On to a deeper question: what is the explanation for the remarkable structure of Fig.3?

The graph consists of three copies of a smaller graph, linked by three single edges to form a triangle. But each smaller graph in turn has a similar triple structure. Why does everything appear in threes, and why are the pieces linked in this manner?

If you work out the graph for 2-disc Hanoi you will find that it looks exactly like the top third of Fig.9 Even the labels on the vertices are the same, except that the final 1 is deleted. In fact it is easy to see this, *without* working out the graph again. You can play 2-disc Hanoi with three discs: just ignore disc 3. Suppose disc 3 stays on needle 1. Then we are playing 3-disc Hanoi, but restricting attention to those 3-digit sequences that end in 1, such as 131 or 221. But these are precisely the sequences in the top third of the figure. Similarly 3-disc Hanoi with disc 3 fixed on needle 2 --- that is, disguised 2-disc Hanoi --- corresponds to the lower left third, and 3-disc Hanoi with disc 3 fixed on needle 3 corresponds to the lower right third.

This explains why we see three copies of the 2-disc Hanoi graph in the 3-disc graph. And a little further thought shows that these three *subgraphs* are joined by just three single edges in the full puzzle. To join up the subgraphs, we must *move* disc 3. When can we do this? Only when one needle is empty, one contains disc 3, and the other contains all the remaining discs! Then we can move disc 3 to the empty needle, creating an empty needle where it came form, and leaving the other discs untouched. There are six such positions, and the possible moves join them in pairs.

The same argument works for any number of discs. The graph for 4-disc Hanoi, for example, consists of three copies of the 3-disc graph, linked at the corners like a triangle. Each subgraph describes 4-disc Hanoi with disc 4 fixed on one of the three needles; but such a game is just 3-disc Hanoi in disguise. And so on. We say that the Tower of Hanoi puzzle has a *recursive* structure; the solution to (n+1)-disc Hanoi is determined by that for n-disc Hanoi according to a fixed rule. The recursive structure explains why the graph for (n+1)-disc Hanoi can be built from that for n-disc Hanoi. The triangular symmetry arises because the rules treat needles 1, 2, and 3 in exactly the same way. You can deduce the graph for 64-disc Bramah, or any other number of discs, by repeatedly applying this rule to the graph for 0-disc Hanoi, which is a single dot!

Two final observations.

1 As the number of discs becomes larger and larger, the graph becomes more and more intricate, looking more and more like the *Sierpinski gasket* (**Fig.4** overleaf). This shape is a fractal, having detailed structure on all scales. This is a striking and surprising result, because the puzzle was invented almost a century before fractals were.

2 Pascal's famous triangle of binomial coefficients

```
        1
      1   1
    1   2   1
  1   3   3   1
1   4   6   4   1
```

is defined by the property that each number is the sum of the two above it to left and right. If you colour odd numbers black and even numbers white, you get another Sierpinski gasket shape.

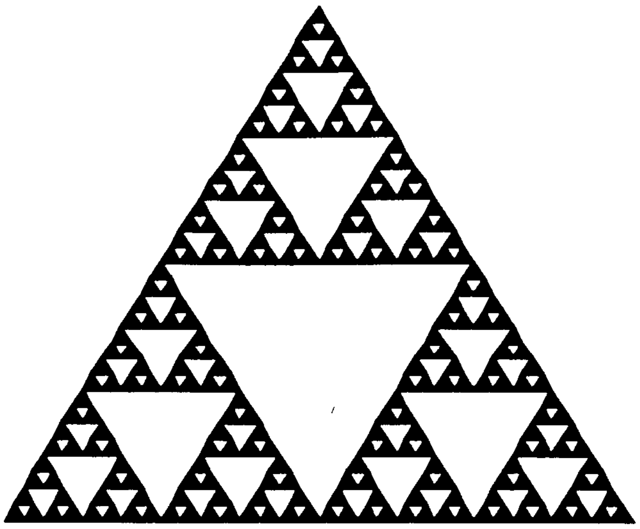
Fig.4. The Sierpinski Gasket.

FURTHER READING

W.W.Rouse Ball, *Mathematical Recreations and Essays*, MacMillan, London 1959.
Ronald Gould, *Graph Theory*, Benjamin-Cummings, Menlo Park 1988.
Ronald L. Graham, Donald E. Knuth, and Oren Patashnik, *Concrete Mathematics*, Addison-Wesley, New York 1989.
J.Hopcroft and R.Tarjan, Algorithm 447: Efficient algorithms for graph manipulation, *Communications of the Association for Computing Machinery* 16 (1973) 372-378.
Edward Kasner and James Newman, *Mathematics and the Imagination*, Bell, London 1961.
Eduard Lucas, *Recréations Mathématiques*, Paris 1892.
E.F.Moore, The shortest path through a maze, *Proceedings of the International Symposium on Switching Theory 1957* Part II, Harvard University press, Harvard 1959, 285-292.
A.F.Ozanam, *Récréations Mathématiques et Physiques*, Paris 1694.
Ian Stewart and John Jaworski (editors), *Seven Years of Manifold 1968-1980*, Shiva Publishing, Nantwich 1981.
Ian Stewart, *Game, Set, and Math*, Penguin, Harmondsworth 1993.
Ian Stewart, *Another Fine Math You've Got Me Into...* Freeman, San Francisco 1994.
R.Tarjan, Depth first search and linear graph algorithms, *Society for Industrial and Applied Mathematics Journal of Computation* 1 (1972) 146-160.

Gresham Geometry Lecture
9 October 1996

Games, Graphs, and Gaskets

Mazes are a common form of puzzle. The aim of this lecture is to show that the same principles that can be used to solve mazes can also be used to solve 'logical mazes' --- mathematical problems of many kinds.

One way to solve a maze is to look at a map and block off all the dead ends. But what if you don't have a map? Well, you *can* try the left-hand-on-wall trick: when you first enter the labyrinth --- or any maze, come to that --- you put your left hand on the wall, and keep it there. That guarantees that eventually you'll find your way back to the start; but not necessarily that you will find your way to the desired goal in the maze. (You won't get lost, at least.) What happens is that you traverse one complete connected system of walls, and if that system isn't connected to the goal, it gets you nowhere useful.

There is a more abstract formulation of the whole problem. The most important thing about a maze is how the junctions connect up. The lengths of the passages don't affect the path you take to get out, just how long it takes. So threading a maze is a topological problem. We can represent the topological essentials by a *graph*: its nodes correspond to the junctions in the maze, its edges correspond to the tunnels. The problem of getting out of a maze --- or of finding a particular place within it --- thus become that of *traversing a graph* from one node to another.

There is a key theorem that governs the possibility of doing this. *Two nodes can be joined by a continuous path if and only if they lie in the same connected component of the graph.* A *connected component* is the set of all nodes that can be reached from a given one by a continuous path. So what the therorem says is that two nodes can be joined by a continuous path if and only if there exists a continuous path that joins them. This may sound trivial, and it is, but it points to an important concept: connectivity. However, it's useless for solving the problem.

A more constructive approach is to devise a maze-threading *algorithm*. The word comes from the arab mathematician Muhammad ibn Musa abu Abdallah al-Khorezmi al-Madjusi al-Qutrubilli. 'Al-Khorezmi' became 'al-Gorizmi', then 'algorism', and finally 'algorithm'. It's used to describe a specific procedure, a computer program. The first general maze-threading algorithm was invented around 1892 by M. Trémaux. It was rediscovered nearly a century later by J.Hopcroft and R.Tarjan in the context of graph theory, which is just the same as mazes, really. They named it *Depth First Search* or the DFS algorithm: it goes like this.

Depth First Search

This visits all nodes in the same connected component as the starting node: in particular it can if desired be terminated when it hits a particular 'finishing' node.

- Begin at any chosen node.
- Visit any adjacent node that has not yet been visited.
- Repeat this as far as possible.
- If all adjacent nodes have been visted already, backtrack through the sequence of nodes that have been visted until you find one that is adjacent to an unvisited node: then visit that one.
- Delete any edge that has been backtracked.
- Repeat until you return to the starting node and there are no unvisited nodes adjacent to it.
- Then you have visited *all* nodes in the connected component of the graph that contains the starting node.

Depth First Search is especially appropriate for threading mazes, because it is possible to program the algorithm so that it can be carried out without having a map of the maze. It involves only *local* rules at nodes, plus a record of nodes and edges already used. You can *explore* the graph and traverse it as you go. The name is fairly appropriate: the idea is to give top priority to pushing deeper into the maze. The algorithm is quite efficient: the number of steps is at most twice the number of edges in the graph.

Wolf, Goat, Cabbage

Many puzzles are really maze-threading problems in disguise: the objective is to thread a *logical* maze. For example, recall the famous old problem of a farmer who has to cross a river. He has with him a wolf, a goat, and a cabbage. The boat can hold only the farmer plus one item of 'produce'; but he can't leave the wolf alone with the goat, because the goat will get eaten, and he can't leave the goat alone with the cabbage, because the cabbage will get eaten. What does he do? This puzzle is usually attributed to the mediaeval mathematician Alcuin (735-804). It is certainly quite ancient, and appears in Ozanam's *Récréations Mathématiques et Physiques* of 1694.

The important thing is which side of the river each of the three items is on. We can represent the position of a single item by the digits 0 and 1, using 0 to represent *this* side of the river and 1 to represent the far side. Thus the configuration of all three items is represented by a triple (w,g,c) in three-dimensional wolf-goat-cabbage space. For example (w,g,c) = (1,0,1) represents w = 1, g = 0, c = 1; that is, the wolf on the far side, the goat on this side, and the cabbage on the far side.

How many configurations are there? Well, each coordinate w, g, or c can take one of the two values 0 or 1. Thus there are 2x2x2 = 8 possibilities. What's more, they have a beautiful geometric structure: they are the eight vertices of a unit cube in wolf-goat-cabbage space (**Fig.1**).

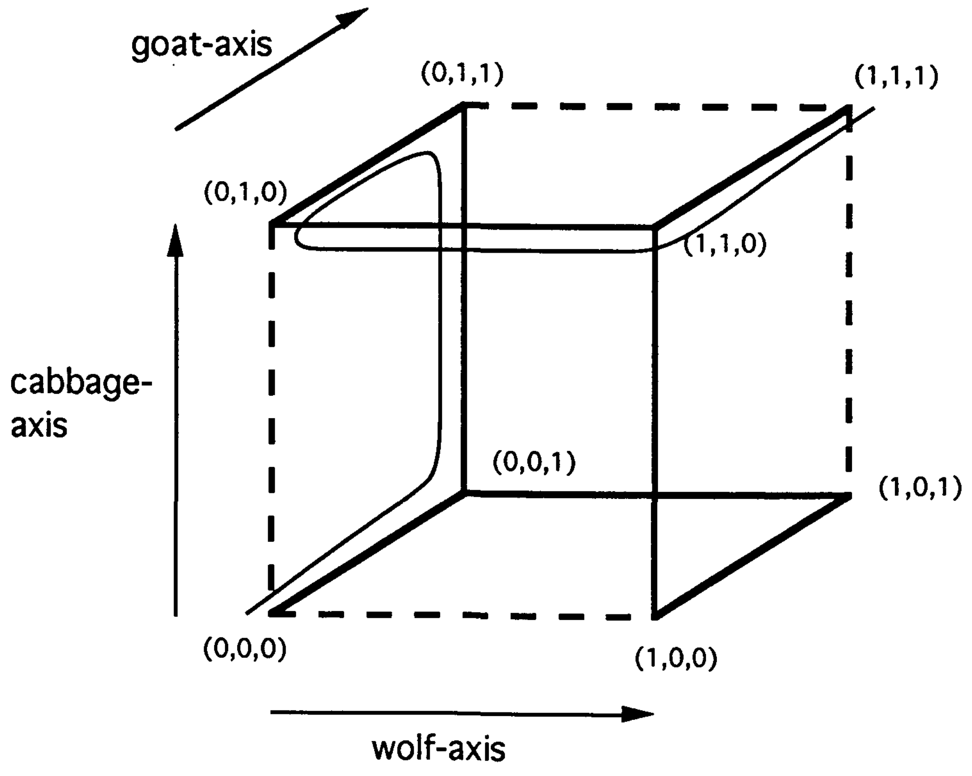

Fig.1 One solution. Find the other.

We may move only a single item at a time; that is, we may traverse only the edges of the cube. But some edges are forbidden. For example, the edge from (0,0,0) to (1,0,0) corresponds to taking the wolf across the river on its own. But this leaves goat and cabbage unchaperoned, so we would shortly be greeted by a fat goat and no cabbage. In fact these gastronomic constraints rule out exactly *four* edges, which are drawn in grey. The rest, representing permissible moves,will be coloured black.

The problem thus geometrized becomes: can we start at (0,0,0) --- all items on *this* side --- and get to (1,1,1) --- all items on the other side --- passing only along green edges of the cube? And of course the answer is 'yes'. Indeed, from a topological viewpoint, we can lay thé edges out flat, and the solution stares us in the face. There are two solutions, in fact, and *only* two ifwe avoid unnecessary repetitions. They differ only by a wolf/cabbage symmetry operation.

How to Get Across With Your Produce Intact

Solution 1

(0,0,0) Start
(0,1,0) Take goat over
(0,1,1) (Return and) take cabbage over
(0,0,1) Bring back goat
(1,0,1) Take wolf over
(1,1,1) (Return and) take goat over.

Solution 2

(0,0,0) Start
(0,1,0) Take goat over
(1,1,0) (Return and) take wolf over
(1,0,0) Bring back goat
(1,0,1) Take cabbage over
(1,1,1) (Return and) take goat over.

Notice what we did to represent the puzzle geometrically. Each possible position in the puzzle can be thought of as a node in a graph, and each move from one position to another as an edge joining the corresponding nodes. You can also use depth first search instead of visual inspection. Here's the general procedure when you do not have a 'map' of the graph, but can only construct it edge by edge as you go.

Using Depth First Search to Solve Puzzles.

Many puzzles involve the movement of people, animals, or objects from a given position to a selected finishing position, subject to various rules. If the number of possible positions is finite, then Depth First Search can be employed. It takes the following form:

- Whenever you first come to a new position (including the initial position at the start of the puzzle) list all possible positions that can be reached from it.
- From the starting position, make any move that leads to a "new" position (one that has not yet occurred), at random.
- Repeat this as far as possible.
- If all possible moves lead to "old" positions, backtrack through the sequence of moves that have been made until you find a position from which it is possible to move to a "new" position. Then make that move.
- Henceforth ignore any move that has been backtracked.
- Repeat until you either reach the desired position, or return to the starting position with no moves available, in which case the puzzle is impossible.

We'll apply this to the wolf-goat-cabbage puzzle. We represent the participants by F = farmer, W = wolf, G = goat, C = cabbage, and positions by symbols such as [FG || WC] where the symbol || ("Styx") represents the river. Here farmer and goat are on the left bank, wolf and cabbage on the right.

step	current position	possible moves	move made	comments
1	[FWGC ǁ -]	[WC ǁ FG]	[WC ǁ FG]	starting position
2	[WC ǁ FG]	[FWGC ǁ -]		old position
		[FWC ǁ G]	[FWC ǁ G]	forced by algorithm
3	[FWC ǁ G]	[WC ǁ FG]		old position
		[W ǁ FGC]	[W ǁ FGC]	random choice of available moves
		[C ǁ FWG]		other possible choice
4	[W ǁ FGC]	[FWC ǁ G]		old position
		[FWG ǁ C]	[FWG ǁ C]	forced by algorithm
5	[FWG ǁ C]	[W ǁ FGC]		old position
		[G ǁ FWC]	[G ǁ FWC]	forced by algorithm
6	[G ǁ FWC]	[FWG ǁ C]		old position
		[FCG ǁ W]	[FCG ǁ W]	random choice (bad move but it
		[FG ǁ WC]		shows how the algorithm copes)
7	[FCG ǁ W]	[G ǁ FWC]		old position
		[C ǁ FWG]	[C ǁ FWG]	forced by algorithm
8	[C ǁ FWG]	[FWC ǁ G]		old position } backtrack required!
		[FCG ǁ W]		old position
9	[FCG ǁ W]	see step 7		backtrack to step 7
10	[G ǁ FWC]	see step 6		backtrack to step 6
			[FG ǁ WC]	alternative move from position 6
11	[FG ǁ WC]	[G ǁ FWC]		old position
		[- ǁ FWGC]	[- ǁ FWGC]	forced by algorithm
12	[- ǁ FWGC]			finish

The algorithm does not require the graph to be drawn in advance: instead sections are explored as needed. The solution takes a "wrong turn" at step 6 but the algorithm successfully corrects this by backtracking.

This is a powerful way to solve all sorts of puzzles that involve moving objects around. Just as for mazes, the rules of the algorithm are local, and require only a record of past moves.

Another traditional puzzle leads to a graph of considerable beauty. The *Tower of Hanoi* was marketed in 1883 by the great French recreational mathematician Edouard Lucas (under the pseudonym M.Claus). In 1884, in *La Nature*, M. De Parville described it in romantic terms:

> In the great temple at Benares, beneath the dome which marks the centre of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four discs of pure gold, the largest disc resting on the brass plate, and the others getting smaller and smaller up to the top one. This is the Tower of Bramah. Day and night unceasingly the priests transfer the discs from one diamond needle to another according to the fixed and immutable laws of Bramah, which require that the priest on duty must not move more than one disc at a time and that he must place this disc on a needle so that there is no smaller disc below it. When the sixty-four discs shall have been thus transferred from the needle on which at the creation God placed them to one of the other needles, tower, temple, and Brahmins alike will crumble into dust, and with a thunderclap the world will vanish.

The Tower of Hanoi is similar to the Tower of Brahma but with eight (or sometimes fewer) discs. It is an old friend of recreational mathematicians, and it may seem that little new can be said about it. But, as we shall see, the graphical approach leads to a delightful surprise.

For definiteness, consider *3-disc Hanoi*, that is, the Tower of Hanoi with just three discs (**Fig.2**). To construct the graph, we must first find a way to represent all possible positions, then work out the legal moves between them, and finally draw up the graph. I'll describe what I actually did, because to begin with it's not obvious how to proceed --- and then we'll observe, with twenty-twenty hindsight, that there is a much cleverer method.

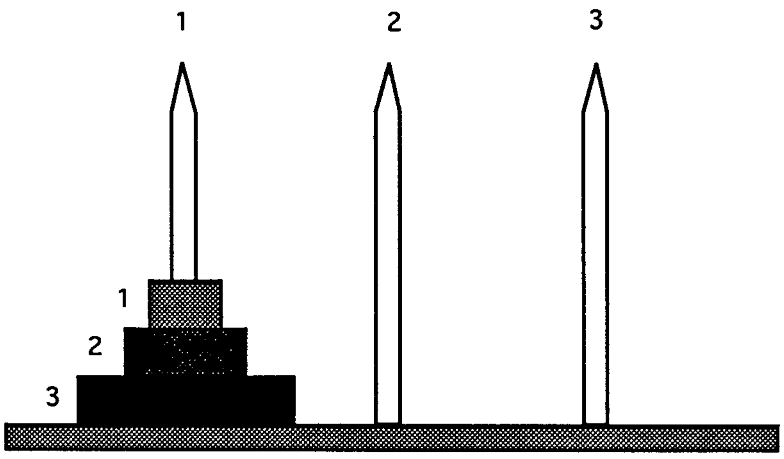

Fig.2. 3-disc Hanoi

How can we represent a position? Number the three discs as 1,2,3, with 1 being the smallest and 3 the largest. Number the needles 1,2,3 from left to right. Suppose that we know on which of the three needles each disc is: for example disc 1 is on needle 2, disc 2 on needle 1, and disc 3 on needle 2. Then we have completely determined the position, because the rules imply that disc 3 must be *underneath* disc 1. We can encode this information in the sequence 212, the three digits in turn representing the needles for discs 1, 2, and 3. Therefore each position in 3-disc Hanoi corresponds to a sequence of three digits, each being 1, 2, or 3.

It follows that there are precisely 3x3x3 = 27 different positions in 3-disc Hanoi. But what are the permitted moves?

The smallest disc on a given needle must be at the top. It thus corresponds to the *first* appearance of the number of that needle in the sequence. If we move that disc, we must move it to the top of the pile on some other needle, that is, we change the number so that it becomes the *first* appearance of some other number.

For example, in the position 212 above, suppose we wish to move disc 1. This is on needle 2, and corresponds to the first occurrence of 2 in the sequence. Suppose we change this first 2 to 1. Then this is (trivially!) the first occurrence of the digit 1; so the move from 212 to 112 is legal. So is 212 to 312 because the first occurrence of 3 is in the first place in the sequence.

We may also move disc 2, because the first occurrence of the symbol 1 is in the second place in the sequence. But we cannot change it to 2, because 2 already appears earlier, in the first place. A change to 3 is, however, legal. So we may change 212 to 232 (but *not* to 222).

Finally disc 3 cannot be moved, because the third digit in the sequence is a 2, and this is *not* the first occurrence of a 2.

To sum up: from position 212 we can make legal moves to 112, 312, and 232, and only these.

We can list all 27 positions and all possible moves by following the above rules: the result is:

The legal moves in 3-disc Hanoi

start here...	finish on any of these...		
111	211	311	
112	212	312	113
113	213	313	112
121	221	321	131
122	222	322	132
123	223	323	133
131	231	331	121
132	232	332	122
133	233	333	123
211	111	311	231
212	112	312	232
213	113	313	233
221	121	321	223
222	122	322	
223	123	323	221
231	131	331	211
232	132	232	212
233	133	333	213
311	111	211	321
312	112	212	322
313	113	213	323
321	121	221	311
322	122	222	312
323	123	223	313
331	131	231	332
332	132	232	331
333	133	233	

All but three positions give exactly three legal moves, but the other three positions give only two legal moves. Why?

The next task requires care and accuracy, but little thought. Draw 27 dots on a piece of paper, label them with the 27 positions, and draw lines to represent the legal moves. A bit of thought, rearranging the vertices and edges to avoid overlaps, leads to (**Fig.3**).

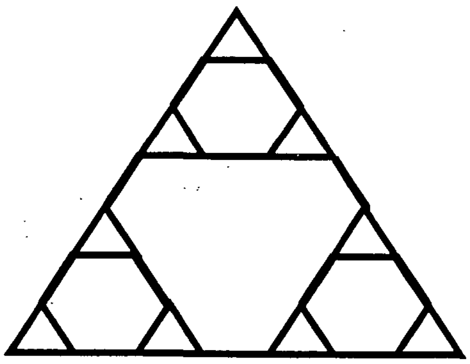

Fig.3. Graph of 3-disc Hanoi.

Something that pretty can't be coincidence!

But before we investigate *why* the graph has such a regular form, let's observe that it answers the original question. To transfer all three discs from needle 1 (position 111) to needle 2 (position 222) we merely run down the left-hand edge, making the moves

$$111 \to 211 \to 231 \to 331 \to 332 \to 132 \to 122 \to 222$$

Indeed, by consulting the graph, it is clear that we can get from *any* position to any other --- and it is also clear what the quickest route is.

On to a deeper question: what is the explanation for the remarkable structure of Fig.3?

The graph consists of three copies of a smaller graph, linked by three single edges to form a triangle. But each smaller graph in turn has a similar triple structure. Why does everything appear in threes, and why are the pieces linked in this manner?

If you work out the graph for 2-disc Hanoi you will find that it looks exactly like the top third of Fig.9 Even the labels on the vertices are the same, except that the final 1 is deleted. In fact it is easy to see this, *without* working out the graph again. You can play 2-disc Hanoi with three discs: just ignore disc 3. Suppose disc 3 stays on needle 1. Then we are playing 3-disc Hanoi, but restricting attention to those 3-digit sequences that end in 1, such as 131 or 221. But these are precisely the sequences in the top third of the figure. Similarly 3-disc Hanoi with disc 3 fixed on needle 2 --- that is, disguised 2-disc Hanoi --- corresponds to the lower left third, and 3-disc Hanoi with disc 3 fixed on needle 3 corresponds to the lower right third.

This explains why we see three copies of the 2-disc Hanoi graph in the 3-disc graph. And a little further thought shows that these three *subgraphs* are joined by just three single edges in the full puzzle. To join up the subgraphs, we must *move* disc 3. When can we do this? Only when one needle is empty, one contains disc 3, and the other contains all the remaining discs! Then we can move disc 3 to the empty needle, creating an empty needle where it came form, and leaving the other discs untouched. There are six such positions, and the possible moves join them in pairs.

The same argument works for any number of discs. The graph for 4-disc Hanoi, for example, consists of three copies of the 3-disc graph, linked at the corners like a triangle. Each subgraph describes 4-disc Hanoi with disc 4 fixed on one of the three needles; but such a game is just 3-disc Hanoi in disguise. And so on. We say that the Tower of Hanoi puzzle has a *recursive* structure; the solution to (n+1)-disc Hanoi is determined by that for n-disc Hanoi according to a fixed rule. The recursive structure explains why the graph for (n+1)-disc Hanoi can be built from that for n-disc Hanoi. The triangular symmetry arises because the rules treat needles 1, 2, and 3 in exactly the same way. You can deduce the graph for 64-disc Bramah, or any other number of discs, by repeatedly applying this rule to the graph for 0-disc Hanoi, which is a single dot!

Two final observations.

1 As the number of discs becomes larger and larger, the graph becomes more and more intricate, looking more and more like the *Sierpinski gasket* (**Fig.4** overleaf). This shape is a fractal, having detailed structure on all scales. This is a striking and surprising result, because the puzzle was invented almost a century before fractals were.

2 Pascal's famous triangle of binomial coefficients

```
        1
      1   1
    1   2   1
  1   3   3   1
1   4   6   4   1
```

is defined by the property that each number is the sum of the two above it to left and right. If you colour odd numbers black and even numbers white, you get another Sierpinski gasket shape.

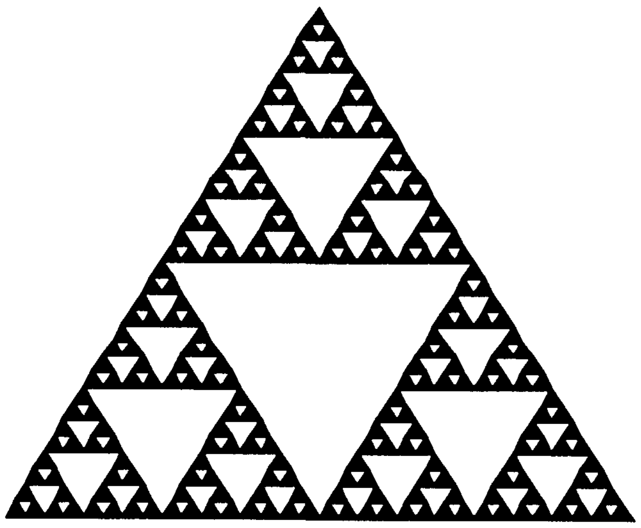

Fig.4. The Sierpinski Gasket.

FURTHER READING

W.W.Rouse Ball, *Mathematical Recreations and Essays*, MacMillan, London 1959.

Ronald Gould, *Graph Theory*, Benjamin-Cummings, Menlo Park 1988.

Ronald L. Graham, Donald E. Knuth, and Oren Patashnik, *Concrete Mathematics*, Addison-Wesley, New York 1989.

J.Hopcroft and R.Tarjan, Algorithm 447: Efficient algorithms for graph manipulation, *Communications of the Association for Computing Machinery* 1 6 (1973) 372-378.

Edward Kasner and James Newman, *Mathematics and the Imagination*, Bell, London 1961.

Eduard Lucas, *Recréations Mathématiques*, Paris 1892.

E.F.Moore, The shortest path through a maze, *Proceedings of the International Symposium on Switching Theory 1957* Part II, Harvard University press, Harvard 1959, 285-292.

A.F.Ozanam, *Récréations Mathématiques et Physiques*, Paris 1694.

Ian Stewart and John Jaworski (editors), *Seven Years of Manifold 1968-1980*, Shiva Publishing, Nantwich 1981.

Ian Stewart, *Game, Set, and Math*, Penguin, Harmondsworth 1993.

Ian Stewart, *Another Fine Math You've Got Me Into...* Freeman, San Francisco 1994.

R.Tarjan, Depth first search and linear graph algorithms, *Society for Industrial and Applied Mathematics Journal of Computation* 1 (1972) 146-160.