TOURING PROBLEMS – A MATHEMATICAL APPROACH

VIPUL NAIK

Abstract. The article surveys "touring", a problem of "fun" math, and employs serious techniques to attack it. The text is suitable for high school students interested in Olympiad mathematics, as well as undergraduate students studying combinatorics and discrete mathematics. It can be used to introduce the concept of a graph. A sequel to this article, on a similar analysis for "tiling" problems, shall also appear soon.

1. Prebeginnings

1.1. **The knight's tour.** Chess pieces, with their variously defined moves, are a rich source of puzzles in **recreational mathematics**. **Chess puzzles** include the following types :

- **Nonattacking pieces puzzle** : Given a piece with some moves defined, place as many copies of this piece as possible so that no two can kill each other. The prototypical example of this is the **eight queens problem** – placing eight queens on the chessboard so that no two can kill each other.
- **Board domination puzzle** : Given a piece with some moves defined, place as few copies of the piece as possible so that every square is either covered by some copy or is under direct attack by a copy of the piece.
- **Touring puzzle** : Given a piece with some moves defined, construct a tour of the entire chessboard with that piece. The only nontrivial case of this for a **standard chess piece** is the **knight's tour**.

Each of these problems revolves around a piece, with certain moves defined for that piece. Instead of standard chess pieces, we may sometimes be interested in defining chess pieces with somewhat different moves. Such pieces are termed **fairy chess pieces** .

In this article, we focus on the problem of the knight's tour, and its formulation in a manner that helps us consider general variations – such as replacing the knight by some fairy chess piece, or changing the shape and size of the board. Interestingly, we will find that the formalism that we develop on the way is useful even for understanding the other classes of chess puzzles.

1.2. **Tour finding, maze hunting and sudoku solving.** When a rat wants to find its way through a maze, it must, at every stage, make some choices as to which path to take (whenever there is more than one choice). If the rat makes the wrong choice, it will soon reach a **dead end**, and will go *back* to the last choice it made, and try the other one. When it finally reaches the starting point with no more paths to try out, it throws up its paws and concludes that there is "no way".

We do something similar while solving **sudoku** puzzles. By and large, our attempt is to use **scanning** techniques to locate positions that can be filled up. That is, we scan the rows and columns for entries we are sure of, using the fact that every row, every column and every 3×3 cell has all the numbers from 1 to 9. For eliminating positions for numbers and numbers for positions, we use techniques of **contingency spotting** – that is, contingencies that force numbers to be within certain squares, and **cross hatching** – that combine multiple contingencies to draw conclusions. Repeated application of these helps us fill in positions we are sure of.

However, there may be situations where we are not able to be sure of anything further to fill in. In such cases, we do a **what if** – we try out filling a number in some place and see if it works out.

If this what if does not lead to a solution, then we will soon come up with a contradiction and shall be forced to remove all the entries filled in after the last what if. We can then try one of the other possibilities at that stage.

In much the same way, when trying to construct a **knight's tour** of the chessboard, we are, at every stage, confronted with some options and pick one option over the others. However, the option that we pick might lead to a dead end, in which case we need to **backtrack** and choose some other option.

In our attempts to construct knight's tours of the chessboard, we shall study more formally these kinds of strategies, under the title **depth first search**. As we can already see from the above three examples, we need to improve on the depth first search by minimizing **backtracking** because that indicates time wasted without any reward. We will come up with techniques for doing this in the knight's tour situation that help us construct the tour.

1.3. **Travelling salesmen.** The **travelling salesman problem** is an important problem both in theoretical and in practical computer science.

Problem 1 (Travelling Salesman Problem)**.** *A salesman has to visit n cities. There is a network of roads between the cities and the shortest distance between any pair of cities is known to the travelling salesman. How can he visit each of the cities so as to minimize the total distance he travels? (The distance between the cities is much larger compared to distances within each city)*

In a map, the cities may be represented as **vertices** and the roads may be represented as **edges** joining them, with each edge having a **length**. The length of the road may not be exactly equal to the **crow's distance** between the cities. The travelling salesman's task is to visit each city while minimizing the total road length he travels.

This problem bears many similarities to the knight's tour problem, in terms of both formulation and solution strategies. The travelling salesman problem is an example of an **optimization problem** and is outside the scope of this article. Nonetheless, the terminology developed here (for the purpose of analyzing the knight's tour) is used in analyzing problems like this one.

2. The touring problem – general formulation

2.1. **The motivating problem.** The famous **knight's tour problem** asks whether a knight can tour an entire 8×8 chessboard visiting each square exactly once. Here, a knight can move in any of 8 ways, provided that the final destination is within the board. In each of these ways, one coordinate of the knight's position changes by 2 units (positively or negatively), and the other coordinate changes by 1 unit (positively or negatively). If the two directions are labelled *up/down* and *left/right* the eight moves are :

- Up two steps, right one step

- Up two steps, left one step
- Right two steps, up one step
- Right two steps, down one step
- Down two steps, left one step
- Down two steps, right one step
- Left two steps, up one step
- Left two steps, down one step.

From the above listing of moves, it is clear that the knight's move is *symmetric*, in the sense that if a knight can move from a square A to a square B in one move, it can also move from B to A in one step.

There are many variations of this problem :

- Whether a tour is possible with a given initial position.
- Whether a tour is possible with a given initial and a given final position.
- Whether there is a **closed tour** or **re-entrant tour** , that is, a tour where the last square is a knight's move away from the first. If there is a closed tour beginning at some square, there is a closed tour beginning at any square.

Essentially, the knight needs to cover as many squares of the chessboard such that after each square, the next square it goes to is a knight's move away. We would like a structure that captures this relationship between squares (of being separated only by a knight's move). Such a structure exists in mathematics – a **graph**.[1]

The purpose of this abstraction is to help us develop *more generic tools* that can be applied to variants of the knight's tour problem. For instance, we may change the :

- Shape of the board (we might even choose not to keep it rectangular)
- Moves allowed on the piece (we might even choose them not to be symmetric)
- Conditions imposed on the tour – we might put some additional conditions or relax some constraints.

2.2. Necessary background and problem formulation.

A graph comprises a set of **vertices** or **nodes** and a set of **edges** or **arcs** or **lines**, where each edge joins two nodes. We use the following terms :

- The two nodes that an edge joins are termed its **endpoints**. Given two vertices, if there is an edge joining them, they are termed **adjacent**.
- A vertex and an edge are **incident** provided that the vertex is one of the endpoints of the edge.

Typically graphs are represented by pictures in the plane with the vertices represented as thick dots, and the edges represented as line segments or arcs joining the endpoints. Most graphs cannot be drawn in the plane such that no two edges cross each other, though such a depiction would be visually ideal. [2]

In our case, the vertices of the graphs are the squares of the chessboard, and there is an edge joining two vertices (that is, the two vertices are adjacent) if they are separated by a knight's move. This graph completely *captures* the behaviour of the chessboard in the

[1]The term "graph" is used in a completely different sense when we talk of the graph of a relation or a function.

[2]Graphs which can be drawn without edges crossing each other are termed planar graphs. A graph is planar if and only if it does not contain a K_5 or a $K_{3,3}$

context of the knight's move. Thus, any problem pertaining to the knight's move should be expressible as a problem in terms of properties of this graph. For convenience, we shall call this graph the **knight's move graph**.[3]

We can define the graph in this manner because the knight's move is symmetric – if the knight can move from a vertex (square on the chessboard) A to a vertex (square on the chessboard) B in one move, it can also move from B to A in one move. Cases where the moves are not symmetric require us to use the concept of directed graphs, where each edge has a definite direction. We will not be discussing them in this article.

A **tour**, or more technically, a **Hamiltonian path** of a general graph is a sequence of its vertices such that every vertex occurs *exactly* once, and vertices occurring adjacent to each other in the sequence are adjacent in the graph. The sequence describes the order in which the vertices are covered by the Hamiltonian path. The knight's tour that we had encountered earlier on is simply a Hamiltonian path in the knight's move graph.

The tour is termed a **closed tour** or **re-entrant tour** iff[4] the first and last members of the sequence are adjacent in the graph. Thus, every cyclic permutation of a closed tour is a closed tour. Technically a closed tour is termed a **Hamiltonian cycle** or **Hamiltonian circuit**. The closed knight's tour that we had encountered earlier on is simply a Hamiltonian cycle in the knight's move graph.

This suggests that we concentrate on the following general problem :

Core Problem 1. *Given a graph, determine all the Hamiltonian paths, and Hamiltonian cycles in the graph.*

As we shall see, this core problem formulation is very ambitious. Later in this article (in section 2.4), we shall water it down.

2.3. **Construction of the knight's move graph.** Here, we try to get a hands on feel for how graphs look like.

The squares on the chessboard can be given **coordinates** in a natural way. For an $m \times n$ board, label the columns (vertical lines) $1, 2 \dots n$ from left to right. That is, the left most column is labelled 1, the next is labelled 2 and so on.

Similarly label the rows $1, 2 \dots m$ from bottom upwards. That is, the lowest row is labelled 1, the row above that is labelled 2, and so on.

Every square is identified with an ordered pair, the first entry giving the column number, and the second entry giving the row number. Thus, the left most bottom square is labelled $(1, 1)$ and the right most bottom square is labelled $(n, 1)$. The right most top square is labelled (n, m) and the left most top square is labelled $(1, m)$.

Let us now construct the knight's move graph for the 3×3 board. The vertices are :

$$\begin{array}{ccc} (1,3) & (2,3) & (3,3) \\ (1,2) & (2,2) & (3,2) \\ (1,1) & (2,1) & (3,1) \end{array}$$

The edges are as follows :

[3]In some places, it is referred to as the knight's tour graph.

[4]iff means if and only if

- Between $(1,1)$ and $(3,2)$.
- Between $(3,2)$ and $(1,3)$.
- Between $(1,3)$ and $(2,1)$.
- Between $(2,1)$ and $(3,3)$.
- Between $(3,3)$ and $(1,2)$.
- Between $(1,2)$ and $(3,1)$.
- Between $(3,1)$ and $(2,3)$.
- Between $(2,3)$ and $(1,1)$.

We observe that :

- The vertex $(2,2)$ has no edges incident on it.
- Every other vertex has precisely two edges incident on it.

In fact, we can *redraw* the graph as a **cyclic graph** by making the vertices $(1,1)$, $(3,2)$, $(1,3)$, $(2,1)$, $(3,3)$, $(1,2)$, $(3,1)$ and $(2,3)$ the vertices of an octagon in that order, and our edges as simply the sides of that octagon. The graph is termed cyclic because the whole graph is a Hamiltonian cycle – by just starting at any vertex and moving along the graph we get a Hamiltonian cycle.

Thus we observe that, although a tour of the entire 3×3 board is not possible, a closed tour of the board obtained by removing the middle square is possible.

We note that redrawing the graph in a cyclic fashion made it conceptually clearer compared to just joining the vertices in the actual chessboard (which would give a star like shape). However, as graphs, they are *the same*. The graph structure is not sensitive to how we draw it on paper. It *only encodes certain information* – namely information as to what are the vertices and what are the endpoints of each edge. All knowledge about the graph should be dependent only on this data.

Concept Testers

(1) A graph is called a **planar graph** if it *can* be drawn in the plane with the vertices as points and the edges as arcs, with no two edges intersecting except at a vertex. Is the knight's move graph for the 3×3 graph planar?
(2) Construct the knight's move graph for the 4×4 chessboard, and show that it is planar (by giving a planar drawing).
(3) Construct the bishop's move graph for the 3×3 chessboard. Show that it is planar. How many triangles does it contain?
(4) A graph is said to be **connected** if given any two vertices, there is a way of going from one to the other, via edges. Which of the above three graphs is connected?

2.4. **The existential paradigm of reasoning.**

Combinatorics as it is done in high school is typically *enumerative* in nature. It is concerned with questions such as – what is the size of a certain set? In many situations, such as the touring problems of this article, questions such as : "how many tours are possible?" are notoriously difficult to solve. Moreover, in most such cases, knowing the *number* of tours is not of much use.

Rather, a more basic question occupies our attention : is there *any* tour? The paradigm of combinatorics that handles questions of existence is termed **existential combinatorics** as opposed to **enumerative combinatorics**.

Sometimes, we are also interested in *finding* the tour rather than showing that it exists. The two problems are *not* the same because there are non-constructive methods of proving existence.

At times, we are interested in finding *all* tours even if we are unable to provide any expression for the number of tours. The paradigm of combinatorics that deals with such issues is termed **constructive combinatorics**.

The core problem formulation that we shall restrict ourselves to for the bulk of this article is :

Core Problem 2. *Given a graph, determine whether or not a Hamiltonian path (Hamiltonian cycle) exists. If a Hamiltonian path (Hamiltonian cycle) exists, exhibit any one such Hamiltonian path (Hamiltonian cycle).*

A graph that possesses a Hamiltonian cycle (also called a Hamiltonian circuit) is termed a **Hamiltonian graph**. Thus, one aspect of Core Problem 2 is to determine whether or not a given graph is Hamiltonian.

In particular, we shall not try to tackle the problem of constructing all Hamiltonian paths (Hamiltonian cycles), or giving the number of Hamiltonian paths (Hamiltonian cycles).

2.5. **Affirmative versus negative.** We shall see repeatedly that the techniques employed to show that a Hamiltonian path (tour) or Hamiltonian cycle (closed tour) *does* exist are widely different from those employed to show that a tour *does not* exist.

Key Point 1. *Proving an affirmative result on existence typically requires either constructive or quasi-enumerative techniques. Proving a negative result requires methods of contradiction. The techniques used for the two kinds of proofs are so different that before starting out, it is necessary to correctly guess which alternative is true.*

For instance, if we guess that a tour does not exist and try to prove it for a long time, but fail, then that seems to be good evidence that a tour probably does exist. On the other hand, we can only show that the tour does exist by switching to the methods for establishing existence.

Moreover, there is often a gap between the techniques for proving negative results and the techniques for proving positive results – there will always be some cases that do not yield to either class of techniques. Ingenuity in constructing new techniques is then the only way out! No wonder existential combinatorics is a tough subject!

3. Methods for showing non-existence

3.1. **Why non-existence first?** We first concentrate on methods used to show that a Hamiltonian path of a graph does *not* exist. For many graphs, the existence of a Hamiltonian path can be eliminated by simple application of these methods. This saves us the futile labour of trying to come up with a tour (viz Hamiltonian path) when we can easily show that there is none.

If none of the non-existence methods work, we can suspect that there *might* be a tour.

Most of the non-existence techniques make use of some kind of colouring or size arguments. Some of these arguments can be understood even without the explicit use of graphs. However, the graph theoretic language provides a natural *direction* and also a firm *foundation* necessary for the somewhat more complicated proofs. So we'll take a quick look at it, after discussing an elementary colouring argument.

3.2. **Some quick examples.** A **bishop** can move along the diagonals, that is, it can make any of these as moves :

- p steps up and p steps right
- p steps up and p steps left
- p steps down and p steps left
- p steps down and p steps right

Thus, analogous to the knight's move graph, we can define the bishop's move graph of the chessboard. In this graph, two vertices are adjacent if and only if they lie on a common diagonal.

We now show that a bishop's tour of the chessboard is not possible.

The chessboard, as it comes to us, already has a colouring with the two colours White and Black. The corner left square is coloured black. Two squares that share an edge are given opposite colours.

As per this colouring, each time a piece moves one step up, or down, or left, or right, the colour of its square changes. This indicates that if a piece takes an *even* number of steps between its initial and final square, it lands on a square with the same colour.

A bishop always takes an equal number of steps along the vertical and horizontal directions, and hence, the colour of its initial and final squares is always the same. Another way of saying this is that all the squares on a diagonal have the same colour. Since each bishop's move joins two vertices with the same colour, it is clear that a bishop can never, through a sequence of moves, land on a square with a different colour.

This indicates that the bishop's move graph is not *connected*, because there is no path from a white square to a black square. Clearly, a graph that is not connected cannot have a Hamiltonian path, and hence we conclude that a bishop's tour of a chessboard is not possible.

The argument we have just given is an example of a **colouring argument**. A **colouring** is, roughly, a partition of the set of vertices into equivalence classes, that bears some relation with the graph structure. To properly understand how to construct colourings and how to exploit them in proofs, we develop a little more graph theoretic notation.

3.3. **A little more terminology.** Before plunging into the main techniques, we must familiarize ourselves with some terms.

A **simple graph** is a graph as described earlier, where given any two vertices, there is at most one edge joining them, and no edge has the same endpoints.

In graphs which are not simple, we may have :

- **Loops** – edges where both the endpoints are the same vertex
- **Parallel edges** – multiple edges with the same pair of endpoints

3.4. **Graph homomorphisms – and a metamathematical issue.** The contents of this section are a little more abstract and it can be skipped at a first reading without much loss of value. Its primary purpose is to help put colouring arguments in a proper graph theoretic perspective.

For existence, there is, by and large, only the direct way – actually construct. For non-existence, on the other hand, there are plenty of diverse tools. Why so?

The basic reason for this can be traced to a very general construction – that of the *homomorphism* [5]. A homomorphism may be thought of as a map between two objects of a certain type (or more technically *category*) that preserves relationships of some sort, but not all. We now come up with a notion of homomorphism for graphs that is suitable for our purposes.

Let G and H be two finite graphs, with loops allowed, but no directions and no parallel edges. Then we define a **graph homomorphism** from G to H as a map from the vertex set of G to the vertex set of H such that if two vertices are adjacent in G then their images are adjacent in H. So, the property of adjacency is not lost on performing a graph homomorphism. However, the fact that the images of two vertices are adjacent cannot be used to conclude that the two vertices themselves are adjacent.

Though the map preserves *some* structure, it is accompanied by loss of information. Here, the images of vertices v and w not being adjacent in H means that v and w are not adjacent in G. However, if the images are adjacent, then we cannot conclude anything about v and w.

In the above definition of a graph homomorphism, there may be vertices of H that never arise as images of vertices of G. It is also possible that there are edges of H that never arise as images of edges in G. These vertices and edges can be removed while still keeping the map a homomorphism. The modified graph H' whose vertices are *precisely* the images of vertices of G and whose edges are *only* the edges that join vertices which are images of adjacent vertices of G, is termed the **image graph**.

Many proofs of the impossibility of tours are based on the following heuristic :

Heuristic 1 (The Homomorphism Approach)**.** *We construct a homomorphism from the given graph to some other graph, and look at the image graph under this homomorphism. We now* use *the image graph to demonstrate that if we take the* image *of a tour on this graph, we get a contradiction.*

3.5. **Some standard colouring arguments.** In the problem of the bishop's tour discussed earlier, our strategy had been to colour the squares white and black, in the fashion usually found on the chessboard – squares adjacent on the chessboard are given opposite colours. This *colouring* strategy is an elementary example of a graph homomorphism, as is explained below.

We define a map from the vertices (in this case, squares of the chessboard) to a two element set : { White, Black }. All the vertices coloured white map to White and all the black vertices map to Black. In the case of the chessboard, we chose the standard colouring of the chessboard.

We next make this map a graph homomorphism. For every pair of vertices adjacent in the original graph, we need to make sure that the images are adjacent in the two element graph { White, Black }.

In the case of a bishop, we had observed that a bishop's move either joins two white vertices or two black vertices. Thus, in the image graph on the two vertices White and Black, the only edges are the loops – the loop joining White to itself, and the loop joining Black to itself. The crucial fact that we used for this was that every move of a bishop is accomplished by an *even* number of steps.

[5]The idea of a homomorphism is a general notion of category theory and is most commonly encountered in the study of algebraic and geometric structures, and the ideas here are valid in all those contexts

In the case of a knight, all the moves involve two steps in one direction and one step in a perpendicular direction. As a result, the total number of steps involved in a knight's move is *odd*. Thus, the initial and final position of the knight will always have *different* colours.

So, in the image graph, there will only be an edge joining the vertices WHITE and BLACK. There cannot be any loop joining WHITE with WHITE or BLACK with BLACK because there are no knight's move taking white squares to white squares or black squares to black squares.

Suppose now that there were a knight's tour of the chessboard. The tour, as described earlier, is a sequence of vertices of the graph that covers every vertex exactly once, and where adjacent vertices in the sequence are adjacent in the graph. Then, the *image* of this sequence as per the graph homomorphism would alternate between the vertices WHITE and BLACK.

This has the following implications for a knight's tour of a board:

- There cannot be a closed tour (viz Hamiltonian cycle) unless the number of white vertices equals the number of black vertices.

  In the 8×8 chessboard, this condition is satisfied, so we cannot rule out the possibility of a closed tour. On the other hand, any chessboard with an odd number of vertices clearly cannot have a closed tour by the knight.
- If the difference between the number of white and the number of black vertices (for a board of any shape) exceeds 1 then there is no tour.
- if the number of white vertices is 1 more than the number of black vertices, then any tour must begin at a white vertex and end at a white vertex. (Analogously if the number of black vertices exceeds the number of white vertices).

A graph with a homomorphism to a two vertex set so that the image of every edge is an edge connecting the two distinct vertices, is termed a **bipartite graph**. In other words, a graph whose vertex set can be partitioned into two subsets such that there is no edge joining two vertices in the same subset, is termed bipartite. All the above observations hold for all bipartite graphs, replacing white and black with whatever names we give to the equivalence classes.

The **chromatic number** of a graph is the minimum integer n such that there exists a graph homomorphism to a graph on n vertices, so that the image of every edge connects two distinct vertices. That is, the vertices can be partitioned into n parts such that no two vertices in the same part are adjacent. Such a homomorphism is termed a **colouring** in ordinary parlance. The only graphs with chromatic number 1 are the **empty graphs** or graphs with empty edge set. A bipartite nonempty graph has chromatic number 2.

For finite simple graphs, the chromatic number is bounded above by the number of vertices in the graph. If n is the chromatic number the graph is termed an n **partite graph**.

The following general ideas hold for graph homomorphisms :-

- Given any graph homomorphism, if the image is not connected the original graph is also not connected. Thus, a tour is not possible.

  This rules out, for instance, a bishop's tour of the chessboard – a map to the two element set { WHITE, BLACK } gives an image graph where WHITE and BLACK are not connected. (because it's not possible to go from a white square to a black square via a bishop's move).
- Given a graph homomorphism where the image is a cyclic graph, a closed tour of the original graph is possible only if all the equivalence classes have equal cardinalities. The closed tour observation for a bipartite graph is a special case of this.

- In the above case, a (not necessarily closed) tour is possible only if the cycle can be divided into two arcs such that the cardinalities of the equivalence class corresponding to one arc is one more than the cardinalities of the equivalence classes corresponding to the other arc. The observation for a bipartite graph, that the cardinalities of the two classes of vertices must differ by at most 1, is a special case of this.

  This shall become clearer with some examples.

In the next section we look at the chessboard-style graphs and analyze what sort of colouring arguments are feasible for them.

3.6. **Chessboard style graphs.** A chessboard can be thought of as a special case of a finite subset of the lattice of points with integer coordinates in the plane. In fact, the way we had coordinatized the chessboard in section 2.3 is a special case of this.

In general, we are interested in the following setup :

- The vertex set is a finite subset of the lattice of points with integer coordinates in the plane. It is termed the **board**.
- There is a finite set of vectors corresponding to the moves, such that two vertices are adjacent iff their difference is a vector corresponding to the move. That is, each move is of the type – "increase the x coordinate by p and the y coordinate by q". For convenience, the vectors are themselves designated as the **moves**, so the move will simply be called (p, q). We assume symmetry of the moves, so the negative of a move is also a move. Thus, if (p, q) is a move, so is $(-p. - q)$.

Then colourings may be divided into two classes :-

(1) Colourings that can be extended to colourings of the whole plane – with the property that given any translate of the board, the corresponding colouring on the translate is the same (possibly with the equivalence classes permuted).
(2) Other colourings – typically those that are dependent on the shape of the board and exploit its special features.

3.6.1. *Colourings of the first type.* This subsection concentrates on the first type of colouring. Here, we assume the vertices to be given x and y coordinates, the x coordinate indicating the row number, and the y coordinate indicating the column number. (refer section 2.3).

A simple colouring rule is given as follows :

$$\chi(x, y) \equiv (x + y) \mod n$$

That is, $\chi(x, y)$ is the equivalence class of $x + y$ modulo an integer n. Two vertices are given the same colour iff they leave the same remainder modulo n. Moving n steps to the right gives a vertex of the same colour, and moving n steps up gives a vertex of the same colour. Moving one step down and one step right gives a vertex of the same colour.

The entire lattice is partitioned into n equivalence classes, viz colours. If the board is now translated by a certain amount horizontally or vertically, the number of vertices of each colour will change, but this is only up to a permutation of the equivalence classes.

Any move (p, q) takes the equivalence class r (modulo n) to the equivalence class $r + t$ (modulo n) where t is the equivalence class of $p + q$ modulo n. Thus each move permutes the equivalence classes in the lattice plane.

However, each move may permute the equivalence classes in a different manner. It would be preferable if all the moves induced the same permutation on the equivalence classes.

Let us see what happens in the case of the knight's tour. We colour as above with $n = 2$, to get the usual chessboard colouring, with the remainder 0 indicating the colour Black and the remainder 1 indicating the equivalence class White.

The vectors corresponding to the eight knight's moves are :

- $(2, 1)$: The sum is 3, which is 1 modulo 2
- $(1, 2)$: The sum is 3, which is 1 modulo 2
- $(2, -1)$: The sum is 1, which is 1 modulo 2
- $(1, -2)$: The sum is -1, which is 1 modulo 2
- $(-1, 2)$: The sum is 1, which is 1 modulo 2
- $(-1, -2)$: The sum is -3, which is 1 modulo 2
- $(-2, 1)$: The sum is -1, which is 1 modulo 2
- $(-2, -1)$: The sum is -3, which is 1 modulo 2

All the remainders are 1 modulo 2, which is just a reformulation of our earlier observation – a knight's move always changes the colour of the square.

We would like something similar whenever we define moves of a piece. That is, we would like to choose a colouring with n such that all the moves have the same effect modulo n. The issue is explored in more detail in the coming Concept Testers.

What happens once we choose n? Then the image graph becomes a union of disjoint cycles. That is, the equivalence classes of lattice points modulo n are partitioned into disjoint cycles.

Analogous to a colouring based on the sum of coordinates, we can have a colouring based on the difference of coordinates (not the absolute difference, but the signed difference).

More generally it is possible to have a colouring based on the equivalence class of $ax + by$ modulo n, under the proviso that a, b, and n are pairwise relatively prime.

Concept Testers

(1) Consider an $a \times b$ board, with the colouring defined by

$$(x, y) \mapsto (x + y) \mod n$$

Prove that there are an equal number of squares of every colour if and only if $n|a$ or $n|b$. [6] Under what conditions can the colours can be partitioned into two classes, with all colours in one class having one more square than all colours in the other class?

(2) Suppose we consider a piece such that for every move (u, v) of it,

$$|u + v| = k$$

Using the colouring of the previous problem, determine some necessary conditions for a and b for which the piece can tour an $a \times b$ board.

(3) What happens if we consider the colouring

$$(x, y) \mapsto (px + qy) \mod n)$$

where p and q are both relatively prime to n?

(4) The **taxicab distance** between two points on the board, with coordinates (x_1, y_1) and (x_2, y_2) is given by :

$$|x_2 - x_1| + |y_2 - y_1|$$

[6] $n|a$ means that n is a factor of a, and is spoken as "n divides a".

A k leaper is a piece that can move, from a given square, to all squares with a taxicab distance of k from that given square.

Prove that the biggest n modulo which all moves by a k leaper have the same effect for the $(x+y)$ colouring is 2. Thus, show that for a k leaper to be able to tour a board, it is necessary that k be odd. What happens if we choose a $(px+qy)$ colouring?

We shall later try to look for sufficient conditions.

(5) The **Euclidean distance** or **crow's distance** between two points on the board, with coordinates (x_1, y_1) and (x_2, y_2) is given by :

$$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$

A **Euclidean k leaper** is a piece that can move, from a given square, to any square whose Euclidean distance from that square is k. For instance, a knight is a Euclidean $\sqrt{5}$ leaper.

Prove that a necessary condition for a Euclidean k leaper to be able to tour a board is that k^2 be $4w+1$ where w is a nonnegative integer.

3.6.2. *Colourings and partitionings of the second type – based on the board shape.* This is best illustrated by a classical problem :[7]

Problem 2. Prove *that there is no closed knight's tour of a board of dimensions* $4 \times n$.

The colouring of the lattice plane, by the coordinate sum method, turns out to be the usual chessboard colouring. This is not sufficient in itself to derive a contradiction as the number of squares of both colours are equal.

A little experimentation with trying to look at a board of size 4×4 or so, tells us that the problem is the *lack of elbow space* of the knight. The move is *too big* for the board, or looked at another way, the board is *too narrow* for the move.

This can best be captured by the following observation : the board has 4 rows and n columns. Suppose that the top and bottom rows are termed *extreme* rows, and the others are termed *middle rows.* Then every move from an extreme row goes to a middle row.

For a closed tour to exist, it becomes necessary that every move from a middle row go to an extreme row (because the total number of vertices in the middle rows equals that in the extreme rows). But where does that leave us?

Now comes the crucial idea : to *combine* the two colourings. That is, we consider assigning 4 colours : the white extreme row vertices, the black extreme row vertices, the white middle row vertices, the black middle row vertices.

Then as every black extreme row vertex goes only to a white middle row vertex, and the white middle row vertex must again go to a black extreme row vertex, it is not possible to have a tour from the black extreme row vertex to the white extreme row vertex. Similarly it is not possible to go from a white extreme row vertex to a black extreme row vertex. So the tour is not possible.

Here we have used two relatively subtle techniques:

[7] I first learnt of this problem during the IMO training camp, in a class by S.S. Sane. Later, I also read about it in **Problem Solving Strategies** by Arthur Engel, in his chapter on **Colouring Proofs**.

- Two colourings have been combined to give a single colouring. These type of combinations are typically termed *direct products* or *Cartesian products*. The final number of distinct colourings is the product of the number for each colouring. In some cases, it may happen that certain colour combinations do not exist.
- While every move from an extreme row vertex goes to a middle row vertex, there is *per se* no need for a move from a middle row vertex to go to an extreme row vertex. However, as the number of extreme row and middle row vertices are equal, each move must alternate between the colours for a tour to be possible.

Concept Testers

(1) Suppose we are to give an (not closed) tour of a $4 \times n$ board. **Prove** that such a tour must begin and end at a square in the corner rows, and that it goes from a square in a middle row to a square in a middle row exactly once. **Find** a tour of a 4×3 board. (There are 8 such tours).

(2) Prove that for $n = 1, 2, 4$ there is no knight's tour of a $4 \times n$ board. (For $n = 4$ the earlier knight's move graph of a 4×4 board drawn in section 2.3 might prove useful).

(3) We have seen that it is not possible for a knight to tour a $4 \times n$ chessboard. Can we say something similar about a Euclidean $\sqrt{13}$ leaper?

More generally, what are the boards that a Euclidean $\sqrt{n}$ leaper cannot tour if n can be written uniquely as a sum of squares of an odd integer and an even integer that are relatively prime? Apply size arguments analogous to those used for the knight.

4. Constructing a tour

4.1. **The naive algorithm.**

We now discuss methods to construct a tour with a given initial point. A tour may be treated as a sequence of moves. The **naive algorithm** or **brute force algorithm** for determining whether or not a tour exists, is based on a notion of blocking vertices that have already been visited. The unblocked vertices are also called free vertices. A procedural listing of the algorithm is described below :

ALGORITHM : Tour determination for board

```
(1) Initialize all vertices as free.
(2) Start from the initial point, and block it.
(3) List all the adjacent vertices, that are currently free.
(4) If the list is nonempty, move to any of the vertices. Block this vertex. Then repeat
    the previous step with that as the initial vertex.
(5) If the list is empty, and all the vertices have been visited, then we have a tour.
(6) If the list is empty, but all vertices have not been visited, free the current vertex and
    move to the previous one (now we try for a tour only from the remaining moves in
    the list).
```

This search for a tour is an instance of **Depth First Search** or D.F.S.. A D.F.S. is a search method on a rooted tree where each possibility is explored to its full depth, before exploring the next. For instance, here, we try each possible path for touring to its full depth and turn back only when forced to.

4.2. **Understanding and improving the D.F.S..**

In a D.F.S. there is a distinct possibility of going down a **dead end** or **blind alley** – that is, it may well happen that we choose a move that does not give rise to any correct tour. Then as soon as the list of free adjacent vertices becomes empty, we retreat, and try the other possible moves. This retreating is, somewhat naturally, termed **backtracking**.

The more the backtracking, in general, the more the time taken to find a tour. Thus, the key idea to determining a tour most quickly is to minimize backtracking. The standard approach of doing so is to impose an **order of preference** on the moves, that is, to decide on what order we try to perform the moves in. Deciding this order of preference must be a very quick operation, hopefully a function of the total number of moves possible, rather than of the size of the board, or the length of the tour so far. A D.F.S. that makes us of some order of preference is often termed a **best first search** – a D.F.S. where the best is given the first chance.

This order of preference is typically generated by trying to optimize using some **local heuristic**. Algorithms that use a local heuristic to make the decision at each stage form a class known as **greedy algorithms** . A greedy algorithm can be thought of as being truly successful, if it requires *no* backtracking.

Thus, in the problem of the Hamiltonian path (Core Problem 2) the greedy algorithm shall be considered successful provided that :

- In case a Hamiltonian path exists, the greedy algorithm gives one such Hamiltonian path, without hitting any dead ends.
- In case a Hamiltonian path does not exist, the greedy algorithm gives a dead end.

4.3. **Giving a greedy algorithm.** The question we need to ponder is : what is the best local heuristic for a successful tour? How should we decide how to make the next move?

We introduce the concept of **degree** of a vertex in a graph, as the number of vertices to which it is adjacent. If a vertex has zero degree, it is not adjacent to any other vertex, and hence cannot be contained in any Hamiltonian path.

The degree of a vertex essentially contains information about how it can be placed in paths. Vertices with a low degree are in some sense *more critical* as there are fewer ways of including them in paths. Vertices with a higher degree are *less critical* as there are plenty of ways of including them in paths.

A little thought reveals that :

- Every vertex except the initial and final vertex must have degree at least 2.
- If v is a vertex of degree 1 other than the initial vertex, and w is its neighbour, then not only must v be the last vertex, w must be the second last vertex.
- Further, if v is a vertex of degree 2 with neighbours u and w, then either v is the last vertex and one of u and w is the second last vertex, or else v must be between u and w. This means that as soon as u is visited, v must be the next vertex, and w must be the vertex after that.

Generalizing we can say : if there is a neighbour of very low degree, visit that when the chance comes. The *degree* here should not be construed as simply the degree in the initial graph, rather it must be viewed as the number of adjacent vertices that are still free. This tells us that the following simple rule can be used to assign the order of preference :

Heuristic 2 (Warnsdorff's rule)**.** *A vertex of lower (free) degree (which is more critical) is given preference over a vertex of higher (free) degree. Vertices of equal degree can be given any order of preference.*

This is termed the **Warnsdorff rule**. Surprisingly, the rule always gives a tour (though not necessarily a re-entrant one) for an 8×8 chessboard.

An extensive literature is available on the knight's tour, and there has also been investigation on the theoretical justification of Warnsdorff's rule and the graphs to which it is applicable. Details are to be found in the appendix.

While Warnsdorff' rule does guarantee a tour from any starting point, it does not guarantee a *re-entrant tour* from every starting point (though it may sometimes gives a re-entrant tour). In fact, the heuristic is not naturally suited for giving re-entrant tours. Historically, the existence of a re-entrant knight's tour was established by Vandermonde by actually constructing the tour. Vandermonde used powerful notions of symmetry, some of which we discuss in section 4.4.

Warnsdorff's rule is also not very suited to the problem of finding the *longest possible path* in graphs where a Hamiltonian path may not exist. This is because, in an effort to go to vertices of low degree, the path might reach a dead end much more quickly. This is a bit like saying that if we want *everybody* to live, we must feed those who are hungriest. But if it simply isn't possible to save the hungriest even by trying to feed them, we might do better to feed those who have chances of surviving.

Concept Testers

(1) What happens if we apply Warnsdorff's rule to the 4×4 knight's move graph?
(2) Starting from a corner most square in the 4×3 knight's move graph, can we get a tour using Warnsdorff's rule?
(3) Warnsdorff's rule simply gives first preference to the adjacent vertex with lowest free degree. Suppose that to each vertex, we associate the sum of free degrees of the vertices adjacent to it. Does this give a good local heuristic for the chessboard?
(4) The 3×10 and 5×6 board are the smallest ones for which a closed knight's tour exists. Can such a tour be found by Warnsdorff's rule?

4.4. **Symmetry principles in tour construction and Vandermonde's work.** Here we move along with Vandermonde's train of thought for the construction of a closed knight's tour of the 8×8 chessboard.

Here is his broad strategy :

ALGORITHM : Finding a closed tour

(1) List all possible squares on the board and their corresponding co-ordinates. i.e. 64 sets of co-ordinates.
(2) Choose an arbitrary starting point and remove it from the list of all possible squares. This square becomes the beginning of path list *one*.
(3) Using successive rotational symmetry of $\pi/2$ about the center of the board create three further path lists of co-ordinates that begin with each of the three rotations co-ordinates. Then delete these sets of co-ordinates from the list of all possible squares leaving 60 squares.
(4) From the starting point in path list *one* pick an arbitrary knight's move from this square (such that it is in the remainder of the list of all possible squares) and note this as move 2 in path list *one*. Remove it from the list of all possible squares that will now be left with 59 sets of co-ordinates.

(5) Repeat the three rotations for move 2 and add them to their respective path lists remembering to delete them from the list of all possible squares leaving 56 sets of co-ordinates.
(6) Continue with this process until all squares are used up and there will be four paths lists each of 16 sets of co-ordinates. After Vandermonde we call them our **symmetric lists of moves**.

Vandermonde noted that the end of the first symmetric list was a knight's move away from the end of the fourth, and the end of the second symmetric list was a knight's move away from the end of the third. Thus he got two symmetric sequences of length 32.

To get a closed tour of length 64 he tweaked this a little bit.

4.5. **Conrad's algorithm.**

While Warnsdorff's rule works well for the chessboard, it runs into problems for huge graphs. In particular, it runs into blind alleys (and hence requires backtracking) for chessboards of dimension bigger than 76×76.

The algorithm works by decomposition of the chessboard into smaller chessboards for which previous solutions are known. Though the running time of the algorithm is linear, it is somewhat complicated to code.

5. Variations in the basic problem

5.1. **Infinite tours.**

The graphs arising as finite subsets of integer lattices in the plane with some defined *moves* have been studied in detail. Two variations on the theme are :

(1) Move from the plane to higher dimensional lattices of integer coordinates.
(2) Allow infinite subsets of the lattice plane.

In the first case, there is no fundamental change to the theory : most of the colouring ideas can still be applied in much the same way.

The second case makes it necessary for us to define what precisely is meant by a tour of a (countably) infinite graph. There are two types of tours :

- Bijections between $\mathbb{N}$[8] and the vertices of the graph such that the images of n and $n+1$ are adjacent in the graph, where $n \in \mathbb{N}$. This can be thought of as a tour with a starting point.
- Bijections between $\mathbb{Z}$[9] and the vertices of the graph such that the images of n and $n+1$ are adjacent in the graph where $n \in \mathbb{Z}$. This can be thought of as a tour extending indefinitely in both directions.

The existence of a tour of the entire graph is a stronger condition than the existence of a tour covering a subgraph containing all of an arbitrarily large finite subset. This distinction between *arbitrarily large finite* and *infinite*, though subtle, is a crucial one, and occurs in different guises across mathematics.

Concept Testers

(1) Prove that a 1 leaper can tour the lattice plane $\mathbb{Z}^2$.
(2) Consider $\mathbb{Z}$ as a one dimensional board, with the move set being a subset $S \subseteq \mathbb{Z}$, such that $s \in S \implies -s \in S$. Does there exist a tour if S is the :
    - Set of primes and their negatives

[8] $\mathbb{N}$ is the set of natural numbers, or positive integers
[9] $\mathbb{Z}$ is the set of integers

- Set of squares and their negatives

(3) Solve the above problem replacing $\mathbb{Z}$ by $\mathbb{N}$.

(4) Solve the above two problems under the further assumption that every element in S can be used only finitely many times as a move.

5.2. **Simultaneous and sequential touring.** Before discussing this, we make a small comment. Chess pieces, both standard chess pieces and fairy chess pieces, are classified according to the way they are allowed to move. The knight falls under the class of **leapers**. Leapers are pieces that move *directly* from the departure square to the destination square, regardless of whether or not any other square is occupied.

The distinction between leapers and other pieces becomes important when the chessboard has more than one piece. We are in general concerned with the **sequential motion** :

> All the pieces on the chessboard are leapers, with some well defined (symmetric) moves. At each stage, one leaper makes a move, and all the leapers must occupy distinct squares at each stage.

The advantage of sticking to leapers in this scenario is that the sequential motion scenario can be expressed *purely in terms of the graph properties*.

A somewhat more powerful scenario (that makes sense for leapers) is the **simultaneous motion** scenario :

> All the pieces on the chessboard are leapers, with some well defined (symmetric) moves. At each stage, some of the leapers simultaneously move, and all the leapers must occupy distinct squares at each stage. Also, no two leapers can interchange their squares.

We now consider a classical problem :

Problem 3 (Simultaneous touring)**.** *Consider a 3×3 board with a knight placed on each of the squares that are the middle squares of sides (that is, neither the middle nor the corners). There are four knights in total. We allow these knights simultaneous motion, that is, at each stage, one or more of the knights move, with no two knights ever landing on the same square, and no two knights interchanging their places. Is it possible that, after some moves, the knight in the left column and right column interchange positions while the knights in the upper row and lower row both get back to where they were?*

I came across this problem in **Mathematical Circles**.

The answer is *no*.

Proof. We had seen earlier (in subsection 2.3) that the 3×3 knight's move graph is the union of a cycle and an isolated vertex (the middle square).

In each move, each knight either stays where it is, or moves one step clockwise or one step anticlockwise along the cycle. However, by the conditions we have imposed, it is never possible for one knight to overtake another. This indicates that the **cyclic order** of the knights is an **invariant** throughout the touring process.

However, the cyclic order in the initial configuration and in the final configuration differ. Hence, the final configuration is not attainable from the initial configuration. $\square$

The proof technique here makes use of the **Invariance Principle**.

Concept Testers

(1) Consider a 4×4 chessboard with knights placed on $(1,2)$, $(2,4)$, $(4,3)$ and $(3,1)$. (Refer the notation used in section 2.3). A **permutation** of these knights is a rearrangement where these knights occupy the same four squares, but possibly with some knights occupying different squares. What permutations can be achieved through simultaneous touring? What permutations can be achieved through sequential touring?

Appendix A. General problems

A.1. **Three dimensions and higher.** These problems are meant to encourage further exploration related to the text of the article covered so far. Most of them are *not* straightforward, and they are of a somewhat higher level than the Concept Testers found within the main body of the article.

We now consider three dimensional boards, where each point is given by three coordinates. A **knight** is a piece that, in one move, changes one coordinate by 2 and another by 1.

(1) Construct the knight's move graph for a $3 \times 3 \times 3$ board. Is a knight's tour of this graph (minus the middle vertex) possible?
(2) Prove that there is no closed knight's tour of a $4 \times m \times n$ board. Is there a knight's tour of a $4 \times 4 \times 3$ board? What about a $4 \times 3 \times 3$ board? A $4 \times 4 \times 4$ board?

A.2. **More sophisticated colourings.**

(1) Prove that any colouring defined for some rectangle, upto periodic translation, can be expressed in terms of a congruence. In particular, give a single expression for the following colouring :
   - **Red** if $x \equiv 0 \mod 2$ and $y \equiv 0 \mod 2$.
   - **Green** if $x \equiv 0 \mod 2$ and $y \equiv 1 \mod 2$.
   - **Blue** if $x \equiv 1 \mod 2$ and $y \equiv 0 \mod 2$.
   - **Yellow** if $x \equiv 1 \mod 2$ and $y \equiv 1 \mod 2$.

   Hint : Think of the binary expansion with one digit storing x modulo 2 and the other storing y modulo 2.
(2) Generalize this to periodic patterns for any tiling.
(3) Prove that the colouring based on $x^2 + y^2 \mod 5$ can be obtained as a colouring based on linear combinations of x and y modulo 25. Can it be obtained based on linear combinations of x and y modulo 5?

A.3. **Other graphs.** The famous **Icosian game** or **Hamiltonian game** asks for the determination of a Hamiltonian cycle in the graph of a dodecahedron where the vertices are given as the vertices of the dodecahedron, and the edges are given as the edges of the dodecahedron.

Find a solution to this game.

Appendix B. Miscellanea on the knight's tour

B.1. **Some online references.** Introductory reading on the knight's tour can be obtained at :

`http://mathworld.wolfram.com/KnightsTour.html`
`http://en.wikipedia.org/wiki/Knight's_Tour`
`http://www.ktn.freeuk.com/ca.htm`

`http://www.mathpuzzles.com/leapers.htm`
`http://home.freeuk.net/ktn`

The Wikipedia and Mathworld pages also contain extensive information on other chess puzzles.

A site containing links to many knight's tour pages is :

`http://www.velucchi.it/mathchess/knight.htm`

The algorithm developed by Conrad *et al.* can be found in the paper by Axel Conrad, Tanja Hindrichs, Hussein Morsy, and Ingo Wegener titled "Solution of the knight's Hamiltonian path problem on chessboard" published in Discrete Applied Mathematics, Volume 50, Number 2, year 1994.

B.2. **Magic and semimagic tours.** A **magic square** is a $n \times n$ square of cells with numbers in each cell such that all the row sums, all the column sums and both the principal diagonal sums are equal. A **semimagic square** is one where only one of the diagonal sums fails to be equal.

We can fill entries from 1 to 64 on the chessboard based on the knight's tour by marking the k_{th} square of the tour with the entry k. It thus makes sense to use properties of this way of writing the numbers from 1 to 64, to give properties of the knight's tour. In particular, a **magic knight's tour** is a knight's tour such that the resulting filling of entries gives a magic square, while a **semimagic knight's tour** gives a semimagic square.

B.3. **Other good tours.** Some other properties that are looked for in knight's tours include:

- The property of being **uncrossed**, that is, the paths of the knight should not cross itself.
- The property of possessing **axial symmetry** and/or **rotational symmetry**.

The properties of being magical, semimagic, uncrossed and symmetric are *not* purely graph theoretic as they depend on the geometry of the chessboard, which is not captured in the knight's move graph. Since our treatment has been largely of aspects that are *captured by the graph*, we have not gone into these ideas.

Appendix C. Terms and definitions

C.1. **Basic definitions of graphs.**

(1) **Simple Graph** : A structure given by a set of **vertices** V, and a set of **edges** E where each edge is an (unordered) pair of vertices. If $|V|$ is finite then we say that the graph is finite.
(2) **Vertex** or **node** or **point** is an element of V in the definition of graph.
(3) **Edge** (or **arc** or **line**): An unordered pair of vertices, as an element of E in the definition of graph. The two vertices are **endpoints** of the edge. The edge is said to be **incident** on the vertices and the vertices are said to be **incident** on the edges. The vertices are also said to be **adjacent** to each other.
(4) **Directed graph** : A graph where the edges are *ordered pairs* of vertices rather than unordered pairs.
(5) **Loop** : An edge whose endpoints are both the same vertex. Loops are not allowed in simple graphs.

(6) **Parallel edges** : Multiple edges with the same endpoints. These are also not allowed in a simple graph.
(7) **Path** : A sequence of vertices of the graph such that any two vertices adjacent in the sequence are adjacent in the graph.
(8) **Cycle** : A path that begins and ends at the same vertex.
(9) **Hamiltonian path** : A path that covers every vertex exactly once.
(10) **Hamiltonian cycle** or **Hamiltonian circuit**: A cycle that covers every vertex exactly once.
(11) **Graph homomorphism** : A map from the vertex set of one graph to another such that the images of adjacent vertices in the original graph are adjacent in the other graph.
(12) **Image graph** : Given a graph homomorphism, a subgraph of the graph on the right whose vertex set and edge set are the precise images of the vertex set and edge set of the original graph.
(13) **Chromatic number**: The minimum number of parts into which the vertex set of a graph can be partitioned so that no two vertices in the same part are adjacent.
(14) **Degree** : The degree of a vertex in a (simple) graph is the number of vertices adjacent to it.

C.2. **Properties of simple graphs.**

(1) **Connected** : There is a path from any vertex to any other vertex.
(2) **Finite** : The vertex set is finite.
(3) **Cyclic** : All the edges of the graph can be covered by a cycle.
(4) **Acyclic** : It has no cycles.
(5) **Hamiltonian** : It has a Hamiltonian cycle.
(6) **Bipartite** : The vertex set can be partitioned such that there is no edge joining two vertices in the same part.
(7) **Planar** : The graph can be drawn on a plane with the vertices as points and the edges as arcs such that no two edges intersect except at vertices.

C.3. **Kinds of searches.**

(1) **Depth First Search** or D.F.S.: A form of search where one path is explored to its full depth before trying out alternative paths. When a **dead end** is encountered, the path is retraced till the last choice point is identified, and an alternative path is tried. D.F.S. is a **tree search algorithm** an example of an **uninformed search**.
(2) **Best First Search** : A variant of D.F.S., where the first path to try is determined by the use of some heuristic, that makes it look more probable than other paths.
(3) **Local heuristic** : A heuristic for making a choice locally, that typically requires a small amount of time, rather than time proportional to the whole problem size.
(4) **Greedy algorithm** : An algorithm that tries to determine a globally optimal solution using local heuristics.
(5) **Backtracking** : The process of retracing the path after encountering a dead end in a D.F.S..

Appendix D. Other chess and fairy chess puzzles

D.1. **Graph theoretical formulation of the other puzzles.** The other two problem classes mentioned at the outset have the following graph theoretical formulations :

- The **nonattacking pieces puzzle** generalizes to the problem of finding an **independent set** of maximum cardinality. An independent set of vertices in a graph is a set of vertices such that no two are adjacent in the original graph.
- The **board domination puzzles** generalizes to the problem of finding a **dominating set** of minimum cardinality. A dominating set is a set such that every vertex is adjacent to some vertex in the set.

D.2. **Types of fairy chess pieces. Fairy chess** is chess with some added pieces with their own rules. Fairy chess pieces are of three types :

- **Leapers** that move directly from the departure square to the destination square, regardless of what pieces are there in the intermediate square. The knight is a leaper.
- **Riders** are pieces that can move an arbitrary amount in a given direction, provided there are no pieces in between. The rook, bishop and queen are examples of riders from orthodox chess.
- **Hoppers** are pieces that can move only by jumping over other pieces. There are no hoppers in ordinary chess, but the moves of hoppers can be thought of as similar to moves in checkers. Hoppers, however, capture the piece on the square they finally reach, not the piece they jump over (termed the **hurdle**).

More information on fairy chess pieces is to be found on Wikipedia at :

`http://en.wikipedia.org/wiki/Fairy_chess`

D.3. **Chess maneuvers and algorithms.** Ideas used for mastery in games are often closely related to tricky algorithmic steps in computer science. In a game like chess, for which a deterministic analysis is computationally infeasible, several local heuristics have been designed.

These local heuristics typically tend to focus on maximizing some quantity. An instance of application of local heuristics was seen in the use of greedy algorithms to convert depth first searches to best first searches. In the case of standard chess, local heuristics typically include things like *save the queen* and *destroy the opponent's pieces.*

The programming paradigm ideal for applying local heuristics is **constraint programming**. This is a variation in the basic theme of **logic programming** and lies in the realm of **declarative programming**. More on this can be read up at Wikipedia.

It would be interesting to explore, from this viewpoint, the significance of **forks**, **pins**, **skewers**, **exchanges**, **sacrifices**, and other chess tactics.

Appendix E. Related topics and references

E.1. **Colouring arguments and invariance principle.** An excellent collection of problems on colouring arguments in touring, tiling and other related problems is the chapter on **Colouring Proofs** in **Problem Solving Strategies** by Arthur Engel, published by Springer International.

Another good source of interesting logical-mathematical puzzles is **Mathematical Circles** by Fomkin, Genkin and Itenberg, published by the **American Mathematical Society** as part of the **Mathematical Worlds** series.

E.2. **Graph theory.** As this article illustrates, graphs have the propensity to turn up at the most unexpected places. Their utility lies in their ability to model a large number of situations. Consequently, the range of possibilities and properties for graphs is very huge. Some references for graph theory are :

- **An Introduction to Graph Theory** by Douglas B. West
- **Graph Theory** by Harary
- **Graph Theory** by Clark and Houlton, published by World Scientific

E.3. **Algorithms.** In this article, we studied the blind depth first search as a naive algorithm for finding a Hamiltonian path in a graph, and then, how the use of Warnsdorff's rule (a local heuristic) gave a best first search with no backtracking.

We also discussed that for problems attacked by a D.F.S., the solution is more efficient if there is minimal backtracking.

In general, there is a wide range of algorithmic techniques used to solve problems, and, for each technique, there are notions of how the solution can be improved. There are also general notions of the **complexity** of an algorithmic solution to a problem. Some good references on algorithms include :

- **Algorithms : The Spirit of Computation** by David Harel
- **Introduction to Algorithms** by Corman, Leiserston, and Rivest (the new edition also has Stein as an author), published by Prentice Hall

A webpage where information on algorithms can be found is the **Dictionary of Algorithms and Data Structures** at :

`http://www.nist.gov/dads/`

Index