A mathematical model can be used to help design a traffic light for a complicated intersection of roads. To construct the pattern of lights, we shall create a program that takes as input a set of permitted turns at an intersection (continuing straight on a road is a "turn") and partitions this set into as few groups as possible such that all turns in a group are simultaneously permissible without collisions. We shall then associate a phase of the traffic light with each group in the partition. By finding a partition with the smallest number of groups, we can construct a traffic light with the smallest number of phases.

For example, the intersection shown in Fig. 1.1 occurs by a watering hole called JoJo's near Princeton University, and it has been known to cause some navigational difficulty, especially on the return trip. Roads *C* and *E* are oneway, the others two way. There are 13 turns one might make at this intersection. Some pairs of turns, like *AB* (from *A* to *B*) and *EC*, can be carried out simultaneously, while others, like *AD* and *EB*, cause lines of traffic to cross and therefore cannot be carried out simultaneously. The light at the intersection must permit turns in such an order that *AD* and *EB* are never permitted at the same time, while the light might permit *AB* and *EC* to be made simultaneously.

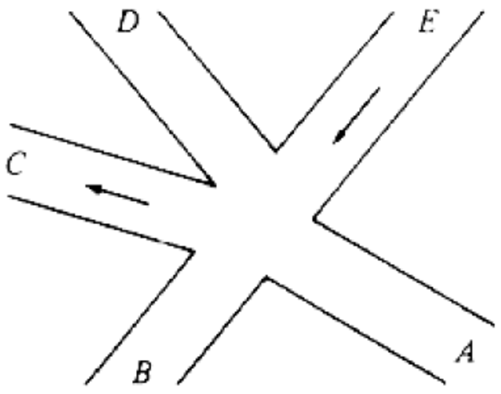


Fig. 1.1. An intersection.

We can model this problem with a mathematical structure known as a graph. A *graph* consists of a set of points called *vertices*, and lines connecting the points, called *edges*. For the traffic intersection problem we can draw a graph whose vertices represent turns and whose edges connect pairs of vertices whose turns cannot be performed simultaneously. For the intersection of Fig. 1.1, this graph is shown in Fig. 1.2, and in Fig. 1.3 we see another representation of this graph as a table with a 1 in row *i* and column *j* whenever there is an edge between vertices *i* and *j*.

The graph can aid us in solving the traffic light design problem. A *coloring* of a graph is an assignment of a color to each vertex of the graph so that no two vertices connected by an edge have the same color. It is not hard to see that our problem is one of coloring the graph of incompatible turns using as few colors as possible.

The problem of coloring graphs has been studied for many decades, and the theory of algorithms tells us a lot about this problem. Unfortunately, coloring an arbitrary graph with as few colors as possible is one of a large class of problems called "NP-complete problems," for which all known solutions are essentially of the type "try all possibilities." In the case of the coloring problem, "try all possibilities" means to try all assignments of colors to vertices using at first one color, then two colors, then three, and so on, until a legal coloring is found. With care, we can be a little speedier than this, but it is generally believed that no algorithm to solve this problem can be substantially more efficient than this most obvious approach.

We are now confronted with the possibility that finding an optimal solution for the problem at hand is computationally very expensive. We can adopt one of three approaches. If the graph is small, we might attempt to find an optimal solution exhaustively, trying all possibilities. This approach, however, becomes prohibitively expensive for large graphs, no matter how efficient we try to make the program. A second approach would be to look for additional information about the problem at hand. It may turn out that the graph has some special properties, which make it unnecessary to try all possibilities in finding an optimal solution. The third approach is to change the problem a little and look for a good but not necessarily optimal solution. We might be happy with a solution that gets close to the minimum number of colors on small graphs, and works quickly, since most intersections are not even as complex as Fig. 1.1. An algorithm that quickly produces good but not necessarily optimal solutions is called a *heuristic*.

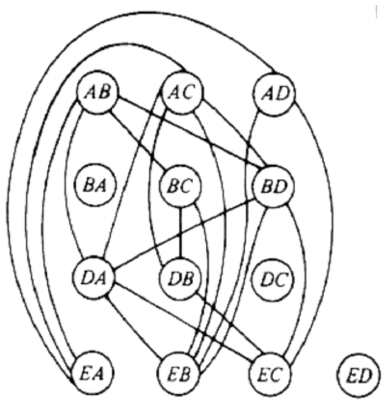
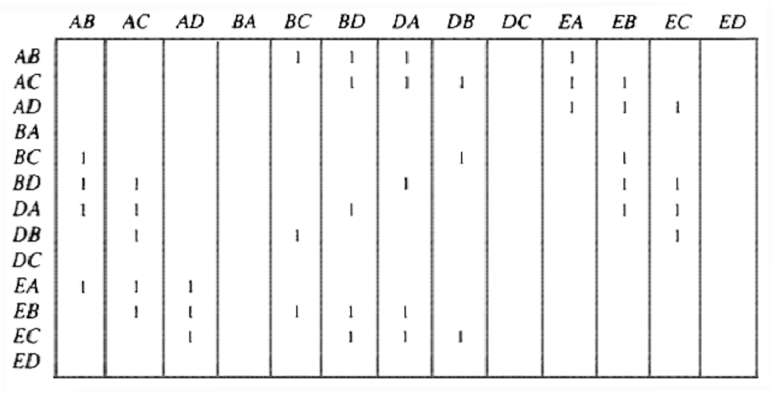


Fig. 1.2. Graph showing incompatible turns.



	AB	AC	AD	BA	BC	BD	DA	DB	DC	EA	EB	EC	ED
AB					1	1	1			1			
AC						1	1	1		1	1		
AD										1	1	1	
BA													
BC	1							1			1		
BD	1	1					1				1	1	
DA	1	1				1					1	1	
DB		1			1							1	
DC													
EA	1	1	1										
EB		1	1		1	1	1						
EC			1			1	1	1					
ED													

Fig. 1.3. Table of incompatible turns.

One reasonable heuristic for graph coloring is the following "greedy" algorithm. Initially we try to color as many vertices as possible with the first color, then as many as possible of the uncolored vertices with the second color, and so on. To color vertices with a new color, we perform the following steps.

1. Select some uncolored vertex and color it with the new color.
2. Scan the list of uncolored vertices. For each uncolored vertex, determine whether it has an edge to any vertex already colored with the new color. If there is no such edge, color the present vertex with the new color.

This approach is called "greedy" because it colors a vertex whenever it can, without considering the potential drawbacks inherent in making such a move. There are situations where we could color more vertices with one color if we were less "greedy" and skipped some vertex we could legally color. For example, consider the graph of Fig. 1.4, where having colored vertex 1 red, we can color vertices 3 and 4 red also, provided we do not color 2 first. The greedy algorithm would tell us to color 1 and 2 red, assuming we considered vertices in numerical order.

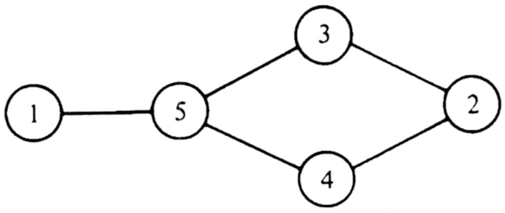
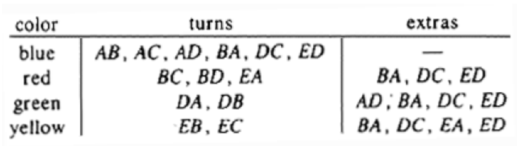


Fig. 1.4. A graph.

As an example of the greedy approach applied to Fig. 1.2, suppose we start by coloring *AB* blue. We can color *AC*, *AD*, and *BA* blue, because none of these four vertices has an edge in common. We cannot color *BC* blue because there is an edge between *AB* and *BC*. Similarly, we cannot color *BD*, *DA*, or *DB* blue because each of these vertices is connected by an edge to one or more vertices already colored blue. However, we can color *DC* blue. Then *EA*, *EB*, and *EC* cannot be colored blue, but *ED* can.

Now we start a second color, say by coloring *BC* red. *BD* can be colored red, but *DA* cannot, because of the edge between *BD* and *DA*. Similarly, *DB* cannot be colored red, and *DC* is already blue, but *EA* can be colored red. Each other uncolored vertex has an edge to a red vertex, so no other vertex can be colored red.

The remaining uncolored vertices are *DA*, *DB*, *EB*, and *EC*. If we color *DA* green, then *DB* can be colored green, but *EB* and *EC* cannot. These two may be colored with a fourth color, say yellow. The colors are summarized in Fig. 1.5. The "extra" turns are determined by the greedy approach to be compatible with the turns already given that color, as well as with each other. When the traffic light allows turns of one color, it can also allow the extra turns safely.



color	turns	extras
blue	*AB*, *AC*, *AD*, *BA*, *DC*, *ED*	—
red	*BC*, *BD*, *EA*	*BA*, *DC*, *ED*
green	*DA*, *DB*	*AD*, *BA*, *DC*, *ED*
yellow	*EB*, *EC*	*BA*, *DC*, *EA*, *ED*

Fig. 1.5. A coloring of the graph of Fig. 1.2.

The greedy approach does not always use the minimum possible number of colors. We can use the theory of algorithms again to evaluate the goodness of the solution produced. In graph theory, a *k-clique* is a set of *k* vertices, every pair of which is connected by an edge. Obviously, *k* colors are needed to color a *k*-clique, since no two vertices in a clique may be given the same color.

In the graph of Fig. 1.2 the set of four vertices *AC*, *DA*, *BD*, *EB* is a 4-clique. Therefore, no coloring with three or fewer colors exists, and the solution of Fig. 1.5 is optimal in the sense that it uses the fewest colors possible. In terms of our original problem, no traffic light for the intersection of Fig. 1.1 can have fewer than four phases.

Therefore, consider a traffic light controller based on Fig. 1.5, where each phase of the controller corresponds to a color. At each phase the turns indicated by the row of the table corresponding to that color are permitted, and the other turns are forbidden. This pattern uses as few phases as possible.