Activity 14

Tourist town—*Dominating sets*

Summary

Many real-life situations can be abstracted into the form of a network or "graph" of the kind used for coloring in Activity 13. Networks present many opportunities for the development of algorithms that are practically useful. In this activity, we want to mark some of the junctions, or "nodes," in such a way that all other nodes are at most one step away from one of the marked ones. The question is, how few marked nodes can we get away with? This turns out to be a surprisingly difficult problem.

Curriculum Links

✓ Mathematics Level 2: Position and orientation

Skills

✓ Maps.
✓ Relationships.
✓ Puzzle solving.
✓ Iterative goal seeking.

Ages

✓ 7 and up

Materials

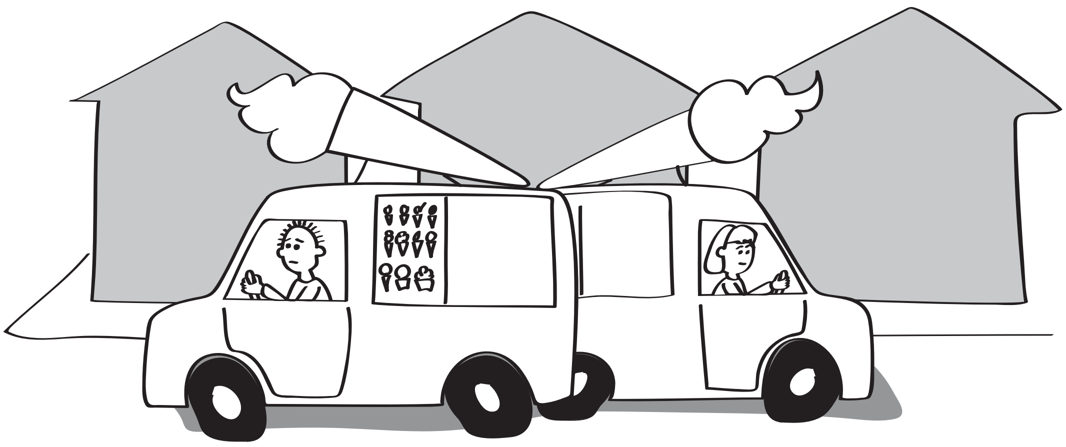
Each group of children will need:

✓ a copy of the blackline master *Ice Cream Vans*, and
✓ several counters or poker chips of two different colors.

You will need

✓ an overhead projector transparency of the blackline master *Ice Cream Vans Solution*, or a blackboard to draw it on.

Dominating Sets



Introduction

Shown on the *Ice Cream Vans* worksheet is a map of Tourist Town. The lines are streets and the dots are street corners. The town lies in a very hot country, and in the summer season ice-cream vans park at street corners and sell ice-creams to tourists. We want to place the vans so that anyone can reach one by walking to the end of their street and then at most one block further. (It may be easier to imagine people living at the intersections rather than along the streets; then they must be able to get ice-cream by walking at most one block.) The question is, how many vans are needed and on which intersections should they be placed?

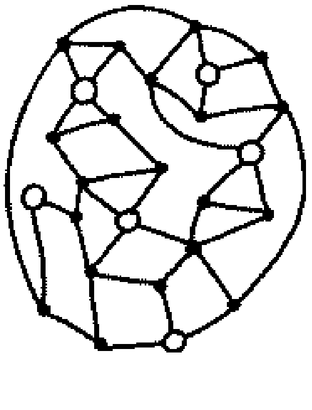
Discussion

1. Divide the children into small groups, give each group the Tourist Town map and some counters, and explain the story.

2. Show the children how to place a counter on an intersection to mark an ice-cream van, and then place counters of another color on the intersections one street away. People living at those intersections (or along the streets that come into them) are served by this ice-cream van.

3. Have the children experiment with different positions for the vans. As they find configurations that serve all houses, remind them that vans are expensive and the idea is to have as few of them as possible. It is obvious that the conditions can be met if there are enough vans to place on all intersections—the interesting question is how few you can get away with.

4. The minimum number of vans for Tourist Town is six, and a solution is shown here. But it is very difficult to find this solution! After some time, tell the class that six vans suffice and challenge them to find a way to place them. This is still quite a hard problem: many groups will eventually give up. Even a solution using eight or nine vans can be difficult to find.



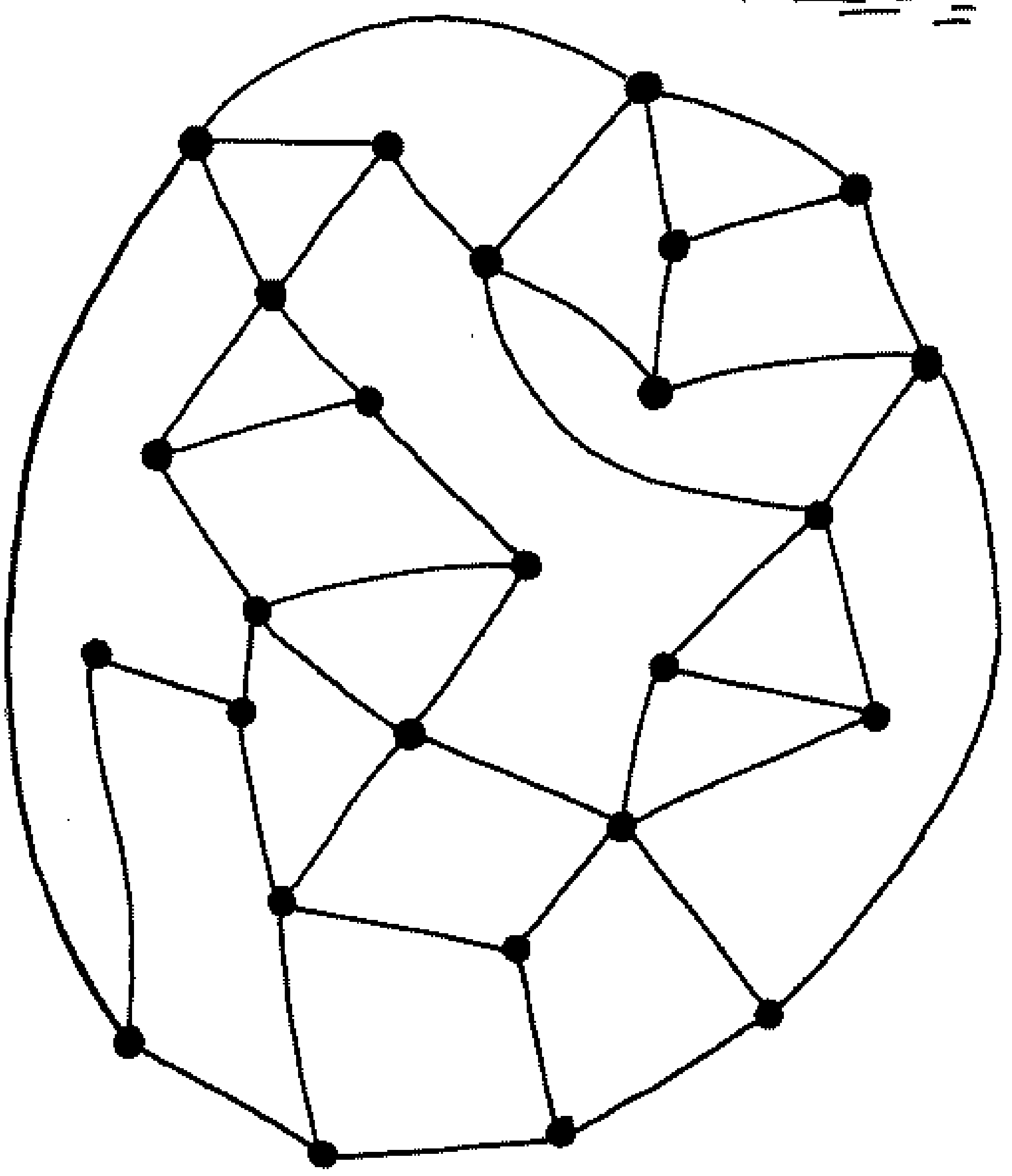
5. The map of Tourist Town was made by starting with the six map pieces at the bottom of the *Ice Cream Vans* worksheet, each of which obviously requires only one ice-cream van, and connecting them together with lots of

streets to disguise the solution. The main thing is not to put any links between the solution intersections (the open dots), but only between the extra ones (the solid dots). Show the class this using an overhead projector, or by drawing it on the board.

6. Get the children to make their own difficult maps using this strategy. They may wish to try them on their friends and parents, and will get a kick out of devising puzzles that they can solve but others can't! These are examples of what is called a "one-way function": it's easy to come up with a puzzle that is very difficult to solve—unless you're the one who created it in the first place. One-way functions play a crucial role in cryptography (see Activities 17 and 18).

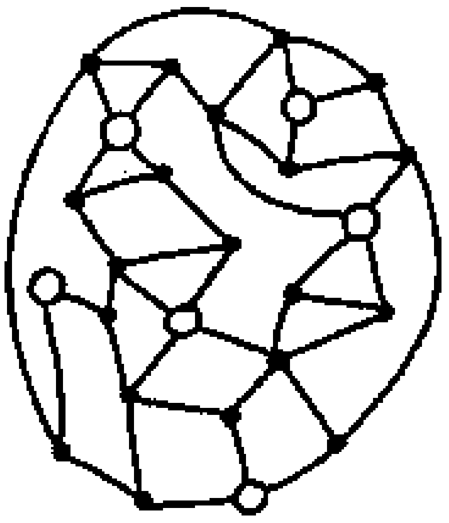
Worksheet Activity: Ice Cream Vans

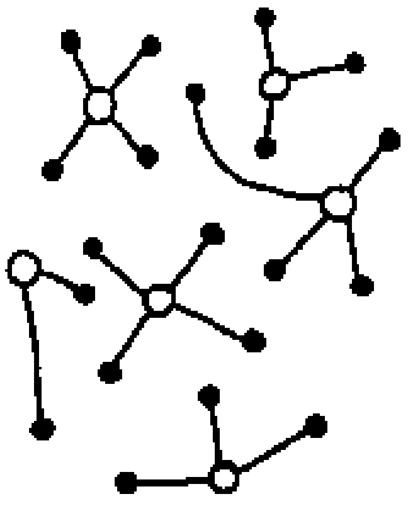
Work out how to place ice-cream vans on the street intersections so that every other intersection is connected to one that has a van on it.



Worksheet Activity: Ice Cream Vans Solution

Display this to the class to show how the puzzle was constructed.





Variations and extensions

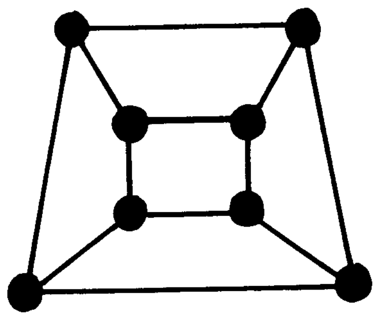
There are all sorts of situations in which one might be faced with this kind of problem in town planning: locating mailboxes, wells, fire-stations, and so on. And in real life, the map won't be based on a trick that makes it easy to solve. If you really have to solve a problem like this, how would you do it?

There is a very straightforward way: consider all possible ways of placing ice-cream vans and check them to see which is best. With the 26 street corners in Tourist Town, there are 26 ways of placing one van. It's easy to check all 26 possibilities, and it's obvious that none of them satisfies the desired condition. With two vans, there are 26 places to put the first, and, whichever one is chosen for the first, there are 25 places left to put the second (obviously you wouldn't put both vans at the same intersection): $26 \times 25 = 650$ possibilities to check. Again, each check is easy, although it would be very tedious to do them all. Actually, you only need to check half of them (325), since it doesn't matter which van is which: if you've checked van 1 at intersection A and van 2 at intersection B then there's no need to check van 1 at B and van 2 at A. You could carry on checking three vans (2600 possibilities), four vans (14950 possibilities), and so on. Clearly, 26 vans are enough since there are only 26 intersections and there's no point in having more than one van at the same place. Another way of assessing the number of possibilities is to consider the total number of configurations with 26 intersections and any number of vans. Since there are two possibilities for each street corner—it may or may not have a van—the number of configurations is 2^{26}, which is 67,108,864.

This way of solving the problem is called a "brute-force" algorithm, and it takes a long time. It's a widely held misconception that computers are so fast they can do just about anything quickly, no matter how much work it involves. But that's not true. Just how long the brute-force algorithm takes depends on how quick it is to check whether a particular configuration is a solution. To check this involves testing every intersection to find the distance of the nearest van. Suppose that an entire configuration can be tested in one second. How long does it take to test all 2^{26} possibilities for Tourist Town? (Answer: 2^{26} is about 67 million; there are 86,400 seconds in a day, so 2^{26} seconds is about 777 days, or around two years.) And suppose that instead of one second, it took just one thousandth of a second to check each particular configuration. Then the same two years would allow the computer to solve a 36-intersection town, because 2^{36} is about 1000 times 2^{26}. Even if the computer was a million times faster, so that one million configurations could be checked every second, then it would take two years to work on a 46-intersection town. And these are not very big towns! (How many intersections are there in your town?)

Since the brute-force algorithm is so slow, are there other ways to solve the problem? Well, we could try the greedy approach that was so successful in the muddy city (Activity 9). We need to think how to be greedy with ice-creams—I mean how to apply the greedy approach to the ice-cream van problem. The way to do it is by placing the first van at the intersection that connects the greatest number of streets, the second one at the next most connected intersection, and so on. However, this doesn't necessarily produce a minimum set of ice-cream van positions—in fact, the most highly connected intersection in Tourist Town, which has five streets, isn't a good place to put a van (check this with the class).

Let's look at an easier problem. Instead of being asked to find a minimum configuration, suppose you were given a configuration and asked whether it was minimal or not. In some cases, this is easy. For example,this diagram shows a much simpler map whose solution is quite straightforward. If you imagine the streets as edges of a cube, it's clear that two ice-cream vans at diagonally opposite cube vertices are sufficient. Moreover, you should be able to convince yourself that it is not possible to solve the problem with fewer than two vans. It is much harder—though not impossible—to convince oneself that Tourist Town cannot be serviced by less than six vans. For general maps it is extremely hard to prove that a certain configuration of ice-cream vans is a minimal one.



What's it all about?

One of the interesting things about the ice-cream problem is that *no-one knows* whether there is an algorithm for finding a minimum set of locations that is significantly faster than the brute-force method! The time taken by the brute-force method grows exponentially with the number of intersections—it is called an *exponential-time* algorithm. A *polynomial-time* algorithm is one whose running time grows with the square, or the cube, or the seventeenth power, or any other power, of the number of intersections. A polynomial-time algorithm will always be faster for sufficiently large maps—even (say) a seventeenth-power algorithm—since an exponentially-growing function outweighs any polynomially-growing one once its argument becomes large enough. (For example, if you work it out, whenever n is bigger than 117 then n^{17} is smaller than 2^n). Is there a polynomial-time algorithm for finding the minimum set of locations?—no-one knows, although people have tried very hard to find one. And the same is true for the seemingly easier task of checking whether a particular set of locations is minimal: the brute-force algorithm of trying all possibilities for smaller sets of locations is exponential in the number of intersections, and polynomial-time algorithms have neither been discovered nor proved not to exist.

Does this remind you of map coloring (Activity 13)? It should. The ice-cream van question, which is officially called the "minimum dominating set" problem, is one of a large number—thousands—of problems for which it is not known whether polynomial-time algorithms exist, in domains ranging from logic, through jigsaw-like arrangement problems to map coloring, finding optimal routes on maps, and scheduling processes. Astonishingly, all of these problems have been shown to be equivalent in the sense that if a polynomial-time algorithm is found for one of them, it can be converted into a polynomial-time algorithm for all the others—you might say that they stand or fall together.

These problems are called *NP-complete*. NP stands for "non-deterministic polynomial." This jargon means that the problem could be solved in a reasonable amount of time if you had a computer that could try out an arbitrarily large number of solutions at once. You may think this is a pretty unrealistic assumption, and indeed it is. It's not possible to build this kind of computer, since it would have to be arbitrarily large! However, the concept of such a machine is important in principle, because it appears that NP-complete problems cannot be solved in a reasonable amount of time without a non-deterministic computer.

Furthermore, this group of problems is called *complete* because although the problems seem very different—for example, map-coloring is very different from placing ice-cream vans—it turns out that if an efficient way of solving one of them is found, then that method can be adapted to solve *any* of the problems. That's what we meant by "standing or falling together."

There are thousands of NP-complete problems, and researchers have been attacking them, looking for efficient solutions, for several decades without success. If an efficient solution had been discovered for just one of them, then we would have efficient solutions for them all. For this reason, it is strongly suspected that there is no efficient solution. But proving that the problems necessarily take exponential time is the most outstanding open question in theoretical computer science—possibly in all of mathematics—today.

Further reading

Harel's book *Algorithmics* introduces several NP-complete problems and discusses the question of whether polynomial-time algorithms exist. Dewdney's *Turing Omnibus* also discusses NP-completeness. The standard computer science text on the subject is Garey & Johnson's *Computers and Intractability*, which introduces several hundred NP-complete problems along with techniques for proving NP-completeness. However, it is fairly heavy going and is really only suitable for the computer science specialist.