The science of route planning

How does your sat-nav work out the best route?

Sat-navs (or satellite-navigation systems) are a fascinating offshoot of both the US military's desire to equip its units with the facility to find out their position and the ability to display maps on a computer screen. But though the devices are extremely popular, the algorithms that they use in order to provide the shortest distance route to your destination are less well known.

There are two major algorithms that come into play with sat-navs. The first is perhaps the simplest: the ability of the unit to use the GPS (global positioning system) satellites to work out where in the world the unit is situated. The second is rather more complicated: the ability of the sat-nav to determine the shortest distance from point A – where you are – to point B – where you'd like to be. There are other algorithms in play, mostly dealing with the visual display of the route to take, but these two algorithms are the most important.

The GPS satellite network started out as a US Air Force system to help determine the position of any receiver to an accuracy of 15 metres. Following the downing of the civilian KAL 007 airliner after it drifted into Soviet restricted airspace, Ronald Reagan promised to make the system available for civilian use once it became operational. This happened in late 1993, and the unencrypted civilian signal was deliberately downgraded so that GPS units would only be accurate to about 100m. This limitation (known as Selective Availability) was removed in 2000.

The positioning algorithm is fairly simple. Currently there are 30 satellites spread out in ▸

Path calculation

In order to calculate the best route, a common optimisation is to store the parent node for each node visited. After the search for the target node is complete, it's pretty simple to use a stack to calculate the path in reverse order. You would push each node on to the stack, and then follow the parent link to the next node. Once you reach the starting node again, you can now pop off the nodes from the stack to produce the path. ■

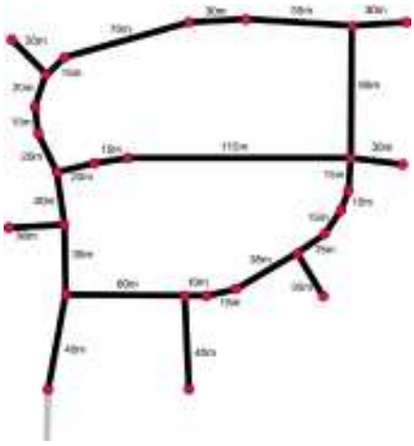
medium Earth orbit, each transmitting the same information. The messages consist of three main pieces of data: the exact time the message was transmitted, precise orbital data for the satellite (known as the ephemeris) and the overall system health. The GPS unit listens for these messages and interprets them. From the time of the message, the GPS unit can work out how long the message took to reach the unit, and, from that and the known speed of light, how far away the satellite is. From the ephemeris data, the unit can work out the direction to the satellite.

Calculating the position

Using just the messages from one satellite, the GPS unit is going to be somewhere on the surface of a virtual sphere centred on that satellite. That's fairly interesting to know, but not very helpful. So the GPS unit listens for messages from other satellites.

Using the messages from two satellites, the GPS unit can work out its position to be somewhere on the circle that forms the intersection of the two spheres centred on each satellite. If you think about it geometrically, either the spheres don't intersect at all, or they intersect at just one point (the spheres just manage to 'touch'), or, in the more general case, they intersect as a circle. Think of soap bubbles joined together. Interesting to know, but again pretty useless.

Using three satellites, the GPS can calculate its position to be at one of two points on that circle from the previous case. Again, thinking geometrically, the intersection between a



▲ **Figure 1: The Pinewoods estate as a vector map or graph, with the exit road marked in grey.**

▲ **No matter who made your sat-nav, it hooks up to the same satellites and works in the same way.**

sphere and a circle is going to be either non-existent, a single point or two points.

So GPS units use the messages from at least four different satellites to resolve their location to a single point. GPS satellites are positioned in orbit so that from any point of Earth about 10 satellites will always be 'visible' in the sky, an ample number from which to calculate the position of a GPS receiver.

The position of the receiver, using just the GPS satellites, is calculated to within about 15m. The reason for the comparatively large error is due to many factors, including the atmosphere (light travels slightly slower in air than in the vacuum of space), any errors in the clocks involved, bouncing of the GPS signals off buildings and so on. Yet, as we all know, a GPS unit seems to be way more accurate than that. The reason is that terrestrial sat-navs do not rely exclusively on GPS satellite signals: they also make use of other signals such as those from mobile phone towers and the like to refine their location to within a few metres.

If the GPS receiver is in a car and forms part of a sat-nav system, the unit will also make use of further information from the car itself, such as speed, distance travelled, acceleration and so on. This helps in urban environments where the GPS signal can be blocked by bridges, or by being in a tunnel, or messed up by being reflected off buildings and the like. Of course, a further refinement is that, usually, the car is on a road, and so the sat-nav can 'fix' the location of the car to a road on its internal map.

Calculating the route

It's now time for the second algorithm: determining the optimal route from the current position (defined as a point with some latitude and longitude values) to a destination (defined in the same way). An immediate prerequisite is the map.

The map in a sat-nav is a peculiar beast. It's first and foremost a vector map – it consists of a set of data that is primarily in vector format. Since it comprises a set of vectors, the display part of the sat-nav unit must draw the vectors (that is, the roads, mostly) on the screen at runtime, and take account of the orientation and position of the car at that moment.

Compare this with the mosaic maps (or tiled maps) used by mapping websites like Google Maps. Here the map is served up as a precalculated set of square images (tiles) which are then painted onto the screen by the browser. The tiles sent to the browser are those needed for the current resolution of the map and for the area covered by the browser's window. The tiles are always oriented in the same manner, with north to the top.

Think of the vectors in the vector map in a sat-nav as straight road segments. They consist of two positions (beginning and end) defined by latitude and longitude, and will have further information tagged with them, such as the name of the street or road and whether the road is a major thoroughfare such as a motorway or A road or a smaller one like a backstreet or one-track lane. A road then consists of several segments placed end to end.

Now the fun starts. We have two points of interest: the current location and the destination. We have a map that's defined as a large set of vectors. We have to use the latter to calculate the most optimal route to go from the current location to the destination.

The usual algorithm used is Dijkstra's algorithm for finding the least-cost path between two nodes in a weighted graph or

Spotlight on… Sat-nav issues

As sat-navs have become more sophisticated, their use has become more controversial.

The first controversy is their use while driving. Put simply, just like mobile phones, entering information into a sat-nav is a big distraction. Drivers should pull over before updating their destination, or have a passenger do it. Some built-in sat-nav systems even go so far as to lock out certain features while the car is moving to avoid this situation.

The second controversy is the accuracy of the maps themselves. There have been numerous stories in the media of drivers uncritically believing the routes described by their sat-nav and getting stuck in narrow lanes, of trying to ford rivers, of following roads that turn into farm tracks, or even ignoring warning road signs and crossing train tracks as a train comes by. The issue is partly errors in the sat-nav vector map and partly lack of driver common sense.

Finally, as sat-navs have become more and more popular, the portable ones have become targets for thieves if they've been left in cars. ■

Finding phones

In November 2004, Qualcomm successfully tested the first assisted GPS unit for mobile phones. This system was created to satisfy a US Federal requirement that the emergency services need to be able to locate a mobile phone that is connected to the network. For this, Qualcomm not only made use of standard GPS, they also made use of signals being sent from mobile phone towers. Since the locations of the towers could be predetermined accurately (and more importantly wouldn't change), the phone and the network could cooperate to make refinements to the phone's raw GPS location and narrow the position down to within 5m or so. ■

network. To recap, a graph is a data structure that consists of a set of vertexes (sometimes known as nodes) with links (usually known as edges) between them. A weighted graph has a cost associated with each edge.

If you think about it, a vector map is such a graph. The vectors are edges, and the ends of each vector can be thought of as the nodes. Since a road will consist of a set of vectors joined end to end, you can visualise that most nodes in the map just have two edges. Without any real loss of generality, we'll assume that the starting and ending locations are defined by latitude/longitude pairs and happen to be the ends of vectors (that is, we don't want to get into the geometric algorithm that determines inside which vector a particular point can be found). Figure 1 shows a simplified example of a vector map of the Pinewoods estate where I live (the lower left node, coloured grey, continues out of the estate). I've marked the lengths of each vector, delimited by the red nodes, to the nearest 5m.

The cost of each vector is a function of the way it has been tagged and also of its length. We can imagine that motorways and A roads have a low 'cost' value whereas streets and C roads have a high 'cost'. The length of a vector also has a cost, one that is proportional to the length of the vector.

Dijkstra's algorithm is intended to calculate what's known as the shortest path tree from a given node, or in other words, to reorganise the nodes in the network into a tree such that, if you follow the links from the initial node (the root of the tree) to any other, you will traverse the path with the smallest cost.

We're not particularly interested in the full tree though, since the number of vertexes and edges in any sat-nav city map is going to be huge, and, in planning the route, the sat-nav will be (or rather, should be) ignoring the vast majority. Dijkstra's algorithm also fails in the sense that all nodes are equally 'important', whereas we know that the nodes in the direction we want to travel are going to be more important than others.

What we need is a directed version of Dijkstra's algorithm, one that takes into account the fact that the sat-nav map database is, well, a map. Certain vectors in the map just won't be considered; they're in the opposite direction, for example. Somehow we need to encode nodes that are closer to the target as being more significant than nodes that are further away.

▲ **Handheld GPS devices generally do not provide route-finding functionality, but their location calculations are much more accurate.**

The A* algorithm

The algorithm used is called the A* algorithm (pronounced 'A star'). It is a graph algorithm that finds the path with the smallest cost from a starting node to the target node. It was devised by Hart, Nilsson and Raphael in 1968.

A* uses two values when considering a node to be added to the smallest cost path: the actual smallest cost of reaching the node from the start node – usually called g – and a heuristic estimate of the distance from the node to the target node, usually called h. The simplest heuristic we can use is a simple 'as the crow flies' distance to the target, taking no account of any roads, corners or whatnot. A further value called f is calculated as $g + h$. (Let's note in passing that Dijkstra's algorithm is equivalent to the A* algorithm with the heuristic h set to 0 for every node.)

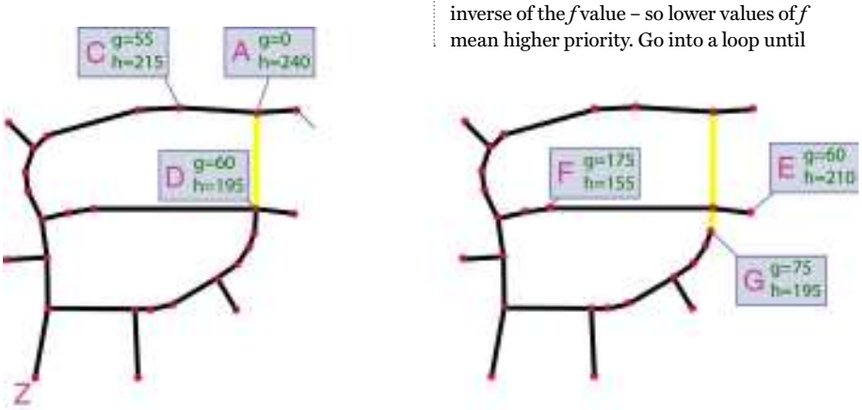
This is how it works. (Follow along with Figure 2 as we find the path from A to Z on my vector map.) We set up the path to be empty. For the source node, g is 0, and h is the direct 'crow' distance to the target node. Add it to a priority queue, which is a data structure that releases nodes with the highest priority first. Our priority here is going to the inverse of the f value – so lower values of f mean higher priority. Go into a loop until the priority queue is empty or we've found the target node (the more preferable outcome).

Each time round the loop, remove from the priority queue the node with the smallest f value. Add it to the current path. Now search through all the nodes that can be reached from this node (that is, find all the vectors that have the current node as an end point). Ignore all nodes that are in the current path. For each of these reachable nodes, calculate their possible g value (it will be the current node's g value plus the edge cost to the other node). If the node has been 'seen' before (that is, it's somewhere in the priority queue), this possible g value may be smaller than the previous one, in which case we can replace it. Calculate the h value and therefore the f value, and, if the node is not in the priority queue, add it.

Of course, if the node from the priority queue is the target, we add it to the current path and stop as we've solved the problem.

If we run out of nodes in the priority queue, it means that the target node is not reachable from the start node. We've failed to find a path. Of course, in the real world, we wouldn't expect this to happen (unless we were trying to get from London to the Isle of Man with a sat-nav that doesn't 'know' about ferries).

After we have the shortest route (or, rather, the route with the smallest cost), the software in the sat-nav unit then has to display it on the screen, either as a 2D map or, more commonly nowadays, as a 3D projection of the map, and then update it as you travel along. But that, as they say, is an article for another day. PCP



▲ **Figure 2: Here are the first few steps taken by the A* algorithm as we find the route out of the estate.**

Julian M Bucknall has worked for companies ranging from TurboPower to Microsoft and is now CTO for Developer Express.
feedback@pcplus.co.uk