(12) **UK Patent Application** (19) **GB** (11) **2 443 472** (13) **A**

(43) Date of A Publication   07.05.2008

(21) Application No: 0621508.1

(22) Date of Filing: 30.10.2006

(71) Applicant(s):
    Cotares Limited
    (Incorporated in the United Kingdom)
    67 Narrow Lane, HISTON, Cambridgeshire,
    CB24 9YP, United Kingdom

(72) Inventor(s):
    Alan Henry Jones

(74) Agent and/or Address for Service:
    Marks & Clerk
    4220 Nash Court,
    Oxford Business Park South, OXFORD,
    OX4 2RU, United Kingdom

(51) INT CL:
    G06Q 10/00 (2006.01)   **G01C 21/34** (2006.01)

(56) Documents Cited:
    EP 1335315 A2

(58) Field of Search:
    INT CL **G06Q**
    Other: **WPI,EPODOC**

(54) Abstract Title: **Method of generating routes**

(57)   A method of generating a plurality of diverse
        routes from a source to a destination in a
        weighted directed graph. Such a method may be
        used for route planning or navigation with the
        weighted directed graph representing a road
        network, but may also be used in other
        applications. A source routeing tree is generated
        from the source to a first set of points of the
        graph, which may comprise some or all of the
        points. A destination routeing tree is generated
        from some or all of the points of the graph to the
        destination. The trees are then combined to
        form the routes. For example, the sub-routes
        common to and traversed in the same direction
        by the source and destination trees may be
        selected. The sub routes may then be formed
        into the routes by extending each sub-route as
        necessary to the source and destination along
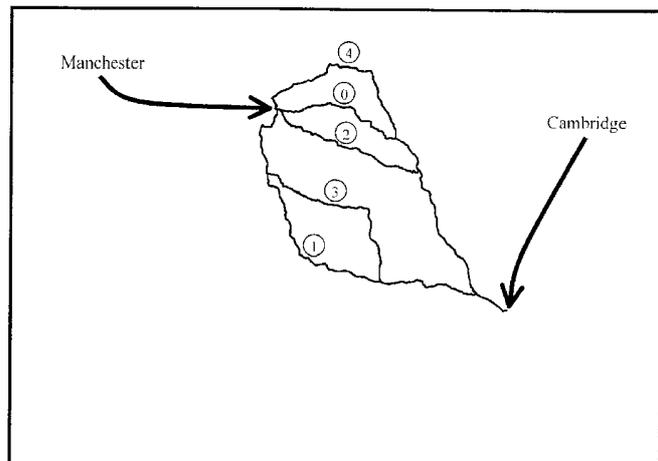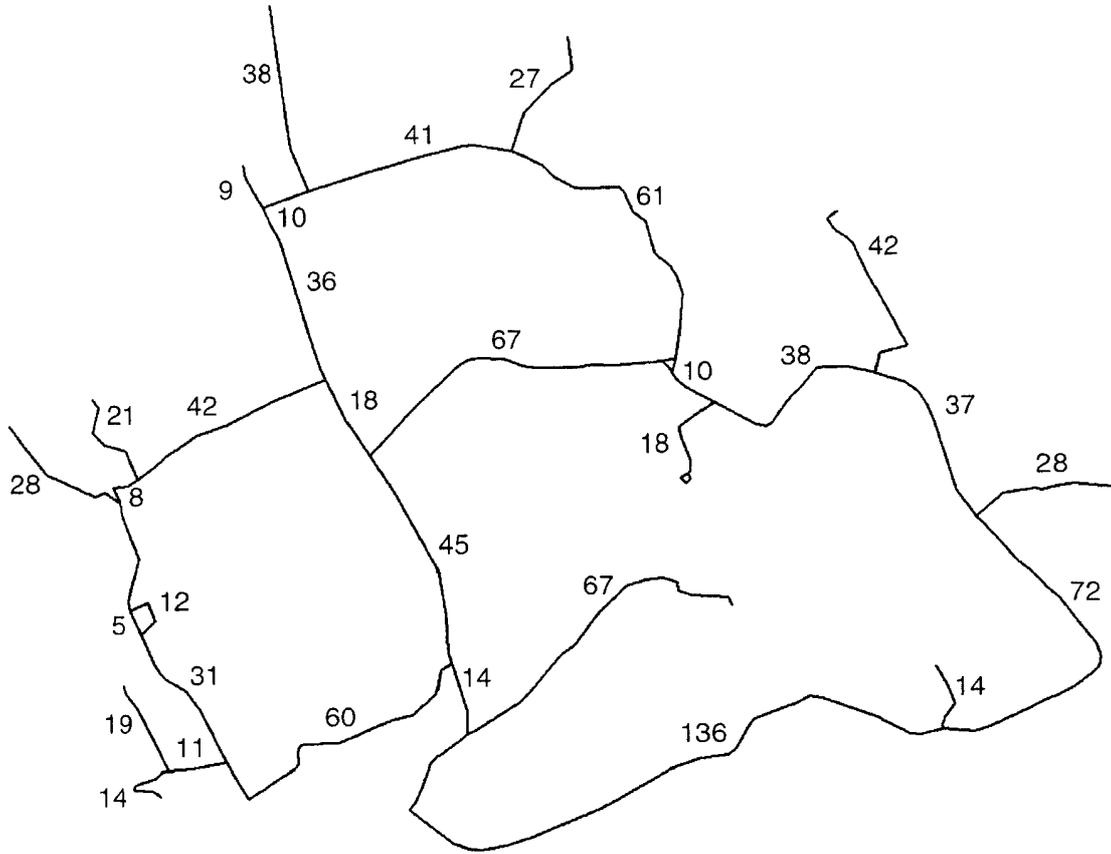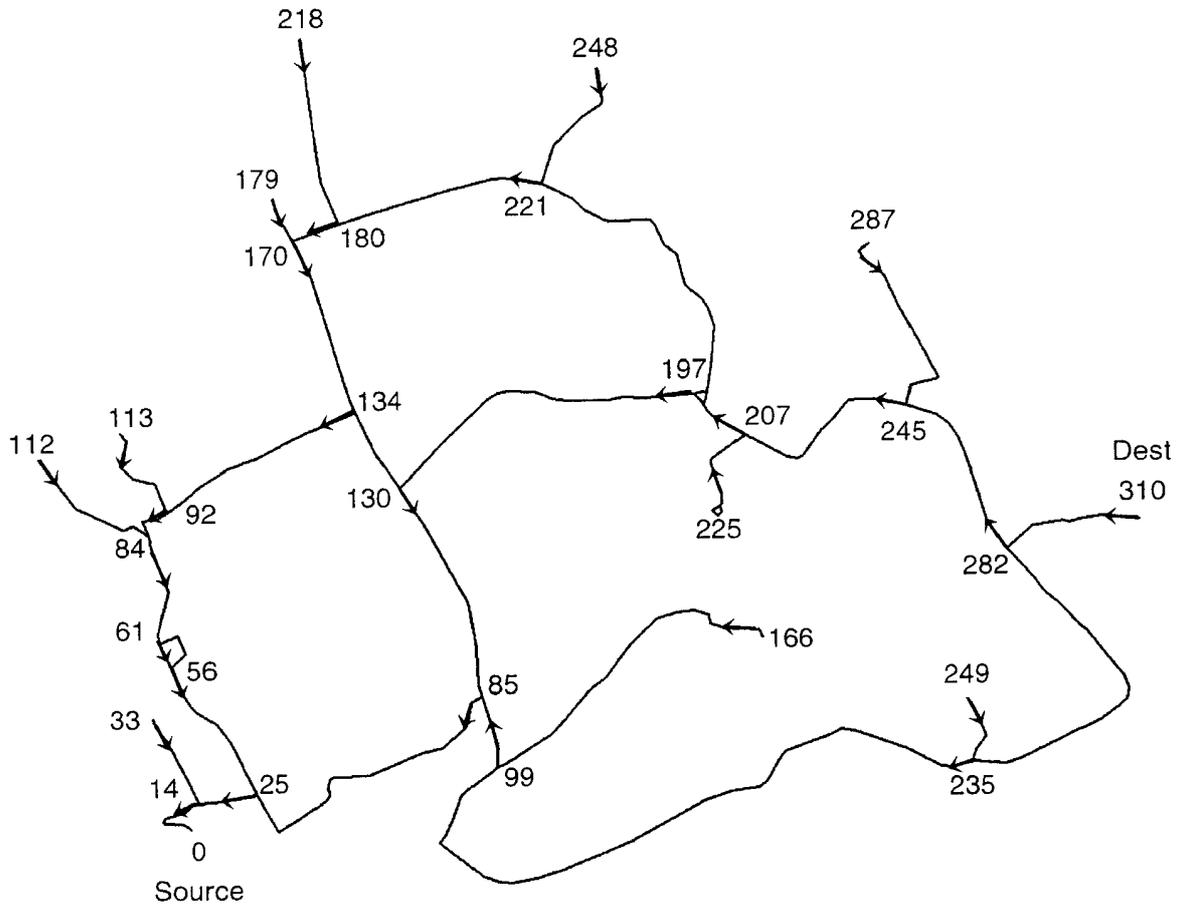   the source and destination trees.

Figure 10 Plateau routes

GB 2 443 472 A

Figure 1. Distances

Figure 2. Source Tree

253

201

234

174

225   215

107

198

113

103

65

Dest
0

261

276

240

180

103

121

28

248

271

276

303

114

315

225

285   225

100

296

236

310
Source

114

100
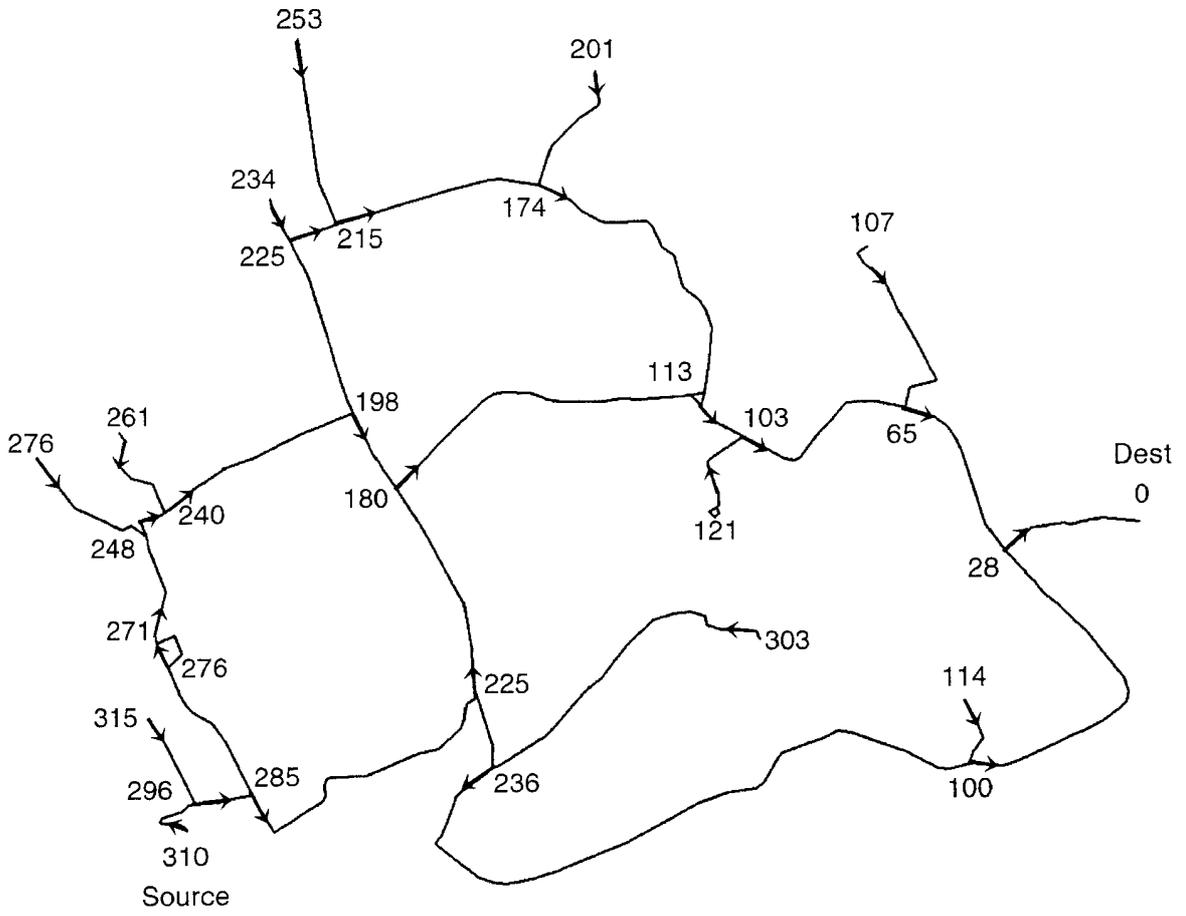
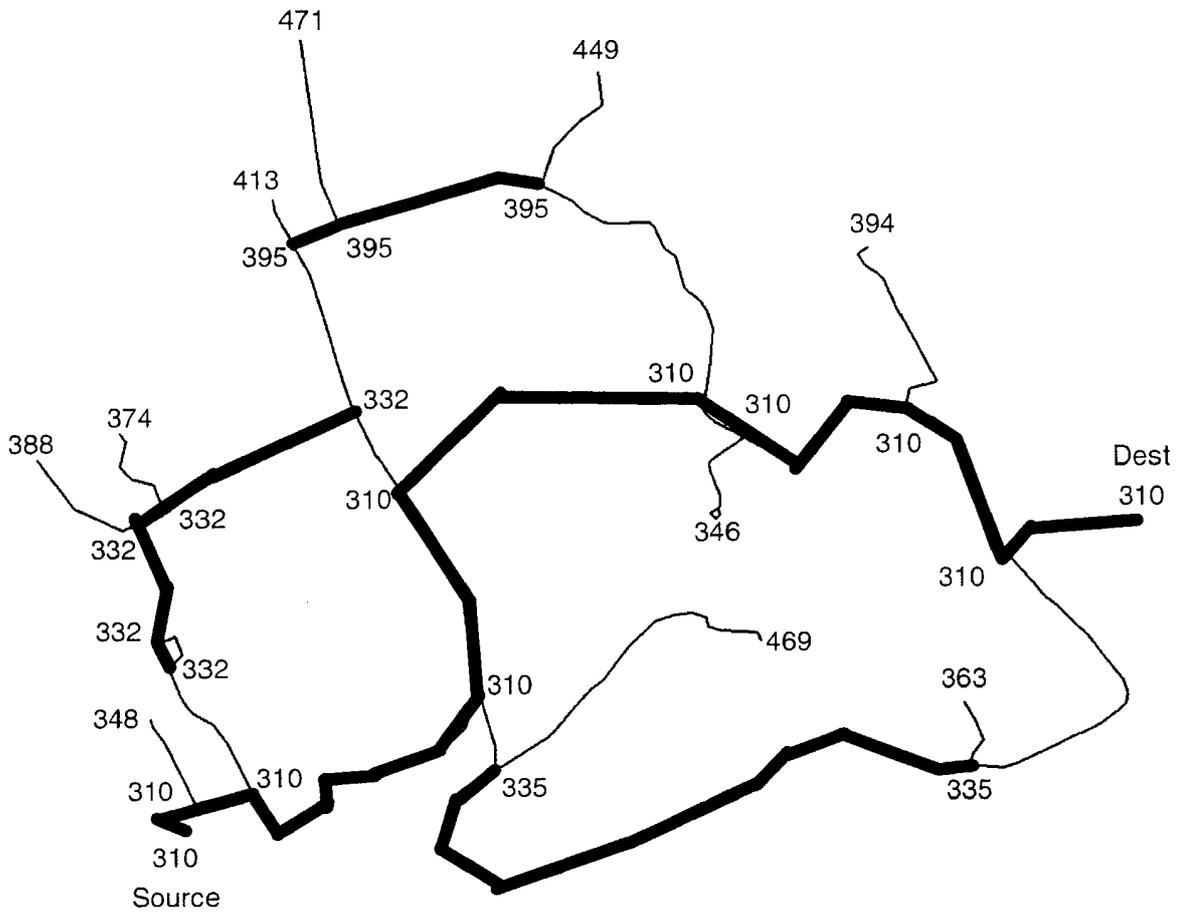**Figure 3. Destination Tree**
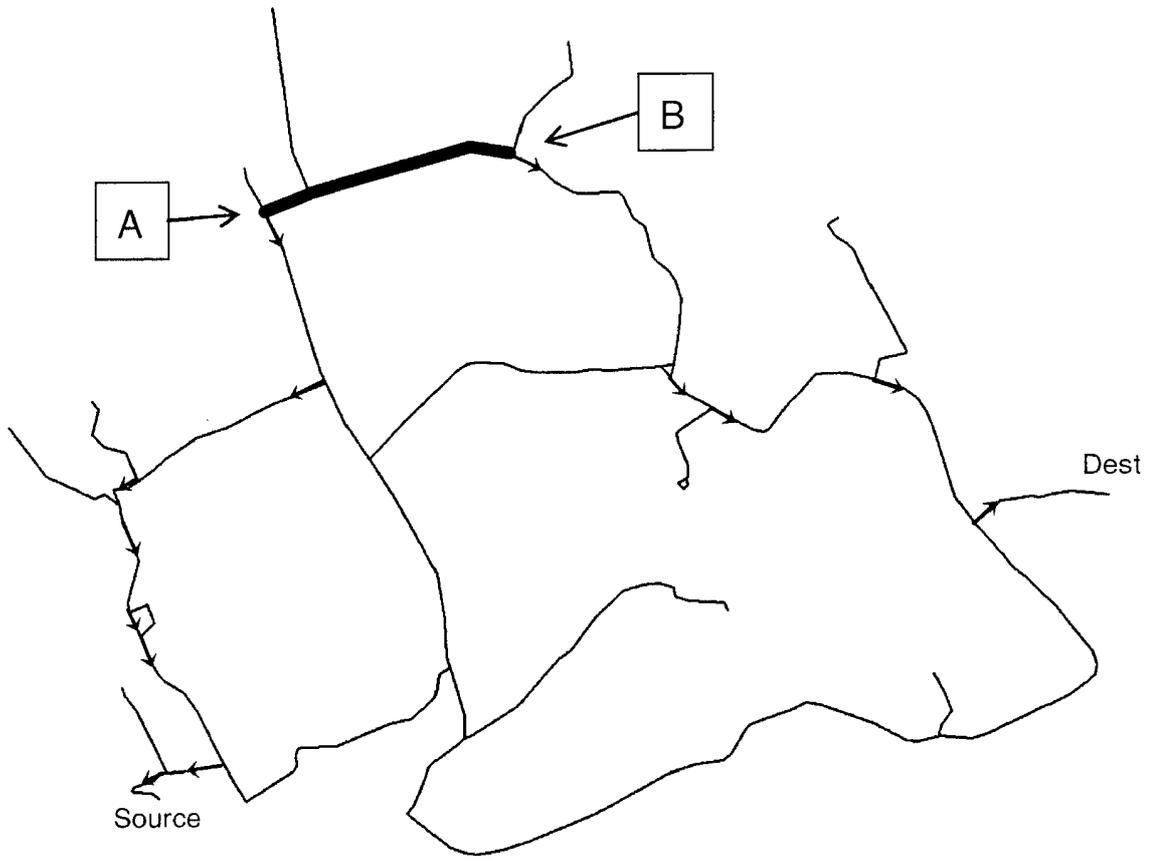
Figure 4. Combined Tree

Figure 5. Plateaux

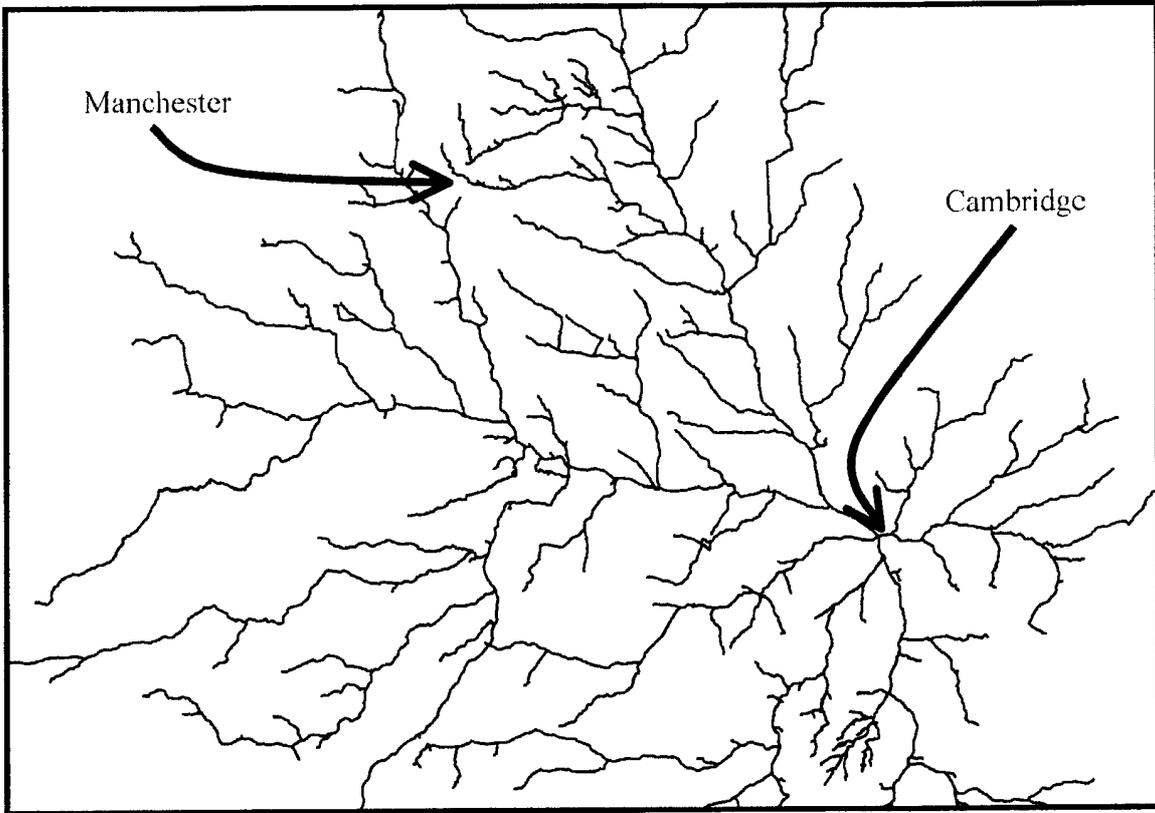**Figure 6. Plateau Goodness**

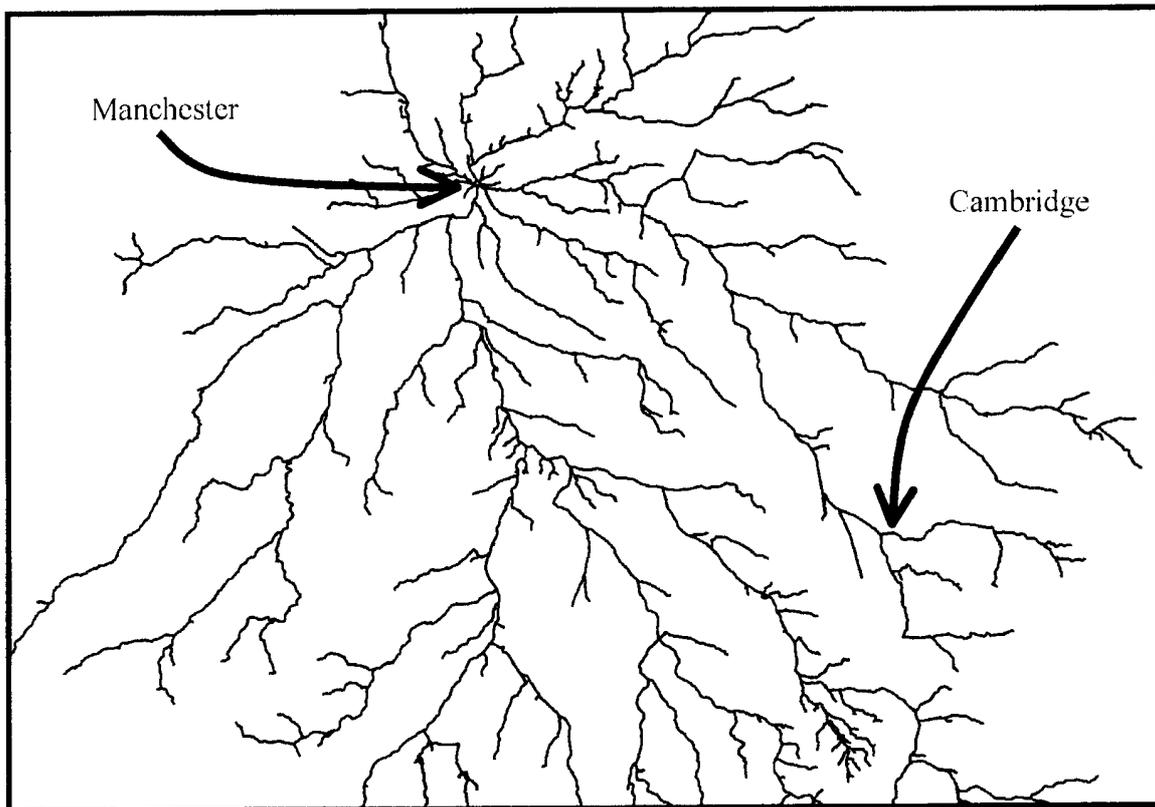**Figure 7 Source tree from Cambridge**

Figure 8 Destination tree to Manchester
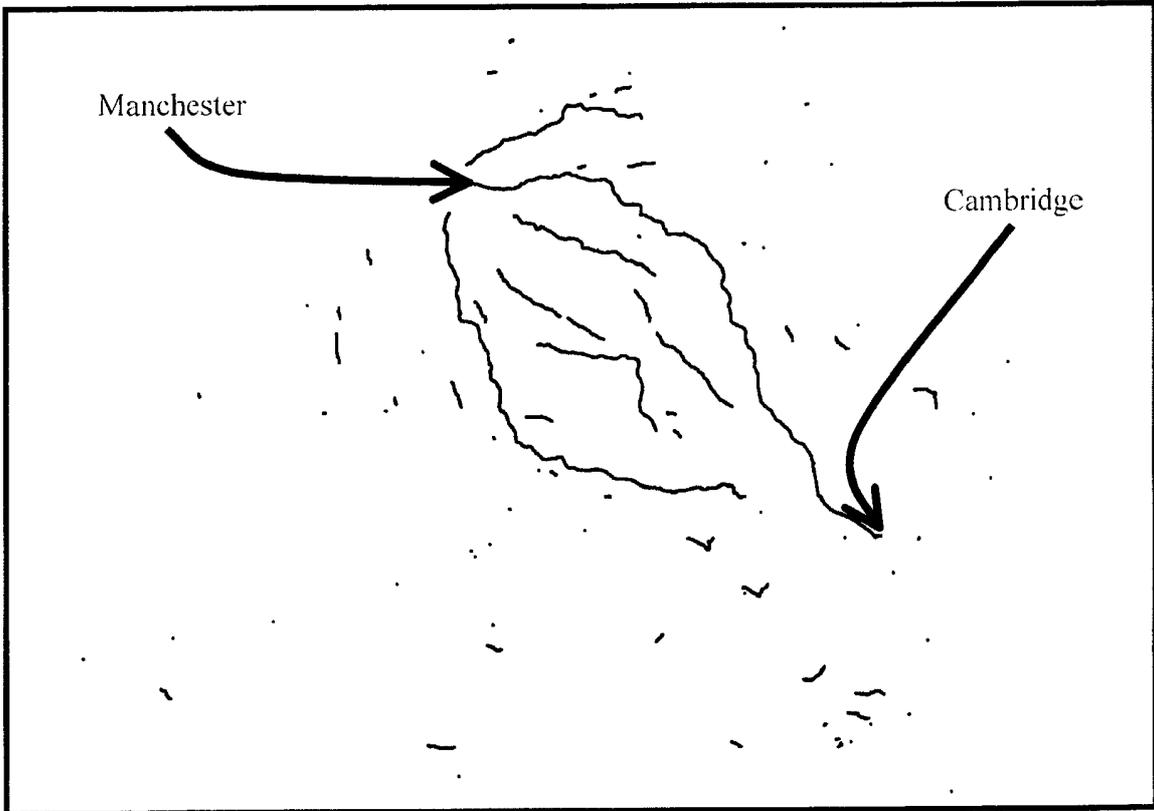
Figure 9 Combination tree

Figure 10 Plateau routes

Optimal routes
from A to all nodes

Fig 11

Optimal routes to
B from all nodes

Fig 12

Locally optimal
routes from A to B
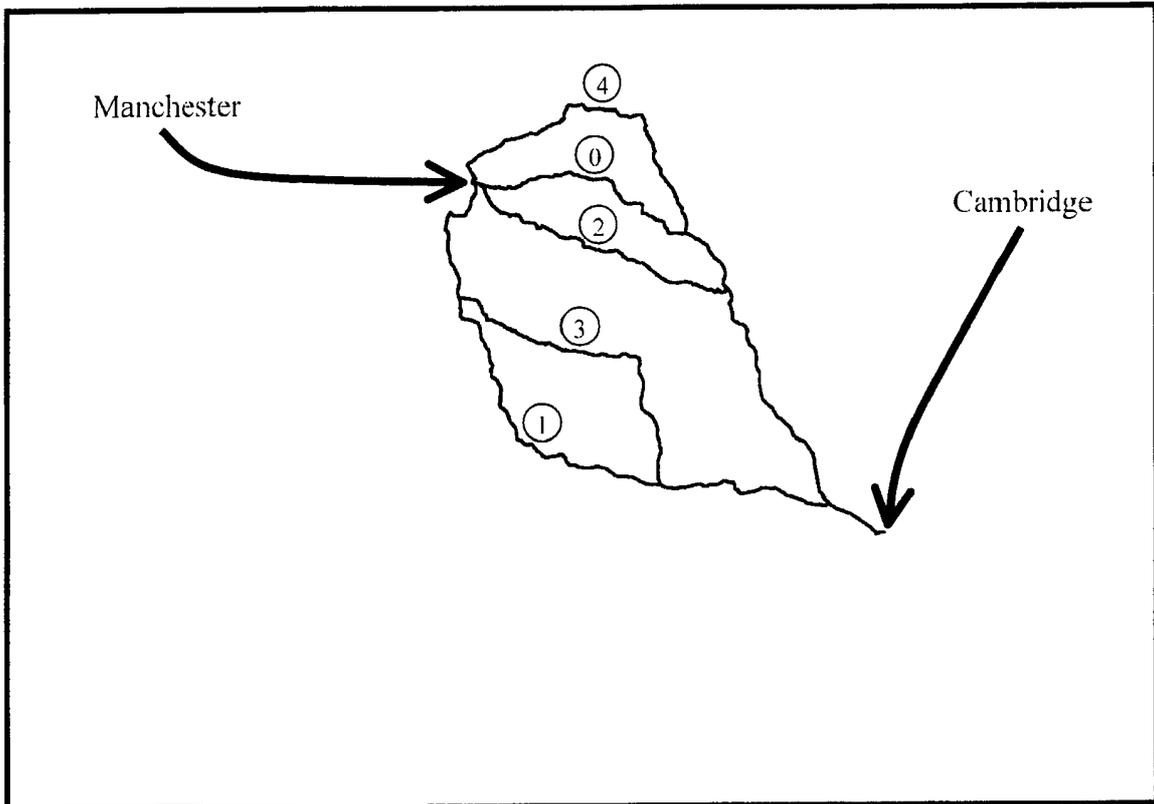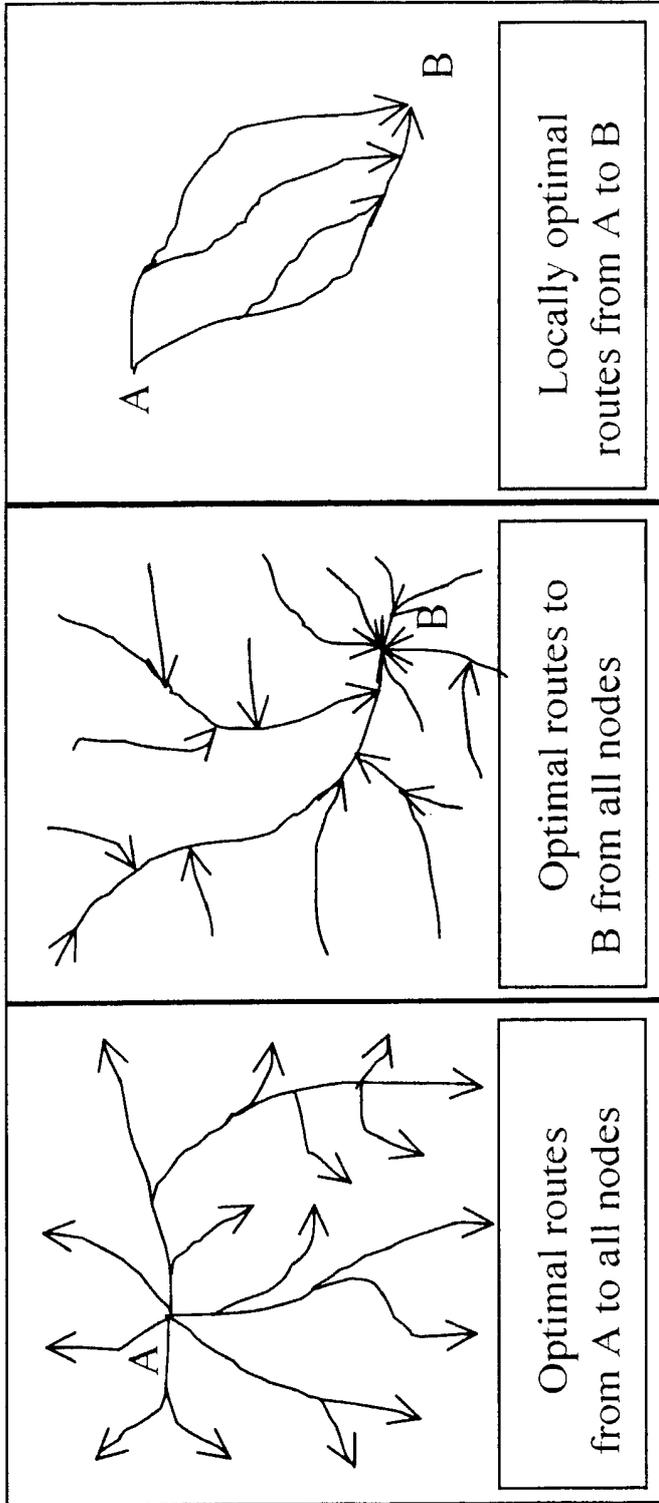
Fig 13

Sort by:	familiarity	motorway	time	distance	tolls

— · — · —	A14/A1/A606/A6	3hrs 13min	154mls	£4.00
— · — · — · ·	A14/A1/A57/A628	3hrs 21min	162mls	£4.00
— — — — — ·	A14/M1/A6	3hrs 20min	172mls	£10.00
——————→	A14/M6 toll/M6	3hrs 01min	178mls	£14.00
— — — — -	A14/M6	3hrs 06min	180mls	£10.00

Fig 14

Turn left, 1ˢᵗ exit, Manchester

M1/A6
+ 15min
£6

M6
2hrs
£10

Fig 15

NextLinks

PrevLinks

Fig 16

Figure 1 7

Method of and Apparatus for Generating Routes

The present invention relates to a method of and an apparatus for generating a plurality of diverse routes in a weighted directed graph.

5

The term "diverse routes" as used herein means routes which share the same links over less than a predetermined proportion of their lengths, typically over less than 85% of their lengths.

10   A graph G=<V,E> consists of a set of *vertices* (also known as points or nodes) V and a set of *edges* (also known as arcs or links) E. An edge connects two vertices u and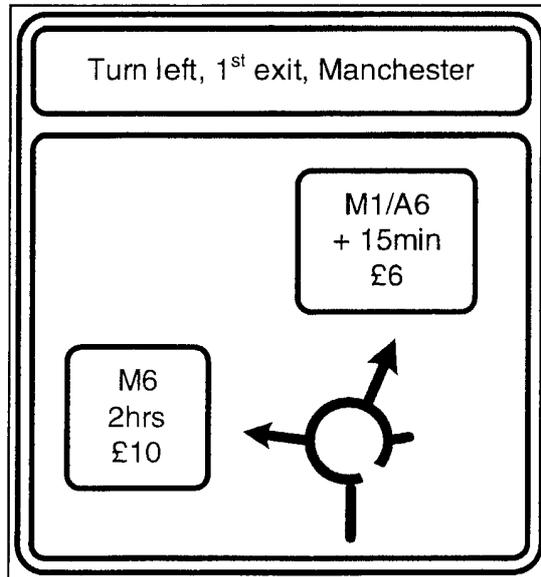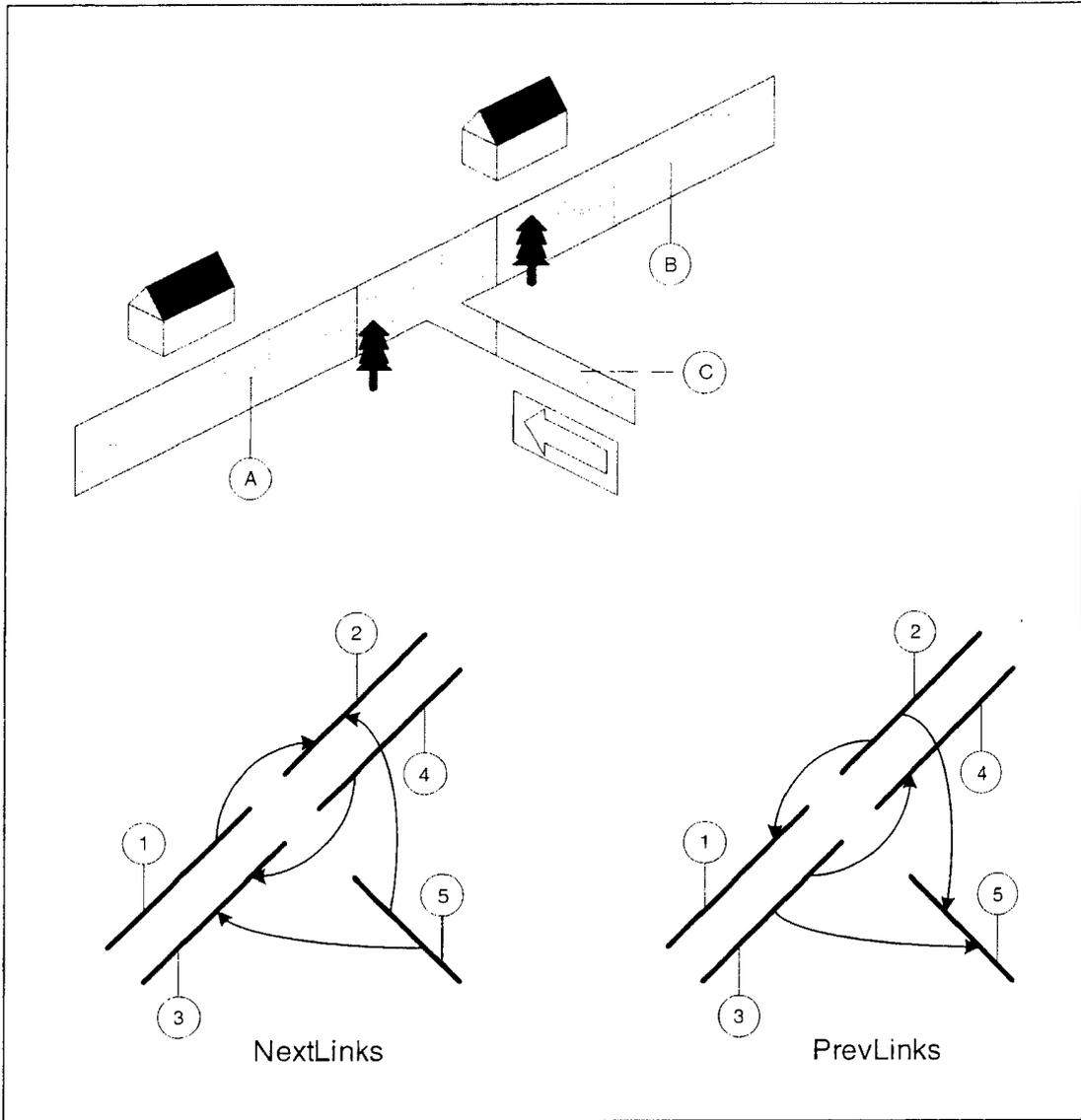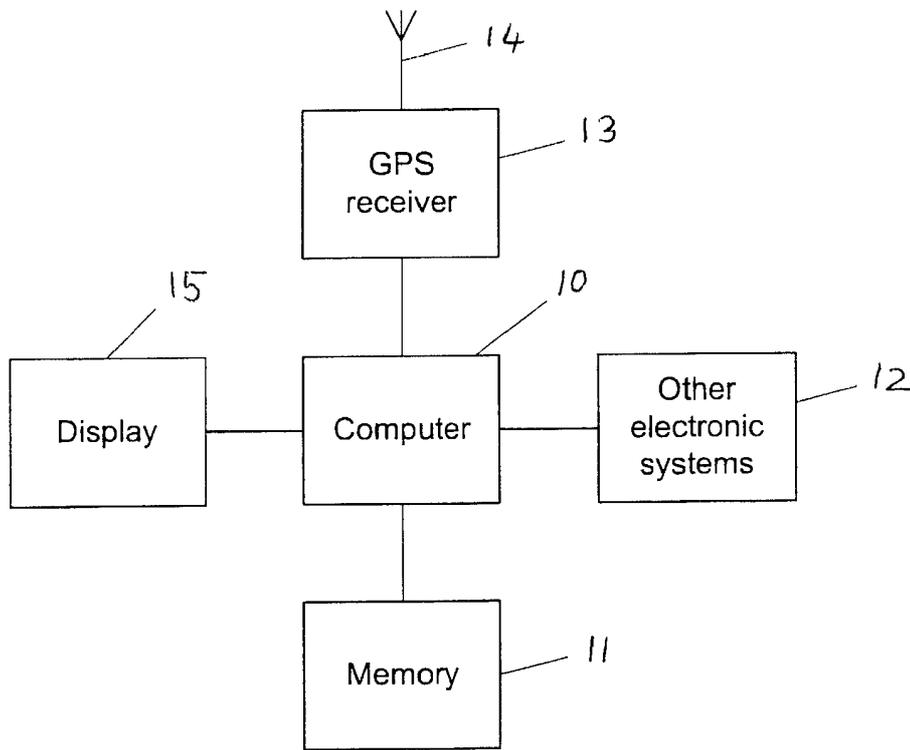 v; v is said to be *adjacent* u. In a *directed* graph, each edge has a sense of direction from u to v and is written as an ordered pair <u,v> or u->v. In an *undirected* graph, an edge has no sense of direction and is written as an unordered pair {u,v} or u<->v. An undirected graph can be

15   represented by a directed graph if every undirected edge {u,v} is represented by two directed edges <u,v> and <v,u>.

Both directed and undirected graphs may be *weighted*. A weight is attached to each edge. This may be used, for example, to represent the distance between two cities, the driving

20   time, the cost of the journey, the resistance of an electrical path or some other quantity associated with the edge. The weight is sometimes called the *length* or *cost* of the edge, particularly when the graph represents a map of some kind. The weight or length of a path or a cycle is the sum of the weights or lengths of its component edges.

25   The methods described in this application are generally applicable to any domain where the costs can be described by a weighted directed graph, and where there are no cycles of negative cost in the graph. We will be using routeing on a road network in many of our examples, but it should be understood that the techniques can equally well be applied in other domains such as routeing of packets around a network of switching points, or the

30   finding of paths for wiring in an integrated circuit, printed circuit board or a building.

Road route planners are designed to find the optimum route from source A to destination B. They define this as the single route with the minimum cost, where the cost function is a simple weighted sum of times, junction delays, distances, financial costs, type of road of the links that make up the route. The methods are often called "shortest-path" where this is

5   understood to mean shortest in the sum of the costs, whatever they represent, not necessarily in metres. For a given cost function and road graph, there exists a lowest cost going from A to B, and this be found by using well known algorithms such as Dijkstra's algorithm. In a degenerate case, there may be a plurality of routes which share this globally lowest cost but the algorithms will find just one of them. Dijkstra's algorithm

10   actually finds the optimum route from A to all nodes, but variations on it are designed to terminate early and to explore preferentially in the direction of B. The main constraint on the graph is that there should be no cycles with a negative overall cost, which is simple enough to satisfy for a road network where all the costs are positive.

15   Different route planners have slightly different road databases and cost functions. They can come up with equally plausible routes from A to B, sometimes very different from each other. It would be useful for a user to see all such plausible routes, particularly the ones that are very different from each other.

20   It is also common for users to find that the recommended route is not the one that they would have guessed or preferred, and so they try to make the route planner choose their route so that they can compare the time and distance. This is achieved by adding compulsory stops along the way, or by designating road links, junctions or areas that the router is to avoid, or by changing the weightings of the cost function to favour quicker,

25   shorter, more motorway routes or other criteria. This can be quite time consuming, and one is often left wondering if there are more good routes that other people would have tried.

As an example, consider travelling from Cambridge to Manchester. One route finder

30   suggests using the A14, then M6, M56. Another uses the M6 toll. Yet another recommends the A14 followed by the A1. These are all amongst the routes that a good human planner would choose, but how would we find the others?

The basis of most single shortest path routers is some variation of **Dijkstra's algorithm**.

A good original reference to this algorithm is:

E.W. Dijkstra. *A note on two problems in connexion with graphs*. Numerische Matematik

5   1:269-271, 1959.

These algorithms maintain the lowest cost found so far to reach any node from the source. They repeatedly select an active node (one whose outgoing links still need to be explored) with a low cost (based on different heuristics, e.g. the A* algorithm), and explore outgoing

10   edges to see if they can reach another node at lower cost than so far recorded.

A good original reference to this A* algorithm is:

P. E. Hart, N. J. Nilsson and B. Raphael. *A formal basis for heuristic determination of minimum path cost*. IEEE Transactions on Systems Science and Cybernetics, 4:100-107,

15   1968.

If a node can be reached with a lower cost than that known so far, then each such node is added to the list of active nodes. Once they terminate, these algorithms have in fact found the lowest cost routes to a whole group of nodes (those with lower cost than the

20   destination), and have effectively computed a routeing tree (**RT**) from source to all nodes that might be useful in getting to the destination. Whatever clever pruning is performed on this tree, e.g. A* where the tree terminates in a roughly elliptical shape with foci at the source and the destination, our methods can be used to convert it into a diverse alternative route finding algorithm.

25

There is a class of algorithms designed to find the K shortest paths from source to destination. Examples are Yen's (Kth shortest paths with no constraints) and Suurballe's (shortest disjoint path pair). (J. Y. Yen. *Another algorithm for finding the K shortest loopless network paths*. In Proc. of 41st Mtg. Operations Research Society of America,

30   volume 20, 1972; J. W. Suurballe, R. E. Tarjan. *A Quick Method for Finding Shortest Pairs of Disjoint Paths*. Networks, 14:325336, pp 325-336, 1984.)

Yen's algorithm is not useful for finding good alternative routes, as the Kth shortest paths will simply be minor variations on the most optimum route. It works by repeatedly running Dijkstra's algorithm on a graph where edges and nodes are prohibited, by effectively setting their cost to infinity. In a road network, this is a non-starter for finding

5   truly diverse routes as there is no way of knowing if the prohibited node or link is vital to one of the good alternative routes.

Suurballe's algorithm and others are designed to find edge- or node-disjoint paths. These are unhelpful for finding good alternative routes because some road segments may be

10   shared between several of the diverse alternative routes that we would like to find, which are therefore not disjoint. They work by running Dijkstra's algorithm to find a primary route, then altering all of the link costs, and running Dijkstra's algorithm again to find a backup route.

15   For road route finding, we may not need such a mathematically rigorous definition of the K routes.

For example, we can find successive routes by weighting against road links that have been used in earlier routes. Unfortunately, such routes become progressively more baroque as

20   more links are weighted against, and in many cases, several of the good routes will share a common beginning or ending, which is ruled out too early by this technique.

We can alter the road link costs or the cost function itself in an effort to generate other routes. This can be useful for expressing different user preferences such as a liking for

25   Motorways rather than back roads, but it will not present alternative routes that share these characteristics, such as the choice between using the M6 or the M6 toll around Birmingham.

While driving, some systems can alter the costs of a subset of the road links, and

30   recompute a route. This may be because the user has pressed a button to say that they want a diversion, or because the system has received new traffic information that changes the cost of using some links. In these cases, current systems would make another run of a single routeing algorithm, and then guide the user on the new shortest path (lowest cost)

route. This will usually result in a short diversion around the affected links, back onto the original route. This is a very different technique from ours as it uses new costs for the links, and then looks again for the single most optimum route, not for diverse choices.

5   US patent 6199009 discloses a technique for computing several routes based on the different preference settings (quickest, shortest, combination) and allowing the user to select between them. The problem with this approach is that in most cases the routes will be similar or even identical to each other (e.g. the shortest route may also be the quickest), and a radically different route that is perhaps 5% longer will just not be shown.

10

Known systems guide the driver along one selected route without showing any en-route alternatives. Of course, if the driver strays from the selected route, then many commercial systems will recompute a new route, and US patent 5675492 shows how to precompute some alternatives and even display them. These will almost always be a short diversion

15   back onto the earlier selected route, and are not interpreted as a choice for a completely different route.

According to a first aspect of the invention, there is provided a method of generating a plurality of diverse routes from a source to a destination in a weighted directed graph,

20   comprising the steps of: generating a source routeing tree from the source to a first set of points of the graph; generating a destination routeing tree from a second set of points of the graph to the destination; and combining the source and destination trees to form the routes.

The first set may comprise all of the points of the graph.

25

The second set may comprise all of the points of the graph.

The first and second sets may comprise points of the graph adjacent both the source and the destination.

30

The first and second sets may comprise the same points.

The source and destination trees may be minimum cost trees.

The combining step may comprise selecting each sub-route common to and traversed in the same direction by the source and destination trees. The source and destination trees may include back-pointers and the sub-routes may be selected by finding sequences of adjacent points which are pointed to by back-pointers of both the source and destination trees. As an alternative, the points of the source and destination trees may be associated with costs and the sub-routes may be selected by finding sequences of adjacent points for which the sums of the costs from the source and destination trees are the same. The combining step may further comprise extending each sub-route as necessary to the source and destination along the source and destination trees to form one of the routes.

The method may further comprise assigning a measure of goodness to each route. The measure may be a function of the length of the sub-route and the length of the route. The measure may be a function of the difference between the length of the sub-route and the length of the route.

The method may further comprise selecting only some of the routes in accordance with the measure of goodness. The method may comprise selecting the N routes of highest measure of goodness, where N is a positive integer. The measure may comprise selecting at least some of the routes whose measures of goodness are greater than a threshold.

The method may comprise storing at least one of the routeing trees.

The graph may represent a road network and the routes may be road routes.

The graph may represent an integrated circuit or a printed circuit and the routes may be interconnections.

The graph may represent a wiring installation and the routes may be wiring interconnections.

The method may further comprise choosing the order of placement of the interconnections in accordance with the number of selected routes and/or the measure of goodness. The interconnections with the lowest number of selected routes may be placed first.

5   The graph may represent a communications network and the routes may be communication paths. The network may be an internet.

According to a second aspect of the invention, there is provided a method of navigation, including generating road routes by a method according to the first aspect of the invention.

10

The method may comprise presenting information about the road routes to a user. The method may comprise presenting information about a choice of routes from a point from which a plurality of the routes diverge when the user approaches the point. The information may be displayed in a form representing a road sign.

15

According to a third aspect of the invention, there is provided a computer program for performing a method according to the first or second aspect of the invention.

According to a fourth aspect of the invention, there is provided a computer-readable

20   medium carrying a program according to the third aspect of the invention.

According to a fifth aspect of the invention, there is provided transmission across a communication path of a program according to the third aspect of the invention.

25   According to a sixth aspect of the invention, there is provided a computer programmed to perform a program according to the third aspect of the invention.

According to a seventh aspect of the invention, there is provided a computer containing a program according to the third aspect of the invention.

30

According to an eighth aspect of the invention, there is provided an apparatus arranged to perform a method according to the first or second aspect of the invention.

8

It is thus possible to provide a technique which generates a plurality of diverse routes from a source to a destination in a weighted directed graph. It is necessary only to generate two routeing trees irrespective of how many routes are required. Further, it is not necessary to

5   alter, for example, weighting or cost functions or individual link costs.

The invention will be further described, by way of example, with reference to the accompanying drawings, in which:

10   Figure 1 illustrates a small hypothetical road network;

Figure 2 illustrates a source routeing tree generated by a method constituting an embodiment of the invention;

15   Figure 3 illustrates a destination routeing tree generated by the method;

Figure 4 illustrates the result of combining the routeing trees of Figures 2 and 3;

Figure 5 illustrates the sub-routes or plateaux derived from Figure 4;

20
Figure 6 illustrates an example of a route derived from Figure 4;

Figure 7 illustrates a source routeing tree generated by the same or a similar method constituting an embodiment of the invention;

25
Figure 8 illustrates a destination routeing tree generated by the method;

Figure 9 illustrates the result of combining the routeing trees of Figures 7 and 8;

30   Figure 10 illustrates the diverse routes generated by the method with ranking in accordance with a goodness factor;

Figure 11 illustrates a source routeing tree found by the same or a similar method for a very simple graph;

Figure 12 illustrates a destination tree for the same graph;

5

Figure 13 illustrates the diverse routes generated by combining the routeing trees of Figures 11 and 12;

Figure 14 illustrates an example of information display for displaying information about

10 the generated routes to a user;

Figure 15 illustrates a navigation display which may be presented to a user;

Figure 16 illustrates how a road junction may be analysed within the method; and

15

Figure 17 is a block schematic diagram of an apparatus constituting an embodiment of the invention.

For purposes of initial illustration and explanation, a simple hypothetical example of

20 diverse route generation will first be described. We will use a representation where all we know is the distances between junctions along hypothetical roads. Figure 1 shows a tiny subset of roads with their lengths annotated in tens of metres.

We will compute routes from the end of one road (the source node, subsequently marked as

25 "Source") to the end of another (the destination node, subsequently marked as "Dest"), and find not only the shortest route, but also several other routes that are good diverse alternatives. Of course, on such a restricted network, the alternatives take up many of the other roads, but on a real network with millions of nodes, we would still find only the few best diverse alternatives amongst the billions of possibilities.

30

Our subsequent detailed implementation description uses a more complex road network, which would be harder to draw, where a typical road is represented by two links, one in

each direction. We then encode the costs of going from each link to each possible successor link, incorporating knowledge that includes turn delays, different conditions in each direction and forbidden turns.

5   Thus we will show that our method works equally well for either a node-based or a link-based representation. It should enhance any routeing algorithms that are based on the computation of a shortest path (minimum cost) tree. It is independent of how those trees were arrived at, working solely from the minimum cost values and the back pointers that form the tree. From the computation of just two trees, it evaluates thousands of possible

10   routes to find a small subset that are all good alternatives, are each locally optimal, and are globally diverse.

The first step is to compute the minimum cost tree from the source node to all other nodes. This is typically performed using a variant of Dijkstra's Algorithm or the A* Algorithm,

15   often enhanced by clever use of trunk roads, precomputation and graph restrictions to speed up the computation or reduce the storage requirements. For our example, we show the result of this as Figure 2.

At each end of each road segment, we have annotated the distance *from* the source *to* that

20   end using the shortest path. From each end of each road segment, there is exactly one outgoing arrowhead that shows the way back towards the source using the shortest path. We call this a **back pointer**. Note that for a source tree it will be in the opposite direction to the direction of travel. These are computed and stored as a necessary part of the Dijkstra's or A* Algorithm.

25

For example, the shortest path from the source to the node marked as the destination (distance 310) can be traced backwards by following the arrowheads back through the nodes whose distances are 282, 245, 207, 197, 130, 85, 25, 14 and finally, 0. This is the shortest path route from source to destination, and would be traced in just this way by

30   Dijkstra's or the A* Algorithms.

The second step is to compute the minimum cost tree to the destination node from all other nodes. This is just a variant of the previous algorithm and the output is shown as Figure 3. This time, the annotations give the distance *to* the destination node along the shortest path, and the arrowheads, again exactly one from the end of each road, show the way forward to

5   the destination using the shortest path.

For example, the shortest path to the destination from the node at the top left with distance 234 is found by following the arrowheads through the adjacent nodes whose distances are 225, 215, 174, 113, 103, 65, 28, 0. This tree also encodes the globally shortest path, which

10   is found by following the arrowheads from the source node through to the destination node, and will always be identical to the one found from Figure 2.

Note the subtle difference between the trees, in that the first tree (Figure 2) encodes the shortest path routes *from* a single node *to* many others, while the second tree (Figure 3)

15   encodes the shortest path *to* a single node *from* many others.

The next step in our method is, for each node, to add together the minimum costs from each tree. We call this the cost-sum. The result of this is shown in Figure 4. For example, the number 471 at the top was arrived at by adding the corresponding numbers 218 and 253

20   from the tops of Figure 2 and Figure 3.

These numbers have a powerful interpretation. At any node P, they are the cost of the shortest path route from the source node to the destination node via the node P. Thus we have computed the set of shortest path routes from source to destination via any other node

25   in the graph. This in itself is a powerful result, but there are a huge number of such routes, and mostly they will not be relevant when planning how to get from source to destination.

We now observe that there are chains of adjacent nodes which have the same cost-sum. Of course, the nodes that lie along the shortest path route from source to destination must all

30   have the same cost-sum, which is exactly the cost of the shortest path route. However, there are other such chains. In this example, they have cost-sums of 310, 332, 335 and 395, and are highlighted with bold lines in Figure 5. We call each of the maximal-length chains

with the same cost-sum a **plateau**, as if the cost-sum represented the height above some mythical plane.

To find these plateau chains, a variety of known algorithms could be used. We will
5   describe a typical example.

For this example, we begin by giving each node a single bit that indicates that we have visited it, initially 0 for not-visited. We then scan over every node in turn, the order does not matter. If it is marked as visited (1), we move on to the next node in the scan. If the
10   node is marked as not-visited (0), then we change that to 1 to indicate that we have visited it. For each such newly-marked node, call it node Q, we begin a list of adjacent nodes in the chain by adding just a reference to that node Q. We then follow the link (arrowhead) in the source tree to an adjacent node, say node R1, and if it has the same cost-sum, we mark it as visited and we add a reference to R1 to the list and repeat for R2 etc. When this has
15   finished, we return to the node Q, and follow the link (arrowhead) in the destination tree to an adjacent node S1. If the cost-sum at S1 is the same as for node Q, then we add a reference to S1 to the list and mark S1 as visited. We then follow the link (arrowhead) in the destination tree from S1 to a new node S2, and again, add it to the list and mark it as visited if it has the same cost-sum as node Q. When this has finished, we have a list that
20   comprises all of the nodes in the chain that node Q is a part of.

At this point we compute a goodness factor for the chain, and keep it in the list only if it is one of the best n found so far, where n is typically 5, but could for example be 1000. We will describe the goodness factor later.
25

We then continue to the next node in the scan.

A plateau is formed when the source and destination trees traverse a chain of road segments in the same directions. This indicates that the chain is both useful for *getting*
30   *away from the source*, and for *getting towards the destination*. This is a powerful indication that the chain may be useful for *getting from the source to the destination*. Such chains tend to use the best roads in their vicinity, and are aligned to help in getting from source to

13

destination. On a real road network with millions of nodes, there are many thousands of such chains, many of them very short.

To make a complete route out of a Plateau, we simply have to follow the arrowheads in the source and destination trees back to the source and destination nodes themselves. Figure 6 shows the relevant arrowheads for the plateau whose combined cost (call it **CC**) was 395 in Figure 5. This route is the shortest path route from source to destination that incorporates the plateau.

For that plateau, its length (call it **L**) is simply the difference in the values at nodes A and B from the source tree (221 – 170 = 51 from Figure 2) or the destination tree (225 – 174 = 51 from Figure 3).

The shortest path route from source to plateau has a length (call it **SP**) equal to the smaller of the values at A or B from the source tree (min(170,221) = 170 from Figure 2). So **SP = 170**.

The shortest path route from plateau to destination has a length (call it **PD**) equal to the smaller of the values at A and B from the destination tree (min(225,174) = 174 from Figure 3). So **PD = 174**.

Thus the total length of the optimum route that incorporates the plateau is given by SP + L + PD, which in this case is 170 + 51 + 174 = 395. This value must be exactly the length of the shortest path route from source to destination via any one of the nodes in the plateau, which we have already found as the combined cost CC from the combined tree (Figure 4).

A useful plateau for getting from source to destination will tend to be longer than those that are less useful, as it indicates a long stretch of route that is fast and well-aligned compared to others in its vicinity. Thus we are looking for a larger value of L.

A useful plateau for getting from source to destination will tend to be part of a route that is not too long, as we are not interested in long plateaux if they are found at a great distance from both source and destination. Thus we are looking for a smaller value of SP + L + PD. Thus a good first estimate of the "goodness" of a plateau is the length of the plateau minus
5 the length of the route that the plateau is a part of, which is L – (SP + L + PD) = -(SP + PD). In our example it is –(170 + 174) = -344. In general, for more complex representations, it will be the cost of traversing the plateau minus the cost of the entire route. This is a negative number.

10 To make this measure independent of the length units (or in general, the cost units), we divide it by the length (cost) of the globally shortest path route, which in this case is 310. Thus our raw goodness (call it **RG**) is -344/310 = -1.11.

To make this more palatable to use, we can transform it by any function that preserves its
15 ordering. We call the resulting value the "goodness" or **G** for short.

The function that we shall use hereafter is $G = 100 - 99^{-RG}$

It is chosen so that the optimum route has a goodness G of 99, routes where the route outside the plateau is up to 85% of the optimum route length (RG = -0.85 or more positive)
20 have a value of G greater than 50, routes where the route outside the plateau is about the same as the optimum route have a value of G around 1 (there are usually thousands of these, they are all the minor variations on the globally optimum route), and worse routes have a value less than 1, which can go rapidly negative.

25 The other chains shown in bold in Figure 5 have the following values:

CC	SP	L	PD	-(SP + PD)	RG=-(SP+PD)/310	G
395	170	51	174	-344	-1.11	-64
335	99	136	100	-199	-0.64	81
332	56	78	198	-254	-0.82	57
310	0	310	0	0	0	99

For presentation to the user, we can use the values of G as a threshold, and a value of G>50 gives good results. If they wish to see more routes, we can always keep more, down to some lower threshold, say G>10. Typically, this will be accomplished by the user asking for more alternatives, or altering the threshold value for G. This should not be necessary for the average user.

In testing on the complete Great Britain road network, from Cambridge to Canterbury, there is only one route with G > 50. This uses motorways for almost the entire length, namely the M11, M25 and M2. For purposes of investigation, we relaxed the threshold, but the routes with G < 50 were not at all appealing because those motorways are so well aligned and bounded by poor roads in London to the West and the sea to the East.

For some applications, if there turn out to be many routes with G>50, then we may choose to limit the number shown to avoid overloading the user, typically showing only the 5 routes with the highest values of G. Again, the user can be allowed to change this parameter if they so wish, but we find that the top 5 usually encompass the most appealing routes. For a desktop route planner, we might show all routes with G > 50, but when planning in a vehicle or on a PDA, we may show only the top 5.

In testing on the Great Britain network, an example of this would be Cambridge to Manchester, where there are 8 good alternative routes with G > 50, but the top 5 probably encompass the most popular choices.

The function that is used to find the minimum cost route (in Dijkstra's or A* Algorithm for instance) can be much more complex that simply the distance travelled. In most situations, it is in fact dominated by the time taken, but may also be influenced, for example, by type of road, familiarity to the driver, financial costs, safety record in ways that are already known. We would typically use the best available time-independent cost function in the computation of the plateaux. It is simply a matter of labelling the roads with the appropriate cost instead of the distance, and the rest of the algorithm including the thresholding remains the same.

Time dependent information could be used in the computation of the plateaux, but care would have to be taken that only one cost is associated with each road segment. Otherwise the source and destination trees might have different costs for the same segment, which would prevent us from identifying the chains using the Combined Cost (CC) values. A

5   method of overcoming this is presented below.

An alternative method for handling time-dependent information is to compute the good alternative routes using the time-independent parts of the cost, and then compute the time-dependent factors for presentation to the user. Time dependent factors include the extra

10   time taken due to congestion, variable toll costs and time-varying road charges. In this way, a user can see how the good geographical routes have been affected by the time-varying factors, and decide for themselves if a journey at another time might be more appropriate, or which route they would still like to select. For example, a driver's usual route may have become five minutes slower than the optimum due to congestion, but if they were given the

15   choice, they may well wish to choose their usual route anyway. On another day, they might make the choice another way. The computer will never know enough to make this decision correctly under all circumstances, and so the driver should be informed about the realistic choices with the computer having done the hard work of estimating the distances, times, costs etc. of each choice.

20

If the costs are allowed to differ between the source and destination trees, perhaps because they are time-dependent and the estimates of traversal time are necessarily different, then we could not use the CC values to find the plateaux. An alternative is to use the back-pointers in the source and destination trees (arrowheads in Figure 2 and Figure 3) to

25   identify the links which form the plateaux. That is, to find consecutive links that are traversed in the same driving direction (but arrowheads in opposite directions) in both the source and destination trees. Any link that has an arrowhead on it in one direction in the source tree, and the other direction in the destination tree, is part of such a chain, and the plateau is found by combining all adjacent links that also have that property using a scan

30   technique similar to that described above.

Once we have reduced the number of plateaux to the most interesting ones by using G values, we can filter them in many other ways. For example, we may choose to order them based upon user preferences (motorways, fewer junctions, lower tolls, driving costs, familiarity). We can also show the user how historical and real-time traffic information has

5   altered our estimates of how long a journey will take, so that they can see why some particular route has become the fastest today, and how it compares with the others that they might be used to. The user, especially at the desktop, can be given the ability to sort the good alternative routes by any of these factors at the click of a mouse button.

10   We would typically present the advanced user with two controls for occasional use, one which sets the threshold value that G must exceed (default 50), and one that limits the number of alternatives displayed (default 5).

In our example above, the table shows that with a threshold of G > 50, we would only

15   recommend the routes generated from the plateaux with Combined Costs (CC) of 310, 332 and 335. Note that these happen to be the lowest values of the combined costs in the table, but that need not always be the case. Also note that all these routes have a cost that is within 10% of the globally optimum route, so other factors known only to the driver may easily outweigh the little extra distances involved, and they will be glad to have seen them

20   all.

We would typically present the routes sorted in order of goodness, which conveniently means that the first one will be the globally optimum route with goodness 99. We can present other computed factors such as time, distance, cost in the columns of a table, and

25   allow the user to sort and display according to these other factors if they so wish.

There are already systems that can compute alternative routes at the request of the user, either by disallowing the next link, or all links for some distance ahead in the planned route, and then finding the single most globally optimum route given those new constraints.

30   They typically find a short diversionary route around the problem area, and then head back onto the original route, although sometimes they may find an altogether different route. Our technique could also be used at this time, and could present more choices, but it may

be inadvisable to distract the driver with such information. Rather, our techniques could be made available as an extra option to complement the diversionary function.

For purposes of further illustration and explanation, a relatively simple actual example of a diverse route generating method is now described. This is illustrated in Figures 7 to 10, which are maps drawn to the same scale and extent. The example chosen is routes from Cambridge, UK to Manchester, UK.

The first step is to compute the tree containing the best (minimum-cost) routes from Cambridge to all other points. This is shown in Figure 7.

The tree has been simplified for printing, showing only its major branches, as the leaves of the full tree would fill the land almost completely at this scale. Methods for computing such trees are well-known in the art, and include the Dijkstra and A* Algorithms.

The second step is to compute the tree containing the best (minimum-cost) routes to Manchester from all other points. This is shown in Figure 8.

The third step is to find roads that are traversed in the same direction in both trees, that is, away from Cambridge in Figure 7, and towards Manchester in Figure 8. Such roads are not simply the overlap of the two trees, as the direction of traversal is important. The resulting roads are shown in Figure 9.

The fourth step, whose output is shown in Figure 10, begins by selecting the chains of roads from Figure 9 that are the longest, and generating complete routes from each by linking their endpoints back to Cambridge using the source tree of Figure 7, and back to Manchester using the destination tree of Figure 8. The routes are then ranked according to their goodness (the cost of the overlapping chain minus the cost of the entire route), and in this case, we have chosen to display the top five, numbered from 0 to 4 in order of goodness.

The final step in this embodiment is to display relevant information to the user, such as that shown in **Table 1**.

Plateau ID	Description	Duration	Distance
0	A14,A1,A57,M1,A628	2hrs 40min	256km
1	A14,M6,A556,M56,A5103	2hrs 41min	284km
2	A14,A1,A617,A623,A6	2hrs 55min	249km
3	A14,M1,A50,A500,M6	2hrs 50min	290km
4	A14,A1(M),A1,M62,A56	2hrs 53min	297km

Table 1.

A more comprehensive example of a diverse route generator and its application in a vehicle navigation system will now be described. Headings are provided for convenience.

Routeing Trees

To find these useful chains, we first look at routes from A to all nodes, and from all nodes to B. We can compute the optimum routes from A to all nodes using Dijkstra's algorithm, and observe that these can be stored in a single tree structure, a Routeing Tree from A (RTa), where each node or link contains a pointer to the previous one along the optimum route. Figure 11 shows a simple example to illustrate such a tree. The node or link can also contain the cost of the optimum route to that point. This tree has a tendency to use roads that are locally optimal and that are aligned well for getting away from A. By **locally optimal**, we mean that there are no minor diversions off those routes that can lower the overall cost.

We then compute a second Routeing Tree that stores the most optimum paths from all nodes to B (RTb). This tree favours road segments that are locally fast and useful for getting towards B and Figure 12 shows the resulting tree for the example illustrated in Figure 11.

Combining Trees

Now, we combine the two RTs to find chains of nodes that are used in the same direction in both trees. These chains are useful in travelling away from A, and for travelling towards B. To generate a complete route from such a chain, we trace back to A from the beginning of the chain using RTa, and trace forward to B from the end of the chain using RTb. The resulting routes for the example of Figure 11 are illustrated in Figure 13. This route will be locally optimal. That is, there does not exist any small deviation from it which has a lower overall cost for getting from A to B. If there was such a deviation from the chain, then the chain would have formed at the lower cost deviated route, and if there was such a deviation from the traced ends of the route, then the appropriate Routeing Tree (RTa or RTb) could not have been optimal.

Note that although the routes generated from these chains are locally optimal, they are not globally optimal. There is, in general, only one globally optimal route, which is the one found by single routers such as Dijkstra's algorithm. The next most optimal route will usually be a minor variation upon the first, with only a tiny increase in cost. For example, on a three hour road journey that begins and ends in cities, there will be potentially hundreds of small deviations through side roads that add mere minutes to the computed journey time. All of their permutations will generate millions of routes with journey times up to, say three hours and ten minutes. Only after that might we see the first journey that used a very different route between the cities, and takes an extra 12 minutes. This journey will also have potentially millions of small variations that are all just slightly more costly, but it will be a local optimum. Of course, even though it is the one millionth shortest route, it may still be a completely viable alternative, as the journey time estimates probably have an error of +-10 minutes, and if we are presented with the choice, we can take into account other factors such as how pleasant we find the route to drive, how familiar we are with it, the importance of financial cost versus time on this occasion etc.

Of course, there are many such chains, although an upper bound on their number is the number of edges in the graph, but they can be characterised by the length (in general, the

cost) of the overlapping chain of road segments (which we call a **plateau**), and the length (cost) of road for the optimum route that uses that chain. The cost information for this is already available in the RTs data, so this is a very efficient way to assess the **goodness** of the alternative routes. We have found an appropriate goodness function that not only orders the best diverse routes from the global optimum downwards, but which also provides a dimensionless threshold to ascertain how many alternatives are really useful.

Using this method, we can compute the N best diverse alternative routes from A to B in a time which is dominated by running Dijkstra's algorithm twice. We can assess and order thousands of such routes in much less time than running Dijkstra's algorithm, although usually there will be less than a dozen with a goodness factor above the threshold.

In some implementations of known single routers (shortest single path from A to B), it may be desirable to use a variation of Dijkstra's algorithm. These use special heuristics to explore the nodes or edges in a better order, or to terminate earlier, or to explore major roads only, or to explore outwards from source and destination simultaneously. They do this in order to run faster or to use less memory. It should be understood that our method of combining RTs to find plateaux is independent of how the RTs were computed, so if these variations are appropriate, then our method can be applied to them to yield diverse alternative routes while taking advantage of their better running time or memory use characteristics.

Cost Function

The cost function that is used to compute the best diverse alternative routes (the **choice routes**) can incorporate any of the known factors. It may be sensitive to both time and duration, can incorporate real-time or historical traffic information, can take into account user preferences, and can use financial information such as road pricing.

However, in a preferred embodiment, we use a basic cost function that weights mostly for time, with a small allowance for distance. This will generate most of the routes that a person might consider for getting from A to B. We can then present them in a route

planner with data for each that gives the time, distance, financial cost, congestion information, and any other parameters that can be computed such as number of turns, safety. We can compute these for different times of day, days of week, different types of vehicle, all very simply as we have only to consider the handful of choice routes found in the first step. Now we have put the user in the loop so that they can quickly see where each route goes, and make their decision based upon whatever weighting they wish to give the different cost criteria.

We believe that this will become particularly important if road user charging becomes commonplace, as it will not be possible to balance time, distance and charges in a way that is correct for all journeys, and certainly not for all users. Even the same user on the same day may have different priorities depending upon whether they have a fixed deadline for completing their journey, are becoming tired, or are just short of cash. They may normally prefer a much shorter non-motorway route, but might change their mind if the alternatives involve a lot of turns.

Choice Presentation

Another problem with known route planners is that the user may wonder why some other route was not recommended. By showing them the alternative routes along with their characteristics, it is easy for the user to see how their alternative route compares to the best, and to make an appropriate choice. For example, the usual route may be subject to unusually high congestion, or an alternative route may have become preferable because of the construction of a bypass.

In a route planning tool, we could highlight the different routes using different colours, and present a text box with relevant details about each route such as time, cost etc. An example of such a text box for the Cambridge to Manchester routes described hereinbefore is shown in Figure 14. Selecting a route on the map could highlight the relevant line in the text box, and selecting a line in the text box could highlight the relevant route on the map. In this way, a user could make their choice of route with as much information as we can give.

Instead of presenting the choice routes all at once in a route planning tool, we can consider other user interfaces. For example, for en-route guidance we would not wish to distract the driver with maps showing the alternatives. Rather, we can compute the points in the journey where a choice can be made, which we call **Choice Points**, and inform the driver

5 as they approach them. For the example of the six best routes from Cambridge to Manchester, the driver will be presented with choices at most three times during their journey. Further, these can be presented with a relevant subset of the information available, perhaps as a junction diagram, rather like roadside junction signage as illustrated in Figure 15. Such diagrams are commonly used in car navigation systems, and show the

10 driver the correct turn to make to stay on the optimum route. Our system would differ when a choice point was approached. We would show the junction diagram with both the optimum route and any other choice routes highlighted. We would augment this with information such as a brief summary of the roads used, the relative times to their destination, and the financial costs. These could be shown in the form of a road sign (and

15 should be no more distracting than that), and can also be read out using voice synthesis. It could take the form of "At roundabout take first exit for M6 and M56, or for ten minutes longer and seven pounds cheaper, take second exit for M1 and A6". This could be especially useful where traffic information has been taken into account and the user may be wondering why the optimum route is not the one that they expected. The time or cost

20 saved will make this clear to them.

The Choice Points themselves are straightforward to find once the Choice Routes are known, by following the small number of Choice Routes from the Source Link to the Destination Link.We begin with the Source Link, which is found at the beginning of every

25 Choice Route. If the next link on every Choice Route is the same, then we move on to that. If the next link is different on different routes, then we have reached a choice point, and follow each route separately as if we had just begun again at the source link. We finish when each route arrives at the Destination Link. We can either compute all of the Choice points as we begin the journey, or we can compute them as we go, always keeping at least

30 one known in advance.

There are other methods that could be used to compute the choice points, such as marking all of the the links used by one Choice Route, then marking each other Choice Route, noting where it encounters changes from marked to unmarked nodes (a choice point) and back to marked points (a point of convergence). Any of these methods can be used to determine the choice points, and can be chosen according to the precise needs of the application.

Constructive, not Destructive

An important property of our method is that it is constructive, finding all the good roads simultaneously, and assessing them using the very best weightings in the cost function and without having to blindly alter the cost elements of individual road links. Methods that alter the weightings or individual cost elements to cause the most optimum route to become less optimum can never be sure that they have not also lowered the optimality of the other good routes, as yet unknown, which will therefore never be found.

We believe that our method is the first such method to be discovered, as all the others that we have found in our research involve the alteration of the weightings or the costs of using individual road elements. Some even operate by disallowing one or more road elements used by previously found routes, equivalent to artificially giving them a cost of infinity.

Representation of Road Graph

There are a variety of representations that can be used for the weighted directed graph. For road networks for example, we may represent junctions as nodes in the graph, and road links between them as edges in the graph. To represent the costs of traversing the roads, for example the distance and average time taken to traverse a length of road between junctions, we can assign costs to the edges. To represent the costs of passing through a junction, for example the time taken to proceed straight on, or to turn left or right, we can assign costs to a node that are dependent on the edge used to reach the node, and the edge used to leave that node. Many variations on this representation are possible, and our methods should be applicable to all of them.

For this implementation, we use a representation that is particularly fast for computing the routeing trees. This representation does not represent road junctions as nodes. Instead, each edge is uni-directional, so a stretch of two-way road is represented by two edges, one

5   for each direction of travel, and these strictly form the nodes of the graph. This allows us to encode different delays for each direction of travel, which may be important at times of congestion. We call each edge a **Link**.

To encode the properties of junctions, each Link has a set of pointers to the next Links that

10   can be reached from it in the graph. These are called the **NextLinks**. These NextLink pointers are strictly the edges of the graph. Associated with each such pointer are the costs (in time, distance, financial cost etc.) of moving from the original Link to the next one. Our implementation chooses to use the time taken to move from the midpoint of one Link edge to the midpoint of another (NextLinkTime), but others may choose to use the

15   beginning of the Links, or their ends, it makes no difference to the outcome. We store the length of each Link (in LinkDistance), and compute the distance from the middle of one link to the middle of the next by adding the lengths and dividing by two. In this way, we have encoded both the cost of traversing a length of road, and the costs of using the junction for a particular turn, in one NextLink, which makes computing all of the optimum

20   routes from a source edge or node to other edges or nodes particularly efficient.

For computing optimum routes *to* a particular destination Link rather than *from* a source Link, we give each Link a set of **PrevLinks**, which encode the costs of entering that Link from all possible predecessor Links, that is, the segments of road from which it can be

25   entered. There is some redundancy here, as every Prevlink has a corresponding NextLink, so we use more memory than some representations, but we have a lower running time.

Figure 16 shows a picture of a road junction, where the side road is one-way towards the junction. The section of road marked A is represented by two links, marked 1 and 3, as it

30   is a two-way road. Similarly, road segment B is represented by two links, 2 and 4. Road segment C is only represented by one link, 5, as it is a one-way road towards the junction.

Now, vehicles travelling towards the junction on road segment A can only continue on to segment B, so there is only one NextLink (arrow) emanating from link 1, which points to link 2. Similarly, vehicles travelling towards the junction on road segment B can only continue on segment A, so there is a NextLink emanating from link 4 which points to link

5   3. Vehicles travelling on road segment C towards the junction can turn onto either segment A or segment B travelling away from the junction, so the corresponding link 5 has two NextLinks emanating from it, pointing to links 2 and 3. An important factor here is that the NextLinks can be enumerated from the link from which they emanate, not from the links that they point towards.

10

The PrevLinks are the inverse of the NextLinks, and are again enumerated from the link that they emanate from. They indicate the links from which vehicles may enter this link. For example, vehicles may enter road segment A at this junction from either road segment B or C, so link 3 has PrevLinks that point to link 4 and to link 5.

15

This link data is stored as a set of files, one for each type of information, that can easily be mapped into memory for efficient access. Each link is given a simple integer ID, length 32 bits, from 0 upwards, which is then used as an offset into various files. We shall use linkID 435 as an example. The length of link 435 will be found at the 435th 16-bit word of file

20   LinkDistance, encoded as the length in metres in 16 bits. The links that can be reached from link 435, its NextLinks, have 32-bit linkIDs in a file called NextLinks. The 435th 32-bit word in file NextLinkIndex gives the offset in the file NextLinks where the linkIDs of the next links will be found, contiguously. The number of them is found at the 435th 8-bit word in file NextLinkCount. The time taken to move to a particular NextLink is given in

25   the file NextLinkTime, also indexed by NextLinkIndex and NextLinkCount.
We also add a Master file that stores the number of links (nLinks), the number of NextLinks (nNextLinks) and the number of PrevLinks (nPrevLinks). In general, nPrevLinks = nNextLinks.

30   Here are the file definitions so far, with the number of elements in the file given in square brackets after the file name, followed by the type of each element:

File descriptions:

Master[3]: INT32: Contains the number of links (nLinks) and the number of next-links (nNextLinks), and prev-links (nPrevLinks) and maybe other top level information about how the files were generated etc. So far, all three fields are INT32.

5

All the other files are simple arrays. A link is defined by its link index, an integer between 0 and nLinks-1, which indexes all of the arrays contained in files whose length is a multiple of [nLinks].

NextLinkIndex[nLinks]: INT32: Contains the offset in the NextLinks file where the next

10   links can be found

NextLinkCount[nLinks]: INT8: Contains the number of next links

NextLinks[nNextLinks]: INT32: Contains the next links (by index) that can be accessed from one link

A similar set of 3 files called PrevLink... encode the routeing tree for destination routeing.

15   **PrevLinkIndex[nLinks]**: INT32: Contains the offset in the NextLinks file where the next links can be found

PrevLinkCount[nLinks]: INT8: Contains the number of next links

PrevLinks[nPrevLinks]: INT32: Contains the next links (by index) that can be accessed from one link

20   **LinkType[nLinks]**: INT8: The type of the road link. 0 is fast, 255 is slow.

NextLinkTime[nNextlinks]: INT32: The incremental time (milliseconds) taken to get from the middle of this link to the middle of the next link (including junction delay)

PrevLinkTime[nPrevlinks]: INT32: The incremental time (milliseconds) taken to get from the middle of the previous link to the middle of this link (including junction delay)

25   **LinkDistance[nLinks]**: INT16: The incremental distance (metres) when using this link

Routeing, Algorithm, Source Tree

The routeing algorithm that we use (a version of Dijkstra's algorithm) to construct the min-

30   cost routes from a given link to all other links (the Source Routeing Tree) is as follows:

Begin with all links having a min-cost which is the minimum cost found so far for reaching that link. Make it infinity for all links except the origin of the route (the origin link), for which the min-cost is zero. Construct a variable length list of active nodes, those for which we have not yet explored all of their outgoing links. Populate this list initially with the

5   source link alone.

REPEAT this:

If the active list is empty, then we have finished and we return.

Otherwise remove the first active link from the list of active links and call it linkP.

10   It has a min-cost costP that we know we can reach it with.

For each outgoing link linkPN, add the incremental cost costPN of going from linkP to linkN, and see if it is less than the min-cost (costN) stored with the outgoing link. If so, update costN with the lower value, update the back-pointer of linkN to refer to linkP, and add linkN onto the list of active links (if it is not already there).

15

When this algorithm terminates, we have constructed the Routeing Tree.

Storage of Routeing Tree

20   In our implementation, we may wish to re-use previously computed Routeing trees, so we store each one in the filing system as three files. Different trees are distinguished by using a trailing integer value <n> on each file. For example, TTBackIndex1 contains the back pointers for tree 1. By convention, the trees for the currently selected source and destination links are stored in files where n is 0 and 1. Again, the file lengths are given in

25   square brackets after the file name:

File descriptions:

TTMaster<n>[4]: A series of INT32s: nLinks, fromSource (0=dest, 1=source), rootLinkIndex, rootSubnodeIndex.

30   **TTBackIndex<n>[nLinks]:** INT32: The link that was used to get to this link at lowest cost, for tree <n>. Works for Next or Prev (source or destination) trees.

TTMinCost<n>[nLinks]: INT32: The minimum cost found to get to this link, for tree <n>

Tracing back the Optimum Route

To find the optimum route from the source link linkA to the destination link linkB, we need

5   only use the TTBackIndex0 file. For example, suppose that linkB had ID 678,123. The link that led us to linkB along the optimum route is found at offset 678,123 in the file TTBackIndex0. Suppose this link had ID 1,456,789. The link that led us to this one along the optimum route is found at offset 1,456,789 in the file TTBackIndex0. We keep repeating this until the link ID extracted from the file matches the linkID of the source link

10   linkA. Then we have extracted all of the link IDs of the optimum (shortest) path in reverse order. This is a common way of extracting the shortest path once Dijkstra's algorithm has terminated.

This extraction also works to find the shortest path from linkA to any other link linkC in

15   the graph, so long as we have allowed the Routeing Algorithm to run until there were no more active nodes. If the routeing algorithm was terminated early, then we can only use the tree to find the optimum route to a subset of the nodes in the graph.

Routeing Algorithm, Destination Tree

20

The routeing algorithm that we use (a version of Dijkstra's algorithm) to construct the min-cost routes from all links to a given link (the Destination Routeing Tree) is as follows:

Begin with all links having a min-cost which is the minimum cost found so far for

25   travelling from that link to the destination link. Make it infinity for all links except the destination of the route (the destination link), for which the min-cost is zero. Construct a variable length list of active nodes, those for which we have not yet explored all of their incoming links. Populate this list initially with the destination link alone.

30   REPEAT this:
    If the active list is empty, then we have finished and we return.
    Otherwise remove the first active link from the list of active links and call it linkQ.
    It has a min-cost costQ that we know we can reach the destination with.

For each incoming link linkNQ, add the incremental cost costNQ of going from linkN to linkQ, and see if it is less than the min-cost (costN) stored with the incoming link. If so, update costN with the lower value, update the back-pointer of linkN to refer to linkQ, and add linkN onto the list of active links (if it is not already there).

5

When this algorithm terminates, we have constructed the Routeing Tree.

Tracing back the Optimum Route

10 To find the optimum route from the source link linkA to the destination link linkB using the Destination Tree, we need only use the TTBackIndex1 file. For example, suppose that linkA had ID 238,345. The link that linkA preceded along the optimum route is found at offset 238,345 in the file TTBackIndex1. Suppose this link had ID 1,432,876. The link this one preceded along the optimum route is found at offset 1,432,876 in the file 15 TTBackIndex1. We keep repeating this until the link ID extracted from the file matches the linkID of the destination link linkB. Then we have extracted all of the link IDs of the optimum (shortest) path in forward order. This is a common way of extracting the shortest path once Dijkstra's algorithm has terminated.

20 Note that this path will be identical to the one extracted from the Source Routeing Tree in the section above. It does not matter if Dijkstra's algorithm is run from the source node outwards or the destination node inwards, it will always find the single shortest path route from linkA to linkB. However, the Routeing trees that are a by-product of Dijkstra's algorithm are very different. The only route that they share completely in common is the 25 global shortest-path route.

This extraction also works to find the shortest path from any other link linkC in the graph to linkB, so long as we have allowed the Routeing Algorithm to run until there were no more active nodes. If the routeing algorithm was terminated early, then we can only use 30 the tree to find the optimum route to a subset of the nodes in the graph.

Combining Routeing Trees

Once we have computed the tree RTa from the source link linkA, and RTb to the destination link linkB, we also have a wealth of cost information.

5

For example, to find the cost of going from linkA to linkB along the global optimum route, we look at offset <linkA> in the TTMinCost1 file that contains the min-costs for the destination Routeing Tree.

10 We can also look at offset <linkB> in the TTMinCost0 file that contains the min-costs for the source Routeing Tree. This value will be the same.

Now, what if we are interested in some other link linkC. The cost at offset <linkC> in the file TTMinCost1 is the cost of the minimum cost route from linkC to the destination linkB.

15 The cost at offset <linkC> in the file TTMinCost0 is the cost of the minimum cost route from the source link linkA to linkC. If we add these two quantities together, we have found the cost of the minimum cost route from linkA via linkC to linkB.

Thus, without any further iteration, we have found the cost of going from linkA via any

20 other link linkC in the graph to linkB. We can also rapidly trace the optimum route for display or other purposes by using the back pointers contained in the files TTBackIndex0 and TTBackIndex1. This is a very powerful result, as we have effectively computed a whole set of alternative routes (as many as there are links in the graph to go via) along with their costs. Of course, these are only a tiny subset of all of the possible routes, as they are

25 constrained to follow the globally optimal route from linkA to linkC, and the globally optimal route from linkC to linkB.

Plateaux

30 Now we observe that the routes are not unique. There will be some links, say linkP and linkQ, where going from linkA to linkB via linkP generates exactly the same route as going from linkA to linkB via linkQ. This happens when the route between linkP and linkQ in

one Routeing Tree (say the Source Routeing Tree RTa) is identical to the route between linkP and linkQ in the other Routeing tree (RTb). That is, the sections of road between linkP and linkQ have been used consecutively by both trees.

5   These are exactly the "good" chains of roads that we are interested in as part of diverse alternative routes. They are the chains that appear consecutively in both trees. We can attempt to find them by comparing the way that links are entered and left in each tree, or we can use the sum of the costs from RTa and RTb.

10   The chains are identified by the cost of going from linkA to linkB on mincost routes via any of the links in the chain being the same cost. For this reason, we call such a chain a plateau, as the sum of the costs is constant along the links that it is composed of.

The next two subsections give alternative methods for finding the links that are part of

15   plateaux, which may have different advantages according to the properties of the cost function, and the memory and processor architecture on which the methods are implemented. It will also be apparent that these methods can be adapted to different graph representations, means of storing the flags, and properties of the cost functions.

20   **Finding Plateau Links from Combined Costs**

To find all of the plateaux, we scan through all of the linkIDs. For each one, we look at the links pointed to by the BackIndex in each tree (RTa and RTb), and see if the sum of costs is the same on both of them. If they are the same, then we flag this link as part of a plateau.

25   The flags are stored, as usual, in a memory mapped file. It is called TCFlags01, and contains eight bits per link. It is initalised to have all bits set to 0 (flags unset) at the beginning of this procedure. We use the least significant bit (bit7) as the flag showing if the link with that offset has been identified as part of a plateau.

30   Here is the file definition for the flags, with the number of elements in the file given in square brackets after the file name, followed by the type of each element:

File description:

TCFlags01[nLinks]: A series of INT8's used as flags when combining RTs denoted by 0 and 1.

Once this scan is complete, we have flagged all links that are part of a chain (using bit7), excluding the end two links.

This method only works if the cost values are integers, so that rounding errors do not spoil the equality that we are looking for. Further, it only works if the cost for traversing a link in the tree is always the same, no matter what order the tree was constructed in. This is true in general, but if, for example, the link costs are time-dependent and traversed at slightly different times in the two trees, then this method will not work, and an alternative method based on back pointers must be used instead.

Finding Plateau Links from Back Pointers

To find all of the plateaux, we scan through all of the linkIDs. For each value of linkID, say linkP, we see if that link has a back pointer in the Source Tree. All links except the source link will have a valid pointer. Suppose that it points to linkQ. We then mark linkQ as having a source pointer by setting a nominated bit (say bit5 in the TCFlags01 file) at the byte in the TCFlags01 file whose offset is the ID linkQ. We also see if linkP has a back pointer in the destination tree. If it does, say linkR, then we set a different bit (say bit4 in the TCFlags01 file) at the byte whose offset is the ID linkR. If we set either of the flags at bit4 and bit5, we look at the other, and if it is also set, then we know that this link is part of a plateau and we can also set the plateau bit (bit7) as in the costs method.

Once this scan is complete, we have again flagged all links that are part of a chain (using bit7), excluding the end two links.

Gathering the Plateau Links

We now need to find the chains themselves, and characterise them. We do this with another scan through all of the links.

For each link, we check to see if it has been flagged as part of a plateau (using bit7). If it has, we look at bit6 to see if it has already been processed. If it has not, then we mark it as processed (using bit6), and trace back in both the source and destination trees for as long as

5   the links encountered are also marked as part of a plateau. We mark all such links as processed using bit 6 so that we will not process any links in this plateau if we encounter them again in later stages of the scan.

As we trace the plateau in this way, we could gather information about the plateau for later

10 use, such as its length and the cost of traversing it, but in this particular implementation, we will only use costs as the basis for selecting the best plateau, which are already stored in the TTMinCost files, so we do not need to compute them again. Rather, we note the linkIDs of the ends of the plateau, linkIDps being the end of the plateau nearest the source, and linkIDpd being the end of the plateau nearest the destination. For each plateau that we

15 have thus found, we extend one link closer to source and destination from the ends (if possible), and compute the difference in the mincosts for those links looked up from either RTa or from RTb (the results have to be identical as the plateau forms a connected part of each tree). This is what we take as the cost for traversing the plateau.

20 **Goodness Criteria**

Now, as we scan, we want in general to reject the shorter plateaux, and keep the longer ones. We also want to favour plateaux that form parts of routes that are not too long. If we are going from Cambridge to Manchester, we are probably not interested in a long plateau

25 that is in Scotland, which can sometimes form from certain alignments of roads.

To select only the plateaux that are of greater interest, we compute a "goodness" value, which we arrange to have higher values for the plateaux that we are interested in. We will then keep a note of the end links of the plateaux with the highest n goodness values, where

30 n is greater than the number that we could possibly want to display to the user, say 100. The goodness could be computed in any way, using any of the properties of routes such as length, financial cost, types of road, distance from source or destination.

In this implementation, we choose to base the goodness value on cost (as used to compute the mincost). We want a higher goodness for the plateaux that have a greater cost difference between their ends, as they represent longer parts of the Routeing Trees that

5   have been pulled into coincidence, and hence represent longer stretches of road that are locally useful in getting from linkA to linkB. However, we also want lower goodness values for plateaux that form routes from linkA to linkB that have higher overall costs.

Thus a simple goodness value is just the overall cost of traversing the plateau minus the

10   cost of the route from linkA to linkB using the plateau. This is in general negative, and is in fact the negation of the cost of the route using the plateau that is not contained within the plateau itself. Thus, we are preferentially keeping the plateaux whose routes have the least cost contained outside of the plateau.

15   As we complete each scan of a plateau, we compute its goodness, and save the top n (typically 100) that we find in a sorted list. We know that we only have to save the details of a plateau if its goodness is greater than the plateau with the lowest goodness in the list, which we then replace.

20   **Best Plateaux Selection**

The goodness value that we have defined has now put the plateaux in the order in which we would want to show them to a user, but how many should we show?

25   To do this, we need to transform the goodness value into something that can rate the plateaux routes on a scale that is independent of the source and destination links, and of the costs and scale of the road graph. To do this, we first make the goodness dimensionless by dividing it by the cost of the globally minimum cost route (GMCost). Call this the raw goodness.

30

To review, raw goodness is the negative of the cost of the route involving the plateau that is not part of the plateau, divided by the cost of the globally mincost route (GMCost).

This raw goodness will have a value of zero for the globally minimum cost route, as there is zero cost outside the plateau.

5 Now consider any tiny deviation from the globally minimum cost route (which is itself a plateau for its whole length) can only involve a tiny plateau, as no two plateaux can ever contact each other. The cost that is outside the tiny deviation will be just slightly lower that the GMCost, so the raw goodness will be a negative number that is slightly greater than -1, e.g. -0.994, -0.987, -0.992 etc.

10

Now, what about a plateau whose traversal cost is roughly identical to GMCost. If it extends very close to the source and destination link, then it will have little cost outside of itself, and so the raw goodness will be just slightly more negative than 0, perhaps 0.027, 0.012, 0.023 etc. If it is far enough from the source and destination link that the costs outside of the plateau are comparable with the GMCost, then the raw goodness will be

15 roughly -1. If it closer, such that the costs outside the plateau are half that of the GMCost, then it will have a raw goodness of -0.5.

To make these values easier to understand, we convert them monotonically to another

20 range, where 0 (the best) maps to 99, -0.5 maps to 90, -0.85 maps to 50, and -1 maps to zero. This is only used for display to the user, where the values between about 20 and 99 span the range of poor to good routes.

The function used is: goodness $= 100 - (99^{-\text{raw goodness}})$

25

To select automatically the minimum set of diverse alternative routes that are worthy of consideration, we set the goodness cut-off point at 50 (or -0.85 in raw goodness), and via display or otherwise make the user aware of only these choices.

30 For Cambridge to Manchester, this selects just six routes (our test database pre-dates the M6 toll).

Success Criteria

To be considered a success, our method must *only* come up with routes that seem reasonable, nothing outlandish. We could put limits on the time or length of journeys to

5 achieve this, but the goodness factor outlined above seems to make this extra step unnecessary, at least in this application.

The set of diverse alternative routes should include *all* those that we knew about as good routes from our own experiences, or at least minor variations on them (as the routeing

10 database does not perfectly reflect the user's view of costs, but is roughly correct over the longer stretches).

The set of diverse alternative routes should include *each* of the best single routes computed by other route planners such as (for Great Britain) Microsoft AutoRoute, ALK CoPilot, Medion Navigator, ISYS Personal Navigator.

15

We have found that the methods outlined above yield a set of diverse alternative routes that meet these criteria, whether the source and destination are many hundreds of miles apart, or just ten miles apart. If the source and destination are fairly close together (say 1 mile), then the routes cannot be very diverse, so only a few routes of almost identical length but good

20 diversity might be suggested. If they are so close together that one would only ever conceive of one route, then only one route, the globally mincost, will be suggested. Of course, many others will have been generated by our method, but only one will have a goodness of greater than 50.

25 **Plateau Algorithm Costs**

The scans involved in finding, classifying and sorting the plateaux are in general at least one hundred times faster than running Dijkstra's algorithm or its variants, so this entire phase takes much less time than the finding of the single optimum route. Thus the overall

30 cost of our method is approximately that of running a single route-finder twice, plus some fraction of that. As a rule of thumb then, it will take between 2 and 3 times longer than

finding the single shortest path route, but we will have considered many thousands, and selected our best few for display to the user.

Use of Other Criteria

5

Our example implementation uses a cost function that is 100 times the time in seconds plus the length in metres. Thus if we could save 1 minute of a journey (60 seconds) by driving an extra 1km, we would consider each route about even. This seems a good general setting to generate a good set of choices. It is weighting heavily towards the quickest route.

10

To favour shorter routes, we might 10 times the time in seconds plus the length in metres, which gives almost identical choices.

Ignoring time completely, by using pure distance as the cost function, does work, and we
15 get a different set of choices. In most cases, all of the routes are quite a lot slower, but we still get a diverse set of choices of similar cost (distance in this case).

There is no reason why we should not include anything else in the cost function that might be used for a single-router. These include financial cost (petrol, wear-and-tear, road
20 pricing, tolls), historical and real-time congestion information, safety, weather, type of road, areas to avoid, familiarity to the driver and many others. Some of these will change as the journey progresses, and we may wish to adapt to their new values. For example, the phrase dynamic routeing is used to describe systems where route choices are made according to the latest congestion information that is being fed to the vehicle. To use such
25 information, we could update the delays on the links in the database that are affected by congestion or other factors, and re-run our method. This is what single-routers do when new information is made available.

We have another, more interesting approach available. Since we have a set of good diverse
30 routes already computed, we can assess each one according to the new cost criteria in a fraction of the time that it would take to run a single router. We can then use these new cost criteria to re-order the routes for displaying to the user.

We can also check each of the diverse routes for cost and duration at different times of day, and provide the user with the optimum time to make their journey, or even the optimum days. If this were done by re-running a single-router for every ten minutes of every day of the week (1000 times), the time taken could be prohibitive (4000 seconds = 66 minutes). Our method, of selecting the set of diverse routes first and then assessing them using the time-varying cost parameters, will run in a tiny fraction of the time, probably about 12 seconds on this scale. It is rather analogous to a person looking at a route planning map and selecting the likely routes to use based on road type (speed) and direction, then assessing them individually for their properties with the aid of a computer to do the hard work of adding up all the costs. This is far less work for the computer than if it had to assess the costs of all possible optimal routes to come up with the least cost.

There will be some types of dynamic information where a major recomputation will be necessary, and so we would see our system as an addition to existing techniques. For example, if new information came in during a journey that showed a large delay ahead, before any other diverse routes left the current route, then we would fall back on existing technology to compute a single route around the new obstruction. Of course, once this was done, we could continue and compute the diverse routes from where we currently are to the destination, and offer a choice where appropriate.

An important feature of this choice routeing is the information that it gives to the user. In many cases where a single-router is run, the user does not understand how the optimum route compares to one that they might have chosen. Indeed, in most cases, both routes will have very similar costs. If we show the user the relative times and distances, then they can make the choice to use the route that they prefer, or follow the newly recommended route. What we will have avoided is the uncertainty that the user might have done something wrong, such as picked the wrong destination, set the wrong preferences etc.

If the optimum route suggested has been made because of unusual congestion, then this can be made apparent to the user at the (very few) choice points by showing them both options, the original best route, and the one that is recommended today, with the journey time

information for each. Indeed, we may even highlight if the journey time is significantly different from the usual.

For example, suppose a route involving the M6 is subject to an extra 20 minutes delay.
5   This may now make a route up the M1 some five minutes faster. A single router would automatically guide the user up the M1 route, whereas our method would allow them to make an informed choice to stay on the M6 (that they may prefer for other reasons) and sacrifice the five minutes.

10 This information could be given in the common forms of voice prompts, or schematic junction displays (just like road signs). The difference with our system, is that at critical parts of the journey, the guidance does not just give the single route to be followed, but gives two or more options (very rarely more than two) with some information about their relative journey times. Examples of this are shown in Figure 15.
15

Performance

In order to speed up the routeing computations, we can save the Routeing Trees both *to* and *from* recently selected links in the graph. This is an extension of saving recently entered
20 locations for easy recall, sometimes called a "favourites" list. If one end of a route is on the list, then we do not have to compute either of its Routeing Trees. The best single route to or from any other node can be found immediately by a backtrace through the tree. If both ends of a route are on the list, then we can recall both Routeing Trees that are necessary for our Choice Routeing, and no longer need to run Dijkstra's algorithm. Thus
25 we can come up with the diverse alternative routes quicker than a conventional router could compute the single best route.

The trees can be saved by keeping the files in which their costs and back pointers have been computed, or if an alternative implementation has been used, then new file formats
30 could be defined for the storage of this information. If storage is at a premium, then strictly only one of either the back pointers or the costs need to be stored, as a scan through the

routeing database can be used to reconstruct the other without the need to run a full routeing algorithm again.

Time Dependent Routeing

5

Where the speeds of the links may vary with the time of day, where road pricing changes with time, or where real-time traffic information is available, a conventional router will run something similar to Dijkstra's algorithm to find the best route, where the link costs have been altered to reflect the latest time-dependent information, and where they are sensitive

10 to the time that the links are expected to be traversed.

Our technique could be used in just the same way, running Dijkstra's algorithm twice with the latest cost information. The choice routes that take much longer than the globally optimal route might be using time-dependent information for slightly earlier or later times

15 than their expected time of traversal, because there is no longer a unique journey time from Source to Destination, but rather a small range of journey times. This is unlikely to be a problem as there is some inherent variability in journey times and predicted traffic levels anyway.

20 To remove this uncertainty in the expected time of traversal, we can take the few best generated routes, and quickly reassess them individually against the time-varying cost parameters, getting the most accurate result possible.

This then suggests a further technique. Instead of using the time-dependent cost

25 parameters in the first routeing runs, we could use a basic cost function, such as time (congestion-free) and distance alone. Once we have found the few choice routes using this basic cost function, we re-assess each of them against the new time-varying criteria, and simply re-order them for presentation to the user. In this way, we can be quicker to react to changes, we can make clear to the user why one route has been chosen rather than another,

30 and we can additionally show the user the choices that are almost as good as the best route.

If the cost function has changed radically due to the time varying or other cost parameters, then the choice routes computed under the restricted cost function may no longer include the globally optimum route under the full cost criteria. In this case, we can run our choice routeing algorithm in both modes, a restricted cost function and the full cost function. We can then choose to display to the user the best routes from each, so that they see both the very best routes currently available, and how they compare with the ones that they might have chosen based on time and distance alone.

The use of the full time-dependent cost function could be counter-productive in some circumstances. Under severe congestion, the time-dependent optimal routes are likely to involve more minor roads, along which there are no traffic sensors or reports. These roads appear to be free-flowing, where in reality they may be clogged, so the chosen routes may not be optimum after all. Thus we would always recommend that at least some of the choice routes computed without the time-dependent information, but reassessed with the time-dependent information, should be displayed so that the user can choose to stay on the major routes if they so wish. This could also be the safer option in the case of severe weather where the emergency services will be attempting to keep the major routes moving, and will be attending breakdowns.

Other Cost Factors

The RTs using the basic cost function do not have to be re-computed when the other cost factors are varied. When we re-assess each of the diverse alternative routes, we can use new weightings in the cost function, and new cost factors on the road links, to give us a new ordering of the routes for presentation to the user. Indeed, a good desktop interface should allow the user to sort the alternative routes by time, distance, financial cost, type of road, number of turns etc. as they explore their options. This does not necessarily require any recomputations, as those factors can be computed and stored along with the alternative routes when they are first generated.

It is not yet known whether it will be possible to define a useful cost function when road user charging becomes more commonplace. The problem is that the tradeoff between

speed of journey and cost may vary according to the importance of punctuality at the destination, which varies from day to day in an unpredictable manner. It may also be the case that the tradeoff varies according to how late the driver is setting off on their journey. If they are early, they may not wish to pay the charges for a faster route, but if they are
5   running late, then charges may be more acceptable.

Our techniques of computing the Choice Routes using only basic cost parameters (time and distance), would allow us to show the driver a choice of sensible routes along with their estimated durations and costs, allowing the driver to make the decision for themselves
10   taking into account all of the extra information that the route planner might never know. This is sometimes called "putting the human in the loop" and can make the resulting system much simpler and more comfortable to use, as the users are making the complex tradeoffs for themselves, while the computer is doing the hard work of estimating the journey times, distances, costs etc.

15

Traffic Planning

The Choice Routes do not have to be used singly. If we have journey information in the form of Origins and Destinations of journeys, we can use the choice routes to allocate
20   traffic to all of the links used in those routes in proportion to the goodness of the routes. When calibrated against real data by the adjustment of the parameters used in the goodness computation, the use of a Choice Routeing algorithm at the heart of such tools may give a more accurate prediction of traffic patterns than with current tools based on flow or single-route algorithms.

25

Methods of this type may also be applied to the routeing of wires, conductors or cables, for example in buildings, aircraft, printed circuit boards and integrated circuits. When finding a route for a wire, there are a few differences from the case for road routes, particularly the routeing of wiring nets, and the additional variable of placement order.

30

Wiring Nets

There may be several points from which a wire can originate, and several endpoints, any one of which is a sufficient destination. More complex are signal nets, where a wire is

5   required to connect at least one point from each of several sets of points. We will call all of these points origins, as there is no direction of flow as there is with traffic, so there is no clear concept of origin or destination. Similar techniques to Dijkstra's algorithm are often used, perhaps starting from several origin points simultaneously, and keeping the usual values of minimum cost and back pointer at each node that is traversed, terminating and

10   tracing back the optimum routes when all of the origin points have been joined. There are many variations of these techniques, but so long as they have the underlying architecture of a minimum cost and a back pointer at each node, we can apply our technique to add together separate routeing trees from each origin node, find the chains of identical cost-sum (the plateaux), and generate several alternative Choice Routes (or signal nets) by tracing

15   them back to the origins.

Placement Order

Another important difference between road routeing and wiring routeing is that for wiring,

20   we are not usually interested in optimising the path of a single wire, but of a whole collection. This brings in the problem of the order in which we should try to route the individual wires. Again, there are many variations, but a common technique is to find routes and fix the positions of some of the wires (called placing those wires), and then if a further wire has problems such as congested paths, we take up some wires that have been

25   placed, adding them to the list of unrouted wires, route the new wire, and then continue. This technique of rip-up and re-route is very expensive computationally, and may still terminate with quite poor routes. Various known enhancements try to estimate which signal wires are going to be the most difficult to place by looking at their lengths or the width of routeing channels along their rough path, or can observe areas of congestion as they arise,

30   and try to avoid clogging them up further by weighting the cost function against them.

Choice Routeing for Placement Order

Our technique of Choice Routeing can provide an important factor for choosing the order in which wires are placed. By computing the choice routes for each wire before any are

5   placed, we can see which have only one good route available (like Cambridge to Canterbury in the roads case), and which have several good routes (like Cambridge to Manchester). It then makes sense to place the wires with fewer choice routes first, because if they block some of the routes for later wires, those later wires will have other good alternatives available. Further, our goodness value can be used so that we are not only

10   taking into account how many good routes are available, but also how good they each are. A typical implementation will, for each wiring net, add up the goodness values of the top ten routes that are above a threshold (say a minimum goodness of 20), and then route the wires with the lowest sum first. Of course, this technique does not replace the known techniques, as we may still have to rip-up and re-route some nets, but we will sort the

15   unplaced nets by the amount of choice available, always placing those with the least choice first, which should speed up the time to find a solution, and more importantly, will improve the final result by keeping the wiring nets shorter overall, or not requiring such wide routeing channels.

20   Figure 17 shows an example of a navigation apparatus in the form of a computer 10 with a program memory 11, such as a ROM, flash memory, a hard disk drive and/or an optical disk drive, installed in a vehicle such as an automobile. However, such an apparatus is not limited to use in vehicles and may be attached to or installed in other objects. For example, such an apparatus may be installed in a cellular or mobile telephone, for example of the

25   type which receives information about its geographical location and about the current time.

The vehicle comprises vehicle electronics 12, which monitor and control many of the systems on board the vehicle. The computer 10 is thus able to receive information about various aspects of the current state of the vehicle.

30

The vehicle is further provided with a GPS receiver 13 for determining the location of the vehicle and the current time from signals received from the GPS via an aerial 14. This

information is also supplied to the vehicle electronics 12, for example so as to provide the function of a satellite navigation or "Sat Nav" system.

5   The computer 10 is also provided with an output device 15 for providing a human-perceivable output. The output device 15 is illustrated in the form of a display, for example forming part of the Sat Nav system or of an in-vehicle entertainment system, but may alternatively or additionally comprise an audio output device such as a loudspeaker.

10   Although the computer 10, the program memory 11 and the receiver 13 are shown as separate items in Figure 17, they may be embodied by a single apparatus which may also include the display 15.

CLAIMS:

1.   A method of generating a plurality of diverse routes from a source to a destination
in a weighted directed graph, comprising the steps of: generating a source routeing tree
5   from the source to a first set of points of the graph; generating a destination routeing tree
from a second set of points of the graph to the destination; and combining the source and
destination trees to form the routes.

2.   A method as claimed in claim 1, in which the first set comprises all of the points of
10   the graph.

3.   A method as claimed in claim 1 or 2, in which the second set comprises all of the
points of the graph.

15   4.   A method as claimed in claim 1, in which the first and second sets comprise points
of the graph adjacent both the source and the destination.

5.   A method as claimed in any one of the preceding claims, in which the first and
second sets comprise the same points.
20
6.   A method as claimed in any one of the preceding claims, in which the source and
destination trees are minimum cost trees.

7.   A method as claimed in any one of the preceding claims, in which the combining
25   step comprises selecting each sub-route common to and traversed in the same direction by
the source and destination trees.

8.   A method as claimed in claim 7, in which the source and destination trees include
back-pointers and the sub-routes are selected by finding sequences of adjacent points
30   which are pointed to by back-pointers of both the source and destination trees.

48

9. A method as claimed in claim 7, in which the points of the source and destination trees are associated with costs and the sub-routes are selected by finding sequences of adjacent points for which the sums of the costs from the source and destination trees are the same.

10. A method as claimed in any one of claims 7 to 9, in which the combining step further comprises extending each sub-route as necessary to the source and destination along the source and destination trees to form one of the routes.

11. A method as claimed in any one of the preceding claims, further comprising assigning a measure of goodness to each route.

12. A method as claimed in claim 11 when dependent on claim 10, in which the measure is a function of the length of the sub-route and the length of the route.

13. A method as claimed in claim 12, in which the measure is a function of the difference between the length of the sub-route and the length of the route.

14. A method as claimed in any one of claims 11 to 13, further comprising selecting only some of the routes in accordance with the measures of goodness.

15. A method as claimed in claim 14, comprising selecting the N routes of highest measure of goodness, where N is a positive integer.

16. A method as claimed in claim 14, comprising selecting at least some of the routes whose measures of goodness are greater than a threshold.

17. A method as claimed in any one of the preceding claims, comprising storing at least one of the routeing trees.

18. A method as claimed in any one of the preceding claims, in which the graph represents a road network and the routes are road routes.

19.   A method as claimed in any one of claims 1 to 17, in which the graph represents an integrated circuit or a printed circuit and the routes are interconnections.

5   20.   A method as claimed in any one of claims 1 to 17, in which the graph represents a wiring installation and the routes are wiring interconnections.

21.   A method as claimed in claim 19 or 20 when dependent directly or indirectly on claim 14, further comprising choosing the order of placement of the interconnections in 10   accordance with the number of selected routes and/or the measures of goodness.

22.   A method as claimed in claim 21, in which the interconnections with the lowest numbers of selected routes are placed first.

15   23.   A method as claimed in any one of claims 1 to 17, in which the graph represents a communication network and the routes are communication paths.

24.   A method as claimed in claim 23, in which the network is an internet.

20   25.   A method of navigation, including generating road routes by a method as claimed in claim 18.

26.   A method as claimed in claim 25, comprising presenting information about the road routes to a user.
25

27.   A method as claimed in claim 26, comprising presenting information about a choice of routes from a point from which a plurality of routes diverge when the user approaches the point.

30   28.   A method as claimed in claim 27, in which the information is displayed in a form representing a road sign.

29.   A computer program for performing a method as claimed in any one of the preceding claims.

30.   A computer-readable medium carrying a program as claimed in claim 29.

5

31.   Transmission across a communication path of a program as claimed in claim 29.

32.   A computer programmed to perform a program as claimed in claim 29.

10   33.   A computer containing a program as claimed in claim 29.

34.   An apparatus arranged to perform a method as claimed in any one of claims 1 to 28.

Application No: GB0621508.1	**Examiner:** Jake Collins
Claims searched: 1-34	**Date of search:** 3 July 2007

Patents Act 1977: Search Report under Section 17

Documents considered to be relevant:

Category	Relevant to claims	Identity of document and passage or figure of particular relevance
X	1 at least	EP 1335315 A2 (UNIVERSITY OF TOKYO) See figs 3-5 in particular

Categories:

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.

Field of Search:

Search of GB, EP, WO & US patent documents classified in the following areas of the UKC^X :

Worldwide search of patent documents classified in the following areas of the IPC

G06Q

The following online and other databases have been used in the preparation of this search report

WPI,EPODOC

International Classification:

Subclass	Subgroup	Valid From
G06Q	0010/00	01/01/2006
G01C	0021/34	01/01/2006