TimeContours: Using isochrone visualisation to describe transport network travel cost

Final Report

14 June 2006

Nicholas Street
Department of Computing
Imperial College London
ns1602@doc.ic.ac.uk

Supervisor: Naranker Dulay, nd@doc.ic.ac.uk
Second Marker: Morris Sloman mjs@doc.ic.ac.uk

Abstract

Modern transport systems and cities form complex spaces which are not easily represented on maps. Cartographic techniques have been developed to describe connectivity and shape, but these often serve to confuse rather than clarify the relation between space and time. Despite the modern world's high value placed on the commodity of time, there has been a lack of research into time based maps. In part, this reflects the difficulties in representing the fluid and shifting nature of time. Computers however offer the opportunity to redraw maps dynamically to reflect changes in the state and location of both the environment and viewer.

This project explores the potential in creating dynamic maps to describe time accessibility of networks from arbitrary starting points. It achieves this through the demonstration of the value of these "isochronic" maps, and in the creation of a framework for further modelling and experimentation.

Acknowledgements

I would like to offer my appreciation for the assistance that my supervisor Naranker Dulay provided me during the development of this project. His direction helped me retain focus on a broadly defined project, and his support remained positive throughout the ups and downs of the research process.

I would also like to thank my family who have suffered my reduced attention over the past year, and friends who have supported with ideas, feedback and food after late nights in the library.

Contents

Table of figures

1 Introduction

> *Arriving at King's Cross station I look up at the large Underground display, and see at a glance that my journey to Notting Hill Gate is going to be slower than expected; there are delays on the Circle line and TFL's journey time estimate has risen from the normal 15 minutes to 20. The news is even worse for my friend Sarah who is going to Hammersmith, as the Piccadilly line is down between Green Park and Earl's Court. The system now estimates the journey will take 35 minutes, but Sarah has a pushchair and is keen to avoid stairs and escalators. Thankfully we walk past an interactive version of the display, she swipes her Oyster card on a small reader to load her personal preferences and touches Hammersmith on the map – she should change at Green Park to the Jubilee Line and get the District from Westminster, the journey will take 42 minutes.*

This is a highly attractive scenario: a system that can offer all users relevant journey time information at the same time, with a simple mechanism to bring out personalised and hidden information when required. Providing this information is important since people increasingly judge space within a city in terms of journey time rather than geographic distance. Research shows that most people "gauge their travel time in terms of the 'number of stations' between A and B"[1]; creating this application will make time information explicit rather than inadequately inferred.

The technology to implement this vision largely already exists – TFL's journey planner estimates journey times and offers various personalisation options such as wheelchair usage or minimising walking between stations. The visualisation aspect however has not been sufficiently investigated, and we believe that various different techniques could offer journey time information. In particular we suspect that the implementation of isochrones will provide the most successful approach as it does not adversely disrupt the underlying map. Isochrones are lines on a map which draw a path over all points of equal journey time away from an origin. Multiple isochrones can then be overlaid on a map to indicate time boundaries at regular intervals. The lines closely resemble the relief contours frequently found on maps, and are occasionally referred to as "time contours". We will use the two terms interchangeably.

One of the earliest examples of the technique, seen in Figure 1, was created by Francis Galton in 1881[2]. The map charts travel across the earth in days, with green areas being accessible within ten days, yellow within twenty and so on. Isochrones are not the only kind of visualisation technique that can be used to describe journey time. Two other possibilities strike us as being lucrative: central point cartograms and 3D time relief maps. More detail can be found on these in the map visualisation section (page 15).

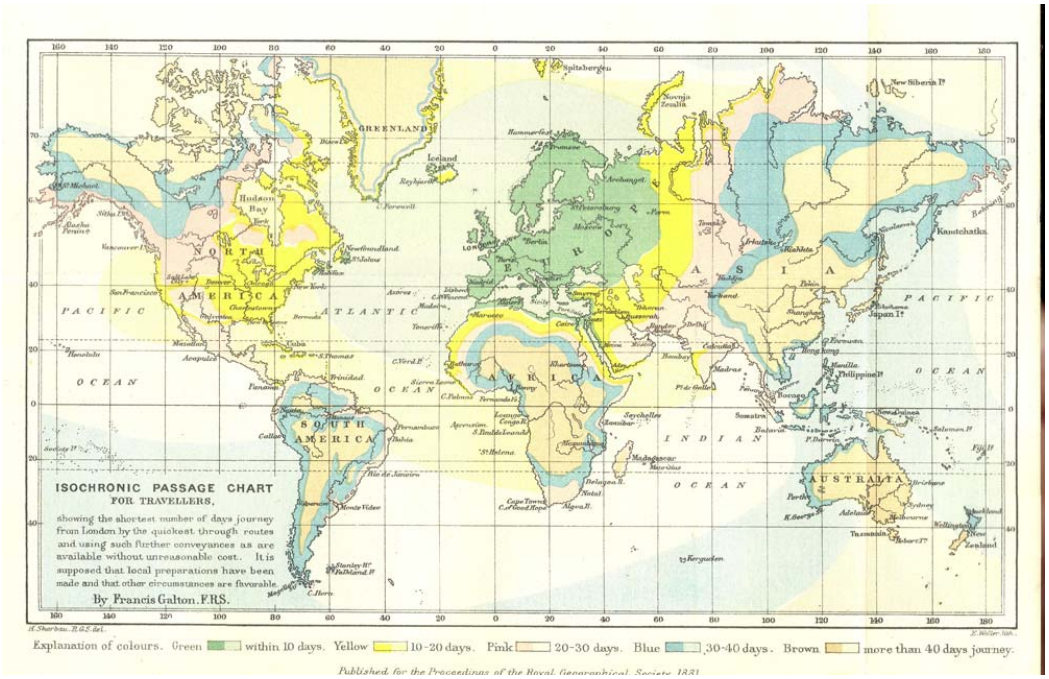


Figure 1 – Isochronic Passage Chart for Travellers by Francis Galton

The implementation of time-based visualisation techniques would be beneficial on travel maps at many scales: city-wide, nationally and internationally. London-wide for example it would allow people to investigate the accessibility of different areas around places of interest like the home or workplace. However the implementation on such a large scale would be an enormous task, and take attention away from visualisation towards the issues of modelling complex networks. The London Underground appeals as pilot test case for three strong reasons: it is a self-contained well defined system with plenty of available data, it is deeply in need of more time-based information and it is used by 2.7 million people every day[3].

The London Underground map has had time and distance between stations abstracted out in favour of connectivity, leaving a navigational map with an enormous amount of hidden time-based information. We cannot tell from looking at the map which lines are the fastest or which connections require long walks. Moreover, whilst users learn average journey times for their own particular routes, it is difficult to estimate journey times in a system that is constantly changing. The Underground is a turbulent system: time of day changes frequency of service and congestion, signal failures cause delays on line, bomb scares close whole stations and engineering works shut down whole sections of lines. When these events occur we often have very little information to go on in estimating how they will affect our journey; stations frequently have boards up and announcements telling us which lines are “delayed”, but these descriptions are vague and ill-defined. An isochronic version of the underground map could replace this with what people are actually interested in – how long it will now take them to reach their destination.

When brought down to the individual scale the purpose of time contours shifts. Whereas before they were a mechanism of offering time information to a large number of people about their specific journey, now they offer time information about a large number of places to a specific person. A user is likely to ask two

types of questions: what is the fastest route to a particular station and where can they get to within a particular time-frame. The first question is simple to deal with – the user can click the destination station and the suggested route will be lit up. The second question is more complex, and we must ask why somebody would be interested in this information.

The shift to an individual focus is fundamental, and gives us a chance to step back and re-evaluate the purpose of the program. In many ways the concept of the system as a whole is a mechanism for rating all the other stations in the network according to a person's likely desire to want to go there. Time is the most general criterion which any user of the underground can relate to – however, when we bring the system down to an individual level new contextual possibilities arise. When Sarah was planning her journey, time was not the only factor she was concerned about. The inconvenience of having to deal with a pushchair on stairs or escalators was perceived as being a greater "cost".

This realisation opens up a whole wealth of different criteria that could be considered: time of day, age, medical conditions, wealth, number of travellers, current activity, weather and events could all dramatically shift the "costing" of a travel network. The system could even use Oyster Card travel data to learn a user's previous journeys, and reduce its weighting for stations that they frequent – a person who has never been to Marble Arch is unlikely to be considering travelling there now. Meanwhile Russell Square might have a lower costing if the user frequently drops off there to visit their brother. Alternatively a group of people might tell the system they were going out for a meal: the system could then cost the different stations according to how well they cater for a large group's eating needs in combination with journey time. Isolines could then be drawn to visualise this information, with the interesting side-effect that the current station may no longer be the point of lowest cost. With all this criteria being taken into account, two different individuals' view of the map at the same station and time could be markedly different.

TimeContours is our implementation of an isochrone generating system for transport cost visualisation. The application is supports multiple network types, generating contours depending on user preferences and network state. We support two types of time surface output, and also investigate new types of surface such as using multiple start points. The system provides an engine for time visualisation, allowing developers to easily transform data and develop new ways of describing travel cost through a network. We conclude that time maps have great potential in helping people come to terms with the complex spaces that we live in. This project report looks into the background to map theory, related work, the TimeContours implementation, technical details and an evaluation of our findings

2 Maps – Principles and Practice

The map provides a way of modelling a larger complex space with clarity in a simple form. They achieve this by filtering and reforming information, converting places to icons, roads to lines and stripping out details that are unnecessary to the current purpose. Without this process the map making process would simply be a reproduction of something in its original form, and we would have gained no understanding at all. Jorge Luis Borges parodies this in his short story *On Exactitude In Science*:

> *"… In that Empire, the Art of Cartography attained such Perfection that the map of a single Province occupied the entirety of a City, and the map of the Empire, the entirety of a Province. In time, those Unconscionable Maps no longer satisfied, and the Cartographers Guilds struck a Map of the Empire whose size was that of the Empire, and which coincided point for point with it. The following Generations, who were not so fond of the Study of Cartography as their Forebears had been, saw that that vast Map was Useless"*

The discipline of Cartography has developed various principles and practices to achieve informative representations of reality. Before developing our own time based maps it is vital to have an understanding of these principles.

2.1 Representation of objects

Things on maps are normally represented as a point, line, area or volumetric object. Scale then often determines the method of representation that is chosen – on a close up map a house might be represented as an area, and further away as a point. A collection of houses might then be represented as an area or even as a line if they follow a road. Further away again a point might be used to classify the collection as a village. To take a different example, relief might be represented as a series of contour lines in one map, and through an isometric projection in another.

On the standard London Underground diagram each station is represented as a point, and each connection as a line. Various different point symbols are then used to indicate interchanges to tube stations, railway lines, riverboat services and airports. The map also implements shaded areas to represent the zones that each station is in. The proposed visualisations would only adapt these slightly. For the isochrone approach the zone regions would have to be removed as they would obstruct the clarity of the contours. The isochrones are lines, although if the areas within them are shaded they might be considered areas too. The proposed 3D visualisation would involve creating a volumetric surface and laying the map onto this.

2.2 Representing Geographic Space

When it comes to representing geographic space we have a choice between (or a combination of) raster and vector approaches. The raster method cuts the study area up into small units, each of which represents one section of the space in question. In general an area is divided into equally sized squares, which when put together form a grid. Raster data structures are not limited to this layout however,

and can consist of variably sized (as in the quadtree) sections built from any geometric shape. The only requirement is that the pieces can be fitted together to form a planar surface covering the study area. The most familiar use of the raster data structure is with images. In this case each unit, called a pixel, is given a colour; when all the pixels are put together in a grid they resemble an image. With maps each unit is often assigned multiple types of data, such as type (lake, sea, land etc.) and height above sea level.

Vector based maps are built out of sets of nodes and lines. Each node is given a location in a coordinate space, pairs of nodes describe a line segment, and sets of line segments describe more complex lines, which can be looped together to form areas. Attributes can then be stored against nodes, lines or areas as required. This model allows for complex analyses of the relationships between sections of the map, as well as scaling much better than raster implementations. In particular, the vector model lends itself to the description of networks. For a road network nodes become junctions and lines roads; further detail can then be added to inform that certain roads are one-way, or that you cannot turn right at a particular junction. Vector networks can then be analysed using *Graph Theory*.

Raster	**Vector**
Data structure is simple.	Data structures are complex.
The method is compatible with remotely sensed or scanned data.	The method is not as compatible with remote sensing data.
Procedures for spatial analysis are simple.	Some spatial analysis procedures are difficult.
Greater disk storage is often required.	Compact data structure requires less disk storage.
Topological relationships are difficult to represent.	Topological relationships are readily maintained.
Unless extremely small cell sizes are used, the graphic output is often aesthetically less pleasing.	Graphic output is aesthetically more pleasing and more closely approximates hand-drawn maps.
Projection transformations are more difficult.	
	Overlaying multiple vector maps is often time consuming.
	Output graphics may take hours to draw on plotters.
	Software and hardware for vector systems are often more expensive.

Table 1 - Adapted from *Comparison advantages and disadvantages of vector and raster methods as revised from Burrough (1986) and Aronoff (1989)*[4]

In choosing an approach for the TimeContours application it is clear that the use of vectors for the underlying data is fundamental to the implementation. The underground is very much a network of the form modelled by vectors, and the use of graph theory to analyse it will be imperative. Creating the output however

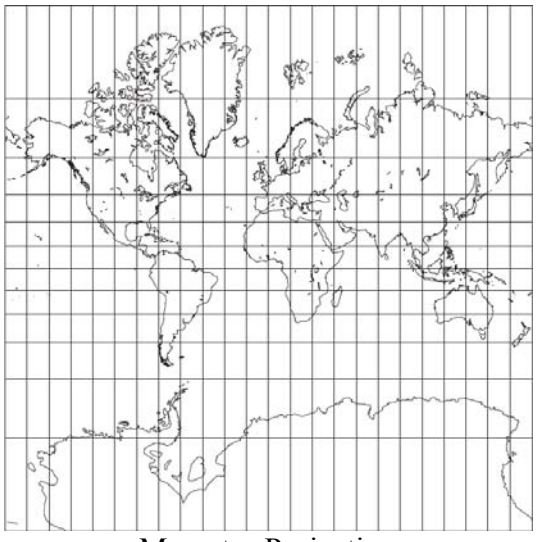
becomes more complicated. Using a raster image is not appealing as it reduces our flexibility – we would like to be able to dynamically highlight stations and the connections between them. We would also like the flexibility of being able to actually move the stations themselves if we so desired. However, it will be difficult to get the dynamic edges to precisely match those in the London Underground map and since keeping as close as possible to the original map has been stressed as an important point, the map may be implemented as a raster image. Stations and lines with vector representations can then be placed on top of this image as desired

2.3 Map Visualisation

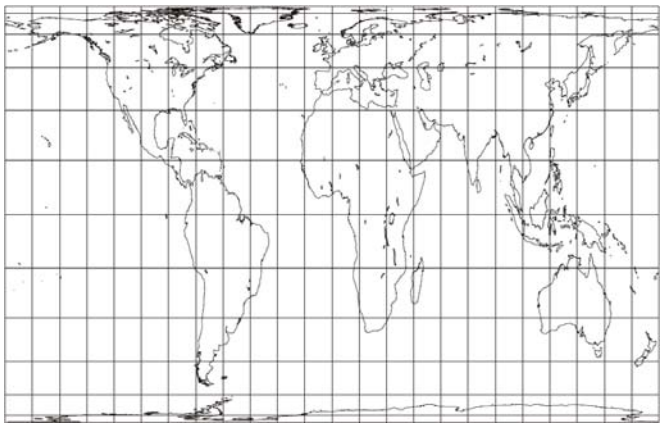
The visualisation of the map is central to the project. It is therefore important to understand what a map is, and the different approaches to visualisation. Maps are models of reality displaying selected information in order to record and describe some aspect of the space around us. They are by definition distorted versions of reality, and what a map tells us is defined by the manner of this distortion. Mark Monmonier underlines the importance of this point: "Not only is it easy to lie with maps, it's essential"[5]

2.3.1 Projections

The problem of how to display even a simple map is a difficult one, particularly because of the issue of projections. The Earth is a sphere, and any map that attempts to represent a section of the Earth must deal with the problem of converting a 3D curved surface into a 2D flat one. Projections are the mechanisms that we use to perform such transformations, and various different ones exist to control the resulting distortion. The popular Mercator projection for example was designed to preserve the angles that related different points on the map, a property particularly useful for navigators. The Peter-Gall projection meanwhile ensures that the relative areas of places and countries are kept to scale, at the cost of distorted shapes.



Mercator Projection



Peter-Gall Projection

For the small area being dealt with on the London underground, dealing with the curved surface of the Earth is not enormously important. For the underground map, which is not based on geography, it becomes entirely irrelevant, and on a true geographic map we can happily use the projection used on an already existing map and overlay our time contours on top. There may however be something to learn from the concept; the idea of the cartogram for instance is closely related.

2.3.2 Cartograms

Cartograms are formed from maps where the areas of subsections in the map are distorted according to some other detail being measured. Cartograms are powerful tools, demonstrative of the fact that our attachment to a spatial representation of information can be very misleading. Until recently their generation however was very difficult; the reforming the map whilst still retaining a recognisable relationship between the reformed version and the original had to be done largely by hand. Now, thanks to an algorithm developed by Michael Gastner and Mark Newman of the University of Michigan in Ann Arbor the task can be more easily automated[6]. An archive of their maps is available on the web[7], two examples are shown below.

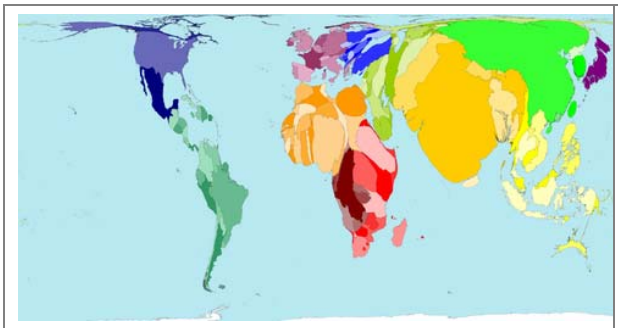


Figure 2 – Cartogram of population in 2050

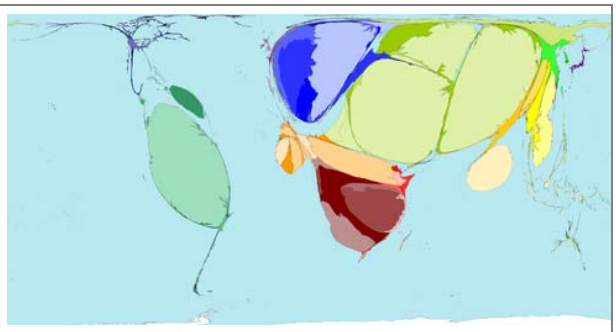
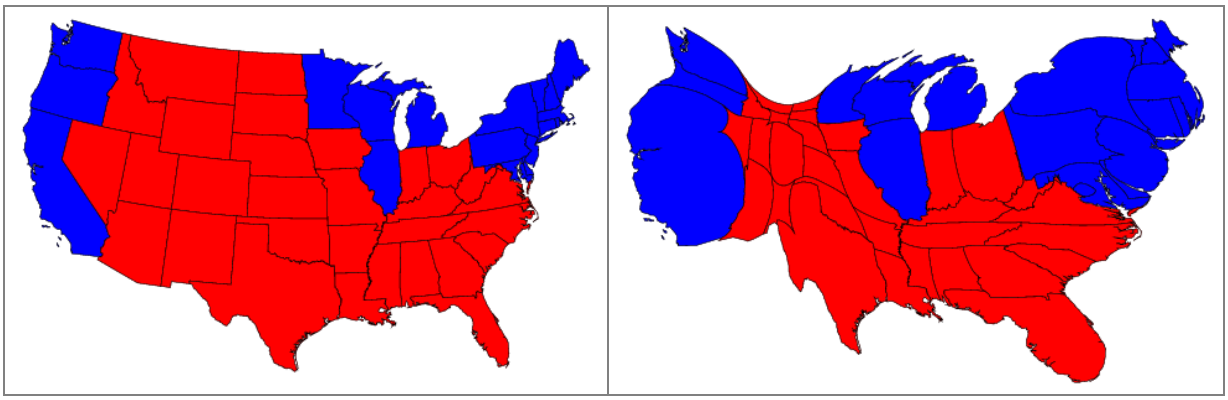


Figure 3 – Cartogram of refugee origin

For example, a geographically based map of American election results in 2004 easily misleads a viewer into thinking that Bush's dominance is much greater than it actually is. The population cartogram corrects this by rescaling the states according to their populations rather than area, and revealing the reality of an evenly divided vote.



Geographic Representation of Election Results	Population Cartogram of Election Results
Red represents states won by George W. Bush Blue represents states won by John F. Kerry	

Cartograms are not limited to being based on area. *Linear Cartograms* remove both distance and spatial relationships between places. For this kind of map it is the connectivity and shape of the network is most important. As has already been mentioned, the London Underground map itself is a famous example:

Figure 4 – Detail of the London Underground Map[8]

The London Underground Map is very easy to read, and the original version by Harry Beck has been widely applauded as a breakthrough in representation. However, it still distorts our idea of time, since distance, speed and transfer time are all omitted from the diagram. It seems that a whole dimension has been lost.

2.3.3 Contours

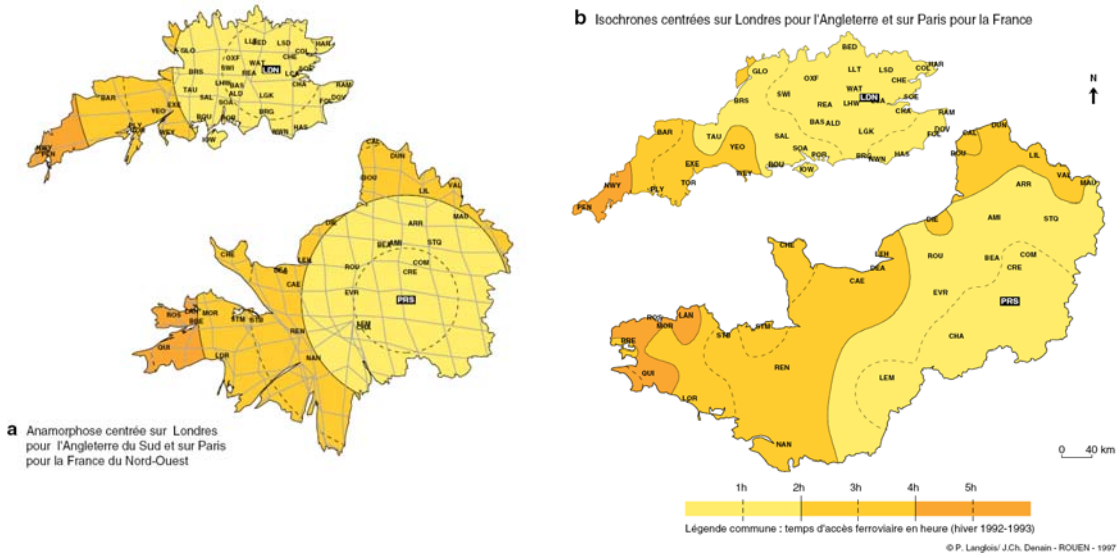
The spaces that standard maps are drawn for are in three dimensions, and have to be described in two on paper. In many cases we are able to make use of a map in which the relief of the land is not described, but in many cases it is vital information. Cartography has come up with an effective method of displaying the missing height dimension in the form of isolines called contours. Contours trace a line on a map where the elevation of the land is constant – and are labelled with this elevation. By putting sets of contours for different elevations together we are able to build a three dimensional mental image of the area in question. Lines that are close together indicate a steep gradient and ones farther apart a shallow one. In the sections between contours we have incomplete information which the brain is forced to roughly approximate. A careful balance is then required between increasing contours to get a more accurate surface and reducing them to stop other information in the map being hidden.

Contours are method of displaying a third dimension on a two dimensional image. The concept can be simply borrowed and applied to time – this kind of contour is called an *isochrone*. An isochrone is therefore a line on a map joining points together which are all a constant time distance away from a starting point. They are used across various disciplines from looking at the propagation of

tsunamis[9] and radio waves[10] to journey times on maps[11]. It is unusual to apply the concept to a map that has already been distorted in space, but there is no reason in principle why it shouldn't work. In fact the concept of distorting maps is closely related to isochrones and anamorphosis maps are based upon them.

2.3.4 Anamorphosis Maps

Anamorphosis maps distort a map according to the time that it takes to travel to different points on the map. These are also known as *central point cartograms*. With these maps the distance between the starting point and any other point on the map exactly corresponds to the length of time it takes to travel there. On this kind of a map isochrone lines become concentric circles. The below diagrams[12] offer a comparison of the two different approaches. We find the isochrone approach to be slightly more effective, since it is easier to relate the diagram to our existing understanding of shape of France and Southern Britain:



Anamorphosis map centred on London (top) and Paris (bottom)

Isochrone map centred on London (top) and Paris (bottom)

Figure 5 – A comparison of anamorphosis and isochronic maps

The above maps both offer clear information about travel times around London and Paris. The maps' clarity is based on the combined use of the visualisation technique and choice of colour. The different blocks of colour make the isolines stand out from one another, and provide an easy referencing system against time. The use of a sequential scale of colours is important here – we quickly learn that a darker colour indicates an increased journey time allowing us to make relative judgements without having to refer back to a key or read numbers. This is an example of the value of using discontinuous colour approach, but continuous shading can also be effective. Such shading is frequently used on relief maps, and allows us to naturally gain an understanding of the lie of the land. The results can also be aesthetically pleasing, as the below relief map demonstrates:

Figure 6 – "Karte der Gegend um den Walensee" (section) by E. Imhof 1:10,000, 1938.

2.3.5 3D Rendering

Another method of describing relief is the use of 3D renderings of a terrain. The below images from the Grand Canyon taken from Google Earth demonstrate how effective this can be. The left hand image is a satellite picture of the region, and whilst natural shading allows us to get an idea for how the lie of the land should be, the addition of height information and occlusion in the right hand image is dramatic.

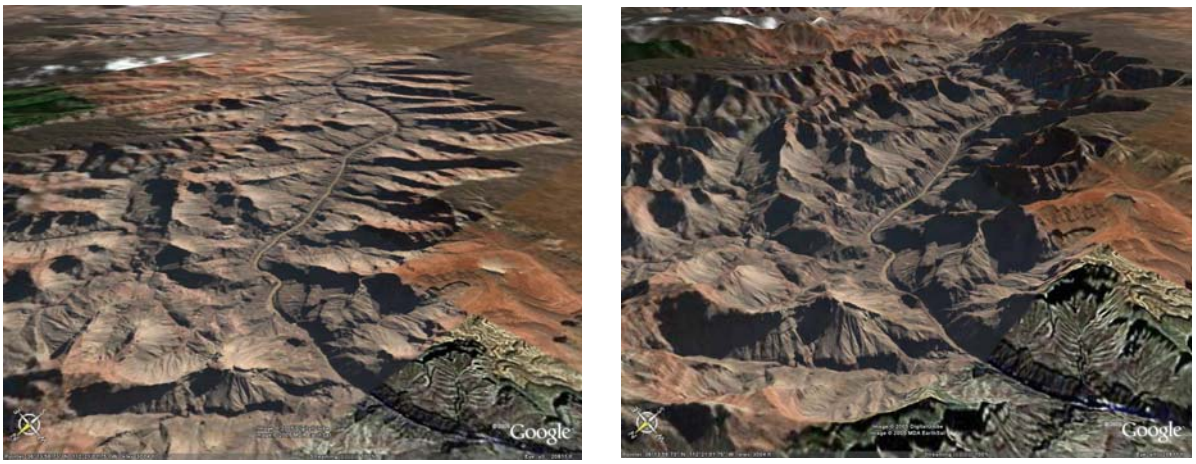


Figure 7 – Two Shots of the Grand Canyon in Google Earth.

Again, since the adaptation of other relief description techniques to time has been successful, we think that a. Although there appears to be little past research in this area, the navigation software Fugawi has already proven that three dimensional presentations of two dimensional maps can be effective:

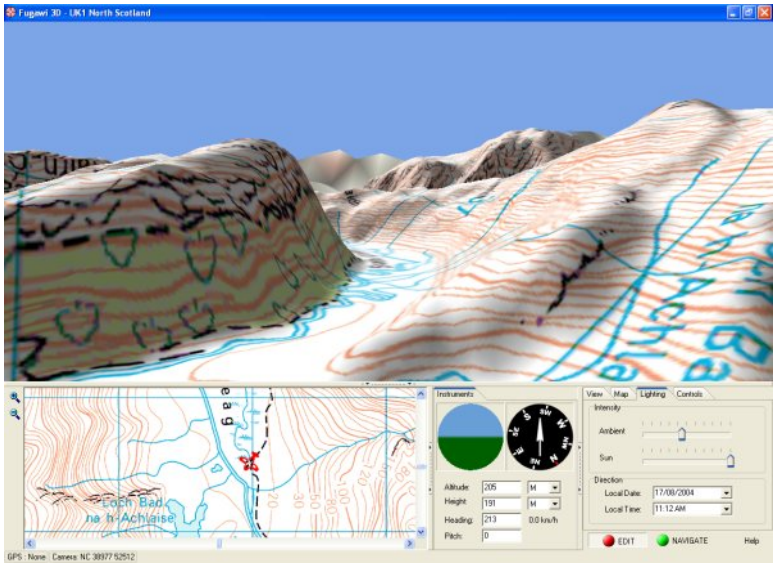


Figure 8 – Using Fugawi to produce 3D presentations of maps

2.4 Geographic Information Systems

A system that can analyse networks and spatial data to construct maps is normally referred to as a **Geographic Information System** (GIS). GIS is in fact a larger field geared towards the storage, refinement, analysis and output of geographic data in many forms. The systems can be used to generate complex maps with alternative views, but are not limited to conventional cartographic outputs. Alternative displays of information include both graphical displays such as wire-frame diagrams to show relief and tabular data such as distance charts.

In general a GIS consists of four subsystems[13]:

- Data input subsystem
- Data storage and retrieval subsystem
- Data manipulation and analysis subsystem
- Reporting subsystem, which displays all or part of the database in tabular, graphic or map form.

2.4.1 Where is GIS used

Modern technology such as the internet and mobile phones appear to have allowed us to transcend borders and the ties of the geography around us. In reality however the importance of understanding the relationship between spaces is more important than ever. For example, the telecommunications companies that built these systems were only able to do so with the help of GIS, which could balance their needs of mobile masts in high open areas, whilst avoiding dense residential populations. A massive proportion of compiled information in the modern world is tied to geography, and most fields can benefit from the ability to analyse and interpret this information. A few of the fields that use GIS today are[14]:

- Property developers
- Retail sector
- Utility companies

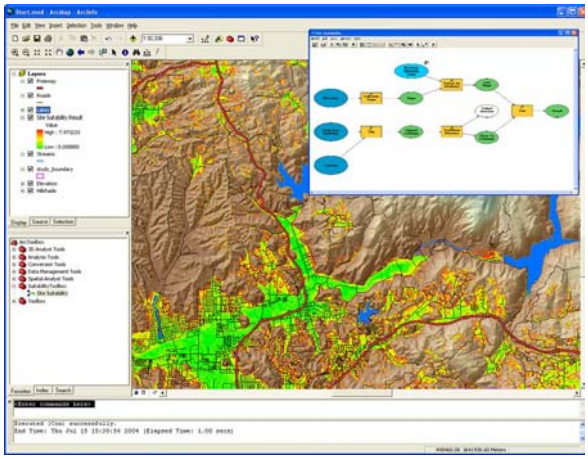
- Environment
- Local government
- Health care
- Transportation
- Financial services

The importance and power of GIS has been recognised by many major fields – but as yet it has failed to reach the everyday market. Current GIS software is constructed for large companies with complex needs and the casual user has been neglected. The reception of the recent release of *Google Maps* and *Google Earth* has demonstrated the demand and interest in this field by the general public. Google Maps took the standard static representation of maps on the internet and made them dynamic – allowing the user to pan around the map, zoom in, zoom out and switch between map and satellite views, all without refreshing the page. There was an enormous amount of interest in the development, and the additional inclusion of an API led to the creation of many "mashups" which overlaid information on the maps. Whilst the usage of "AJAX" to allow all this in a web browser may have been new, the product itself was not. GIS applications capable of panning around a map and switching between different "layers" have been around for decades.

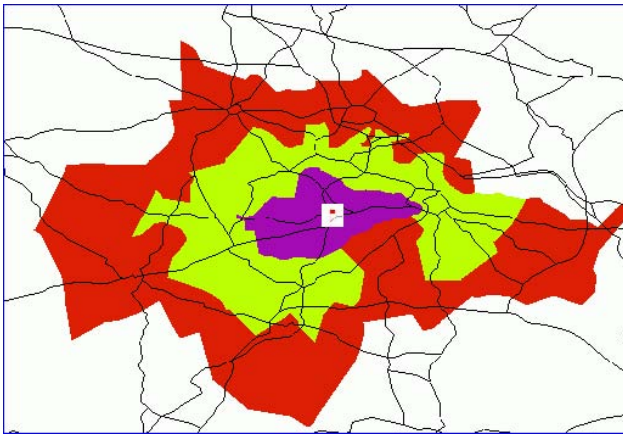
The public is clearly interested, but GIS has a lot more to offer than the capability to browse maps. I believe that with the dynamic map come new possibilities in the representation of spatial data. In particular, time is a dimension that static maps have found it difficult to represent. A standard map offers information to the user about all the places shown on the map. When a map is distorted by time however a starting point, or *time-zero*, is chosen, about which the map is realigned. Once this process has occurred however the map's applicability is reduced to the point at which time is zero. With the dynamic map we can throw away old versions of maps as desired, and reproduce them focussed on a new point.

2.4.2 GIS Toolkits

GIS software currently comes in three main categories: High-end GIS toolkits, Desktop GIS applications and Browser-based GIS. The High-end toolkits are large and powerful commercial applications which include tools to solve a plethora of problems including data input, database management, analysis and visualisation. The most well known GIS package in this genre is ArcGIS, which is used in many disciplines from cartographers to utility companies. The size, expense and complexity of these tools means that they are better suited to GIS experts and large commercial organisations than everyday users. ArcGIS does not support isochrone generation natively, and a plugin such as RouteFinder[15] or ProTERRIRORY[16] is needed to do this.



ArcGIS screenshot



Polygon isochrone generated with the RouteFinder plugin to ArcGIS

Desktop GIS applications attempt to reach a middle ground, targeting users who don't have the level of GIS expertise required to use high-end toolkits like ArcInfo, nor the resources available to invest in such a system. Products such as MapInfo offer the ability to do spatial analyses, and can be tailored for individual needs. These kinds of applications are still directed at company use.

Browser based GIS is a fast developing field. Traditional such technologies like the MassGIS browser (http://www.mass.gov/mgis/mapping.htm#browser_apps) provide various tools, but the interface is clunky and slow. The mapping website Google Maps meanwhile has stimulated this field by including an API. This development has allowed people to combine the maps with other data sources to construct a primitive GIS. Unfortunately however the API does not include access to the underlying network data, so it cannot be used as a data source for our investigation.

2.5 Space Syntax

Space Syntax Theory is a relatively new way of looking at and analysing space. The theory was originally devised in the late 1970's by Bill Hillier and Julienne Hanson at University College London for looking at the social effects of architectural designs. Since then the set of analysis techniques the theory provides have been applied to a wider range of disciplines, including looking at the structure of cities.

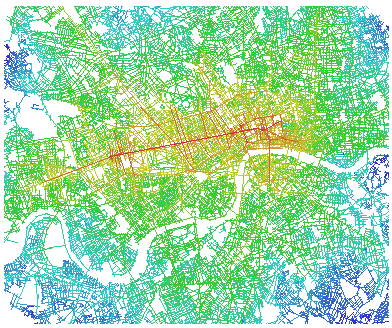
In *The Common Language of Space*[17] Hillier notes that perhaps the most important feature of a city it its complexity. Science has recognised this fact, but with no way of measuring or comparing forms of complexity it has largely ignored the problem of its analysis. Hillier argues that the complexity of a city is

> *"best captured by representing it not as physical stuff, but as the system of space created by the physical stuff."*

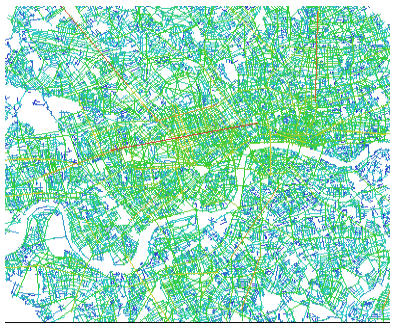
After investigating space syntax's efficacy in elucidating information about factors such as movement, land use patterns and economic performance, Hillier suggests it be used "as a general means for investigating the relation between the structure and function of cities".

Axial Maps are the primary tool in Space Syntax for analysing space. An axial map is built up by drawing lines over communication routes such as roads. The points where lines overlap are called junctions. Different kinds of analyses can then be carried out to look at the relationship between different lines. On a Global Integration map each line is weighted according to the number of steps across junctions that are needed to get to any other line on the map. In the below map of London this has highlighted Oxford Street, as needing the fewest connections to get to any other point in London. In the Local Integration map we limit the number of steps that are considered, this brings out important communication routes away from the City Centre.

Axial Map of Central London



Global Integration of Central London



Local Integration of Central London

Space Syntax is not focussed on time, and as a result provides a general rather than station-centric description of space. There may be scope for combining the advantages of both Space Syntax and Time Maps as analysis techniques, both in the arena of the cityscape, and underground network. This is an undeveloped research area, and a recent study by the Space Syntax Lab stated that "no published precedent for analysing underground systems was found in published space syntax literature"[18].

2.6 Summary

An effective map requires great care to be taken on how its components are to be represented. A map that fails to do this is likely to include "1+1=3" artefacts[19], where poorly displayed details combine together to suggest something that is not implied by the actual data.

Choice of colour, icons and other map components is therefore very important.

A variety of technologies exist for analysing and representing space. Space Syntax demonstrates the scope for new and innovative ways of looking at networks including both analysis and demonstration of findings. Cartography meanwhile has developed many techniques for visualising extra dimensions on the two dimensional planar surface of a map or computer screen. These can be taken and adjusted for implementation in the dimension of time. Of these tools, this project will focus on isochrones, which we see as being the most accessible to the uninitiated onlooker; however, within the field of time maps the other techniques should not be overlooked.

3 Modelling networks

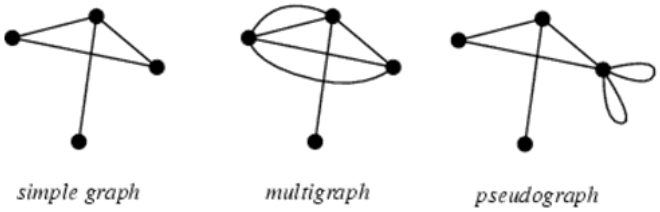
In order to visualise the time properties of a network, it is necessary to build a model to compute these times. These times can then be used in combination with node position data to construct a cost surface for contour generation and colour shading. The two primary techniques for constructing such a model are to use either timetabling or graph analysis. Timetabling as a technique is particularly appropriate for public transport networks where trains or buses arrive and depart from nodes at a particular time. A simple graph model meanwhile largely assumes that you can start and stop as desired.

Timetable data is not freely available in Britain, but it is possible to use online journey planners such as Transport for London's Journey Planner[20] and National Rail Enquiries[21] to obtain times between stations in the network. A web based robot can be built which when given a particular station would fill in the form hundreds of times to obtain journey times to different stations throughout the network. By coupling this information with station locations a surface can then be generated. This is the technique employed by the recent work on time maps by MySociety[22]. The approach creates accurate models and includes the ability to achieve this for different times of day or days of the week. There are however certain drawbacks which led to the decision to avoid this approach. Firstly we do not have an accurate database of station locations, without which the information is largely useless. The method is also slower than a direct analysis, since the information has to be downloaded each time a start station is selected. This conflicts with our intention to create an implementation that can be experimented with and browsed by a user. Research has shown that people understand environments much more when they can directly interact with them[23]. Since the isochrone for most people is a new way of representing information, we did not want to lose this feature in our prototype.

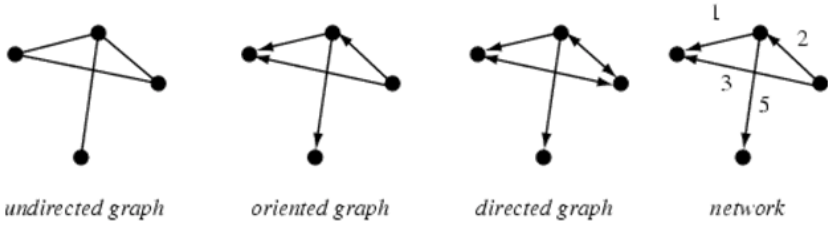
In lieu of accurate timetable data we instead made use of a graph based network, which simply gives estimations of journey times between adjacent nodes. Journey times to any other part of the network can then be constructed by traversing the graph to find the shortest route. This kind of model is admittedly not as accurate as a timetabled solution, but it does provide a certain degree of flexibility. Changes to the network for example can be modelled by adjusting the journey times of certain edges (as we have implemented with Graph Filters, see section 8.1.3). The same engine meanwhile can be used for modelling most network types, as we have demonstrated with the road based Open Street Map prototype.

3.1.1 A quick introduction to Graph Theory

Graph theory is the study of vertex-edge graphs. A graph in this context is a collection of vertices (also known as points or nodes), which are connected together by edges (also known as lines or arcs). Graph theory is the mathematical model used for analysing networks and a number of algorithms have been devised for solving problems such as the shortest path problem. Graphs are classified into different types according to whether multiple connections between nodes, and self-connections are allowed[24].



Edges meanwhile can be directed or undirected. Directed edges allow flow between a pair of nodes to be blocked in one direction, and allowed in the other. Edges can also have weights attached to them, indicating the cost experienced in travelling along that edge.



Graphs can be used to model many different types of systems, from flow through pipes to computer networks and transport networks. Each of these systems has a different set of problems that may need different algorithms to solve. The shortest path algorithm is one that is required across many disciplines, and is particularly needed in the context of isochrone generation. The problem of finding the shortest path between two vertices was solved by Dijkstra[25] and can be computed in linear time.

3.1.2 Summary

A graph based network provides the best base from which to build a generic network model foundation. The analysis of graphs is well researched and documented, and with the potential for filtering it is possible to add large degrees of complexity if required. Although a timetabling system is appealing, using a graph for the underlying model does not rule out inclusion of this data at a later date.

4 Related Work

When this research project began almost all investigations into the use of dynamically generated time-contours had been done for commercial purposes rather than every day users. Recently however that has begun to change. This section summarises some of the related work being done in this field.

4.1.1 Karlin's "Time Travel"

At the end of November 2005 Oskar Karlin released an artistic representation of the underground using isochrones[26]. His version is centred on Elephant & Castle, and also rearranges the tube lines themselves. The piece generated a huge amount of interest on the internet and also reached national newspapers[27]. The discussions on Karlin's webpage have also been useful in directing our research.

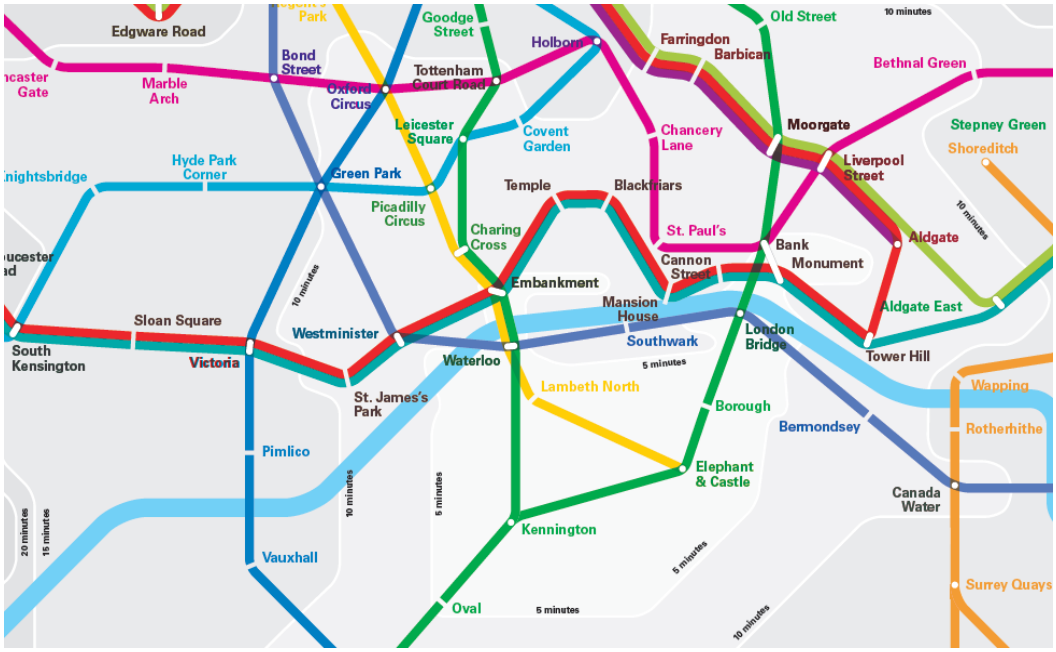


Figure 9 – 'Time Travel' Oskar Karlin's representation of the underground using isochrones. Each isochrone represents a further five minutes travel time from Elephant & Castle.

Whilst the map is interesting, it is unfortunately not that useful. Underground users have built up an understanding of the current tube map, and can quickly and effectively use it for navigating. Whilst the original map deliberately removed geographic spatial information for brevity, people's understanding and use is still very much based on spatial memory and the relationships between the stations. For example, people expect the top edge of the circle line to be a straight line – when this line is rearranged and recoloured the instinctual understanding of the line as representing the underground is lost. If isochrones are to add to a user's understanding of the London Underground then they must build on their present understanding.

The top edge of the circle line according to TFL

The top edge of the circle line according to Oskar Karlin

Figure 10 – A comparison of two representations of the Circle Line

Various people have picked up on this problem, and investigated correcting it. Rod McClaren[28] has drawn up a possible ten-minute time contour on the standard tube map (see below). Despite being rough it succeeds in conveying more information to a user very quickly. However, it could clearly benefit from displaying multiple contours on the same diagram and being dynamically generated

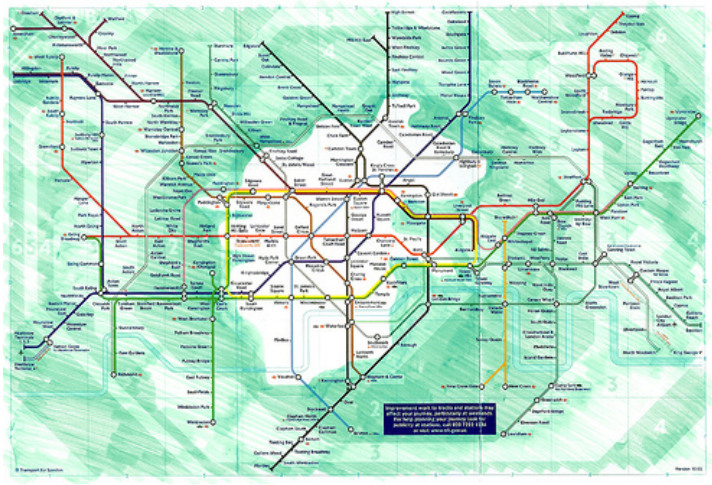


Figure 11 – Rod McClaren's sketch of a 10 minute contour from Oxford Circus

4.1.2 Carden's Travel Time Tube Map

Initial "geographically correct" view of the London Underground

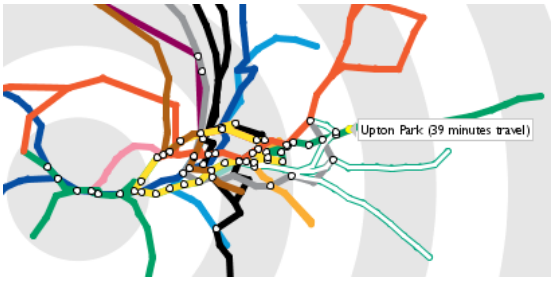
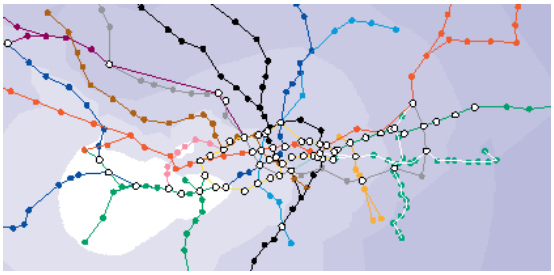
	*Redrawn view of the Underground according to journey time from Turnham Green.*
	*Redrawn with Time Contours*

Figure 12 – Screenshots from Tom Carden's travel time tube map java applet[29]

Tom Carden's Travel Time Tube Map was inspired by Oskar Karlin's diagram above, and introduces dynamically generated visualisations and animates the transitions when the focus changes to a different station. The first version of the applet generates dynamic anamorphosis representations of the underground, using concentric circles to indicate 10 minute intervals. A station's distance from the selected station indicates journey time, and the angle relative to the focussed station is kept correct for as-the-crow-flies direction on a map. The second version[30] released earlier this year displays ten minute isochrones on the map. These are effective prototype implementations, but the unfamiliar unlabelled layout makes it difficult to relate to the underground.

4.1.3 Pinsky's "In Transit 3D"

The artist Michael Pinsky has explored the issue of journey time being higher on a traveller's agenda than mere distance. One of his creations to explore this issue is *In Transit 3D*:

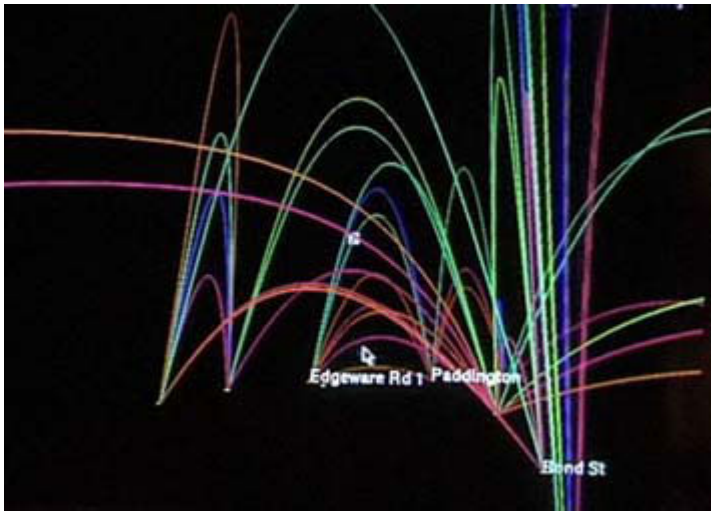
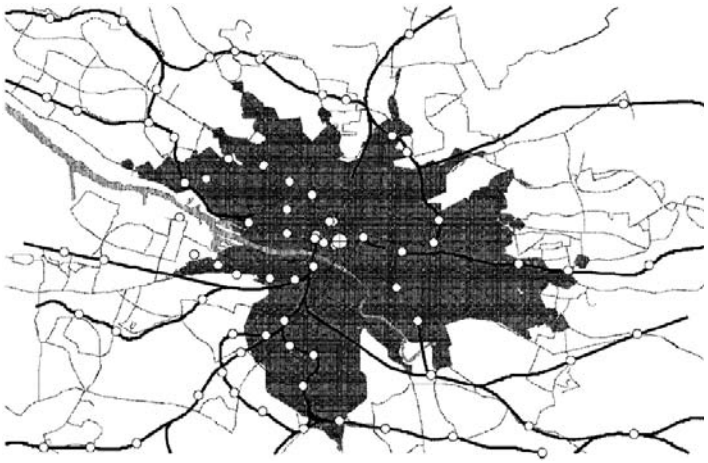
	*"In Transit 3D is an animated interactive map in three dimensions. The viewer chooses the nodes that he/she wants to travel between and then chooses the mode of transport that he/she wants to take. The slower the journey the greater the arc between the nodes, so the architecture of the space is created through time itself."*[31]

Figure 13 – In Transit 3D by Michael Pinsky

Pinsky's concept of using arcs in the third dimension appears to have potential, and may form the basis of a possible visualisation implementation. However without experimenting with the system or getting more information on it, it is difficult to perform a substantial analysis of it.

4.1.4 O'Sullivan, Morrison and Shearer – Using desktop GIS for the investigation of accessibility by public transport: an isochrone approach[32]

O'Sullivan et al's research article on using isochrones to describe a public transport network in Glasgow covers similar ground to this one. Its investigation does not cover the possibility of a dynamic system, and its visualisations are low quality and difficult to interpret. By focussing on a single network and developing a system created and personalised for the end user we hope to improve on their design.



5 Specification

The objective of this project is to produce a visualiser for time-based maps. The application will build on and extend the existing work discussed in the background and be focussed on implementation for the London Underground. It will differentiate itself from other the other underground time based maps by offering dynamic visualisation with reference to the original tube map. Furthermore it will extend previous research work by offering multiple visualisations and response to changes in the underlying network such as time of day and closed stations.

The project aims to produce a product that will be of real value to any person who uses the London Underground. It should offer the user intuitive and clear information about journey times to all parts of the underground network in the form of a dynamic map. In doing this it will provide users with new information over and above what they can discover from the static underground map.

5.1 User Interface

The project is directed towards anybody who uses the underground. This is a large group of people (2.7 million people use the underground every day), and a non-technical market. It is also to be used as a tool in making navigation decisions, with particular concern put towards saving people time. To achieve this it is vital that the user interface is intuitive and quick to use. From the moment of invocation the map should be displayed, and the basic visualisation functions should be available – any personalisation or extra settings should be layered on top of this basic level. Interaction should be natural, allowing the user to click on stations to select them and refocus the visualisation. A quick method of doing route-finding should also be provided, such as using a mouse-over. To maintain the simplicity of the basic interaction, other tools should be provided in a non-intrusive control panel. This will provide the functionality to change between visualisations and search for stations.

Ideally the interface should be built using a system that can be accessed via a website, allowing any London Underground user to access it. Java or Flash are powerful technologies that support this functionality. A specialised station-based implementation could also be considered – for this attention should be directed towards the fact that many people may be using the system concurrently or in quick succession – personalised settings must therefore be easily removed.

5.2 Visualisations

The visualiser module will be able to take network state information and convert it into a visual display for the user. The module will support multiple visualisation possibilities, but the basic procedure will be the same for all:

1. Create a cost-surface from the network data and station journey-times
2. Perform calculations to build up the necessary internal model for the visualisation

3. Generate the visualisation

The displays should endeavour to intuitively convey journey time information throughout the network. This should be achieved “at a glance”, without in depth analysis being required on a user’s part.

An ideal implementation of the project would include multiple visualisations, such as time based renditions of the isochronic, anamorphosis and 3D rendering techniques described in section 2.3. These visualisations would then be composable, so that a user could select a set of active visualisation methods, and all would update the same map at the same time. For example, you could choose turn contour lines and 3D rendering on and off independently. Alternatively multiple visualisations could appear at the same time, this would allow the effects of changes in the network to be represented in different ways concurrently.

5.3 Data Storage and Management

The most important aspect of the data storage and management system is the ability to support multiple data storage mechanisms. It should be possible to use both a database, so that a server-style implementation with a large amount of data could be implemented. Furthermore it should be usable as a standalone application, so support for simple file-based storage should be included. In implementing a file system care should be taken to ensure compatibility with data standards, particularly if a graph based system is used. The formation of many standards for holding graph data has held back the graph software community, and there is a drive to achieve a global standard. Meeting this standard will not only open up more data sources, but also encourage greater interest from the wider community in the engine. The data format must also be flexible, capable of holding the required data for complex networks without breaking compatibility with other applications or alternative network class models. Ideally this should be achieved with minimal redundancy.

The data management system also needs the support for building new networks and inputting data, even if this is only for the generation of the prototype implementations. An interesting extension would be introduce a collaborative mechanism for acquiring journey time information over the internet.

5.4 Network Structure and Analysis

The network model should be capable of supporting various network types, and of providing journey times to every node in the network from a particular start node. It should also support the ability to determine the fastest route between two nodes. The analysis should also be as fast as possible; quick analysis will assist in two areas: providing immediate feedback in an interactive system, or supporting larger networks in a research based implementation. The background research in section 3 suggests that the most likely tool to be used in doing this is a graph. If graph analysis is used it is important that the analysis is carried out using a Fibonacci heap, which can massively increase the efficiency of graph algorithms.

The model should be able to support complex adaptations, including changes in state of both the network and user. An accurate model of the underground for example should also consider details such as transfer times, times to wait for a

train and disruption to the network. Ideally this information should update to match real time information. User adjustments meanwhile should ideally take account of walking speed and accessibility requirements. Further extensions could combine the network with more complex analyses, such as taking account of the purpose of the trip or current events.

6 TimeContours a walkthrough

The concept of a contour through time is not necessarily a natural one, and so it is important to have a feel of what they show and how they work before looking at how the project has been implemented.

6.1 The general interface

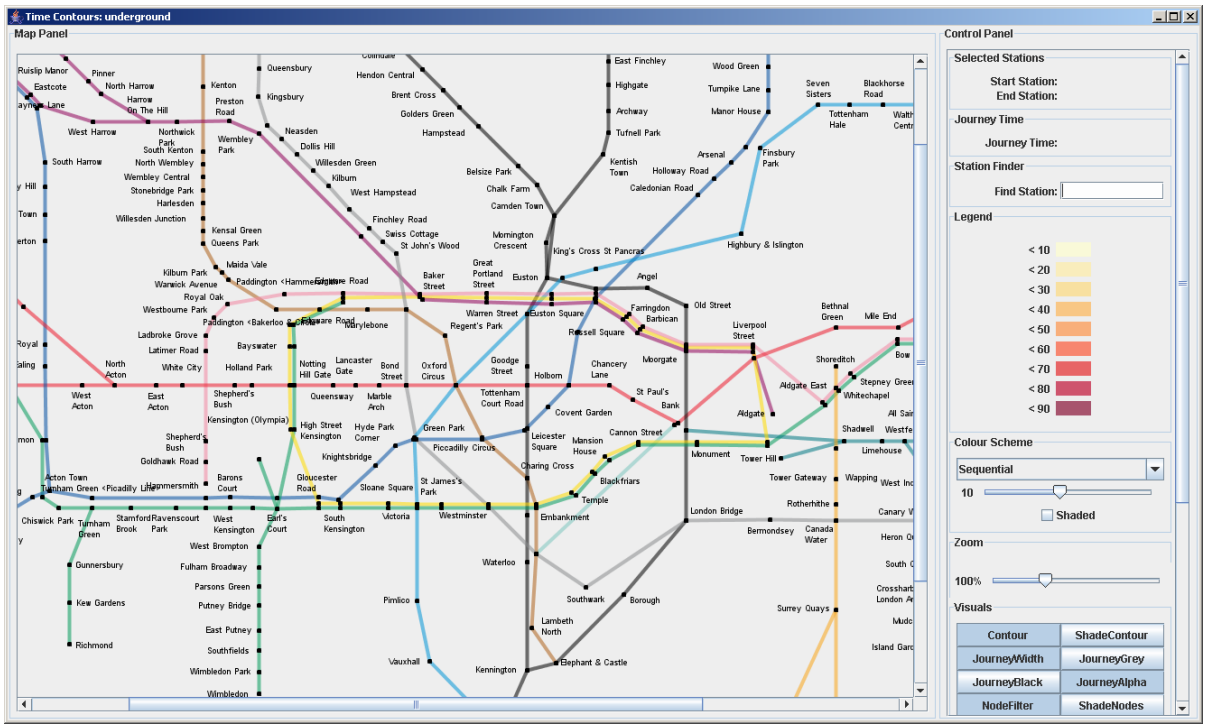


Figure 14 – The Underground Implementation of TimeContours

The application shows a large map view on the left hand side, and a control panel on the right hand side. The map is interacted with by left or right clicking nodes, or by hovering the mouse over them. The control panel includes various pieces of status information and tools to affect the display of the visualisation. Status updates include the currently selected start and end stations, the journey time for this transition, a search box for finding stations, and the current colour scheme legend. Tools meanwhile include a station finder, colour scheme changer, contour mode selector, line disabler, network display changer, graph filter manager and action toolset.

6.2 Generating isochrones

When a station node is selected, isochrones are generated and displayed on the map. Figure 15 shows the visual output when South Kensington is selected. The start station is selected with a black outline. The journey times to each station can be ascertained by comparing the colour of the contour they are placed on with the legend on the right. We can see that Piccadilly Circus is less than fifteen minutes away, and Oval is less than 30 minutes away.

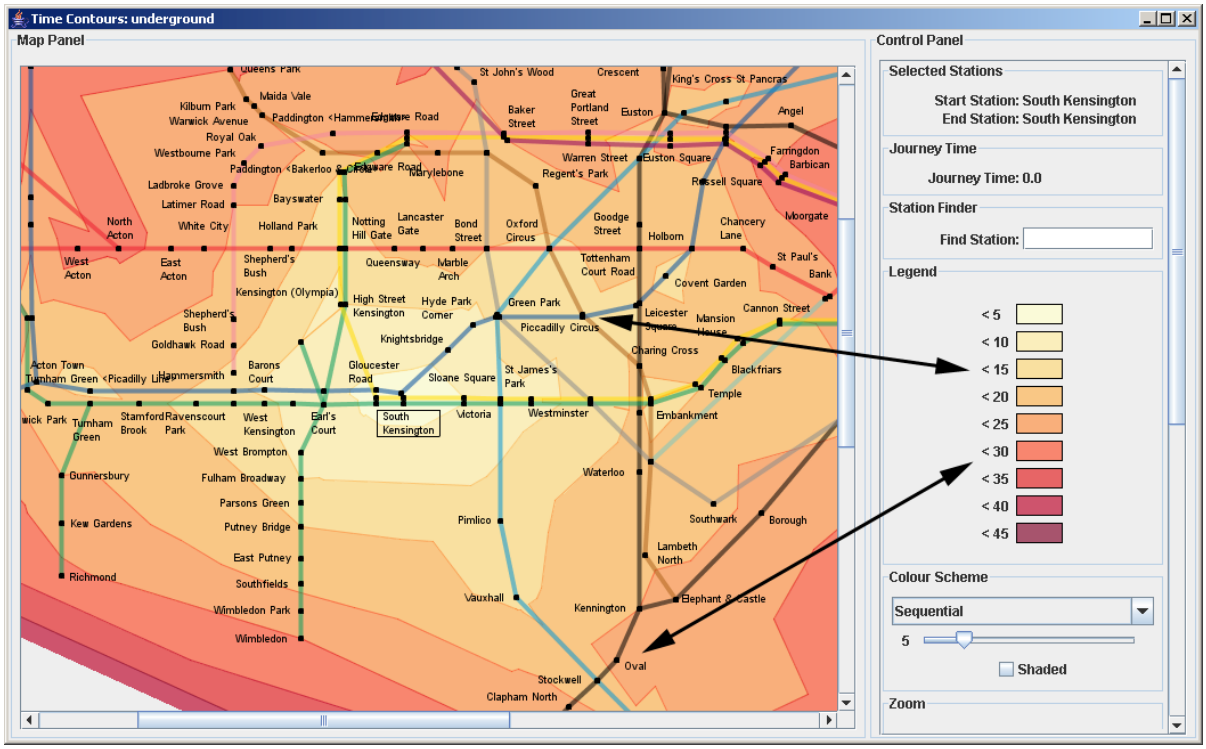


Figure 15 – Five minute contours from South Kensington

6.3 Determining the fastest route

To display the fastest route to another station, and the journey time for this, we simply mouse over the station in question. The route is lit up and the journey time in the control panel updates to reflect the change.

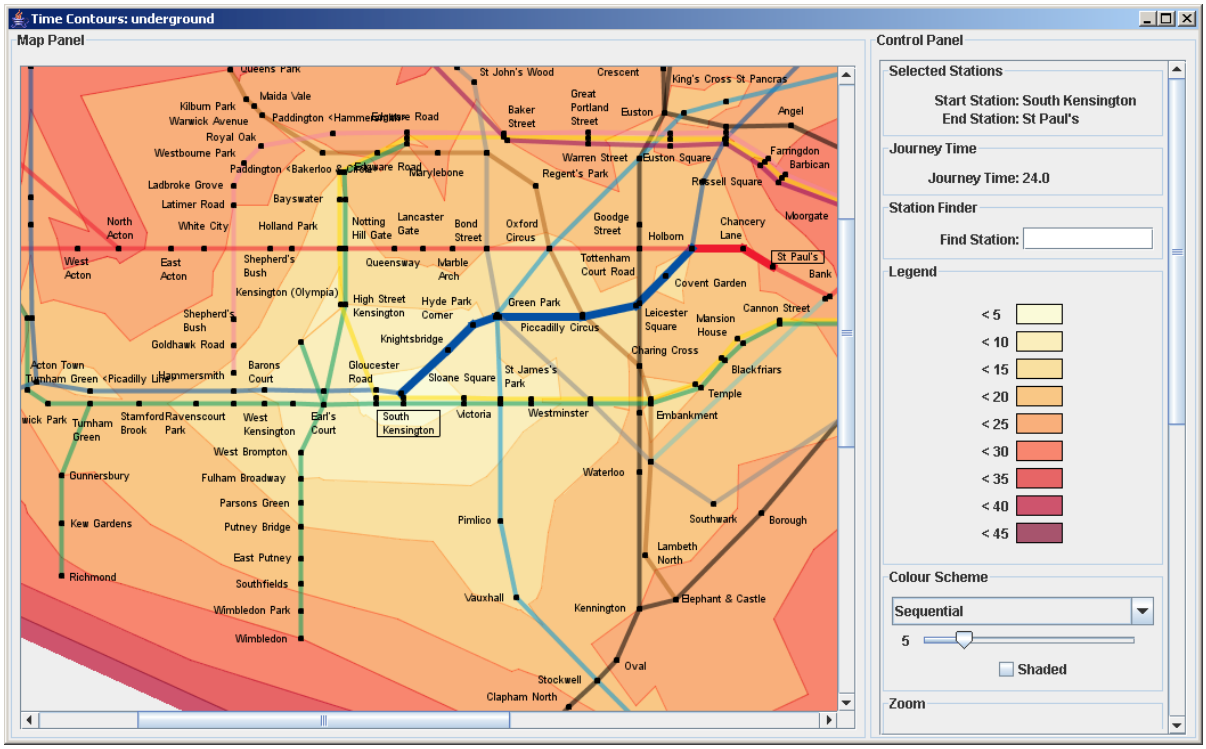


Figure 16 – Route from South Kensington to St Pauls

6.4 Finding Stations

Finding a station is simple with the Find Station tool. For example to find Baker Street we would type the name into the Find Station box on the control panel. As soon as letters are entered, matching station platforms are lit up on the map.

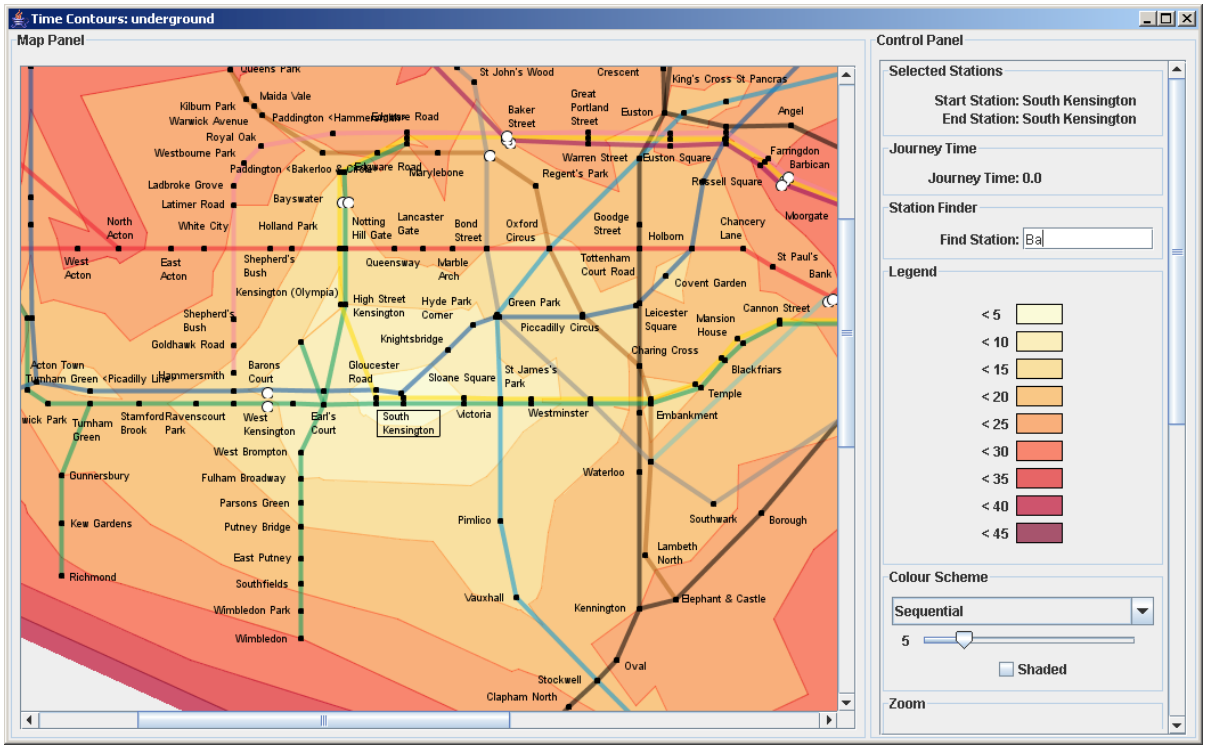
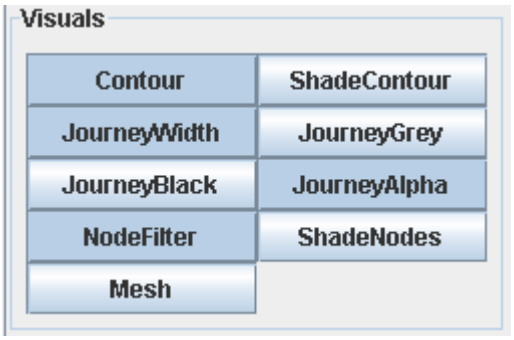


Figure 17 – Searching for stations beginning "Ba" with the Station Finder

Pressing enter meanwhile will select the first matching station as the Start Station, the Shift and Ctrl modifiers can be used to select the End or Action station instead.

6.4.1 Selecting visualisation effects



The visuals toolbox provides a set of toggle boxes to enable and disable different visualisation effects.

The visuals toolbox provides a set of toggle boxes to enable and disable different visualisation effects.

Contour – Adds an outline to the shaded contours, this brings out the boundaries of the shaded areas. Sometimes however they are not wanted, for example it is nice to be able to create a shaded effect without displaying the contours.

ShadeContour – Displays an alternative contour drawing technique, which is fast, but inaccurate at surface edges.

JourneyWidth – Highlights the currently selected journey by changing the width of the network edges that form it.

JourneyBlack – Highlights the currently selected journey by making journey edges black.

JourneyGrey – Makes all transport edges except those that form part of the selected journey render in greyscale.

JourneyAlpha – Makes all transport edges except those that form part of the selected journey alpha transparent.

NodeFilter – Highlights selected nodes according to their type. For example platforms are increased in size and changed to white for maximum visibility, whilst station names are given a box outline.

ShadeNodes – By default platforms are rendered in black. This option colours them according to the journey time and current colour scheme.

Mesh – Displays the triangulated mesh that is used for surface generation.

Note, see section 9.5.1 on page 62 for a discussion and examples of the different journey filters

6.4.2 Adjusting walking speed

Distances between platforms and foyers are modelled according to an average person's walking speed. This calculation can be adjusted by using the Walk Speed combo box in the Graph Filters panel. When this option is selected the visualisation is automatically updated. Other filters will be added to this panel as they are developed.

Figure 18 – The Graph Filters Sub Panel

6.4.3 Changing network state

Underground lines are assigned states by Transport for London according to the level of service they are running at. Currently these are not updated automatically, but by checking the TFL's realtime travel update webpage a user can obtain the status information and manually input it using the Line Disabler tool. To update a line's state, select the line using the left hand combo box, and the status using the right hand combo box. After this is done, click the set status button and the visualisation will be updated.

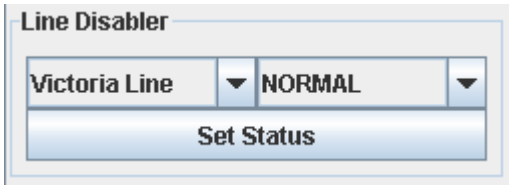


Figure 19 – The Line Disabler Sub Panel

For example, on June 12th 2006 the London Underground was suffering delays on many lines at once. Figure 20 shows how TFL represented this information on its website. This information was input using the Line Disabler sub panel and the visualisation updated. The effect in this example is dramatic, as the good service on the Victoria line stretched the contours out to the North East and South.

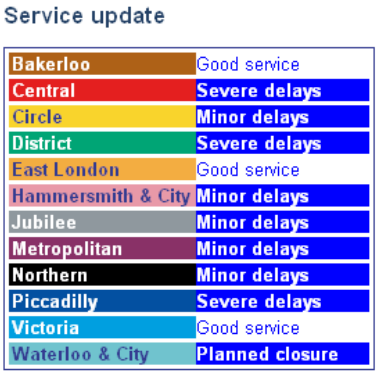


Figure 20 – TFL's Service Update on June 12 2006

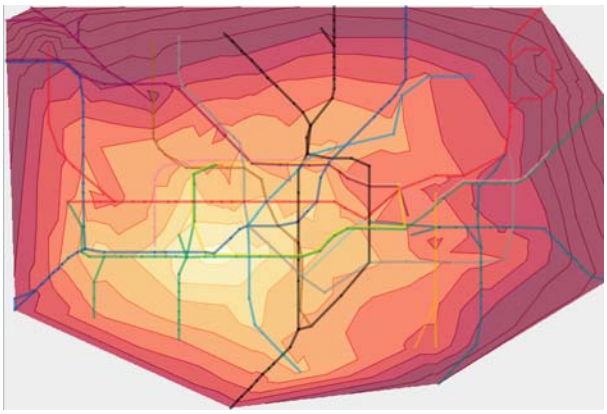


Figure 21 – Ten minute contours from South Kensington with good service on all lines

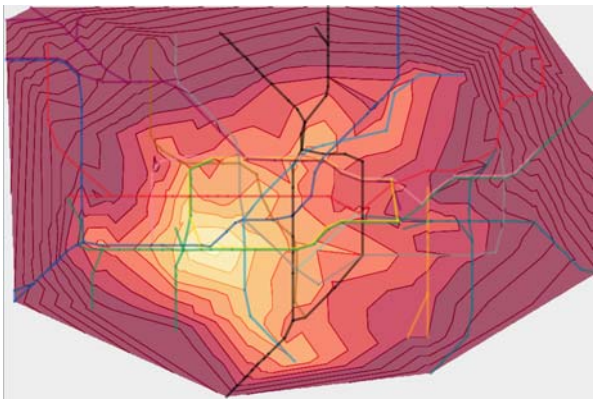


Figure 22 – Ten minute contours from South Kensington with service updates applied as per Figure 20

6.5 Open Street Map implementation

The Open Street Map implementation interprets xml files downloaded from the Open Street Map database. Support for downloading from the online server on the fly has not yet been implemented, but pre-downloaded files provide a proof of concept. The analysis provided uses a speed filter to estimate where you can reach in a particular time – the default speed is 5km/h, which is considered to be

the average walking speed. Currently no account is taken for the class of road, although this is because of the data rather than capabilities of the engine. The below diagram shows contours generated from Oxford street.

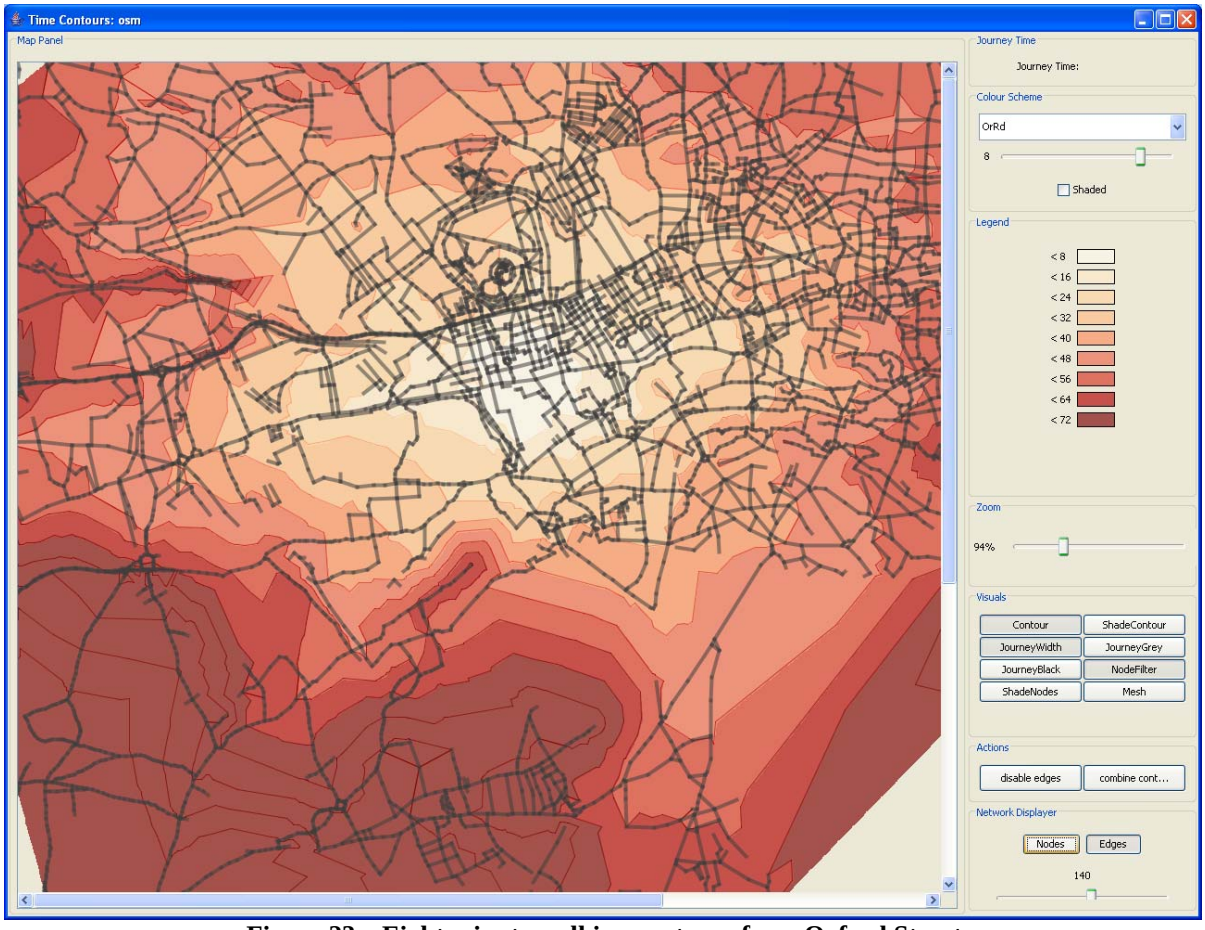


Figure 23 – Eight minute walking contours from Oxford Street

6.5.1 Changing the active colour scheme

For the above example we have used the colour scheme sub-panel to select a pallette that offers greater contrast against the dark roads. The panel can also be used to change the contour interval and use a shaded rather than block paint system.

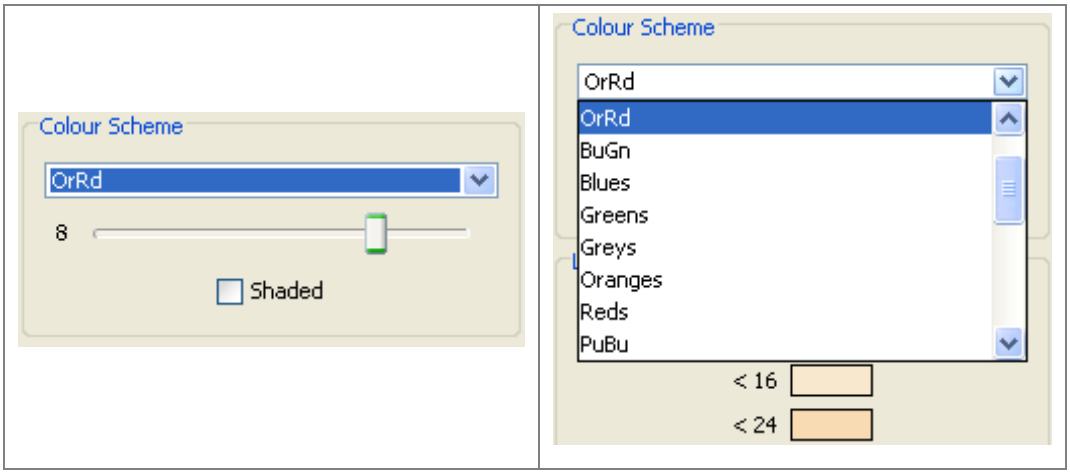


Figure 24 - The Colour Scheme Sub Panel

6.5.2 Adjusting road transparency

The roads can also hide details of the contours. To change the alpha transparency, or remove the roads from the view altogether, use the slider and toggle buttons on the Network Displayer sub panel. Although removing the roads altogether makes it difficult to get relate the contours to space, it can nevertheless be interesting to view them, or for exporting purposes.

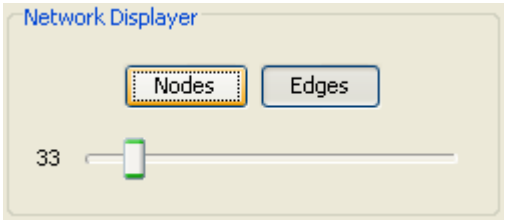


Figure 25 – The Network Displayer Sub Panel

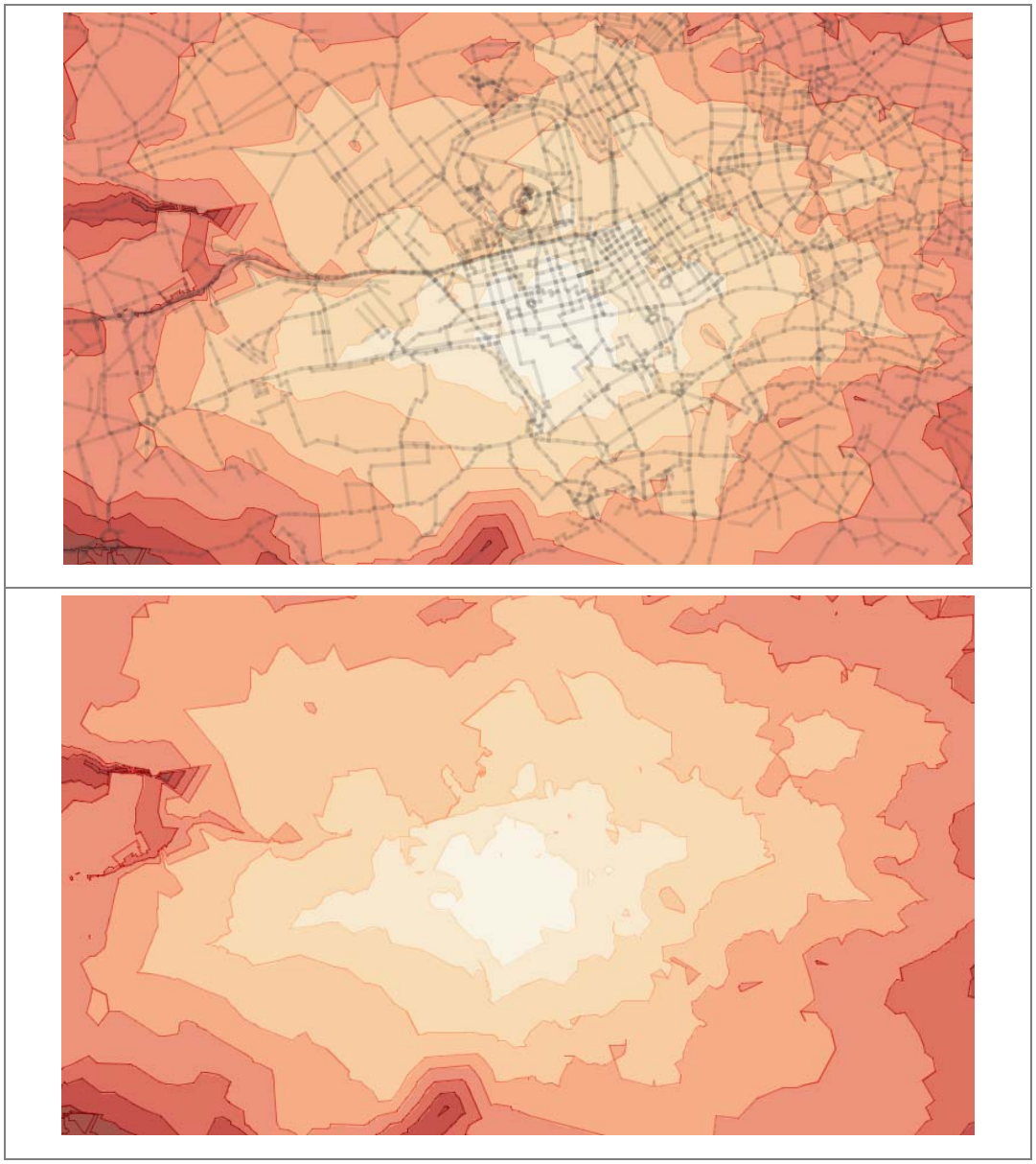


Figure 26 – Detail of Oxford Street contours with shaded and removed edges

7 System Architecture

The below diagram shows a simplified view of the process flow from data input at the top to the formation of contours at the bottom. Network data is firstly read in from some source, such as an XML file or Database, and used to construct a model of the network. When a start node is selected network analysis is performed to obtain journey time data to every node in the network, this is turned into a surface using one of two different techniques. Contours and cost based colour shading are then visualised from the cost surface. The network model is also used to construct a visualisation of the network itself, when an end node is selected this is filtered to display the journey between the start and end node, which is ascertained from the network analysis.

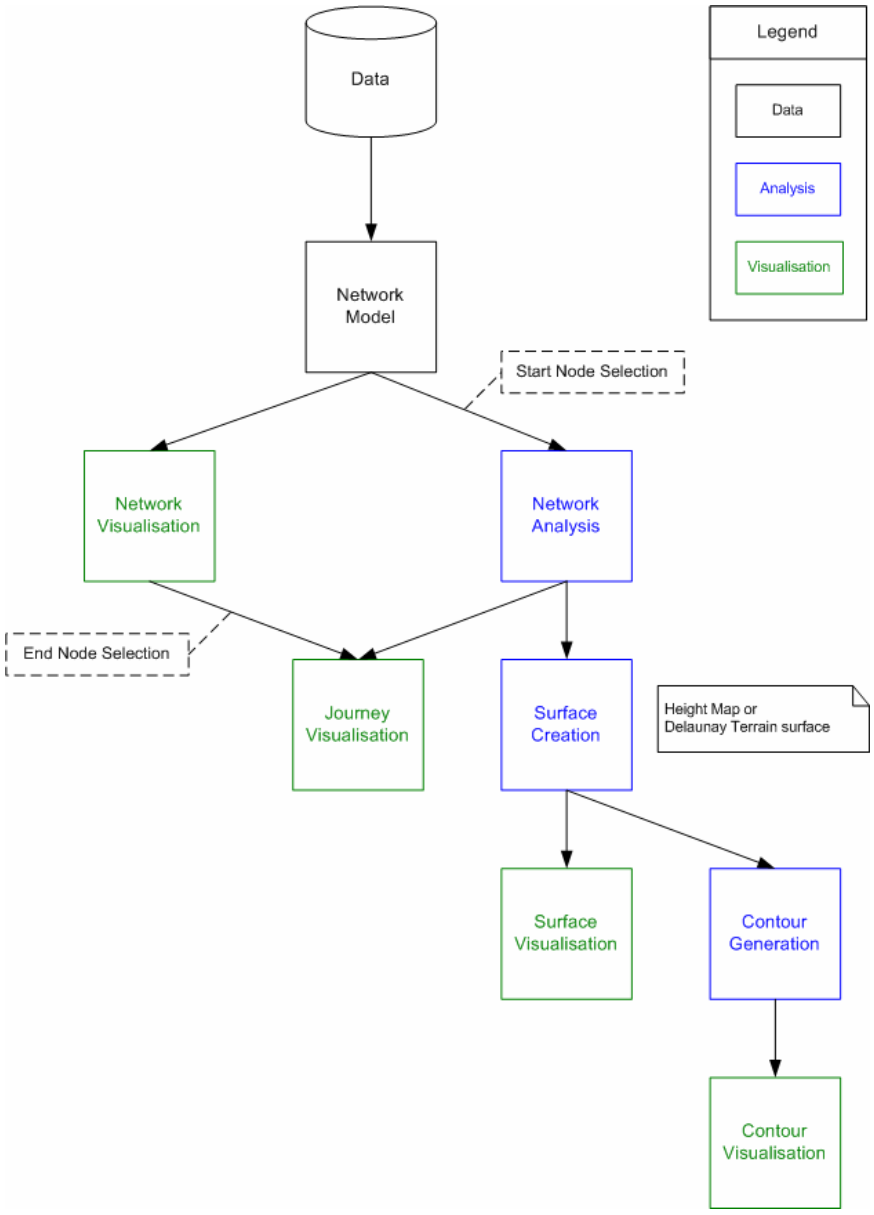


Figure 27 – TimeContours Workflow: From Data to Visualisation

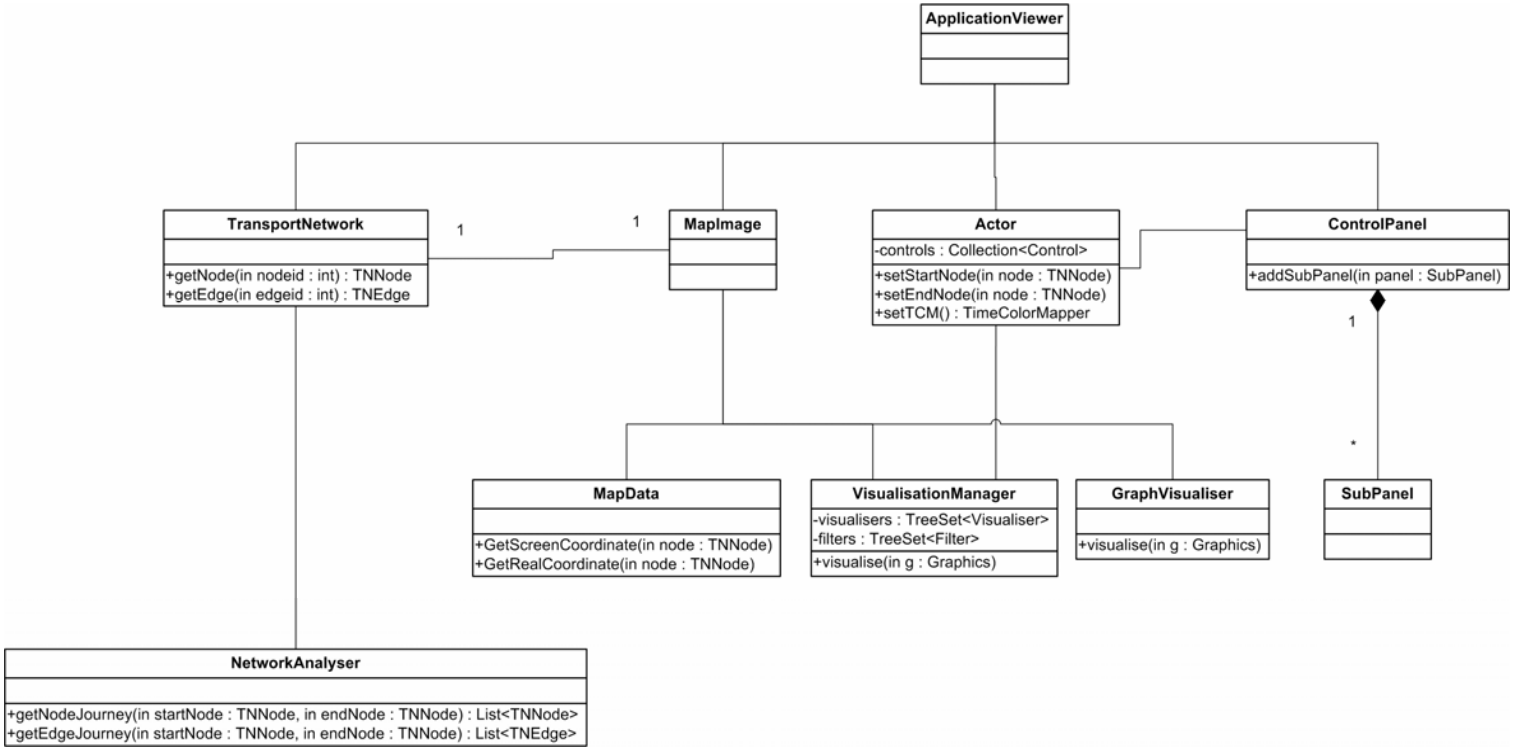


Figure 28 - Core classes in TimeContours

To implement the workflow described in Figure 27, including the graphical user interface and control mechanism, it was important to break the application down into separate modules that could be tacked in an independent manner. The main identified aspects of the application were data I/O, modelling the transport network, rendering the contours and network, and managing user interaction. The above diagram of the core TimeContours classes shows how these problems were broken down and how they interlink.

The *TransportNetwork* class holds the model of the underlying network. In our implementation this was built using the JGraphT graphing library. The class also contains a NetworkAnalyser object, which allows us to query the network and request network properties and routing information. This is discussed further in section 0

The *MapImage* class manages the display of the network and contours, and is made up of four major classes. The MapData class gives all the nodes in the network a location; by keeping this separate from the network model it becomes possible to have two different MapImages using the same TransportNetwork, but with alternative visual layouts. The GraphVisualiser manages the display of the network itself, painting edges onto the map and placing Graphical Node objects in the relevant positions. VisualisationManager contains a set of *Visualisers* which add details such as contours to the MapImage and *Filters* which adjust the appearance of Visualiser items. The GraphVisualiser is kept separate from the VisualisationManager since visualisers in the latter class are dependent on the user's state, whereas the GraphVisualiser is not. The MapImage class and its members are discussed in section 9

The *Actor* class provides a single point of entry for changes in state of a user. A change of state generally consists of choosing a different start node for the visualisations. The actor class relays the change in state to the VisualisationManager and ControlPanel – ensuring that all components of the application are updated correctly. More details of the control flow design are in section 10

The *ControlPanel* class provides an area to add information about the map, and tools to affect its appearance. The class is built from a collection of SubPanels, which are built and added by the Application Viewer. When a SubPanel is added to the ControlPanel, the ControlPanel transparently links it to receive state messages. This design allows SubPanels can be constructed in a reusable modular fashion with minimal understanding of the application architecture. More details on this can be found in section 10.2

8 Network model and analysis

In order to create a representation of time maps, it is necessary to be able to model traversal times in the network being represented. This model produces journey time estimations for all nodes in the network, and can be used in surface generation. The background research determined that a graph would be the best structure to use in achieving this. Our implementation uses a graph library, JGraphT, which manages putting together, holding and basic analysis of graphs. One of the main advantages of using this library is that the graph iterators used in the graph algorithms includes significant optimisations, in particular in using a Fibonacci Heap. This heap reduces the search time for algorithms such as finding the minimal spanning tree from $O(V^2)$ to $O(E + VlogV)$[33], where E is the number of edges in the graph, and V is the number of vertices.

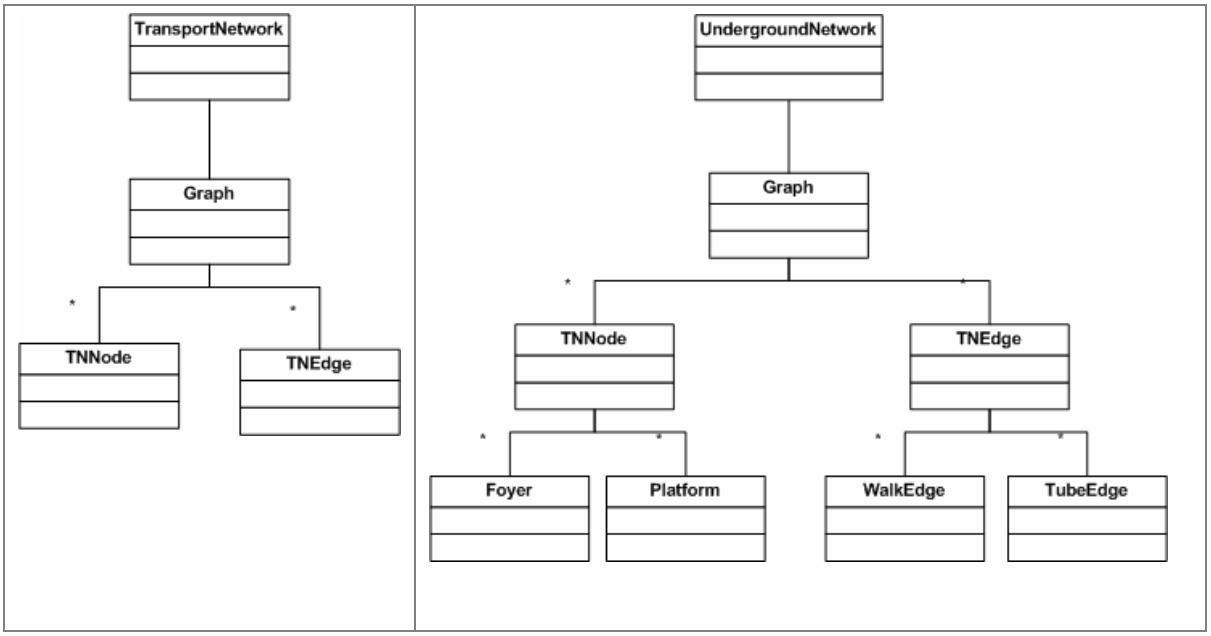
8.1.1 Obtaining the minimum spanning tree

JGraphT, includes various "graph iterators", which go through each node in a particular order. One of these iterators is a closest first iterator, which on each next() request returns the node with the next shortest journey time to reach. The iterator also returns metadata including the journey time and edge used to reach the node. The nature of this process means that for each node returned we can be sure to have a value for the fastest possible journey time to that node. If a shorter route existed then it we would have to go via another node with a shorter journey time to reach, but since the present node is the one with the next least journey time, this other node cannot exist. This process effectively forms the minimum spanning tree, and is the first stage in Dijkstra's algorithm. The cost information is put into a NetworkJourneyData class, which can be interpreted by any surface generation algorithm to generate estimates of journey times for points on the surface between nodes. The different solutions to this problem are dealt with in the Cost Surfaces section on page 52. The second stage to Dijkstra uses the minimal spanning tree to construct the fastest path between two nodes in the network, this is used for journey representation, which is discussed on page 62.

8.1.2 Modelling different network types

The network model in TimeContours is held in the TransportNetwork class, which acts as a container for the graph, analysis classes and graph filters for adjusting behaviour. The graph itself is constructed from an interrelated set of nodes and edges, which are packaged into two classes: the TNEdge and TNNode. These hold basic functionality that will be required for all types of network, such as whether they are drawn on the map, are included in contour calculations and their colour. Other networks types require further information; this is catered for by extending the TransportNetwork class and the graph component classes. For example, the UndergroundNetwork class includes extra information about the tube lines in the network, and its graph components are built out of Platforms, Foyers, WalkEdges and TubeEdges which extend TNNode and TNEdge.

Platforms provide information about the line they are a part of, whilst the different class types means analyses can differentiate the two.

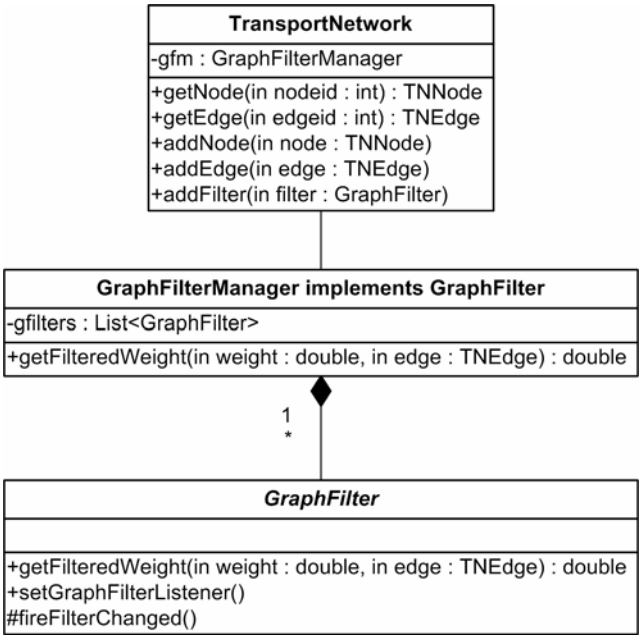


This structure means that generic classes can be built for displaying and analysing all types of networks, without losing the capacity to treat different network types differently.

8.1.3 Graph Filters

The graph based model of the network combined with traversal algorithms provides a good method of modelling a network under standard conditions. However, one of the most important requirements of the TimeContours system was that it should be able to model change in the system. The visualisation can then reveal how disruptions to one area of the network affect accessibility in general. To implement this on the graph we developed the concept of a Graph Filter, which transforms the journey times between nodes according to the state of the interconnecting edges. The solution means that disruptions can be simulated and then rolled back by simply adding and removing filters, without affecting the original underlying model. The filter architecture allows different Graph Filters to be linked together, so that the output of one is fed into the input of the other. The design allows us to compose transformations using high level constructs such as accessibility needs and network status.

JGraphT's graph traversal algorithms call each edge's getWeight() function to calculate minimal journey times to each node in the network. We ensure that analysis of the network uses filtered weights by overriding the edge's getWeight() function and passing it through the Graph Filters. To do this each edge is given a reference to a GraphFilterManager when it is added to the graph, which performs the necessary filtering before returning a weight.



Each graph filter has a `getFilteredWeight()` function, which is passed the edge being filtered, and a working weight value. The function returns an adjusted weight value according to the properties of the edge in question and the characteristics of that filter. The GraphFilterManager is itself an example of a filter, which contains an ordered list of graph filters. When its getFilteredWeight() function is called, it simply passes the passed weight and edge through each of the graph filters in turn

```
public double getFilteredWeight(double weight, TNEdge edge) {
      double workingWeight = weight;
      for (GraphFilter gf : graphFilters)
            workingWeight = gf.getFilteredWeight(workingWeight, edge);
      return workingWeight;
}
```

Figure 29 – GraphFilterManager's implementation of getFilteredWeight()

The final important aspect of the GraphFilter is that it alerts a GraphChanged listener when some aspect of the filter is changed. This means that we can detect when a change is made that means the current visible (and cached) surfaces are no longer correct, and need to be recalculated.

The graphFilter architecture makes writing powerful GraphFilters a very simple process. A good example of their use is for calculating delays on the tube. Transport For London issue alerts for sections of the tube, defining them to be in any of four states: normal, delays, major delays and closed. Each TubeEdge in the network is therefore in one of these states, and the GFUGTubeDelays filter will adjust the journey time for traversing a TubeEdge according to its status.

```
public double getFilteredWeight(double weight, TNEdge edge) {
      double cost = 1;
      if (edge instanceof TubeEdge) {
```

```
            TubeEdge e = (TubeEdge) edge;
            switch (e.getStatus()) {
            case NORMAL             : cost = 1                ; break;
            case DELAYS             : cost = 1.5              ; break;
            case MAJOR_DELAYS       : cost = 2                ; break;
            case CLOSED             : cost = Double.MAX_VALUE ; break;
            }
      }
      return weight * cost;
}
```

Although this is a simple and crude model, it still produces some interesting effects. The suggested route between Euston and South Kensington stations provides an interesting test case

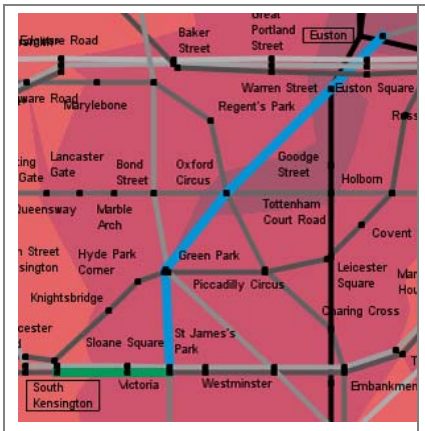


Figure 30 – Route between Euston and South Kensington, no delays – 26 mins

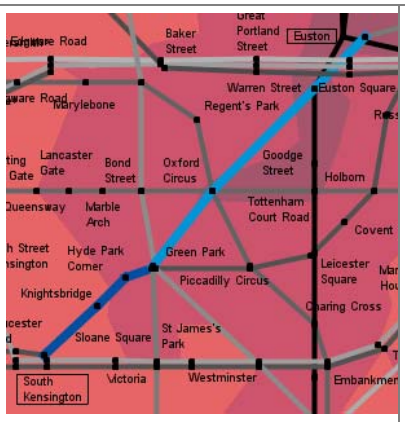


Figure 31 – Route between Euston and South Kensington, delays – 29.5 mins

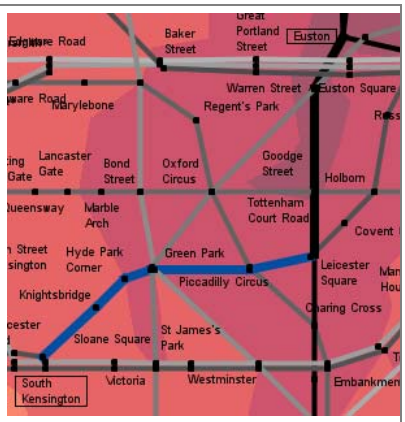


Figure 32 – Route between Euston and South Kensington, major delays – 30 mins

Some other example GraphFilters are:

GFNodeDistance – takes edges with source and target nodes having positions in Longitude/Latitude format and returns a weight based on the distance between the two.

GFDistanceToTime – transforms a weight in metres into a time according to a user's speed.

GFUGStopChanges – Restricts the number of line changes that can be made on the Underground.

GFUGWalkSpeed – adjusts the weight of WalkEdges according to a user defined walking speed.

GFUGAccesibility – adjusts the weight of WalkEdges depending on whether the edge is easily traversable by people with mobility problems. The weight change that the filter performs depends on the importance that the user specifies for avoiding non-accessible WalkEdges. For example a wheel-chair user might request to avoid them at all costs, whilst a person with a push-chair might define them as a minor annoyance.

8.1.4 Extending the network model for the underground

Creating an effective data structure to represent the underground is not as simple as it might appear from looking at the tube map. At first glance the network looks like a simple weighted graph, with stations representing nodes and the connections between them being represented by weighted edges. The weights on each edge represent the journey time between the stations. Unfortunately this fails to model two important aspects of the network: when inter-line transfers are required and how long these transfers are, and when intra-line transfers are required. Transfers are a significant time-cost when travelling on the underground, and must be included in the model if an accurate representation of journey times is to be acquired.

We look at solving these problems with reference to Earl's Court as a case study. This station was chosen as it includes both multiple lines and is the hub of the District Line, probably the most complex line on the tube. A diagram of Earl's Court and its adjacent stations is shown below:

Figure 33 – Earl's Court and adjacent stations only

8.1.5 Determining when inter-line transfers are required, and how long they are

Inter-line transfers are important because they incur two costs: walking time between platforms and an added waiting time for a new train. Both of these are significant, the walking time between the Jubilee and Piccadilly lines at Green Park for example is about five minutes. To model this in a graph the station must be split up into multiple nodes: a platform node for each line and a foyer node for the entrance to the station. To complete the model we add edges between each of these nodes to signify transfer times.

8.1.6 Determining when intra-line transfers are required

From Figure 33 we can extract the journey sections in which no change is necessary. Note that this leaves many pairs of stations such as Kensington Olympia and West Brompton between which an intra-line transfer is required.

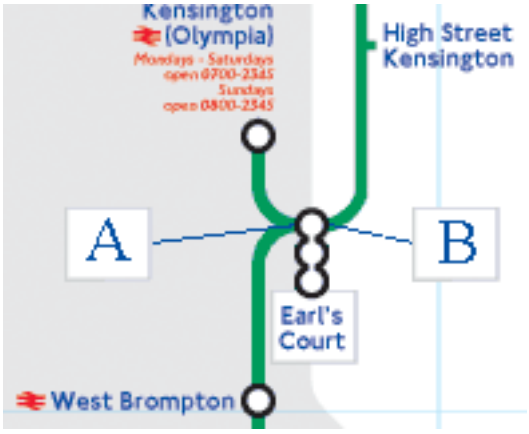
Station A		Station B
High Street Kensington	◀▶	Kensington Olympia
High Street Kensington	◀▶	West Brompton
West Kensington	◀▶	Gloucester Road
West Brompton	◀▶	Gloucester Road
Barons Court	◀▶	Gloucester Road

Figure 34 - Direct Routes Through Earl's Court

With the redesign given above for inter-line transfers we would have one District platform node with edge connections to District Line platforms at all five of the above District Line Stations. This model allows journeys between any of these stations through Earl's Court as readily as any other pair. We need to add something to restrict certain transfers.

The Sided-Vertex

In order to do this we could extend the model of the graph and create a new version of the vertex, called the sided-vertex. A sided-vertex has two distinct "sides", which we will label "A" and "B". Every edge that connects to a vertex connects to one of these sides, note that there can still be many edges connected to each "side". A journey through a sided-vertex must then leave that vertex from the opposite "side" to that it entered. This system models the way the London Underground map displays this information. The below diagram shows its implementation for the stations Kensington Olympia, High Street Kensington and West Brompton:



In order to model the frequency of trains leaving a station from a particular side, each connection on a side can be given a weight. In the above example if we wanted to indicate that 90% of trains from High Street Kensington go on to West Brompton we would give Kensington Olympia a weight of 0.1, and West Brompton a weight of 0.9.

8.1.7 The Redesigned Earl's Court

Modelling Earl's Court with our new system still requires some work. It turns out that Earl's Court is one of a handful of stations requiring two platforms on the same line: one for the High Street Kensington to West Brompton section of the

line, and another for the West Kensington to Gloucester Road section. This directly reflects the layout on TFL's underground map. The system works elegantly with intra-line transfers being carried out using the sided-vertices and transfers between the two sections of the District Line being modelled using the inter-line mechanism.

9 The Visualisation Engine

Visualisation is a complex problem, especially when there are a wide range of different factors contributing to how something should appear. There are two subtly different problems in the display and update of the map. Firstly there is the creation and painting of objects on the map – this is handled by *visualiser* classes. Examples of these are the network lines, contour lines and nodes. These objects however often need to change in appearance, but not structure, for example if you move the mouse over a different node, the edges between them representing the ideal route may change in form, or when you click on a node the nodes change in colour to represent the amount of time it takes to travel to that node. These changes are managed by *filter* classes.

The VisualisationManager holds a set of both filters and visualisers. When changes in state occur (i.e. a node is selected by the user), the manager passes this message on to each of the filters and visualisers, which use the information to update their status. The visualisation manager also contains a visualise method, which is called by the map image with its Java2D Graphics object whenever it is painted. The VisualisationManager passes this to each visualiser in turn so that they can use it to paint on the map. For example the ContourVisualiser uses the Java2D Graphics API to paint a contour line for each contour. The solution gives visualisers a large degree of flexibility in how they update the map.

Another important aspect of the design is prioritisation. For visualisers prioritisation allows us to set the order in which visualisations are applied – for example we want contour lines to be painted on top of contour shading. Visualisations thus form "layers" in the traditional map sense, which are ordered according to their priority. Filters meanwhile are not parsed when the scene is updated, and their changes are static. We therefore implemented a setting prioritisation scheme, which allows us to control for each setting type which filter's change takes effect. For example, a filter may be being applied to make a line with delays appear in a shade of red. Another filter for displaying the fastest route causes to the line to be painted black. The filter prioritisation system allows us not only to define that the black setting should override the red, but also that if the journey filter is removed, the line should revert to displaying in red rather than its original colour.

The filters allow separation of concerns. One user may want journey edges to be represented as thicker lines, another as black lines, and another as both. The filter architecture means we can achieve the latter behaviour through the composition of the previous two filters.

9.1 Positioning the nodes

The visualisation of both the network and the surfaces are dependent on being able to give a specific position to each node in the network. To do this there were various choices: basing it on their geographic location, on their location on the standard tube map, or following Carden and Karlin's approaches in reforming their spatial relationship altogether. We believe that the reforming of the underground layout was the greatest weakness in both of these representations,

since the lack of familiarity in the network layout amplifies the difficulty in learning to interpret the isochrones. However, there is clearly value in being able to build a time map for both the cartographic and geographic representations of the underground. The former is useful for gaining insights into the underground network itself, and the latter can be used to discover its impact on accessibility in London itself. In response to this realisation we have separated the node position data from the network itself, allowing us to support cartographic, geographic and dynamic layouts for all network types. Furthermore, when multiple map displays are introduced it will be possible to use the same network to generate surfaces for different layouts concurrently.

9.2 Painting lines

Lines need to be painted by both the contour and network visualisers. To solve the visualisation problem we produced a LineVisualiser class to handle the painting of lines. Lines to be painted are held as LineSegments and added by an addLineSegment() call. The edge and contour visualisers both extend this class, separating the problem of painting and line management

9.3 Displaying Nodes

Nodes require the ability to interact directly with the user. If they were to be drawn on the surface manually then we would also need to implement our own event capture mechanism for responding to mouse clicks and interaction. Nodes are therefore implemented as Swing JLabels. We therefore do not need to draw them ourselves (the purpose of the visualiser), but we do need to update its appearance (see Filter Design).

9.4 Cost Surfaces

The network analysis described in section 8 provides us with journey time data for every node in the network. This can be combined with node location data to give us elevation information for points on the map. In order to visualise this information we need to create a cost surface, so that an estimate of cost can be acquired for any point on the map. In this project we looked at two different schemes for doing this: firstly using the concept of the relief terrain from GIS and secondly building a height map by treating nodes as potential exit points from the network and estimating journey times for the surrounding area.

Each approach is better suited to different situations. We have used the terrain model for the Open Street Map prototype, since when walking or driving one can potentially stop at any point on the map – the concept of an “exit point” does not exist. It was also the approach of choice for the cartographic representation of the underground. In this situation, where nodes are positioned on the map according to connectivity rather than geographic location, points in between the stations do not map on to visitable locations. The contours therefore serve to provide a lucid representation of how long it takes to reach nodes, rather than the space between them. The diagram below compares the two representations of ten minute contours from Oakwood station on the Piccadilly line. In the terrain based image on the left, it is possible to get a fast snapshot idea of how far away different stations on the network are. King’s Cross for example in the middle of the picture

is reachable within 40 minutes, whilst the outer extremities of the Northern Line are over an hour away. The same cannot however be said for the journey based representation on the right. Although the immediate vicinity around the Piccadilly line is noticeably darker and correspondingly closer in time, it is difficult to distinguish between stations in the rest of the network. The multitude of circles fills the scene with too much information, and the relation between colour and time is largely lost.

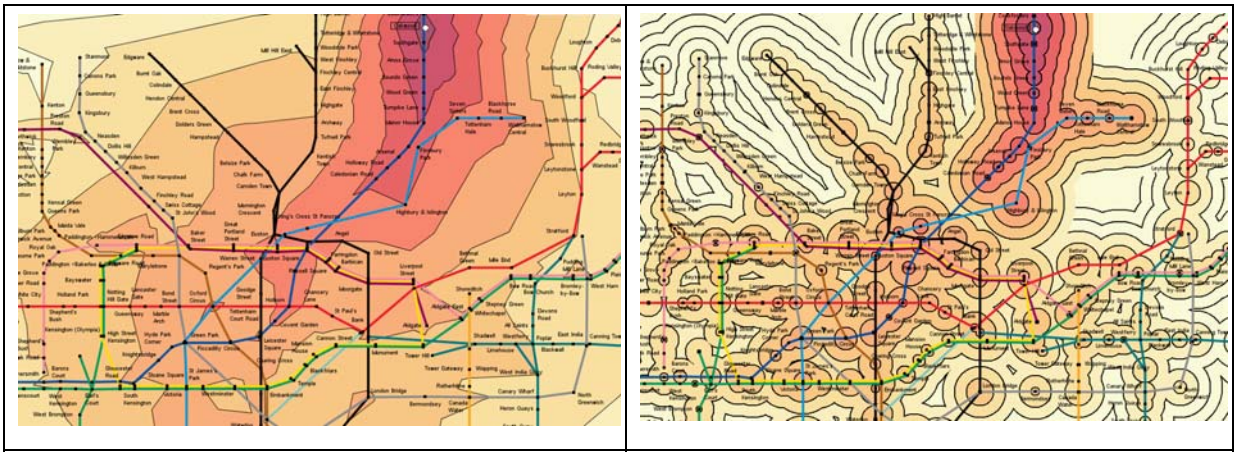


Figure 35 – Comparison of surface representations on a cartographic underground layout using 10 minute contours about Oakwood

When the underground is projected onto a geographic layout however the terrain based contour approach is less effective. The terrain based representation (Figure 36, left) makes the misleading suggestion that areas between Oakwood and Epping in the top right of the map are relatively accessible. For example the midpoint is marked as being reachable within 45 minutes. Such a journey by underground however is clearly not possible. The representation on the right, calculated using a height map combined with an estimated walking speed of 5 km/hr from each station node gives a better idea of the accessibility of areas of London via the underground.

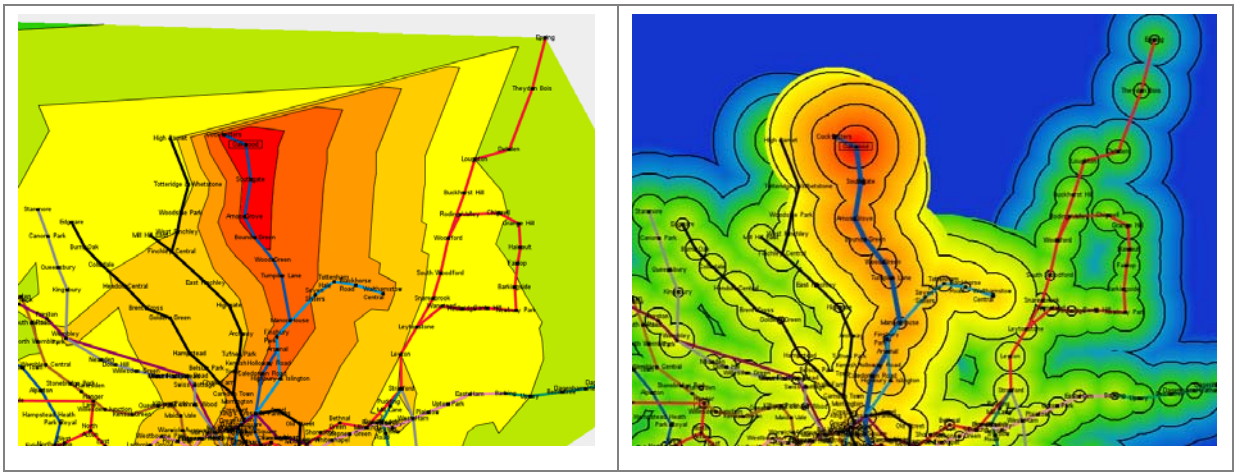


Figure 36 – Comparison of surface representations on a geographic underground layout using 15 minute contours about Oakwood

9.4.1 Building a Terrain based elevation surface using triangulation

Terrain based surfaces consist of a set of points joined together by edges to form a collection of triangles. Every node is given a cost value such as elevation or journey time. Each triangle then becomes a facet in a three dimensional triangulated mesh. Triangles are generally used for this process since they can model any three dimensional shape, whilst other polygons such as squares cannot. Every point in the map is inside a particular triangle, or on its edge – and its elevation can be calculated using vector maths and the known cost values of the triangle's corners

Figure 37 below shows the three possible triangulation schemes that can be used: a regular uniform grid, a semi-regular grid and a Triangulated Irregular Network (TIN). The regular uniform grid provides a simple easy to analyse model, but can require a large number of points to create a representative terrain. The TIN version meanwhile is much more flexible, allowing points to be placed at any position on the surface, and for complex surfaces to be represented with comparatively few nodes. The semi-regular grid provides a compromise between the two, with the grid structure simplifying analysis, and irregular aspect increasing the range of terrains it can model. Since the positioning of nodes in a transport network is arbitrary, and not restricted to a grid layout, the TIN is the most suitable type of surface to use.

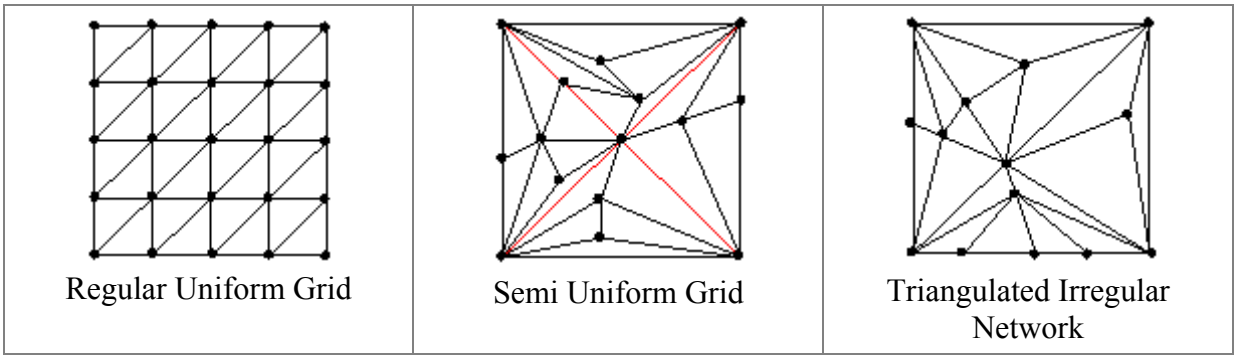


Figure 37 - Different Grid Constructions

To construct the TIN we use a technique called Delaunay Triangulation. Delaunay meshes fulfil the Delaunay Property, which enforces that no triangle has points inside its circumcircle[34]. The technique maximises the minimum angle of all the triangles in the triangulations, and so avoids stretched triangles. The Delaunay mesh can be generated by first constructing the Voronoi polygons for the point set. Voronoi polygons are tessellated polygons each of which contains one of the coordinates in the set. The polygons are constructed so that all points inside a polygon are closer to the generating point than any other on the map. An example of such as set of polygons is shown below.

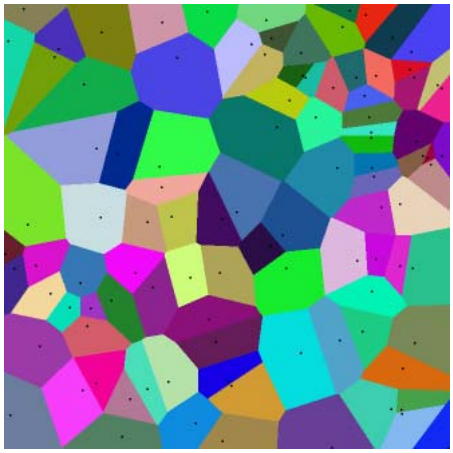


Figure 38 – Voronoi Polygons enclosing points in a network, all points in a polygon are closer to the enclosed point than any other[35]

The Delaunay triangulation is then constructed by connecting all points which share a common Voronoi tile edge together. This can be seen in the following diagram.

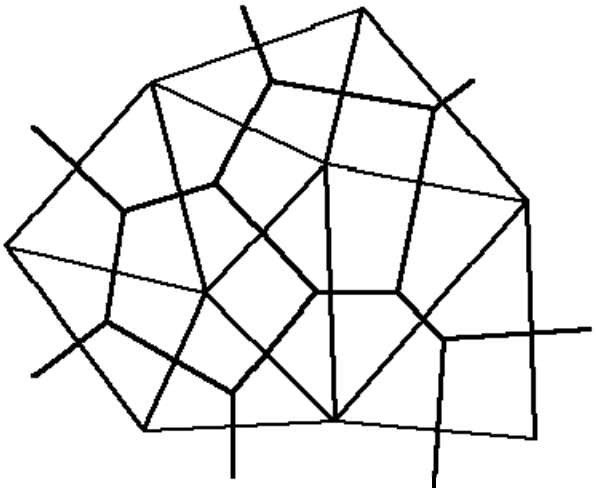


Figure 39 – Voronoi polygons (thick lines) overlaid on Delaunay triangles (thin lines) [36]

To complete the surface we simply assign each node an "elevation" using the NetworkJourneyData obtained from our construction of the minimal spanning tree.

Generating Contours

Contours can be generated from the cost surface using a simple plane intersection technique. For each contour a plane is constructed at that contours elevation, wherever this plane intersects the cost surface we draw a contour line. To implement this we used an adapted version of the CONREC contouring algorithm by Paul Bourke[37].

Figure 40 – Contours generated on the Underground

The contour generation scheme using Delaunay triangulation provides us with a set of lines coupled with their elevation. Our first attempt at drawing the contours on the map simply painted these onto the map, whilst assigning the colour according to the current elevation and shading scheme. This was implemented by extending the LineSegment class, allowing us to use the same visualiser module as for lines and the surface mesh. Whilst this approach did express the desired information, it failed to do so with clarity. We can see for example that King's Cross can be reached faster than Hyde Park Corner, but it is difficult to get a general feel of the time information. For this we needed to shade the internal regions a colour, as well as just the contour boundaries.

The contour lines computed so far were not directly amenable for generating shaded regions. The internal data structure only provided us with a set of lines at particular elevations, but no information about how they are connected together, and the spaces that they enclose. We investigated three different solutions to this problem.The first approach is to use a block fill algorithm, as used by painting packages to fill regions. Whilst this sounds conceptually simple, the problems of finding regions to fill and identifying the elevation for these regions make for an ineffective approach.

An alternative system was to join the individual contour segments together to form polygons which can be filled using the fillPolygon() method. Forming these polygons is complicated because the lines do not form perfect loops. Flat areas will produce triangles, whilst ridges will produce lines, even though neither side of the line is an area that needs to be filled. Many of the undesired contour lines can be removed using the concept of "hard" and "mass" edges, which is described in the paper *Correction of Horizontal Areas in TIN Terrain Modelling*[38] by Barbalić and Omerbegović. Contours going off the edge of the surface also do not form loops and finally holes within contour regions cause problems which cannot be easily solved with polygons. A single point can also be part of multiple contours, and deciding how to trace it to form separate polygons is complicated.

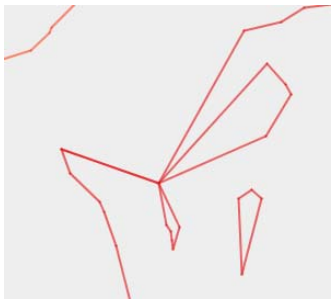


Figure 41 – Touching Contours and holes

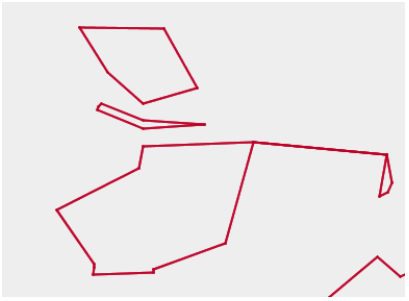


Figure 42 – A ridge

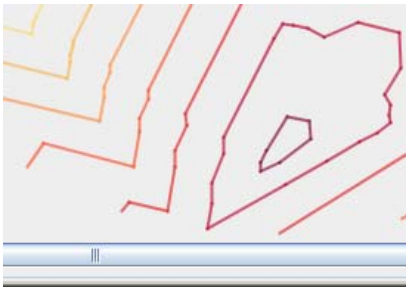


Figure 43 – Off the edge of the map

Attempting to form polygons from contour lines

After removing some of the loose and incorrect contours, we attempted two different solutions for joining the contours together. The first algorithm takes a set of linked contour segments and a set of unlinked contour segments. The initial linked contour set is a single segment taken at random from the unlinked set. The unlinked is then searched for a segment that could be joined on to either side of the linked set. Once such a segment is found, it is moved to the linked set and the process is repeated until either no segments are found, or the linked contour set formed a loop. A completed set is saved as a polygon, and the process repeated with the remaining contour segments. The algorithm was fast enough for acceptable use, but produced incorrect results where flat areas occurred, or at the edge of the map.

A more sophisticated algorithm modelled the contours as a graph and used Fleury's algorithm to identify cycles. This resolved some of the problems around flat areas, but proved to be too slow to be an effective solution. It also failed to deal with the problem of how to complete contours at map edges.

Another problem with the polygon approach to rendering contours is that gaps inside a polygon are difficult to represent. The polygon approach generally relies upon painting the most distant contours first, and layering the "closer" contours on top, much like an architect would build a terrain model. Holes inside a polygon however cannot be represented, and so will be painted the colour of the surrounding polygon. Urs-Jakob Rüetschi at the University of Zurich has developed the concept of a "Supergon" to provide a system for building polygons with holes in them[39], but its implementation would only have overcomplicated our solution.

Figure 44 – Contour Hole Correctly Rendered

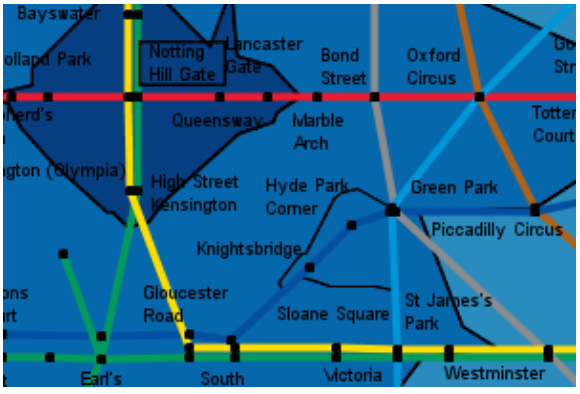


Figure 45 – Contour Hole Incorrectly Rendered Using Polygon Approach

Shading by painting each pixel according to elevation

The contour joining approach proved to be intractable, and so we turned to alternative techniques. 3D surfaces are normally rendered by painting each triangle in the surface in turn, or by stepping through each visible pixel and painting it according to its elevation. We do not want to paint each triangle a fixed colour, since many of the contours split the triangles in our surface in two, so the option of painting each pixel was most appealing.

In order to calculate the elevation for a particular point in a surface, it is necessary to find the triangle in the surface that the point is within. Simple vector mathematics can then be used to calculate the elevation. The surface produced by the Delaunay scheme however is a Triangulated Irregular Network rather than a Uniform Grid, so finding the triangle from a coordinate reference is more difficult. We can avoid having to locate this triangle by painting each triangle in turn, rather than each pixel.

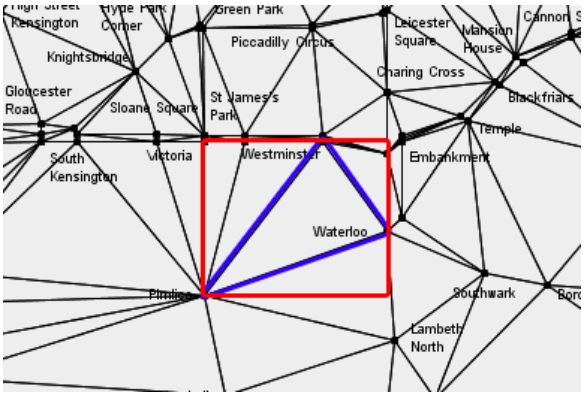


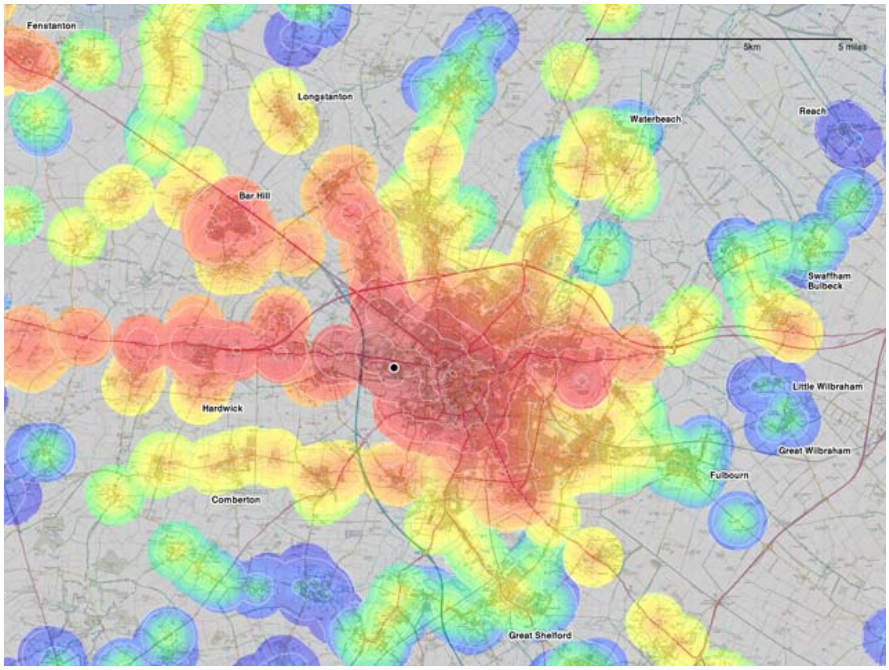
Figure 46 – Detail of the Underground Delaunay Grid

The above diagram shows a small section of the Delaunay grid on the Underground, with the triangle (in blue) connecting Westminster, Waterloo and Pimlico. To paint this triangle we need to iterate through each pixel in the triangle, calculate the elevation and use the current colour scheme to select a colour. A naïve way to do this it to firstly find the bounding box of the triangle (in red) and step through each pixel in the box checking for each one whether it is

inside the triangle. This approach requires us to search a large proportion of the area surrounding the triangle. Instead we start our search at the Westminster point and paint that pixel, we then move down to the next line, and starting at the point with the same x value as Westminster, we scan out to the right painting each pixel as we go until we are outside the triangle, and repeat this process to the left. We also take note of the x values at which we hit the right and left edges of the triangle, and the midpoint of these is used to start the search on the next scan-line down. This process allows us to paint the triangle without having to search unnecessary pixels.

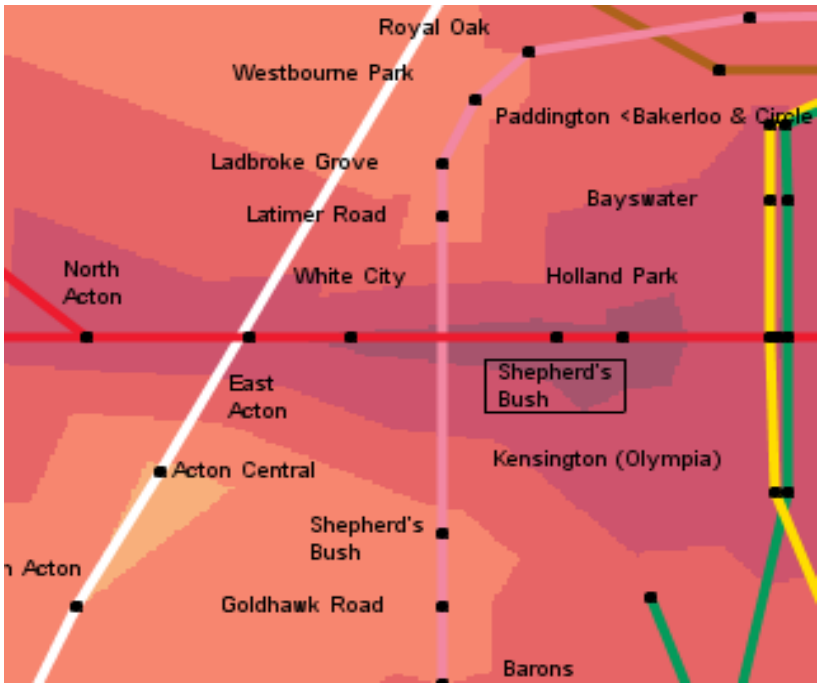
Contour Island Problem

A contour island is a contour enclosing areas that are reachable within a certain time period, but that does not enclose the start point. The MySociety implementation provides a good example of the phenomenon:



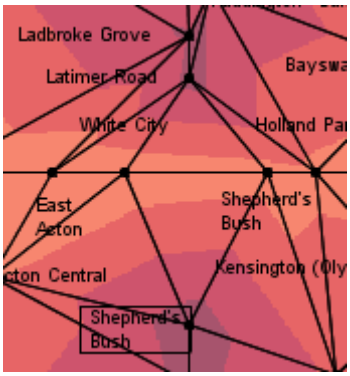
The islands are a true representation of the data and are desirable when on a geographically based layout. However, the underground example uses contours to express an idea and the points between stations do not directly map onto a particular point in geographic space. In this situation the islands become less desirable, since they appear to form a break in a person's journey.

In the above diagram, the contours are centred on the Hammersmith and City branch of Shepherd's Bush. There are two dark contours representing areas that can be reached in less than ten minutes. The first encloses Shepherd's Bush and Goldhawk Road, the other Latimer Road and Ladbroke Grove. This division is unintuitive – since a user will be travelling along the connecting line, they would expect all parts of that line to be represented as reachable in less than ten minutes. In other words, contours should hug the lines.



This behaviour can be demonstrated by centring the contours on the Central Line branch of Shepherd's Bush. In this example we get the desired behaviour: there is a single unbroken contour between Shepherd's Bush and White City.

The problem is an artefact of the Delaunay Triangulation approach used for estimating journey times. We only have precise journey times for nodes, and we must calculate the journey time for all intermediary points as a function of nearby nodes. Where there are four nodes forming a diamond however the question arises for points within this diamond as to which nodes should be used for calculating the journey time. In this situation the Delaunay triangulation scheme constructs triangles such that a triangle edge connects Shepherd's Bush on the Central Line to White City, all points on this line are thus calculated as a weighted average of these two nodes only. This can be seen most clearly when we display the triangulation with the lines removed:



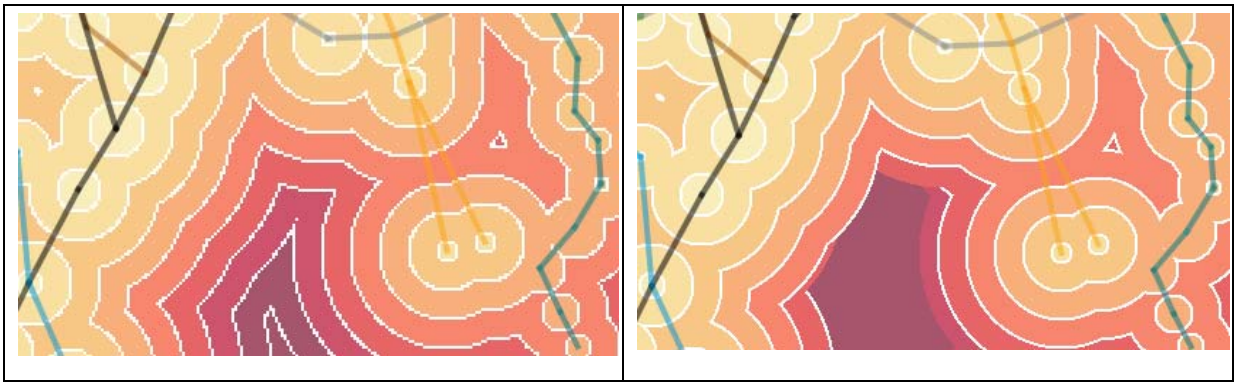
To solve this problem we need to redesign the surface algorithm so that it will bias elevation prediction towards using nodes with shorter journey times. For the example where the contours of focussed on Shepherd's Bush (H & C), points in the centre of the diamond would then use the journey time calculated between Latimer Road and Shepherd's Bush (H & C) since this is smaller. Achieving this may require adjusting the tessellation algorithm.

9.4.2 Building a cost surface using an exit point height map

To implement the exit point based terrain generation we made use of the height map. A height map is simply an array which ascribes a height value to each position in the array. The approach allows us to take each station in turn, and introduce its influence on journey times in the network at large, these effects are combined onto a single surface, avoid issues like the contour island problem where the effect of particular nodes dominates over others.

The first stage of the algorithm initiates the height map at the defined resolution, setting each entry to the maximum journey time. A translation is also defined between the screen space that the nodes are defined on, and the height map space. Each node is then taken in turn and elevations for surrounding pixels in the height map are adjusted. To perform this change, each position in the height map is iterated through, for each position an elevation is computed according to the distance from the current node, the set walking speed and the time to reach the node via the network. If the calculated value is lower than the current value for that entry in the height map that entry is replaced with the new value. To make this faster a maximum time that a person is willing to walk is set, so that for each node only a small subset of the height map around the node's position in height map space needs to be analysed.

One of the most important things about the height map is its size. The resolution of the height map has a dramatic impact on the visualisation, an effect that is particularly noticeable on the geographic layout of the underground. With this layout the large spread of the outlying nodes means that a high resolution is required to get enough detail when viewing the centre of the network in Zone 1. A comparison of this is shown below. The left hand image is a detail of the surface rendered using a height map with a resolution of 1500 by 1000, the height map of the right hand image has double the resolution. The right hand image is more attractive and resultantly people are likely to examine it more carefully.



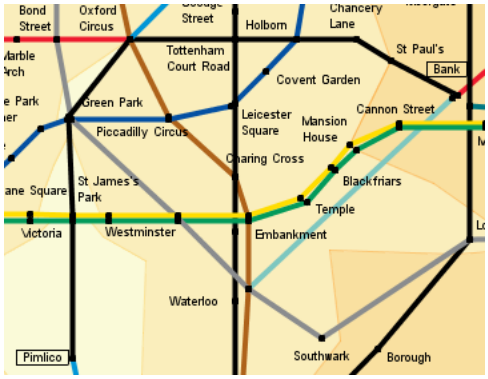
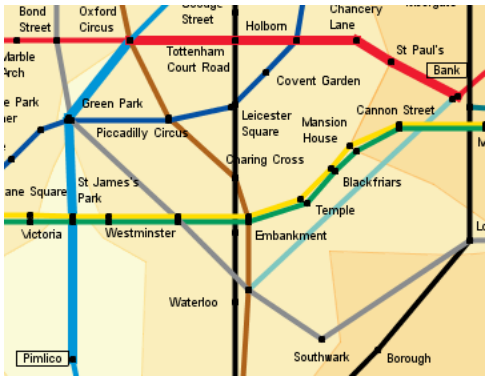
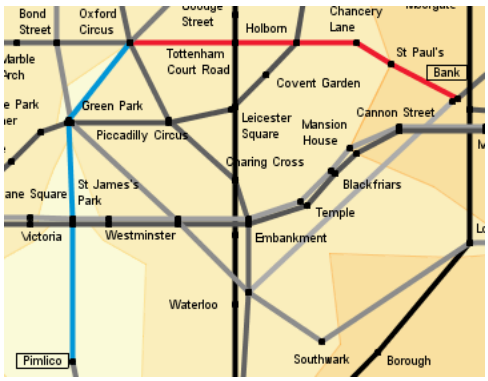
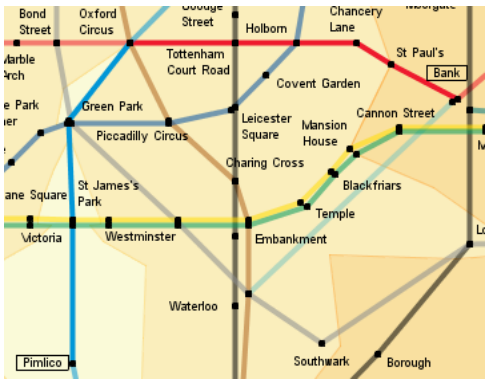
With such large arrays, memory usage begins to be become a problem. For a 3000 by 2000 array of doubles we require 3000 * 2000 * 4 = 24MB. We can improve efficiency fourfold by reducing the resolution of the height and using bytes instead of doubles. It is rare to need a greater height resolution than the 256 that a single byte provides, and our journey time scale rarely rises above 100 minutes, so the change doesn't sacrifice detail. Even with this adjustment the memory usage is very high. The height map algorithm in future incarnations of the software will therefore be adjusted to only paint the area being viewed, rather than the entire network space.

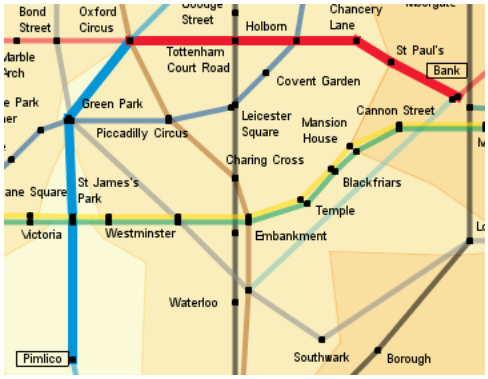
9.5 Alternative Visualisations

9.5.1 Indicating Journeys

The visualisation of time contours has proved to be an effective approach for describing the accessibility of a network. However, the implementation is not necessarily intuitive, and people will take time to learn how to read them. An important tool in doing this is to couple the contour visualisation with shortest journey information.

People interpreting the map and discovering that they can reach a certain destination in just one hour will immediately want to know *how*. The time contours themselves go some way towards presenting this information, but they can be ambiguous and require analysis. To resolve this, the fastest route between two points is displayed when the user hovers the mouse over another node. There is however a large amount of information already displayed on the map, with the different coloured lines, contours, nodes and station labels, so it had to be designed carefully. We solved each of the different problems that describing the journey presented with separate filters, and could then compose them together to create an ideal, but customisable solution.

	The initial implementation uses a JourneyFilter to set the colour of the selected journey to black. The line does not stand out significantly from the background, and the visual conflicts with the black Northern Line
	Here we add a JourneyWidth filter, which enlarges the width of the selected route. The journey stands out as being rendered in a different style to other lines.
	The JourneyGreyFilter makes the unselected lines greyscale, so that the selected journey can be easily discriminated from the rest of the network. In greyscale the other lines can still be distinguished, but for general analysis it is still preferable to have the option off.
	The AlphaFilter makes the unselected lines slightly transparent. This not only makes the journey more vivid, but also helps resolve the issues of the grey and black lines not changing in the greyscale example



The Journey width and alpha transparency filters combined provide the best general solution to the problem of indicating journeys. It can also be combined with the grey filter if desired. This shows off the power of the filter architecture in both using and designing a journey indication system.

9.5.2 Route playing

When the "play the route" tool is selected, right-clicking on a node will cause TimeContours to sequentially generate contours for each node on the route between the start node and the right clicked node. This allows a user to view how accessibility changes as you move through the network.

9.5.3 Where to meet up?

When two people at different points on the network wish to meet up, they have to decide where to meet. The TimeContours application can generate shared contours to describe the ideal locations to meet in terms of time. For example, if one person was at Tufnell Park, and another at Canary Wharf, then the combined contours will appear as shown below.

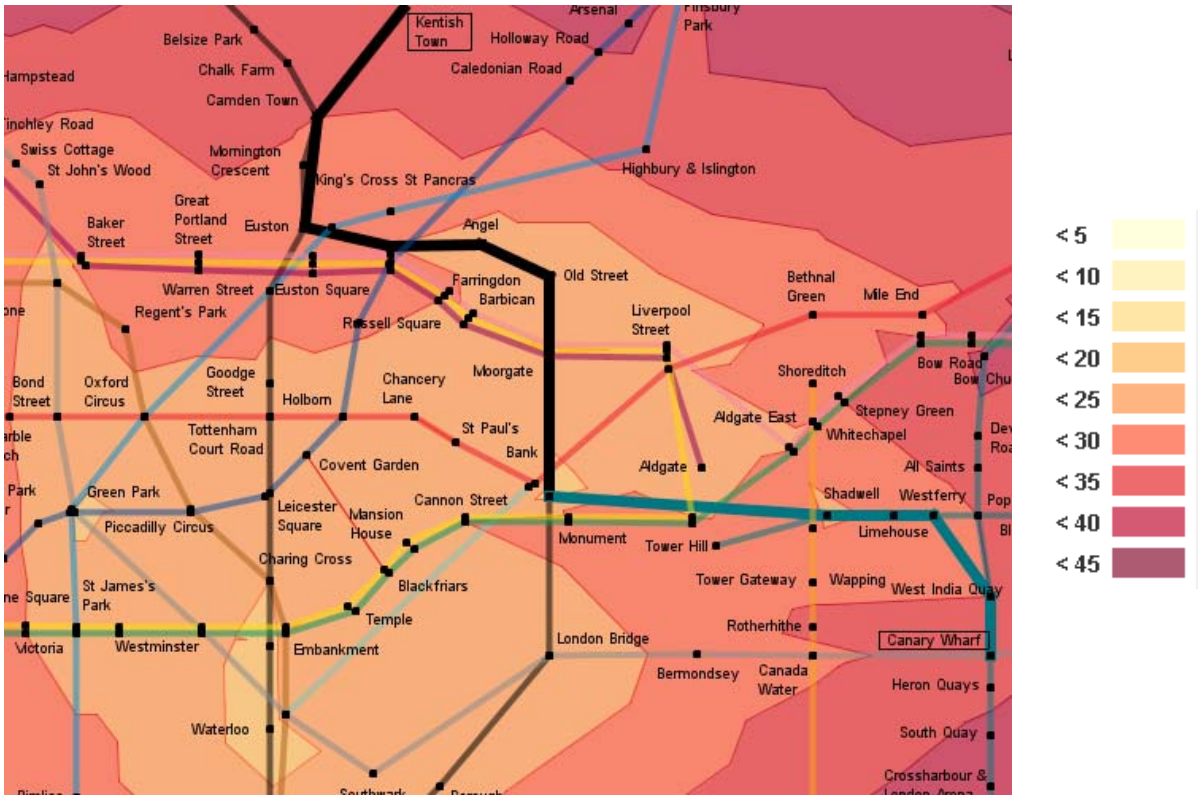


Figure 47 - Combined contours for Kentish Town and Canary Wharf

From this diagram we can see that Bank, Green Park and Waterloo are the three best places to meet, each of which are accessible to both people within twenty minutes. The heavy line meanwhile displays the fastest route between the two. This information gives users a chance to evaluate the best place to meet, given their own preferences between different places, as well as other activities during the day.

Calculating this surface is a simple process: journey times are calculated to all other nodes for each of the two starting points. We then go through each node, taking the larger journey time for each one, and use this data set to construct the contours.

9.6 Selecting Colour

The importance of colour selection in visualising information is often overlooked. Poor use of colour can make an image difficult to interpret, or worse misrepresent the information entirely[40]. Colour has various different functions. Edward Tufte in *Envisioning Information* describes four: to label, measure, represent or imitate reality and to enliven or decorate[41]. In our maps we use colour to label in describing the underlying network, and to measure in building the contours. The main issue with regard to choosing colours for the network was to ensure that they did not conflict with those in the contours. For the underground implementation this was a particularly important problem (see page 67). The primary issue however was the selection of colours for the contours themselves.

Map theory has provided us with various different types of colour scheme: binary, sequential, diverging and qualitative. The different types are shown in the diagram below, courtesy of Cynthia Brewer

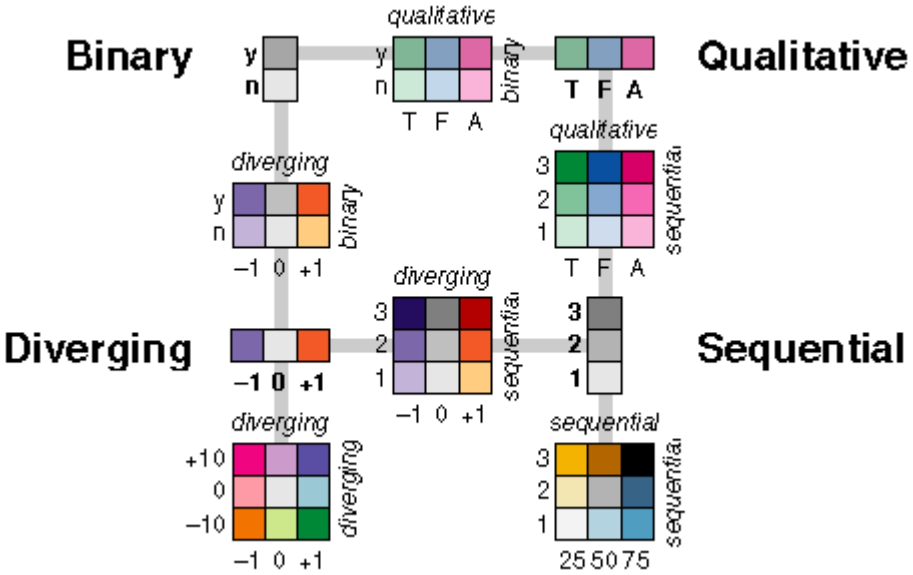


Figure 48 – Comparing colour scheme types

Since we are using colour to measure on a scale from low to high (journey time) we use a sequential colour scheme. We implemented various different colour schemes, including one varying from warm (red) to cold (blue), and various ones

taken from Brewer's *Colour Brewer* tool. The colour schemes are mapped so that light colours indicate close areas and dark colours distant ones – this matches human's natural association that light is less and dark is more. Light colours also have less impact on other information and it was preferable to make the focus of the visualisation as clear as possible. In order to ensure that dark areas do not distract at the edges of the map, we used an alpha transparency effect to soften the effect.

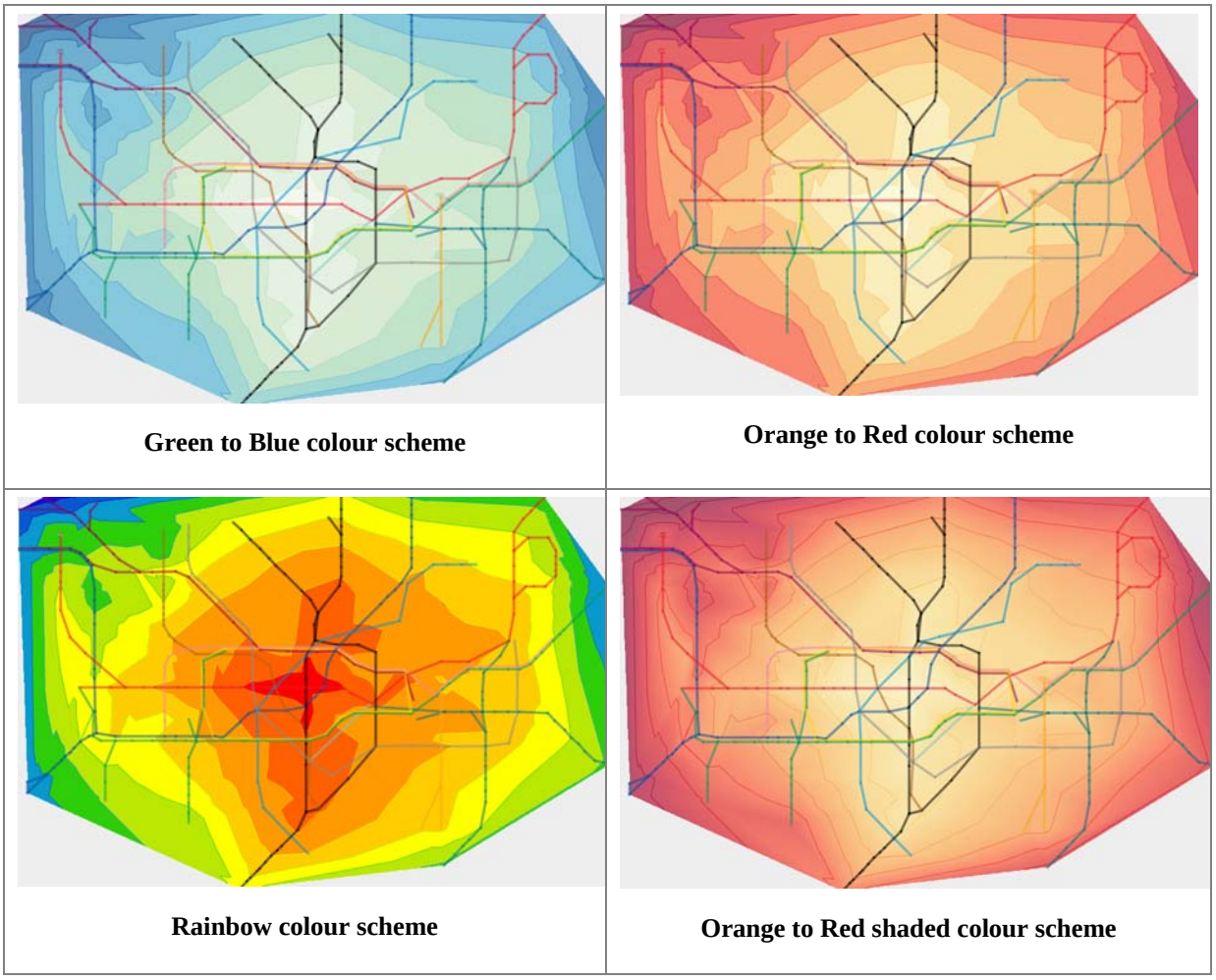


Figure 49 – A comparison of various colour schemes

By default we use a block approach to colouring, shading for example all elevations between 10 minutes and 20 minutes the same colour. The blocks of colour have slightly darker contour lines at their edge to bring out the boundaries. The presence of these is important as "human cognitive processing gives considerable and often decisive weight to contour information"[42]. We also offer a "shade" option, which sets the colour for each elevation as a weighted average of the two colours on either side – the images this forms are however generally more aesthetically pleasing than functional. Although the rainbow colour scheme provides the most distinct changes between regions it is not in general a good choice for a sequential scheme. As Tufte remarks, "the mind's eye does not readily give an order to R O Y G B …". The green to blue and orange to red colour schemes were taken from Cynthia Brewer's excellent "Colour Brewer" application[43].

9.6.1 Resolving colour conflicts with the Underground

The underground prototype created particular difficulties with regards to colour. With most road maps it is possible to convey the different types of roads without resorting to using colour. Roads can be drawn in grey, and the thickness adjusted depending on the road's class – A roads and motorways are thick, B roads and side roads are thin. Colour in the London Underground map however is fundamental to its design. Lines are distinguished by colour, the Piccadilly Line for example is a shade of purple, and the District Line is green. Changing the colours significantly impedes the recognition of the map as the London Underground, especially in our example where the network is similar, but not exactly the same as TFL's design. The selection of colours in the map covers the entire spectrum (including greys), making it difficult to use a palette for the contours that does not to some degree overlap with those in the network.

One solution to this problem is to use the greyscale journey filter, which sacrifices the colour line information, but retains the grey intensity so that lines can be distinguished and recognised. We found that this was better as an optional setting, as people still like to see the lines in their original colours. For our default display we chose a colour scheme which stretches from a light orange to a dark red. This colour scheme does not conflict with too many of the lines, and was found to be better than the blue green colour scheme for example, which hides both the Piccadilly and Victoria lines. The option however remains for the user to change the colour scheme to one that they personally prefer. One of the other useful options is the grey colour scheme, which makes time information more subtle.

10 Interface Interaction

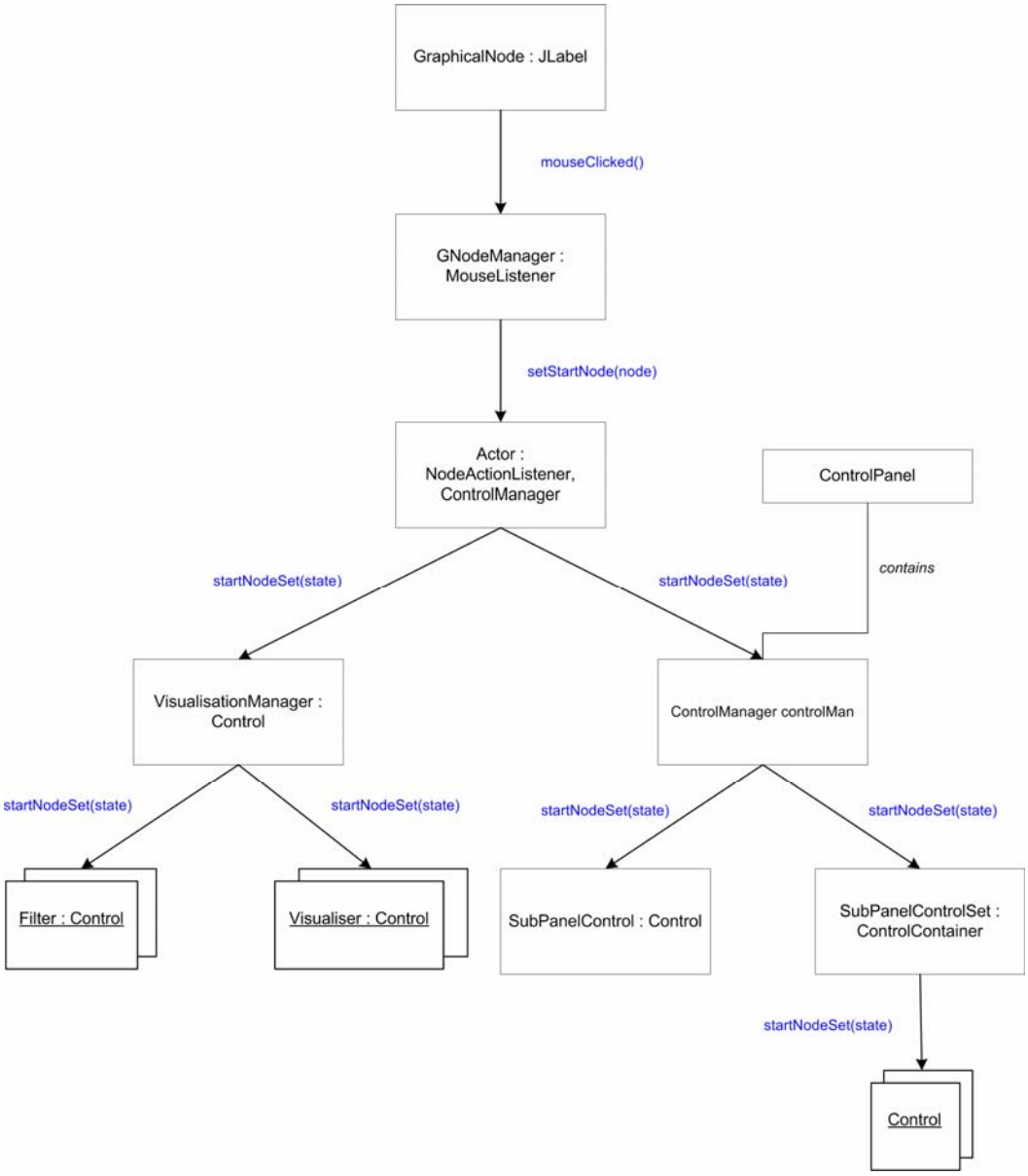
One of the requirements of the TimeContours engine is that it should be simple to add tools to the application. The addition of informative displays, tools such as for animation and visualisations should be separated from the complexity of the underlying engine. One of the difficulties in achieving this is managing the flow of interaction between modules. The development of an information panel should not need to concern itself with how it receives state updates. A visualisation tool should be automatically informed of surface changes. A change in node selection should inform every graphical node in view. The interaction design implemented in this project achieves this, partly through the construction of a simple but powerful interaction interface.

10.1 Interaction design

A user interacting with the transport network can make three different kinds of node selection. They can set a start node, which acts as the focus point for generated contours and journey paths and an end node to set the end point of a journey. The selection of the start node is performed by a left click on a node, and an end node by a mouseover. This allows for the user to quickly scan routes to different locations on the network. A final type of action is performed by a right-click – which provides a mechanism for the user to perform actions on a section of the network; for example to form combined contours, or to set a series of interconnecting stations as “experiencing delays”. These different types of interaction together form the state of a user. The current user therefore at any one time has particular values for the start, end and action node. Changes to this state have to be reported to the various different parts of the application, this includes graphical items such as nodes, control panel displays, the visualisers which add items to the display, and filters which control their appearance.

The control flow implementation is influenced by the standard Model View Controller paradigm used with user interfaces. The controller is any module that changes the state of a user, for example the nodes that people click on and the subPanels in the ControlPanel. The Model is the current state of a user, combined with the surface details and the View is the representation of the map via Visualisers and Filters, as well as the ControlPanel itself.

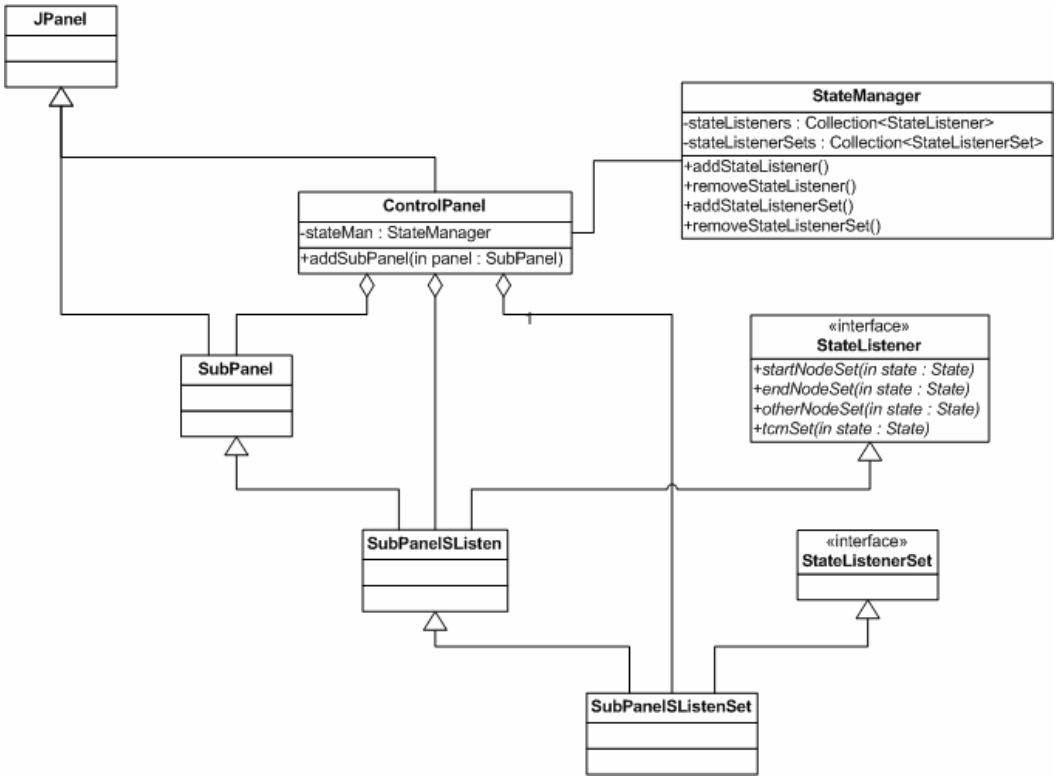
To implement this, the State class holds values for each of the possible node properties described above. This information is stored in an Actor class, which provides a single point of entry for all changes in state. The Actor passes the new state information to classes that are registered to listen to this Actor object. This design was created with the extension in mind of allowing multiple users to be modelled on the system at the same time. The below diagram describes the flow of a state change through the application. The example is of the selection of a start node, but would be much the same for any of the types of state change.



The application makes use of a Model-View-Controller style design pattern. There is a model for each user, which holds their current state. The state includes four pieces of information: the start, end and action stations and the active TimeColorMapper. When one of these values changes the state change is reported to all the concerned objects (i.e. the View). Each one of these actions needs to be detected by a large range of types of objects. Nodes may update their look according to their selection properties, items on the control panel change to inform a user of the selected node, and filters update the display of journeys and other details.

The Visualisation Manager meanwhile also acts as a model, holding the current network journey time data for all parts of the network. When this information is updated it informs all registered listeners of the change, allowing all surface generators and dependent visualisers to update according to a common set of data.

10.2 Flexible Control Panel Design



It was intended that the TimeContours engine should be flexible so that people could use the contour generation functionality in a variety of different applications. Different applications would require different types of interaction with the map, whilst at the same time certain tasks such as showing the current colour legend are generic. The Control Panel architecture was built with this in mind. The Control Panel itself is simply a JPanel, which is registered with the Actor class to receive notifications of changes in state. An application viewer will create an instance of the ControlPanel class, and then add various SubPanels as desired. These SubPanels can either be prebuilt subpanels from the TimeContours library, or ones developed specifically for that application.

Building a SubPanel is very straight forward, this fits the specification that extending and making use of the engine should be simple. For a basic panel which does not need to listen for changes in state we override the SubPanel class. If we need to listen to changes in state though, we override the SubPanelSListen class. The ControlPanel then looks after informing the panel of changes in state as they occur. A basic panel displaying the currently selected start and destination is therefore achieved with very little code:

```
public class SPNodeStartEndDisplayer extends SubPanelSListen {
      JLabel labelStartStation, labelEndStation;

      public SPNodeStartEndDisplayer() {
            //add labels to panel
      }
      public void startNodeSet(State state) {
            labelStartStation.setText(state.getStartNode().getName());
```

```
	}
	public void endNodeSet(State state) {
		labelEndStation.setText(state.getEndNode().getName());
	}
}
```

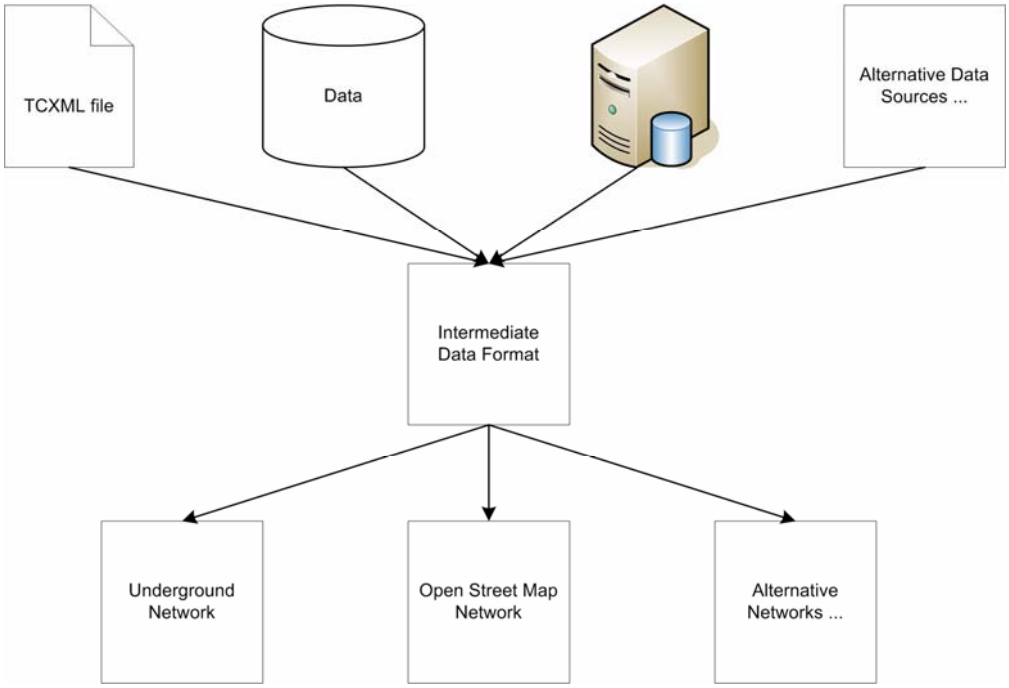
Each SubPanel on creation is provided with access to the particular subset of core classes it requires. For example, SPNetworkDisplay has access to the GraphVisualiser class, as it needs to be able to set the various nodes and edges to visible or hidden. SPJourneyTime meanwhile is given the NetworkAnalyser class, so that it can calculate the journey time between the current start and end nodes. The core classes are outlined in the application overview on page 42.

10.3 Interaction summary

The SubPanel architecture decouples a problem from the interface. As a result we can reuse SubPanels in many different Viewers. For example, the StationEditor and ContourViewer both use the legend display sub panel. The StateListener interface and hierarchy meanwhile means that the passing of state data becomes transparent an application designer.

11 Data Format Design

Intermediate Data Format



The TimeContours Engine specification poses two demanding data management requirements. It must provide flexibility in terms of the data formats it can read, and also the network representations it can construct. We do not however want to have to program a new network constructor every time we develop support for a new data type. Instead we create a flexible intermediate data format. This consists of two hashmaps, one between data name and value, and another to give each data name a type. Each different network type has a set of required data names and types – so a single data set can be used to generate different types of network, as long as it fulfils the requirements of each. For the most basic network type we require:

Node	
Id	Int
Segment	
edgeId	Int
Source	Int
Target	Int

The Underground implementation adds a few extra required fields. The name field is used when searching for a node, whilst the DisplayName defines how it appears on the map – allowing us to use multiple lines for a cleaner display. The NodeType tells us whether the node represents a Foyer or a Platform

Node	
Id	Int
Name	String
DisplayName	String
NodeType	Int
Segment	
edgeId	Int
Source	Int
Target	Int
JourneyTime	Double
EdgeType	Int
LineId	Int

This thus provides a consistent format from which we can construct nodes and edges depending on the network type.

Node Position Data

As discussed in the visualisation section, we decided to decouple a node's location from its definition. This decision increased the flexibility in the layout of the network, allowing both cartographic and geographic layouts on the underground for example. To reflect this decision, we produced a separate data format for node position data, which closely follows the network format, so that we can use the same parser.

11.1.1 Data Sets

In order to build a graph a suitable data set had to be found. The two data sets that were investigated in this project were the London Underground and Open Street Map. The London Underground was a particularly attractive network to use. The network spans a large geographical area, and poses navigation problems to millions of people every day; however it is still small enough for the entire network to be held in memory at once. We were also able to get the journey time data for between stations from an online travel times map[44]. The smaller underground network simplifies the problems of both acquiring and analysing the network. Open Street Map meanwhile shows off the power and flexibility of the system – allowing contours to be generated for any area of the country for which data has been acquired.

11.2 Data Format Design

The standard approach for building a graph is to firstly add all the vertices, and then add the edges that join these vertices. This ordering minimises inter-dependencies, and we can be sure that all the necessary data is available when adding the edges. This process therefore also determines the representation of the

network data. Firstly all the nodes must be defined, and then the conjoining edges, referencing the nodes via a unique ID.

In designing the file based data format for the network, we looked at various different graph format standards. The Graph Modelling Language[45], or GML, is a popular standard for storing graphs born out of the Graph Drawing Symposium 95. The wide level of usage is attractive, since it would increase compatibility of TimeContours with other graphing applications. An XML version of the standard is also being developed called XGMML (eXtensible Graph Markup and Modeling Language)[46]. This was an appealing option, since it is easy to add meta-data to nodes and edges and the XML base makes it simple parse and read. GXL is a similar format, but includes more extraneous data making it more difficult to analyse correctly.

Our final choice for graph format was GraphML, a simple XML based design. The strongest feature of this format is its use of keys to assign types to extended attributes. This meant that if we added a "displayname" field to nodes, we could assign a key at the top of the document mapping the type of displayname to a String. The GraphML parser then interprets this, and when a displayname attribute is found it automatically constructs a String object to put it in. This nicely fits the Intermediate Data design, and means that a single parser can cope with GraphML documents for all network types.

11.2.1 TCXML

TCXML is our extension to the GraphML file format for TimeContours networks. The filetype is essentially a standard GraphML file, except that it insists on particular keys for certain network types. GraphML[47] at its most basic simply declares nodes and edges in the format below:

```
<node id="n1"/>
```

```
<edge source="n1" target="n2"/>
```

To add metadata to a graph component, keys are added at the top of the XML document which defines that metadata field's type. To add the data to an individual graph component, data tags with an id attribute set to the key name are added. An example of the foyer node South Acton in the Underground implementation is shown below:

```
<key attr.type="string" id="name"/>
<key attr.type="string" id="displayname"/>
<key attr.type="int" id="nodetype"/>
```

```
<node id="1014">
      <data id="name">South Acton</data>
      <data id="displayname">South Acton</data>
      <data id="nodetype">1</data>
</node>
```

11.2.2 Database Implementation

XML provides an excellent format for a stand-alone version of the application. The format is human readable, allowing for minor tweaking by hand, but can also be interpreted by various graph applications for further analysis. During development however we used a database implementation, which provided greater flexibility when aspects of the data structure needed to be adjusted, and came with the added advantage of being a lot faster than XML. If a server based online implementation is ever produced it could take advantage of this database implementation. The database schema also provides a good model of the overall data structure:

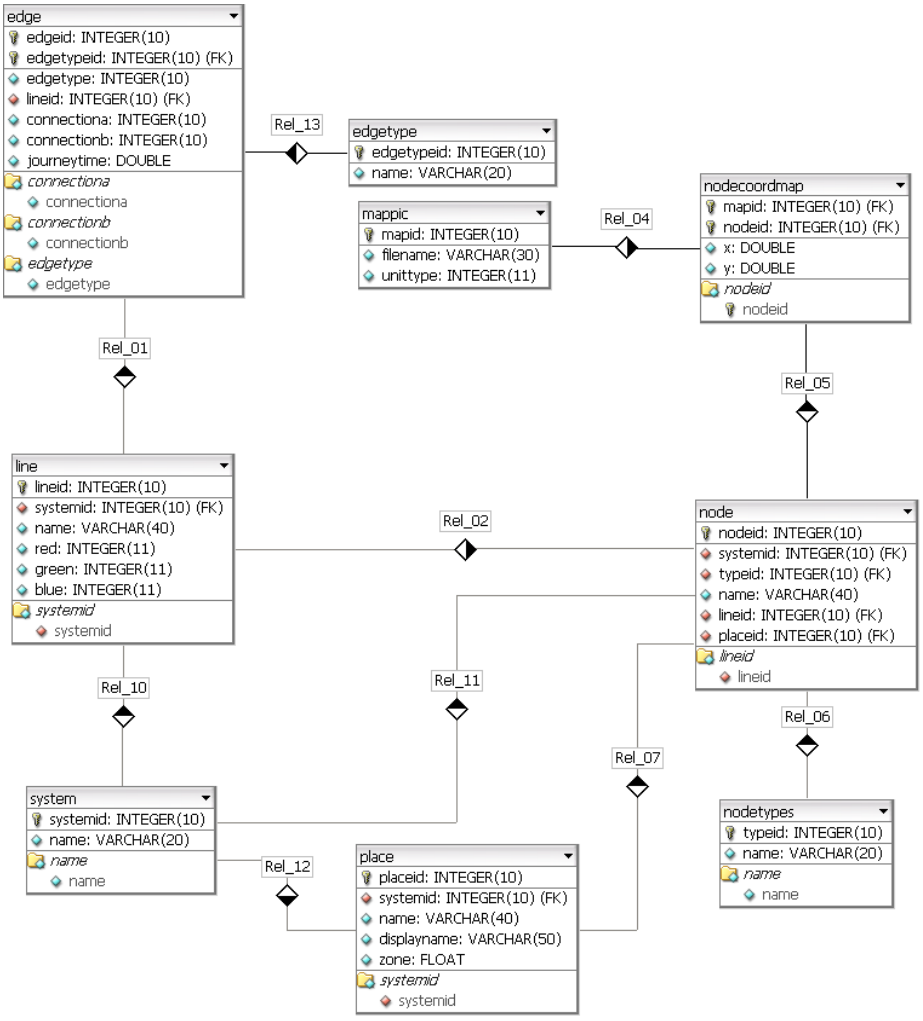
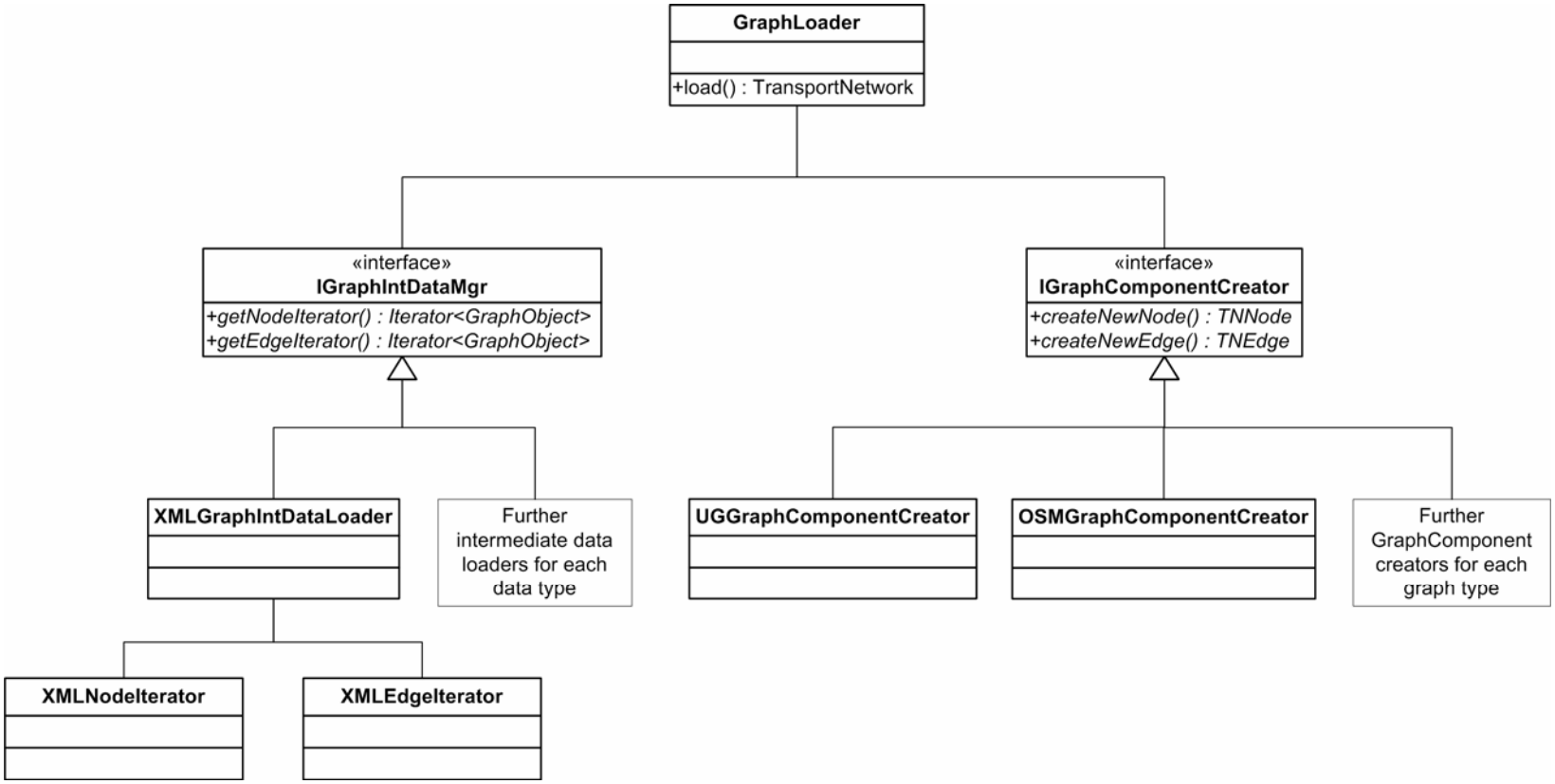


Figure 50 - Relational Diagram of Underground Database

The *edge, edgetype, line, node and nodetypes* tables are self-explanatory. The place class allows multiple nodes to be registered as being part of the same place – this is particularly useful where many platforms and nodes are part of the same station, and share its name. The *mappic* table holds map specific information, such as projection method, whilst the nodecoordmap holds the actual node position data.

11.3 Loading the data



The above class diagram shows how the loading of graph data into TimeContours was achieved. The GraphLoader provides a bridge between compatible data formats and the different network constructions. The Intermediate Data is held in a GraphObject, and the intermediate data managers (on the left of the diagram) provide access to these one by one for interpretation by the GraphComponentCreator (on the right of the diagram).

11.4 Extending to support Open Street Map

Open Street Map is a project to collect and provide geographic data such as street maps free of charge. Data is collected by individuals using GPS logs, which can then be used to edit and create map data. This data can be accessed in two different ways. Firstly you can use their online browser applet to browse the maps, and more usefully you can request map data in an XML based format using Representational State Transfer, or REST[48]. Their REST service allows you to request an XML representation of part of the map using an HTTP style GET request. For example, the request "`map?bbox=bllon,bllat,trlon,trlat`" returns map data for the area enclosed by the given bounding box.

The OSM XML data format is relatively simple, and consists of nodes, edges, areas and ways. Nodes are defined by *id*, *longitude* and *latitude*, whilst the edges have an *id*, a *from* and a *to*. Ways are sets of edges and define roads or paths, it may be possible to represent these using the Line functionality of the Underground network, but for the time being Ways are not implemented. Areas are also ignored. Each of these components can be extended with the addition of an XML tag called *tag*. This can include meta-data about the component, such as its name or class. Edge segment class information could be used to define different speeds for different parts of the network, but currently there is a lack of this information in the database.

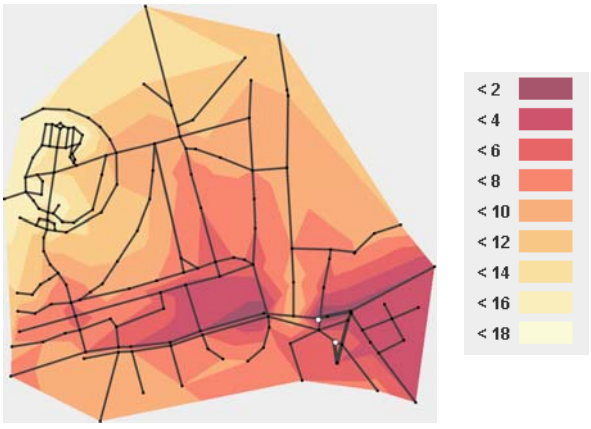


Figure 51 – TimeContours' representation of the Regent's Park using OSM data. The contours are calculated assuming a walking speed of 5km/hr

Figure 52 – OSM data as rendered using the Open Street Map java applet

Figure 53 – Regents Park according to Google Maps

We can get an idea of the accuracy of the Open Street Map data by comparing the map to the same area in Google Maps. The Open Street Map version lacks the resolution of Google Maps, although it has enough information for a prototype application. The OSM implementation's biggest drawback is the lack of meta-data for classes of road. Most of the additional paths in the above maps are footpaths rather than roads – their inclusion could easily distort driving based time contours maps, especially if items such as footbridges are included. As a result the OSM implementation of the application is focussed on modelling a walker rather than driving. To do this we use a combination of two Graph Filters. The first filter uses the great circle formula to convert the endpoints of an edge in longitude/latitude format into a distance in metres. Distances are then converted into time working on the assumption that a person walks at a speed of 5km/hr.

12 Evaluation

When this research project was started, there had been very little recent research into time based travel maps. Since then Oskar Karlin released his representation of the London Underground, Tom Carden his applet based on the idea and now just few weeks ago mySociety released details of their own investigation into time maps in Britain. The concept may not strictly be new, but it is evidently gaining ground as a modern approach for analysing the spaces we live in. In this section we firstly assess attempt to assess the value of the isochrone technique. We then compare our TimeContours implementation to alternative implementations, assessing their strengths and what needs to be worked on. Section 13 deals with extensions that could be made to the technology in the future, and the final section looks at some of the different applications it could be put to.

12.1 Evaluating the isochrone technique

The concept of the isochrone is fundamental to this research project, and so it is important to assess how effective the information it provides is. By analysing how people read the standard tube map, we can assess the advantages that isochrones offer us in assisting with navigation and understanding. The most basic way to assess journey time on a map is to measure the distance between stations on the map. We therefore investigated comparing times based on distance to those estimated using our network model.

Figure 54 – 10 minute contours and concentric circles from Oxford Circus

Figure 54 shows ten minute time contours from Oxford Circus with concentric circles overlaid. The overlay attempts to emulate a basic distance mental model that a viewer might use. Although the circles match the underlying contours loosely, there are clearly major deficiencies. To model these we constructed two surfaces in TimeContours: one based on times calculated by the underground model, and one modelled with a simple map distance to time relationship. By

subtracting one from the other we are able to visualise the error that the distance model introduces. In Figure 55 below white shading indicates places where the difference between the two estimation techniques is less than 2.5 minutes. Areas are shaded progressive shades of brown where the distance model underestimates the journey time and shades of purple where it overestimates the journey time.

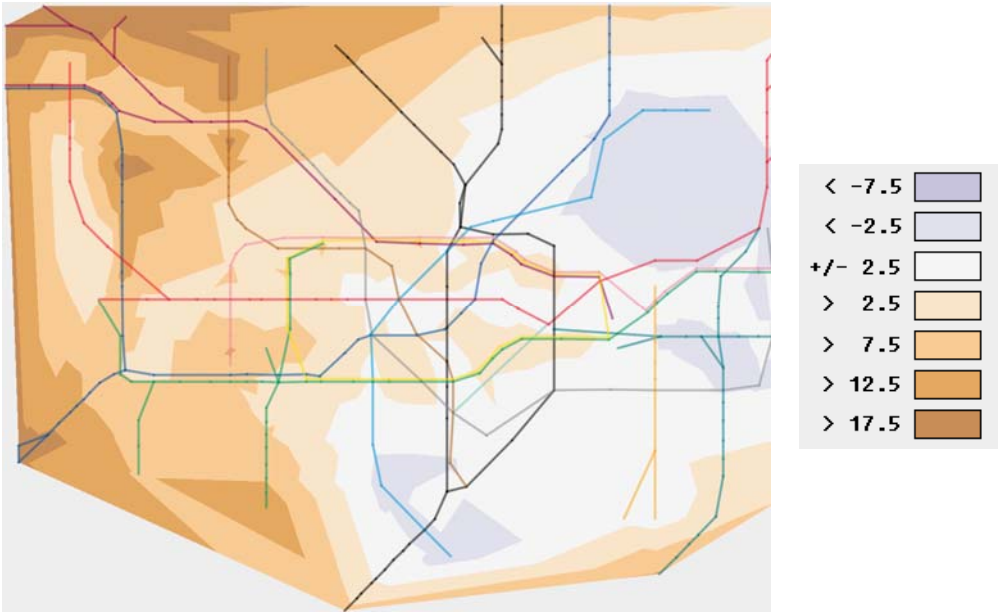


Figure 55 – Modelling map distance to isochrone error

The Victoria Line is one of the fastest on the Underground, and leaves Oxford Circus in a relatively straight line. For this reason the distance model overestimates journey times to ends of this line. For most other areas of the tube the distance mode underestimates the journey time. A journey to the western end of the Central Line requires the traversal of two sides of a triangle, whilst the distance model traverses the shorter hypotenuse; stations on the Metropolitan line meanwhile require costly interchanges. In these areas to the north east of the map the distance model underestimates journey times up as much as 20 minutes. From this it can be seen that the distance model is an inadequate system for estimating time on the underground, and that there is value in an isochrone approach.

12.1.1 Flexibility

The flexibility to model alternative network types was one of the most important parts of the specification outlined in section 5. Our system successfully meets this requirement through the way it supports multiple data formats and unrestrictive extensibility of network models. The Open Street Map model is an example of the simplest network type, and the Underground implementation builds on this adding functions to respond to changes in line status, user preferences and interchange times. There is no reason why the isochrone functionality of TimeContours could not be applied to new other networks.

12.1.2 Network Size

TimeContours is capable of modelling and rendering large networks. The Open Street Map implementation gives a good example of this, and the network displayed on page 37, has 6193 nodes and 7116 edges. This compares to the London Underground, which uses just 767 nodes and 1061 edges. The larger network does however come at a price, and the generation of isochrones can take up to fifteen seconds. Most of this time is taken to render the isochrones, rather than calculating the surface, and the incorporation of a more powerful rendering technique would largely resolve this.

12.2 Comparing TimeContours to alternative time map implementations

12.2.1 Comparing to Space Syntax

Space Syntax, as discussed in the introduction, provides an alternative approach to analysing networks. Whereas the isochrone system is inherently focussed on particular points at any one time, Space Syntax provides a generic way of describing the routes of most importance. The axial map below is a global integration map of Greater London in which the line are shaded from red to blue depending on the “complexity distance”, that is, the number of connections required to reach all other lines. The shading pattern clearly resembles that of the time contours expanding out from a central point.

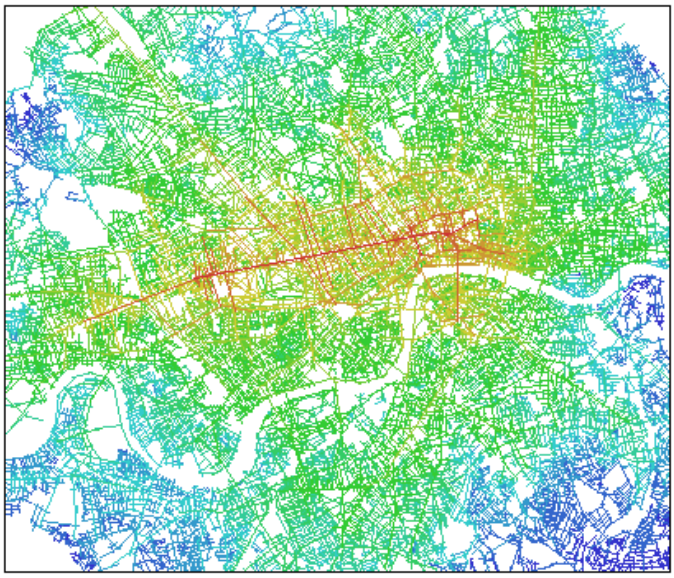


Figure 56 – Global Integration Axial Map of Central London

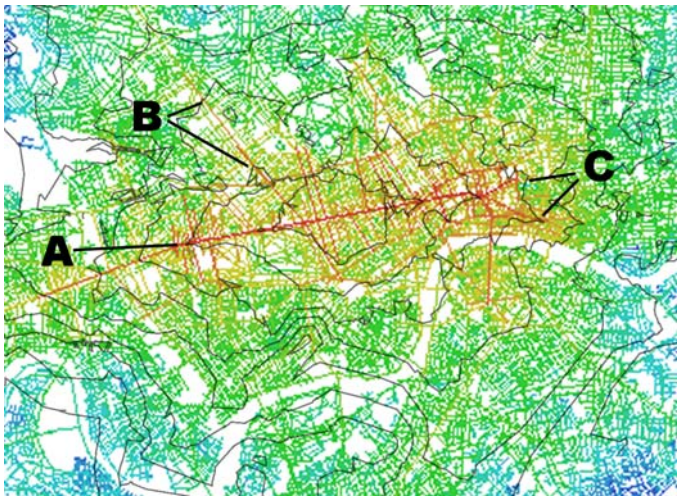
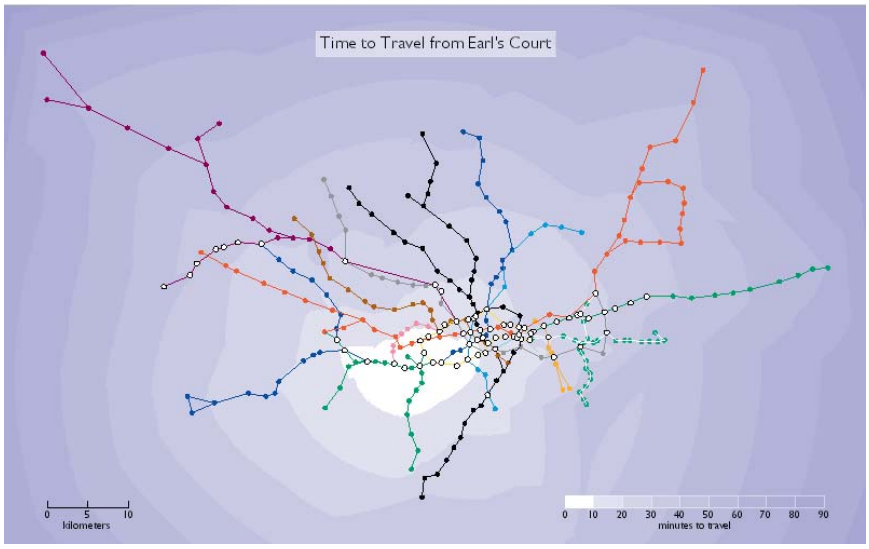
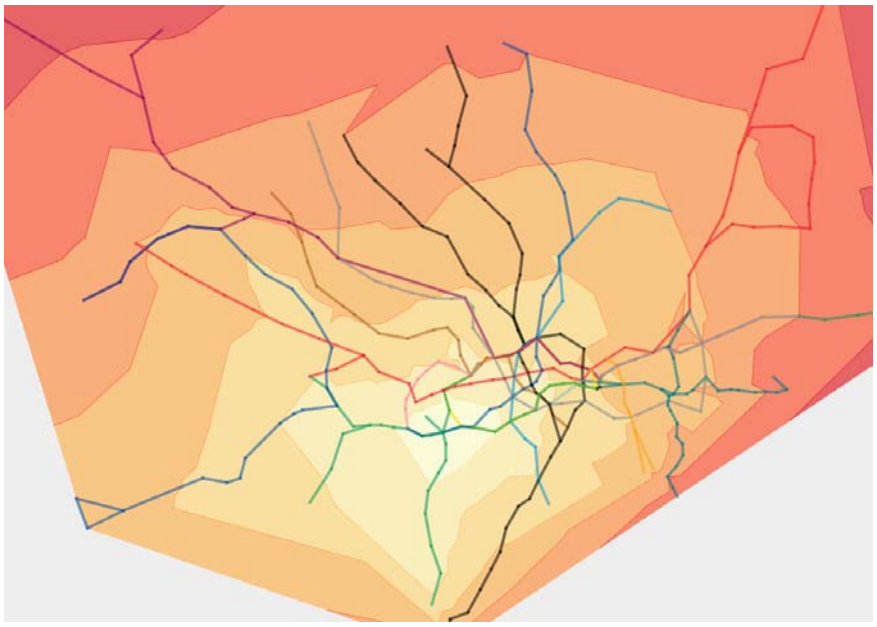


Figure 57 – Ten minute time contours overlaid on the axial map

In the above diagram we have overlaid contours generated in TimeContours using Open Street Map data on the Axial Map presentation. In this example it appears that the contours are influenced by the lines with the highest complexity distance, an effect which can be seen at points A, B and C. There is likely to be a relation between the two approaches, but without better data sets and more examples it is difficult to make any conclusive judgements.

12.2.2 Comparing to Tom Carden's Tube Travel Contours:





Tom Carden's Tube Times Applet is probably the application closest in design to TimeContours. The isochrones between the two implementations are similar, although in the TimeContours version the larger costs calculated in for transfers means the contours are slightly more closely packed. Carden's applet has the advantage of being faster, and that the transition between contours is animated. The extension of the contours to the edge of the viewing area meanwhile is preferable to the sudden break that occurs in our implementation.

In our opinion the TimeContours application is both more attractive and functional. The map can be zoomed, and browsed, the colour scheme can be changed and shaded, and the contour interval can be adjusted. There are also many more tools for investigating and modelling changes to the network. In summary, Carden's applet provides an excellent proof of concept for isochrones on the underground, whilst TimeContours provides an engine for their analysis.

12.2.3 Comparing to Karlin's "Time Travel"

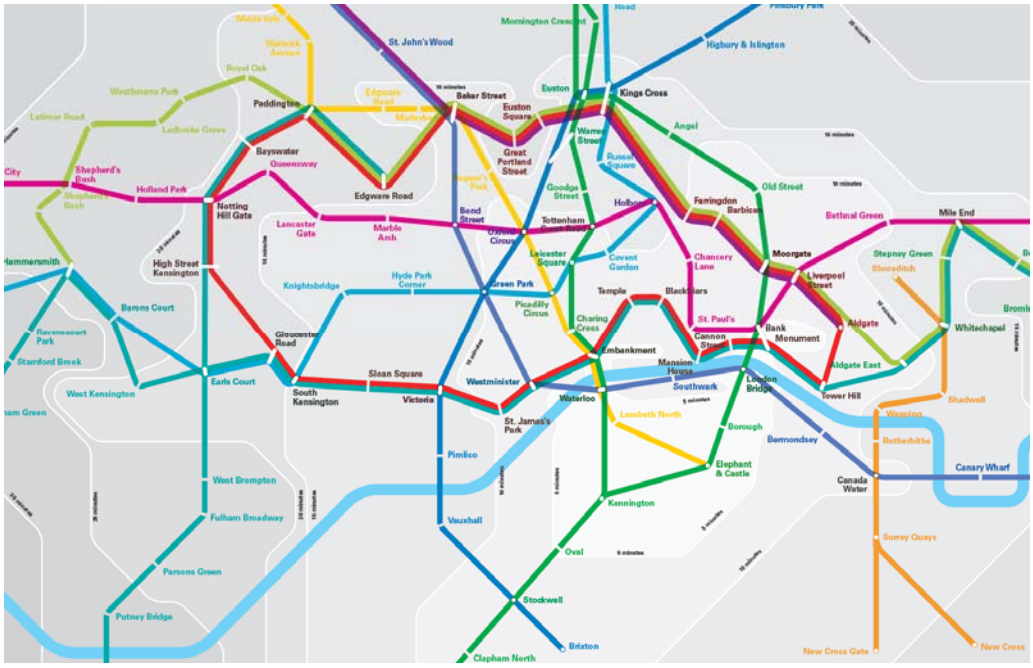


Figure 58 – Detail of Karlin's TimeTravel five minute contours from Elephant & Castle

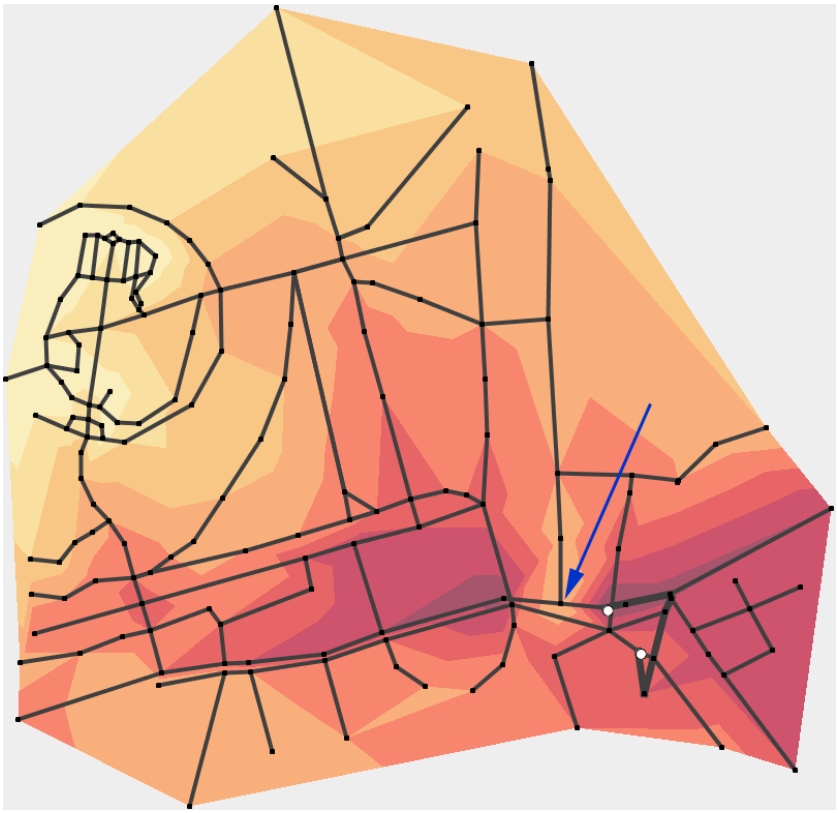
Figure 59 – Five minute contours from Elephant & Castle in TimeContours

Karlin's drawing has a few advantages over our implementation. The river gives Londoners a figure of reference and the contours have times by the side rather than relying on a legend. The contours themselves are slightly less jagged and he also provides us with the extra information of line speed through colour. We have however taken on board the deficiencies of his design discussed in the background in section 4.1.1, particularly with regard to using a familiar layout

and line colours. The colour contours are also easier to discriminate than Karlin's greys and the ability to dynamically generate the scene transforms the map from an item of interest to a functional tool.

12.2.4 Assessing the Open Street Map implementation

One of the main problems with Open Street Map is the quality of data it provides. An interesting application of the application however could be in improving this data. The below map is of the Regent's Park area with two minute time contours. The dark area representing regions accessible within two minutes is split by a light area in the centre. The reason for this is that the node marked with an arrow is not correctly linked to its adjacent roads. The contours can therefore be used to visually highlight errors in the maps construction.



13 Conclusions and Future Work

Surface improvements

Even with a superior data set, the algorithms for generating surfaces should still be improved. The terrain based elevation scheme used with the cartographic underground map for example includes unintuitive artefacts such as disconnected contours as described on page 57. Another point of confusion noted by some users is the meaning of the shaded areas between stations. These people felt that the contours should fit the tube lines more tightly. Instead of using a linear function for calculating elevations at points on the surface, a Guassian could be employed so that the immediate region around a node has a similar elevation to that node, but that this then rises quickly as you move further away. The surface could also be made smoother by converting it to a Bezier surface.

Tiling

One of the deficiencies in the engine as it is currently structured is that it does not support tiling. Tiling breaks down a large surface into a set of smaller tiles which can be dealt with individually. Tiles can then be added as a user explores new areas, and tiles no longer being viewed can be removed. Processing then becomes more efficient, since areas far away from that being viewed by the user do not have to be held in memory or analysed for journey time information. Once this is implemented the Open Street Map support will be massively improved, and a user will be able to pan around the world, and recalculate time contours for any point.

Large Maps

The generation of contours for very large areas could pose computational problems. Where the generation is being done for analysis of a particular location simply waiting for the analysis to take place may be an acceptable solution. Optimisations and heuristics could however be used to attain estimations over a shorter time period. One isochrone system used for ocean navigation for example[49] finds sets of points on the edges of isochrones and builds further isochrones out from these in order to reduce the search space. Another technique may be to change the resolution of the underlying network depending on the scale at which the analysis is taking place. On large scale road maps for example the smaller roads can be stripped out.

Rendering

The currently implemented rendering engine is slow and the drawing of the map can take a second to complete on simple networks such as the Underground, and much longer with large networks. For the terrain based surface, a large proportion of the wait is because the rendering has not been optimised and does not use hardware acceleration. This could be radically improved by employing a specialised visualisation library. A popular choice here is *Processing*[50] a java based 2D visualisation tool, as used in Tom Carden's applet (described on page 28). A disadvantage of this tool is the large amount of processing power that it

consumes, Carden's applets for example will use 100% of the CPU even when idle. A better alternative would be to use a 3D library, which already has support for terrain visualisation. This would not only speed up the drawing of the standard contours and take advantage of hardware acceleration, but also open up the possibility of drawing the time based terrain in three dimensions.

Extra visualisation techniques

Various visualisation techniques are employed to ensure that information is as clear an uncluttered as possible. For example colour is changed according to journeys or line status. We have not however implemented filters for all the different visualisation effects that we would like to see. One of the most important additions would be for time information to be written on the contours, as is done with most geographic contours and Karlin's implementation. Another idea is to make the brightness of the network itself fade out according to journey time. Nearby areas would be strong, and distant ones washed out.

The journey filters do highlight routes clearly, but around line changes the visualisation technique can sometimes be confusing. It would be nice to add an itinerary tool, to allow the user to see each stage of the journey with ease[51].

Another idea is to highlight the most important travel edges. In any network, from a particular starting point there will be a set of edges that will not be traversed in the fastest route to any other destination. These edges are generally the ones that do not form a part of the minimal spanning tree. By highlighting this information it may be possible to make fastest route information viewable for all parts of the network at once, in a similar way to how the isochrone describes journey time information for all parts of a network.

Improving the interface

The current interface has been created as a development tool, and would benefit from more careful design and added functionality. Whilst many of the required additions are cosmetic (better panning, smoother polygons, improved control panel design etc.) others are more fundamental. The most interesting development would be to create a web-based implementation making such a time based analysis of the Underground accessible to the online community. Another of the more interesting additions would be the ability to view multiple map displays at any one time. For example, with the underground prototype one map panel could be used to display the geographic layout, and another the cartographic layout. A user would then be able to investigate in parallel how a change in the network affects both the behaviour of the network, and the accessibility of London on a larger scale. The application has always been designed with this in mind, hence the separation of node location, map visualisation and actor control initialisation, however further decoupling between the VisualisationManager and Actor classes would be required.

Service implementation

There is potential to extend the functionality of TimeContours from an engine to a service. A TimeContours service would offer various built in data sets, and the ability to interpret user provided ones. Surface maps and contour plots could then

be generated according the given start node and journey cost criteria. This model fits the growing trend for online services which solve one particular problem, and are then combined with other services to build new applications. Applications designers could then experiment with the potential of isochrones in traditional environments such as Google Maps, and perhaps new situations which we have not considered.

Poor data

The availability of useful and accurate data for the visualisations was a major limiting factor for the project. The Open Street Map prototype lacks sufficient road class information, and so provides an incomplete model for generating isochrones. The Underground model is better, but further information on transfer times, waiting times and even current train locations could offer greater accuracy and new insights. The visualisations would also be much more functional if they could take into account a variety of public transport methods, such as bus and rail. The research carried out by mySociety achieves much of this, but they also note the need for better underlying models and reliability information.

One of the most basic parts that could be improved from this perspective is the modelling of walking speed. With better elevation information we could use Naismith's rule to adjust speed according to the gradient of the land around you[52]. This would not only improve the model within cities, but also open up new opportunities for isochrones in the countryside.

The TimeContours engine is built to be extensible, and future work would definitely involve investigating some different data sets. In an increasingly globalised world, international travel is becoming steadily more common. More and more people take holidays, and businesses work across borders. The concept of a travel time based map would therefore be equally applicable on an international as on a national or city-wide scale. The concept has already been investigated with respect to air travel by an online flash applet called the Personal World Map[53]. This map allows you to produce an anamorphosis representation of the Earth around various different starting airports, such that other airports are moved so that their distance from the centre point is determined by cost, time or kilometres.

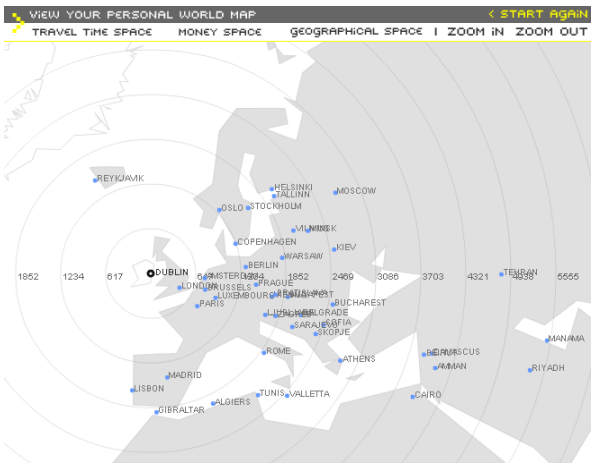


Figure 60 - Geographical Space representation of air travel from Dublin

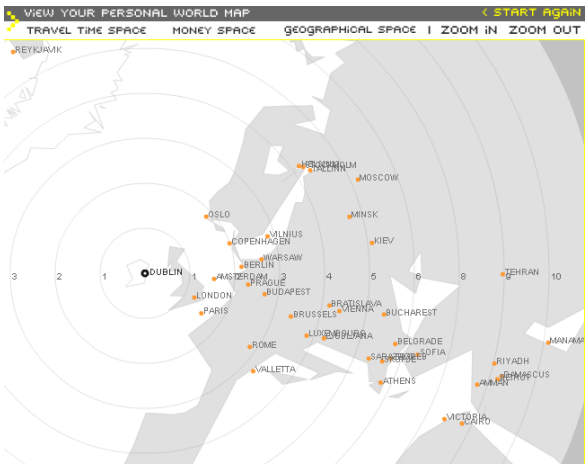


Figure 61 - Anamorphosis Travel Time Space representation of air travel from Dublin in hours

Unfortunately this implementation is biased towards the airports, and does not include this in conjunction with intranational transportation such as trains. Also, on what is already a stylised map, it is difficult to make sense of the change that occurs when the map is drawn in time-space. Time contours offer an alternative approach for visualising this information.

Background Maps

Having maps underneath the visualisation would greatly improve their meaning to users. The mySociety implementation for example uses faded versions of the Ordnance Survey maps in the background. Unfortunately obtaining maps for this purpose is costly, but given support from other organisations this would be a valuable extension. A final change that we would like to see implemented, with Transport for London's permission, is to implement time contours on their own version of the underground map. This would enhance people's recognition of network and information being displayed. An example of how this might look for South Kensington is displayed below

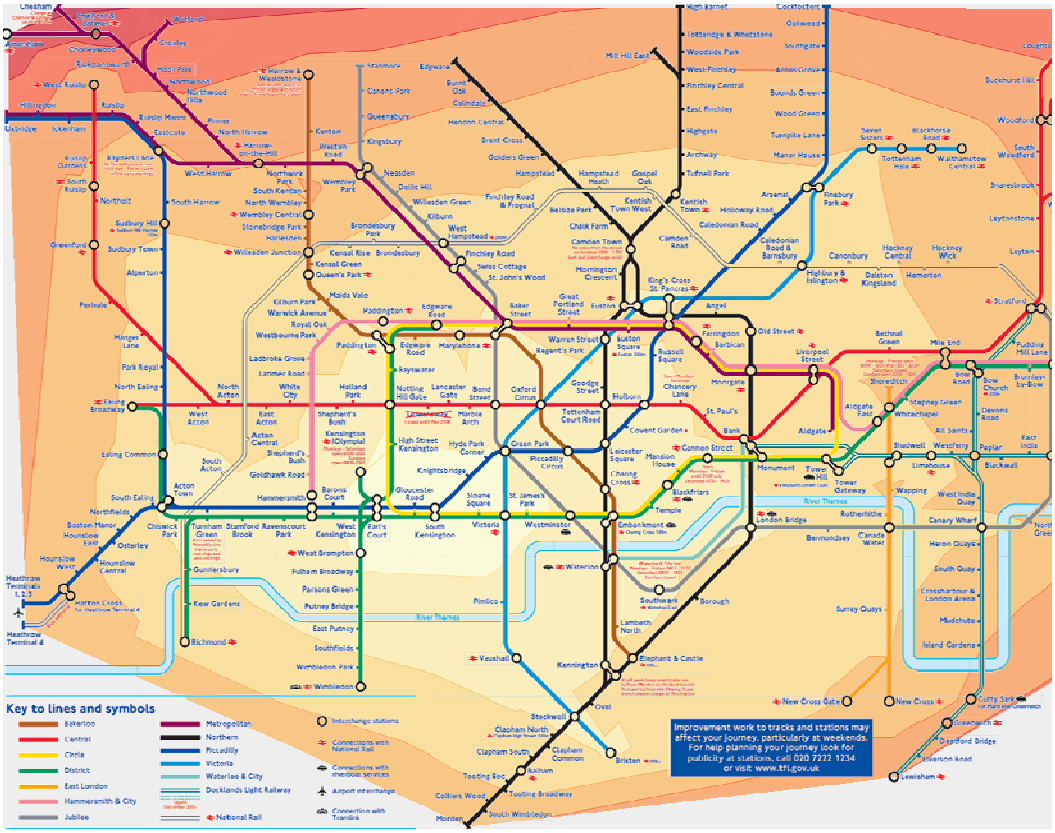
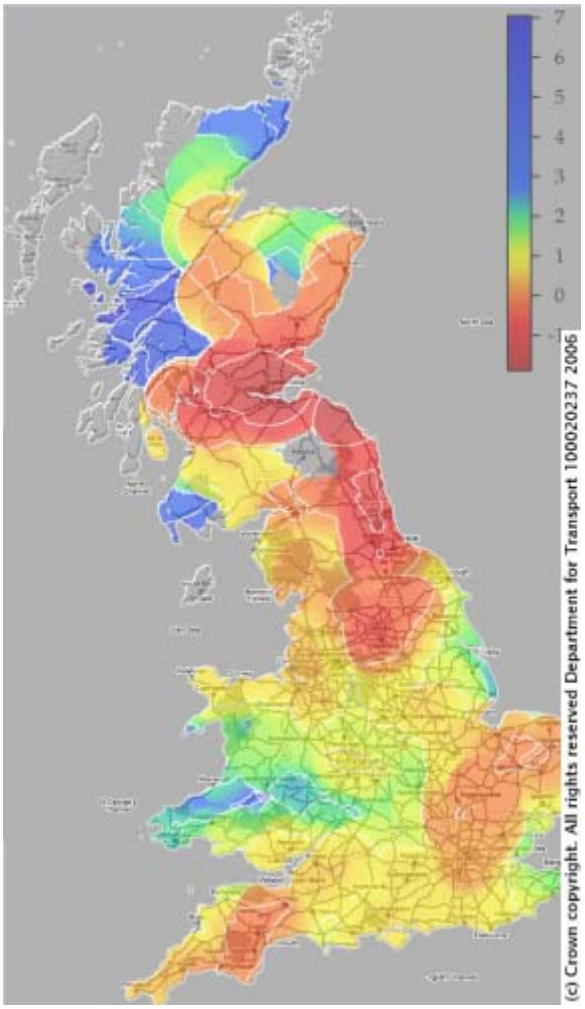


Figure 62 – Ten minute time contours from South Kensington on TFL's underground map

Value based Mapping

As discussed in the introduction, time is not the only useful criterion for which a cost surface over a region can be produced. Time provides a useful example since it is applicable to most people and can have a wide range of applications. We have already started investigating the potential of this with the “Where to meet up?” tool – which forms shared time contours describing preferable nodes in the network to meet. The mySociety investigation also made some interesting adaptations to the approach. One of their maps compares train and car travel, forming a time based cost surface for each, and subtracting one from the other to create a map indicating which mode of transport is fastest for different places. In the diagram below blue areas indicate locations that are fastest to travel to by car, and red areas are faster by train. We would like to see more complex surfaces being constructed, taking account of factors such as activity and amenities at particular locations to convey to a user the value of travelling or investigating areas of a map.



Real-time data incorporation

One of the initial visions for Underground based time contours was a network description tool to supplement their network status reports. Our system has the ability to crudely model the changes in the network that occur when a train line is experiencing delays. One useful extension would be to link this directly to the real-time travel information that is offered by Transport for London on their website[54]. As TFL updates its service information page so the contours would reform to reflect the new network behaviour. If this were coupled with an online implementation then users could quickly check the visualisation on the web before making a journey to see how the network currently looks.

13.1 Alternative Applications of TimeContours

We round off this section by looking at some of the other potential applications of this technology

Setting 1 – Military

Logistics is of vital importance to military strategists, who need to know the capabilities of their forces and the key locations in their environment. By including models of the speed and terrain capabilities of troops or tanks, time contours would allow planners to quickly visualise the range of movement they

can achieve. Combination with alternative cost surface representation, such as including the value of reaching a stronghold or crucial communication points could throw up new options and force users to assess their situation in a different way. The potential is particularly useful where real-time data can be incorporated. One of the greatest advantages of time contours is that they can be read quickly. This property is the main reason for their suitability on the underground for modelling network state, where the majority of travellers move through quickly, and don't have the time to perform analysis for themselves. In a war based situation where for example a bridge being destroyed can shift the isochrones dramatically, and where major decisions need to be taken at speed, the real time isochrone could become and important part of a strategists arsenal.

Setting 2 – Emergency Situations

Disasters pose situations where time and organisation are of critical importance. For example after the Tsunami in South East Asia it was necessary to distribute water and other supplies as quickly as possible to large numbers of people across a massive area. Many workers in these situations are unfamiliar with the accessibility of the network, and even a local will find it difficult to evaluate after such major disruption to transportation routes. In this situation isochrones could be combined with population density, supply need, risk etc. to build a cost surface indicating where supplies should be sent first, or to indicate poorly supported regions, and where supply bases should be set up. Such a tool could be invaluable in making sense of a large number of variables and in coordinating agencies' responses.

Setting 3 – Commercial settings

Isochrones have already been widely exploited in the commercial market, and are used for a variety of tasks including calculating delivery times and assessing where to build services. Tools such as TimeContours still have a role however in filtering this information down to the wider community; very few people have seen isochrone maps, yet they are invaluable in major decisions in people's lives such as where to live in relation to their workplace. Smaller companies that cannot afford the advanced packages such as ESRI's ProTERRITORY could also take advantage of the technology.

Setting 4 – Transport Management

One of the most fundamental uses of iscohrones is in developing the networks that they describe. Time contours can highlight inaccessible areas which need more coverage, or be used in simulating both additions or disruptions to a network. Whilst network analysis is not a new field, the opening up of time maps to the wider community will offer the opportunity for people to analyse networks from new perspectives. The release of mySociety's map for example has stimulated such discussions as using time maps in conjunction with timetables to ascertain how discontinuities in timetables (where connections don't meet up) can make medium distance trips more difficult than long distance ones. Alternatively a wider understanding of isochrones by the general community could stimulate their use in campaigns to convince people of the advantages of public transport.

Setting 5 – different environments completely

How people can travel can also highlight different kinds of information entirely. For example, the potential spread of a disease like SARS could be demonstrated using air flight contours. Not all applications of the technology have to be centred on the movement of people. The dispersion of a drug or poison through a person's body for example could be modelled and visualised using the isochrone approach. Isochrones have been used for similar purposes, such as for heart-excitation models[55], but nevertheless are an underutilised tool. It is hoped that applications such as TimeContours and the other recent time maps creations will increase their recognition and exploitation in many disciplines.

13.2 Conclusion

The field of time based maps, and its application in particular to public transport networks, has developed a long way during the period of this research project. High profile investigations by groups such as mySociety have generated a lot of interest in isochronic mapping techniques, but in such a fast moving field it is difficult for people interested in developing these ideas to know where to start. This report therefore acts as a timely evaluation of the recent developments, as well as making its own valuable contribution.

In this project we have presented the concept of the isochrone as a way of redefining how to view complex networks such as the London Underground. We have shown by comparison to a basic distance model that isochrones offer valuable information that is difficult to elucidate from a standard map. Our comparison to Transport for London's real time travel information shows the potential of the isochrone in describing change. Although our system for modelling tube line status is crude, it does demonstrate the scope for more creative descriptions of network status than TFL's text based solution.

The TimeContours system offers the first tool for interactively generating, browsing and customising isochrone maps. One of the most original contributions is the "Where to meet?" function, which allows isochrones to be shared to find ideal meeting points. This is a unique and functional tool which we hope will be taken and extended by the larger community.

Most other mapping and journey planning tools view the end point as a fixed pre-chosen detail. TimeContours provides a fluid approach to looking at destinations, allowing people to "browse a destination", making discovery of both journey time and route information fast and simple.

Although TimeContours investigates fewer time based visualisation techniques than we had originally intended, the system has nevertheless met the majority of our initial aims and objectives. The data storage system is highly flexible, and supports accepted graph modelling standards. The network model is fast and easily extensible and the Open Street Map support demonstrates the engine's ability to model multiple systems. The engine is also one of the only freely available tools for investigating and easily creating ones own surface representations. As a result the addition and experimentation with functions such as *Where To Meet?* becomes straight forward, requiring no understanding of surface generation or graph analysis.

14 Bibliography

1 Configurational Exploration of Public Transport Movement Networks: A Case Study, The London Underground, Alain Chiaradia, Edouard Moreau, Noah Raford

2 On the Construction of Isochronic Passage-Charts by Francis Galton, Proceedings of the Royal Geographic Society, Volume 3, pp 657-658, 1881,

3 Tube Prune LU Statistics Page, http://www.trainweb.org/tubeprune/Statistics.htm [accessed 13-Jan-06]

4 Comparison advantages and disadvantages of vector and raster methods as revised from Burrough (1986) and Aronoff (1989), http://www.npwrc.usgs.gov/resource/1999/research/struct.htm

5 How to Lie with Maps, Mark Monmonier, Chicago, 1996, p1

6 Diffusion-based method for producing density equalizing maps, Michael T. Gastner, M. E. J. Newman, 2004

7 WorldMapper, http://www.sasi.group.shef.ac.uk/worldmapper/index.html [Accessed 09 June 2006]

8 Transport for London, Tube Maps, http://tfl.gov.uk/tfl/tube_map.shtml [Accessed 15 Jan 2006]

9 Optimal Placement Of Tsunami Warning Hydrophysical Stations, http://www.science.sakhalin.ru/Pub/Popl/India/Index.html [Accessed 04 Jan 2006]

10 Isochrone, http://www.its.bldrdoc.gov/fs-1037/dir-020/_2883.htm [Accessed 04 Jan 2006]

11 Isochrone analysis of the incidence of local monopolies, http://www.competition-commission.org.uk/rep_pub/reports/2000/fulltext/446a6.3.pdf [Accessed 13 Jan 06]

12 Artographie En Anamorphose by Jean-Charles Denain and Patrice Langlois,

13 Michael N. DeMers, Fundamentals of Geographic Information Systems, Third Edition, John Wiley & Sons Inc, 2005

14 The Essential Guide to GIS, Ordnance Survey

15 RouteFinder 1.04 for ArcGIS 8 & 9, http://www.routeware.dk/routefindera/routefinder.php

16 ProTERRITORY, http://www.esriuk.com/products/product.asp?prodid=57&groupid=21 [Accessed 12 June]

17 The Common Language of Space, Bill Hillier, Space Syntax Lab, UCL, http://www.spacesyntax.org/publications/commonlang.html

18 Configurational Exploration of Public Transport Movement Networks: A Case Study, The London Underground, Alain Chiaradia, Edouard Moreau, Noah Raford,

19 Edward Tufte, Envisioning Information, 1990, p61

20 Transport for London Journey Planner, http://journeyplanner.tfl.gov.uk

21 National Rail Enquiries, http://www.nationalrail.co.uk

22 Travel-time Maps and their Uses, Chris Lightfoot and Tom Steinberg, http://www.mysociety.org/2006/travel-time-maps/ [Accessed 10 June 06]

23 The feeling of colour, New Scientist, issue 2484, p40, 29 Jan 2005

24 Eric W. Weisstein. "Graph." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/Graph.html [accessed 06 Jan 06]

25 E. W. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik (Historical Archive), Volume 1, Issue 1, Dec 1959, Pages 269 - 271

26 Time Travel, Oskar Karlin, http://www.oskarlin.com/2005/11/29/time-travel/ [Accessed 04 Jan 2006]

27 Guardian Web Watch, http://blogs.guardian.co.uk/technology/archives/2005/12/03/time_travel_on_the_london_underground.html [Accessed 06 Jan 2006]

28 Time travel with the London Tube Map, http://rodcorp.typepad.com/rodcorp/2005/12/time_travel_wit.html

29 Travel Time Tube Map, Tom Carden, http://www.tom-carden.co.uk/p5/tube_map_travel_times/applet/ [Accessed 15 Jan 2006]

30 Tube Travel Contours, http://www.tom-carden.co.uk/p5/2006/02/tube-travel-contours.php [Accessed 10 June 2006]

31 In Transit 3D, Michael Pinsky, http://www.michaelpinsky.com/transit/vtwo.html [Accessed 13 Jan 2006]

32 Using desktop GIS for the investigation of accessibility by public transport: an isochrone approach, O'Sullivan et al, Int. J. Geographical Information Science, 2000, Vol 14, No. 1, 85-100

33 Dijkstra's Algorithm, http://en.wikipedia.org/wiki/Dijkstra's_algorithm [Accessed 11 June 06]

34 Texture Model Formulation, http://www2.imm.dtu.dk/~aam/main/node13.html

35 Voronoi Diagram, http://en.wikipedia.org/wiki/Voronoi [Accessed 14 Jan 2006]

36 Efficient Triangulation Algorithm Suitable for Terrain Modelling, Paul Bourke, 1989, http://astronomy.swin.edu.au/~pbourke/modelling/triangulate/ [Accessed 14 Jan 2006]

37 CONREC, A Contouring Subroutine, Paul Bourke, 1989, http://astronomy.swin.edu.au/~pbourke/projection/conrec/ [Accessed Jan 14 2006]

38 Correction of Horizontal Areas in Tin Terrain Modeling - Algorithm http://gis.esri.com/library/userconf/proc99/proceed/papers/pap924/p924.htm

39 Supergons, Urs-Jakob Rüetschi, http://www.geo.unizh.ch/~uruetsch/supergon.html

40 How not to lie with visualisation, http://www.research.ibm.com/dx/proceedings/pravda/truevis.htm [Accessed June 8 2006]

41 Envisioning Information, Edward Tufte, 1990

42 Ibid. p 94

43 Colour Brewer, Cindy Brewer, http://colorbrewer.org

44 London Underground Travel Times map, http://www.geofftech.co.uk/tube/sillymaps/travel_times.jpg [Accessed June 13 2006]

45 GML: A portable Graph File Format, http://www.infosun.fmi.uni-passau.de/Graphlet/GML/gml-tr.html

46 XGMML (eXtensible Graph Markup and Modeling Language), http://www.cs.rpi.edu/~puninj/XGMML/

47 GraphML Primer, http://graphml.graphdrawing.org/primer/graphml-primer.html

48 Representational State Transfer, http://en.wikipedia.org/wiki/REST

49 A revolution in routing, Sven Donaldson, https://www.oceannavigator.com/article.php?a=1249 [Accessed June 08 2006]

50 Processing, http://processing.org

51 London Tube Journey Planner, http://tubejp.co.uk/

52 Accessibility as an important wilderness indicator: Modelling Naismith's Rule, http://www.geog.leeds.ac.uk/papers/98-7/ [accessed June 08 2006]

53 Personal World Map, http://pzwart2.wdka.hro.nl/~rtorre/pwm/index.html [accessed June 06 2006]

54 Realtime travel news – Transport for London, http://www.tfl.gov.uk/tube/travelinfo/realtime/

55 Noninvasive three-dimensional activation time imaging of ventricular excitation by means of a heart-excitation model, Bin He, Guanglin Li and Xin Zhang, 2002, http://www.tc.umn.edu/~binhe/publications/ref2_phys.Med.Biol.pdf

56 Exploring Geographic Information Systems, Second Edition, Nicholas Chrisman, John Wiley & Sonces, Inc. (2002)