University of Kent

Rail Route

A web-based route validity checker

CO 600 FINAL YEAR PROJECT

TECHNICAL REPORT

RailRoute - A web-based route validity checker

Aled Williams *Agw9@kent.ac.uk*

Eleonora Klidzia *Ek50@kent.ac.uk*

Kyle Ratteree *Kbr3@kent.ac.uk*

Phi Dang *Phd4@kent.ac.uk*

Steven Daniels *Sjsd2@kent.ac.uk*

Abstract

Much confusion remains amongst train users concerning the validity of their tickets. This report details our attempt at limiting this confusion by providing a system that automates this process of checking for route validity between any two stations without the worry of having to understand the detailed rules of the ATOC Rail Routeing Guide.

1 Introduction – the problem

Rapid developments of the railway network during the industrial revolution was enormous; the ability to cover vast distances across the country in record times, and its increased accessibility to smaller towns benefitted the country's industrial growth and even prompted a general change from solar time to Greenwich Mean Time (GMT) to avoid missing trains. The introduction of 6000 miles of railway in the short gap between 1830 and 1850 caused considerable disorganisation to the railway company whilst trying to deal with the rapid advancements and nationalisation.

The company was eventually privatised, and broken down into 26 smaller railway companies with each managing its own separate region. ATOC (Association of Train Operating Companies) was then set up as a body that represents the 26 operating companies to the government and the media; forming standards and the rules that should apply – a questionably unambiguous document - that each railway company must follow in order to make the buying of tickets easier; agreements as to what constitutes a valid journey on varying tickets. This is where the problem arises. What exactly constitutes a valid journey on a given ticket? Many train journeys require you to change trains en route; also some kinds of ticket allow a "break of journey" where you can use a ticket to visit an intermediate station as part of a longer journey. It is not immediately obvious what journeys are allowed. For example if I have a ticket going from Edinburgh to London, am I able to go via Birmingham and take a break of journey at this intermediate station? It is not immediately obvious to the user exactly what is a valid route without first consulting the ATOC website.

2. Technical Report Structure

The rest of this report will be structured in the following format:
Firstly, a background to the current system and the problems associated with it, which intrinsically links to the requirements of our project which aims to improve this system. Next we will document our technical accomplishments in this project both with the coding (and its associated algorithms) as well as the front-end aspects (GUI applet and the website). We will then finish with an analysis of our system; what it achieved, its various limitations and what remains for future development.

2.1 Background

There exists a web-based rail routeing system in which the valid routes are described in a document called the "National Routeing Guide" on the ATOC website (http://www.atoc.org/index.asp).
Unfortunately the guide is rather complex, so it is not easy for a user to readily work out which routes may be valid. Finding the next train travelling between two stations is fast and simple using the national rail website, but obtaining the cheapest, fastest or a list of valid routes can be a time consuming process. Understanding and using the current routeing guide on the ATOC website is a laborious process where the user has to put in a lot of effort to understand the rules relevant to their route. ATOC chose to make the Routeing Guide part of the contract between us – the customers - and them when buying a ticket. We cannot change this contract; if we are found to be travelling on an invalid route on a given ticket we will be penalized. We aim to provide our end users with a system that can automate this process of checking route

validity for the customer given any two stations and be able to provide a list of possible valid routes, together with the fastest route available for that journey.

2.2 Requirements

The initial scope of our project is to produce a system that communicates and extends the information provided on the ATOC website to provide an automated service to:

1. validate itineraries
2. Obtain the fastest routes from one station to another

We also sought to possibly implement additional features regarding journey information that the user may find helpful once we familiarised ourselves with the initial scope and the conditions of a valid route, as specified by ATOC.

As well as these functional requirements we want to produce a user-friendly system to allow the user to readily access the information they are entitled to.

3. Planning and Management

3.1 Requirements Planning

Our first few meetings consisted of the brainstorming of the requirements; analysing the limitations of the current system and speculating what additional features may benefit the end user. We were aware that the set of requirements were likely to change as we became more familiar with the system, its boundaries and the constraints on the resources available to us.

3.2 Documentation and Management

Given the nature of the project we decided to use an iterative design process, with ongoing documentation of each iteration detailing the changing specifications as limitations were discovered and the scope fully comprehended. At the beginning of each iteration an initial specifications plan was drafted to provide a framework for the direction of development and as a rough measure of progress, which outlined specific features that we aimed to incorporate into our project and to ensure all group members were aware of the intricacies of the current system, its restrictions and what it implied to the project. A short evaluation was produced at the end of each iteration highlighting the main accomplishments, setbacks and limitations and to what extent the initial plan was adhered to. This also served as a basis for the specifications for the next iteration as ideas were adapted given the changing course of development.

3.2 Initial Plan

Having been introduced to the problem, we decided we needed a reliable method to represent and process the route information provided by the ATOC website.

4. Technical accomplishments

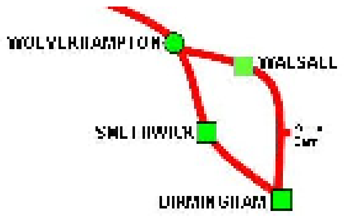
- System architecture

4.1 Back-end

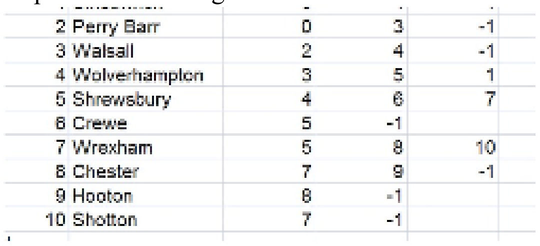
4.1.1. Working with maps

Researching route information on the ATOC website is a cumbersome task. Once information regarding associated routeing points is obtained, users must then look up corresponding maps containing those routeing points to figure out the valid route on the visual maps. Typically, a given station may be associated with several different routeing points (between 1-4) or may be a routing point itself

Ultimately, we aim to automate this process and manipulate the map data to return information that may be useful for the end user. We thus devised a method to reliably represent the information on a workable database, so that it corresponds to the information from the ATOC maps, typically illustrated below:



This information was represented on our database of maps on a spreadsheet for each map in the following manner:



2	Perry Barr	0	3	-1
3	Walsall	2	4	-1
4	Wolverhampton	3	5	1
5	Shrewsbury	4	6	7
6	Crewe	5	-1	
7	Wrexham	5	8	10
8	Chester	7	9	-1
9	Hooton	8	-1	
10	Shotton	7	-1	

Each station corresponds to a number, listed to the left of the station, and every subsequent column after the station denotes which stations are immediately connected to that station. From this we can develop methods to manipulate and process this information.

Since there is no raw data available to represent the maps in a computationally useful way, our only option was to manually input the data of each of the maps into its own spreadsheet in order that they may be represented and processed accordingly. This was a daunting task (over 90 individual maps), and one that was gradually completed over the course of the project.

Having completed a first map, we then sought to find a method to return the shortest route within that map given the data provided for it.

To manage this we implemented the depth first search algorithm, as seen in iteration 1, by creating a 2D adjacency matrix representing the connections between all the stations on that map, which the algorithm searched through to find the shortest path between two stations. An adjacency matrix was generated automatically from the information on the spreadsheets, with the edges of the maps being allocated a value of 1 or 0 to identify a connection between two stations. It was intended that travel times or distances as edge weights could be implemented in further developments; for the time being however, the shortest path returned between two stations was nothing more than the route with the shortest number of intermediate stations.

The next step involved the combining of two maps or as many maps as necessary to determine the shortest route over longer distances. In further iterations, a new class was developed to enable our system to create maps and combine them to form hybrid maps while preserving the connectivity details of each map. This was essential as many maps had common *overlapping* stations that are connected to different stations on different maps. Certain issues were discovered during this phase of development that related to the confusion of 'grouped stations' that led to complications of combining maps. This arose because on certain maps, some stations, for example Glasgow Queen Street, would be represented as a single station, while on others it would be represented under an umbrella of grouped stations, namely, the Glasgow group stations. We looked to resolve this issue by agreeing on a standard to represent all grouped stations across all the maps when they are combined. A simple method was devised to represent all stations that belonged to a group as a single representative station when maps were combined. This was based on the idea that travelling to any station that belonged to a group also permitted travel to any other stations belonging to the same group (thus from Ashford International to London Charing Cross, any route that ended in a station belonging to the London group was permitted). This however, produced complications of its own later.

4.1.2. Routeing points

Combining several maps enabled the system to search for routes between stations that were further apart; however, forming a large matrix of all the stations across all of the possible maps was not a feasible solution in finding a possible route for one particular journey given the computational resources to create and store a matrix of that size. Thus we needed a method of narrowing down the number of maps (and the number of subsequent stations) in creating our adjacency matrix, which is where routeing points come in. Every station has an associated set of routeing points (or it may be a routeing point itself), and the current system on the ATOC website requires the customer to manually look up the associated routing points of the start and the destination stations, and find a common routeing point between the two stations. If a common routeing point exists, then the permitted route is simply the shortest distance between origin and destination, regardless of whether or not it passes through a routeing point. If no common routeing point exists between start and destination (as is the case between stations that are further away), a further table had to be consulted to determine the correct maps to use in order to find a permitted route. A permitted route is specified as the shortest distance between two associated routing points. With stations often having more than one associated routeing point however, finding the shortest route can become a lengthy task. Our primary concern was to automate this process of looking up the various routeing points associated between the start and destination stations, and construct the relevant maps between all routeing point combinations, obtained from the information on the ATOC website (for example, travelling between a particular routing point combination of Cardiff Central to Birmingham required the maps BD + CE) and thus generate the required matrix to process the shortest route. The depth first search method would iterate over each of the maps created by its particular routeing point combination and return the shortest route for each (with the edges still all having a weight of 1). In order to automate the process of looking up routeing points associated with each station, look up tables were created in further spreadsheets from the information pasted from the ATOC website that details all possible routeing points associated with a station. A further look up table was created to form the required maps from any two routeing point combinations.

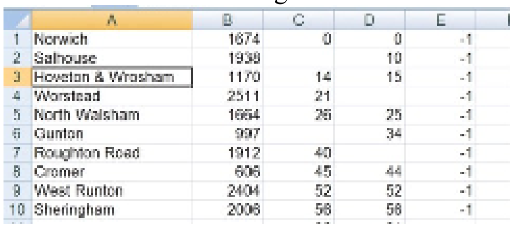
4.1.3. Extracting data from timetables

Maps provided an effective way to determine the connectivity of stations and the valid paths, but with edges that hold only Boolean values, the routes returned weren't representative of the distance or the length of time it would take to travel from one station to another. For our program to determine a valid route (given as the fastest route between two sets of routeing points), we needed to provide information regarding the length of each possible route.

To improve the system, we had a view to incorporate fare and timetabling data concerning all the routes in the UK into the existing matrix; giving a weight to edges between two stations. Potential Data files were obtained in several formats; a Portable Document Format (PDF) file from the networkrail.co.uk website, as well as time and fare data sent as raw data in plain text format.
The raw data obtained through plain text files devised by ATOC proved unworkable and were not meaningfully deciphered considering the highly specialised nature of the data and resources of our project and were thus discarded for a more practical method. Timetabling information provided via the PDF files (from National Rail), although easily deciphered, proved extremely difficult when trying to implement a process to extract the information automatically into a workable form; using the PDF java libraries from PDFBox led to inconsistent data being extracted forcing us to finally resort to more manual input into workbooks.

The limited time frame allocated to us meant that manually extracting all the data contained in the PDF file was impossible, and the scope was thus reduced: information being extracted was cut down to the timetabling content of a single hour window during weekdays, extracted manually from each of the 100+ tables. This approach we felt provided a good compromise between what could be reasonably achieved whilst providing meaningful content. A smaller scope meant that actual departure and arrival times were not included; instead the times were modified to represent the journey time in minutes for a given route between all stations, effectively stating the length of each route between stations.

The first step in manually extracting the data was to allocate all the possible different stations its own unique ID number, provided by a lookup table in another spreadsheet. Obtaining the actual timetabling data comprised of creating a single worksheet for each of the 100+ tables in the original PDF file and including every station with their index looked up in the All.xls file. The third part comprised of adding the time in minutes a train takes to travel from the source station to every other station on its route. The end result was formatted to a single worksheet containing over 100 sheets each mapped to a route table and containing all the routes within the defined hour window. A sample of the structure of the data is given below:



	A	B	C	D	E
1	Norwich	1674	0	0	-1
2	Salhouse	1938		10	-1
3	Hoveton & Wroxham	1170	14	15	-1
4	Worstead	2511	21		-1
5	North Walsham	1664	26	25	-1
6	Gunton	997		34	-1
7	Roughton Road	1912	40		-1
8	Cromer	606	45	44	-1
9	West Runton	2404	52	52	-1
10	Sheringham	2006	56	58	-1

The immediate column to the right of the station name was the station's unique ID number that would identify it as such on a created map. Subsequent columns denote the time value that the station has on a given route. Each column thus represented a route, with the time value of '0' indicating the start of the journey. Note, that the routes work backwards as well, thus '0' minutes could indicate the beginning or end of a journey.

4.1.4. Uniform-cost search

Using these worksheets, a process was created in the Generate class to produce a Super map detailing all the immediate connections that every station has to other stations, detailing the length of the connection and on which particular route it refers to. In turn these supermaps are used to lookup the fastest route between a given start and destination station within the Algorithms class using Uniform-cost search algorithm. The uniform cost search works by constantly expanding to nodes (stations) that are immediately connected to the current node, working with the heuristic that the next node to expand to is the one with the least path cost from its root (start) node.

This guarantees that the shortest route would always be found if the next path to be considered is always the one with least path cost from the root node, with the algorithm stopping if the destination is reached. For routes that traversed between stations over a wide network however, the algorithm may prove problematic as it will need to consider every possible path that is strictly shorter than the shortest path.

The implementation of this algorithm contained two main parts: the startProblem method and the expand method. The startProblem method initialised the departure station as a root node and looks up its immediately connecting stations by consulting the super map that was generated. Child nodes were then produced that contained information regarding current path cost and step cost from the parent node; the possible nodes for further expansion were placed in a queue sorted by its current path cost. The expand method takes the node with the shortest path to the route node as its parameter and produces a tree of nodes in a recursive manner, utilizing the same logic as the startProblem method. The expand method is recursively called until a child node matches the "end" string specified as the second argument in both methods, signalling the end of the search, the tree ranging from the end child node to the route node is then returned along with the path cost of each step giving us the fastest path from a start station to a destination. All nodes upon being expanded produces a list of other nodes up for expansion, with each of them added to a priority queue and ordered using the Collections.sort method after each expansion which in turn uses the compare method from the NodeComparator class. The end result is a sorted list of nodes ordered according to the pathCost, representing the time of travel, to order the queue, and finally the head of the queue is constantly removed from the queue and tested for the goal state.

It was important to note that this method required several considerations to work properly; whenever a node is generated as a child from its parent, as it is in turn expanded, its parent nodes is also included in the list of possible nodes to visit next. We thus had to prevent this from happening to avoid loops between parent and child nodes. Another complication arose because of the format of the supermaps; each station may have been connected to each other in more than one instance as the connections related to different routes. This problem required us to guard against duplicate child nodes as well as preventing parent nodes to be added to the queue.

Despite being able to deliver the shortest time taken to travel from one station to another, the system encountered several limitations; because of the specific implementation of the uniform-cost algorithm, the result obtained for routes only details which stations are passed through and the time taken to complete a journey, however the routes returned are very rarely single train journeys and in consequence do not represent an actual shortest path but rather an ideal time in which switching trains does not increase the travel time. Similarly the system is missing features to enable it to indicate to the user where a train switch is necessary and how long delays at stations may last, however these may be implemented in the future simply by extending the algorithms used to return more complete route information from the tables.

4.1.5. Limitations

Our method initially produced a list of possible routes between two sets of routing points using varying map combinations. This sometimes produced a vast array of possible routes that may have included nonsensical extensive routes that simply were not practical. The current system relies on the user to select the appropriate routeing points in order for a sensible route/map to be selected whereas our system returned a whole array of possible routes that could conceptually be created. We therefore chose to apply our shortest time algorithm to a selected route that the user would choose once the initial list of possible routes is returned. Thus from a route between Inverness and Bathgate, a possible route that was returned would be 'Inverness – Aviemore – Dalwhinnie – Pitlochry – Perth – Stirling – Falkirk Group – Bathgate. If the user were to select this route, the shortest route algorithm would be applied to get to the next intermediate stop, and then again to subsequent stops until the end of the route is reached, thus an end result may read: Inverness (0), Carrbridge (34), Aviemore (42), Kingussie (55), Dalwhinnie (67), Blair Atholl (95), Pitlochry (104)… until the destination is reached. Our first limitation was due to the fact that the shortest route algorithm was a computationally slow process and could not be feasibly applied across a whole set of routes to determine the times for all of them, only a single selection at a time. The second major limitation related to the way the map information was represented; it was decided earlier on that grouped stations would be represented by a single station which conflicted with the timetabling information that listed all stations as individual entities. To get around this problem, the shortest route algorithm would have to be applied to every station belonging to a group to the next stop, with the shortest one returned after they have been compared. This massively increased the processing time required to compute any routes that went via a group of stations, sometimes taking a processing time of 10 minutes when a route goes through the London group.

4.2 Front-end

4.2.1. Website

It was intended that the end user interface was to be implemented via a web applet, where solutions regarding journey times and possible routes would be returned to the user after entering a query in the web applet. Certain security issues however, prevented us from completing this phase as the applet prevented external java class libraries from being successfully loaded (our project was heavily reliant on the JExcel libraries). A website was nevertheless developed before these issues came to light, and now exists as an introduction to the rail routeing problem and our aims.

4.2.2. GUI java

One of the primary aims of the GUI was simplicity and user-friendliness, which was used as a guideline when implementing the system's functionality. Composed of three dropdown menus to select the start and destination station with the option to go via an intermediary station together with a text box for the search results, the layout proved to be very simplistic to prevent ambiguity during its use. When entering origin and destination station names the user is assisted with an auto-complete text-field as well as a drop down menu to provide more precision and to prevent invalid strings being inputted. To initialise this, the main method of the 'BasicGui' class is invoked which would display the GUI and prompt the user to input a journey to query. Once a journey is inputted, the 'Get Routes' button is pressed which would return a list of possible routes indexed from 1 onwards. The user is then required to select one of the possible routes by inputting a number indicating the desired route into a separate text field. The Uniform-cost search within the Algorithms class is then used to calculate the time of travel between all stations on the specified route and finally returned as complete journey information, including names of each station on the route as well as the time taken to reach each of them and the destination.

5. Conclusions

5.1 Accomplishments

Despite the limitations of our system and the various problems associated with finding accurate (useful) train times, we nevertheless managed to implement successfully the basic workings of the ATOC website; the association of routeing points to stations and the representation of its maps on a reliable database. Our initial decision to represent a group of stations as a single entity proved to be costly in our development, with complications arising later that severely limited further progress, since it was a lot harder to mesh with the timetabling information. We resolved this problem slightly by implementing an option to go via a particular station; this narrowed the search results returned to only include those routes that contained this intermediary station. Problems still remained however when the times were obtained for that route in that they did not convey complete route information; how many changes are needed and how long those changes took. We feel that given more time/resources and perhaps some hindsight, a fully informed search result detailing the precise train route information could be implemented. Given the data available to us however and that no feasible method for automating the process of data extraction could be devised, we nevertheless managed to map the database from the ATOC website successfully and used the routeing points information meaningfully, with some integration to timetabling information from the National Rail website.

5.2 Further Developments

Areas of the project to be improved upon include the modification of maps to represent all individual stations for a more precise representation of the problem space which would lead to easier integration of timetabling information and to include more detailed route information when devising a shortest route method between two stations; that included information regarding the changing of trains, information that would be of use to the user.

5.2.1. Cheapest Route

The difficulty of checking the validity of routes is not the only problem associated with the current system, important functionality is still lacking or difficult to extract. It was not possible to implement any features relating to the pricing of a certain ticket since that information was not available to us. The different prices that companies charge per mile produced complications when trying to devise a cheapest route. If a cheapest route feature were to be implemented, a comprehensive tariff regarding the various intricacies of the pricing structure would need to be obtained from a reliable source that could be reasonably read and stored for processing.

5.2.2. Distance travelled We have already allowed for this future provision; currently the distance between each station is set as 1 unit in the matrixes, and currently the algorithm adds up these units to produce a distance which is equivalent to the number of stations as the distance between each station is 1. It would be easy enough to change this unit in the matrix to appropriate data mileage data, and therefore the algorithm would automatically calculate the distance without any further changes. The data was readily available enough though through the time tables provided on the National Rail website. The feature was not implemented however due to the extensive manual labour required to extract the distances between the stations and the raw data file from ATOC proved unworkable.

5.2.3. Complete timetable data Complete timetabling data would enable the system to produced precise times during the day when a given train is operating. Currently, only a small cross section of times was derived from the timetables to represent the journey lengths and not the actual times itself. A fully implemented feature that would convey precise timetabling information would require that we had a database containing all the train times across the country which lied beyond the scope of the project.

6. Acknowledgements

The authors would like to primarily acknowledge and thank the contribution of Colin Johnson for his supervision and guidance throughout the project, providing a time for us to collate our ideas and demonstrate our developments. Secondly we wish to acknowledge the contribution of those people who have attempted to produce a similar solution and given us ideas on how to improve the current ATOC system, as well as ATOC themselves for providing us with sample data of times and distances on CD. Finally, we thank the contributors to forums which provided us with great help.

7. Bibliography

[1] Robert Ayres, *"Essence of Professional Issues in Computing",* Prentice Hall, 1999,
[2] http://en.wikipedia.org/wiki/Look_and_feel
[3]http://en.wikipedia.org/wiki/Human-computer_interaction
[4] http://www.nationalrail.co.uk/
[5] http://www.atoc.org/rsp/Routeing_Guide.asp
[6] http://www.w3.org/
[7]http://mars.wnec.edu/~grempel/courses/wc2/lectures/industrialrev.html