Project Completion Report

DelayExplorer
A Proof Of Concept For Visualising Train Delays

Introduction

Visualisation of data is an important activity in many fields such as scientific research, mathematics and engineering, and business. By leveraging the power of the human visual system, visualisations enhance not only the understanding of data and its interrelationships, but also the analysis - a picture is worth more than a 1000 row table of numbers or text, let alone a 5 million row table.

Given how a modern transportation system such as the UK rail network is extensively computerized (and with additional IOT systems in the future), there is (and will be even more so) a very large amount of associated data, i.e. "Big Data". A sophisticated tool for analysing and visualising this data is therefore necessary - not only for academic research purposes, but train operators and other companies, their business analysts, and even rail operational staff in relation to day to day operations.

DelayExplorer is one such proposed tool. Its core idea is that train services and other associated concepts are best stored, analysed, and visualised as graphs, not just tables. A rail network is a graph, a train service is a path within that graph (and a graph itself), and the chain of delays when one train delays another and so on also a graph. Besides the primary or initial delays to a train (e.g. due to a large number of passengers getting onto a train), there are reactionary delays. They are ones that are knock-on effects caused by another part of the system, typically another train that is running late. For example, if a passenger refuses to pay or someone trespasses on the tracks, that is an incident that causes a primary delay to a train. That train, now running late, causes other trains to run late due to disrupting their schedules and thus reactionary delays follow the primary one. The term “delay propagation network” or “delay network” summarizes this concept of how one initial delay cascades and creates a chain of reactionary delays. DelayExplorer thus allows users to easily visualise delay networks along with providing analytical functions to understand the data, its relationships, and ultimately the behaviour of trains and the processes underlying the UK rail transportation network.

The initial idea for DelayExplorer grew out of a grant awarded to UEA from the Rail Safety and Standards Board (RSSB) as part of its 2017 Data Sandbox competition. This grant was for investigating how machine learning and Artificial Intelligence could be used to reduce or prevent reactionary delays in the UK rail network. In the competition brief, Network Rail stated they did not have a system by which reactionary delays other than the most immediate one following a primary delay could be deduced. A question that logically follows from this is "how can reactionary delays be predicted? Is historical data of any use in that it shows common patterns of behaviour for trains over time?". Thus the idea for DelayExplorer was born.

Part of the research grant involved developing a prototype to address the most basic questions: analysing Network Rail's Historical Delay Attribution (HDA) data, deriving graph structures from it, and developing software to demonstrate the concept. Staff from the rail industry were enthusiastic about the system, as crude as it was, because its value was immediately obvious and they did not have any comparable system nor had ever seen one. On this basis, an application was made to UEA's development fund to further develop the system, specifically focusing on further refining the requirements for a commercial system, improving the user interface, and also investigating how to best layout the graph diagrams to address common questions related to analysing the data.

The rest of this report details the development of this version of DelayExplorer. The next section is an overview of the HDA data, issues found, and an explanation of how it is turned into graph based data. Then a summary of the key requirements for a commercial version of DelayExplorer is presented in Section 3 and Appendix A contains more details. Section 4 covers the development work done so far, focusing on the research done into multivariate graph layout algorithms and how different types can be used to visualise the data for different purposes. The improvements to the user interface are also discussed. Finally, there is a summary which briefly describes the work still required and possible future applications.

Historical Delay Attribution Data

As part of their transparency efforts (https://www.networkrail.co.uk/who-we-are/transparency-and-ethics/transparency/datasets/), Network Rail periodically publishes a CSV file of information about delays, e.g. what train was delayed by which train, how long it was, and to what TOC the delay was attributed. This CSV file is essentially a copy of records from the Network Rail database system used to track this information and, for the purposes of this project, deemed to be authoritative.

Starting with the railway year 2014-15 (a Network Rail year is composed of 13 four week periods which starts every April 1st), a total of 6 years of data has been collected, except for the last two periods of 19/20. The total number of rows is about 30 million; a basic statistical analysis of the data along different types of axes can be found in Appendix B. Each row of this CSV file is essentially a record of a delay event containing the ID of the train that was delayed, where and when the delay happened, the length of the delay, what train (if any) delayed it, the category of delay, and the ID of the associated incident that caused the chain of delays. There are other minor fields as well; descriptions can be found in Appendix C.

Unfortunately, as with all real world data sets, there were issues in processing it into our own relational database for storage and basic analysis. Network Rail has changed the format of the CSV file repeatedly over the years (e.g. the format of dates) so a notable amount of time has been spent doing basic data munging tasks. Another example would be that, for the last 6 recent periods of data, they have not only changed the ordering of fields, but now have added new columns. Therefore, any further development should account for this ongoing issue and allocate time towards maintaining and upgrading the Unix shell script that has been developed for cleaning and processing these raw CSV files. The alternative of acquiring the data directly from the right division of Network Rail would be the ultimate solution, but how to accomplish that and data ownership questions arose, so using publicly available data was decided upon.

The process of constructing the graph data is a straightforward one. Basically, each row in the CSV is a segment or two edge and three nodes of a graph - the delay (which is a vertex) is connected to the two associated train nodes. Once a train node has been created, it of course can be connected to other delay nodes (delays to that train that happened in the same period as the first delay, or other ones the train caused) and thus the entire history of the train can be constructed along with the delay networks it was involved in.

The graph database is thus comprised of delay networks that happened on each day, connected to each other through the trains in common. There are also vertices representing stations and timing points that trains and delays are connected to (timing points are locations along the rail tracks that record when a train passes). Additionally, there are nodes that represent incidents or problems that cause initial delays; these serve as the root node of a delay network. Their ID numbers are extracted from the HDA data, but Network Rail does not make extra internal information about them publicly available. Future work will involve incorporating this information when it becomes available as well as adding vertices that represent the planned schedules of trains and the actual performance of a train.

As a consequence, complex queries based on all this information are easily constructed. If the data was in a relational database, these queries could certainly be supported, but they would be extremely complex and convoluted. This is the benefit of storing the data in a graph database - instead of recursive SQL to generate a very long set of table joins with a number of case, if, or when clauses, only 5 lines in a graph query language are needed. A simple example would be finding the trains that were delayed downstream from the starting train an hour or two after the initial delay; this is not a linear process because it essentially is a depth and breadth based search through a graph. Relational databases are not ideal for storing graph structures.

It is anticipated that this ability to query the database in a manner orthogonal to table based queries will result in the rail industry realizing questions that have never been raised or even thought of could be answered; relational databases (and perhaps NoSQL) are predominantly used by Network Rail, but the way they work naturally limit or guide the types of questions that are asked.

Other issues that have been addressed by this project have involved improving the generation of the input files that are loaded into the graph database. The work on the first prototype consisted of figuring out the overall process and setting up the tables and data. This current project involved improving the performance of the SQL (there is a sequence of very large table joins), adding more raw data to the system (e.g. the Activation records for trains that contain trains' schedule IDs), and finding errors in the process and data. One notable example is delay records that do not immediately make sense, e.g. a train is recorded as having been delayed by a train that ran the week before it did. The last 2 digits of a train ID are the day of the period that it was running, so if the affected train and delaying train days are widely different, then this is a possible error. Some instances are obviously data entry errors, e.g. there are two records entirely equivalent except for the delaying train days being 20 and 30 (the 20th and 30th of the month). Why the first record wasn't deleted is an open question. Other errors are not obviously data entry errors and only the description of the delay is a possible clue as to what actually happened. But those descriptions are very short and filled with industry jargon, therefore rail industry staff will be needed to fully understand these quirks in the data and if corrections need to be made.

Another issue that has been addressed is the proper incorporation of BUGLE data into the system. BUGLE (and other systems like it) are used by the Train Operating Companies (e.g. Greater Anglia) to record data they need about delay incidents. This information supplements that available from Network Rail. Greater Anglia supplied UEA with their BUGLE database for the past few years, so the first prototype used it as additional fields for the Network Rail incidents it could be matched with. For this version, a design decision was made to make BUGLE information distinct nodes that are linked to Network Rail's incident nodes. This change was made for two reasons. First, the "classes" of the nodes were not entirely compatible and so the change streamlined some operations. Second, when additional BUGLE type data is acquired from other TOCs, making them distinct nodes will assist in a problem related to storing company proprietary information in a database used by multiple companies. Ideally, a commercial system for the UK rail industry would just be one system or database that contains all of the Network Rail based data; having multiple systems that just essentially copies of one another creates the opportunity for problems related to the timely updates of data. But proprietary information such as BUGLE should not be accessible by other companies even though it may be in the system. Essentially, this is an authentication and authorization problem which is a well understood topic in Software and Systems Engineering; redesigning the graph database schema to account for this upcoming issue was the most opportune time.

A final change made to the Extract, Transform, Load (ETL) processes which add data to the database is the mapping of multiple freight train IDs to a master ID. Freight trains frequently change their IDs on a single day for various reasons and these changes should be tracked so that individual train nodes can be linked together. This enables freight trains involved in multiple delay networks on a single day (under multiple IDs) to be more easily identified. This change involved writing extra SQL procedures to perform table joins with the table containing Change Of Identity messages when the input file of train IDs is created.

The already anticipated future work in regard to processing the HDA data is twofold. First, the data from other systems like BUGLE will need to be incorporated, but as long those systems can output data in CSV files, it should be straightforward to load the data. The fields will have to be matched or mapped to the BUGLE fields, but this should not be difficult given what is already known. Second, internal information Network Rail has about incidents should be incorporated along with whatever information the industry deems necessary. But that is just more straightforward loading of data as well; there are no indications of a need to further modify or extend the process developed so far for creating graph data from the HDA data, except for developing a filter to mark unusual records for evaluation by a human.

Specification of DelayExplorer Requirements

As mentioned in the Introduction, there are several distinct groups of potential users of a DelayExplorer type system. The type of user and their exact needs are important variables that influence the functional requirements. For instance, academic researchers have different interests and goals than a business' business analysts; Network Rail's delay attribution department has different concerns than signallers at control centres. Therefore, a one size fits all software system would be impractical because then users in one category would have to learn to ignore functionality that is not relevant to them and that is a substandard design for a user interface. Instead, variants of DelayExplorer could be readily assembled by creating functional "building blocks" and a modular design.

The first step in the process of developing such a system is a functional analysis. "Use Cases" or "User Stories" are a common starting point, but these can be overly elaborate and only useful when little is known about the domain problem. Instead, a useful starting point is a concise description of the system and its functionality: "DelayExplorer is a system for visualising and analysing the relationships between train delays in the UK rail transportation network using graph data structures as the fundamental basis". Each of these nouns and verbs can then be further analysed to create more specific requirements. This section lists the generic functional requirements for any version of DelayExplorer (as well as the high level system engineering requirements); a brief explanation of each is included. Appendix A provides more detail about the requirements specific to each type of user.

Fundamental Requirements

Graph database software

Neo4j was initially chosen based on a pre-existing familiarity with it in order to get a prototype quickly working. It was the first graph database on the market, but more recent products are trying to address its limitations and so should be investigated; there are no specific advantages to Neo4j.

An ETL process for importing Network Rail HDA data into the graph database

As discussed, this process and the code has been refactored and only maintenance work is foreseen.

Related data such as locations, Network Rail and BUGLE incident data, and their ETL processes

This data has also been collected and the current HDA ETL process incorporates it as part of the load.

Databases for storing planned train schedules, train performance, weather, and Nexala train data

This data has also been already collected and stored into UEA's RailML database system. Future work will involve integrating appropriate database drivers into DelayExplorer to fetch data as needed.

A browser based UI for displaying the graphs (e.g. delay networks, the rail network, train paths)

A web based architecture was chosen over developing a desktop application because then certain questions and issues could be avoided (e.g. Mac or Windows?). Network Rail and TOCs already currently use public train oriented websites (e.g. Raildar) in their day to day processes, so integrating DelayExplorer in the same manner would be straightforward. Security issues and concerns can be addressed through any number of standard techniques. Browsers also have built-in capabilities that would require extra development time to emulate in a desktop application.

Display of trains, train services, delay instances, incidents, and locations as different labelled icons

A noticeable deficiency of some existing graph visualisation software is the simplicity or crudeness of the graphical elements; computer graphics has progressed since the 1980s. This is another motivation for developing a browser based UI - existing Javascript libraries have extensive capabilities for developing high quality UIs that can display nodes as more than just unlabelled coloured circles.

Custom layout algorithms for displaying delay networks

Section 4 will explain in detail why the standard graph layout algorithms that exist are not sufficient. This primarily has to do with delay networks being multivariate graph networks, meaning nodes and edges have attributes and so there are multiple angles from which the data can be viewed or visualised. For instance, a time of the delays focused layout is essential, but there are no existing algorithms to accomplish this.

The ability to display a delay network in a tabular format

This function is one already developed as part of the prototyping work in the RSSB grant. Displaying a delay network as a table rather than a picture of circles and lines may have advantages in certain situations, depending on the type of user, their tasks, and how quickly they comprehend the information. Further research is needed, i.e. a literature search to find any equivalent research and a study to compare and contrast the different UIs and users' reactions to them.

Support for different types of searches through the database

Each type of node (incident, train, delay, and location) can be the basis for a number of different questions asked of the data. Each type of search also comes with a number of possible parameters and the time period being searched is another significant variable. Therefore, elucidating all of these possibilities is out of scope for this document; the current DelayExplorer manual contains a description of these search functions as implemented in the protoype, but needs to be updated.

Support for displaying other types of charts such as line graphs or scatterplots

Currently, it is not clear to what extent charts of other data will be necessary (e.g. weather data for a geographic area where delays are being examined), but there will be some need. There are numerous Javascript libraries for implementing this functionality (e.g., Plotly, Highcharts) and they only need to be integrated into the DelayExplorer system and connected to the relational databases holding the data.

The ability to save images of delay networks and save/load data in graph file formats

Saving images as PNG or JPG files has already been supported as well as saving data in JSON. Other formats to be supported include GEXF and GraphML so that delay networks can be imported into other graph analysis software such as Gephi. SVG is another possible image format, but further research is needed.

Support for multiple monitors

This requirement enables users to organize the display of information as they see fit (versus a cramped single window on a single monitor). It implies multiple browser windows or at least tabs in a browser. No extra software development is necessary because of how browsers work, but it does guide how user sessions should be handled by the software. Modern PCs, OSes, and display cards can typically handle multiple monitors, so there are no extra or unusual hardware requirements.

UI support for basic editing functions of the displayed graph

The term "editing functions" includes the ability to move nodes around, select groups of nodes, pan the display, add/delete/hide/show nodes, zooming in and out, etc. These are standard functions that come with any visualisation software library that handles the display of graphs.

UI support for right-clicking on a node to get context specific menus to perform functions on that node

UI support for modal or modeless windows

This is a design choice about how the application should "look and feel". The design of the first prototype had a static modeless window on the side with forms to fill out to perform different types of searches, but the UI became cluttered due to all the choices and it took up valuable screen real estate. The classic menu bar and drop down menus style of design is an alternative, but strictly using that could result in unnecessary complexity comparable to a Microsoft Office application. Modern Javascript UI libraries already support this type of feature so the work will merely involve creating the windows.

UI support for displaying delay networks in 'layers' as well as a flattened format

This concept is much like of Photoshop or other image processing software. This will allow multiple delay networks to be viewed and analysed, e.g. everything involving a specific train service over a long time period. The core idea is to be able to compare delay networks by laying them over the top of each other; differences will then be highlighted through the use of different colours or drawing styles. Conversely, flattening these layers will provide the opportunity to analyse the networks as a single conceptual unit.

UI support for displaying networks of delay networks in a hierarchical manner

Preliminary research has revealed delay networks on a single day to be highly interconnected, i.e. a train involved one delay network because it was running late was involved in another delay network earlier in the day that made it late. Therefore, displaying all the connected delay networks will inevitably result in a very complex graph and hundreds if not thousands of nodes and edges. The ability to zoom in and out in a hierarchical manner is a common feature of graph visualisation software because it alleviates the problem of these 'hairball' graphs.

Display of a delay network on a geographic map, coded accordingly to clarify the time dimension

This feature was developed in the prototype in response to rail staff queries, but its actual utility is debatable. Knowing the spatial location of where a train was delayed is useful, but the order of events is also important. Otherwise the user merely understands where delays happened, but not the cause and effect or sequence of them. The time information can be encoded either thru shading of icons (e.g. darker implies later in the day), labelling them, or animation. The prototype currently animates the display by drawing the delay icons in the right order, but further consultation with rail staff is necessary to determine what is actually useful. A more useful option would be to display multiple delay networks over a time period, resulting in a heatmap of how frequent delays are relative to a location.

Graph database of the UK rail network (e.g. stations and track)

This is not an essential requirement, so it has not been currently addressed. But having a graph version of the UK rail network in the database will be useful in several ways. Currently, the only geographic or spatial location way of viewing the delay network data is by using the imprecise latitude and longitude data associated with stations and timing points. But it is incomplete and also reveals nothing about the relationship between delays and the tracks - Network Rail only has track information suitable for a GIS system which is unusable by DelayExplorer. Having a graph of how stations are connected and how trains

were scheduled to go along those paths (the train path) would elucidate the relationships between trains, their paths/schedule, the tracks, and the delays and where they occurred. In graph terms, the delay network would be overlaid on the graph network of the UK rail system; this extra level of detail could possibly reveal new insights about why trains were delayed where and when they were.

Display of a delay network on an abstract representation of the rail network that shows train paths

This display is also not an essential one, but as it explicitly links delays, trains, and the tracks together, it should be more useful than a display focusing on the geographic location of delays. The caveats about time and the geographic display also are applicable as well as the ability to display multiple delay networks to create a heatmap.

The Development Process

This section describes the design and development of the current prototype (version 2) of DelayExplorer, specifically its system architecture, the user interface, the search functionality, and the custom graph layout algorithms that were developed. The differences between this version and the first proof of concept (PoC) are also detailed including the motivation and rationale behind the changes.

System Architecture

Figure 1 is a block diagram of the DelayExplorer system architecture. It is the standard three tier architecture common to web based systems: the presentation layer (the browser), the application layer (the web server) and the data layer (the graph database). The first PoC system also used this architecture; the differences lie in the exact responsibilities for each tier in the two systems.

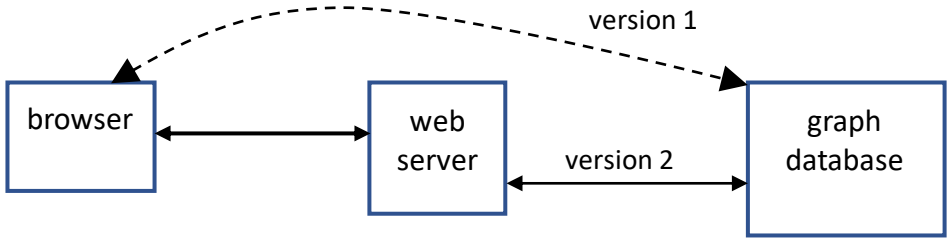

Figure 1: DelayExplorer system architecture

In the first PoC, the browser was a "fat" client in that a lot of the functionality not directly related to presentation was also implemented in Javascript. A good example would be the SQL-like Cypher statements sent directly to the Neo4j database; they were constructed using Javascript and the web server was bypassed entirely in the process of fetching graph data. This is because Neo4j has an optimized Javascript based driver more efficient than standard HTTP - the web server's only responsibility was to serve up static files (HTML and Javascript). This is an acceptable way of assigning responsibilities to each layer, especially for a PoC. But long term maintenance of such an architecture would be troublesome - the code to dynamically create Cypher statements can be convoluted. Changing to another graph database system would also require a large effort in refactoring the code as Cypher is specific to Neo4j. An alternative would be to create a Java based subsystem for Neo4j that can act as a HTTP based interface to implement an RPC or stored procedures type of functionality. That design would have required more time than what was available.

In this current version of DelayExplorer, the browser is a "thinner" client in that it is no longer responsible for assembling the SQL. It sends query parameters entered in a form on the browser side to the web server. But the Javascript to manage the visual elements and interact with the browser is still present. As for the web server, it now runs a Python process built upon Plotly's Dash data visualisation framework which has been enhanced to handle the responsibility of interacting with the graph database. It accepts the form data and processes it into Cypher queries akin to stored procedures. The web server also serves static files referenced in the HTML Dash generates. Nothing on Neo4j's side has changed; it still accepts Cypher statements and returns graph data.

There were several motivations for using Dash. First, it is a framework for building web based applications exactly like DelayExplorer and so it comes with a number of useful plugins. The most important one is Cytoscape - Cytoscape.js is a Javascript library for displaying and managing graph based data in the browser. It replaces the obsolete Linkurious/Sigma library used in the first PoC; that was buggy, is no longer maintained, and only a commercial version is now available. Dash also comes with Highcharts and Plotly plugins (for line charts, etc.) as well as support for D3 (the most widely used visualisation library).

Using Dash therefore makes an entire ecosystem designed for visualising data available (versus trying to choose the best of the numerous libraries and gluing them together individually).

Second, as Dash is based in Python, more functionality appropriate to the application tier could be implemented. There are Python based drivers for Neo4j that make constructing Cypher queries easier. Another benefit is that the application layer can translate between Neo4j JSON and other formats as required. Also, if a cluster of Neo4j instances is used, browser clients would not be required to understand that - they only would deal with the one web server. Using Python to interact with the relational databases that hold data such as train performance data will also be advantageous because of Python's various libraries for manipulating data, e.g. numpy. Finally, another example would be the integration of NetworkX, a notable Python library for analyzing graph data. Doing this for the first PoC by implementing the equivalent of NetworkX in Javascript would have been possible, but far too much effort; the library JSNetworkX does exist, but is not finished.

In short, developing an application based primarily (or entirely) in Javascript is possible, but there are a number of tradeoffs including extra development time, extra tools and the Javascript code ecosystem is very fragmented (what framework? React, Angular, Vue? use Node.js? use jQuery?). Most importantly, the benefits of a Javascript entirely application are not immediately obvious. A web server will always be necessary, so it makes sense to offload functionality to it when possible.

User Interface

Figure 2 is the user interface of the first prototype and Figure 3 is of the current prototype. Both are displaying the same delay network.

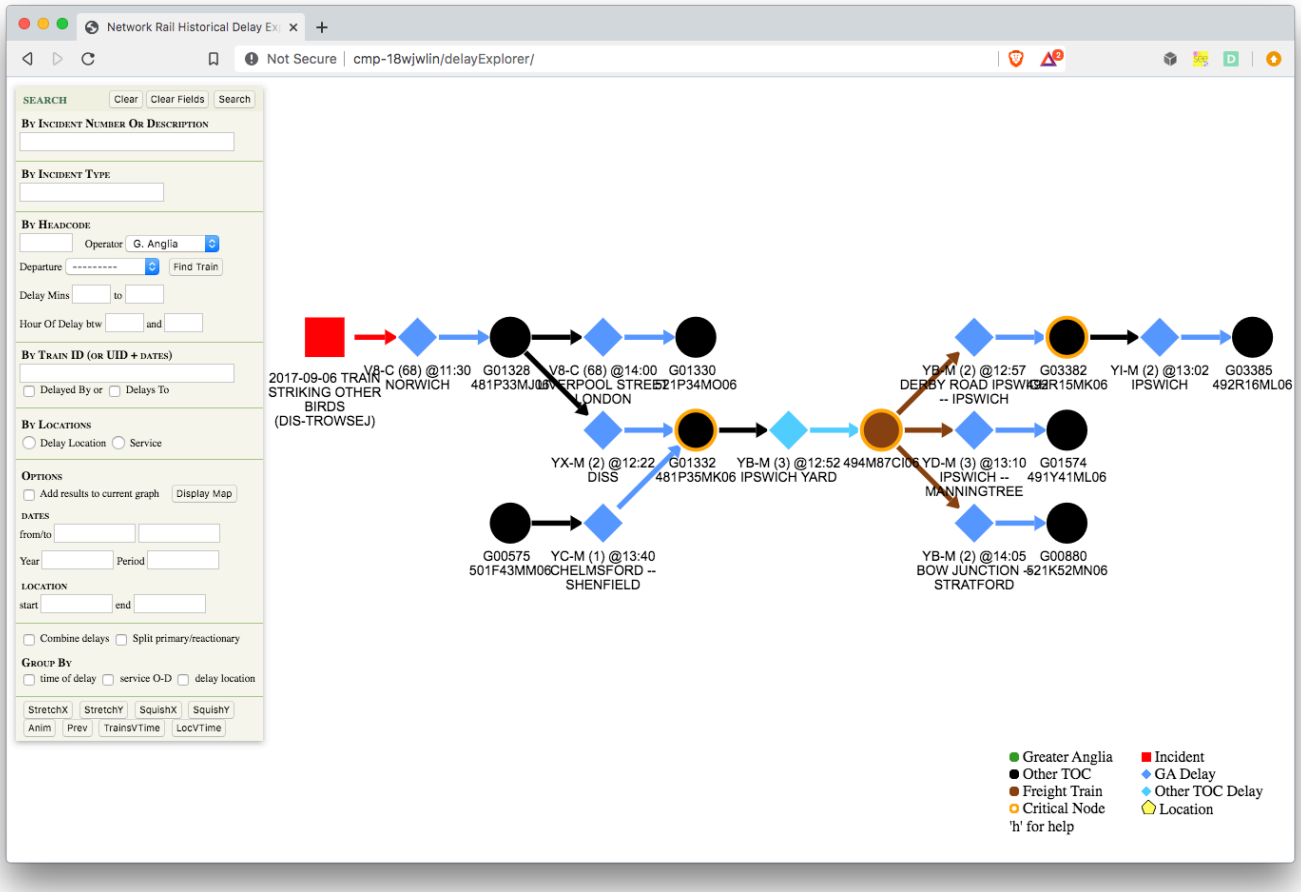

Figure 2: First POC User Interface

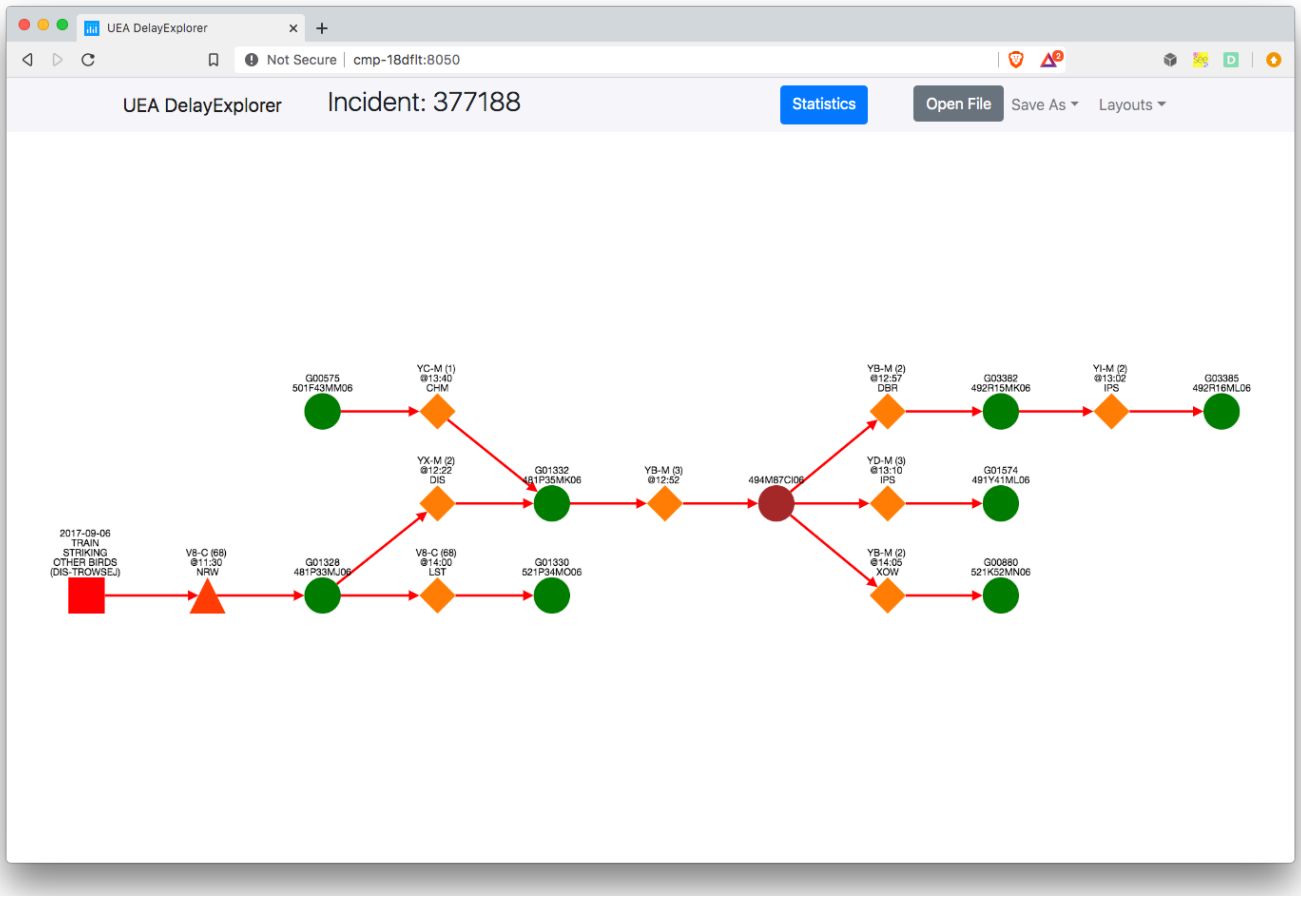

Figure 3: Current POC User Interface

The always present search forms in the first POC are now hidden and accessible through Ctrl-<something> - this change now ensures the most screen real estate as possible is available; showing all forms was also getting confusing for users. Commonly used functions like opening up another tab of statistical measures about the current graph, saving (or opening) the graph in different formats, and switching to different layouts are now easily accessible as well.

Search Functionality

Figure 4 is an example of the new interface for the search functionality. The large static set of forms has been broken up and moved into individual modeless dialogs that can be opened or hidden as need be. Further work is needed to implement all of the forms, but a library of common code to handle each type of field is being built up, making the development process efficient. The initial analysis of what types of searches could be done is still valid and no new fundamental searches have been devised; the following list of questions are ones the search functionality is based upon.

1. What incidents match a certain set of attributes and what are the other nodes of the delay network?
2. What trains were delayed as a result of an incident, and when, where, and how did the delays cascade?
3. For any train (or train service), where is it typically delayed, when, and what other trains does it affect?
4. For any location where a delay happens, when do they typically happen and to what train services?
5. For any train service (based on origin/destination), where do delays typically happen and when?

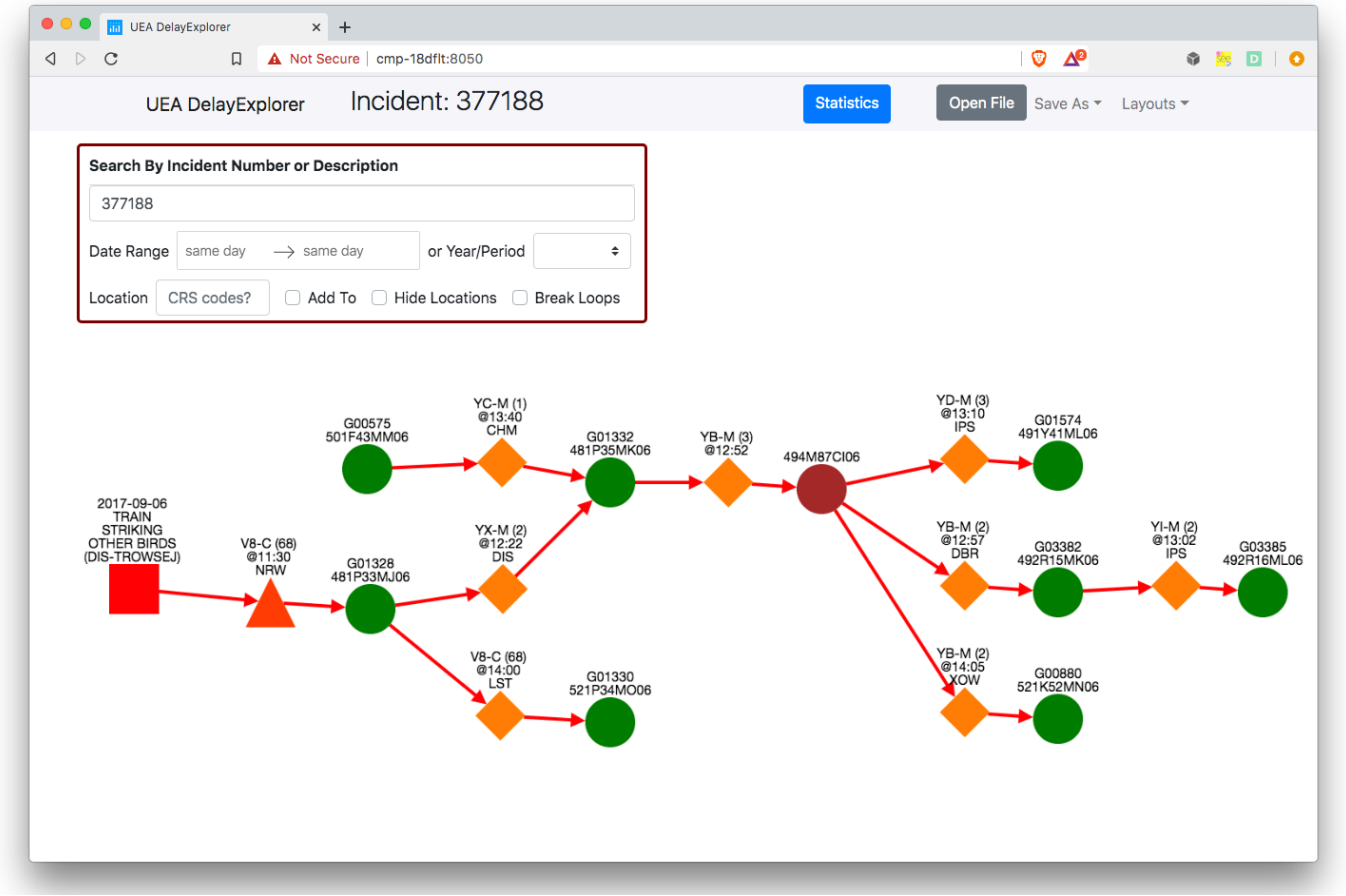

Figure 4: Example Search Form

The delay network in Figure 4 is the same network as in Figures 2 and 3; the difference is the version of the directed acyclic graph layout algorithm used. Future work will be focused on creating new types of graph layouts or visualisations of the data to manage situations where there is a large amount of data resulting the classic graph "hairball" problem.

Custom Graph Layout Algorithms

Research has been done into the development of graph layout algorithms for several decades with algorithms falling into two general classes. The first group contains algorithms that position nodes and draw edges based on optimizing the aesthetics (e.g. reduce the number of edge crossings or compact the size of the image). The second class are the spring or force layout algorithms that treat the data as a group of objects (nodes) connected to each other thru mechanical and/or electrical forces.

But both of these classes do not deal with node attributes; they are for univariate data and are concerned mostly with the relationship between nodes. General purpose software libraries implementing layout algorithms focus exclusively on these standard algorithms because dealing with multivariate data necessitates domain specific knowledge about the nature of node attributes. A general purpose layout algorithm that can handle domain specific data in an abstract way, yet create meaningful domain appropriate layouts, would be cumbersome to implement and to use.

So these two classes of algorithms are of only limited use given the specific requirements for visualising delay networks in a meaningful way. These requirements are examined in the subsequent sections that discuss the utility of each of the current set of DelayExplorer layout algorithms. Research was done into domain specific algorithms for inspiration, but most were found to focus on biology and bioinformatics; nothing related to transportation research was similar to the needs of visualising delay networks.

The layout algorithms currently implemented can be grouped into five different categories based on the general intent behind the layout and the node attributes used in positioning nodes: grid, logical relationship, node importance, time order, and geographical. For example, the "logical" category emphasizes the logical relationships between nodes (cause and effect) while "geographical" use the latitude and longitude attributes of delays to place them appropriately on a map. An important aspect of these layouts is that the usual criteria about aesthetics (symmetry, avoidance of edge crossings, minimization of the aspect ratio, uniform edge length, etc.) is de-emphasized in order to make the graphs more "readable". "Readability" is defined as the ability to visually trace the connections between train to delay to another train easily so that understanding relationships is clearer versus having a nice looking but cramped image. Therefore, the number of edge crossing is not a concern as long as there is sufficient empty space around nodes and the angles made by two edge lines are as convex as possible. The graphs are also ideally as spread out as possible given the extremely complex diagrams that have been found so far. All of these algorithms have various settings which can be adjusted to fine tune results to improve spacing, node placement, etc. Future work will involve developing forms that allow the user to adjust these settings dynamically.

Grid of nodes

This layout is the simplest and is used to show the results of a search on incidents that match the specified criteria. Incident nodes are considered the 'root' of a delay network and are a logical starting point for browsing through the database. A grid of trains, delays, or locations could also be presented but a search of just that node type does not make sense because the importance of them is always in relation to other types of nodes - of what use is a search of just locations that begin with the letter C? If necessary, a relational database would be a more appropriate tool for this type of search on those nodes. Figure 5 is an example - incidents with the word "birds" in the description, i.e. incidents where a train struck a bird. They are ordered by date.

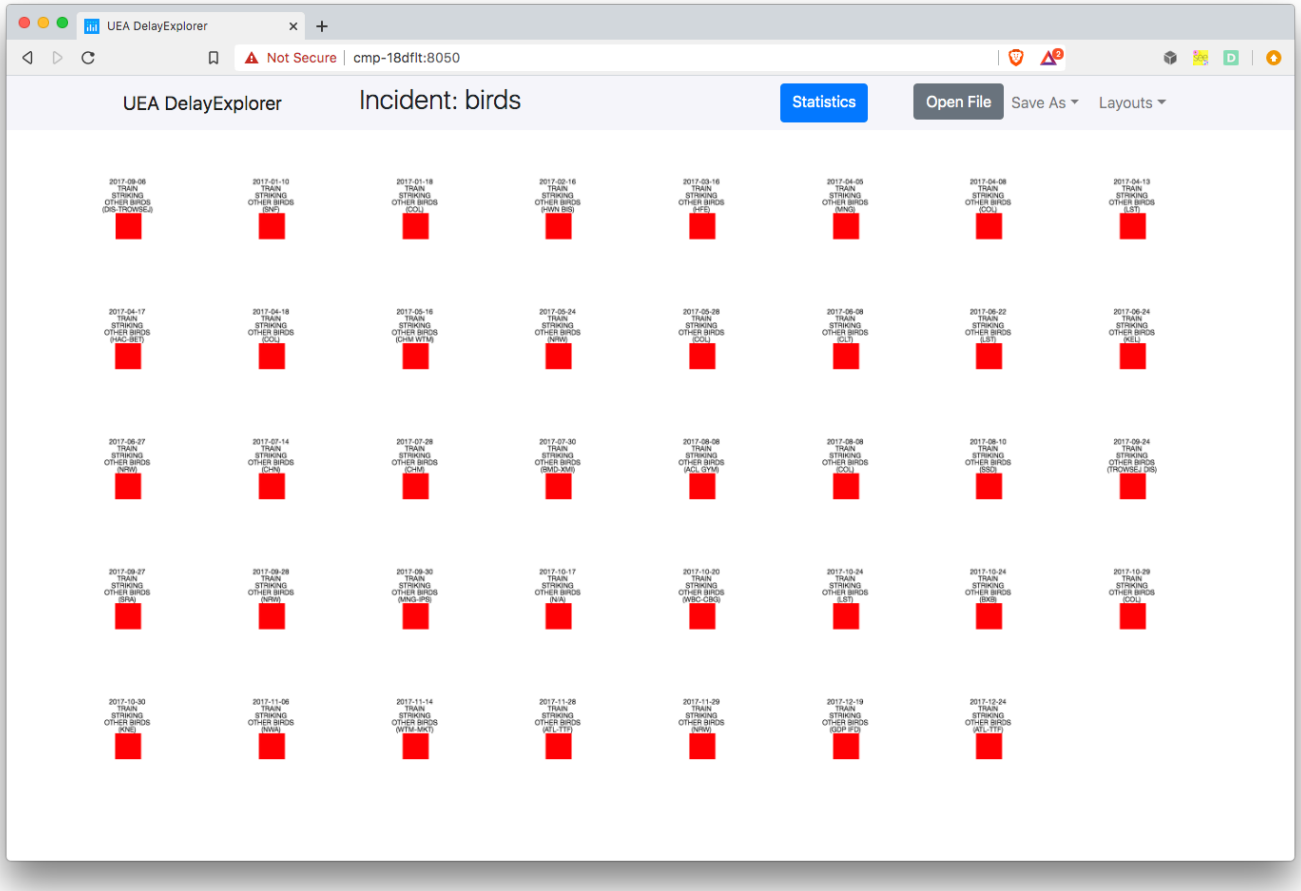

Figure 5: Incidents involving bird strikes

Logical relationships

The next set of layout algorithms treat the delay network as a directed acyclic graph (DAG). This type of algorithm emphasizes the logical relationships of the nodes - trains that cause delays are to the left of the delay and trains that are delayed are to the right. The cause and effect relationship is inherent within the directed edges. But as these are standard layout algorithms, the time attribute of when delays happened is ignored and nodes are positioned based entirely upon their rank and the level or depth within the graph.

Figure 6 is an example of the problem with the first PoC - the library used had issues and so complex graphs ended up very badly drawn. Figures 7 and 8 are examples of the new algorithms and library (all figures are using the same delay network). The difference between Figure 7 and 8 are due to the specifics of each algorithm and the default settings.

Which algorithm is better for certain types of delay networks is an open question because 'better' involves a qualitative judgement and there is no currently defined way to categorize delay networks. Further research is necessary to determine how the existing research on graph aesthetics could be leveraged to develop these categories.

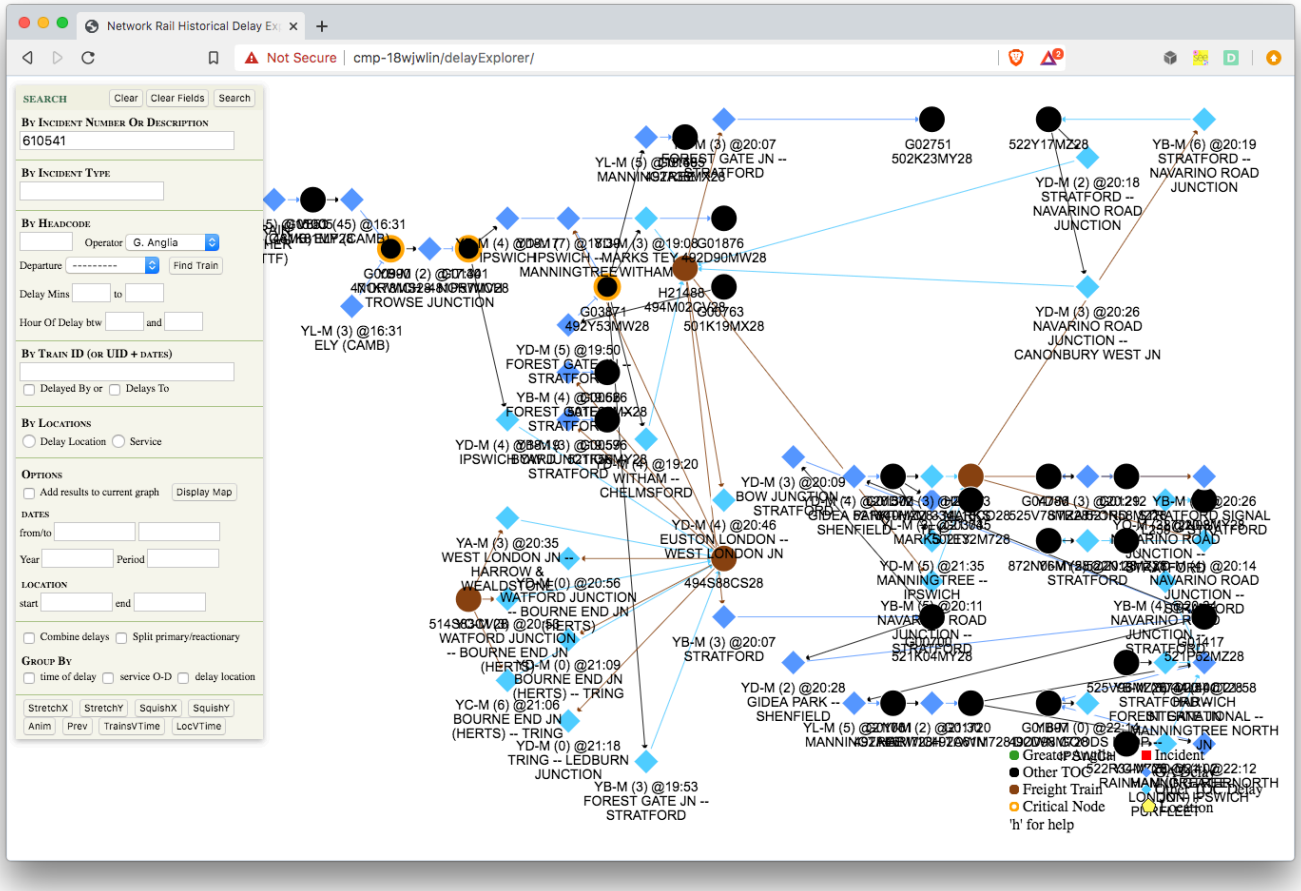

Figure 6: Incident 610541 in the first PoC

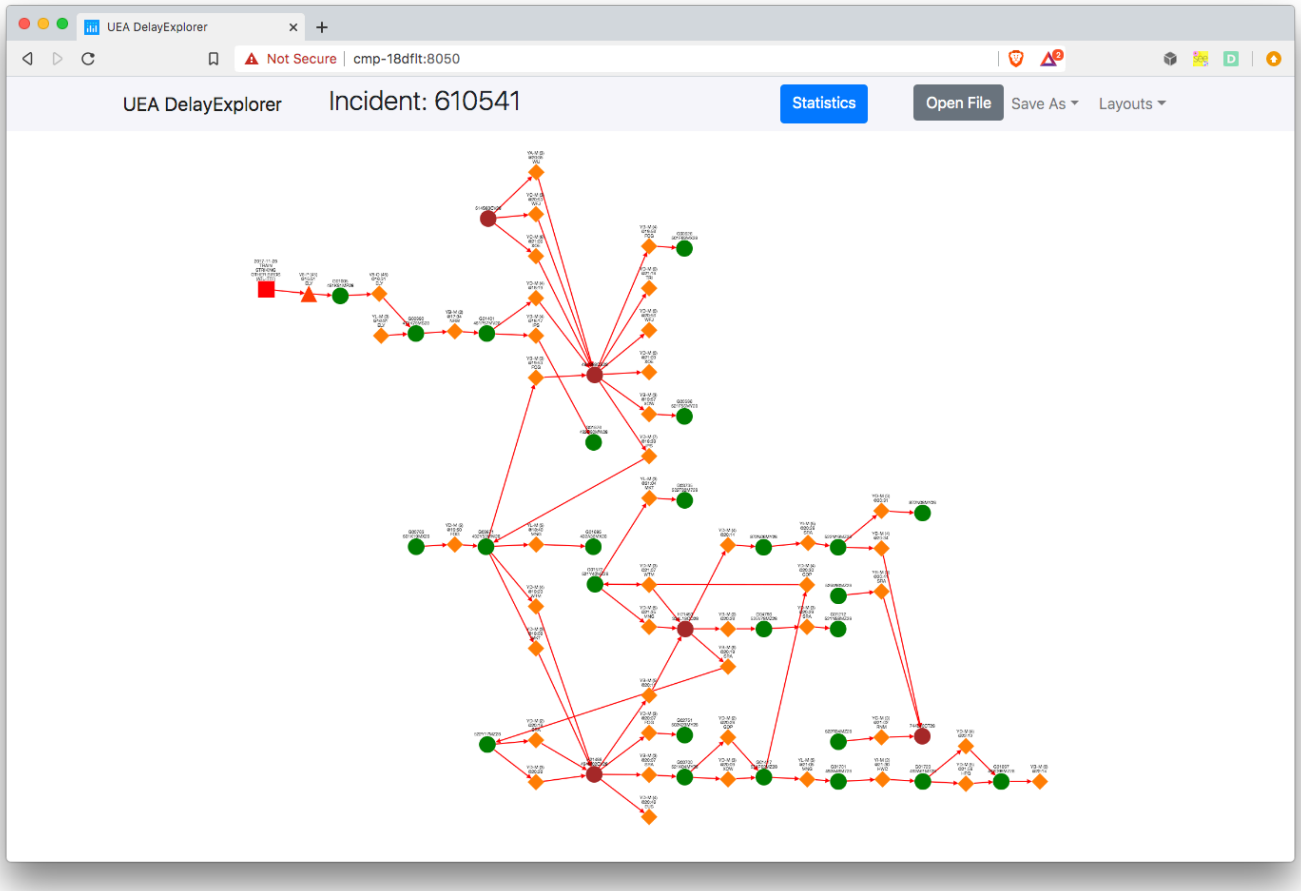

Figure 7: Incident 610541 (algorithm: klay)

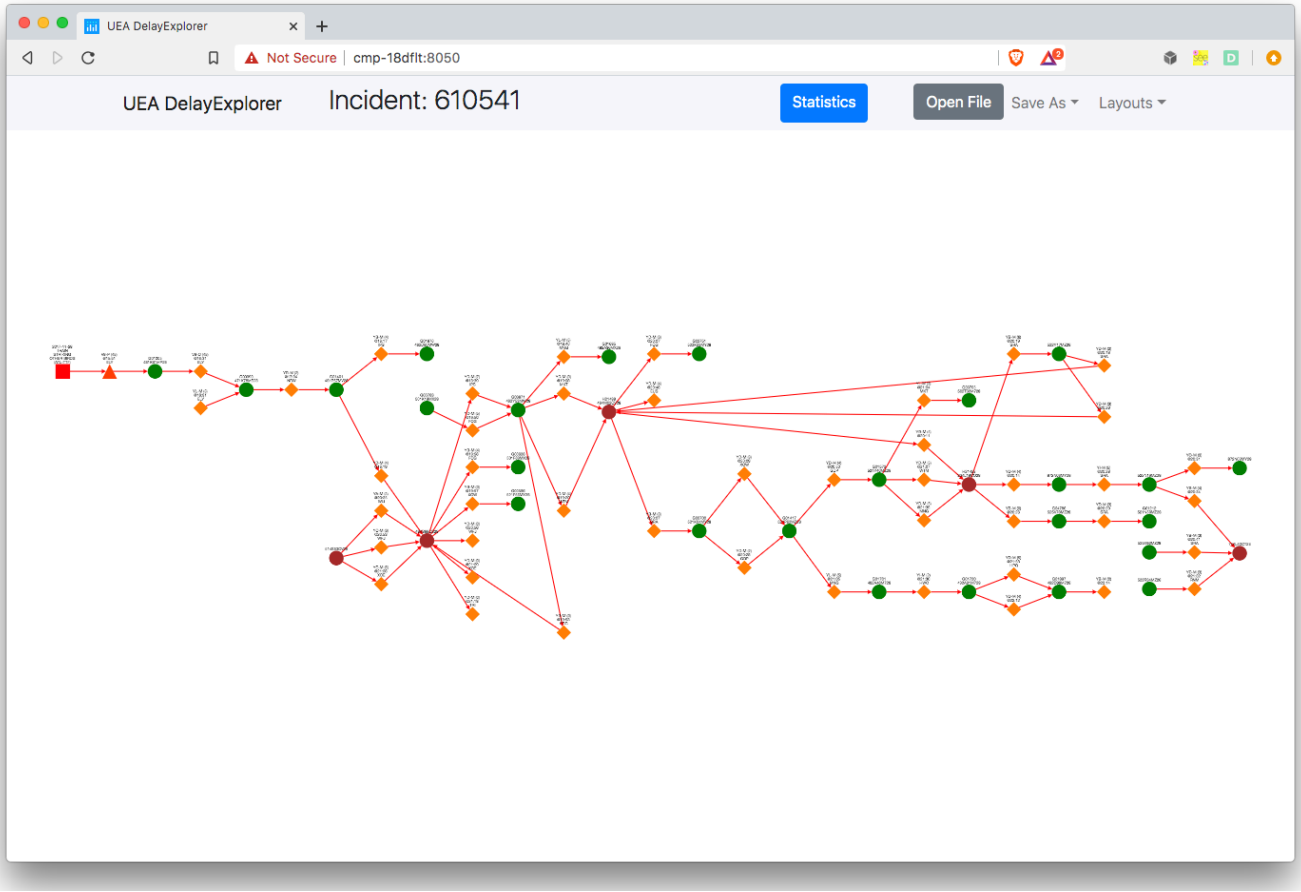

Figure 8: Incident 610541 (algorithm: dagre)

Node importance

Figures 9 and 10 are examples of this category of layout algorithm. They are prebuilt standard algorithms ("concentric" and "cola") so do not utilise any node attributes. Concentric arranges nodes in concentric circles based on the number of edges a node has while cola is an example of the force layout algorithms. At first, it was unclear of what use these two algorithms actually would be, but they helped refine the concept of "node importance". The importance of a node (more specifically, a train) can be defined based on its relationship to other trains - e.g., how many trains did this train delay? Or rather, if the delay network was projection of what was going to happen, "will delay". These two layouts help the user immediately grasp what trains are more "important" than the others - for the concentric layout, it is the ones in the center while for "cola" (or any force based layout), the hub nodes.

Also, a train can be defined as important when it is a critical node - the first PoC highlighted nodes that were calculated as being ones that, when removed, would bisect the graph. Bisecting the graph is equivalent to preventing all the subsequent delays the critical node is ultimately responsible for causing. So therefore, "critical" trains are ones that if they had been regulated properly, would have prevented subsequent delays from happened. This is an example of how abstract graph theory can have practical applications.

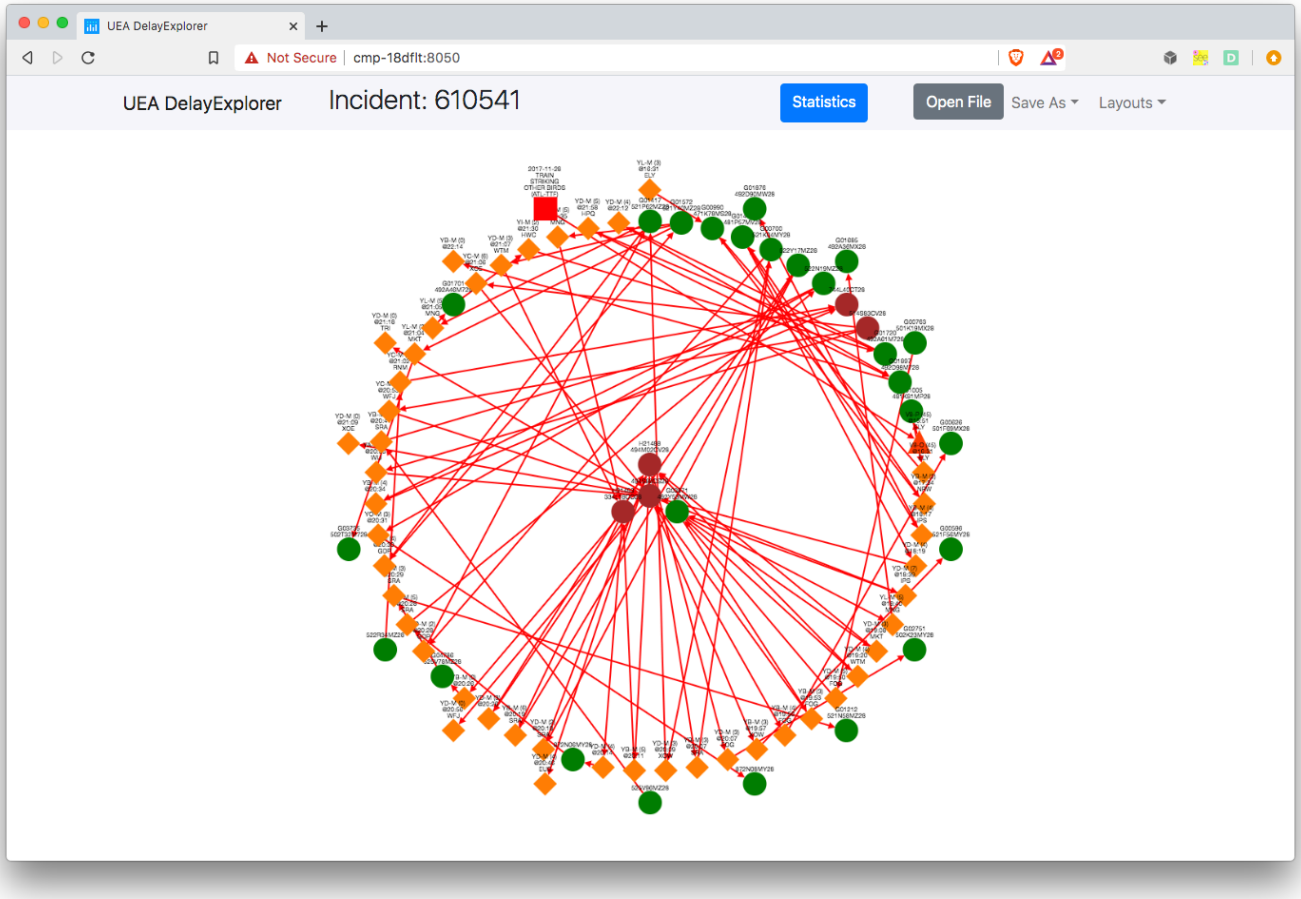

Figure 9: Important trains (1)

Future improvements or customization of these layouts will involve improving certain aspects of the picture. For example, the concentric layout could be improved by making the edges crossing the center of the circle more transparent to reduce the distraction and also positioning trains on the outer ring as close as possible to their delay. They are trains that were delayed but caused no delays. The trains in the center could also be spaced out more. Exactly how useful these layouts actually are will be determined by feedback from the rail industry.

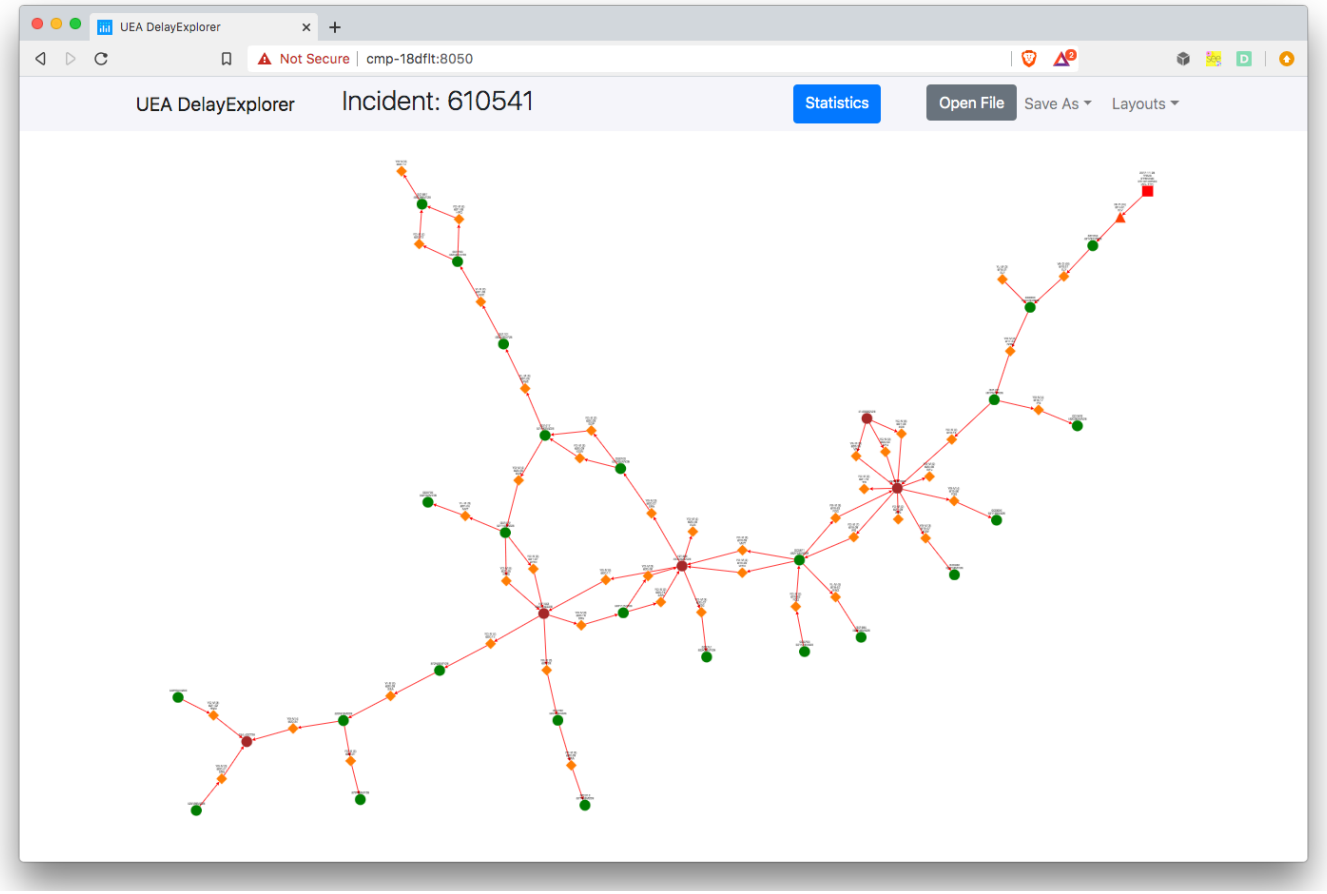

Figure 10: Important trains (2)

Another useful aspect of the force directed layouts is that loops in the delay network become very evident. Loops happen when freight trains (whose schedules are actually loops through a set of stations) cause a series of delays that end up impacting hours later the first train that started the sequence. It is currently not clear how much the rail industry is concerned with them, but understanding them better should be beneficial in preventing such situations. The DAG layouts are not designed to handle loops very well and the visualisation always ends up drawing an edge going backwards thus reducing the aesthetic quality. For these situations, there will be an option to "break" a loop by adding a virtual copy of the node that started the loop. The graph will then truly be a DAG and the two copies clearly marked as being copies.

Time order

This is the first custom layout designed specifically for DelayExplorer. It emphasizes the order of delay events by placing them on a timeline along the X axis. Delays can be positioned on an absolute basis or a relative basis. The absolute basis spaces out nodes by the amount of time between each pair so that the gap between them is proportional to the amount of time. The relative basis positions delays strictly by their order in time with a set amount of horizontal space between each. Train nodes are then positioned on the same X position as the delay that delayed them and the Y position is based upon the X value of the latest delay the train causes. In this manner, the "more important" trains that cause later delays and multiple delays are positioned higher on the Y axis. If a train only cause one subsequent delay, it is positioned at the lowest Y level right above the timeline. Trains can also be impacted by multiple delays and for those nodes, the X position is set at a median point between the minimum X of the first delay and the maximum X of the last delay. For more complex graphs where there is a surplus of edge crossings, logic will be implemented to detect this and train nodes duplicated to reduce the amount of visual clutter. Alternatively, trains can be placed below the timeline and lower on the Y axis. Figure 11 is an example of the 377188 delay network while Figure 12 is of the 610541 delay network that is more complex.

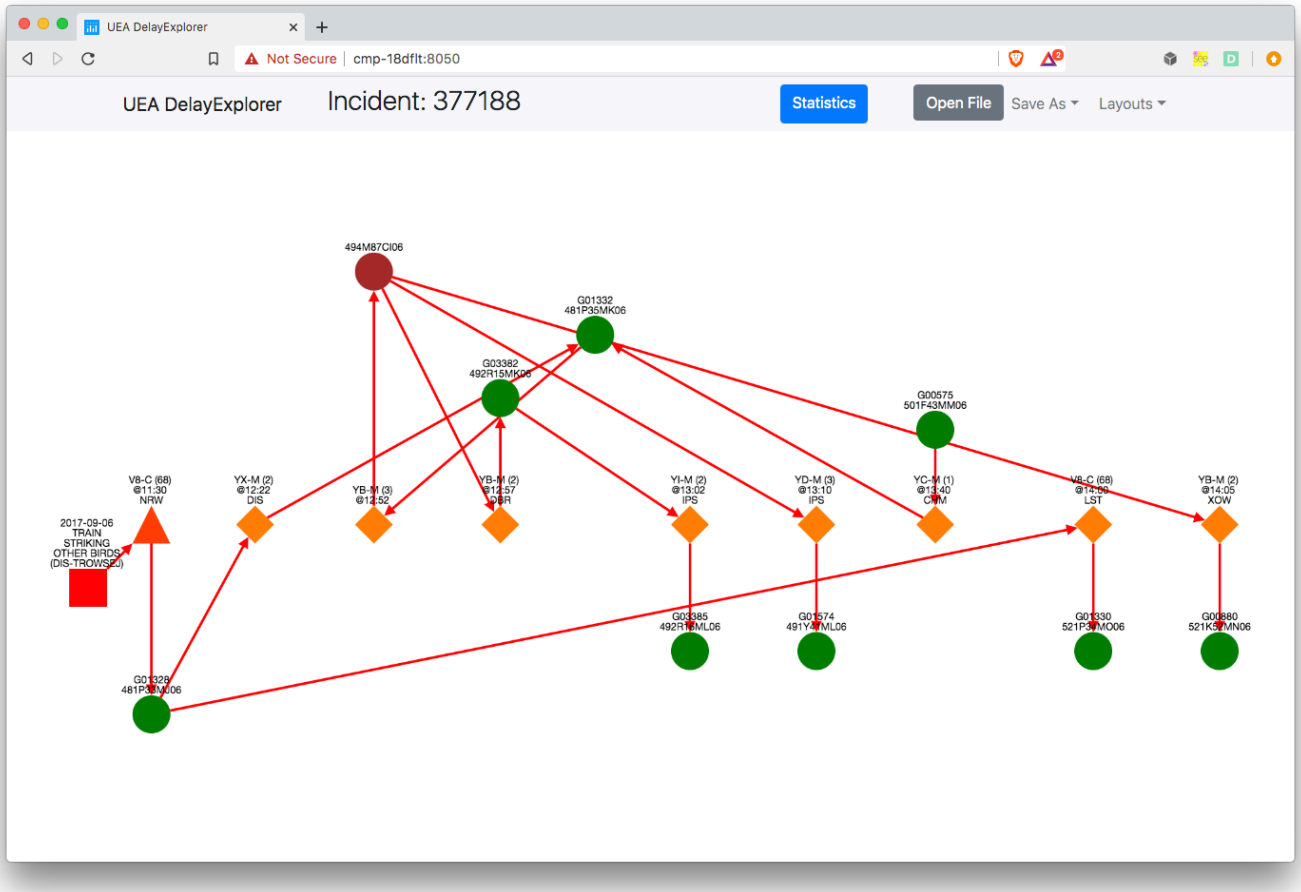

Figure 11: Time ordered example 1

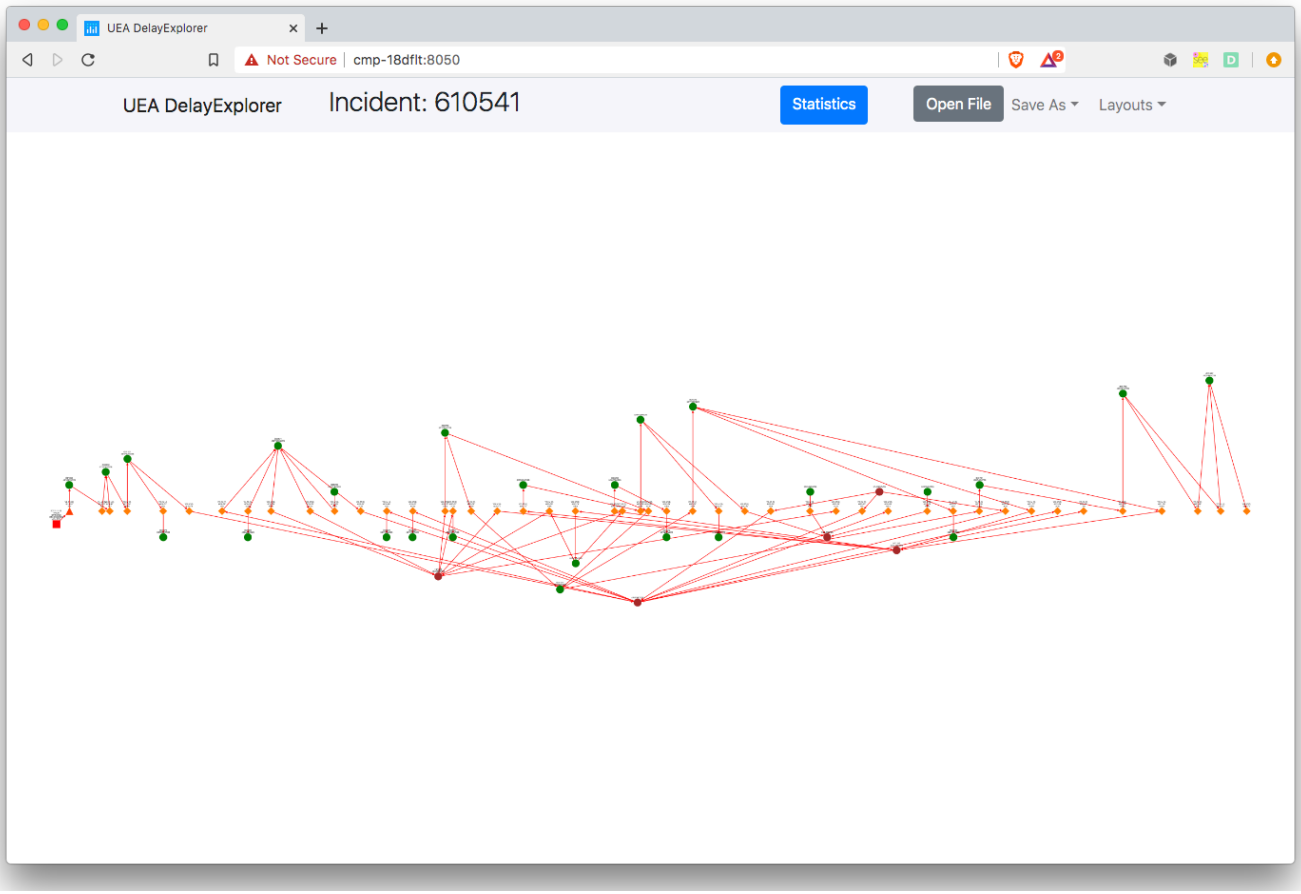

Figure 12: Time ordered example 2

Geographical Information

Figure 13 is an example (incident 377188) of the map view based developed for the first PoC. The control bar at the bottom animates the display by simulating the passage of time and placing markers representing the train delays as they occur. The black lines are the actual train tracks. At first glance, this seems impressive, but its usefulness is uncertain.

First, if the area over which delays occur is large, then to see all of them requires a zoom level that reduces the ability to distinguish closely placed markers from one another. At best, one gets a sense of how delays are clustered at certain spots, but the resolution is a significant factor - exactly how useful it is to see that one delay occurred 100 meters away from another one? Having multiple delay networks on the same screen would provide more certainty where common hotspots are.

The animation provides an understanding of how the delay network evolves over a geographic area, but again, of what use is understanding the ordering of events in relation to geography? With their domain knowledge, rail industry staff might find this display valuable, but labeling the markers (i.e. numbering them) may be more valuable than the fancy 'glitz' of animation because all of the markers and time information would be presented at once. Currently, once the animation is finished, the ordering of events is no longer obvious.

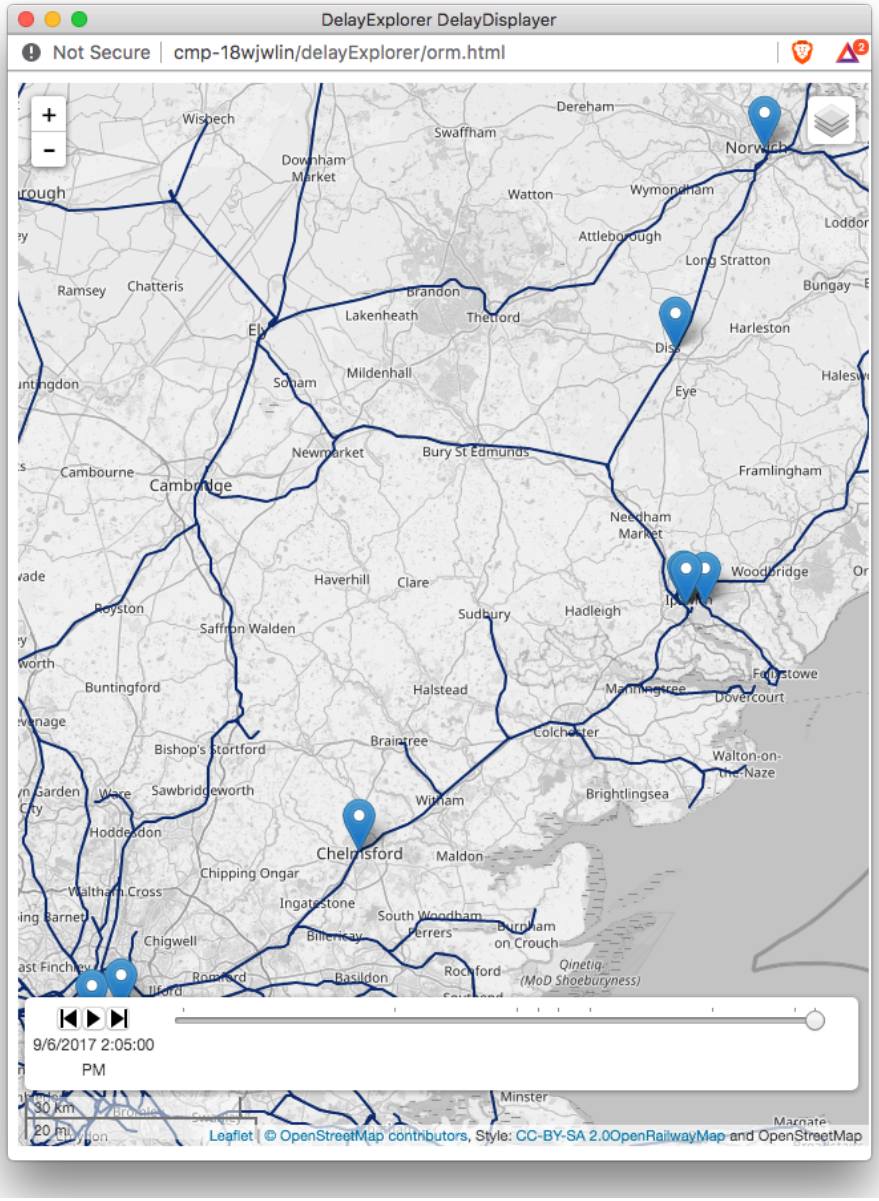

Figure 13: Map based layout

The detailed map is also extra glitz that somewhat detracts from the display; making it more transparent is one option to deemphasize it. To investigate these various concerns, geographical aware layouts were developed for the current PoC that eschewed the use of the use of publicly available GIS tile servers. Instead the intent in Figures 14, 15, and 16 is to place icons representing significant landmarks (e.g. towns and rail stations) at the right latitude and longitude. Delays are also placed according to their coordinates and are connected by their logical relationships (delay to train that caused another delay). This enables the user to easily trace the cause and effect relationships between nodes unlike that on the first map based display. Having more lines to represent relationships on the first display would result in visual overload; it is also not apparent how useful the actual train tracks are. It should be noted the placing of nodes to represent fixed points is not yet finished. Figure 14 is a layout that uses both the latitude and longitude to position delay nodes and future work will involve adding a more transparent map based background.

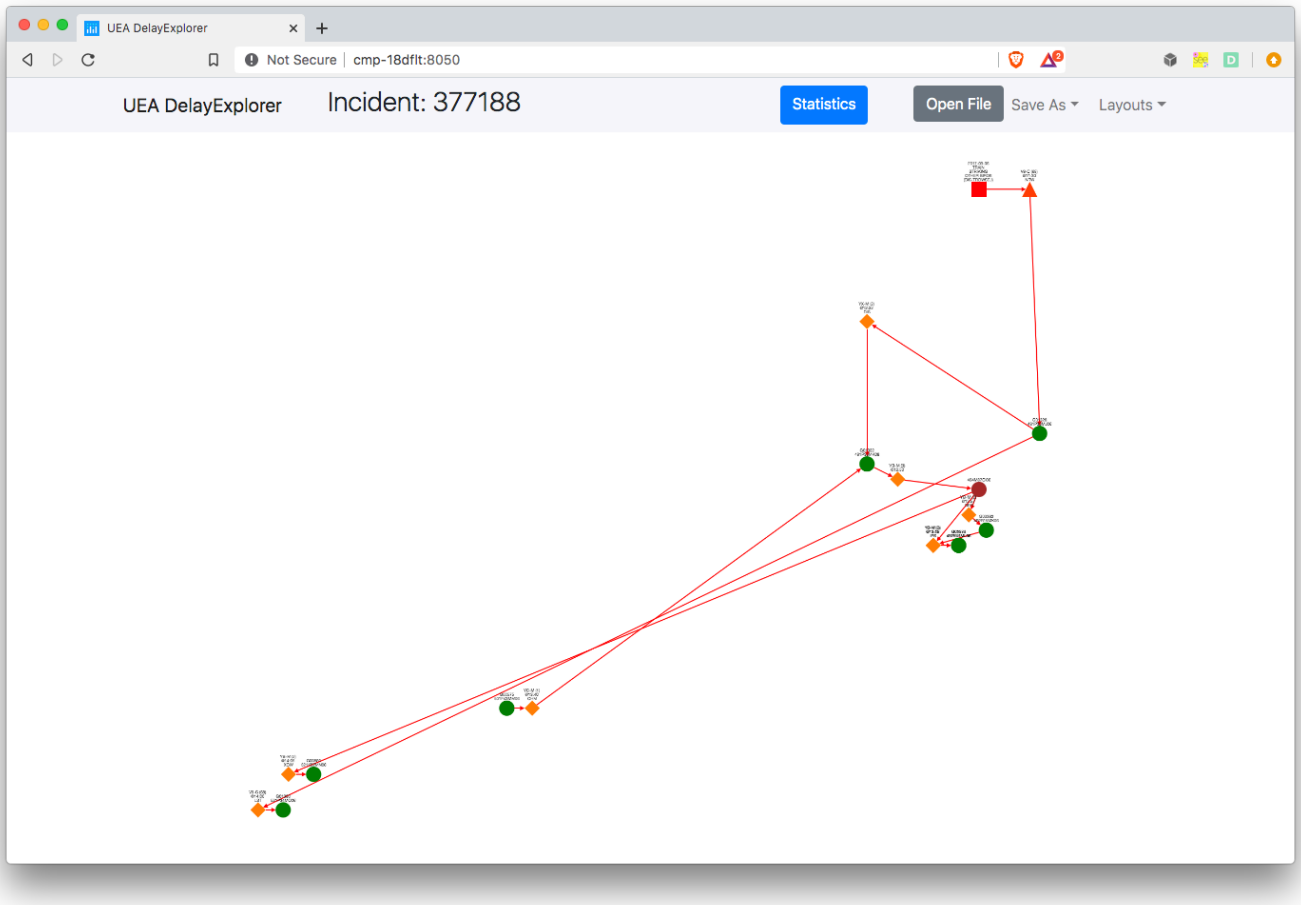

Figure 14: Geographic layout

Figures 15 and 16 are experimental variations upon the idea of using geographical coordinates. The axes in Figure 15 are latitude (the Y axis) and time (the X axis, from earlier to later). The resulting display shows how a delay network, over time, progresses north or south - the layout is named "Down To London". The direction of travel of a train service is an important consideration in the train industry as London is a central focus. Again, fixed locations need to be placed on the layout, but it is anticipated the utility of this layout will lie in revealing how delay networks that are currently in progress will spread - delays that affect the London area typically cause more reactionary delays that spread outward from London, so preventing these issues is paramount. Figure 16 is the mirror of Figure 15 - time is on the Y axis (earlier at the top) while longitude is on the X axis. It shows how a delay network spreads east to west or vice versa.

Both are unusual layouts that require some effort to learn how to read, but it is anticipated their utility will be apparent once the rail industry provides feedback and they are improved. Using time as an axis alleviates some of the problems associated with animation of the display. There is also no reason all of these layouts can be provided as they show the same data from different angles which could be useful.

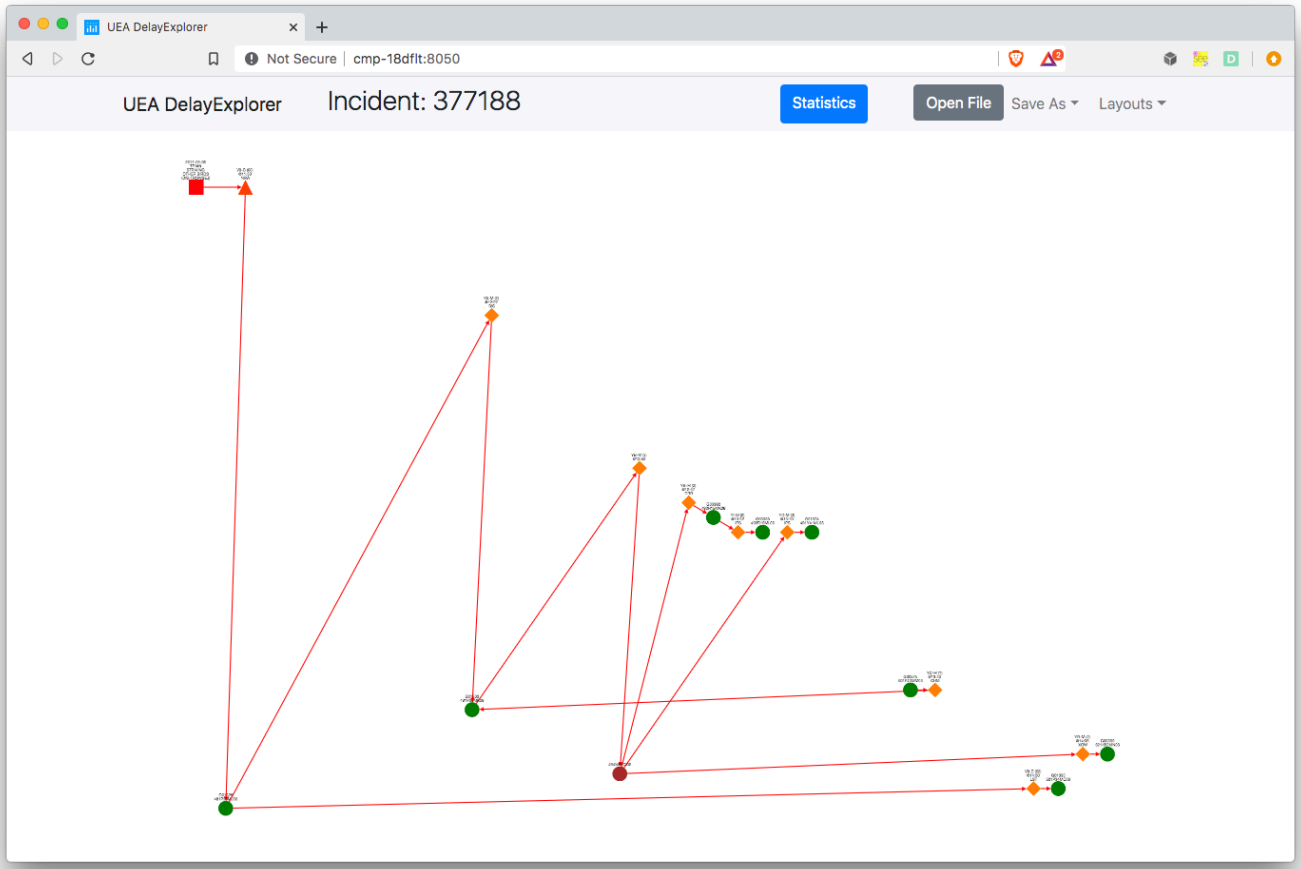

Figure 15: Down To London layout

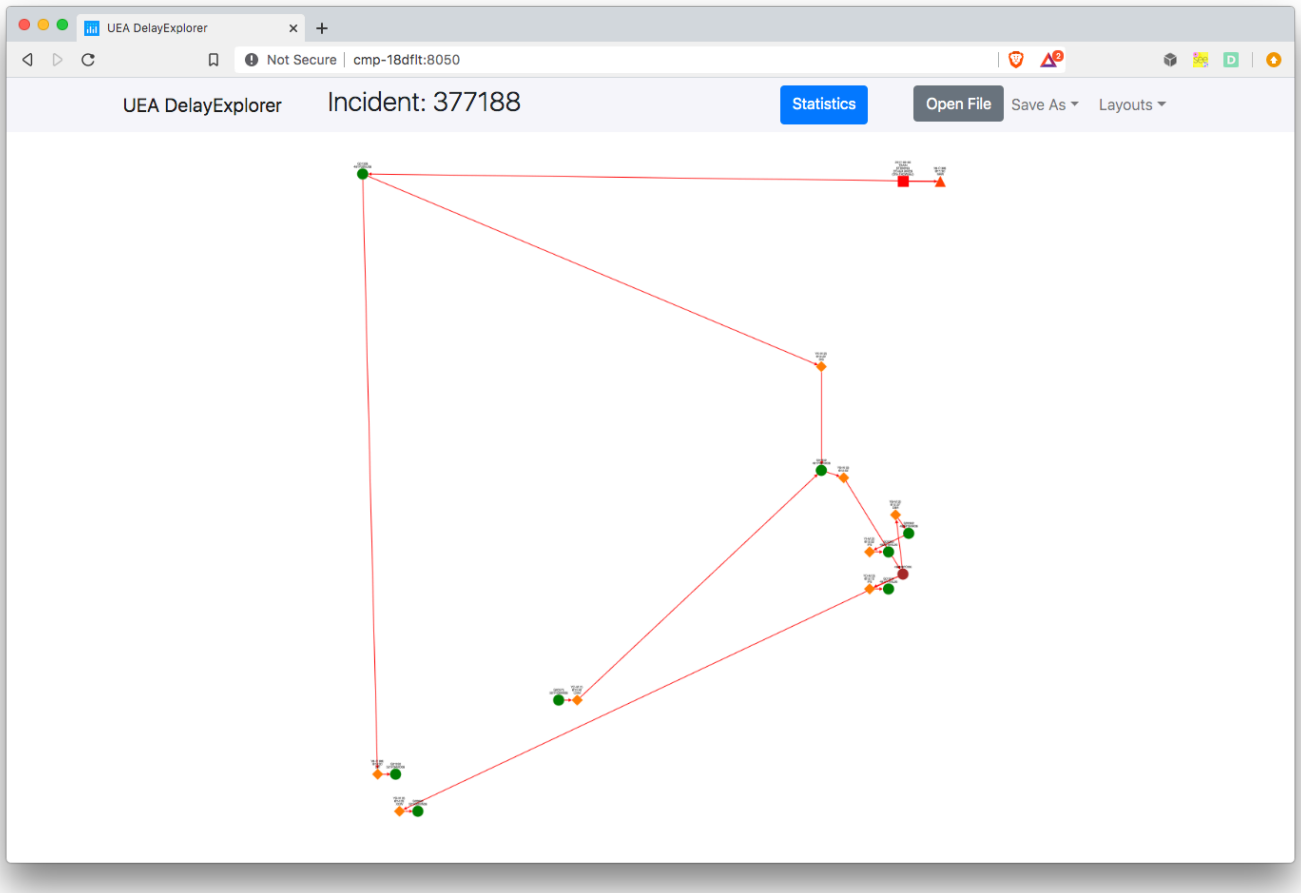

Figure 16: Across Britain layout

Summary

The use of graph based data is growing at an increasing rate in many industries (e.g., banking, computer security) due to the innate ability to reveal the relationships between different data points in a way table based data does not. It is for this reason graphs are ideal for understanding the workings of transportation systems such as the rail industry and how trains relate to one another as delays occur. But the DelayExplorer system will not be the only use of graphs; there is a large number of aspects of the rail industry that could be addressed using graphs as a conceptual basis and thus several potential avenues of research.

One notable one would be the use of graphs as a source of information for artificial intelligence applications to address problems such as predicting how trains might be delayed - this is a link prediction problem from a graph based perspective. Case based logic (i.e. expert systems) could also use the delay network graph data as a basis for prediction and machine learning research is using graph based data as a source of features. More collaboration with the rail industry will be necessary to determine exactly what is useful and how their problems can be addressed with this new paradigm.

The foundation of DelayExplorer is now well established and the essential design and technical problems understood. However, there is still quite a lot of work left to create a finished product given the long list of requirements and ideas in this paper. Key improvements will be implementing all of the search functions, display of multiple delay networks in a layered fashion, new visualisations to summarize data, and a touchscreen interface. At this point in time, none of it is anticipated to be very complicated or to involve definite unknowns; it is merely a matter of time and more code.

But despite the fact that DelayExplorer is still an incomplete work in progress, the response from the rail industry has been very positive. Grand Central (a TOC owned by Arriva) has even agreed to support UEA by becoming a collaborator in the Innovate UK SBRI First-of-a-kind 2020 competition; this grant will enable DelayExplorer to be developed further as well as tailored specifically for the needs of the rail industry.

Appendix A: User Focused Requirements Specification

As discussed in Section 2, there are four distinct categories of potential users of the DelayExplorer system. This section details the functional requirements specific to each category: academic researchers, business analysts, delay attribution teams, and train signallers. From this analysis, a development road map or project plan can be created based upon how things are prioritized. For the last three categories of users, the rail industry's input is essential for deciding exactly what is needed.

A useful way to view the relationships between different variants of DelayExplorer is to plot them on a graph (the other type of graph). The X axis is the type of data being analysed or viewed; it goes from historical data to "recent" data to real-time data. The Y axis is the complexity and / or amount of functionality offered, from low to high. Thus the academic researcher version of DelayExplorer would be in the upper left corner – high complexity and using historical data. Next, to the right and lower down would be the analyst – their needs are slightly less complex and have more of an operational / planning focus. To the right and lower down is the delay attribution team's DelayExplorer. They are only working with the most recent two weeks of data that hasn't been attributed and their functionality is a simplification of that provided for the analysts. Finally, in the lower right corner, is the controller's DelayExplorer. It is working only with real-time data and its focus is on assisting the controller in making decisions about how to prevent the spread of reactionary delays.

The benefit of viewing the different versions of the DelayExplorer system in this manner is that the development plan becomes obvious. Once the version for academic researchers is built, its functionality can be reduced or modified to support the needs of the other users. As for data, recent or real-time data is merely historical data that has existed for only a short time; there is little need to treat it differently. The major change needed to handle real time data would be a module that can read Network Rail's real time data feeds and send events to the DelayExplorer system as updates. The testing of that version would also involve streaming historical data from a database as well as the test procedures used for the other three versions.

Academic Researchers

The focus for this group of users is on understanding and analysing the rail network as an abstract system. General principles of how transportation systems operate can then be derived or validated. This leads to the need to treat the graphs within the data (the rail network, train paths, delay networks) as abstract graphs as well as representations of actual events.

<u>Functions (and a UI) for summarizing the rail network related characteristics of a delay network</u>
<u>Functions (and a UI) for summarizing the characteristics (based on graph theory) of a delay network</u>

These two requirements have already been partially implemented in the proof of concept: for the currently displayed delay network, there is a secondary screen/browser tab containing a table of statistics such as the number of trains, the number of delayed trains, the total amount of delay minutes, etc. as well as some standard graph theory statistics that characterise the graph.

<u>Functions for computing the "similarity" of one delay network to others</u>

What constitutes "similarity" needs to be fully defined, but one aspect determined to be useful is the "bushiness" of the delay network - it is a quantitative way of describing how (or whether) the delay network spreads, either geographically or through time. Another aspect is determining how the train services involved in multiple delay networks might be mapped or correlated to one another - Network Rail's Service Code and Service Group labels will be a key part of this process.

Functions for computing the “economic cost” (e.g. fines) of a delay network

This is an important characteristic of a delay network but needs to be defined. Network Rail's methods for allocating fines amongst TOCs are quite complex and perhaps not entirely appropriate or useful. More research is necessary to determine the correct process of calculating the economic cost.

Functions for computing the “operational cost” (e.g., time wasted, late passengers) for a delay network

Again, another important characteristic, but one that needs to be researched and fully defined.

Support for automation of functions operating on delay networks in bulk

Such functions would be not only calculating different measures (e.g. the economic cost) but also generating images and saving the data of individual delay networks to files, in bulk so that the entire database can be processed. The ability to generate pictures and files has already been developed.

Display of different tables and charts of associated data

The following list details the relational data associated with trains that will be useful in the analysis of a delay network:

A tabular display of the performance characteristics of a train (its planned versus actual schedule, etc)
A tabular display of the berth level TD data for a train on a particular route and day
A chart type of display of the berth level timing data and comparison to the nominal values
A chart type of display of the DARWIN (to-the-minute) performance of a train vs. the planned schedule
A way to display the data associated with the time dimension of one or more delay networks
(the relative time between events as well as the absolute times involved)

A regex type facility for searching through the graph database

This function would enable interesting types of search methods that are not focused strictly on one type of node. The inspiration for this comes from TREGEX: https://nlp.stanford.edu/software/tregex.shtml and integration of that library (or extending it) looks to be straightforward.

Incorporation of more detailed weather data localized to specific areas of the rail tracks

WeatherQuest is a UEA spinoff that can supply weather data that is more fine grained than the Met Office. Using this data and correlating it with delays would be quite beneficial, especially considering Network Rail's use of weather data is not as sophisticated. WeatherQuest can supply CSV files so importation is a straightforward process while the related UI would involve different types of charts and line graphs which can be handled by the subsystem for displaying other such data.

Incorporation of Nexala type data for individual trains

Nexala is the name of the system onboard trains that records events such as doors opening or when the train slowed down. Correlating this data with other delay related data would certainly be useful, but the extent of the work required (acquiring the data, analysing it, importing, the UI, etc) is currently unclear.

An ability to animate the trains involved in a delay network on a map of the rail network

The animation of data (as it changes) is a potentially useful feature when visualising data. This basic idea leads to a larger one of being able to replay historical events (e.g. the behaviour of trains) in the process of analysis and understanding. With a graph of the rail network (at the berth level) and the TD data, a train's behaviour can be simulated with enough fidelity (both forwards and backwards) to gain a more precise understanding of why delays happened. Such a function would also allow for the creation of "what if" scenarios, e.g. what if a train had been 5 seconds faster – what would the likely effects have been?

This idea can be summarized in three words – a Virtual Train Set. This is not a novel idea as train simulation software already exists for train enthusiasts but using data from the real world is perhaps novel, unless Network Rail already has such a system given how it such an obvious idea. Train simulation software simulates the operation of the signalling system, switches, etc which is too much detail and detracts from the analysis of delays. The utility of DelayExplorer would then be in the combination of the analytical functions and support for delay network graphs to simulate historical events at a higher level.

Business / Operational Analysts

Currently, it is uncertain exactly what custom functions a Network Rail analyst might need for planning or analytical purposes, but most of the generic functionality and the functionality described in the Academic Research section should be useful. The more abstract functionality related to graph theory could be removed unless research reveals the theory has a relationship with the practical concerns of operational analysts. A good portion of DelayExplorer's functionality will probably already exist within the current tools Network Rail has; more discussion of their needs is necessary to determine how much overlap exists. A system that can incorporate WeatherQuest's more detailed weather data will surely be welcomed as well as perhaps the ability to replay what occurred and create "what if" scenarios.

Delay Attribution Team

The focus of Network Rail's (and TOCs) Delay Attribution Teams is to process the most recent weeks' worth of unattributed delays so that agreement can be reached on which organization is responsible. This implies the need for an import data function or perhaps a connection made to the Network Rail PSS system which is the basis for the HDA data. The exact design of this functionality depends on the specifics of the rail industry's business processes and will require their input.

There are two types of delays: initial and subsequent ones. Attribution of initial delays typically involve analysing other data that DelayExplorer does not handle, e.g. train driver reports. But DelayExplorer's ability to replay what occurred should be useful as well as the ability to easily comprehend the more low level data such as the TD data feed through line charts and other such graphs. The more academic type functions, however (e.g. related to graph theory), should be removed in order to simplify the system and the UI.

As for subsequent knock-on delays, DelayExplorer would be an ideal tool to assist in analysing train delays and determining their cause. It would serve as an easy to use tool for correlating different pieces of data, e.g. the performance / behaviour of two trains and a replay events feature would surely assist in understanding cause and effect of delay instances. Also, the UI of DelayExplorer is ideal for quickly understanding how an initial delay leads to subsequent ones which lead to others. As mentioned, the feedback from industry representatives during the RSSB funded study indicated they had not seen anything so comprehensive before. This leads to the question of what visualisation tools does Network Rail and the industry actually use and how can they be improved.

Finally, a significant DelayExplorer function that could be added for the Delay Attribution version of the software is one that can automatically process the incoming data to be analysed and compare it to historical delay networks (and the lower level associated data). This then allows the system to make preliminary attributions for some or all of the reactionary delays (as well as provide a visualisation of the data). The user's task then mostly would be to confirm or correct the output of the rules based "expert system", thus greatly reducing the human workload.

Signallers / Controllers

Signallers are responsible for regulating trains, i.e. making decisions about when and where to direct what trains along what paths. As the prototype of DelayExplorer has revealed, the total number of delays within a delay network could be reduced by making the right decision early enough to prevent a chain of subsequent delays. Modelling what is going on in the rail network in real time as closely as possible thus becomes essential, so that possible or likely future events can be anticipated.

To fulfil this need, this version of DelayExplorer would need a module that is listening to Network Rail's data feeds and updating the model of the network within the DelayExplorer system. The foundation for the other versions of the software, e.g. the graph of the rail network, can be reused. The additional functionality needed would be the ability to anticipate the likely upcoming reactionary delays when two train paths cross at a junction – a graph database of the rail network and train paths along it is essential for this task. DelayExplorer could then create possible scenarios (and the consequences of available actions) to present to the controller to aid in their decision making.

As for the UI, it needs to be as simple as possible given the constraints on signallers and how they work. A point and click type of UI, such as that for the other versions of the software, is too complex and unnecessary. Instead, there are two options. The first is using the previously mentioned tabular view of a delay network which could serve as a quickly comprehensible description, once controllers are taught how to read them. A typical Human Computer Interface type study would be needed to evaluate how useful this idea actually is and what benefits a tabular view provides to signallers.

The second option is developing a touchscreen interface and a library of appropriate gestures. Browsers now support touchscreen interfaces (i.e. mobile phones) so the libraries needed are already available. Supporting multiple monitors and an ability to quickly change between upcoming scenarios would also be essential; the concept of 'layers' of delay networks will serve to support this function. For example, the current situation is the top layer while lower layers are different possible futures. These displays would be differentiated using multiple drawing and line styles, e.g. transparency and dashed lines. However, more research, prototyping, and user evaluation will be essential to further refine these ideas.

Appendix B: Analysis of the HDA Data

Appendix C: The HDA Fields

Field Name	Description
Financial Year	The financial year that starts every April 1st. The lower year of the field is stored in the graph database as the value (e.g. 2014 for 2014-15)
Financial Period	The specific 4 week period (of 13) the delay happened in. Periods are arranged such that the number of each day never repeats.
Date	The date the HDA record was created
TrainID	The ID of the affected train
Origin Stanox	The numerical code of where the train originated
Origin GBTT	The public starting date/time of the train. This is null for freight trains.
Origin WTT	The internal to Network Rail working time
Destination Stanox	The destination stanox code
Destination GBTT	The date/time the train is scheduled to arrive at its final destination
Destination WTT	The working time for arrival
Train Service Code	The code for the set of services the train's schedule belongs in
Service Group	A group of related service codes
Operator	A code for the company operating the train
English Day Type	Weekday, Saturday, Sunday, Bank holiday, or Christmas
Timetable Flag	Flag that indicates if the train schedule was a short term replacement of the initial one
Incident Number	The ID of the incident
Incident Creation Date	Incident numbers roll over every year, so the date makes up part of the unique key for an incident
Incident Start Date	The date the incident was entered into Network Rail's systems (not the date of the incident)
Incident End Date	The date the incident was stopped being tracked
Section Code	A code for the area of the country the incident took place
Location Manager	The organization responsible for the location
Responsible Manager	The party who is responsible for the delay
Incident Reason	A code for the incident from the Delay Attribution Guide
Attribution Status	Whether the delay has been accepted or is still being disputed by the parties involved
Incident Equipment	Free form text about the incident
Incident Description	An abbreviated description about the incident
Reactionary Reason Code	If not blank, then the code categorizes what type of reactionary delay it was
Incident Responsible Train	ID of the train ID responsible for the inciting incident (typically just the headcode)
Performance Event Code	Indicates whether the train has been cancelled or delayed and the specific category
Start Stanox	Stanox where the delay happened (not always the incident)

End Stanox	The exact location where the delay happened is not available, so the section of track is provided
Event Datetime	The date/time of the delay
PFPI Minutes	Process For Performance Improvement statistic about the length of the delay
Non-PFPI Minutes	Delay minutes not categorized as PFPI
Responsible Train ID	The full 10 digit ID of the responsible train
Reactionary Train ID	If a reactionary delay, the ID of the train causing the delay to the affected train