MSc Data Analytics Dissertation: Visualisation of Traffic Flow Data for Glasgow City Council

Client Report

**Elijah Reid**

**202093659**

**2021**

Signed statement

Except where explicitly stated, all the work in this dissertation – including any appendices – is my own and was carried out by me during my MSc course. It has not been submitted for assessment in any other context.

Signed: ……………………………………………………………………….

# Executive Summary

This report includes details about the steps undertaken in the creation of a data pipeline and traffic visualisation app on behalf of the Glasgow City Council. This app would allow Traffcom to better understand and control traffic patterns, whilst also acting as a proof of concept for future data visualisation projects the Council may wish to carry out in the future.

The goal of this project was to find a way to establish the traffic direction for the network of traffic sensors across the city and then subsequently create an informative visualisation making use of pre-existing software tools available to the Council.

Some of the key facts that the Council wished to be included in the design of the visualisation are the following:

* The visualisation must make use of software tools that are either already available to GCC (such as ArcGIS spatial mapping applications) or those which can be freely licenced by the Council.
* The visualisations should be configurable so that different groups within in the council are able to view only the information that is relevant to them.
* The visualisation should be based off historical traffic flow data rather than real time data.
* Following this should be possible to select different time periods when viewing the traffic flow visualisation
* It should also be possible to view and select different routes and areas

The central challenge comes from the fact that the geo-position and the direction of travel for the SCOOT detectors are not included in the master dataset of SCOOT links and will therefore have to be extracted or interpolated from other sources.

To tackle the challenges, research was undertaken into the usage and benefits of using big data in local government at large, as well as more specifically in traffic management. More specific research was undertaken into the best methods for extracting data from text files, spreadsheet files and the two main formats used in the Glasgow City Council traffic API: JSON and XML.

Research was also undertaken into the best programming languages and software packages to use in order to create a generic and reusable data pipeline, that could be scheduled to run at regular intervals without the need for direct intervention from the end user. Time was also spent in understanding the various methods by which geospatial data is recorded by the Council and in other sources, and following on from this, the best way to prepare and display this data to demonstrate the data’s utility to decision makers with the Council. A link to the source code of the final app can be found here: <https://github.com/Wizzzzzzard/Glasgow-City-Council-Project>

The ArcGIS app allowed Traffcom to visualise the direction and volume of traffic flow on a map of the Glasgow City Council area. Over 1000 junctions and their directions were successfully displayed, and the app allowed the flow of traffic to be easily followed from junction to junction. While only two time periods were included in the current version of the app, this can be easily expanded to as large or as small a range as needed using the pre-existing data pipeline and app as a jump off point. Other departments of the council were also impressed by the potential utility of the app, and future plans for additional app functionality have already been suggested, which will ultimately lead to the Data Team’s goal of improving data access and transparency within Glasgow.

# 2.0 Acknowledgements

I would like to thank the Glasgow City Council for giving me the opportunity to work with them on this ground-breaking project, as well as providing me with access to various data sources and for their guidance and assistance over the course of the project. I’d like to particularly thank Stephen Sprott, Brian Davidson and Liz Irvine for supporting me through the entire project and providing a few moments of levity when progress was slow. Sparky the cat in particular deserves an honourable mention for ensuring time spent waiting for programs to load wasn’t wasted. I would also like to thank my advisor, Kerem Akartunali, for his supervision and advice throughout this project.

# Introduction/Problem Statement

This project was carried out at the behest of the Glasgow City Council Strategic Innovation and Technology (SIT) data team and Traffcom. The client has several separate and unconnected data sources: one a large master text file of each SCOOT junction in Glasgow with each of it’s connecting links along with other relevant and irrelevant data. There were also two APIs which were connected to a larger data warehouse, one which output a list of SCOOT links along with their geographical location and a second API which outputs the traffic concentration at each SCOOT link that is updated on a 15-minute basis.

The client wanted to find a way to use these disparate sources of information to create an application that would allow them to see the direction traffic was flowing in at each junction, as well as monitoring the concentration of traffic at each point. The client wanted the script for extracting the junction links from the master text file to be generic and reusable as new SCOOT links are added to it on a semi-regular basis and are even sometimes removed. The client then wanted a script that could match each link-to-link connection to its corresponding coordinates so that the traffic flow between them could be displayed on a map. Finally, a script to update the traffic flow at each junction every 15 minutes, that would also pull in the historical flow data was desired as this would allow for a more informative system than the one currently in place. At present each junction is just a single point on a static map of Glasgow with a simple colour coded number to denote the flow of traffic, which makes it difficult to update the map and to judge at a glance the direction in which traffic is moving.

In terms of the overall data flow, the client wants the data from the master text file and APIs to be extracted and transformed into a useable state, before being passed onto a GIS mapping software- preferably one which the council has already licensed, such as ArcGIS, or an opensource software.

The structure of the dissertation is outlined below. It will begin with an exploration of the literature surrounding big data in local government and traffic management, as well as some literature behind the main problems to be solved in the project. This went on to inform the direction of the methodology used in constructing this project, which will consist of the data extraction, prototyping the processing and transformation of the data, followed by an exploration of the methods used to display the data in a useful form and finishing with a conclusion explaining the results of the project and recommendations for the client to carry out in the future.

# Literature Review

D

D

D

D

D

D

# Methodology

## 5.1 Introduction

Initially the data used in this project came from two major sources:

1. The Master text file used by the Traffcom that contains the network data for the entire GCC SCOOT network.
2. The Glasgow Open Data Traffic API – this is an API that allows a user to query the central GCC database and retrieve information on each SCOOT links geospatial position and the traffic flowing through it at 15-minute intervals.

As the project progressed several other data sources were also included:

1. An internally maintained Traffcom databases with the coordinates of each SCOOT node in Northings and Eastings.
2. A further series of Traffcom databases including corrections to location data present in both the API and previous Traffcom dataset.
3. An ArcGIS shapefile containing point data with the geographic location of each SCOOT detector.
4. The raw CosmosDB behind the aforementioned API which could be directly queried using SQL to retrieve the last set time period of traffic flow data.

## 5.2 Methodology

### 5.2.1 Overview of the Master Text File

To begin with the initial goal was to find a way to convert the raw text file from a document laid out like this:

Text

Description automatically generated with medium confidence

Fig 1. Initial Structure of Network Data

Into a more structured dataset of road connections more like this:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| From\_Node | To\_Node | To\_Link | From | To | hasEntryLink |
| DD1271 | DD1251 | C | DD1271 | DD1251\_C | FALSE |
| DD1351 | DD1251 | D | DD1351 | DD1251\_D | FALSE |
|  | DD1251 | E |  | DD1251\_E | TRUE |
| GA0151 | GA0201 | W | GA0151 | GA0201\_W | FALSE |
| GA0251 | GA0201 | T | GA0251 | GA0201\_T | FALSE |
| GA0451 | GA0201 | S | GA0451 | GA0201\_S | FALSE |
| GA5201 | GA5151 | T | GA5201 | GA5151\_T | FALSE |
| GA5551 | GA5151 | R | GA5551 | GA5151\_R | FALSE |
| GA510A | GA5151 | V | GA510A | GA5151\_V | FALSE |
| GA5171 | GA5151 | P | GA5171 | GA5151\_P | FALSE |

Fig 2. Proposed Final Layout of Network Data

While the initial data structure is laid out in an intuitive manner for a human to read and understand, it is not well structured for a machine to interface with. It also contains a lot of extraneous information that is not needed for this particular project, including many lines of extra descriptive data as well as varying amounts of whitespace between each section of data.

### 5.2.2 Bash Script with Regex

To extract only the relevant data, a script needed to be created that would be able to detect the general pattern of the relevant SCOOT information (i.e., the name of the SCOOT node as well as the upstream and downstream nodes and links it connected to) without picking up any additional data. After carrying out some research into several methods used to extract text data from files, it became clear that a program based on using regular expressions, or regex, would be the way forward.

By making use of the Regex Quick Syntax Reference (Zsolt Nagy, 2018) the below regex pattern was constructed to match the necessary data:

'/SCN/,/^\s\*$/p'

The pattern works by first finding the letters ‘SCN’ which precede the junction’s name at the start of each section of data and then it captures all lines after the SCN until it reaches a line that is entirely whitespace, as this denotes the break between the junction connection section of the text file and further data.

The regex pattern by itself is only capable of matching the first occurrence of the pattern it represents and so to capture the data for each individual junction a more powerful text editing software would be needed. As I had some prior experience in working with the UNIX command line, a software program that contains several text-streaming and string manipulation programs, I decided to proceed using one of the pre-existing features. This also had the additional benefit of allowing a script that would require several lines of code in other programming languages to be condensed into a single line that could be run directly from the Windows Command Prompt, as well as massively decreasing the runtime of the script due UNIX interacting directly with the C programming language.

Ultimately the package chosen was sed (short for stream editor) as it includes an option to capture all patterns that match within a text file and to continue matching until no further patterns can be found. The final shell script, ‘ext-jun-links.sh’ contains the following code:

sed -n -e '/SCN/,/^\s\*$/p' $@ > Junction-Links.txt

The script executes like so:

1. sed calls the sed function from the UNIX shell
2. -n tells the sed program to continue looping until a certain criterion is satisfied
3. -e tells the program to match as many regex patterns in the text as possible
4. $@ allows the user to pass the name of the file to extract junction data from. In the case of this project this would be substituted for NETL1405.txt
5. > Junction-Links.txt – this passes the results of the previous script into a new text file which contains only the relevant data.

A picture containing text, screenshot, receipt

Description automatically generated

Fig 3. Text file after executing Bash Script

### 5.2.3 Cleaning and Splitting Text File with Python

As can be seen above, the bash script is able to remove the majority of the unnecessary data from the text file. However, a number of other steps had yet to be carried out before it could be directly fed into a program to convert it into a table like the one seen in figure 2.

Text

Description automatically generated

Fig 4. RemoveLines function python code

Firstly, a function was defined to remove any unneeded lines from the file by removing lines that contained specific strings. In this case the lines containing the strings Modified, Type and Region contained no other data needed to construct the table of node connections and so they were eliminated by running the function above.

As only the first part of the line identifying the junction was needed, the rest of the line that explained where the junction was located to the human reader could be discarded. A function was written to split the line on a specific delimiter, in this case ‘At’, and then keep the part of the string before that delimiter. If carried out on the first line present in figure 3. the function would return only the ‘SCN DD1251’ part of the string and drop the rest.

Text

Description automatically generated

Fig 5. TrimLine function python code

A final function was then written to finish cleaning the text file, which split each individual junction’s dataset into its own separate file to allow for a more generic program to extract the main junction title from the top of each file and then match it to the other junctions in the file.

Text

Description automatically generated

Fig 6. Split\_junctions function

An overall function to pre-process the text file was created which packages these 4 subroutines into one function was created to allow for the text file processing to be more quickly run whenever the junction file is updated.

Text

Description automatically generated with low confidence

Fig 7. Example of Individual Junction file

### 5.2.4 Extracting Location Data from Traffic API

In comparison to the code needed to extract the necessary data from the text file, the code needed to pull in the link, location and flow data from the API was a much simpler task. As the GCC had already set up an API to interact with the central traffic database it was simply a matter of sending the correct request link to pull in the needed data.

Text

Description automatically generated

Fig 8. API data pull function

The function in fig x works by requesting the data be sent in a csv format and then iterating through the predefined number of pages (currently the dataset contains 20 pages so by setting the page\_num variable to 30 it allows for a substantial amount of location data to be added before the program needs updating.) It retrieves the link name- a node followed by the relevant link suffix- as well as the geographical position of each link in WGS84 Latitude and Longitude.

Text

Description automatically generated

Fig 9. API flow table

### 5.2.5 Converting the text file into a csv table

While the text files have been pared down to only the data needed to create a table of link connections it is still not quite ready to be read directly by a computer. One last cleaner function is used to remove the leading SCN from the junction title and then strip out any extra whitespace to ensure all the strings extracted from the table contain only characters.

Text

Description automatically generated

Fig 10. Final Cleaning function

Now that each of the text files have been completely cleaned, the junction and link information can be extracted using two different regex patterns. These match the layout of both normal junctions (such as GG4812\_R) and links without a preceding junction (such as simply T). After matching one of the two patterns, it is then appended to either a list of incoming connections or outgoing connections. After this is carried out for each junction text file, the resulting list is converted into a Pandas series. This is the initial step needed to convert a list into a more useful data frame and allows for any brackets or other extra characters to be more easily removed.

Text

Description automatically generated

Fig 11. Part of function to extract junction information into Pandas series

Text

Description automatically generated

Fig 12. Part of function to convert Pandas series into csv file

Once in a Pandas series the datapoints are split into two columns, one containing the upstream junction and the other containing the downstream junction to which it connects. These are then each split into two again, with a column denoting the Node and a column denoting the Link for each junction. Finally, to complete the table an additional column is added to show whether a given junction has an entry link or not. Any row in the table that doesn’t contain an originating ‘From’ junction is marked as an entry link while all normal rows with a From and To junction are marked as not being entry links. The resulting table is then output into a csv file laid out like the figure below:

Table

Description automatically generated

Fig 13. csv output from text file extraction

### 5.2.6 Combining Connection and Location Datasets

While initially the plan was to join the resulting junction link table with the API flow table, this would only match the junctions in the ‘To’ column. Because of this another source of data was needed to provide location data for the ‘From’ junctions. Ultimately a table of Traffic signal coordinates was provided by Traffcom, which contained a list of every junction in Glasgow as well as it’s corresponding Eastings and Northings. Since the positional data from the API table was in Latitude and Longitude and the positions for the ‘To’ junctions were also available in Eastings and Northings, it was decided that it would make more sense to proceed using only the location data in the Traffcom dataset. This eliminates the need for coordinate conversion as this is a complex mathematical procedure that can introduce significant errors into the geospatial data.

Before the traffic data could be used however it needed to be prepared. As there was lots of data in addition to the site and coordinates these were dropped from the dataset and additional links from Dunbartonshire Council were added to enable the council boundary junctions to be included in the data. A brief analysis of the coordinates included in the dataset revealed that several of them were missing (had their eastings and northings set as 0,0) or had slightly off values, in some cases having the eastings and northings switched around and in other cases having an incorrect number of decimal places which causes the points to correspond to locations in Wales rather than Glasgow.

Table

Description automatically generated with medium confidence

Fig 14. Subset of Traffcom Location dataset after preparation

After simplifying the dataset and acquiring a table of corrected location data from Traffcom, the corrected data was substituted into the complete dataset in place of the original incorrect data, as seen in Appendix 1.

The final stage of the Python pipeline was to merge the location data with the table of link connections. This was achieved by first joining the position data on the From column, to attach the coordinates of the ‘From’ node, and then carrying out a second join on the ‘To’ column, creating a final dataset with each node followed by its corresponding location.

Table

Description automatically generated

Fig 15. Subset of completed location table

After creating the completed location dataset, the bearing nodes were separated into a separate table to allow for their angles to be calculated at a later stage, and the table of complete node-node pairs was exported as a csv for use in the next stage.

Diagram

Description automatically generated

Fig 16. Process Flow diagram of Python Pipeline

### 5.2.7 Generating and Collecting Flow Dataset from the Traffic API

#### 5.2.7.1 Complete Flow Data Pipeline

Now that the location data had been collected and collated, the final step before the data could be properly visualised was to collect the flow data. There were two main methods considered of how to pull in the flow data from the GCC’s servers, both with their own benefits and drawbacks. The first method was to create an FME pipeline that would directly query the Cosmos DB database that contained the flow data and ask it to send all flow data collected between the current datetime and the past 7 days. This was achieved with the SQL query below:



Fig 17. SQL query to pull in all API data from the past 7 days

While this query was able to pull in the complete data from the last 7 days, the sheer volume of data being pulled in caused the query to take an extremely long time to execute. When interfacing directly with the Cosmos DB server on the web, the query would take approximately 2 minutes to pull in the complete dataset, well within the 15-minute range between fresh readings being added to the database. However, when used in FME, due to the query having to be first sent through the Azure Rest API, this caused the fetch procedure to take closer to an hour, which is much too slow to allow the traffic operators to notice any changes in traffic in anywhere close to real time.

Diagram

Description automatically generated

Fig 18. FME SQL Pipeline to pull in complete data from past week

#### 5.2.7.2 Partial Flow Data Pipeline

The second method used was much more lightweight. It worked by querying the GCC’s Open Data Traffic APIs rather than the Cosmos DB directly, pulling in all the real time flow data for the current datetime, and then pulling in the corresponding data from the historical flow API exactly a week before the current datetime. This method was able to pull in the data far faster than the first method, taking only a few seconds to pull in the requested data. However, this came at the cost of less data being available to the user at any one time as only two time periods would be available at any one time, and these would be constantly changing every 15 minutes. This would drastically limit the usefulness of any analysis that could be carried out by Traffcom engineers and also removes the ability for different time periods to be selected, one of the key features requested in the original problem definition.

Diagram

Description automatically generated

Fig 19. FME SQL Pipeline to pull in current data and data from past week

#### 5.2.7.3 Ideal Flow Data Pipeline

While there wasn’t enough time to implement it during the 12-week project timetable, a solution which would combine aspects of both approaches was proposed. Initially the first pipeline would be run in order to pull in the complete set of traffic data to date, or to whatever timespan as Traffcom currently needs. After pulling in the full dataset, a modified version of the second pipeline would then be scheduled to run at 15-minute intervals, which would append the latest datapoints to the overall table. This eliminates both the drawback from the runtime of the first pipeline, as it only needs now to be run a single time, as well as the lack of data being pulled in by the second pipeline as it is instead merely updating the master dataset with fresh data every 15 minutes.

However, for the purposes of the current project the second pipeline has been implemented to provide a proof of concept for how the final visualisation could work. It will allow for the majority of the desired features to be demonstrated, particularly the direction of traffic flow and a comparison of traffic flows over different time periods (albeit only two).

## 5.2.8 Preparing Data for Visualisation

The method to be used for visualising the final set of data was settled on at an early stage of the project, ArcGIS, in particular ArcMap. This was decided upon for several key reasons, the most important being that it is already currently used within the Council for a number of other similar projects.

As a result of this there are already several people within the council with in-depth knowledge of the entire ArcGIS software suite, which will allow for the final visualisation to be maintained well into the future. It will even allow for the visualisation to be improved as it will be interconnectable with other ArcGIS packages the Council is already making use of, such as their cycling map and bus and taxi lanes map and will also allow for any additional maps the Council creates in the future to be added to the map. This will make the visualisation useful to various decision makers within the Glasgow City Council, in addition to just Traffcom.

To prepare the data already collected and processed for visualisation, a final pipeline was created. Initially both the current flow data as well as the historical flow data from a week prior are loaded into FME. The two tables are both then joined on their site columns, allowing for the current and historical data for each site to be contained in a single table. Once the tables are joined a new column, dist\_flow, is added to the table which takes the difference between the current and historical data flow, allowing for a comparison to be made once the data is loaded into the ArcMap.

Concurrently, the location dataset is loaded into FME, and the eastings and northings for both the from and to sites are converted into vertices so they can be loaded into ArcMap directly as connected lines. Following this, the now converted line data is joined with the combined flow table on the site column, allowing the current, historical and difference in flow to be seen for each site. Finally, the data is converted into an ArcGIS shapefile containing both the location and flow data which is output in an easily portable format that can be used by an ArcGIS package.A picture containing text, sky, map

Description automatically generated

Fig 20. FME Pipeline to join flow and location data into an ArcGIS shapefile

## 5.2.9 Visualisation and Final Tweaks

#### 5.2.9.1 Acquiring an Appropriate Base Map

When creating the final ArcMap visualisation, a simple, but essential first step needed to be carried out- selecting a base map. As the data used for the visualisation makes use of the British National Grid coordinate system, the default base maps included in the ArcGIS suite could not be used as they require data to use the WGS84 coordinate system. Due to this, a base map supporting the BNG system was downloaded from the official ArcGIS map depository and loaded into ArcGIS.

Diagram, engineering drawing

Description automatically generated

Fig 21. BNG Base map for Traffic Flow

#### 5.2.9.2 Adding in Raw SCOOT Link and Junction Shapefile

After the base map was loaded in, the point data for the SCOOT sensors placed around Glasgow was loaded in to allow the directional data to be sense checked. This SCOOT location data had already been made public by members of the GCC Data Team and was available for download directly using ArcGIS online repository of data sources. While the junctions and links don’t match up directly with the SCOOT sensor positions, and some junctions contain multiple sensors, by comparing the placement of the sensors and the final junctions it allows us to see if the junctions have been placed correctly.

Diagram

Description automatically generated

Fig 22. Map with SCOOT sensor point data added

The initial output from the visualisation pipeline can be seen in the figure below. A series of connected points representing the incoming and outcoming node for each junction can be seen, with each junction joining on to the next junction, giving the user a network map of the connections between each junction. However, by default there is no way to tell the direction of the traffic flow, as the node connections are simply displayed as lines with no arrows or other indicators to show the origin and terminal nodes. In order to combat this lack of directional information, further processing was required within the ArcGIS program itself.

Chart

Description automatically generated with medium confidence

Fig 23. Traffic Flow Data when initially loaded into ArcGIS

#### 5.2.9.3 Displaying the Directional Data

To display the direction of traffic flow, the default shapefile shape (in this case a straight line) for the traffic flow dataset should be selected by double clicking on it. Then the ‘Arrow at End’ symbol should be selected from near the bottom of the symbol menu, and for ease of clarity the width of the symbol was increased from 0.40 up to 2.00. This resulted in an ArcGIS map as seen in figure 25 below.

Graphical user interface, application

Description automatically generated

Fig 24. Adding arrows to Traffic Flow

Graphical user interface, chart

Description automatically generated

Fig 25. Traffic Flow with direction added

#### 5.2.9.4 Adding Colour Scales, Labels and Other Relevant Visual Information

While being able to see the direction of traffic flow is useful in and of itself, it doesn’t allow for any deeper level of analysis to be carried out. ArcGIS does offer additional functionality to allow for more data relating to both the junctions and their traffic flow to be displayed up front without requiring the user to delve deep into menus once they have been set up. By right clicking on the traffic flow data layer and selecting layer properties it brings the user to a menu similar to the one seen below in figure x. Selecting Symbology followed by selecting quantities in the Show column allows for the user to adjust the colours, value ranges and the labels of the symbols used in the layer according to a user’s self-defined criteria. In the case of the traffic flow data, these criteria are based on the flow difference quantity which is a calculation of the difference in traffic flow across two different time periods.

For Traffcom, comparing traffic flow as a universal value across multiple junctions doesn’t make much sense as a 5-lane junction can support far more cars driving on it than a 1 lane junction yet if both had a raw traffic flow value of say 30 cars, they would show as suffering from the same amounts of congestion. Due to this it makes far more sense to compare the same junction to itself over different time periods, and as a result of this the flow difference is only calculated for a specific junction at different time periods. The resulting colour scheme for this comparison would work as follows:

* If traffic at the present time is more than 20% worse (i.e., the value of present traffic flow is 20% greater than the past traffic flow) than traffic at the past time to which it is being compared, then the arrow should be coloured red.
* If traffic at the present time and traffic at the past time are within +/- 20% of one another then the arrow should be coloured orange to show that the traffic is relatively similar.
* If traffic at the present time is more than 20% better (i.e., the value of present traffic flow is 20% less than the past traffic flow) than traffic at the past time to which it is being compared, then arrow should be coloured green.

.Graphical user interface, text, application

Description automatically generated

Fig 26. Adding arrows to Traffic Flow

After selecting these ranges in the Symbology tab, each arrow is then changed to match the colours specified above. Labels are also added to provide a brief explanation of what each colour means to any unaware users. Finally, by navigating to the Labels tab in the figure above, the site field is selected as a label to allow each junction to be quickly identified at a glance, with the deeper information still remaining just a single right click away.

# 6.0 Conclusion and Future Work

## 6.1 Results to Date

Map

Description automatically generated

Fig 27. Final Traffic Flow

The results of the project have so far been very promising, with nearly all of the original requirements desired by the Council being met and then some. The map currently displays over 1000 individual link connections, overlaid on to the streets of Glasgow. The map view is capable of being moved around the city to the user’s area of interest and can be zoomed in and out to the desired extent needed for the user’s query.

Map

Description automatically generated

Fig 28. View of All Links Displayed on Map

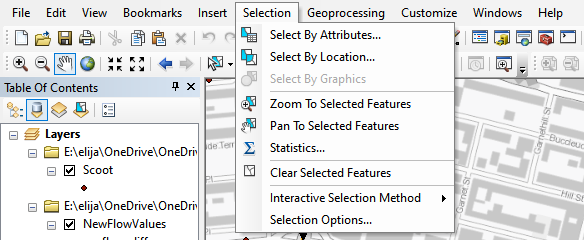
Each link’s traffic flow direction can be ascertained from the directional arrow present on each link, allowing for the user to easily follow traffic flow from junction to junction across the desired subsection of the network needed. The junctions have also been colour coded to reflect the flow differences that were discussed in section 5.2.9.4, allowing the user to compare the relative difference in traffic flow across the two time periods at a glance, with toggleable labels being added for each of the junctions to allow the user to tell what the name of a given junction is without having to access the menu, whilst simultaneously allowing them to turn the label off to prevent certain areas of the map from becoming too cluttered.

Of course, the most useful functionality added is the ability for the user to select a junction and then right clicking on it and selecting identify to view further information on the junction’s properties. The user is able to see the ID of the two nodes that make up the junction and their spatial coordinates, as well as being able to see the junction’s current flow value, the equivalent flow value from a past time period and ultimately the flow difference between the two. Map

Description automatically generated

Fig 29. Sample of Viewable Properties for each Junction in Visualisation

A final useful feature is the ability to manually search for a desired junction, even if the user is unsure of its position on the map. This can be used by navigating to the Selection tab in the ArcGIS taskbar and then using the Select by Attributes option. This brings up a popup box that allows the user to query the flow dataset in a variety of ways, such as selecting a list of unique junctions in the dataset, searching for a specific junction by its name or highlighting all junctions that have flow values or positional coordinates within a certain range to limit the user’s analysis to a specific area or subset of the dataset.



Graphical user interface, map

Description automatically generated  
Fig 30. Using the Selection Search to Locate a Junction

## 6.2 Application of Results

Now that a flow visualisation including both the value of traffic flow and the corresponding direction, as well as the underlying data containing all the information needed to create it, has been realised it allows for various different types of users to benefit from its use. These can be broadly split into Governmental Bodies who directly gather and manage the data, private entities who can make use of the data for their own needs but who don’t have direct access to the traffic control system, and Glasgow City residents who will be able to use both the data and visualisation for their own personal use. In the next few sections, some more in-depth use cases for different end users will be explored.

### 6.2.1 Traffic Engineers

Traffic Engineers will be able to use the ArcGIS visualisation to extend the functionality of the pre-existing traffic manager in a number of ways. While the traffic flow at each individual junction can be seen in the original SCOOT Urban Traffic Control (UTC) system, there is no further detail available for the junctions and it is currently impossible to follow the flow of traffic across more than a single junction without significant trial and error. With the new system, not only can the engineers easily follow the traffic flowing from junction to junction across the entire city network, they’ll also be able to quickly search for any specific junctions they need using the inbuilt search function- a function which is not present in any form in the original system.

More in-depth analysis will be able to be carried out by the traffic engineers in the actual visualisation, without needing to go back to the raw dataset. Traffic flow across different time periods can be easily compared for each junction, which will allow the engineers to discover trends across junctions. This could be noting that a particular junction tends to get busy at the same time on a Tuesday each week, to more overarching trends such as being able to track the increase in traffic flow in certain areas of the city depending on the time of the year, such as city centre streets drastically increasing in traffic flow around the Christmas period. It could also help the traffic engineers to make better decisions by using the already existing data. A prime example would be deciding the best time to carry out roadworks or other services on a given junction. By monitoring the traffic flow through the junction at various times throughout the week, the engineers would be able to find the period with the lightest traffic flow and timetable the road closures necessary for work to be carried out around this, reducing the impact closing the road will have on the surrounding junctions’ traffic flow. Further expansion of this capability could ultimately lead to a model for predicting future traffic conditions based on past data.

A final key feature that doesn’t directly affect the traffic engineer’s day-to-day responsibilities but is no less useful is the ability to quickly add new junctions to the visualisation. With the current UTC system, for a new junction to be added all the data containing its connections, location, and other related data, all of this data needs to be manually entered both into the master text file, and into the actual system itself. This is a classic case of data redundancy as it requires the same data to be entered several times and can lead to all sorts of issues down the line if the two sets of data don’t exactly match. Fortunately, with the new system, the data for a new junction just simply needs to be added to the master text file once, and it can then be passed forwards into the visualisation automatically, simply by rerunning the Python/FME pipeline to process the new data. This dramatically cuts down the time needed to add a new junction and removes the risk of any data inconsistencies, allowing the traffic engineers more time to focus on their actual duties.

### 6.2.2 City Stakeholders and Decision Makers

There are multiple city stakeholders and decision makers who would both benefit from and potentially contribute to the traffic flow visualisation. These include but are not limited to:

1. Hauling Companies – Drivers and Fleet managers can better plan their routes using the traffic flow data to avoid high traffic areas. This helps them to optimise their routes, saving both time and fuel, whilst helping to reduce the emissions produced by the vehicles. This is of particular importance to Glasgow as they are hosting the upcoming COP26 climate change conference, and this will help them to meet their targets in time.
2. Vehicle for hire companies – Companies that carry people from one place to another, such as Taxi firms or Uber, will benefit from free and enhanced access to traffic flow data. This will allow them to better optimise the algorithms they use for route planning, which can enable them to transport higher volumes of passengers and lower costs. This would likely incentivise more members of the public to use these companies instead of their own vehicles, helping to reduce the number of vehicles on the road and minimise congestion.
3. Retail companies, along with bars and restaurants – Companies like these which rely on a steady flow of people passing through an area to do business, would be able to make use of the traffic flow data to locate areas of high and low traffic. As a result of this, decision makers within the companies would be able to better choose locations for their stores to maximise the number of people passing through that could potentially shop there, ultimately leading to an increase in business. At a higher level still, it would help both public and private decision makers to identify potential areas for retail parks and industrial estates.

### 6.2.3 Glasgow City Residents

The residents of Glasgow themselves will also be able to benefit both directly and indirectly from the release of the visualisation and it’s supporting data. Less tech literate citizens will now be able to view the traffic around the city without needing the coding abilities necessary to work with the APIs currently released by the Council. Citizens will also be able to keep themselves informed of traffic policy to a much higher degree due to the release of this data, which will further allow them to form their own opinions on traffic policy and other sectors of the government. It will also enable more tech literate citizens to access the current existing system and augment it in innovative ways, which could lead to greater collaboration between the public and private sectors and lead to an explosion in new products and ideas.

As well as being directly able to make use of the visualisation and data, citizens will also benefit from other indirect ways. For instance, due to traffic around the city being easier to manage for traffic engineers, this will hopefully lead to a knock-on effect of reduced congestion, which will greatly improve the quality of citizens’ lives due to having to spend less time in traffic and more time on personal or professional growth. Furthermore, as traffic engineers will be better able to predict where and when high congestion will occur, additional public transport can be brought in to help offset this increase. If more citizens make use of public transport, particularly trains and the subway although buses will still help to a lesser degree, this will help to reduce the congestion at these peak times even more, which can then allow Glasgow Council to better enact its future plans to increase pedestrianisation and green areas around Glasgow.

## 6.3 Limitations

While the project was successful in allowing the direction and intensity of traffic flow to be accurately displayed on a map of Glasgow, it still suffers from a number of limitations. Currently the system only contains data from two particular time periods, which have to be hardcoded into the FME pipeline for fetching flow data. This unfortunately limits the current usefulness of the visualisation as comparing different time periods requires both a tech savvy user and a not insignificant amount of coding. This needs to be expanded upon to include a wider time period of flow data to compare, without slowing down the speed of importing data to a greater than 15-minute wait (as this is the amount of time between live data updates).

The visualisation itself suffers from a few imperfections, mainly that not all the links perfectly match their corresponding road. While the majority of the junctions match up cleanly, more complicated road systems or even just lanes that have a large degree of curvature don’t match up. Additionally, in denser areas such as the city centre that have multiple junctions on the one road, the junctions can end up being overlaid on top of one another, making it difficult to ascertain which junction is flowing in what direction. The only fix for this requires a user to manually adjust the road links to make them better match their roads or to slightly offset one another to improve clarity.

There are also some deficiencies in the data behind the visualisation. As some of the junctions are missing either positional or flow data, this means that when a join is carried out during the visualisation pipeline, any junctions without all the necessary data don’t get matched up and are therefore not displayed on the visualisation. The pipeline itself is also not completely future-proofed as while it currently has a hard coded data pull limit of roughly twice the size of the current Cosmos DB dataset, this will require someone with sufficient understanding of Python to manually increase it once the size of the dataset has surpassed this threshold. Further to this, the save locations for the intermediate files used when processing the text and location data are manually specified and will need to be adjusted when the project is handed over to the Council, either to allow each individual user to use the necessary files, or more likely to allow it to run on the Council’s server set up.

A final potential limitation involves the coordinate system that the visualisation is built upon- the Ordnance Survey National Grid reference system. This is a geographic grid referencing system that measures coordinates from an origin point off the southwest coast of Great Britain, in eastings and northings, both in metres. While this allows for a more accurate ability to locate places in Britain, it is being increasingly phased out by the majority of geographic systems, in particular the EU and Google, and nowadays the main system in use is the Universal Transverse Mercator coordinate system, which makes use of latitude and longitude measured from the equator in metres, rather than from an arbitrary point near the British coast. As a result of this, and the fact that both systems use slightly different projections of the globe, it is difficult to use the two coordinate systems interchangeably, due to complex mathematical transformations necessary for conversions, and this severely limits what datasets could be added to the visualisation. Both solutions are costly and time consuming, either converting all the Council’s datasets to use latitude and longitude or converting all other future datasets that may be used into eastings and northings.

## 6.4 Suggested Further Work

### 6.4.1 Data Gaps and Data Augmentation

At the time of this project, there are several missing data points for both the position and the traffic flow for some of the SCOOT links. The position for the missing data points could be found and added to the master database to increase the number of links the ArcGIS visualisation is able to display, while further investigation could be taken into the reason for missing periods of flow data in the dataset.

The project currently also excludes a number of more difficult to model links, including entry, exit and filter links. While it is theoretically possible to calculate the positioning of entry links by taking a bearing halfway between the two links it is connected to, this is programmatically difficult and so has not been included in the current incarnation of the project. Even more difficult still are exit and filter links, as there is no way to easily calculate their positioning from the data contained in the master text file alone, and so additional data would be required to supplement the original data and allow for their inclusion.

The utility of the visualisation could also be improved by adding additional data or transforming the currently used data. Glasgow City Council already collects a wide variety of geospatial data from across the city including pedestrian activity, bike hire and cycle traffic as well as train and subway usage. These could be layered on top of the existing map to provide further insight into transport patterns across Glasgow. A heatmap or some sort of visual symbology to differentiate the volume of traffic flow across the city could be used to highlight areas of key interest- these could be averaged out over a variety of time periods to allow traffic engineers and urban planners to see trends more clearly than from the live map alone. Time periods of particular interest as suggested by the Council include:

1. Weekdays AM peak (from 7 to 10 am)
2. Interpeak (Between 10am and 3pm)
3. Weekdays PM peak (from 3 to 7pm)
4. The same time periods as above but for the weekend

Breaking the data into these subdivisions would allow users to better compare time periods to note any surprising differences between them. For instance, they could compare an AM peak weekday from 2020 and compare it to an AM peak weekday from 2019 to ascertain what effect covid may have had on traffic glow around Glasgow. Similarly neutral time periods from months with fewer public/school holidays or reasons for non-standard traffic patterns (e.g., a time period from November) could be used as a benchmark to compare the rest of the data against. Aggregating the data like this could also help to improve the speed bottleneck when loading in the traffic flow data, as if the data is pre-aggregated it requires less processing power, and therefore time to load into the visualisation.

### 6.4.2 App Improvements and Data Sharing

# Appendix

Text

Description automatically generated

Appendix 1. Function to prepare positional dataset for joining followed by joining script