Table of Contents

[1. Introduction 2](#_Toc20741877)

[2. Graph Visualisation Metaphor 2](#_Toc20741878)

[3. The Cycle of Visual Model 2](#_Toc20741879)

[4. Specification of Visualization 6](#_Toc20741880)

[4.1. Specification of Layout Design 6](#_Toc20741881)

[4.2. Specification of Graph Design 8](#_Toc20741884)

[5. Specification of Navigation 13](#_Toc20741892)

[5.1. Specification of View Transformations 13](#_Toc20741893)

[5.2. Specification of HCI Design 15](#_Toc20741894)

[6. Reference 16](#_Toc20741895)

# Introduction

Data visualization is a significant industrial issue which is related to human-computer interaction (HCI). Data visualization can be defined as a way of presenting data to increase the readability of data and accelerate human cognitive processes. Compared with other data visualization, graph visualization tends to emphasize the structure of data. This report will present an appropriate graph method to present some Australian flight data with a suitable navigation scheme.

Our link of graph is: <http://www-student.it.uts.edu.au/~zhiachen/FlightDemo.html>

And this is our GitHub link: <https://github.com/baojian123/UTS_DVVA_A2>

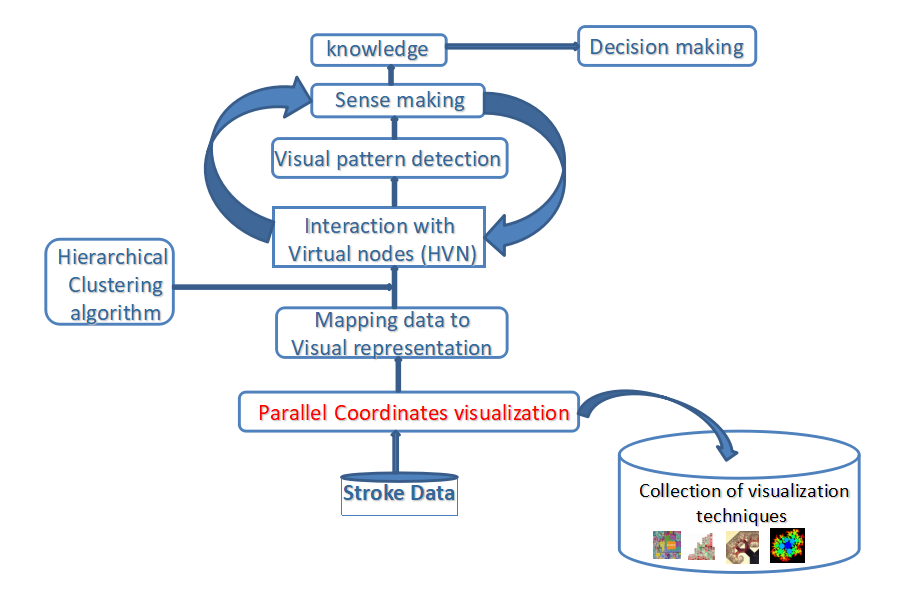
# Graph Visualisation Metaphor

Based on data visualization, graph visualization tends to emphasize the structure of data. In this report, we will design a graph layout method and a navigation scheme given some Australian flight data. A great visual metaphor can present more than 90% of properties of data. With the progress of data visualization, so many metaphors are developed for representing data, such as tree, pie chart, line chart, table and so on. For designing a great metaphor, we decide to use a force directed graph positioned on a map as the main graph visualization metaphor, because force directed graph can apparently describe the complex relationship of flight between cities. Moreover, positioning the force directed graph on a map can positively impact human cognitive process. We place the nodes in the corresponding positions on the map, so that the user can immediately identify the cities. Nevertheless, some details of flights cannot be presented by graph and map. A navigation scheme, which will present formatted text data when user selects one flight on the map, has been designed to address this issue.

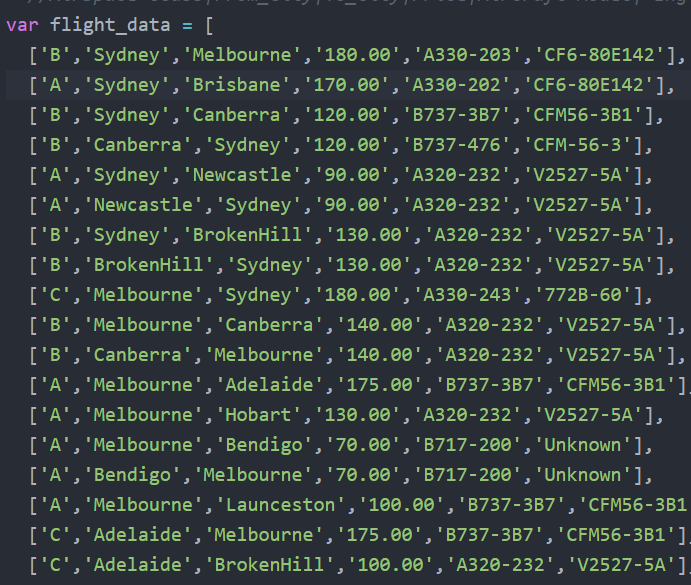
Differ from other traditional charts, force directed graph is more complex for graph drawing due to too many parameters. Fortunately, with the increasing popularity of data visualization, there are some visualization tools which can be searched online, like D3.js, Power BI and Tableau. This report adopts a JS library called amCharts 4 which is similar with D3.js. This library provides a comprehensive interface for different graph, even a world map. For learner, it also provides a document to state how to use the different interfaces.

# The Cycle of Visual Model

Before talking about the graph, we should transform the raw data to the suitable format which can match the interface provided by the graph tool. The general flow chart is like the figure 1.

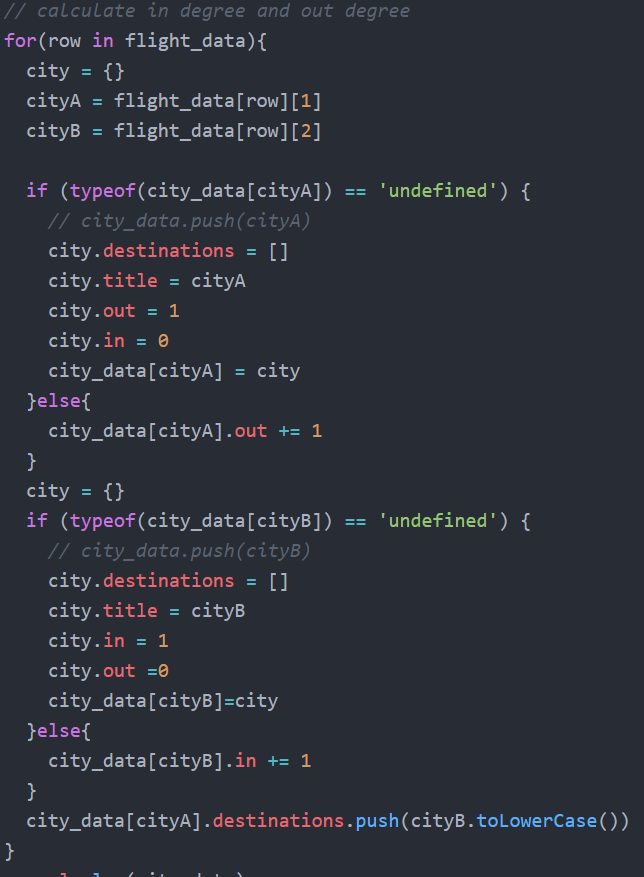


1. The flow chart of visual data analysis (adapt from week 8 lecture slide, p.40)



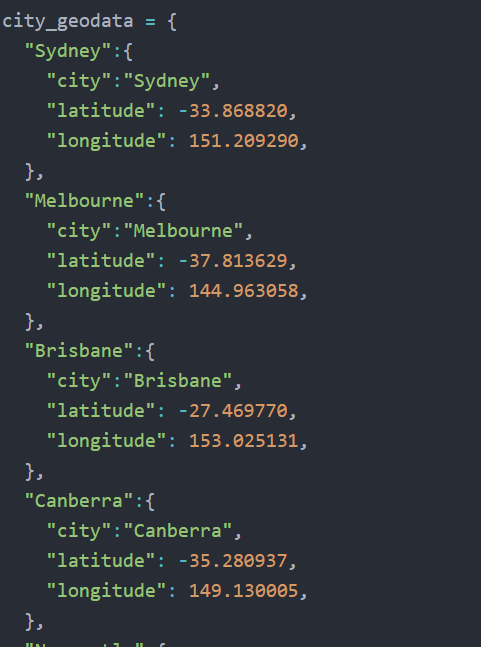
1. The data of flight

Firstly, we should manually collect the data from the pdf file. These data are flight data. In our graph, the flight is corresponding to the edge, so that we should extract the node data from these flight data. As shown in figure 3, we push new nodes into a mapping and calculate their in-out degree. We will use these in-out degree later.



1. The code of selecting the city and calculating their in-out degree

Afterward, we manually search geographical information for each city, involving latitude and longitude, and add them in another mapping.



1. The geographical information of each city

Finally, we transform the data to an array of node. Building a mapping is convenient for searching the information.



1. The code of transforming the data format to the format required by amCharts interface

# Specification of Visualization

We use force directed graph to represent the flight data. This will derive some problems in graph drawing, because the inner relationship of a graph can be very complicated. Jianu et. al. (2009) claim that one of the classic issues in graph drawing is how to satisfy the aesthetic rules, for example, minimizing the edge-crossings. The reason why the graph needs to meet these requirements is that these aesthetic rules are devoted to improving the human cognition process. For instance, the edge-crossings negatively affects the discernibility of edges, and then the readability of the graph will be declined. Jianu et. al. (2009) argue that it is impossible that drawing some graphs sourced from real-life without edge-crossings. In this report, we assign the nodes to their corresponding position on the map. Hence, the edge-crossing problem cannot be escaped, and there will be an edge-overlap problem and node-overlap based on the data. For this section, we will introduce how to address these challenges.

## Specification of Layout Design

**Edge-crossing**

As mentioned above, we have stated that the edge-crossings problem cannot be escaped due to the fixed position of each node. However, the effect caused by this problem can be minimized by assigning color for each class of edge. Jianu et. al. (2009) propose a scheme that they paint the crossing edges with perceptually opposing colors to reduce the impact of readability affected by edge-crossings. In this report, we use a similar scheme to address edge-crossings. Moreover, we set the opacity of each edge into 0.5 and use arcs instead of straight lines to reduce the influence by edge-crossings further. As shown in figure 6, even though the edges are dense and crossed, it does not affect the readability of the graph. The user still can identify each edge.

In addition to edge-crossing, there is also an edge-overlap problem which need to be solved. There are some pairs of opposing flights between some cities. If each pair of flights is one plane with different directions, we will use an undirected edge with a round-trip plane animation to present each pair of flights. However, as demonstrated in figure 6, some pairs of flights consist of different planes with different classes and aircraft model, so that we cannot use undirected edge to present these pairs of flights, and this leads to an issue of edge-overlapping. In order to address this issue, we adopt 2 arcs to replace 2 straight lines as edges.



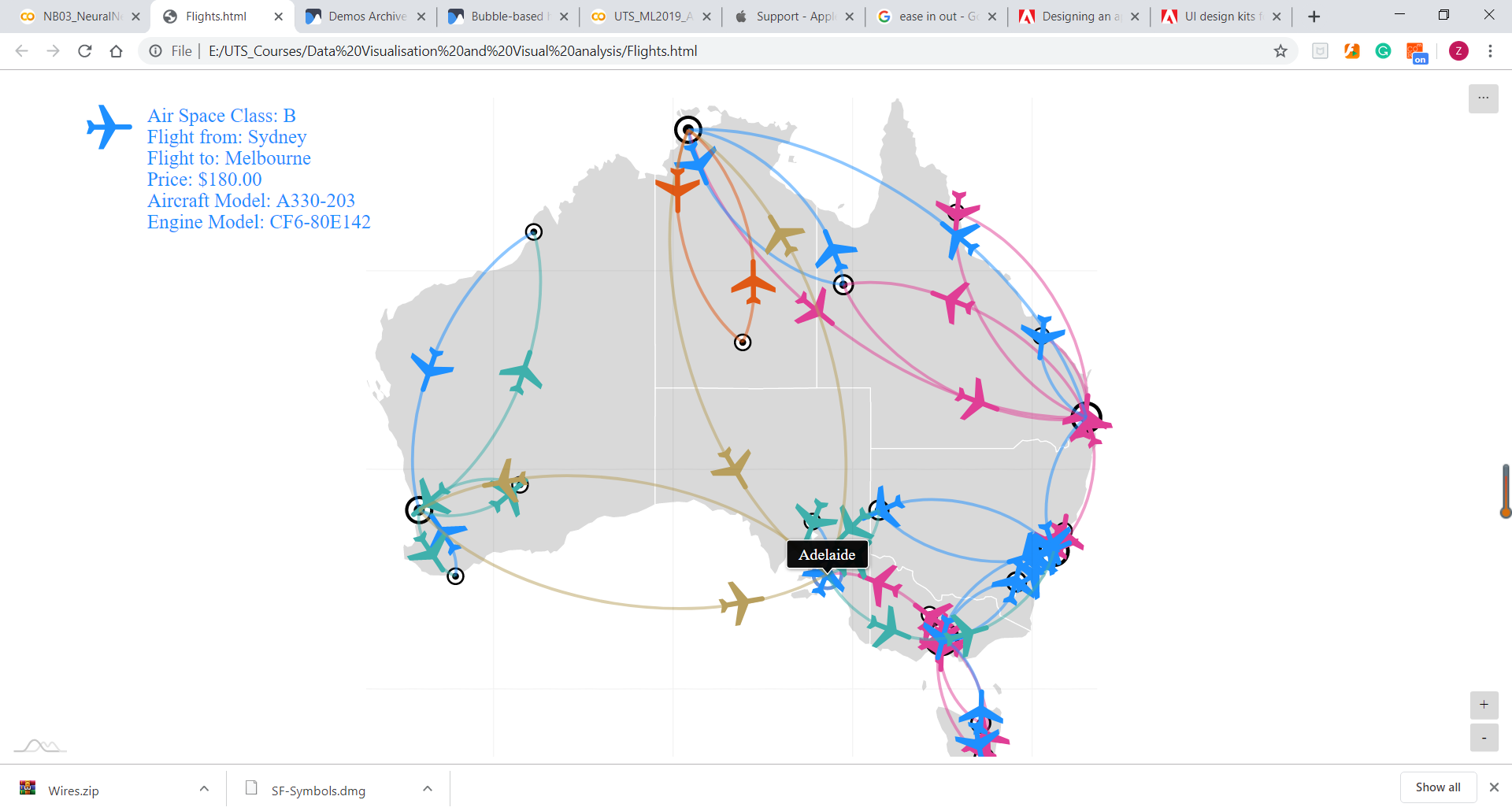
1. The solution of edge-crossings problem

**Node-overlap**

We conduct a city set consisting of unique cities and generate nodes based on this city set, so that each node included in the graph represents a unique city respectively. Because all nodes in graph are positioned by the latitude and longitude of the corresponding cities and positions of all cities are unique, the node-overlapping problem will not happen in our layout algorithm. In addition, the latitude and longitude of each city are searched by LatLong.net <<https://www.latlong.net/>>.

**Labelling techniques.**

Labelling the data is a significant part to enhance the readability of graph. We do the labelling though giving a tip text of the name of the city when the user hovers on a node. Moreover, we use different colors to identify the flights of different airspace classes as shown in figure 7.



1. The overview of the graph

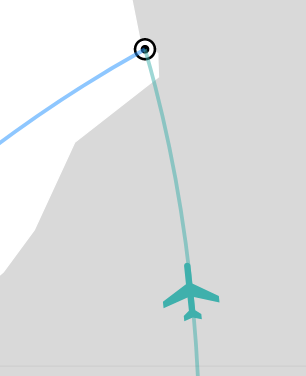
**Increase readability of the layout**

For enhancing the readability of our graph, we have used some methods mentioned above, such as assigning different colors to different class to help identifying different airspace classes; using the real-world position of each city and giving a placeholder for each city to rise the discernibility of node; using arcs instead of straight lines to avoid edge-overlapping and some edges across nodes; and looping an animation of an airplane flying from one city to another on each arcs. These methods all hold positively impacts to human cognition process.

## Specification of Graph Design

**Graphic objects design**

As shown in figure 8, there are two kinds of object involved in our graph, one is city and the other is flight. Each city is located on the real-world position on the map whereas each flight takes off from departure city and lands in destination city. Clicking a flight will display the its details.

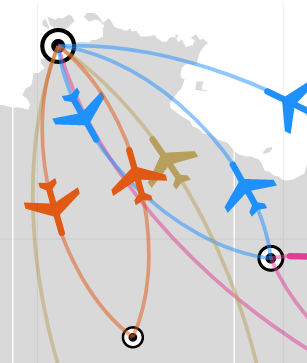


1. Two kinds of object in graph

**Graphic attributes design**

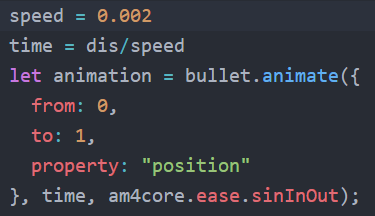
In this section, we will describe the domain-specific attributes design rather than graphic attributes design, because we want to provide the reason why we design the attribute via real-world information. We will describe the mapping from domain-specific attributes to graphic attributes later.

The object of city has 5 domain-specific attributes: city name, latitude, longitude, scale of city and number of flights. The name of city will be displayed when the user hovers the city. All cities are located on map according to their latitude and longitude in real-world. Moreover, we set the scale of cities according to the number of flights they have, because we believe that a city with more flights may mean that it has better development and larger scale. The view is presented in figure 9.



1. The different scale of cities with different number of flights

However, the object of flight has 7 domain-specific properties: speed, airspace class, from city, to city, price, aircraft model and engine model. If all the planes which fly with different distance take off and land at the same time, it will look a bit weird, so that each plane should have a fixed or similar speed. However, the animation provided by CSS and amCharts merely requires the parameter of playing time. We need to transform the speed to time by ourselves. Therefore, we set a fixed speed for each plane by calculating the distance between departure and destination and dividing distance by speed, and we set ease-in-out in animation to simulate the take-off and landing of the aircraft. The code is shown in figure 10.



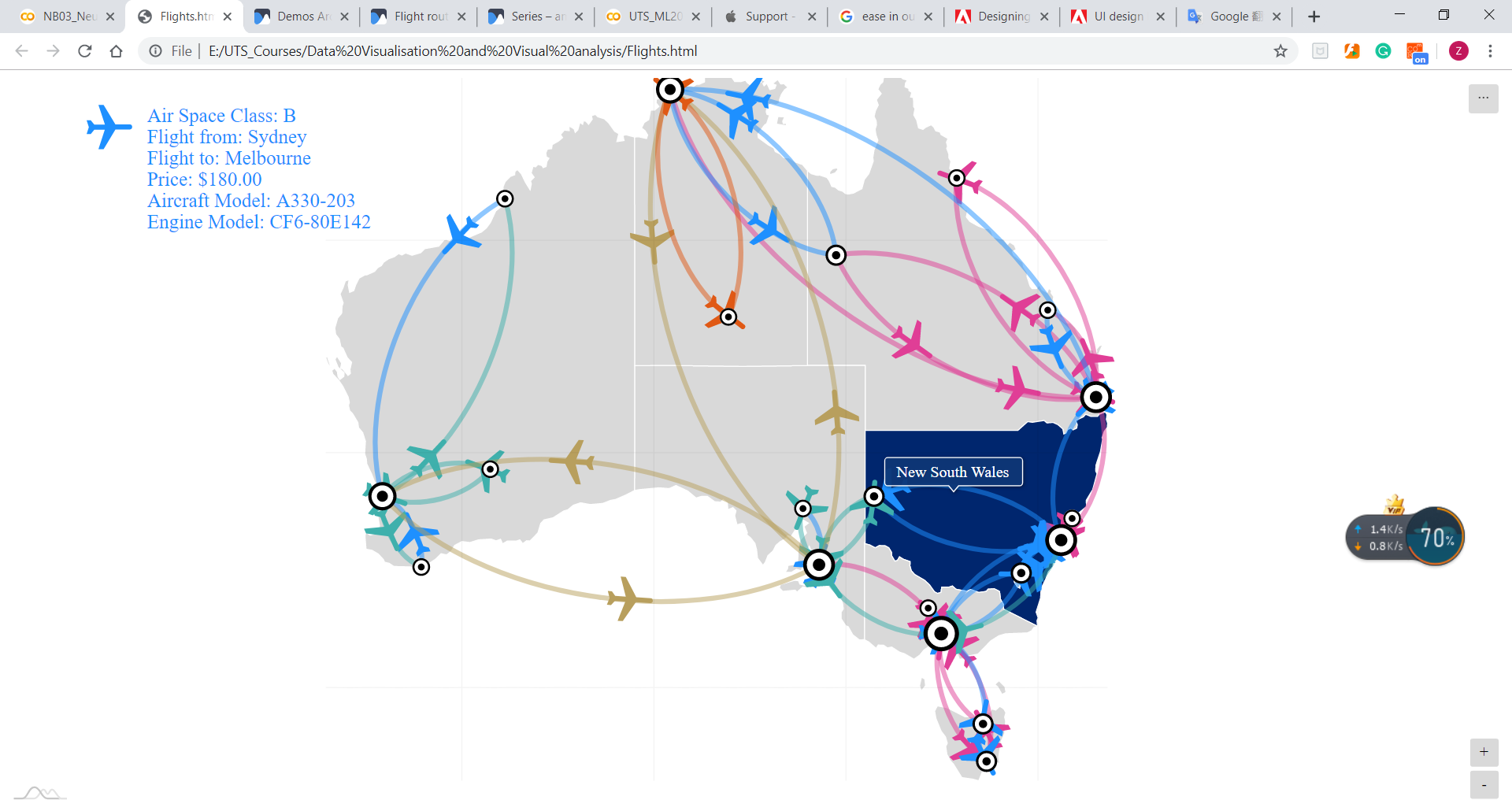
1. The code setting the fixed speed for flights

As presented in figure 11, we can directly identify some attributes of a flight via map, such as from city, to city and airspace class, by adding a plane animation and filling different color based on airspace class. Furthermore, when the user selects a flight or a plane on the graph, all its details, except speed, will be displayed on the upper-left corner of the screen. For instance, in figure 11, we have selected a flight from Perth to Albany, its details will be displayed on the upper-left corner.



1. The example of displaying details of flights

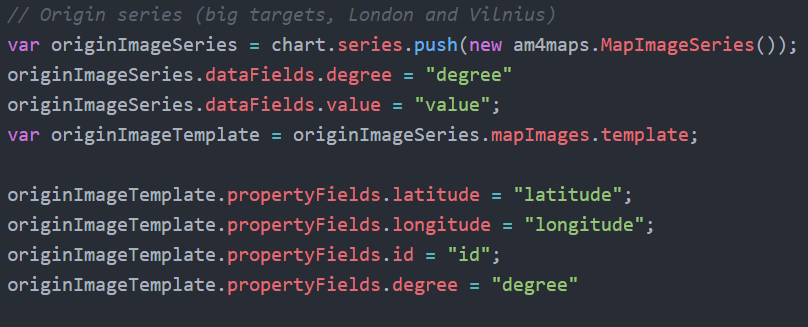
For partitioning, we group all cities in terms of the states of these cities. The data of state is provided by amCharts Geodata library <<https://www.amcharts.com/lib/4/geodata/json/australiaLow.json>>. The partitioning of states is implemented by lots of coordinates, and there is no interface for state. As a result, we cannot transform the city view to state view by clustering all the flights of a state into state. The difficulty of implementing this technology is too high.



1. The node grouping by state

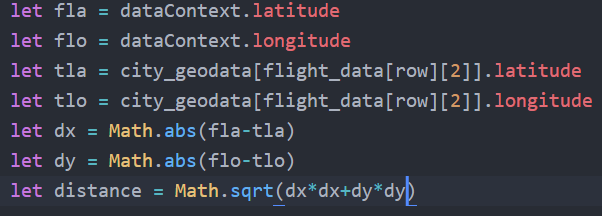
**Mapping domain-specific attributes to graphic attributes**

As mentioned above, the object of city has 5 attributes: city name, latitude, longitude, scale and number of flights. These 5 attributes are domain-specific attributes, but we need to convert these attributes to the graphic attributes which we need in graph drawing. The city name can be directly transformed to node id. The latitude and longitude of a node are related to the position of this node on the map, so that they need to be transformed to X axis and Y axis coordinates. Fortunately, amCharts has provided a map drawing interface which can draw the node via its latitude and longitude on the map, so that we just need to provide the latitude and longitude. For the scale and number of flights, they can be converted to the size of node and in-out degree of the node, and we set the size of node via the in-out degree. We believe that this can largely improve the human cognition process.



1. The code for mapping domain-specific attributes to graphic attributes

The object of flight has 7 domain-specific properties: speed, airspace class, from city, to city, price, aircraft model and engine model. In above sections, we have explained the reason and transformation from flight speed to playing time of animation. However, we still can talk about how to calculate the distance between 2 cities. As illustrated in figure 14, we calculate the difference between 2 cities via latitude and longitude, and then use Pythagorean theorem to calculate the distance. This method is commonly applied in calculating the distance between 2 point on coordinate, but they calculate the distance by x and y coordinates rather than latitude and longitude. Except airspace class which can be mapped via filling different colors, other domain-specific attributes just display on the upper-left corner as shown in figure 11 when user clicks a flight.



1. The calculation of distance

**Data scale problem**

The computational complexity of our code is *O(cm)* while the *c* is the cost of drawing a line and adding an animation and the *m* presents the number of edges. Given a fully connected directed graph with *n* nodes, our code needs to draw the sum from 1 to (n-1) times. This cost is acceptable with several edges. Moreover, if we want to apply our graph in a large dataset and optimize our graph. We just can optimize the *c* cost by removing the animation.

In addition to computational complexity, the key issue of large data scale is that the complicated relationship of graph destroys the readability of graph and leads to the edge-crossings and node-overlapping problems. Our graph cannot escape from these issues, but we can minimize the effect affected by them. For example, we have proposed a scheme to reduce the impact caused by edge-crossings by assigning different colors and a 0.5 opacity for each line. If the data scale become extremely high, we will consider minimize these effects via developing a navigation scheme. We can limit the information displayed simultaneously by adding some filters classifying data into different categories in terms of various properties, such as state of city, airspace class of flight and range of price. As we know, some IT corporations also adopt this solution to display parts of extremely large dataset to customer.

**Enhance the readability of domain-specific attributes**

We have described how we contribute to readability of domain-specific attributes in Graphic attributes specification section.

In this section, we will talk about the improvement of our current plan. Firstly, the text displaying 6 attributes of a flight as shown in figure 11 is very basic representation and does not make sense. We plan to use a table to improve the representation; add a description for different airspace classes; and add an image for each aircraft model and engine model. For map, we plan to add a weather map to remind users to pay attention to the weather.

# Specification of Navigation

For navigation algorithm, we choose zooming and pan as our navigation algorithm and combine it with a zooming interaction scheme. When you click on a node, view will focus on that node. Afterward, when you click on the Australian map on the graph, the view will go home. In previous section, we have mentioned that the size of each nodes is different according to the degree of node, and we try to give the small city a larger magnification, because while the megacities have more flights and information, small cities can have more space to represent them. Therefore, the zooming level for each city is different according to the scale of city. The user also can zoom in or out via using the plus and minus buttons or the mouse wheel. Moreover, our graph allow user dragging the map to pan the view.

## Specification of View Transformations

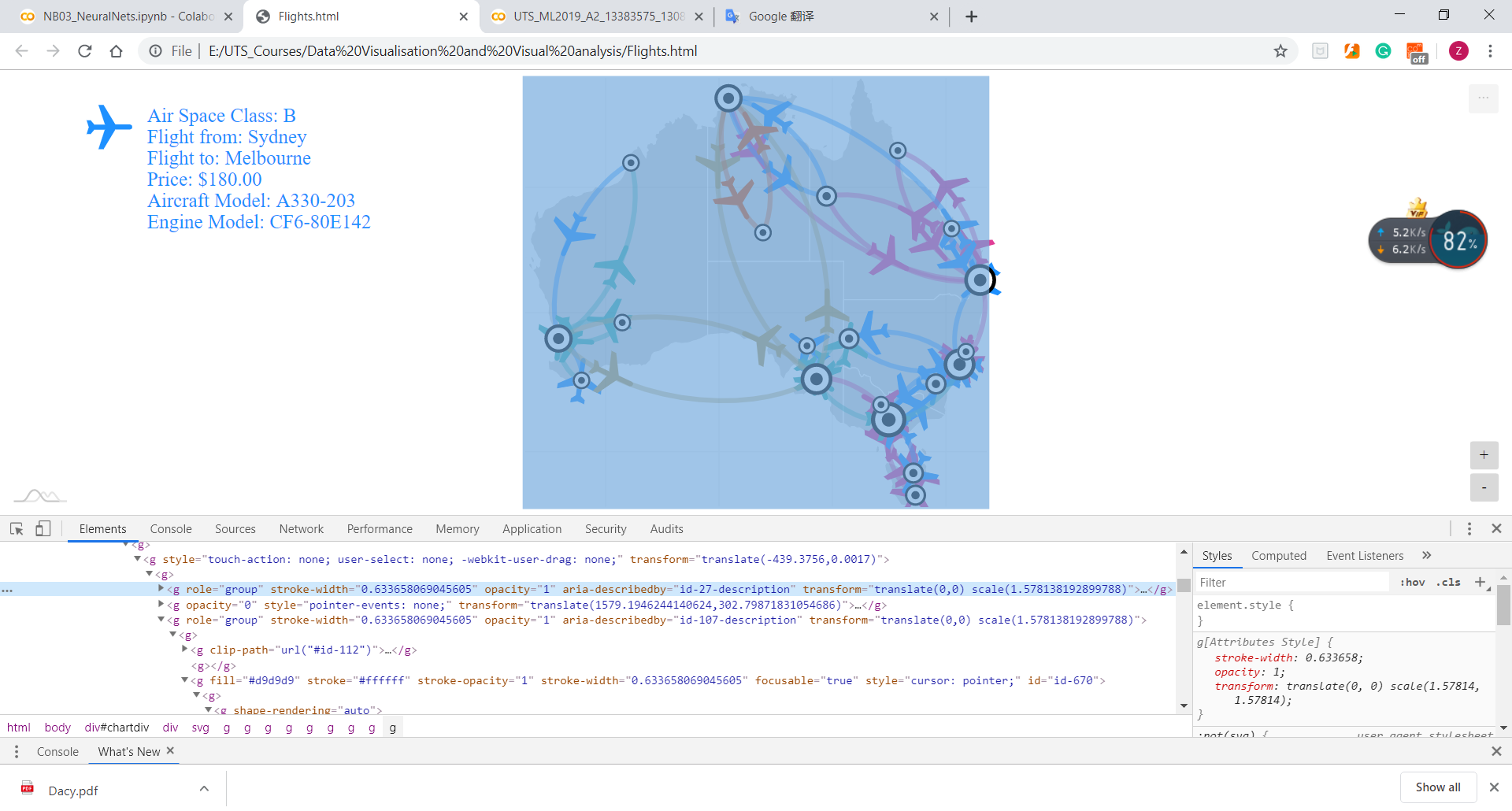
**In-between views design and transformation algorithm**

In in-between views design and transformation algorithm, because we use animated viewing algorithm implemented by amCharts. Therefore, the scale of zooming is the continuous interval from 1 to 32, so that we have so many in-between views and we will describe animated viewing algorithm rather than in-between views.

**Animated viewing algorithm**

For animated viewing algorithm, we have set the animation of plane and zooming. Because when we zoom in on an area without any animation, the suddenly change of view will make user confused and surprised. One of the design principles by Sommervile (1995) is minimal surprise that the action of a system should not make the user surprised. Hence, we add an animation to make the movement of zooming in or out understandable.

we use the zoom control interface provided by amCharts library to control the views. Although we implement the zooming and pan via library, we know how to implement this interface. The CSS provides a property of transform and many functions for transforming the views of tags. In the amCharts interface, they implement zooming and pan via translate and scale function in CSS. When we use the mouse wheel to zoom in or zoom out, the JavaScript code will monitor the position of cursor and calculate the parameter of translate and scale. With the same algorithm, when we click on a node or adjust the width and height of browser, the parameter in translate and scale will be changed. The JavaScript provides listeners of monitoring the modification of width or height of window.



1. The scale in HTML source code

**Human cognition process consideration**

For the reason why we use animated viewing algorithm, we believe that the animation for zooming in and zoom out is to imitate the process that a person holds a magnifying glass to zoom in and out. Moreover, the design of implementing panning via dragging the graph is inspired by the behaviour from real world. These 2 designs satisfy one of the usability heuristics for user interaction design by Nielsen (1994) “Match between the system and real world”. Hence, we believe adding an animation can improve the human cognition process.

## Specification of HCI Design

We also pay attention to Human-Computer Interaction (HCI). In this section, we will estimate our graph though Fitts’ law and Nielsen’s usability heuristics.

**Fitts’ law**

In 1954, psychologist Paul Fitts proposed a concept for HCI, that evaluates the performance of a system via the time it takes for the user to interact with the target area by moving a pointer. According to Fitts (1954), this spending time is proportion to the distance to the target and the size of target. Longer distance and smaller size will lead to longer time. Furthermore, the fast movement and small size of the target also result in higher error rates.

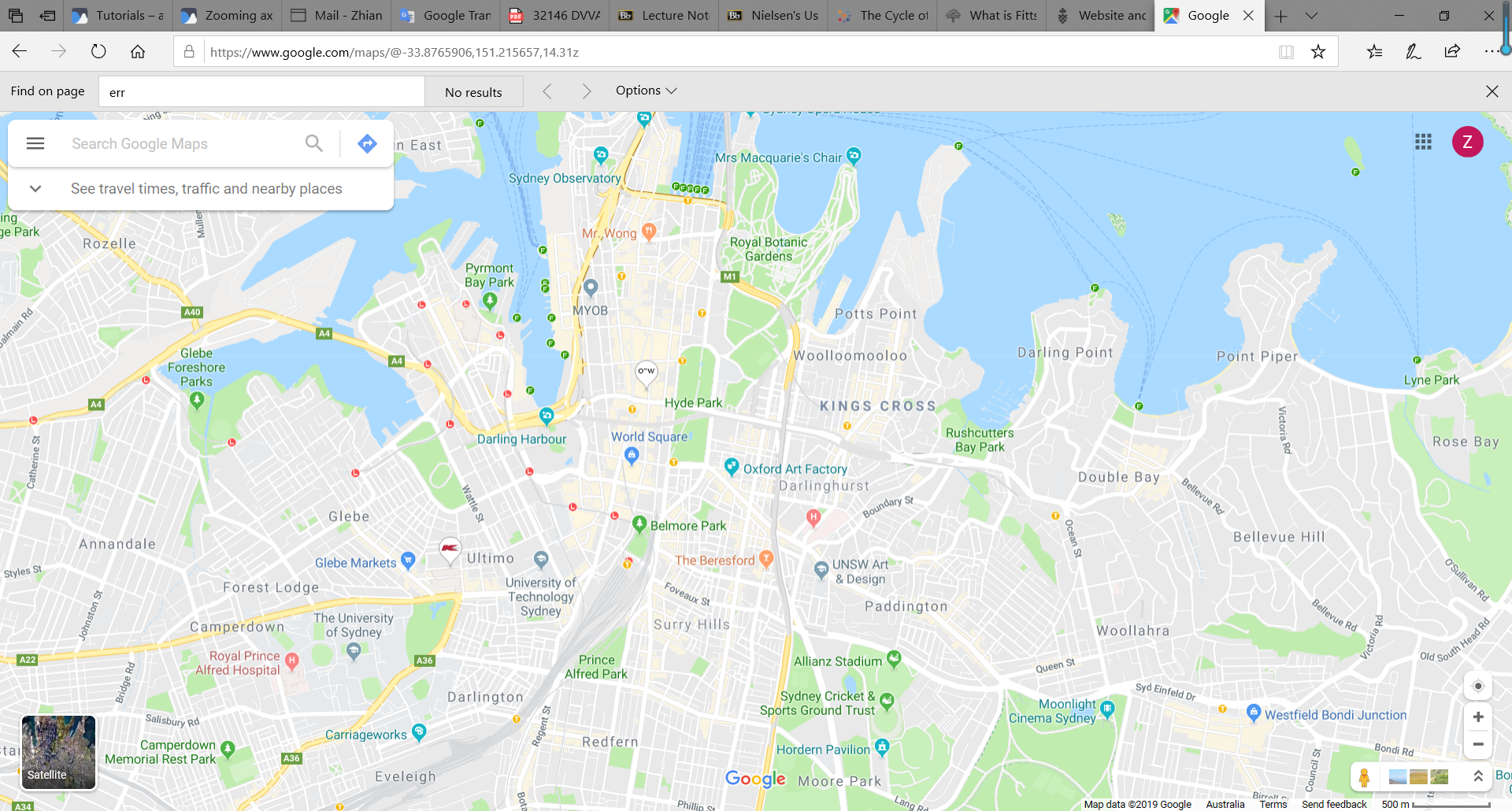
Therefore, for HCI consideration, we estimate our navigation scheme via Fitts’ law. Afterwards, we find that the scale of flights and some small cities which just have 2 flights is too small to interact exactly.

**Nielsen’s Usability Heuristics**

For Nielsen’s Usability Heuristics, our navigation scheme satisfies the following heuristics:

* Match between the system and real world.
* Consistency and standards

In the previous section, we have introduced the satisfaction of heuristic of “Match between the system and real world”. For consistency and standards, our interaction style and the layout for zooming and pan is consistent to the style of Google map. Thus, we keep a external consistency in our graph.



1. The screenshot from Google Map

# Reference

Jianu, R., Rusu, A., Fabian, A.J. & Laidlaw, D.H. 2009, 'A Coloring Solution to the Edge Crossing Problem', pp. 691-6.

Nielsen Norman Group 1998-2019, *10 Heuristics for User Interface Design: Article by Jakob Nielsen*, viewed 3 October 2019, <<https://www.nngroup.com/articles/ten-usability-heuristics/>>.