Max Bowman NEA

Optimised Safe Route Finder

Last updated March 18, 2022
Pages: 16
Written in LaTeX

Contents

Figure 1: Route suggested by google maps

1 Analysis

1.1 Problem Area

There exist a plethora of route finding services for finding the shortest route between two points. These typically optimise for the shortest time taken to travel between two points. Provision for cyclists can be lacking, because it typically consists of an alteration to the existing code for cars, slightly modified to allow for cycle paths and different timings for cyclists.

Many of these route finding applications don't take into account safety considerations. This is shown in Figure 1 which shows the route Google Maps suggests that I cycle to school by. The route includes the A4, which is a very dangerous stretch of road for cyclists as it is a 3 lane road. My plan is to use accident statistics to work out which roads are dangerous and then avoid them. This would be done by imposing a cost for going somewhere that there had been an accident.

1.2 Client

I interviewed Yuvraj Dubey, one of my classmates, to talk about whether they would be interested in a product that attempted to calculate safe routes to cycle.

Do you own a bicycle?
Yes
Do you cycle regularly?
Yes
Do you ever feel in danger while cycling?
Yes
When do you feel in most danger while cycling?
At a Junction near Victoria, where there is a right turn and no cycle lane to do it in. I attempted to go this way once and was almost hit by a bus.
Are you aware of your surroundings while cycling?
Not really
Do you feel aware of the places which to cycle in safely, does this change when you are in places you commonly go?
No, but especially where I don't know where I am because then I do not know where it is safe to cycle. When I cycle home I have learnt a safe route, but if asked to cycle somewhere I did not know I would most likely end up in a dangerous place.
Would you use an application that tells you safe routes to get places?

Yes

Would you be happy using a command line application for this?

Yes

If said application took more than 60 seconds to calculate a route would you lose patience?

Yes, probably

Have you found that cycling directions generated by current products are safe and efficient?

They are definitely efficient, but they often take me onto busy roads and junctions which can be less safe.

From this interview and my own experiences cycling in London I determined that there was an application for my idea, but only if it ran in a reasonable amount of time and was otherwise easy to use, as people struggle to find safe routes themselves, especially when going somewhere they had not previously been, and commercial products such as Google Maps don't generate safe routes.

1.3 Similar Systems

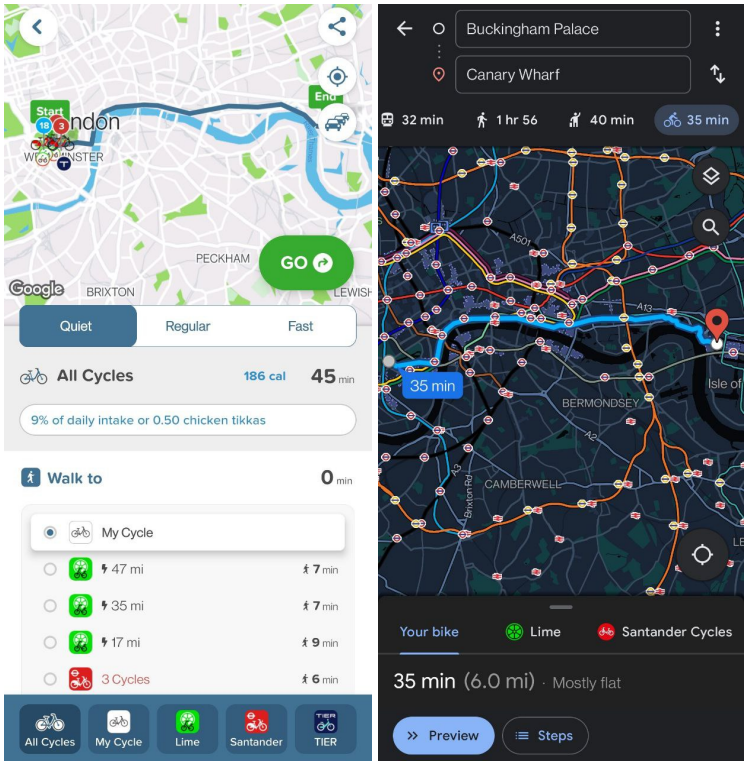

Figure 2: Citymapper and Google Maps respectively, generating the same directions

There exist similar systems for route finding such as Google Maps and Citymapper. These have advantage's over what I will be able to offer such as being able to use live traffic data, and having a good user interface. Citymapper has a option to choose between "Quiet", "Regular", and "Fast" routes, which seems to be based on the type of road you are taken down. However they don't really take accident density into account which will be my aims.

1.4 Features

1.4.1 Map data

I need a data source which can provide data on roads that are legal to cycle on as well as being freely available for me to use. I settled on Open Street Map, a project which combines data gathered by volunteers into one massive freely available map. The map is downloadable in the form of a large XML file or PBF file. A PBF file is just a binary version of the same thing. I will write code to parse one of these myself.

1.4.2 Accident Data

Transport for London has an excellent API which you can download accident data from. The data comes in JSON files and contains information about what type of vehicle was involved in each accident, the severity, and the coordinates of the accident. I need to find a way of mapping the coordinates onto the road network so that the danger of roads can be calculated as accurately as possible. Originally I was looking at adding the accident to the nearest edge in the graph but I

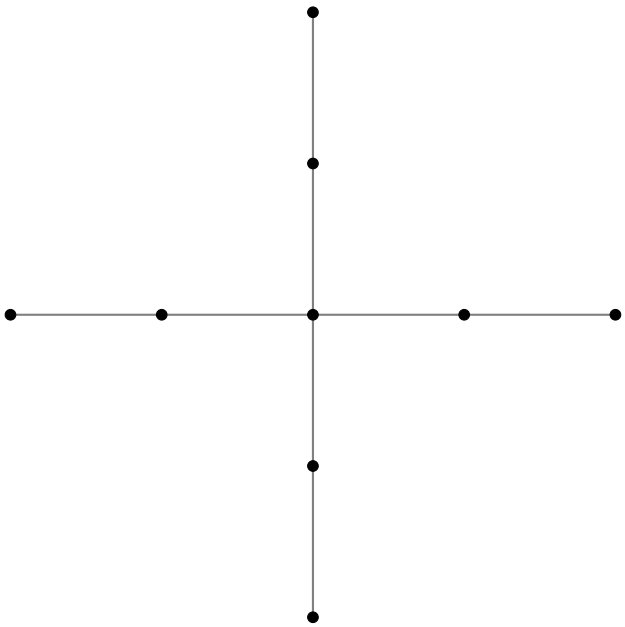

Figure 3: Typical OSM representation of an intersection

decided against this after considering how intersections are typically represented in OSM. Road intersections typically look something like what is represented in Figure 3. If an accident were to occur at the intersection it would end up being added to one of the segments leading into the intersection, so the danger would not be properly calculated if not passing through that segment. Instead I thought about adding the danger to the nearest node, so that the accident would always be counted when a route passes through that intersection. The other problem that I have to deal with, if adding data to the nearest node is that the density of nodes is not constant. Straighter roads will not need to use as many OSM nodes as curved roads, so i might incorrectly add cost to the wrong node. My proposed solution to this is to interpolate in nodes to a very high density to deal with this problem.

1.4.3 Traffic Data

The main problem with this is that the accident data is absolute and can thus not be used to calculate probabilities. For example, more accidents happen on King's Street than the dangerous road I showed earlier, but this doesn't mean that King's Street is more dangerous merely that more cyclists travel on it. This means that I need to get accurate cyclist traffic data for the whole of London in order to turn my accident statistics into accident probabilities. The Department for Transport and the Office for National Statistics both keep data on traffic, but it isn't applicable because cycle data is only given as a total [1] and at specific count points. This means that I will either need to work out traffic data or get it from some dataset, such as Strava's Global Heatmap. Luckily the rest of the application can work without including traffic data so my plan is to deal with this problem at a later time or hope that the avoidance of accidents alone will be enough.

[1] 0.6 billion miles per year in London

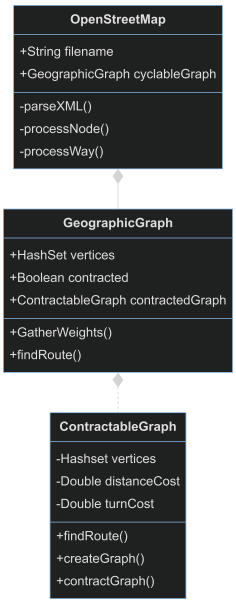

Figure 4: Rough idea of the class diagram I will use

1.4.4 Finding the shortest route

Algorithms for finding the shortest path in a Graph are abundant. The most well known is Dijkstra's algorithm, and it's variant A*. In large Graphs, both Dijkstra's algorithm and A* can be very slow. I will most likely use preprocessing based algorithms such as ALT*/Contraction Hierachies to make my program run a lot faster. This preprocessing will take a long time, but when completed it will significantly reduce the time taken to complete searches.

1.5 Class Layout

As seen in Figure 4, I think I will use 3 major classes which will deal with parsing, storing, and preprocessing Graph data respectively. These will be linked by object composition rather than inheritance. Where possible I will use private fields and methods.

1.6 Critical Path

- Download accidents from the TFL API, parse for accidents that are of interest, and store it in a format that can be easily parsed later
- Import OSM map of London as a Graph which can be easily queried.
- Write code that imports the accident data and connects it to given Graph nodes.
- Write a simple Dijkstra's Algorithm method of finding the shortest path.
- Write a preprocessing based method of finding the shortest path.
- Write a frontend for the input of user options.

1.7 Specification

1. Accident Download System
    - 1.1 The System must be capable of interfacing with the TFL API to download accidents
    - 1.2 The System must be capable of parsing accident data to determine which accidents are relevant
    - 1.3 The System must be capable of storing accident data in a easily accessible file for the route finding algorithm to parse
2. Route Finding System
    - 2.1 The System should be capable of processing an Open Street Map map file into a suitable data structure
    - 2.2 The System should be capable of processing the accident data from the Accident Download System
    - 2.3 The System should be capable of attaching the data from the Accident Download System to the Suitable Data Structure from specification point **2.1**
    - 2.4 The System should be capable of using Accident Data combined with other suitable cost estimation functions to find a directed route between two points that are places on roads in London
    - 2.5 The System should always suggest a route that can be followed while observing all currently known laws of physics
    - 2.6 The System should always suggest a route that can be followed while observing all international, national, local laws and all releveant bylaws
    - 2.7 The System should suggest a safe route wherever possible
    - 2.8 The System should have guards in place for certain types of roads which are deemed too dangerous to consider
    - 2.9 The System should be able to generate this route quickly
    - 2.10 The System should have a mode for preprocessing to make the route generation even quicker
    - 2.11 When using the same parameters, the route generated using a preprocessed graph and associated algorithms should be the same as that generated by the non-preprocessing based approach
    - 2.12 The System should be easy to use
    - 2.13 The System should not crash, and should give appropriate non crashing errors if user data is determined to be bad
    - 2.14 The parameters for the cost estimation functions should be user definable, but sensible defaults should also be defined
    - 2.15 The Route should be output in a format that is easy for the user to follow on a mobile device

1.8 Specification Justification

1. Accident Download System
    - 1.1 This point is required in order to obtain the necessary accident data
    - 1.2 This point is required to reduce the file sizes needed to be packaged with the project, and make it easier to use later
    - 1.3 This point is required as otherwise the system would not do much good in making the Route Finding System simpler
2. Route Finding System
    - 2.1 The suitable data structure is required for all further queries
    - 2.2 The accident data is required for route finding applications
    - 2.3 This is required to tell where the costs for the accidents should be applied in shortest path queries
    - 2.4 This is the main function of the project; if it can't do that it will be worthless
    - 2.5 The Route should be realistic and connected; it can't ask people to teleport to their destination or phase through walls
    - 2.6 The Route should only take people onto roads which are publicly accessible and legal to cycle on, as obviously I do not want to encourage people to break laws
    - 2.7 The Route should attempt to be as safe as possible. Of course in some situations there is only one route which could be unsafe. In that situation that route would be suggested
    - 2.8 Some roads can be very dangerous to cyclists, such as canal paths, and these won't necessarily be represented through the data
    - 2.9 If the route generation takes too long the user will get bored. Other products offering the same features can generate routes very quickly
    - 2.10 London is very large, and route generation may be slow without preprocessing, so the preprocessing mode will help with that
    - 2.11 This makes sure that the preprocessing based approach is not losing information about the best route in any way
    - 2.12 If the system was not easy to use, people would not use it
    - 2.13 This feeds into the previous specification point
    - 2.14 The customisability would allow users to make their own decisions about tradeoffs between time and safety, and sensible defaults should generate sensible routes
    - 2.15 Especially with longer routes, the probability of the user being able to remember the route is slim, so they will need a mobile device to be able to guide them

```
[{
    "id": 0,
    "lat": 0.0,
    "lon": 0.0,
    "location": "string",
    "date": "string",
    "severity": "string",
    "borough": "string",
    "casualties": [{
        "age": 0,
        "class": "string",
        "severity": "string",
        "mode": "string",
        "ageBand": "string"
    }],
    "vehicles": [{
        "type": "string"
    }]
}]

```

Figure 5: The Default output from the API [4]

2 Design

2.1 Accident Download System

As outlined in the Specification, this system should be capable of interfacing with the TFL API, downloading the accidents, deciding which ones are relevant, and storing this in a useful format.

2.1.1 Pulling from the TFL API

The Transport For London API is an excellent API which can provide information on many different aspects of the Transport For London network. The specific API which I used is the AccidentStats API. This API is very simple, you simply request a given year and a JSON object is returned which contains all of the accidents that happened in london that year. These consist of all the accidents that were reported tocdepth the police as happening in that year. Of course, there will be many more accidents than are on the API, but these will mostly be more minor accidents. The Data is returned as a list of accidents, formatted as shown in Figure 5.
As the data about accidents that happened in the past is not going to change any time soon, and TFL only updates this API every year, it is simplest just to download the files once, parse it once, and then use that result in the route finder.
TFL started gathering this data in 2005, and the most recent update was in 2019, so I wrote a simple python script to download all the data from 2005 to 2019 and save it in a subfolder called `accidents`. The program loops through all the years between 2005 and 2019 and fills a JSON file for that year.

2.1.2 Parsing the Data

The next step was to go through the data from all the years, and save all the accidents that were pertinent to my project. As seen in Figure 5, each accident listing has information on casualties, of which there may be many. As this is a cycling application, I only wanted data on accidents where

```
import json
accidents = []
for year in range(2005,2020):
    print(len(accidents), year)
    with open(f"accidents/{year}.json","r") as outfile:
        data = json.load(outfile)
    for x in data:
        if len([person for person in x["casualties"] if person["mode"] == "PedalCycle"
    ]) >= 1:
            accidents.append([x["lat"],x["lon"],x["severity"]])
with open("output.json","w") as outfile:
    json.dump(accidents,outfile)

```

Figure 6: The script I wrote to parse the data

at least one of the casualties was a cyclist. The API provides a lot of data, but all that I decided was relevant was the latitude and longitude, as well as the severity of the incident. All of this data was ultimately saved to a file called `output.json`. The code used for parsing is shown in Figure 6.

2.2 Route Finding System

2.2.1 Overview and Design Choices

I decided to use Kotlin for this project. Kotlin is related to Java in that it allows access to all the Standard Java Libraries, as well as external ones, and it runs in the Java Virtual Machine. Kotlin does not need to maintain backwards compatability with old Java code, so it has a much cleaner API and has nice syntax. As this was a new project not dependent on using Java code Kotlin was an obvious choice. Furthermore, Java code can be easily translated into Kotlin code, so I could code things in Java then translate over if necessary.
As seen in Figure 7, I decided to define 3 main classes for my NEA. `OpenStreetMap` contains methods for parsing OSM files and creating a graph. `OpenStreetMap` contains a single instance of the `GeographicGraph` class by object composition. This is because there will only be one `GeographicGraph` class associated with a given OSM file, and this system could be easily extended to hold information on two separate graph areas at the same time, which could be used if I wanted to extend the system to more cities. The `GeographicGraph` can contain an instance of the `ContractableGraph` class, if it has been contracted. If it has not been contracted, this class will not exist.
I also created two other classes, `IntTuple` and `DoubleTuple`, which are used in priority queues in both the `GeographicGraph` and `ContractableGraph` classes.

2.2.2 Open Street Map Data

Open Street Map is a world map generated by user mapping. It contains multitudes of data on all sorts of mappable things, from the height of stories to the roads that encompass them. Files can be in one of two formats; PBF [2] which is a highly efficient binary format, and XML[3] which is a markdown language. I tried to use a library to parse the PBF file into a Graph, but the main library for that did not work. So instead I decided to parse it manually as an XML file.

Getting the relevant file Openstreetmap has an API for requesting parts of the map, but it does not allow requesting large areas, such as the whole of London. There are many mirrors from which you can download large parts of the world. I decided to use geofabrik [3] because it allows you to download single countries. Next I extracted London from this dataset. This is not strictly necessary, as my program can still deal with large areas such as the whole of the UK, but it does not have accident data for such an area. Furhtermore it would increase the file size needed on disk and increase the running time of any preprocessing. In order to extract London, I used a GeoJSON[1] file, which is essentially a list of coordinates. I found a GeoJSON file on the internet [5] of the M25 boundary, and used the following command[4] to cut out London:

```
$osmium extract -p course_m25_boundary.json united_kingdom.osm.pbf -o london.osm
```

This file could be used by my program, as it intelligently avoids parsing non-routable ways, but I decided to further reduce the file size by removing all nodes and ways that were not part of highways. This was just for quality of life, as it makes parsing the file much faster.

```
$osmium tags-filter london.osm nw/highway -o ways.osm
```

[2] Protocol Buffer Format
[3] Extensible Markup Language
[4] I used the OSM manipulation tool Osmium

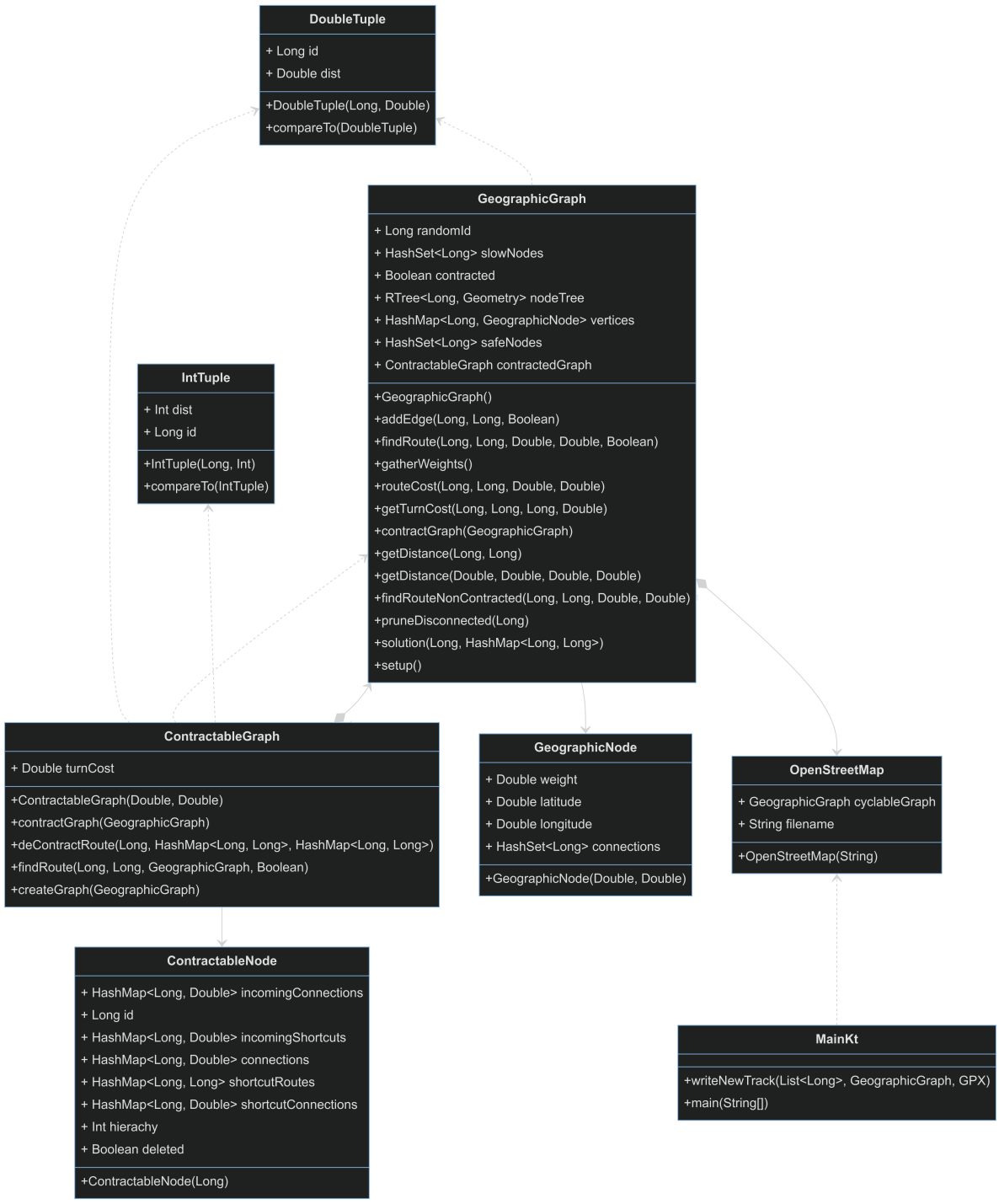

Figure 7: Complete UML diagram for my project

This created a 370MB file called `ways.osm`. I could have reduced this further by removing the ways that are irrelevant to me, but I decided to do that in the parser so that it would be more easily configurable.

Parsing the file Open Street Map files are built out of 3 main elements [6]:

- Nodes, which contain longitude and latitude.
- Ways, which contain between 2 and 2000 ordered nodes. These are used to form roads, but also all kinds of other polygons, such as buildings, rivers, and walls.
- Relations, which contain many ways, nodes and relations. These are used to connect things together, like all roads on a certain bus route. These are not relevant to my project.

```
1         private fun parseXML(filename : String)
2         {
3             val stream = File(filename).inputStream()
4             val saxReader = SAXReader()
5             val cyclableDocument = saxReader.read(stream)
6             val root: Element = cyclableDocument.rootElement
7             val it: Iterator<Element> = root.elementIterator()
8             while (it.hasNext()) {
9                 val element: Element = it.next()
10                when(element.qName.name)
11                {
12                    "node" -> processNode(element)
13                    "way"  -> processWay(element)
14                }
15            }
16        }
17
```

Figure 8: The parseXML function from the OpenStreetMap class

All of these elements can also contain additional information on what type of node or way it is, as well as a unique id. As I only care about nodes that are part of roads I can ignore the relation element completely. In a `.osm` file, the nodes all come first. This means that I can parse the nodes, and then figure out how all the nodes are connected using the ways.

After some research, I determined that the best way to parse the XML was to use the library `dom4j` [2]. The `parseXML` function that I used is shown in Figure 8. Essentially what it does is iterate through the items in the XML file, calling either `processNode` or `processWay`. `processNode` simply adds a new node to the `GeographicGraph`. The `processWay` function is more complex as it has to work out whether ways should be added to the graph, and if additional data needs to be stored about them. Within each way stored in the `.osm` file, there will be additional tags that hold more data about the way [7]. An example way, which is part of the North Circular road, is shown in Figure 9.

First there are a ordered list of nodes which are sequentially connected, then there is a collection of optional Key Value pairs which form tags. The `processWay` function analyzes these tags in order to determine if the road is acceptable for cyclists. The example in Figure 9 would probably not be a good way to include, because the tag `bicycle` is set to `no`, and it is a 3 lane road with a 40mph speed limit. The progam uses accident data to work out which roads are dangerous, but some roads will not have many accidents on them simply because they are so dangerous nobody would ever cycle on them.

```
<way id="1202" version="32" timestamp="2019-05-15T08:24:07Z" uid="7105697" user="
    _Garrison_" changeset="70264933">
    <nd ref="5335693253"/>
    <nd ref="5335693250"/>
    <nd ref="104429"/>
    <nd ref="3330244696"/>
    <nd ref="5335691516"/>
    <tag k="bicycle" v="no"/>
    <tag k="foot" v="no"/>
    <tag k="highway" v="trunk"/>
    <tag k="horse" v="no"/>
    <tag k="lanes" v="3"/>
    <tag k="lit" v="yes"/>
    <tag k="maxspeed" v="40 mph"/>
    <tag k="maxspeed:enforcement" v="average"/>
    <tag k="name" v="North Circular Road"/>
    <tag k="oneway" v="yes"/>
    <tag k="operator" v="Transport for London"/>
    <tag k="ref" v="A406"/>
    <tag k="sidewalk" v="none"/>
    <tag k="surface" v="asphalt"/>
</way>
```

Figure 9: One of the Ways in `ways.osm` extracted

Parsing the Ways In order to convert from XML to a data representation in memory, we simply iterate through all the subelements in the way. If it is a node, the id is appended to a ordered list, and if it is a tag, it is added to a HashMap so it can be more easily queried.

Deciding what is allowed

- Highway tag
  - This tag is present in all roads mapped in OSM, so it's presence must be checked for.
  - This tag represents the "the importance of the highway within the road network as a whole"[7].
  - This does not normally represent anything about the road quality, safety, usage, layout, or maximum speed. The exceptions to this are
    * `motorway` which is obviously not wanted.
    * `living_street` which represents places where pedestrians and cyclists have legal priority, such as Low Traffic Neighbourhoods.
    * `cycleway` which encompasses segregated areas for cyclists.
    * `bridleway` which is a segregated area for horses, where cyclists may be allowed.
    * `footway` which is a footpath which cannot be cycled on.
- Access tag. This tag represents what the access arangements for the area in question are. If it is `no` or `private`, the way would not be wanted.
- Bicycle and Motor tags. These tags are the same as the access tag, but for specific types of vehicle.
- Surface tag. This represents the surface the route is made out of. Ways that are made out of dirt or similar materials will not be parsed.

- Note tag. This can contain lots of different notes, but the one `processWay` looks for is the `towpath`.
- oneway tag. If true, the route will only be connected oneway.
- maxpseed tag. This can be used to exclude dangerous roads.

After analyzing these tags `processWay` has multiple outcomes.

- Way is included in the graph in both directions.
- Way is included in the graph only in the same direction as the ordered list of nodes, because `oneway` is set to `yes`.
- Way is considered too dangerous to include or access to cyclists is not guaranteed to be legal so specification point **2.5** would not be followed.
- Way is included in the graph and the nodes are added to a set of safe nodes because it is in a cycle route or footpath.
- Way is included in the graph and the nodes are added to a set of nodes that will be slower to travel on because of a footpath.

Some of these outcomes can overlap, such as being added to the group of safe nodes and having edges only added in one direction. The code for both `processNode` and `processWay` are shown in Figure . `cyclableGraph` is the instance of `GeographicGraph` contained in `OpenStreetMap`.

References

[1] GeoJSON developers. *GeoJSON website*. URL: https://geojson.org/.

[2] dom4j. *dom4j documentation site*. URL: https://dom4j.github.io.

[3] geofabrik. *Open Street Map mirror*. URL: https://download.geofabrik.de.

[4] Transport For London. *AccidentStats API Documentation Page*. URL: https://api-portal.tfl.gov.uk/api-details#api=AccidentStats&operation=AccidentStats_Get. (accessed: 17.03.2022).

[5] skgrange. *GeoJSON download site*. URL: https://skgrange.github.io/data.html.

[6] Open Street Map Wiki. *Open Street Map Wiki Elements Page*. URL: https://wiki.openstreetmap.org/wiki/Elements. (accessed: 17.03.2022).

[7] Open Street Map Wiki. *Open Street Map Wiki Key:highway Page*. URL: https://wiki.openstreetmap.org/wiki/Key:highway. (accessed: 18.03.2022).