OpenTripPlanner tutorial - creating and querying your own multi-modal route planner

Marcus Young

11 November 2017

Introduction

Data and applications

The datasets and applications used for this project are available from a GitHub repository and can be downloaded in a single zip file from here: https://github.com/marcusyoung/otp-tutorial/data.zip.

Once you have extracted this folder, you should see the following files:

- **otp.jar**, the OpenTripPlanner application, in a single runnable JAR file. This has been compiled from a modified version of the source code with average road speeds and traversal rules that are more appropriate for the UK.
- **fn_otp_api.R**, a set of R functions that can be used to query the OTP API and process the response.
- **router-config.json**, a configuration file to be used with OTP
- **greater-manchester-osm.pbf**, an extract from OpenStreetMap (OSM) covering the Greater Manchester area, obtained from: http://download.geofabrik.de/europe/great-britain/england/greater-manchester.html.
- **rail-gtfs.zip**, a GTFS feed for UK national rail services, limited to operators that provide services in the Greater Manchester area. Based on a full feed (dated 26/06/17) obtained from: http://transitfeeds.com/p/association-of-train-operating-companies/284.
- **tfgm-gtfs.zip**, a GTFS feed for transit services in Greater Manchester (dated 28/04/17), provided by Transport for Greater Manchester (TfGM) and obtained from: http://transitfeeds.com/p/transport-for-greater-manchester/224.
- **aircoach-gtfs.zip**, a GTFS feed for a fictional coach link service between Wigan and Manchester Airport.
- **gm_lsoa_polygons.***, a Shapefile containing polygons for the 1,740 Lower Layer Super Output Areas (LSOA) that cover the Greater Manchester area.
- **gm_lsoa_centroids.csv**, a CSV file containing atttributes and centroid coordinates for the Greater Manchester LSOAs.

Part 1 - Setting up and testing an OTP instance (25 mins)

Create an OTP Graph

An OTP graph specifies every location in the region covered and how to travel between them, and is compiled by OTP using OSM data for the street and path network (used for walk, bicycle and drive modes) and GTFS data for transit scheduling. It can also incorporate a digital terrain model (provided in GeoTIFF format) which is 'draped' over the street network and can be used in routing - for example to request a flatter cycling route.

Our first task is to create the folder and file structure expected by OTP. This requires a base directory called 'otp' which contains the OTP JAR file, and a subdirectory called 'graphs'. Subdirectories created under 'graphs' are known as OTP routers, and contain all the files required to build a graph. A single OTP instance can host several routers - for example covering different regions, or containing alternative transit data for the same region.

Create a router called 'current', and include the GTFS and OSM files for the road network and current transit schedules for Greater Manchester, along with router-config.json, as shown below:

```
/otp
  otp.jar
  /graphs
     /current
         rail-gtfs.zip
         tfgm-gtfs.zip
         greater-manchester-osm.pbf
         router-config.json
```

With the structure created and the files in place, we can now build the graph. We do this by running OTP and specifiying the 'build' option. OTP runs within a Java virtual machine (JVM), which is provided by the Java runtime environment (JRE). Before continuing, run `java -version` from the command prompt or terminal to check that you have version 1.8 of the JVM installed. If you do not have the correct JRE installed you will need to install it for your operating system from: http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html

From the command prompt or terminal, change to the 'otp' directory you created earlier. Then run the following command:

```
java -Xmx2G -jar otp.jar --build graphs/current
```

The java option -Xmx2G allocates up to 2GB of RAM to the JVM. If you do not have this much free RAM available OTP will likely fail during the build process. The --build option tells OTP that we want to build a graph, and ,graphs/current, is the path to the router directory (in this case 'current').

The build process should take 2-3 minutes to complete. Don't worry about the errors that flash up during the process, these mostly relate to railway stations that are located outside of the Greater Manchester area, and there is no road network to link them to. Once complete, you will see a file called 'Graph.obj' in the 'current' directory.

Start up OTP server

The next step is to start up your OTP server, running the router called 'current'. OTP will load the graph you created into memory and you will then be able to plan multi-modal routes using the web interface. From the OTP directory, run the following command:

```
java -Xmx2G -jar otp.jar --router current --graphs graphs --server
```

--router is used to specify the name of the router we want to use (this can be repeated if you want a single instance of OTP to run more than one router); --graphs specifies the location of the graph directory (relative to current directory or a full path); and --server indcates that we want to launch OTP in server mode (rather than the build mode we used earlier). OTP has a built-in web server called Grizzly which runs on port 8080. If you have another application running on your computer that uses this port then you will need specify an alternative port using the --port option, for example:

```
java -Xmx2G -jar otp.jar --router current --graphs graphs --server --port 8888
```

It should only take a minute or two for OTP to load the graph and start the Grizzly server. If all has worked you should now see the message: `Grizzly server running`. You must leave the command prompt/terminal

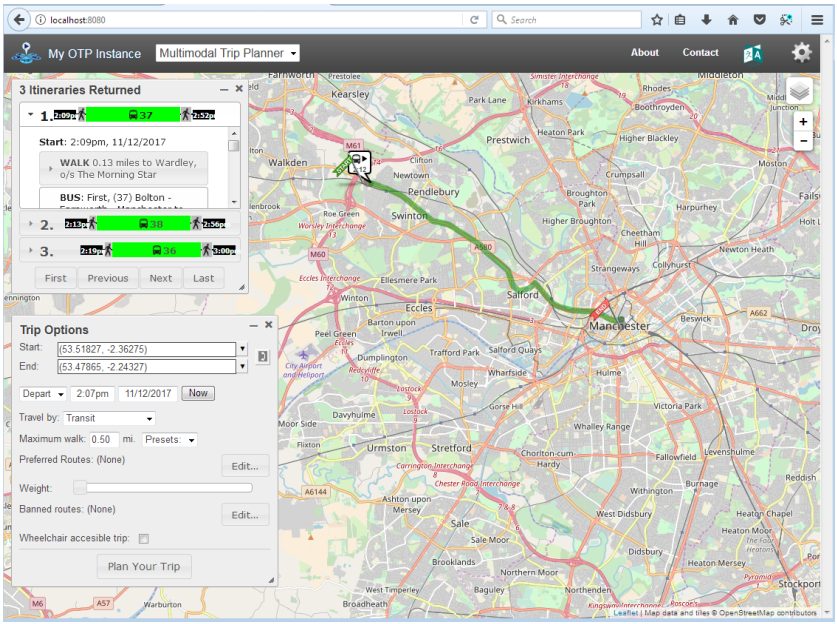

Figure 1: OTP Web GUI

window open. If you close it, the server will stop running. You can stop the server without closing the window by pressing `Ctrl-C`.

You can now access the web interface using the URL: http://localhost:8080. Note: The web interface does not work correctly in Internet Explorer - use Firefox or Chrome. You can now zoom into the Manchester area and request a route by setting an origin and a destination directly on the map (by right clicking your mouse) and you can specify travel dates, times and modes using the 'Trip Options' window (see Figure 1). You can change the background map from the layer stack icon at the top right.

Congratulations, you know your very own multi-modal router planner!

Extras (if you have time)

For troubleshooting routing issues, you can visualise the traversal permissions of street edges, bike safety of edges, and how transit stops are linked to streets. For these additional debug layers to be available, add `?debug_layers=true` to the URL: http://localhost:8080?debug_layers=true. The extra layers will now appear in the layer stack menu. Figure 2 shows the Bike Safety layer (green signifies most safe, red the least safe).

Task 2 - generating travel time isochrones (40 mins)

Now that we have a working instance of OTP, we'll use it to generate travel-time isochrones. We are interested in visualising how long it takes to access Manchester Airport using public transport from different parts of the Greater Manchester area. We will do this by requesting isochrones from OTP for 15, 30, 45, 60, 75 and 90 minutes. This can be acheive with a single query to the 'Isochrone' resource of the OTP API.

To send the query to OTP, you could use your browser to submit a long URL, something like this:

http://localhost:8080/otp/routers/current/isochrone?fromPlace=53.3627432,-2.2729342&mode=WALK,TRANSIT&date=07-10-2017&time=08:00am&maxWalkDistance=1600&walkReluctance=5&minTransferTime=600&cutoffSec=900&cutoffSec=1800&cutoffSec=2700&cutoffSec=3600&cutoffSec=4500&cutoffSec=100000

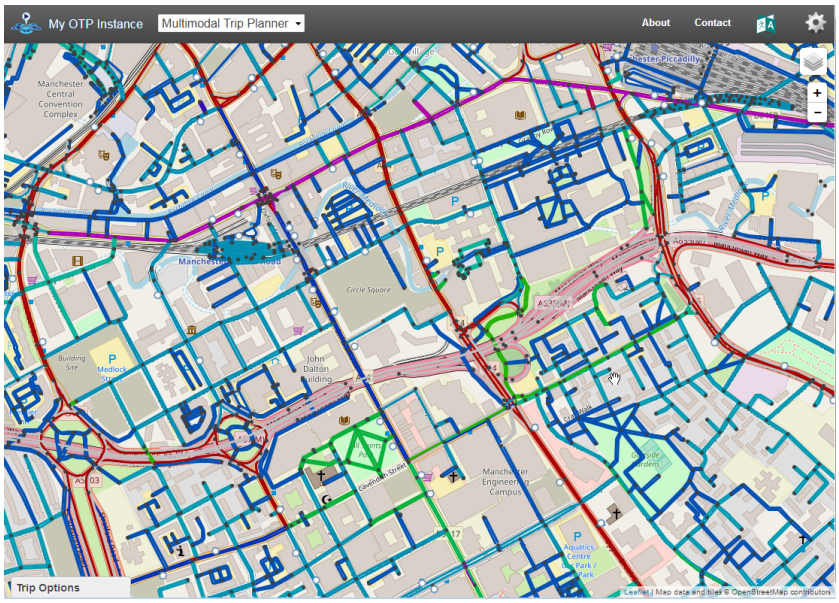

Figure 2: OTP Web GUI with Bike Safety debug layer activated

OTP will then return the isochrone polygons in GeoJSON format, which you can download and then import into a GIS.

A long URL like this will soon get very confusing and prone to errors, so we will use R to construct the URL, submit it to the OTP API, and save the returned GeoJSON object. To do this, we can use the 'GET' method from the 'httr' package. The code you need is shown below. We provide the 'GET' method with the base URL to access the isochrone resource for the router called 'current' and then specify the query parameters to append to the URL. This will submit exactly the same URL as shown above, but it is much easier to see and amend the parameters and their values.

```
library(httr)
airport_current <- GET(
  "http://localhost:8080/otp/routers/current/isochrone",
  query = list(
    fromPlace = "53.3627432,-2.2729342", # latlong of Manchester Airport
    mode = "WALK,TRANSIT", # modes we want the route planner to use
    date = "07-10-2017",
    time= "08:00am",
    maxWalkDistance = 1600, # in metres
    walkReluctance = 5,
    minTransferTime = 600, # in secs (allow 10 minutes)
    cutoffSec = 900,
    cutoffSec = 1800,
    cutoffSec = 2700,
    cutoffSec = 3600,
    cutoffSec = 4500,
    cutoffSec = 5400
  )
)
```

After running the code above, the OTP response will be saved to 'airport_current', and you'll see it in the Environment tab in RStudio. We now just need to convert this to text format and then save the GeoJSON to a file (this will save the file into your Project working directory):

```r
airport_current <- content(airport_current, as = "text", encoding = "UTF-8")

write(airport_current, file = "airport_current.geojson")
```

You can now import the GeoJSON file into QGIS, using 'the 'Add Vector Layer' and then apply a suitable categorized style based on the time attribute. Use the OpenLayers plugin to add a suitable basemap. You should finish up with something similar to Figure 3

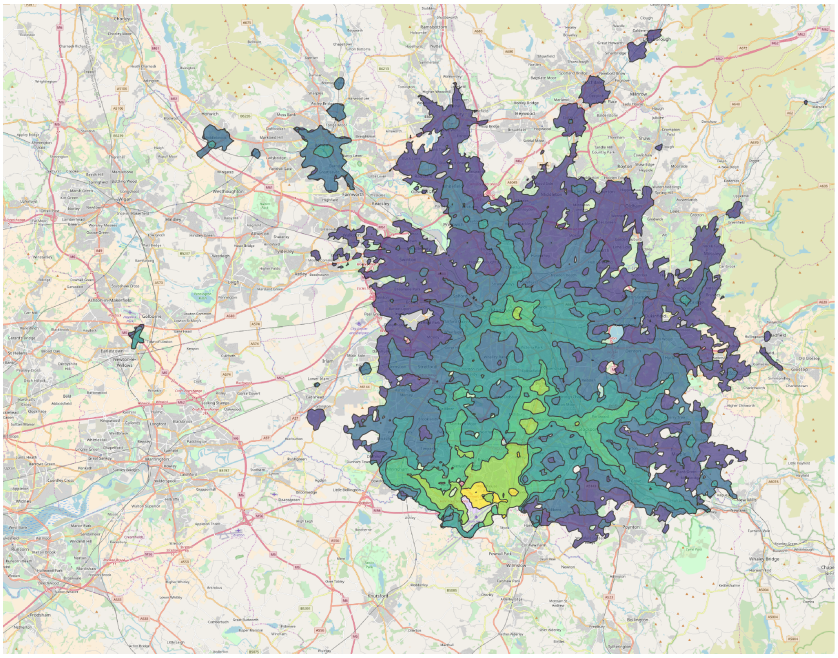

Figure 3: 15-90 minute isochrone for Manchester Airport - based on current services

Challenge: the isochrone map indicates poor accessibility by public transport from the Wigan district. You have been asked to assess the impact of an airport link coach running from Wigan to Manchester Airport every ten minutes between 06:00 and 22:00. You have been provided with a GTFS feed, 'aircoach-gtfs.zip', which contains the information about this proposed service. Create another router called 'new' which incorporates the proposed airport link service. Request an isochrone from the new router and visualise it using QGIS.

Hopefully, your new isochrone will look similar to Figure 4. We can see that the new service has greatly improved accessibility to the airport by public transport from the Wigan district.

This simple example illustrates the major advantage of running your own multi-modal route planner, compared to services such as Google Maps or TransportAPI. You can carry out analysis using amended transport data. This could be changes to transit shcedules, as in this example, or changes to the underlying street network. By editing an offline copy of OSM, you could model the effects of creating new roads, closing roads, or imposing other restrictions. You can also look back in time. For example, you might want to examine the effect of reductions in rural bus services on accessibility to health facilities. To do this you will need a network with bus schedules as they were x years ago.

Task 3 - automating OTP API lookups using R (40 mins)

In this final task we are going to see how we can automate querying the OTP API to generate large quantities of route data - potentially for many thousands of origin:destination pairs. In this example, we will gather

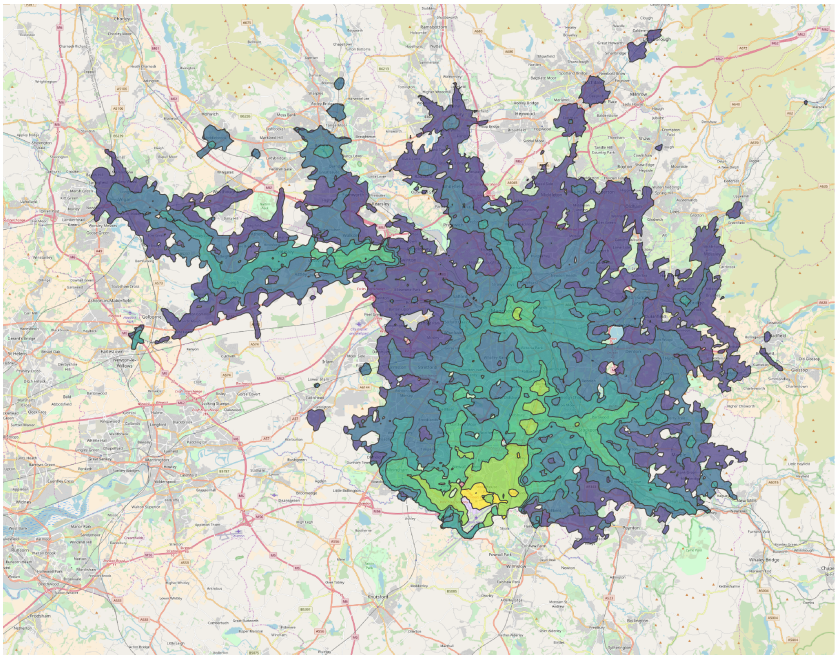

Figure 4: 15-90 minute isochrone for Manchester Airport - with new Wigan Air Coach

data on accessing Manchester Airport using public transport for each of the Lower Super Output Areas (LSOAs) in Greater Manchester.

We will start by importing a CSV file containing the LSOA data into an R dataframe:

```
# Import the LSOA CSV file
gm_lsoa_centroids <- read.csv("data/gm_lsoa_centroids.csv", stringsAsFactors = FALSE)
```

We next need to load some R functions that can be used to query the OTP API and parse the XML response. These functions are not yet part of an R package, so we just read the file, 'fn_otp_api.R', using the 'source' function:

```
# Load the OTP API functions
source("data/fn_otp_api.R")
```

We are going to use two functions. The first is otpConnect(), which defines the OTP instance and router we want to query: The second is otpTripTime(), which retrieves an itinerary between the requested origin and destination.

```
# Call otpConnect() to define a connection called otpcon
otpcon <-
  otpConnect(
    hostname = "localhost",
    router = "current",
    port = "8080",
    ssl = "false"
  )
```

The second is otpTripTime(), which retrieves an itinerary between the requested origin and destination. In this example we use the first LSOA in gm_lsoa_centroids, which has coordinates '53.43329,-2.13357':

```
# Call otpTripTime to get attributes of an itinerary
   otpTripTime(
```

```
    otpcon,
    from = '53.43329,-2.13357',
    to =   '53.36274,-2.27293',
    modes = 'TRANSIT,WALK',
    detail = TRUE,
    date = '2017-07-12',
    time = '08:00am',
    maxWalkDistance = '1600',
    walkReluctance = '5',
    minTransferTime = '600'
  )
```

```
## $errorId
## [1] "OK"
##
## $itineraries
##                 start                 end duration walkTime transitTime
## 1 2017-07-12 08:16:23 2017-07-12 09:05:53     49.5     5.47          34
##   waitingTime transfers
## 1       10.03         1
```

A list is returned. The first item in the list contains the errorId, and the second is a dataframe, containing the start and end date/time, journey duration (minutes), time spent walking, time on public transport, waiting time, and number of transfers.

To find and store this information for every LSOA centroid, we can use a R loop. The code below deals with each row in gm_lsoa_centroids in turn. For each LSOA a query is submitted to OTP and the trip attributes returned are stored in the gm_lsoa_centroids dataframe. As this will take a few minutes to complete, a progress indicator has been included. This will display in the R console. It requires the 'pprogress' package to be installed and loaded.

```
install.packages("progress")
library(progress)

total <- nrow(gm_lsoa_centroids) # set number of records
pb <- progress_bar$new(total = total, format = "(:spin) [:bar] :percent") #progress bar

# Begin the loop
for (i in 1:total) {
  pb$tick()   # update progress bar

  response <-
    otpTripTime(
      otpcon,
      from = gm_lsoa_centroids[i, ]$latlong,
      to = '53.36274,-2.27293',
      modes = 'WALK,TRANSIT',
      detail = TRUE,
      date = '2017-07-12',
      time = '08:00am',
      maxWalkDistance <- "1600", # allows 800m at both ends of bus journey
      walkReluctance <- "5",
      minTransferTime <- "600"
    )
  # If response is OK update dataframe
```

```
  if (response$errorId == "OK") {
    gm_lsoa_centroids[i, "status"] <- response$errorId
    gm_lsoa_centroids[i, "duration"] <- response$itineraries$duration
    gm_lsoa_centroids[i, "waitingtime"] <- response$itineraries$waitingTime
    gm_lsoa_centroids[i, "transfers"] <-response$itineraries$transfers
  } else {
    # record error
    gm_lsoa_centroids[i, "status"] <- response$errorId
  }
}
```

It will take a few minutes to complete the lookup for all the LSOAs. Once complete you can view the gm_lsoa_centroids dataframe, and see the populated data.

```
       code easting northing           latlong status duration waitingtime transfers
1 E01005756  391223   392954 53.43329,-2.13357     OK    44.20        0.03         1
2 E01005757  390660   391186 53.41739,-2.14199     OK    35.28        0.03         1
3 E01005754  390870   392662 53.43066,-2.13888     OK    47.23        0.03         1
4 E01005755  391140   391965 53.42440,-2.13479     OK    34.23        0.03         1
5 E01005752  393251   391229 53.41782,-2.10300     OK    63.52        7.03         1
6 E01005753  391339   392429 53.42858,-2.13181     OK    43.67        5.03         1
```

Extras (if you have time)

Save the dataframe gm_lsoa_centroids to a CSV file (use 'write.csv') and add it into QGIS using the 'Add delimited text layer' tool, and using easting and northing columns as the coordinates. A shapefile for the LSOA polygons is provided in the data folder. Try joining the data from the centroids to the shapefile and then applying a style to the data (hint: go to the properties of the layer and to the 'Joins' section). Explore how duration, waiting time and transfers vary across the LSOAs You should end up with something similar to Figure 5

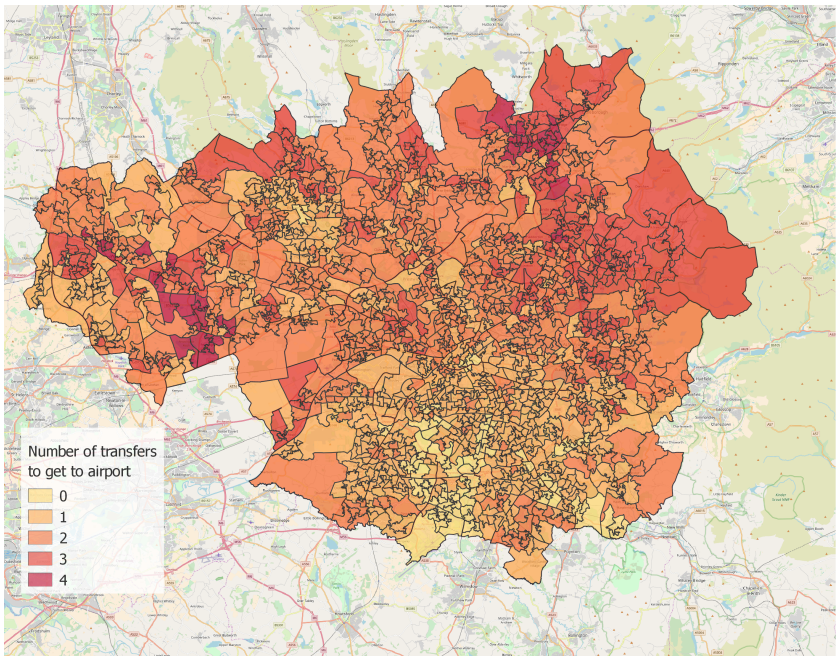

Figure 5: Number of transfers required to get to Manchester Airport by public transport, for each LSOA

Additional resources

links to API docs link to OTP docs link to OTP on Github link to onebuswaytransformer link to OSM editor link to GTFS editor (?)