Table of Contents

[Exercise 5 2](#_Toc89559047)

[Analysis of reachability by public transport: identifying missing links 2](#_Toc89559048)

[Predicting delay of buses 4](#_Toc89559049)

[Difficulty 6](#_Toc89559050)

[Exercise 6 6](#_Toc89559051)

[Choice of Dataset 6](#_Toc89559052)

[Task 9](#_Toc89559053)

[Usefulness 9](#_Toc89559054)

[Planning the transport once predictions are available 9](#_Toc89559055)

[Preparing the Data 10](#_Toc89559056)

[Difficulty 11](#_Toc89559057)

# Exercise 5

In this exercise, we discuss two different possible uses of Spark GraphX and Spark Spark MLlib.

## Analysis of reachability by public transport: identifying missing links

With the threat of climate change, it is essential that cities move away from a personal automobile dominated plan to one where cities rely mostly on public transport. This also has benefits from the social justice point of view. Cars are expensive and not everyone can afford a car. If inability to afford a car affects accessibility of places of work and education, this has far reaching economic consequences for the individual as well as the society and the country at a large.

Ideally, a bus service for a city should be planned in such a way that people can confidently rely on it enough to avoid buying a car altogether. The environmental cost of a car comes not just from running it but also in producing it.

Trains and buses are, by far, the most economical modes of public transport as they can transport a large number of people at a very low cost. Trains are more economical and efficient than buses, however, running trains to places with low population density may not be the best approach. A combination of buses, trains and trams are an ideal solution to a transportation problem.

Research in developing countries have also shown that planning the last mile connectivity is essential for an effective public transport system. Buses form the ideal mode of transport for last mile connectivity.

The public transport infrastructure in Ireland is far from adequate, and while it is not quite as bad as in the USA, it leaves a lot to desire. In many places there are no bus routes, and where there are bus routes, there may not a good frequency of buses. So much so, that given two locations, it may not be practically possible to travel between the two with public transport because of the presence of missing links.

One exercise that may be done is to predict missing links in bus connectivity which prevent people from using public transport.

The assumptions here are these:

1. Commuters must be able to travel from any bus station in the city to any other bus station in the city.
2. Commuters may have to walk up-to a 500 metres to catch a connecting bus, but no more than that.
3. This must be true for the entire day 24/7

This knowledge can be used by the city planners to add or modify bus routes so that places where either 1 or 2 are invalid also come under the bus transport system. This can be done by adding new bus lines altogether, or by modifying existing bus routes to also serve some new stations along the route.

The third assumption is important here that any two places must be connected by the bus route 24/7. It is often the case that many bus routes reduce their frequency at night or completely stop plying. While the demand during those hours may be low, the unavailability of buses would mean that people cannot rely on them and are forced to buy personal vehicles, thereby incurring all the production cost of all the cars.

The problem essentially is a form of connected components. The idea is to find out subgraphs that are unconnected, and then connect them by changing the bus routes.

Both GraphX and GraphFrames facilities for finding connected components. However, we will prefer GraphFrames as it is supported on both Scala and Python, and will not tie us down to one language only. This is important as availability of talent is one important reason to prefer one one programming language over another, and python talent is easy to find as the language is very popular.

The steps to do so will be as follows:

1. Find a list of all stations by looking at the entire dataset in with regular spark RDDs. This can be found by looking at entries where atStop=1. Cache this set of stations for later use. We need to do this step separately because if we take a certain day some buses may not be running and serving some stations, and then we’ll miss those stations entirely. However, it is safe to assume that if a station is active, it will have some bus serving it in the entire dataset, and even if one bus serves it we will have made a note of it.
2. For every hour interval for both weekdays and weekends do the following:
   1. Add all stations as nodes.
   2. Take all physical buses
      1. If a bus travels to two stations in the same line, then add an edge between them.
   3. Take all pairs of stations (regardless of whether there is a bus between them or not).
      1. If the haversine distance between the two stations is less than 500m, then add an edge between them. This is because we have the assumption that commuters may have to walk upto 500 metres to catch a connecting bus. The haversine distance is not accurate as people will not walk as the crow flies, but we don’t have any other information in the dataset that will allow us to get the actual on-fot distance, so we will use this as a proxy.
   4. Find connected components using Graph Frames, and then print all the connected sub-graphs. Also, store all sub graph information with the nodes we had cached earlier. Now that we have all the nodes labelled with the connected sub-graph, we can move to the next phase of this algorithm.

The next value addition would be to suggest stations which may be connected easily with the least cost. Two unconnected sub-graphs may be connected by adding just one edge, although adding more edges will be beneficial.

In this context, it will make sense to add edges between the closest points. Once we have the connected components labelled, this can be done using regular Spark RDDs.

1. First a join must be performed between rows which have different sub-graphs. A join may be expensive, but it is expected that the number of stations in a city will still be limited, and the memory requirements will not explode. The next step is to calculate the haversine distance between all such pairs of stations belonging to different sub-groups.
2. The next step is to group by both sub-graph ids, and then sort in ascending order.
3. The last step would be to report the top N connections that can join two sub-graphs.

## Predicting delay of buses

If delays can be reasonably predicted ahead of time, this might give the bus transport providers a chance to readjust their fleet and use their resources more effectively. This might also give the providers a chance to alert their riders of possible delays. The riders may the adjust their schedules accordingly.

Some researchers have modelled traffic as a second order chaotic system.[[1]](#footnote-1) Chaotic systems may be of the first order or the second order. A first order chaotic system is one in which any prediction made about the system doesn’t affect the outcome of the system itself. Predicting the weather is an example of a first order chaotic system. Any prediction about the weather will not change the weather. By contrast, a second order chaotic system is one in which any prediction about the system is likely to change the system itself. The stock, for example, market is a second order chaotic system. Predictions about the stock market influence people’s behaviours in buying and selling stocks, and that in turn affects the market itself. Traffic can also be seen as a second order chaotic system. Predicting traffic accurately can have the effect of changing people’s behaviour to avoid peak traffic hours and routes, which will end up reducing traffic on a whole thereby having a positive effect. Therefore, it is easy to see how predicting delays and congestion can be useful.

Congestion and delays may also be highly correlated with weather. For example, people may make a rush to reach home before a storm hits, and that may result in excess traffic and congestion before bad weather. Hence there might be value in correlating other datasets that have weather data and augmenting the Dublin bus dataset with the weather information.

While considering weather events, we will not make any attempt to address natural calamities or very severe storms that cause widespread damage or flooding. This is for two reasons:

1. These are rare events, and there may not be enough data to get any meaningful signal
2. Focusing on regular weather events will make the system more useful, as these are more frequent than rare destructive stormy evens

Met Eirann provides datasets with hourly weather measurements. These datasets include precipitation and wind-speed. The steps to deal with this would be the following

1. Load the bus-eirann dataset and enrich it
   1. For every station and route, group by hour, and reduce such that if any bus reported congestion, mark that (route, hour, station) combination as congested, and if any bus reported a delay, then mark that (route, hour, station) combination as delayed
   2. Traffic patterns on weekdays and weekends may be different. Also traffic patterns on different times of the day will be different. The nature of this is essentially periodic. The periodicity can be effectively captured by taking sines or cosines. The information can be captured by adding the following columns.
      1. Take the day of the year, and convert it to a cosine
      2. Take the month of day and convert it to a cosine
      3. Take the day of week and convert it to a cosine
      4. Take the minute of day and convert it to a cosine
      5. Take the hour of day and convert it to a cosine
2. Load the Met-Eirann dataset
   1. For every hour, populate the following features, the Met-Eirann dataset will surely allow us to do that
      1. Convert the wind speed to n-categorical numbers, severe, high etc.
      2. Convert rainfall to n-categorical numbers – severe, high, etc.
3. Join the enriched bus-eirann dataset, and the Met-Eirann dataset
   1. The join will be performed on both time and geolocation
      1. The condition for joining should include weather data 2 hours before and after every bus eirann measurement. This is because weather before and after the current time affects the traffic flow. For example, people may be moving more in anticipation of bad weather.
      2. Anothe r condition would be that the geolocation of the prediction. We should take the MetEirann data from the closest station to the location of the vehicle. This can be accomplished group and reduce after join such that:
         1. Add a new column for every row of the join that captures the distance between the met-eirann station and the bus location
         2. Reduce to take the minimum distance between the station and any bus Met station.
   2. Group by each vehicleID and timestamp, and add the following features
      1. Precipitation in the same hour as the timestamp
      2. Precipitation in the past hour, two hours before, next hour and two hours later
      3. Wind speed in the past hour, two hours before, next hour and two hours later
   3. Finally experiment with various classifiers from mllib.classification to see which classifier behaves the best to first form a baseline, and then tune the hyperparameters with the best classifiers.
      1. A multi-class prediction to predict both congestion and delay can be used

Spark MLLib is dataframe based and is the ideal choice as Spark ML is not under active development. The plan from the Spark community is to first enhance MLLib to achieve parity with Spark ML and then continue developing MLLib further and add no further features to Spark ML. Choosing the platform that has a clear roadmap will be advantageous in terms of maintenance and support, and also to future proof the solution.

## Difficulty

This new use-case is not necessarily more difficult than the other questions attempted but has more steps in the process. Also, since two different datasets must be combined, there is some added complexity. However, all of these steps can be easily done in Spark, and are not necessarily difficult, but may be time consuming just because there are more steps involved.

# Exercise 6

## Choice of Dataset

The dataset used in this case is the DublinBikes DCC dataset.[[2]](#footnote-2) In addition, this will be combined with the met-Eirann dataset[[3]](#footnote-3) because biking demand varies with weather, and people prefer not to bike when the weather is bad.

The DublinBikes dataset has the following structure:

Table

Description automatically generated

Figure 1 screenshot from <https://data.gov.ie/dataset/dublinbikes-api/resource/76fdda3d-d8be-441b-92dd-0ee36d9c5316?inner_span=True>

This dataset is mostly in the form of csv files, one for each quarter. The files are very large, each around 300MB in size and each having about 3 million rows. The rows are arranged as such:

1. First they are ordered by the day when the readings are gathered
2. Then they are ordered by the station from which they are gathered

There is also a REST API that can be used to query the data instead of downloading large CSV files.[[4]](#footnote-4) The API can fetch the last snapshot or the historical data. The details of querying these are pasted below from the website:

Graphical user interface, text, application

Description automatically generated

Figure 2 Screenshot of API from <https://data.smartdublin.ie/dublinbikes-api>

Graphical user interface, text, application, email

Description automatically generated

Figure 3 Screenshot of API from <https://data.smartdublin.ie/dublinbikes-api>

However, when I tried it out, I found that fetching of historical data was unexceptionally slow using the REST API. It might be a better idea to fetch historical data by downloading CSV files.

For example, the historical data for 2021-04-13 17:00:00 can be fetched by the following URL:

<https://data.smartdublin.ie/dublinbikes-api/historical/?init=2021-04-13%2017%3A00%3A00>

or with the following Curl command:

curl -X 'GET' \

'https://data.smartdublin.ie/dublinbikes-api/historical/?init=2021-04-13%2017%3A00%3A00' \

-H 'accept: application/json'

The Met Eirann dataset has the following fields:

date: - Date and Time (utc)

rain: - Precipitation Amount (mm)

temp: - Air Temperature (C)

wetb: - Wet Bulb Temperature (C)

dewpt: - Dew Point Temperature (C)

vappr: - Vapour Pressure (hPa)

rhum: - Relative Humidity (%)

msl: - Mean Sea Level Pressure (hPa)

ind: - Indicator

## Task

The task here would be to predict the number of available bikes at any station.

## Usefulness

The dataset here does not give any data about how many bikes were actually rented out at any station in a given hour. Hence, what we will use is the number of bikes at any station at a given hour. The idea is that if a station has no bikes, then there more demand at that station than there is supply, and bikes from other stations must be transported to that station.[[5]](#footnote-5)

Being able to predict that some bicycles will be out of bikes soon will help make better decisions and streamline logistics for bicycle transport between stations.

## Planning the transport once predictions are available

We will first discuss how predictions can be used, before describing how we will go about predicting. Once a prediction about the number of bikes available at each station is available, for a future hour in the day, this can be used to optimally transfer bikes.

If there are three stations, A, B and C, and A and B have 5 and 6 bikes each, and C has 0 bikes, then bikes could be transferred from A to C or B to C, or a both from A and B to C. Each of them would fulfil C’s demand, but will have a different cost based on based on fuel required and time taken to transport.

Linear programming, or other optimization methods can be used to find an optimal bike transfer strategy to reduce time, costs and fuel.

## Preparing the Data

To prepare the data Spark SQL can be used. The following steps must be performed

1. Join the weather data with the DublinBikes data
   1. The weather data is per weather station. We only consider weather stations that are within a 10 kilometres of Dublin city centre.
   2. The weather data is on an hourly basis, so nothing more needs to be done there.
   3. Next we select all bike stations so that we can assign each bike station to its closest weather. This can be done by discarding other fields and only having the latitude and longitude of the bike stations.
   4. The assignment will be done by a full join, followed by a grouping to and aggregated by the minimum haversine distance.
   5. Now that we have all the bike stations assigned to a weather station in a separate table, we perform a 3-way join between the full bike data, the weather reading and the new table. This will prevent the memory from exploding, as a full join will not be necessary and we can only do an inner join.
   6. This step is optional, what might help is getting weather readings for an hour before and an hour earlier. To achieve this the following modifications to the above steps must be done:
      1. The inner join should be modified so that readings within an hour’s difference will also be included. This would mean that for every bike reading, there will be multiple rows with weather readings from the closest weather station.
      2. This can be aggregated by the timestamp of the bike reading, and the bike station id where the weather readings can be converted to a list.
      3. Finally the list can be converted to columns, so that each bike reading has three columns for weather – one for one hour earlier, one for the current weather, and one for one hour later.
   7. The bike station data is gathered every 5 minutes. We can either take this same window and predict for it, or we can predict for every 1 hour window. If the latter is chosen, then we will have to combine the entries for every bike station by the hour. This can be done by aggregating by the hour and bike station, and then taking the minimum number of bikes present in the station at any point of time.
2. Prepare the data
   1. The rainfall should be converted to categorical numbers
   2. Same must be done for wind
   3. The same must be done for temperature
   4. Demands are periodic in nature, for example demand on weekdays may follow a certain pattern and on weekdays a different pattern, and similarly for different hours of the day. To capture this the following must be done:
      1. Timestamp must be converted to day of year, and the sine or cosine of the value taken
      2. Time stamp must be converted to day of week and the sine or cosine of the value taken
      3. Timestamp must be converted to hour of day and the sine and cosine must be taken
      4. The three above columns will be added to the dataset
3. Now using the above enriched data we have the following columns added:
   1. Weather data for one hour prior
   2. Weather data for one hour post
   3. Current weather data
   4. Timestamps expressed as cosines or sines of day of year, day of week, hour of day
4. Following this Spark MLlib will be used to train a model that predicts the Available Bikes at each station. A variety of classifiers should be used and initial performance estimated. The best performing models must be fine-tuned for the best hyperparameters.
5. Once we have the trained model, at serving time it needs to predict the number of bikes. To do this, it needs the current state of the bike station, the number of bikes etc. It also needs the weather readings for the current time and the last hour, as well as the prediction for the next hour. The same conversions must be done on this data as was done to the training data, and then this can be used for prediction. Spark Streaming can be used at serving time.

## Difficulty

Again, the it is not necessarily more difficult than other problems we have done before, but there are more steps involved, and also there is an additional task of combining two different datasets. All of this can be effectively performed in Spark, but may be time consuming because there are more steps involved.

1. Disbro, J. E., & Frame, M. (1989). *Traffic flow theory and chaotic behavior* (No. Special report 91). New York (State). Dept. of Transportation. [↑](#footnote-ref-1)
2. URL for Dublin Bike dataset: <https://data.gov.ie/dataset/dublinbikes-api?package_type=dataset> [↑](#footnote-ref-2)
3. URL for Met Eirann weather data: <https://www.met.ie//climate/available-data/historical-data> [↑](#footnote-ref-3)
4. Dublin Bike DCC REST API: <https://data.smartdublin.ie/dublinbikes-api> [↑](#footnote-ref-4)
5. One improvement to the dataset in further collections could be to also have a cyclist-in and cyclist-out data associated with the same dataset. This data is available in another dataset, the Cycle Counter Dublin Dataset , however, the stations in these two datasets are different, so the data cannot be joined. This could be a suggestion to the transport bodies that collect this data.  
   <https://data.gov.ie/dataset/dublin-city-centre-cycle-counts/resource/ce0b8fa0-58ad-4b75-941c-93be56b68b96?inner_span=True> [↑](#footnote-ref-5)