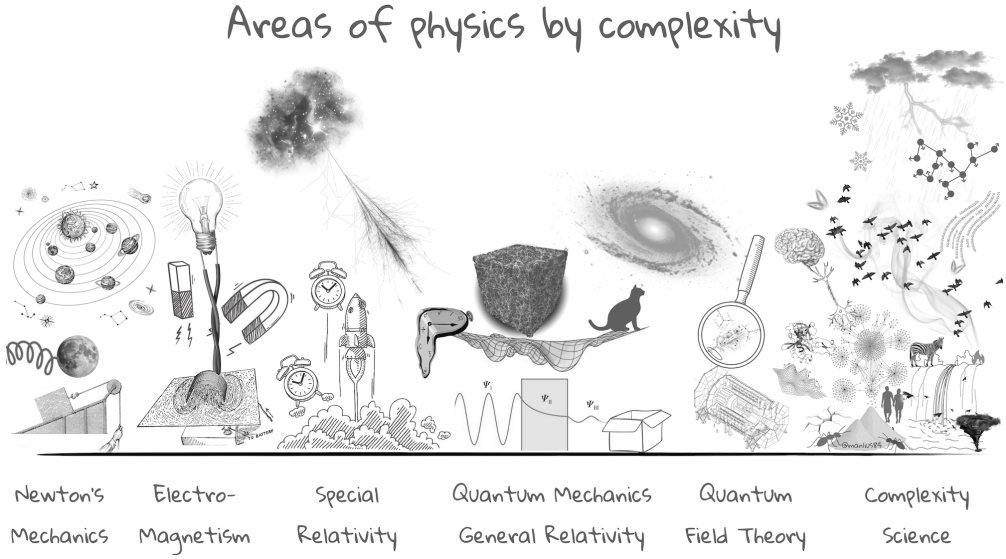
Final Report

Physics of Complex Networks: Structure and Dynamics

Università degli Studi di Padova



Project # 46: European transportation networks II

Zara, Miriam

Last update: June 1, 2024

Contents

1 | Edges and Nodes Extraction

The goal of this project is to extract data about european railways and railway stations and construct a network.
The dataset we used is called 'EuroGlobalMap (EGM)' and is published under an open license by EuroGeographics, the not-for-profit membership association for European National Mapping, Cadastral and Land Registration Authorities (NMCAs), in partnership with the National Geographic Institute (NGI) Belgium. The newest data is avaiable at link https://www.mapsforeurope.org/datasets/euro-global-map. For this analysis, we refer to EGM 2019, released in March 2019.
The task is to extract the data and build a rail network for each EU country separately (only countries with IS03 code starting from IT, onwards) and for whole EU. The output consists of two files for each country: one for nodes, which shall contain the columns (nodeID, nodeLabel, latitude, longitude, country_name, country_ISO3) and one for edges list, with columns (nodeID_from, nodeID_to, node_label_from, node_label_to). Node indexing starts from 1. This section is organized as follows:

- In Paragraph 1.1 we describe the method we used to read the raw data provided in the EGM19 database and present their preliminary visualization.
- In Paragraph 1.2, we highlight some crucial limitations of the data and detail the preprocessing techniques utilized to partially address these issues.
- In Paragraph 1.3, we outline our method for creating the final output files for nodes and edges.

1.1 | Raw data reading

The database 'EGM19' is provided with two useful documentation and metadata files, namely 'EGM19_User_Guide.pdf' and 'EGM19_DataSpecification.pdf'.
We consulted the "EGM19_User_Guide.pdf" (Annex C) to get a dictionary of correspondences between country IS03 codes and country names. We consulted "EGM19_DataSpecification.pdf" (Annex C: Definition of Features and Attributes) to gain information about the raw data.
The raw data is provided in various formats, including the .shp (shapefile) format. We opted for the latter, which we read using the Python library **GeoPandas**. The dataset EGM19 is broad, but the only raw files needed to accomplish our project are ("RailrdC.shp", "RailrdC.shx", "RailrdC.dbf"), containing railway stations in whole Europe, and ("RailrdL.shp", "RailrdL.shx", "RailrdL.dbf"), containing railway lines in whole Europe. The basic command to open a shapefile with GeoPandas is

stations = gpd.read_file("RailrdC.shp")
railways = gpd.read_file("RailrdL.shp")

This command creates a GeoPandaDataFrame object. We select from it only the relevant attributes columns, which are, for stations:

- 'ICC': the 2- character country IS03 code (es. IT, for Italy)
- 'NAMA1' : the station name in first national language, written in the international alphabet (ex: "Ancona")

And for railway lines:

- 'ICC', same as before
- 'NAMA1' , same as before (ex: "Ancona-Lecce", "Circumvesuviana")
- 'EXS' : existence cathegory (0: 'Unknown', 5: 'Under Construction', 6: 'Abandoned/Disused', 28: 'Operational', -32768 : 'Invalid Value')

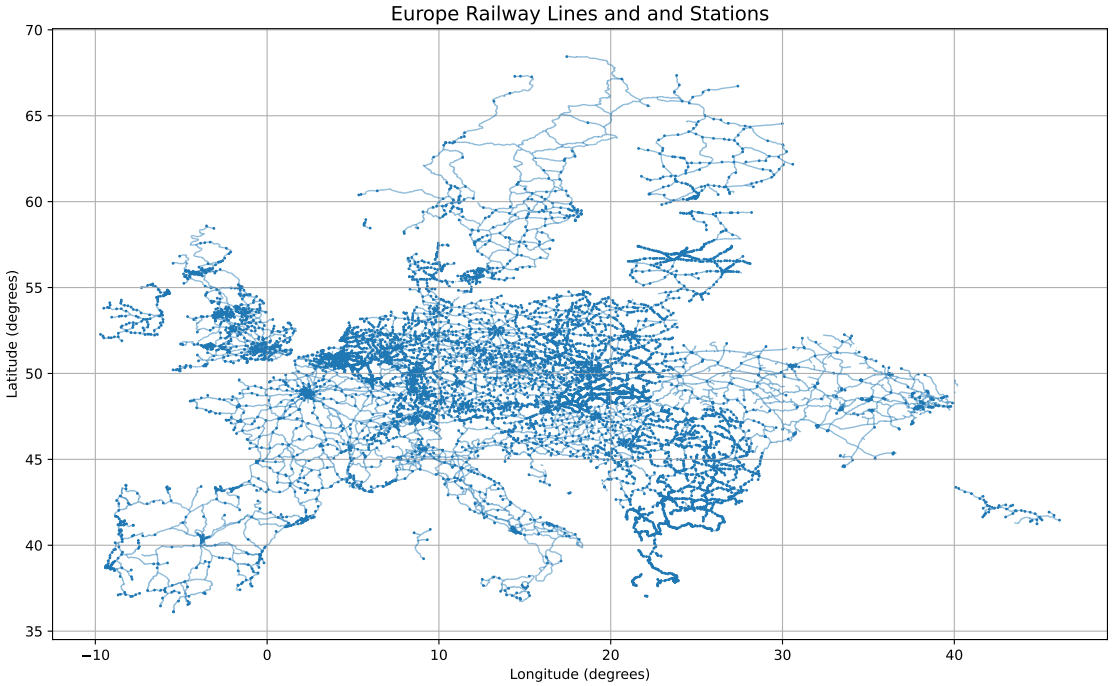


Figure 1.1: Railways and stations of the whole Europe. Each dot (line, respectively) is an element of the dataframe stations (railways, respectively). Only operational lines (EXS code = 28) have been filtered out.

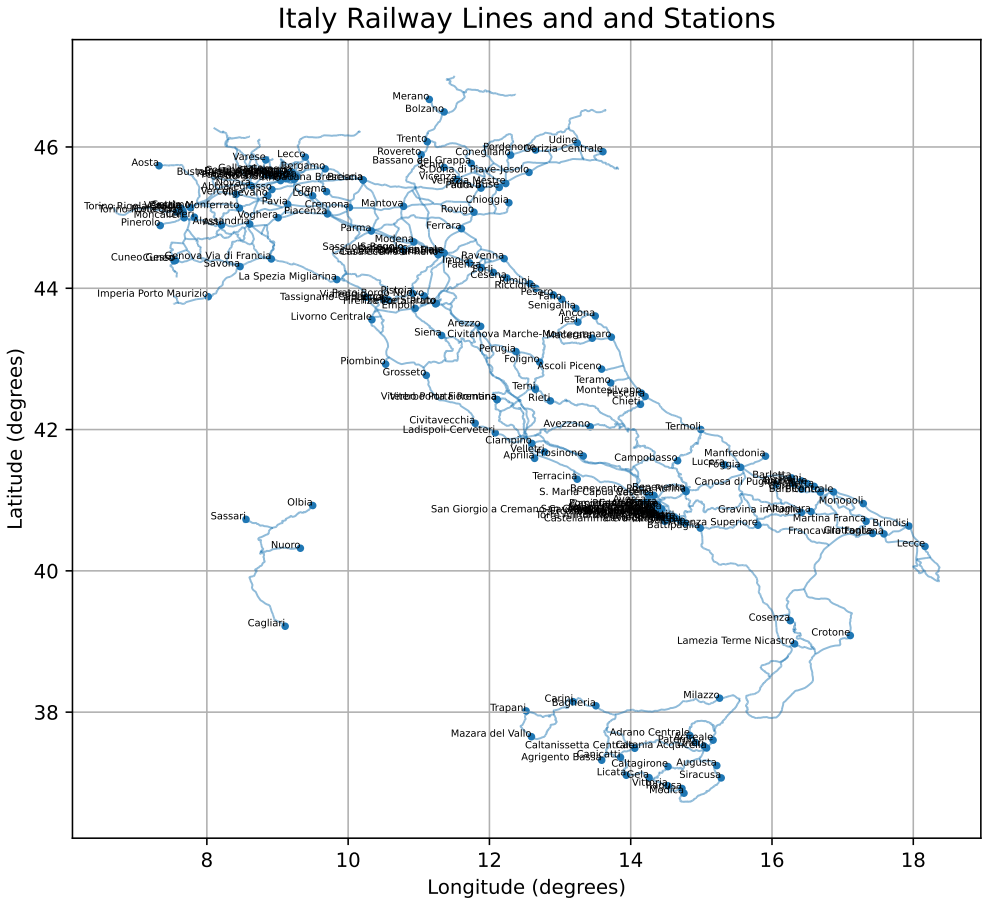


Figure 1.2: Railways and stations of Italy. Each dot (line, respectively) is an element of the dataframe stations (railways, respectively). We can already see by eye that data is incomplete. For instance, the train stations of Verona and Trieste are missing form the chart. Also, in the metropolitan area of Napoli there is a high density of stations, because also the smallest ones are reported in the database, whereas for other areas of Italy (like Veneto) only the biggest stations are reported. We cannot thus expect to recover an accurate network of railways from this dataset.

1.2 | Data Preprocessing

The extraction of an ordered list of stops from each railway has surely been the most demanding part of the project. The rule for edges definition is the following:

Rule for edges: Two stations A and B are connected with an edge if and only if they are *consecutive* stops of a *same* railway line.

The railway line data is encoded in a set of 'LineString' type objects, which are ordered

list of consecutive latitude/longitude coordinates, like in:

LINESTRING (13.89681899999979542.90991199999985,
13.895275999999797 42.92348499999986, 13.883477499999799
42.951797499999856, 13.87953349999978 42.96016299999985)

The essence of our method to recover the list of railway stops is this: we iterate over each point of the linestring object(s) (ex: 13.89681899999979542.9099119999998) and check if point coordinates match with any station's coordinates within a given tolerance threshold: if the condition is met, then the station is added to the list of railway stops. We defined two helper functions to do this: **extract_stops** and **flexible_matching**. We stored the list of railway stops in a separate file for each country, {country_code}_rails.csv"). The first line of this file for Italy are the following:

railway_name	railway_stops
Ancona-Lecce,	['Ancona', 'Civitanova Marche', 'Montesilvano', ...]
Foggia-Manfredonia	['Foggia', 'Manfredonia']
Macomer-Nuoro,	['Nuoro']
Chivasso-Ivrea-Aosta	[]

Finally, to extract the edges we only need to get all pairs of two consecutive stops. However, this simple reasoning is hindered by three major issues in the data, namely:

Issue 1 data is heavily fragmented: for each railway (i.e. for each 'NAMA1' entry in the raiways dataframe, like 'Ancona-Lecce') there are dozens of corresponding linestring objects. Each of them corresponds to a "slice" of various length and when concatenated together they make the railway (see figure Ancona - Lecce 1.3). The conseguence of this issue is that, if the railway is sliced into multiple objects, we will be able to recover only the unordered list of stops, whereas we really look for the ordered one.

Issue 2 data is designed to avoid redundance, in the sense that it is genuine infrastructural data. If two or more different lines (say, "Direttissima Roma-Napoli" and "Roma-Napoli AV") physically share a segment of rails, that segment will appear only in one of the rails (say, "Direttissima Roma-Napoli"). The other rail(s) will seem to stop in the middle of nowhere, whereas physically, it does not. See figure Napoli, Sardegna 1.4 for emblematic cases. The consequence of this issue is that, potentially, many physically existing edges could not be recovered.

Issue 3 there are EU countries (for example Polony, see figure 1.5) for which the railways are not even labeled. This means that our raw data is a completely unorganized set of linestring objects and there is no way to isolate the linestrings. We cannot construct a network for these countries.

For Issue 1, we were able to find at least a partial solution. For the majority of rails (over 90% for Italy), the set of LineString objects composing the railway could be merged into a single LineString, using methods **shapely.ops.linemerge** and **shapely.ops.unary_union** from Python library **shapely**. For these rails, we were then able to iterate from start to end over the rail and recover the ordered list of stops. However, in some cases where data was particularly bad and linestrings where not precisely aligned, the

merging method failed and we were only able to extract an unordered list of stops. As a consequence, for these unlucky railways the edges might be wrongly placed (i.e. between two stations that are not consecutive).
Issue 2 was harder to address. We couldn't find a way to fix the problem, other than manually adding the missing railways, which could be done for Italy with some patience but is totally impracticable for whole EU. The consequence of this issue is that the network will look much sparser than it really is.
Issue 3 was completely unrecoverable. As a consequence, we only built networks for 7 countries out of 17.

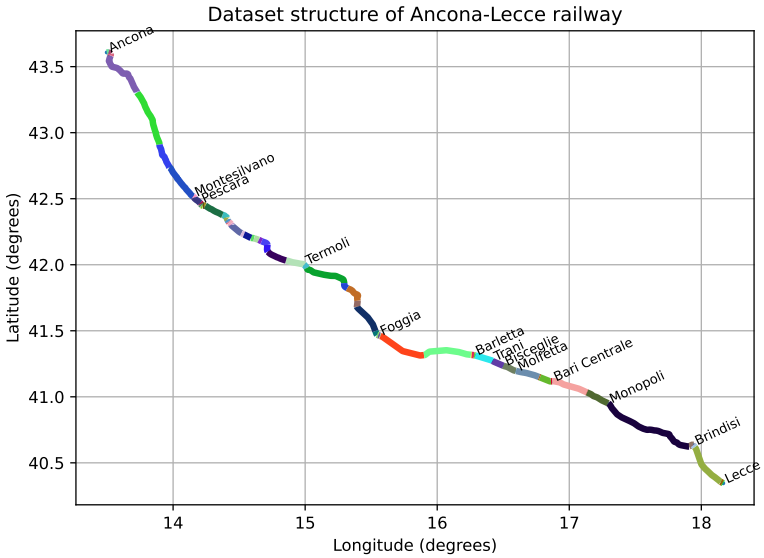


Figure 1.3: The structure of data for the italian railway "Ancona-Lecce": there are 70 different linestring objects making the whole rail. Each is plotted with a random different colors. For this railway the merging method succeeded and we were able to recover edges correctly according to our rule.

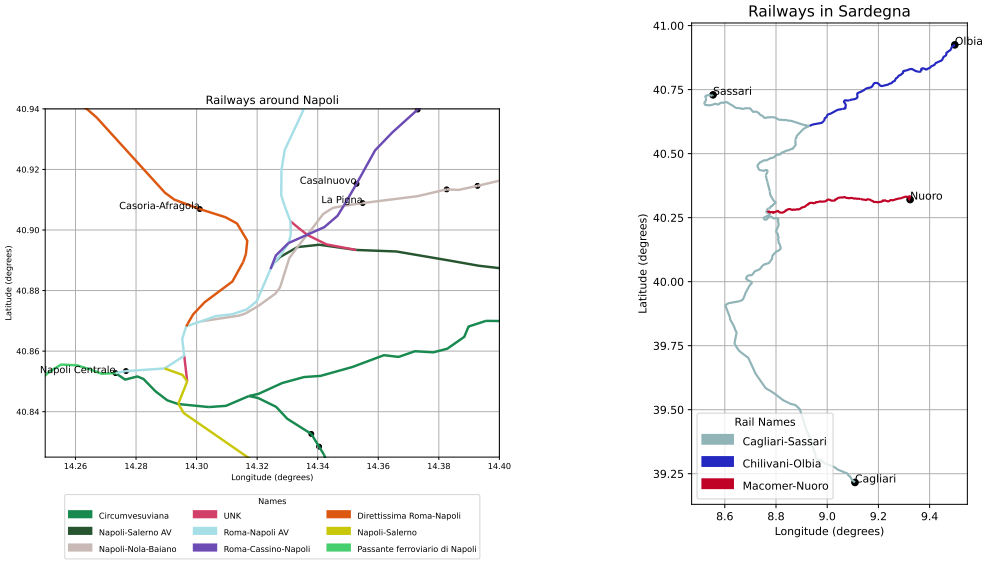


Figure 1.4: Left figure: a zoom of the railways around Napoli. The rail "Direttissima Roma Napoli", in orange, stops before reaching Napoli. Thus, Napoli does not appear as one of its stops, although it obviously is. The same happens for railway "Roma-Cassino-Napoli", in violet, and "Napoli-Nola-Baiano", grey. Right figure: railways structure of Sardegna. The correct list of edges is ([’Cagliari’, ’Nuoro’], [’Cagliari’, ’Olbia’], [’Cagliari’, ’Sassari’], [’Nuoro’, ’Sassari’], [’Nuoro’, ’Olbia’], [’Olbia’, ’Sassari’]), with a total of six edges. But since neither "Macomer-Nuoro" nor "Chilivani-Olbia" railways end in Sassari or Cagliari, our method of edges extraction only recovers ([’Cagliari’, ’Sassari’]) one edge!

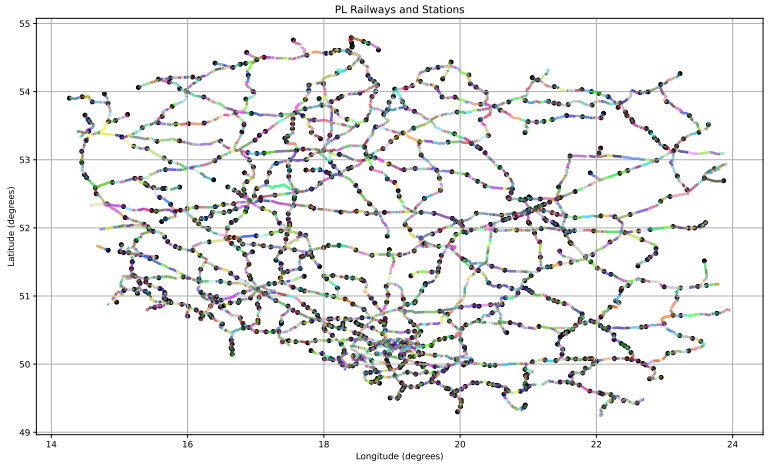


Figure 1.5: The structure of the data for Polony. Data is highly fragmented, railway intersections are ubiquitous and no labels are provided.

1.3 | Output files creation

After the two step preprocessing outlined in the above paragraph (which consisted in merging the linestrings and then extracting a stop list for each railway, stored in file "{country_code}_rails.csv"), only two easy steps remain:

1. create node files (nodeID, nodeLabel, latitude, longitude, country_name, country_ISO3) from dataframe "stations", for each country separately and whole Europe. Node label duplicates might be present (corresponding to two or more different stations in the same town). In this case, nodes must either be given different labels (like "Scafati_1" and "Scafati_2") or be deleted.

2. create edges file (nodeID_from, nodeID_to, node_label_from, node_label_to) combining together info from "{country_code}_rails.csv" files and "stations" dataframe.

2 | Network Visualization and Basic Analytics

In this section we present the results of our analysis. The network could be created only for those countries where the railways are labeled (i.e. fields 'NAMA1' or 'NAMA2' in dataframe "railways" are not both empty). These countries are:

IT (Italy), LU (Luxemburg), ND (Northen Ireland), NO (Norway), PT (Portugal), RS (Serbia)

The countries which had unlabeled railways were LT, LV, MD, MK, NL, PL, RO, SE, SI, SK, UA. We could not create a graph for those countries.

2.1 | Network visualization

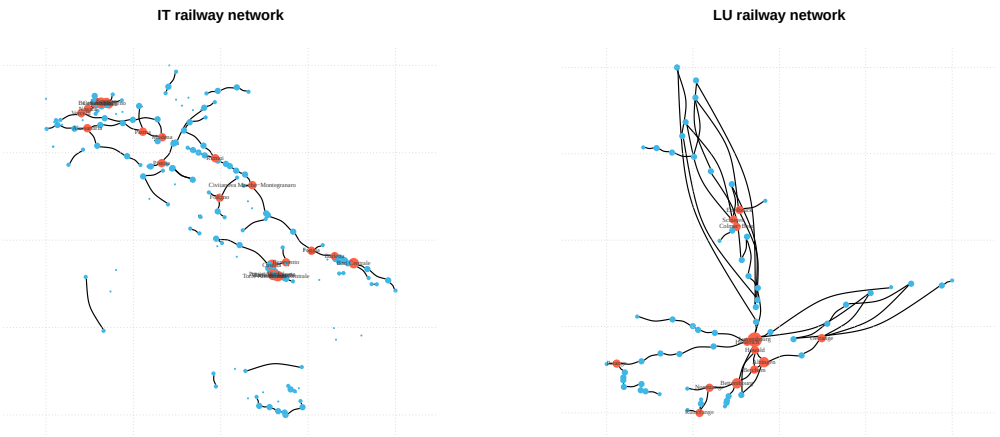


Figure 2.1: Nodes are colored orange if their degree is equal or bigger than the median degree, and blue otherwise.

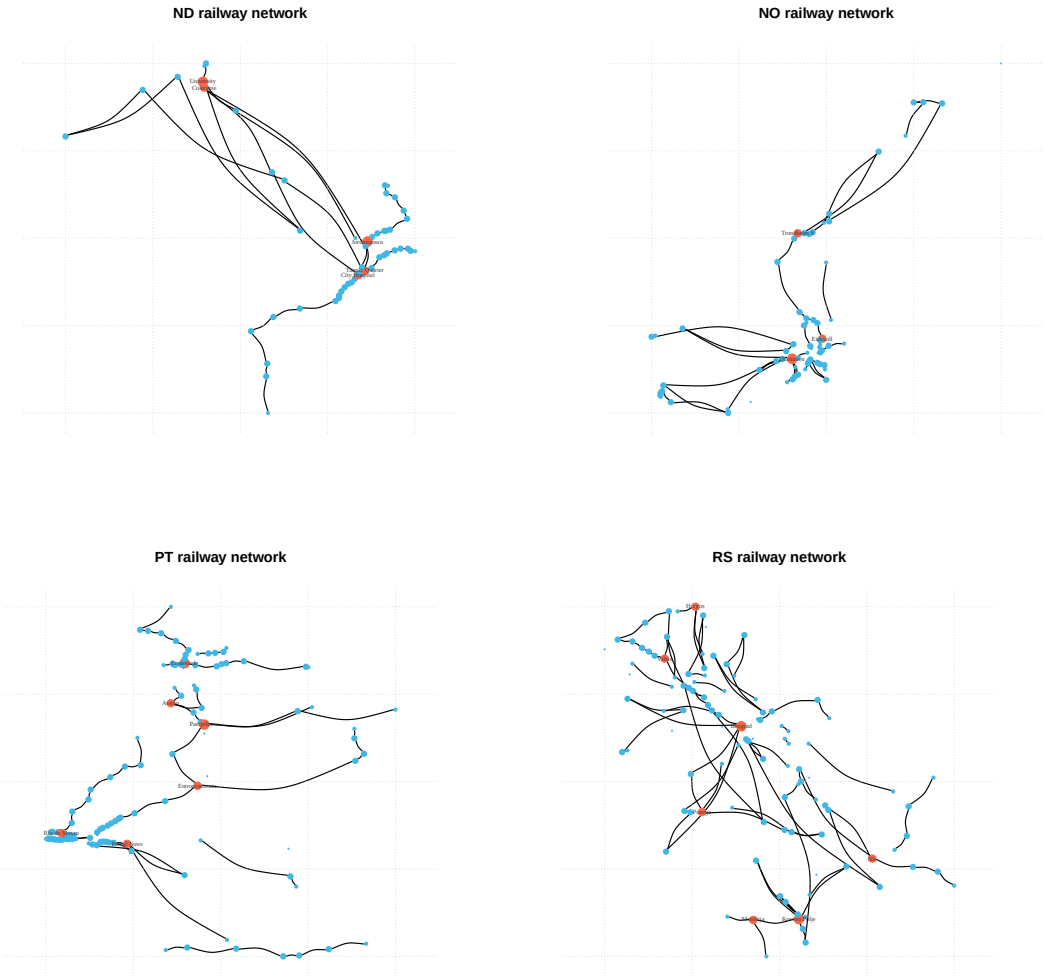
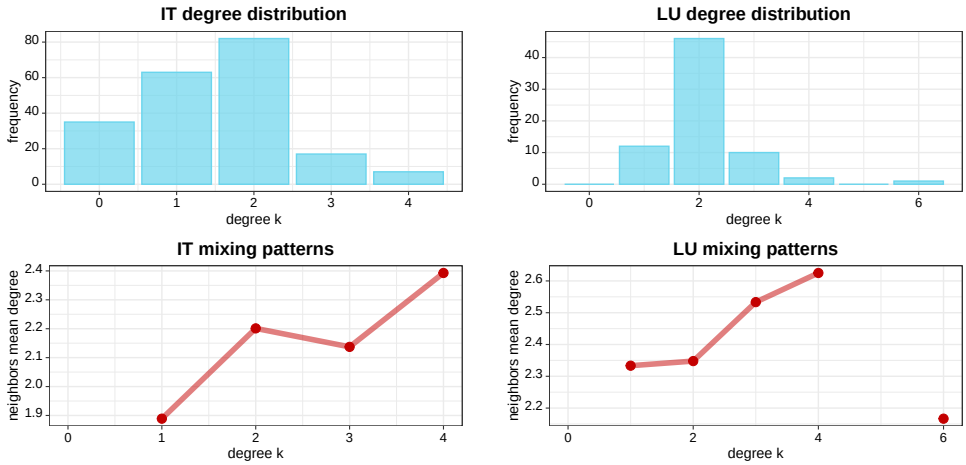
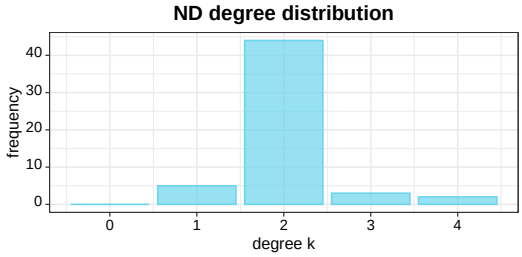
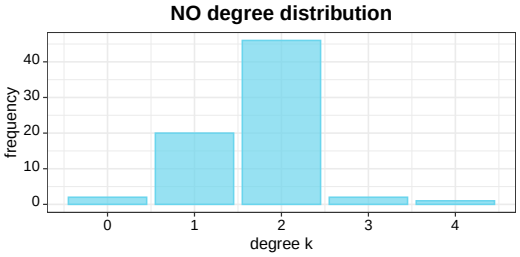


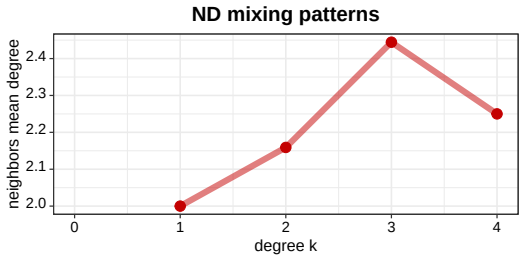
Figure 2.2: Nodes are colored orange if their degree is equal or bigger than the median degree, and blue otherwise.

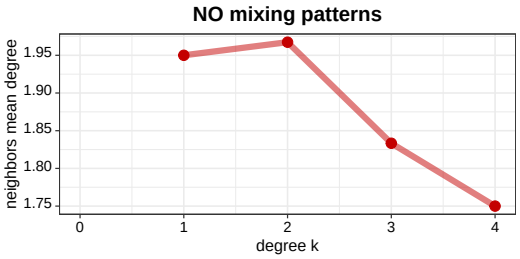
2.2 | Degree distribution and Mixing Patterns

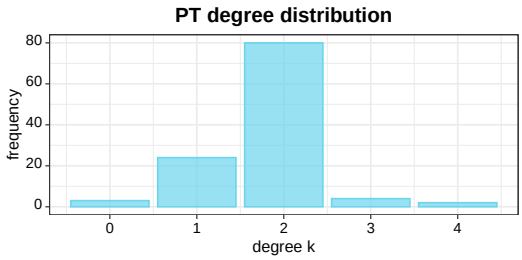


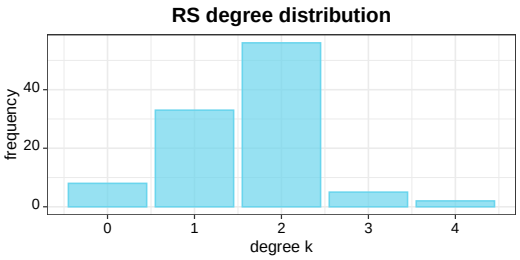

ND degree distribution
frequency
degree k

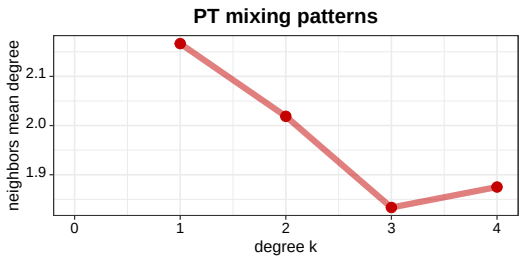

NO degree distribution
frequency
degree k

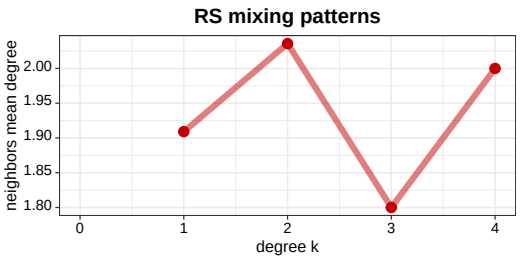

ND mixing patterns
neighbors mean degree
degree k


NO mixing patterns
neighbors mean degree
degree k


PT degree distribution
frequency
degree k


RS degree distribution
frequency
degree k


PT mixing patterns
neighbors mean degree
degree k


RS mixing patterns
neighbors mean degree
degree k