REGION
The Journal of ERSA
Powered by WU

Volume 12, Number 1, 2025, 73–101
DOI: 10.18335/region.v12i1.562

journal homepage: region.ersa.org
ISSN: 2409-5370

Spatial network analysis

Carmen Cabrera[1]

[1] University of Liverpool, Liverpool, United Kingdom

Received: 27 August 2024/Accepted: 14 February 2025

Abstract. The impact of current changes in technology, society, and the environment demand innovative research approaches that can accurately capture the increasing complexity of modern geographic data. To this end, this article provides an introduction to the use and implementation of spatial network analysis for the study of systems where geography matters. Presented as a computational notebook, the article offers practical, hands-on learning resources in R, assuming no prior knowledge from readers. Using a network of road links between African cities as case study, readers will be guided through key concepts and applications of spatial network analysis. The article highlights the relevance of network thinking in addressing contemporary geographic challenges.

1 Introduction

Sweeping transformations in technology, society and the environment are disrupting the status quo, driving innovation and challenging our understanding of the world. Among research fields, geography is significantly impacted by these changes due to its all-encompassing nature (Hartshorne 1939). The increased availability of data, the rise of Artificial Intelligence (AI), the improvement and deployment of urban sensors, the urbanisation and counter-urbanisation trends, the global spread of COVID-19, the evolving human mobility patterns, and the growing economic impact of natural disasters and climate change are just a few of the pressing issues that analysts interested in the study of geographic data are currently addressing.

Quantitative research concerning these new realities calls, more than ever, for approaches that embrace the interconnectedness and contextuality of the various elements involved in driving change. For example, a process like urbanisation is not just about the growth of urban populations but it also involves economic, social, and environmental dimensions and their interaction (Batty 2007, Bettencourt 2021). Similarly, climate change extends far beyond the realm of weather, with impacts on ecosystems, communities, financial markets and even politics (Lawrence et al. 2020). Responses to many of the emerging disruptions are additionally characterised by having ill-defined or malleable goals (e.g. achieving "sustainable urbanisation"), and by the presence of not only interdependent, but sometimes conflicting elements (e.g. designing a pandemic response while ensuring economic stability) (Miller et al. 2021). Therefore, the study of geographic data must adopt holistic frameworks and methodologies that regard the current real-world issues as far-reaching, wicked problems – as defined by (Rittel, Webber 1973) – embedded in complex systems (Miller 2017).

Complex systems are often organised as networked sets of elements which are embedded in space (Barthélemy 2011). These network structures, where space is relevant and where

topology alone does not contain all the information, are ubiquitous in the study of geographic data. Examples include transport networks (e.g. the subway network in a city, the network of flight connections between airports, the street network); mobility networks (e.g. the migration network between different countries, a network of commuting flows between the neighbourhoods within an urban area) or social and contact networks (e.g. the network of Facebook friends in different parts of the world). Therefore, we argue that network analysis remains one of the most significant and persistent research areas relating to the analysis of geographic data (Curtin 2007).

With roots in the natural sciences, the study of networks and complex systems explores how interactions among the individual give rise to larger, macro-level structures (Flake 1998). At the same time, network analysis provides powerful tools to investigate micro-level structures and properties, such as individual nodes and their local interactions, while embedding them within the broader context of the system. In this way, network analysis serves as a bridge between the "micro" and the "macro", offering insights into how local behaviours shape overarching patterns and vice versa.

Networks and complex systems thinking has traditionally focused on "extracting and abstracting", contrasting with the pre-disposition of geographers to "contextualise and specify" (O'Sullivan, Manson 2015). As a result, the use of complex systems approaches in geography and social sciences has faced criticism in the past for allegedly failing to capture human-driven qualitative change or power-dynamics, and neglecting substantive domain expertise (Uitermark, van Meeteren 2021, Franklin 2020). However, in recent times, with the rise of technological and socioeconomic transformations, alongside increasing data availability, there are numerous incentives to integrate complex systems approaches into geography (Nelson et al. 2025). These approaches should move away from sterile, often-critiqued conceptualisations of people and places, and instead, focus on data-driven analyses that consider the heterogeneity and spatial context as key drivers of social behaviours (Miller et al. 2021).

This article aims at providing an introduction of key concepts, definitions and applications of network analysis for the study of geographic data. Network analysis has a theoretical foundation in graph theory and topology (Curtin 2007). Rather than focusing on these foundational approaches to spatial networks, the focus here is on the practical implementation for data analysis and modelling. The article takes the form of a computational notebook, inviting the reader to follow a hands-on approach through a series of computational examples in R. These examples highlight the value of networks thinking in geography and spatial analysis by presenting applications based on a real-world network within a two-dimensional space.

The article is structured in the following way. The sections "Computational environment", "Data" and "Basic conceptual intuition" provide background and instructions to set up the necessary software to run the computational notebook. The body of the article can be found in the "Application" section, which includes learning resources for network construction and visualisation, network metrics, community detection and analysis of network robustness. Finally, there is a "Conclusion" section synthesising the learning outcomes.

2 Computational environment

This computational notebook is designed for reproducibility, so you can obtain consistent results by downloading and re-running without any modifications. To this end, it is necessary to ensure that the machine used to run the code has all the relevant software packages and versions installed.

The notebook is written using Quarto, an open-source, R Markdown-like publishing system which supports the integration of text, code and visualisations within a single document. Quarto documents are saved with the extension .qmd but can also be rendered in various formats, such as PDF, HTML, Microsoft Word or others. To use Quarto, you need to install the appropriate distribution for your operating system, which can be downloaded from the Quarto website. For the creation of this notebook, the version 1.5.55 (Mac OS) was used.

You will need to use RStudio, which requires R to be installed, to open and work on the Quarto notebook. R can be downloaded from the Comprehensive R Archive Network (CRAN), where installers for various operating systems, including Windows, macOS, and Linux, are provided. This notebook has been created using the R 4.4.1 version for MacOS, Apple Silicon (M1-3). Once R is installed, it is possible to proceed with the installation of RStudio, which can be downloaded from the Posit website. This notebook was created using the macOS 12+ version of RStudio.

You can open the notebook saved as a .qmd file on RStudio once you have installed Quarto, R, and RStudio. You will need to install a recent distribution of TeX such as TinyTeX to properly render the .qmd file into a .pdf. This can be done by running the following command on the terminal: quarto install tinytex. You will then need to add the Quarto template used to create this manuscript for REGION. Instructions on how to do this can be found in the README section of the following GitHub repository: https://github.com/region-ersa/REGION/. Furthermore, to run the code included in the notebook, you will need to install some R extensions, known as packages, that will be useful for the applications explored here. The packages, as well as the specific version that was used during the creation of the notebook, that you need to install are:

Package	Version	Description
`igraph`	2.0.3	for network manipulation and analysis
`sf`	1.0.16	to handle spatial data
`tidyverse`	2.0.0	for data manipulation and visualisation
`ggplot2`	3.5.1	for data visualisation
`ggraph-2`	.2.1	for graph visualisation
`patchwork`	1.2.0	to arrange plots
`tidygraph`	1.3.1	for tidy data handling with graphs
`RColorBrewer`	1.1.3	for color palettes
`rnaturalearth`	1.0.1	for natural earth map data
`ggspatial`	1.1.9	for geospatial visualisation

Open RStudio to install any package. Write the following command on the console window, normally situated at the bottom left: `install.packages("name of package")`. Make sure you replace "name of package" by the actual name of the package that you want to install e.g. `install.packages("tidyverse")`. Then, press enter and repeat this process until you have installed all the packages in the list.

Once the packages are installed, you will need to load them in order to be able to use them. This can be done by running the code below:

[1]:
```
# Load necessary libraries
library(igraph)          # Network analysis
library(sf)              # Handling spatial data (simple features)
library(tidyverse)       # Data manipulation and visualisation
library(ggplot2)         # Creating graphics and visualisations
library(ggraph)          # Visualising network data with ggplot2
library(patchwork)       # Combining multiple ggplot2 plots
library(tidygraph)       # Graph manipulation in a tidy data framework
library(RColorBrewer)    # Creating color palettes for visualisations
library(rnaturalearth)   # Accessing natural Earth geospatial data
library(ggspatial)       # Adding spatial context to ggplot2 maps
```

[2]:
```
# Disable scientific notation globally for figures
options(scipen = 999)
```

3 Data

Here, you get to work with a network which represents the main cities in the African continent and the road connections between them. The nodes of this network are the cities with a population greater than 100,000 people, obtained from (Moriconi-Ebrard et al.

2016). The edges represent the road infrastructure linking pairs of cities, sourced from OpenStreetMap (OpenStreetMap Contributors 2023, Maier 2014), and include primary roads, highways, and trunk roads. Each edge is associated with a distance measure based on the length of the roads that it represents. Additionally, to accurately represent the road infrastructure, the network also includes nodes representing road intersections, which are necessary for describing the connectivity between cities. These nodes are labelled as "transport nodes" and help define possible routes between cities. Some transport nodes correspond to towns with less than 100,000 inhabitants, so they are labelled as attached to nearby cities. The urban network enables us to consider the existing roads in the continent and measure the travelling distance rather than the physical distance between cities. The constructed network is formed by 7,361 nodes (2,162 cities and 5,199 transport nodes) and 9,159 edges. More details on how the network was built can be found in (Prieto-Curiel et al. 2022).

The network is connected, meaning that it is possible to find a sequence of nodes and existing roads linking any pair of cities, and therefore, it is also possible to find the shortest road distance between any two cities and define it as the network distance. The network consists of 361,000 km of road infrastructure and connects 461 million people living in African cities, representing roughly 39% of the continent's population (Prieto Curiel et al. 2022).

4 Basic conceptual intuition

Networks are used as a tool to conceptualise real-life systems where a group of items displays connections between themselves, such as the friendships among members of a school year group, hyperlinks between websites, or what-eats-what relationships in an ecological community. A network (or a graph) consists of nodes (or vertices) and edges (or links), where the latter represent the connections between the nodes (Aldous, Wilson 2000). Networks generally represent the arrangement of connections between pairs of nodes, also known as the topological information of the systems they represent. Networks are considered spatial when their topology alone does not provide all the necessary information. In such networks, the nodes are located in a space equipped with a metric (Barthélemy 2011), typically Euclidean distance in two-dimensional space. The probability of a link between two nodes generally decreases as the distance between them increases. An important type of spatial networks are planar networks, where edges can be drawn without crossing each other in a two-dimensional plane. However, they can also be non-planar, such as an airline passenger network that connects airports through direct flights where the flight connections often overlap geographically (Barthelemy 2018).

The geometry of the nodes is fundamental in spatial networks. While nodes may have explicit geographic coordinates, edges may not always directly reflect physical distances or real-world geometry. For example, the edges do not account for the actual routes or distances travelled in the case of straight-line connections between cities in a transportation network. However, the spatial arrangement of nodes still plays a significant role in the structure of the network.

The analysis of spatial networks is therefore an essential tool for studying patterns in geographic data. Spatial networks are found in a variety of contexts, such as transport links between subway stations (Cabrera-Arnau et al. 2023), flight connections between airports (Guimerà, Amaral 2004), neighbouring relationships between geographical locations (Anselin 1988), street networks in cities (Barthelemy, Boeing 2024) or road networks connecting cities (Prieto Curiel et al. 2022, Strano et al. 2017), as is the case of the application that we demonstrate in this article.

5 Application

You will now learn how to implement spatial network analysis for the study of geographic data. As described in the section Data, all the steps in the implementation will be exemplified with a network of African roads.

5.1 *Creating a network from a data frame*

The data that specifies the nodes and edges of the African road network is stored in two csv files, one for nodes and one for edges. This data can be loaded in two data frames:

[3]:
```
# Define URLs for nodes and edges CSV files
url_nodes <- paste0(
  "https://github.com/CrmnCA/spatial-networks-region-data/",
  "raw/main/data/AfricaNetworkNodes.csv"
)

url_edges <- paste0(
  "https://github.com/CrmnCA/spatial-networks-region-data/",
  "raw/main/data/AfricaNetworkEdges.csv"
)

# Read the CSV file containing network nodes data from the URL
df_nodes <- read.csv(url_nodes)

# Read the CSV file containing network edges data from the URL
df_edges <- read.csv(url_edges)
```

To provide a clearer understanding of the data structure, here are the first few rows of each data frame:

[4]:
```
# Display the first few rows of the nodes data frame
head(df_nodes)
```

[4]:
```
  Agglomeration_ID   agglosName      x       y  Pop2015 ISO3  Region
1             2320        Cairo 31.324  30.130 22995802  EGY   North
2             5199        Lagos  3.316   6.668 11847635  NGA    West
3             7098      Onitsha  6.928   5.815  8530514  NGA    West
4             4220 Johannesburg 28.016 -26.050  8314220  ZAF   South
5             4858     Kinshasa 15.293  -4.408  7270000  COD Central
6             5331       Luanda 13.385  -8.924  6979211  AGO Central
```

The `df_nodes` data frame specifies the properties of nodes in the network, including a unique ID for each node (`Agglomeration_ID`) and attributes such as geographic coordinates (`x` and `y`) or population size (`Pop2015`).

[5]:
```
# Display the first few rows of the edges data frame
head(df_edges)
```

[5]:
```
  from      to          l        h      time    timeU  timeUCB border
1 8211    2333  4.2943815 motorway  2.576629 80.44702 80.44702      0
2 8211 1000559  1.7716116 motorway  1.062967 15.14826 15.14826      0
3 8211 1000567  5.4142666 motorway  3.248560 17.33386 17.33386      0
4 8211    5425  0.7988003  primary  1.198201 21.55530 21.55530      0
5 8211 1054396 50.6469094  primary 75.970364 90.05566 90.05566      0
6 8211    8208  8.2435851  primary 12.365378 36.19194 36.19194      0
```

The `df_edges` data frame defines the connections between nodes. Each row contains a pair of nodes (i.e., "from" and "to") that are linked by an edge, along with attributes associated with the edge, such as `timeUCB` for travel time. Due to excessive computational costs to calculate these values within the computational notebook, it is important to note that the distances and travel times for each edge have been precomputed and stored in `df_edges`.

You can create a network as an `igraph` object using these data frames. Specifically, this network will be undirected (`directed = FALSE`), meaning the edges have no direction and the connection between two nodes is bidirectional. In other words, you can travel from A to B or from B to A without any directionality being implied if there is an edge between nodes A and B.

[6]:
```
# Create a network 'g' from data frames 'df_edges' and 'df_nodes'
g <- graph_from_data_frame(
  d = df_edges,
  vertices = df_nodes,
  directed = FALSE
)
```

When printed, the structure of an `igraph` object provides a summary of the network, including the number of nodes (vertices), edges, and key attributes. For example, printing the `g` object yields:

[7]:
```
g
```

[7]:
```
IGRAPH 354be48 UN-- 7361 9159 --
+ attr: name (v/c), agglosName (v/c), x (v/n), y (v/n), Pop2015 (v/n),
| ISO3 (v/c), Region (v/c), l (e/n), h (e/c), time (e/n), timeU (e/n),
| timeUCB (e/n), border (e/n)
+ edges from 354be48 (vertex names):
 [1] 2333--8211    8211--1000559 8211--1000567 8211--5425    8211--1054396
 [6] 8211--8208    8211--1055432 3467--1001315 3467--4936    3467--1296977
[11] 2333--1000607 2333--1001157 2333--1011439 2333--1027090 2333--1068481
[16] 2333--1116973 2333--3833    7356--6261    7356--1000307 7356--1000311
[21] 7356--1000863 7356--1041480 7356--1134459 6261--5163    7254--1000789
[26] 7254--1067779 2938--5889    2938--1032852 2938--1055190 2938--1055917
+ ... omitted several edges
```

UN– indicates that the graph is undirected, with 7,361 nodes and 9,159 edges. The graph also includes vertex attributes (`name`, `agglosName`, `x`, `y`, etc.) and edge attributes (`l`, `h`, `time`, `timeU`, etc.).

You can examine the attributes of the nodes in the network, which are automatically derived from the column names in the `df_nodes` data frame. These attributes provide additional information about the nodes and are an essential part of working with `igraph` objects:

[8]:
```
# Retrieve the attribute names associated with nodes in the 'g' network
vertex_attr_names(g)
```

[8]:
```
[1] "name"       "agglosName" "x"          "y"          "Pop2015"
[6] "ISO3"       "Region"
```

Specifically, `name` is the ID of each node in the network, `agglosName` is the name of the city represented by the node and it is set to `road` if the node is a transport node. `x` and `y` represent the coordinates of each node, `Pop2015` is the population of the city nodes, `ISO3` is the code for the country that each node is situated in, `Region` represents the region within the African continent that each node is situated in, and `Between` and `degree` represent the betweenness centrality and the degree of each node in the network, which you will also compute below.

You can look particularly at the first few values of any node attribute, for example `Pop2015`:

[9]:
```
# Retrieve the first few node names from the 'g' network
head(V(g)$Pop2015)
```

[9]:
```
[1] 22995802 11847635  8530514  8314220  7270000  6979211
```

You can also obtain the names of the edge attributes, which are taken from the columns in the `df_edges` data frame:

[10]:
```
# Retrieve the attribute names associated with edges in the 'g' network
edge_attr_names(g)
```

[10]:
```
[1] "l"       "h"       "time"    "timeU"   "timeUCB" "border"
```

where `l` represents the length in kilometres by road segment and it considers curves, `h` is the type of edge (primary, highway, etc.), `time` is the estimated minutes to travel through the edge, considering different speeds for distinct types of road, `timeU` is also the estimated minutes to travel through the edge, but allowing extra time if the ends of the edge are urban nodes, `timeUCB` allows further extra time for edges that cross a border and `border` is a binary variable taking value 1 if an edge crosses a border and 0 otherwise.

For further reading on `igraph` and related network analysis tools in R, Kolaczyk, Csárdi (2014) offers a detailed introduction to `igraph` functionalities.

5.2 Visualising the African road network as a spatial network

The most basic network visualisation can be achieved with the `plot` function from `igraph`. While the arguments of this function allow for some degree of customisation, the `ggraph` library is integrated with `ggplot2`, a powerful and flexible tool for the creation of graphics which provides advanced control over aesthetics, layering, and themes.

We will load the shapes of the African countries as a spatial feature object, with the `ne_download` function before creating the visualization. This will be used as a basemap.

[11]:
```
# Download world map data with specified parameters
world <- ne_download(
  scale = "small",
  category = "cultural",
  type = "admin_0_countries",
  returnclass = "sf"
)

# Extract unique country ISO3 codes from `df_nodes` to match the network
target_countries <- unique(df_nodes$ISO3)

# Subset the world map data to include only the target countries
world_subset <- world[world$SOV_A3 %in% target_countries, ]
```

We will also need to set the position of the nodes by specifying their layout:

[12]:
```
custom_layout <- data.frame(
  name = df_nodes$agglosName,  # Node names from the graph
  x = df_nodes$x,              # Custom x-coordinates
  y = df_nodes$y               # Custom y-coordinates
)
```

Furthermore, the node size can be set to vary according to the population of the cities that they represent. City population sizes vary over several orders of magnitude, so rather than making the size of the nodes proportional to the population size, it is better to apply a scaling function to reduce the disparity in sizes:

[13]:
```
# Calculate and assign a 'size' attribute to nodes in the 'g' network
# Size is determined based on the population data of each node
V(g)$size <- 0.1 * (V(g)$Pop2015 / 80000)^3
```

Now, with a few modifications to the default plot in order to improve the appearance, including setting the size of nodes as a function of the population, the network is ready to be plotted.

[14]:
```
ggraph(as_tbl_graph(g), custom_layout) + # Basic graph plot
  geom_edge_link(
    color = "gray20",
    alpha = 0.9,
    aes(width = E(g)$l * 0.1)  # Custom edges
  ) +
```

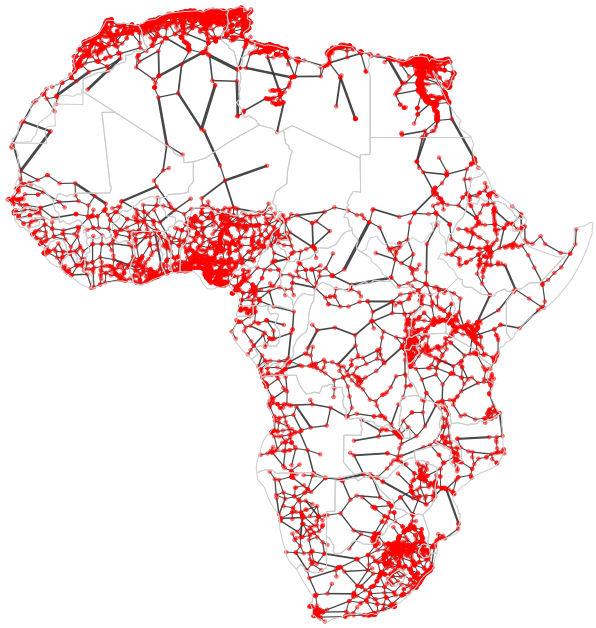

Figure 1: Visualisation of the African road network.

```
  scale_edge_width(range = c(.1, 0.7)) +  # Scale edge size
  geom_node_point(
    aes(color = "red", alpha = 0.8, size = V(g)$size)  # Custom nodes
  ) +
  scale_size_continuous(range = c(.3, 4)) +  # Scale node size
  scale_color_identity() +  # Scale node color
  theme(
    legend.position = "none",
    panel.background = element_rect(fill = NA, colour = NA)
  ) +
  geom_sf(
    data = world_subset,
    fill = NA,
    color = "gray80"  # Basic map plot
  )
```

[14]:
```
Output in Figure 1.
```

As an exercise, you may want to try to plot the default visualisation by simply running `plot(g)`. This demonstrates how simple modifications to the default plot can make a significant difference in the appearance of the outcome.

5.3 Network metrics

Network metrics are useful to obtain measurable insights into the network structure. They are also valuable as a way to characterise the network so that it can be compared to other networks.

5.3.1 Density

The density of a network refers to the proportion of existing edges over all possible edges. In a network with n nodes, the total number of possible edges is $n \times (n-1)/2$. A density equal to 1 corresponds to a situation where $n \times (n-1)/2$ edges are present. A network

with no edges at all would have density equal to 0. We can obtain the density of the African road network by running the following code:

[15]:
```
# Calculate edge density (excluding loops)
dens <- edge_density(g, loops = FALSE)
dens
```

[15]:
```
[1] 0.0003381142
```

The edge density for the African road network is approximately 0.00034, giving an indication that the network is quite sparse, since out of all possible edges, only 0.034% are present.

Beyond suggesting limited connectivity, low edge density also translates into challenges in transportation infrastructure, particularly in regions where road networks are sparse and fragmented due to geographical, economic, and historical factors. From an economic standpoint, low network density can hinder economic integration and regional trade by limiting connections between key locations (Linard et al. 2012). Improving connectivity in such sparse networks can reduce transportation costs, improve access to markets, and create greater economic opportunities, highlighting the importance of road network density for fostering economic development (Prieto Curiel et al. 2022).

5.3.2 Reciprocity

The reciprocity in a directed network is the proportion of reciprocated connections between nodes (i.e. number of pairs of nodes with edges in both directions) from all the existing edges.

[16]:
```
# Calculate the reciprocity of the edges in the 'g' network
reciprocity(g)
```

[16]:
```
[1] 1
```

Every edge is inherently bidirectional in an undirected graph, meaning that there is always a corresponding edge from B to A if there is an edge between node A and node B. Therefore, the reciprocity of an undirected graph is naturally 1, as all connections are reciprocated by definition.

5.3.3 Distances

A path in a network between node A and node B is a sequence of edges joining distinct nodes, starting at node A and ending at node B. Each node in the path is visited only once. In the case of a directed path, all edges must align with the specified direction, ensuring the path follows the network's directional flow.

The length of a path between nodes A and B is generally defined as the number of edges forming this path. The shortest path is the minimum count of edges present to travel from A to B. The path length can be also be defined in alternative ways. For example, the path length can be defined as the sum of the weights of the edges forming a path if the edges are weighted.

We can use the function `shortest_paths()` to find the shortest path between a given pair of nodes, taking into account the geographic road length associated with the edges. For example, between Cairo and Lagos, we can apply `shortest_paths()`, setting `weights` to `df_edges$l`, and store the output in a dataframe called `df_shortest_path`.

[17]:
```
# Calculate the shortest path between "Cairo" and "Lagos"
# Edge length is used as weight
df_shortest_path <- shortest_paths(
  g,
  from = V(g)$agglosName == "Cairo",
  to = V(g)$agglosName == "Lagos",
  predecessors = FALSE,
```

```
  weights = df_edges$l,
  output = "both"
)
```

In this dataframe, the field `epath` stores the edges of the shortest path as a one-element list. We can extract the values of this list as the edge IDs, which we then use to compute the geographic road length associated with the shortest path between the two network nodes.

[18]:
```
# Get the edge path indices from 'df_shortest_path'
idx <- df_shortest_path$epath[[1]]

length(idx)
```

[18]:
```
[1] 143
```

[19]:
```
# Get the lengths of edges along the path
lengths_epath <- edge_attr(g, "l", idx)

# Calculate the total length of the path
sum(lengths_epath)
```

[19]:
```
[1] 6084.359
```

We find that the network length associated with the shortest path is 143, while the geographic road length associated with this path is 6,084.359 km. This road distance, derived from the network representation of the African road network, can be compared with distances provided by routing services. For instance, Google Maps estimates the road transport distance between these two cities to be 6,431 km, which represents only a 5.7% difference from the value obtained here."

The diameter of a network is the longest shortest path between any pair of nodes, measured in terms of network distance. In the case of the African road network, we can incorporate the geographic road length as edge weights. Using these weights, the diameter reflects the greatest road length between any two nodes in the network.

[20]:
```
# Calculate the diameter of the network
diameter(g, directed = FALSE, weights = df_edges$l)
```

[20]:
```
[1] 11987.06
```

We obtain a diameter of 11,987.06 km, which roughly corresponds to the distance between the northernmost and southernmost points of the African continent.

The mean distance is the average length of all shortest paths in the network. The mean distance will always be smaller or equal than the diameter.

[21]:
```
# Calculate the mean distance of the network
mean_distance(g, directed = FALSE, weights = NULL)
```

[21]:
```
[1] 55.8602
```

[22]:
```
# Calculate the mean distance of the network
mean_distance(g, directed = FALSE, weights = df_edges$l)
```

[22]:
```
[1] 4878.73
```

Here we see that the average network distance between any given pair of nodes is 55.86 edges, while the geographic road distance between any given pair of nodes is 4,878.73 km.

5.3.4 Centrality

Centrality metrics assign scores to nodes, and sometimes edges, according to their position within a network. These metrics can be used to identify the most influential or important

nodes. In addition to measuring the prominence of individual nodes, centrality metrics also provide insight into the overall structure of the network. We can understand how local interactions at the node level contribute to global patterns and behaviours within the network by examining the distribution of centrality scores. Additionally, the degree distribution can reveal whether a network is highly centralised, with a few nodes having disproportionately high centrality scores, or whether it is more balanced, with nodes having similar centrality scores. This link between micro-level node properties and the broader macro-level structure highlights the role of network analysis in connecting individual node dynamics with the overall system.

5.3.4.1 Degree The degree of a node is one of the simplest and most fundamental measures of centrality in a network. It is defined as the number of edges connected to the node. In directed networks, the degree can be further divided into the in-degree, which counts the number of edges directed towards a node, and the out-degree, which counts the number of edges originating from it. The `degree()` function enables the calculation of these measures for one or more nodes in a network. Users can specify whether they are interested in the total degree (combining in- and out-degrees), the in-degree, or the out-degree, depending on the focus of their analysis.

We can compute the degree of each node with the `degree` function since the African road network is undirected.

[23]:
```
# Compute degree of the nodes given by v belonging to network g
deg <- degree(g, v = V(g))
```

We produce a histogram to visualise the results.

[24]:
```
# Produce a histogram showing frequency of nodes with certain degree
hist(deg,
     breaks = 50,
     main = "Distribution of degree",
     xlab = "Degree",
     ylab = "Frequency")
```

[24]:
```
Output in Figure 2.
```

We observe that most nodes have degree 3. Nodes of degree 1 are terminal nodes. Nodes of degree 2 are relatively less common than those of degree 1 and 3. This is likely due to the method used to build the network, where all the transport nodes of degree 2 are eliminated in order to simplify the network. It is relatively rare to find any nodes beyond degree 4. From the histogram, we see the maximum degree observed in the network is 13. Below, we obtain the name of the node with the maximum degree as well as the value of the degree (13).

[25]:
```
# Get names of nodes with the highest degree in the 'g' network
V(g)$agglosName[
  degree(g) == max(degree(g))
]
```

[25]:
```
[1] "Duduza Central"
```

[26]:
```
# Get IDs of nodes with the highest degree in the 'g' network
highest_degree_node_names <- V(g)$name[
  degree(g) == max(degree(g))
]
# Calculate the degree of nodes with the highest degree
degree(
  g,
  v = highest_degree_node_names
)
```

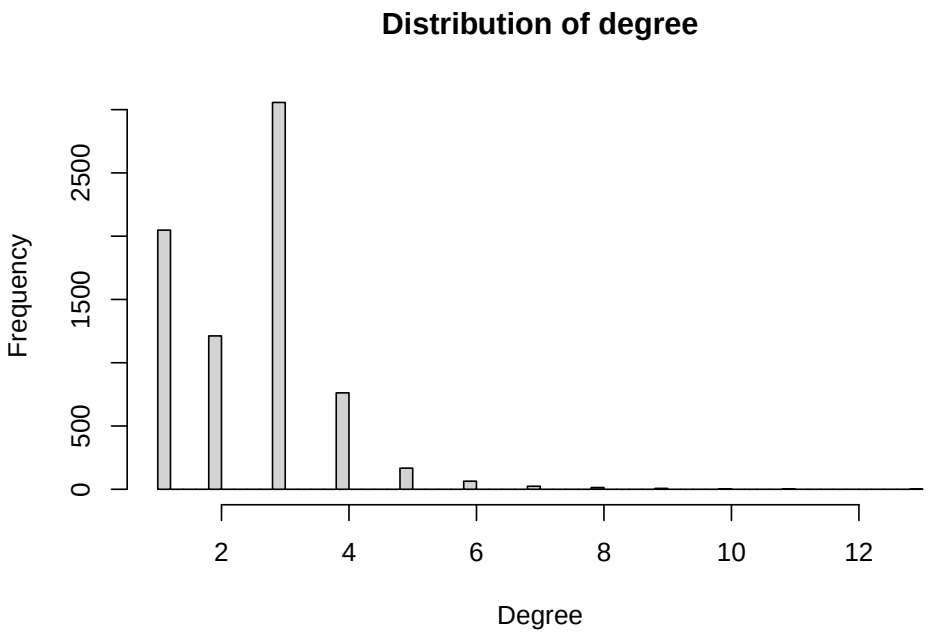

Figure 2: Histogram of node degree.

[26]:
```
2896
  13
```

The degree of 13 for Duduza means that it is directly connected to 13 neighbouring nodes (cities or intersections) in the road network. This degree indicates that Duduza serves as a relatively well-connected hub in the network.

We can also measure the weighted degree of a node. This is known as the strength of a node and it is computed as the sum of edge weights linked to adjacent nodes. Both the degree and strength are considered to be centrality metrics.

5.3.4.2 Closeness centrality Closeness centrality is a measure of the shortest path between a node and all the other nodes, measured in terms of network distance. It is calculated as the inverse of the average shortest path between a node and all other nodes. The closer this value is to 1, the more central the node is in the network. A value of 0 indicates an isolated node. You can use the `closeness()` function to compute closeness centrality for a network. The calculation assumes unweighted edges when setting `weights = NULL`, meaning the shortest paths are computed without accounting for edge weights such as geographic road lengths. Below, we compute the closeness centrality for a network using unweighted edges and visualise the results in a histogram to examine the distribution.

[27]:
```
# Calculate the closeness centrality for each node (unweighted edges)
close_centr <- closeness(g, weights = NULL)

# Create a histogram of closeness centrality
hist(close_centr,
     breaks = 50,
     main = "Distribution of closeness centrality",
     xlab = "Closeness centrality",
     ylab = "Frequency")
```

[27]:
```
Output in Figure 3.
```

The resulting distribution is bimodal. This distribution reflects the effect of geography in determining the overall connectivity structure of the African road network, whereby

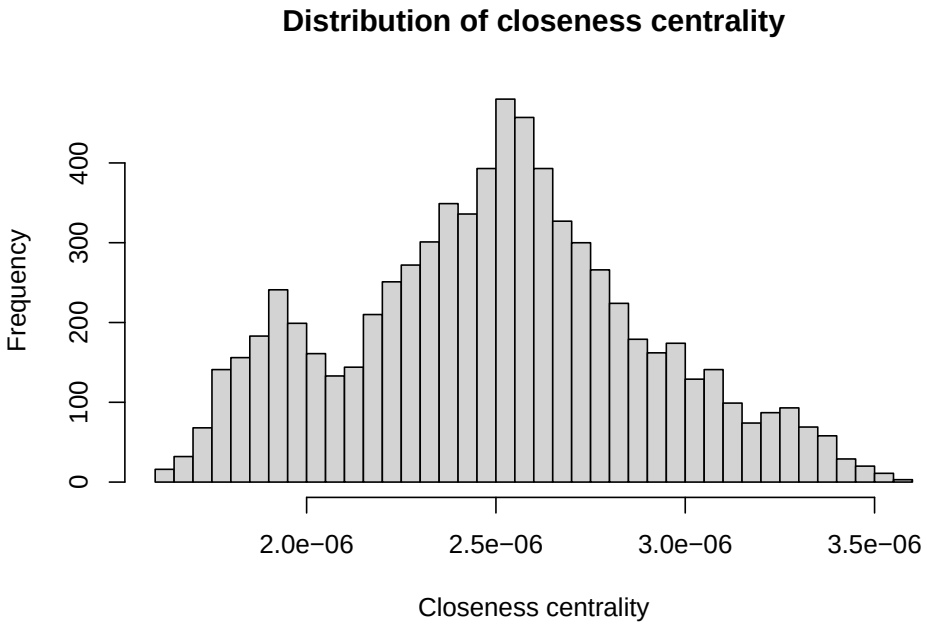

Figure 3: Histogram of unweighted closeness centrality.

the Sahara desert practically separates the network into two groups of nodes. Nodes within each of the groups, corresponding to cities either North or South of the desert, are much closer to each other than to the nodes belonging to the other group. Notably, the closeness centrality values are very low. This is due to the sparsity of the network, as closeness centrality is inversely related to the shortest path distances between nodes. Many nodes are far apart in a sparse network, resulting in very small closeness centrality values.

Closeness centrality can also be computed using weighted edges. In this case, geographic road length can be used as a weight to measure the physical proximity of each node to all other nodes in the network. The resulting distribution can be visualised using a histogram.

[28]:
```
# Calculate the closeness centrality for each node (weighted edges)
close_centr_weight <- closeness(g, weights = df_edges$l)

# Create a histogram of closeness centrality
hist(close_centr_weight,
     breaks = 50,
     main = "Distribution of weighted closeness centrality",
     xlab = "Closeness centrality (weighted)",
     ylab = "Frequency")
```

[28]:
```
Output in Figure 4.
```

Notably, the shape of the distribution changes when considering weighted or unweighted edges, demonstrating the importance of this choice in the outcome.

5.3.4.3 Betwenness centrality Similarly, betweenness centrality is a measure of the number of shortest paths going through a node. High values of betweenness centrality indicate that the corresponding nodes play an important role in the overall connectivity of the network. Betweenness can also be computed for edges. We compute the betweenness centrality for all nodes using the function `betweeness()` and represent it as a histogram. We do this using unweighted edges, so the computation of betweenness considers only network distances but not geogrpahic distances.

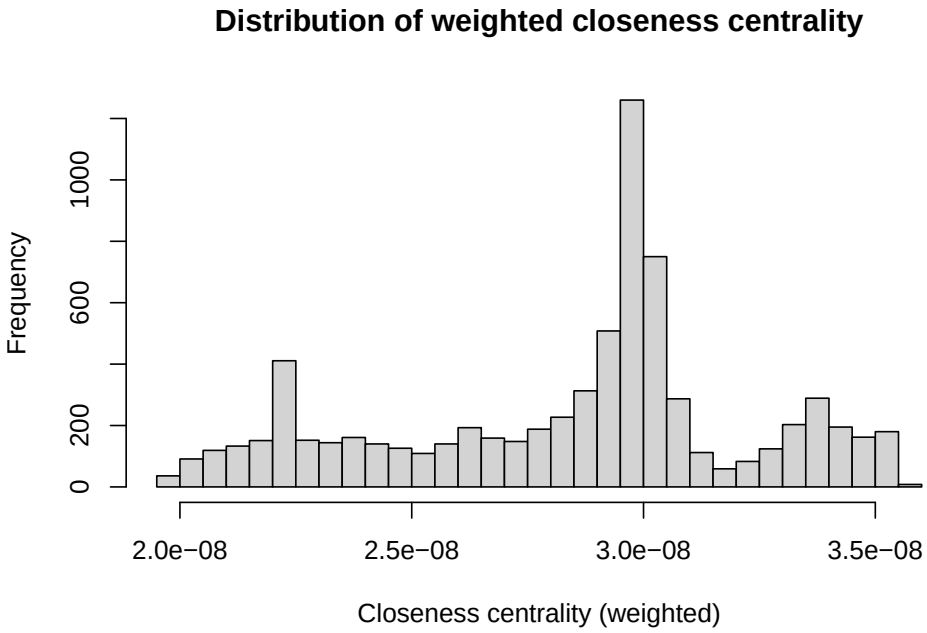

Figure 4: Histogram of weighted closeness centrality.

[29]:
```
# Calculate the betweenness centrality for each node in the g network
between_centr <- betweenness(g, v = V(g), directed = FALSE,
     weights = NULL)

# Create a histogram of betweenness centrality
hist(between_centr,
     breaks = 50,
     main = "Distribution of unweighted betweenness centrality",
     xlab = "Betweenness centrality",
     ylab = "Frequency")
```

[29]:
```
Output in Figure 5.
```

The resulting distribution shows that most nodes have low values of betweenness centrality and very few of them have large values. This means that there is a small number of nodes that are crucial to ensure the existence of a shortest path between any given pair of nodes. Without these key nodes, distances within the network could become larger or even infinite, since the network could be broken into isolated components. The emergence of such skewed distribution of the nodes betweenness centrality is due to the specific geographic features of the African road network, where once again, connections between cities North and South of the Sahara are facilitated by a small number of nodes with high values of betweenness centrality.

5.3.5 Assortativity

The assortativity coefficient quantifies the tendency of nodes in a network to connect to others with similar or dissimilar attributes. This metric can be calculated based on any standard node property, such as degree, or a custom attribute (e.g., population size). Mathematically, it is defined as the Pearson correlation coefficient of the specified attribute between pairs of connected nodes, with values ranging from -1 to 1. Positive assortativity indicates that nodes are more likely to connect with others having similar attributes. Negative assortativity suggests that nodes tend to connect with others having dissimilar attributes. In `igraph`, the `assortativity()` function computes the assortativity coefficient for a specified attribute, while the `assortativity_degree()`

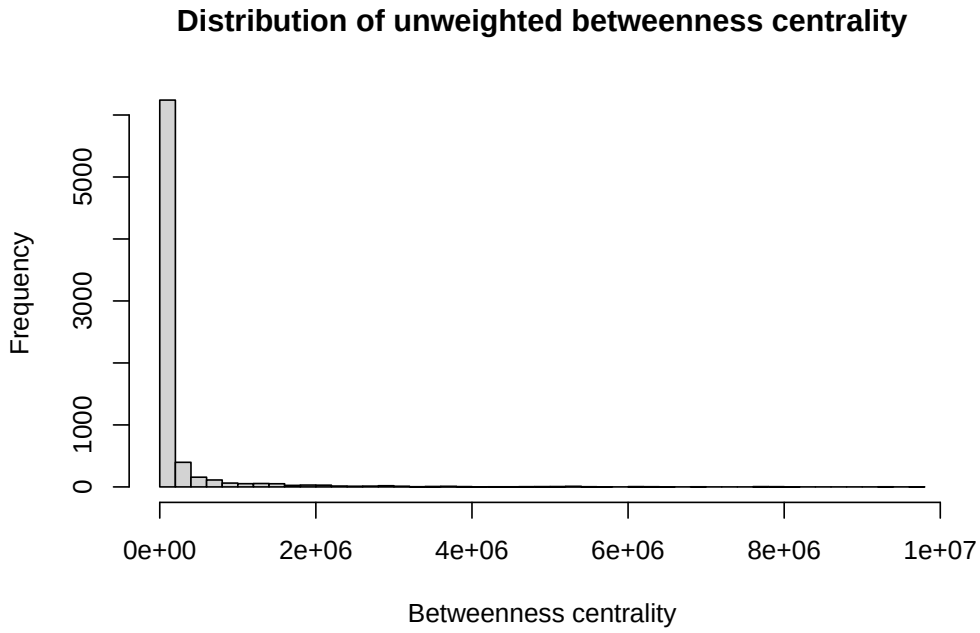

Figure 5: Histogram of betweenness centrality.

function specifically evaluates the assortativity based on node degrees.

Below is the code for computing assortativity of a custom network attribute such as the population size.

[30]:
```
assortativity(
  g,
  V(g)$Pop2015,
  directed = FALSE
)
```

[30]:
```
[1] 0.0006432013
```

We use the `assortativity_degree()` function to compute the degree assortativity:

[31]:
```
# Compute assortativity for degree
assortativity_degree(g, directed = FALSE)
```

[31]:
```
[1] -9.080965e-05
```

Both `assortativity()` and `assortativity_degree()` return values close to zero, indicating the absence of a strong tendency for nodes to connect either to similar or dissimilar nodes.

To better understand how degree assortativity manifests in the network, we can visualise the joint degree distribution, which shows the frequency of edges connecting nodes with specific degree pairs:

[32]:
```
# Extract node degrees
deg <- degree(g)

# Compute degree pairs for all edges
edge_degree_pairs <- t(sapply(E(g), function(e) {
  ends(g, e, names = FALSE) %>%
    sapply(function(v) deg[v])
}))
```

```
# Convert to data frame for visualisation
joint_degree_df <- as.data.frame(edge_degree_pairs)
colnames(joint_degree_df) <- c("Degree_1", "Degree_2")

# Aggregate counts for each degree pair
joint_degree_counts <- joint_degree_df %>%
  group_by(Degree_1, Degree_2) %>%
  summarise(Frequency = n(), .groups = "drop")

# Heatmap visualisation
ggplot(joint_degree_counts, aes(x = Degree_1,
                                y = Degree_2,
                                fill = Frequency)) +
  geom_tile(color = "white") +
  scale_fill_gradientn(
        colors = c("white", "lightblue", "blue", "darkblue"),
        trans = "log10",
        name = "Frequency") +
  labs(title = "Joint Degree Distribution",
        x = "Degree of Node 1",
        y = "Degree of Node 2",
        fill = "Log-scaled Frequency") +
  theme_minimal() +
  theme(panel.grid.major = element_blank(),
        axis.text = element_text(size = 10),
        axis.title = element_text(size = 12),
        legend.text = element_text(size = 10),
        legend.title = element_text(size = 12),
        legend.position = "right") +
  scale_x_continuous(
    breaks = seq(1, max(joint_degree_counts$Degree_1), by = 1)
  ) +
  scale_y_continuous(
    breaks = seq(1, max(joint_degree_counts$Degree_2), by = 1)
  ) +
  coord_cartesian(expand = FALSE) +
  geom_hline(yintercept = seq(0.5, max(joint_degree_counts$Degree_2) +
      0.5,by = 1),
    color = "grey90", linetype = "solid") +
  geom_vline(xintercept = seq(0.5, max(joint_degree_counts$Degree_1) +
      0.5,by = 1),
    color = "grey90", linetype = "solid")
```

[32]:
```
Output in Figure 6.
```

Since the `assortativity()` and `assortativity_degree()` functions in `igraph` provide only the assortativity coefficient, a permutation test can be implemented to assess the statistical significance of the assortativity coefficient. A permutation test compares the observed assortativity coefficient to a distribution of coefficients from randomly shuffled networks. This helps determine whether the observed value is significantly different from what could arise by chance.

Here is how to compute a p-value using a permutation test:

[33]:
```
# Observed degree assortativity coefficient
observed_assortativity <- assortativity_degree(g)

# Set seed for reproducibility
set.seed(123)
```

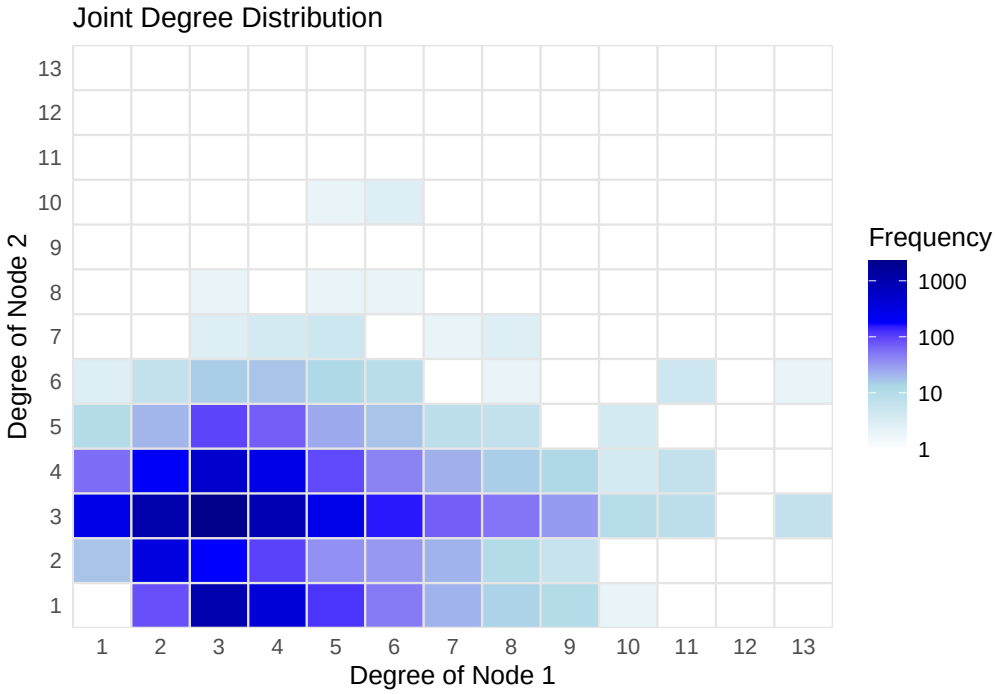

Figure 6: Joint degree distribution

```
n_permutations <- 1000

# Rewire edges and compute assortativity for random networks
random_assortativities <- replicate(n_permutations, {
  g_random <- rewire(g, keeping_degseq(niter = ecount(g) * 10))
  assortativity_degree(g_random)
})

# Compute p-value by comparing observed and random assortativities
p_value <- mean(
  abs(random_assortativities) >= abs(observed_assortativity)
)

p_value
```

```
[33]: [1] 0.997
```

We cannot reject, at the 95% confidence level, the hypothesis that the observed assortativity could arise from a random configuration, since p-value ≥ 0.05. If p-value was < 0.05, the observed assortativity would be unlikely to arise by chance, indicating significant assortativity.

5.4 Community detection

A network displays community structure if the nodes can be grouped into sets such that the nodes within each set are densely connected. For example, in the case of a social network formed by the students in a classroom, we would expect that small groups of friends form within the overall network, where relationships among members of a group are stronger than to everyone else in the classroom. Detecting or searching communities within in a network is a fundamental problem in network analysis, which has attracted much attention in the past decades (Fortunato, Newman 2022). While there are different methods to detect communities, below we review three of them which have been widely

used and studied. These are walktrap (Pons, Latapy 2005), edge-betweenness (Girvan, Newman 2002) and the Louvain method (Blondel et al. 2008).

Each of these methods emphasises different aspects of community structure, leading to varying outcomes. The walktrap method is particularly suited to identifying small, localised communities, making it useful in applications where fine-grained clusters are of interest, such as detecting close-knit friend groups or tightly interconnected modules in biological networks. The edge-betweenness method excels at uncovering well-separated communities, especially in networks where connections between groups are sparse or function as bottlenecks, such as departments in an organisation or transportation networks. The Louvain methdod is best for capturing broader, hierarchical structures and is especially effective for large-scale networks, where computational efficiency and an overview of the global structure are key.

The community structures they produce can differ significantly because these methods focus on different aspects of the network. For instance, walktrap or edge-betweenness may reveal smaller, more detailed clusters, whereas Louvain often detects larger, coarser groupings. It is important to choose a method thoughtfully based on the specific application and research question because the detected community structure can directly shape the interpretation of the network.

It is also worth noting that both walktrap and the Louvain method incorporate random elements in their algorithms. As a result, running these methods on the same network can yield slightly different community structures across iterations. To address this variability, it is recommended to run the algorithms multiple times and assess the stability of the detected communities using similarity metrics, such as the RAND index (Rand 1971), which quantify the agreement between different community partitions. Including such assessments can improve confidence in the robustness of the results and ensure reproducibility, especially for applications where stable community detection is relevant.

5.4.1 Walktrap

This algorithm relies on the concept of random walks on networks. Random walks are sequences of nodes, chosen by following a randomly chosen path. The underlying assumption of the walktrap method is that nodes encountered in a given random walk are more likely to be part of the same community.

The algorithm starts by treating each node as its own community. Then, it performs a series of short random walks on the network, where the length of these walks has to be specified by the user. After performing the random walks, the algorithm calculates a similarity measure between each pair of nodes. This measure is based on the idea that if two nodes are often encountered together during random walks, they are likely part of the same community. Nodes that have high similarity are merged into larger communities. This merging process is hierarchical and agglomerative, starting with individual nodes and progressively combining them. As communities are merged, the algorithm often aims to maximise a measure called modularity, which quantifies the strength of the division of the network into communities. High modularity indicates a good community structure, where more edges fall within communities than between communities. The process continues until the entire network is merged into a single community or until a stopping criterion, like a modularity threshold is met. The algorithm may return a hierarchical structure of communities depending on the implementation, allowing the user to explore different levels of granularity in the community structure.

The walktrap method is implemented in R via the `igraph` function `cluster_walktrap()`, with key parameters including the network of interest as an igraph object, the length of the random walks, and a membership parameter, which is a boolean variable indicating whether to calculate membership based on the highest modularity score (with True as the default). Below, we apply the walktrap method to the African road network and save the result in the variable `g_wt`:

[34]:
```
# Perform walktrap clustering on the graph
g_wt <- cluster_walktrap(graph = g, steps = 3, membership = TRUE)
```

We can get the membership of each node as well as the modularity score according to the solution based on random walks of length 3. We save the results with the name `member_g_wt` and `modularity_g_wt`:

[35]:
```
# Get the membership of clusters in the walktrap clustering
member_g_wt <- membership(g_wt)

# Calculate the modularity of the walktrap clustering
modularity_g_wt <- modularity(g_wt)
```

We can plot the network based on the found communities. We will plot the network as before, but color the nodes based on the communities found using the walktrap algorithm. The plot is included in Figure 7:

[36]:
```
plot_wt <- ggraph(as_tbl_graph(g), custom_layout) +  # Basic graph plot
  geom_edge_link(color = "gray20",
                 alpha = 0.9,
                 aes(width = E(g)$l * 0.1)) +
  scale_edge_width(range = c(.1, 0.7)) +  # Scale edge size
  geom_node_point(aes(color = member_g_wt,
                      size = V(g)$size,
                      alpha = 0.8)) +
  scale_size_continuous(range = c(.3, 4)) +  # Scale node size
  scale_color_identity() +  # Scale node color
  theme(legend.position = "none",
    panel.background = element_rect(fill = NA, colour = NA)) +
  geom_sf(data = world_subset,
          fill = NA,
          color = "gray80") +
  ggtitle("Walktrap method")
```

5.4.2 Edge betweenness community detection

The edge-betweenness community detection method identifies communities within a network by focusing on the edges that connect different communities. It works by progressively removing edges that act as bridges between groups of nodes, from higher to lower betweenness centrality. As high-betweenness edges are removed, the network breaks down into smaller, more cohesive subgroups or communities. Though it can be computationally intensive for large networks, this method is particularly useful for finding natural divisions within a network.

The algorithm starts by computing the betweenness centrality for all edges in the network. Edges with high betweenness are likely to be those that connect different communities. Then, the edge with the highest betweenness centrality is removed from the network. This step effectively "cuts" the bridge between communities. After removing the edge, the betweenness centrality for the remaining edges is recomputed since the removal of one edge may change the shortest paths in the network, affecting the betweenness centrality of other edges. Edges with the highest betweenness centrality keep being removed until all edges have been removed or until the network breaks down into the desired number of communities.

The edge betweenness community detection method is implemented in R via the `igraph` function `cluster_edge_betweenness()`. The key parameters are the network of interest as an igraph object and a membership parameter, a boolean variable indicating whether to calculate membership based on the highest modularity score (with True as the default). Below, we apply the edge betweenness method to the African road network and save the result in the variable `g_eb`:

[37]:
```
# Perform edge betweenness clustering on the graph
g_eb <- cluster_edge_betweenness(graph = g, membership = TRUE)
```

Once again, we can get the membership of each node according to the solution based on edge betweenness. We save the results with the name `member_g_eb`:

[38]:
```
# Get the membership of clusters in the edge betweenness clustering
member_g_eb <- membership(g_eb)
```

Then, we generate a plot of the results. We include the plot in Figure 7.

[39]:
```
plot_eb <- ggraph(as_tbl_graph(g), custom_layout) +  # Basic graph plot
  geom_edge_link(color = "gray20",
                 alpha = 0.9,aes(width = E(g)$l * 0.1)) +
  scale_edge_width(range = c(.1, 0.7)) +  # Scale edge size
  geom_node_point(aes(color = member_g_eb,
                      size = V(g)$size,
                      alpha = 0.8)) +
  scale_size_continuous(range = c(.3, 4)) +  # Scale node size
  scale_color_identity() +  # Scale node color
  theme(legend.position = "none",
        panel.background = element_rect(fill = NA, colour = NA)) +
  geom_sf(data = world_subset,
          fill = NA,color = "gray80") +
  ggtitle("Edge betweenness\n clustering")
```

5.4.3 Louvain method

The Louvain method of multi-level clustering works by finding communities in such a way that the modularity of the network is maximised. The algorithm works in two phases: first, each node starts in its own community, and nodes are iteratively moved to neighboring communities if the move increases modularity. This phase continues until no further improvement is possible. In the second phase, the network is compressed by treating each community found in the first phase as a single node, creating a new, smaller network. The two phases are then repeated on this simplified network, thus refining the community structure at each level. The process continues until modularity no longer increases, resulting in a hierarchical clustering that reflects the community structure within the network.

In R, the Louvain method is implemented via the `cluster_louvain()` function, where the arguments are the graph and the resolution. Higher resolution values will yield a larger number of smaller communities, while lower values will yield a smaller number of larger communities.

[40]:
```
# Perform Louvain clustering on the graph with a resolution of 1
g_mlc <- cluster_louvain(graph = g, resolution = 1)
```

We get the membership of each node according to the communities found by the Louvain's multi-level clustering method. The results are saved with the name `member_g_mlc`:

[41]:
```
# Get the membership of clusters in the Louvain clustering
member_g_mlc <- membership(g_mlc)
```

We can then generate a plot of the results. The plot is included in Figure 7:

[42]:
```
plot_mlc <- ggraph(as_tbl_graph(g), custom_layout) +  # Basic graph plot
  geom_edge_link(color = "gray20",
                 alpha = 0.9,
                 aes(width = E(g)$l * 0.1)) +
  scale_edge_width(range = c(.1, 0.7)) +  # Scale edge size
  geom_node_point(aes(color = member_g_mlc,
                      size = V(g)$size,
                      alpha = 0.8)) +
```

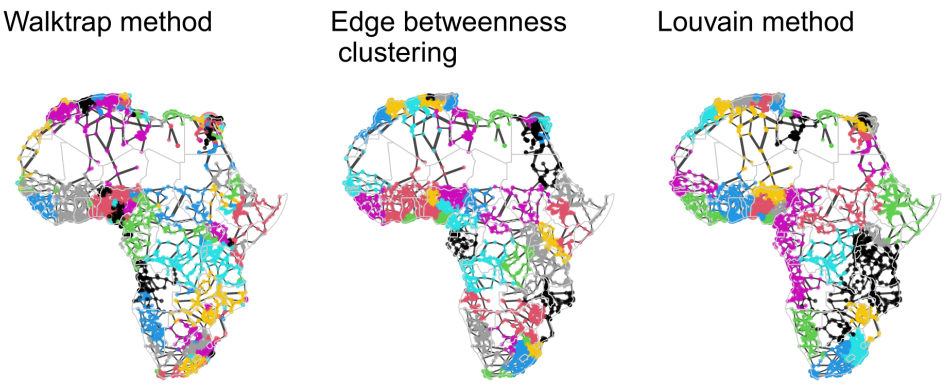

Figure 7: Outcomes of three community detection algorithms

```
  scale_size_continuous(range = c(.3, 4)) +  # Scale node size
  scale_color_identity() +  # Scale node color
  theme(legend.position = "none",
        panel.background = element_rect(fill = NA, colour = NA)) +
  geom_sf(data = world_subset,
          fill = NA,
          color = "gray80") +
  ggtitle("Louvain method")
```

In Figure 7, we visualise the results of the three community detection methods, i.e. walktrap, edge betweenness and the Louvain method. The results show that nodes that are geographically close and are part of densely connected clusters, generally belong to the same community, regardless of the method used. There is some degree of correspondance between the detected communities and the African countries.

[43]:
```
# Combine the three community detection algorithm plots
plot_wt + plot_eb + plot_mlc
```

[43]:
```
Output in Figure 7.
```

5.5 *Analysing network robustness*

Robustness is the ability of a network to maintain its basic functions in the presence of node and link failures (Barabási, Pósfai 2016). Percolation theory, originally developed in statistical physics to explore the relationship between microscopic and macroscopic properties of a medium, is a widely used approach to studying network robustness. It helps identify the conditions under which a network remains well-connected, ensuring that most nodes can still interact or communicate effectively despite potential disruptions. A well-connected network is characterised by the presence of a giant connected component (GCC), which is a single, large cluster of interconnected nodes (Bollobás 2001). The subset of network nodes within the GCC are reachable from each other, either directly or indirectly, through paths. This means that, from any node node within the GCC, it is possible to navigate through the network and eventually reach most of the other nodes.

The Molloy-Reed criterion is an important theoretical condition for the emergence of the GCC in a network (Molloy, Reed 1995). This criterion provides a threshold based on the degree distribution of nodes for a GCC to exist in a network. Specifically, the Molloy-Reed criterion states, in absence of degree correlations (no degree assortativity), that a GCC will emerge if the following is true: $\langle k^2 \rangle - 2\langle k \rangle > 0$, where $\langle k \rangle$ is the average degree and $\langle k^2 \rangle$ is the mean squared degree. The network is sufficiently connected to support the formation of a GCC when the inequality holds. Conversely, the network will fragment into small, isolated clusters when the inequality does not hold.

It is crucial in real-world applications to ensure that networks have a GCC for maintaining a system's resilience. For example, the GCC ensures that most locations remain accessible in transportation networks, while it enables the majority of devices or users to exchange information in communication networks. Understanding the robustness of these types of networks can therefore be valuable for improving regional resilience, because it allows regions to better accommodate shocks and develop new growth paths (Elekes et al. 2024).

More specifically, percolation analysis can be used to determine the threshold at which a GCC emerges. For instance, percolation analysis can help calculate the minimum number of edges that need to be added to keep the network in a well-connected state given a network with a specific number of nodes. On the other hand, inverse percolation (Barabási, Pósfai 2016) helps identify the point at which the removal of edges or nodes leads to network fragmentation, breaking it into smaller, isolated clusters. The following section focuses on inverse percolation, as the selected approach seeks to better understand network robustness by assessing a network's ability to remain well-connected when nodes or edges are removed.

A full inverse percolation algorithm or process is typically run so that the value of a percolation parameter that controls the removal of nodes or edges is updated in each iteration, and nodes or edges are removed accordingly. Key robustness metrics are measured in each iteration. One of the most used robustness metrics is the number of nodes in the GCC after the removal of nodes or edges. This metric is known as the size of the GCC. In many cases, we observe that abrupt changes occur in the size of the GCC for certain values of the percolation parameter. This indicates that some sort of failure occurs in the network that qualitatively changes its connectivity structure.

This type of analysis is demonstrated below. The percolation parameter of choice is the time of travel through each edge, accounting for the presence of borders. This variable is encoded by the `timeUCB` field in the `df_edges` data frame. Edges with `timeUCB` above the value of the percolation parameter are removed from the network in each iteration of the inverse percolation process. Furthermore, instead of considering the whole African road network, a subset is considered, as this facilitates timely execution of the algorithm. A subset formed by nodes and edges from the South region is used.

[44]:
```r
# Subset df_nodes for rows where Region is "South"
df_nodes_sub <- subset(df_nodes, Region == "South")

# Subset df_edges for rows where 'from' values are in Agglomeration_ID
df_edges_sub <- subset(df_edges,
                       from %in% df_nodes_sub$Agglomeration_ID)

# Subset df_edges_sub for rows where 'to' values are in Agglomeration_ID
df_edges_sub <- subset(df_edges_sub,
                       to %in% df_nodes_sub$Agglomeration_ID)
```

We can create an undirected graph from the redefined data frames of nodes and edges.

[45]:
```r
# Create an igraph graph 'g_sub' from 'df_edges_sub' and 'df_nodes_sub'
g_sub <- graph_from_data_frame(d = df_edges_sub,
                               vertices = df_nodes_sub,
                               directed = FALSE)
```

We can also visualise this subnetwork by running the code below. We will add the outlines of the countries in the South region as a base layer for this plot to give more geographical context to the above visualization. These are Botswana, Eswatini, Lesotho, Namibia and South Africa.

[46]:
```r
# Define a vector of target countries
south_countries <- c("Botswana", "eSwatini", "Lesotho",
                     "Namibia", "South Africa")
```

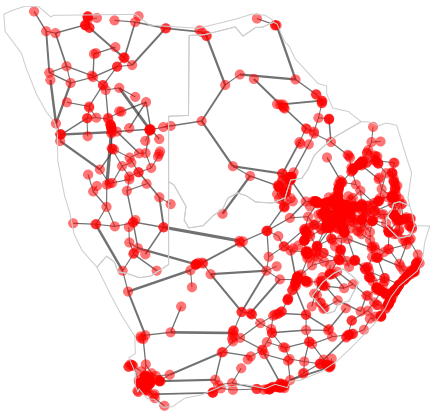

Figure 8: Visualisation of a road network in the South region of the African continent

```
# Subset the world map data to include only the target countries
world_south <- world[world$SOVEREIGNT %in% south_countries, ]

# Specify node layout
custom_layout <- data.frame(
  name = df_nodes_sub$agglosName,  # Node names from the graph
  x = df_nodes_sub$x,              # Custom x-coordinates
  y = df_nodes_sub$y               # Custom y-coordinates
)
```

[47]:
```
# Create plot
ggraph(as_tbl_graph(g_sub),
                    custom_layout) +  # basic graph plot
  geom_edge_link(color = "gray20",
                 alpha = 0.7,
                 aes(width = E(g_sub)$l * 0.1)) +  # custom edges
  scale_edge_width(range = c(.1, 0.7)) +  # scale edge size
  geom_node_point(aes(color = "red",
                      size = V(g_sub)$size,
                      alpha = 80)) +  # custom nodes
  scale_size_continuous(range = c(.1, 6)) +  # scale node size
  scale_color_identity() +  # scale node color
  theme(legend.position = "none",
        panel.background = element_rect(fill = NA, colour = NA)) +
  geom_sf(data = world_south,
          fill = NA,
          color = "gray80")  # basic map plot
```

[47]:
```
Output in Figure 8.
```

5.5.1 The inverse percolation algorithm

The inverse percolation algorithm simulates the progressive breakdown of a network by iteratively removing edges based on a percolation parameter (in this case, `timeUCB`). The algorithm evaluates how the removal of edges affects the structure of the network at each iteration. This process helps us understand how robust the network is and whether it disintegrates as we vary the percolation threshold. We need to store key metrics about the network at each step to analyse the results of this iterative process. We begin by creating empty data structures to store information about the network during each iteration. Specifically, we create four empty lists where we will record:

- The value of the percolation parameter at the current iteration.
- The size of the largest connected component (GCC) in the network, which indicates how much of the network remains connected.
- The total number of connected components in the network, which reflects how fragmented the network becomes.
- The average time to travel between any pair of nodes, weighted by the edge attribute `timeUCB`, to capture changes in the network connectivity.

These lists will each contain as many elements as iterations by the end of the inverse percolation process, corresponding to the results for all iterations.

[48]:
```
# Create empty vectors to store parameters, gccs, ncs, and times
parameters <- c()
gccs <- c()
ncs <- c()
times <- c()
```

We are now ready to perform the inverse percolation algorithm. The algorithm proceeds iteratively, with each iteration representing a step where edges with `timeUCB` values above the current percolation threshold are removed from the network. The logic for each step in the algorithm is as follows:

1) Define the current percolation threshold. At the start of each iteration, the current value of the percolation parameter `i` is defined. This value determines which edges will remain in the network at this step (those with `timeUCB` less than `i`).
2) Filter the network based on the percolation threshold. The nodes remain unchanged, but the edges are filtered to include only those with `timeUCB` values below the current threshold. This results in a modified edge list, which represents the network at the current percolation step.
3) Construct a new graph. Using the filtered edge list and the full node list, we create a new graph `g_perco`. This graph reflects the state of the network after removing edges based on the percolation parameter.
4) Analyse the modified graph by computing and storing the metrics defined before, i.e. the he size of the largest connected component (GCC), the number of connected components and the mean geographic road distance between nodes. Each metric is appended to its respective list for later analysis. By the end of the algorithm, these lists will contain a complete record of how the network evolved at each percolation step.

Below is the code implementation of the inverse percolation algorithm. Each line has been commented to describe its function:

[49]:
```
# Iterate over parameters
for (i in seq(0, max(df_edges_sub$timeUCB))) {

  # Create modified data frames based on the current parameter
  df_nodes_perco <- df_nodes_sub
  df_edges_perco <- subset(df_edges_sub, timeUCB < i)

  # Create a graph g_perco from the modified data frames
  g_perco <- graph_from_data_frame(d = df_edges_perco,
                                   vertices = df_nodes_perco,
                                   directed = FALSE)

  # Get connected components of the modified graph g_perco
  connected_components <- components(g_perco)

  # Append the current parameter value to the 'parameters' list
  parameters <- c(parameters, i)

  # Append the maximum connected component size to the 'gccs' list
```

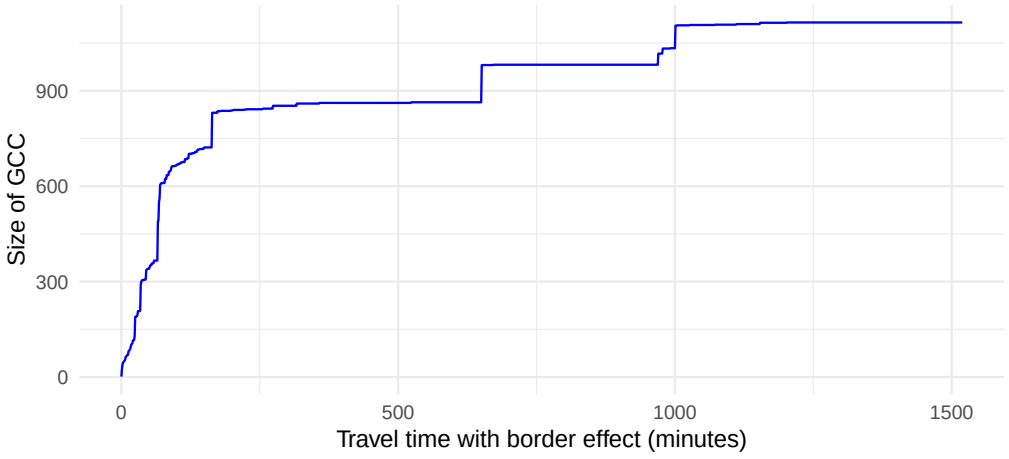

Figure 9: Relationship between the size of the Giant Connected Component and the percolation parameter

```
  gccs <- c(gccs, max(connected_components$csize))

  # Append the number of connected components to the 'ncs' list
  ncs <- c(ncs, connected_components$no)

  # Calculate and append the mean distance weighted by timeUCB
  times <- c(times, mean_distance(g_perco,
                                  directed = FALSE,
                                  weights = df_edges_perco$timeUCB,
                                  unconnected = TRUE))
}
```

5.5.2 Changes in the size of the giant connected component as edges are removed

Once the algorithm is done running, we can plot the size of the GCC as the value of the percolation parameter is varied.

[50]:
```
# Create a data frame for the plot with parameters and gccs
df <- data.frame(x = parameters, y = gccs)
# Create a ggplot2 plot with customised aesthetics and labels
ggplot(data = df, aes(x = x, y = y)) +
  geom_line(color = "blue") +
  labs(x = "Travel time with border effect (minutes)",
       y = "Size of GCC") +
  theme_minimal()
```

[50]:
```
Output in Figure 9.
```

We observe for small values of the percolation parameter that rapid changes occur in the size of the GCC. There are sudden changes in the size of the GCC when the percolation parameter takes approximately the values 150, 650, 1000, indicating that there has been a significant alteration in the network's topology. For example, when edges with associated travel times of 1000 minutes or less are removed, nodes that act like hubs may lose connections, and the size of the GCC becomes smaller as a result.

5.5.3 Changes in the number of connected components as edges are removed

We can also plot the number of connected components as the value of the percolation parameter is varied.

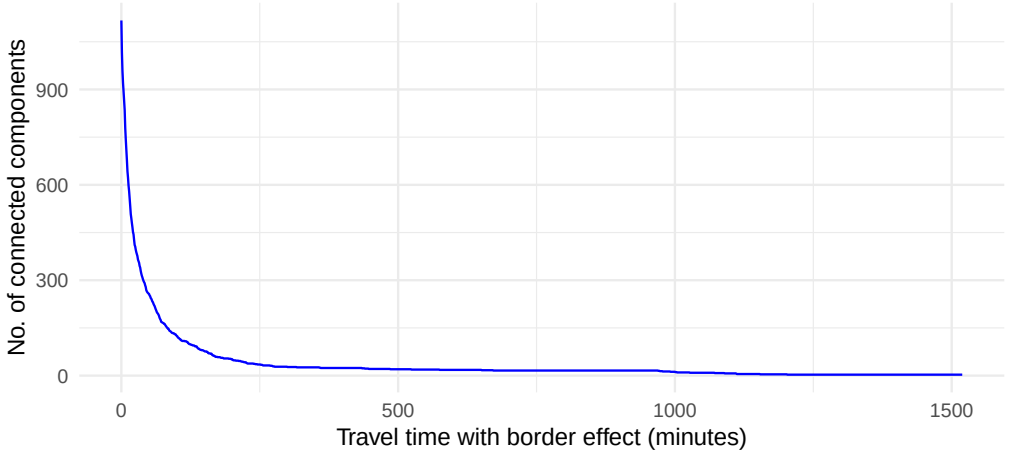

Figure 10: Relationship between the number of connected components and the percolation parameter

[51]:
```
# Create a data frame for the plot with parameters and ncs
df <- data.frame(x = parameters, y = ncs)

# Create a ggplot2 plot
ggplot(data = df, aes(x = x, y = y)) +
  geom_line(color = "blue") +
  labs(x = "Travel time with border effect (minutes)",
       y = "No. of connected components") +
  theme_minimal()
```

[51]:
```
Output in Figure 10.
```

We observe that nearly all the edges in the network are removed for small values of the percolation parameter so that there are as many components as there are nodes. We also see that the number of connected components is reduced if the percolation parameter is above 250 minutes, highlighting that the connectivity of the network is greater above that threshold value as the network is less fragmented.

5.5.4 Changes in the average travel time as edges are removed

Finally, we plot the average travel time between any pair of nodes as the value of the percolation parameter is varied.

[52]:
```
# Create a data frame for the plot with percolation parameter and times
df <- data.frame(x = parameters, y = times)

# Create a ggplot2 plot
ggplot(data = df,
       aes(x = x, y = y)) +
  geom_line(color = "blue") +
  labs(x = "Travel time with border effect (minutes)",
       y = "Average travel time (minutes)") +
  theme_minimal()
```

[52]:
```
Output in Figure 11.
```

Note that all the edges are removed when the percolation parameter is 0, and so the corresponding value of the average travel time is NA. As the percolation parameter is increased, less edges are removed from the original network and more possible paths

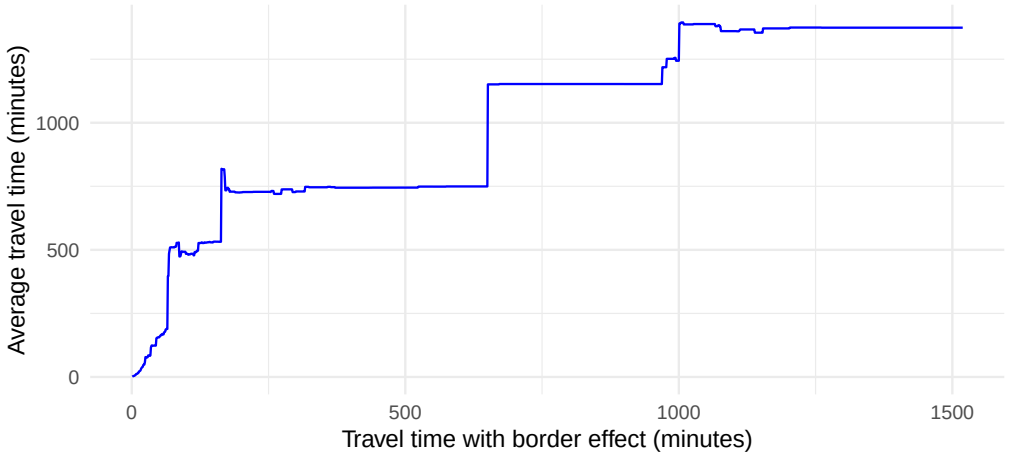

Figure 11: Relationship between the average travel time with border effect (minutes) and the percolation parameter.

arise. Note that the average travel time is only computed for existing paths (hence the `unconnected=TRUE` parameter in the `mean_distance()` function). The sudden changes in the average travel time and the sudden changes in the size of the GCC happen for the same values of the percolation parameter. For example, there is a large increase in the average travel time as the percolation parameter decreases below approximately 650 minutes. This suggests that the two parts of the network that were connected for higher values of the percolation parameter become unconnected for values below 650 minutes as a considerable number of edges get removed. As a result, the average travel time increases since there are possibilities to travel further.

6 Conclusion

This article provides an introduction of key concepts, definitions, and applications of network analysis for the study of geographic data, as well as instructions for the practical implementation of network analysis in R. The approach presented here offers deeper insights into how location, distance, and spatial distribution impact the behaviour of systems where connections are important by integrating geographic data with network analysis methods.

The versatility of these methods highlights their potential for application across diverse geographic contexts. While the African road network serves as a specific example, the underlying principles can be adapted to different contexts and types of networks. This adaptability makes it possible to apply network analysis to various scales, from small scale street networks (Ma et al. 2024), to intraurban transport networks (Zhong et al. 2014), inter-regional (Arcaute et al. 2016), or global scale spatial networks (Colizza et al. 2006).

The methods presented here will serve as a foundational resource for the study of geographic data as the field of spatial network analysis continues to evolve. The integration of theoretical concepts with practical, accessible tools ensures that researchers and practitioners can apply these techniques to address the current challenges. Ultimately, this work contributes to the ongoing development of more robust, efficient, and sustainable networks in an increasingly interconnected world.

References

Aldous JM, Wilson RJ (2000) *Graphs and Applications*. Springer London. CrossRef

Anselin L (1988) *Spatial Econometrics: Methods and Models*. Springer Netherlands. CrossRef

Arcaute E, Molinero C, Hatna E, Murcio R, Vargas-Ruiz C, Masucci AP, Batty M (2016) Cities and regions in Britain through hierarchical percolation. *Royal Society Open Science* 3: 150691. CrossRef

Barabási A, Pósfai M (2016) *Network science*. Cambridge University Press. http://barabasi.com/networksciencebook/

Barthelemy M (2018) *Morphogenesis of Spatial Networks*. Lecture Notes in Morphogenesis. Springer International Publishing. CrossRef

Barthelemy M, Boeing G (2024) A review of the structure of street networks. *Findings*. CrossRef

Barthélemy M (2011) Spatial networks. *Physics Reports* 499: 1–101. CrossRef

Batty M (2007) *Cities and Complexity: Understanding Cities with Cellular Automata, Agent-Based Models, and Fractals*. The MIT Press

Bettencourt L (2021) *Introduction to Urban Science*. The MIT Press. CrossRef

Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008: P10008. CrossRef

Bollobás B (2001) The evolution of random graphs – The giant component. In: Bollobás B (ed), *Random Graphs*. Cambridge University Press, 130–159. CrossRef

Cabrera-Arnau C, Zhong C, Batty M, Silva R, Kang SM (2023) Inferring urban polycentricity from the variability in human mobility patterns. *Scientific Reports* 13. CrossRef

Colizza V, Barrat A, Barthélemy M, Vespignani A (2006) The role of the airline transportation network in the prediction and predictability of global epidemics. *Proceedings of the National Academy of Sciences* 103: 2015–2020. CrossRef

Curtin KM (2007) Network analysis in geographic information science: Review, assessment, and projections. *Cartography and Geographic Information Science* 34: 103–111. CrossRef

Elekes Z, Tóth G, Eriksson R (2024) Regional resilience and the network structure of inter-industry labour flows. *Regional Studies* 58: 2307–2321. CrossRef

Flake GW (1998) *The computational beauty of nature*. MIT Press, Cambridge, MA, USA

Fortunato S, Newman MEJ (2022) 20 years of network community detection. *Nature Physics* 18: 848–850. CrossRef

Franklin RS (2020) Geographical analysis at midlife. *Geographical Analysis* 53: 47–60. CrossRef

Girvan M, Newman MEJ (2002) Community structure in social and biological networks. *Proceedings of the National Academy of Sciences* 99: 7821–7826. CrossRef

Guimerà R, Amaral LAN (2004) Modeling the world-wide airport network. *The European Physical Journal B - Condensed Matter* 38: 381–385. CrossRef

Hartshorne R (1939) The nature of geography: A critical survey of current thought in the light of the past. *Annals of the Association of American Geographers* 29: 173. CrossRef

Kolaczyk ED, Csárdi G (2014) *Statistical Analysis of Network Data with R*. Use R! Springer New York. CrossRef

Lawrence J, Blackett P, Cradock-Henry NA (2020) Cascading climate change impacts and implications. *Climate Risk Management* 29: 100234. CrossRef

Linard C, Gilbert M, Snow RW, Noor AM, Tatem AJ (2012) Population distribution, settlement patterns and accessibility across Africa in 2010. *PLoS ONE* 7: e31743. CrossRef

Ma D, He F, Yue Y, Guo R, Zhao T, Wang M (2024) Graph convolutional networks for street network analysis with a case study of urban polycentricity in Chinese cities. *International Journal of Geographical Information Science* 38: 931–955. CrossRef

Maier G (2014) OpenStreetMap, the Wikipedia map. *REGION* 1: R3–R10. CrossRef

Miller HJ (2017) Geographic information science II: Mesogeography. *Progress in Human Geography* 42: 600–609. CrossRef

Miller HJ, Clifton K, Akar G, Tufte K, Gopalakrishnan S, MacArthur J, Irwin E (2021) Urban sustainability observatories: Leveraging urban experimentation for sustainability science and policy. *Harvard Data Science Review*. CrossRef

Molloy M, Reed B (1995) A critical point for random graphs with a given degree sequence. *Random Structures and Algorithms* 6: 161–180. CrossRef

Moriconi-Ebrard F, Harre D, Heinrigs P (2016) *Urbanisation Dynamics in West Africa 1950–2010*. CrossRef

Nelson T, Frazier AE, Kedron P, Dodge S, Zhao B, Goodchild M, Murray A, Battersby S, Bennett L, Blanford JI, Cabrera-Arnau C, Claramunt C, Franklin R, Holler J, Koylu C, Lee A, Manson S, McKenzie G, Miller H, Oshan T, Rey S, Rowe F, Şalap Ayça S, Shook E, Spielman S, Xu W, and JW (2025) A research agenda for GIScience in a time of disruptions. *International Journal of Geographical Information Science* 39: 1–24. CrossRef

OpenStreetMap Contributors (2023) Openstreetmap. https://www.openstreetmap.org/ [accessed 30-08-2023]

O'Sullivan D, Manson SM (2015) Do physicists have geography envy? And what can geographers learn from it? *Annals of the Association of American Geographers* 105: 704–722. CrossRef

Pons P, Latapy M (2005) Computing communities in large networks using random walks. In: Yolum p, Güngör T, Gürgen F, Özturan C (eds), *Computer and Information Sciences - ISCIS 2005*. Springer, Berlin, Heidelberg, 284–293. CrossRef

Prieto Curiel R, Cabrera-Arnau C, Bishop SR (2022) Scaling beyond cities. *Frontiers in Physics* 10. CrossRef

Prieto-Curiel R, Heo I, Schumann A, Heinrigs P (2022) Constructing a simplified interurban road network based on crowdsourced geodata. *MethodsX* 9: 101845. CrossRef

Rand WM (1971, December) Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association* 66: 846–850. CrossRef

Rittel HWJ, Webber MM (1973) Dilemmas in a general theory of planning. *Policy Sciences* 4: 155–169. CrossRef

Strano E, Giometto A, Shai S, Bertuzzo E, Mucha PJ, Rinaldo A (2017) The scaling structure of the global road network. *Royal Society Open Science* 4: 170590. CrossRef

Uitermark J, van Meeteren M (2021) Geographical network analysis. *Tijdschrift voor Economische en Sociale Geografie* 112: 337–350. CrossRef

Zhong C, Arisona S, Huang X, Batty M, Schmitt G (2014) Detecting the dynamics of urban structure through spatial network analysis. *International Journal of Geographical Information Science* 28: 2178–2199. CrossRef