distance_matrix_extraction

Marco Giovanni Ferrari Prasiddha Rajaure

2025-03-13

Introduction

This code is meant to perform the calculus of the distance between endpoints of cables. To begin, it is important to notice that each cable is a network made of the nodes and the edges connecting them. Each node might be either a terrestrial landpoint, or a point in which the cable simply divides in two in the middle of the ocean. In Section Structure of the Dataset, a visualization of two cables is provided to offer a clearer idea on how the these work. Given the network oriented structure it is possible to compute the distance (in term of kilometers of cable) between nodes network and obtain the so called "distance matrix". This is a symmetric matrix where rows and columns are the ID of each node and the value at the intersection is the distance between the two. In Section Conversion to Network it is provided an example of this type of data object. To compute the distance between nodes, it is possible to use one of the many algorithms that are able to assess the shortest path between nodes. In this case we use the Djikstra Algorithm. Another possible way to store the distance matrix is in the *long* format, namely in a distance dataset where in the first two columns are stored the endpoints, and in the third the shortest distance between the two. Since we are interested into the distance between countries, we assign to terrestrial landpoints their respective country ID.

It is possible to loop this procedure across cables and compute a set of distance matrices/distance datasets. In this way, we end up with a dataset that contains not just the length in km connecting two countries, but a set of distances, each referring to the path of each cable.

The code is structured then as follows. In the first part — Preliminary Example — we provide an example on how we perform the computation of the distance matrix for a single cable. Then, in the second part — Looping Across Cables — we loop across cables so to obtain the distance dataset. In both parts the code is inline commented to ease the understanding of the process.

To perform this analysis we make use of the following libraries:

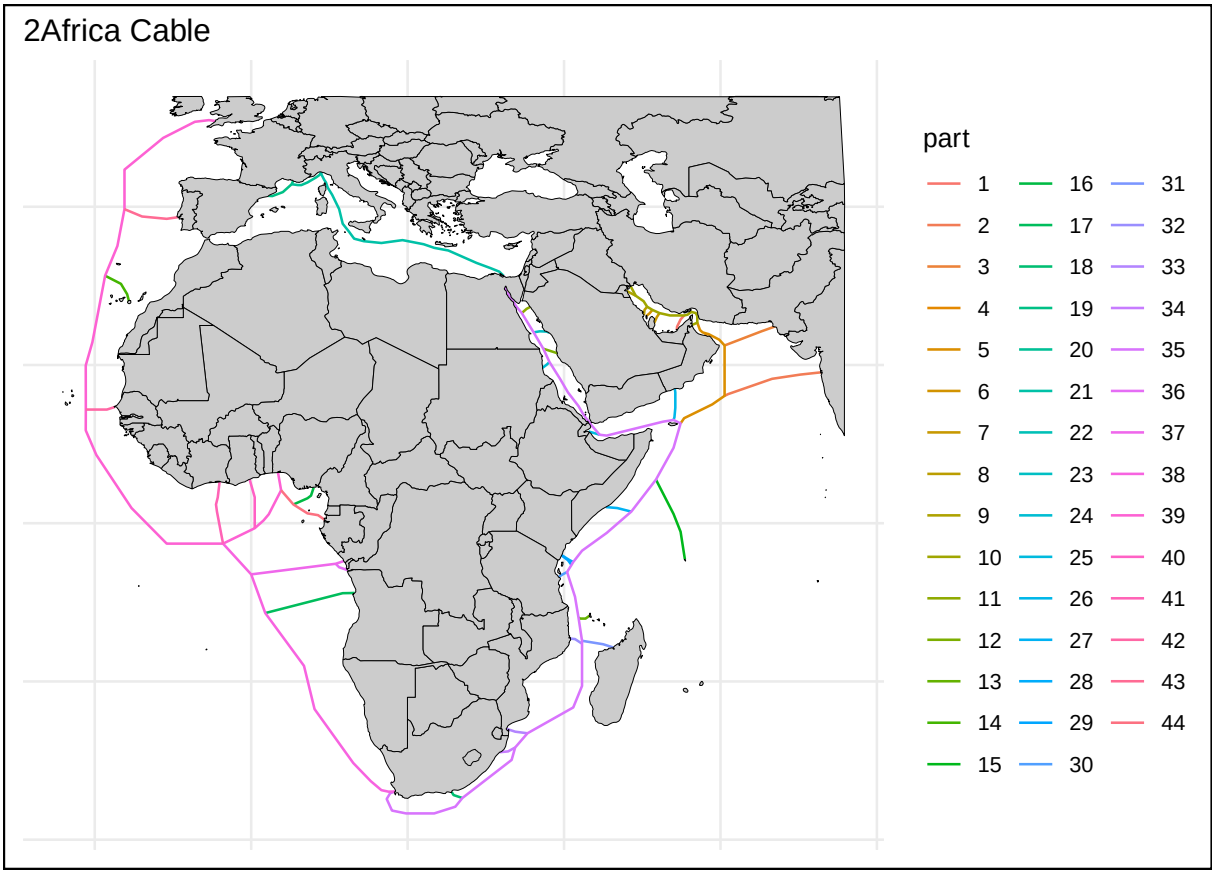
```
library(sf)
library(lwgeom)
library(tibble)
library(dplyr)
library(tidyr)
library(igraph)
library(ggraph)
library(ggplot2)
library(ggpubr)
library(knitr)
library(kableExtra)
library(maps)
library(stringr)
```

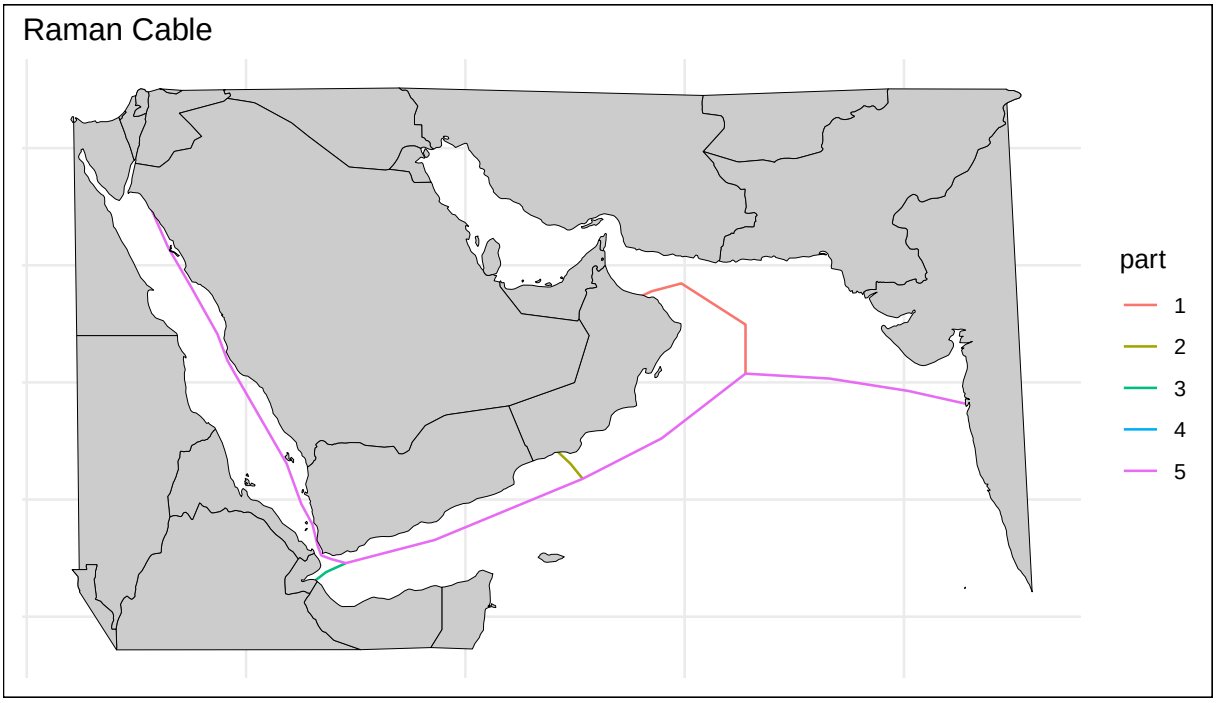
Preliminary Example

We import data on cables from our original shapefile, while world countries are aggregated according to the procedure exposed and performed in file `???`.

Structure of the Dataset

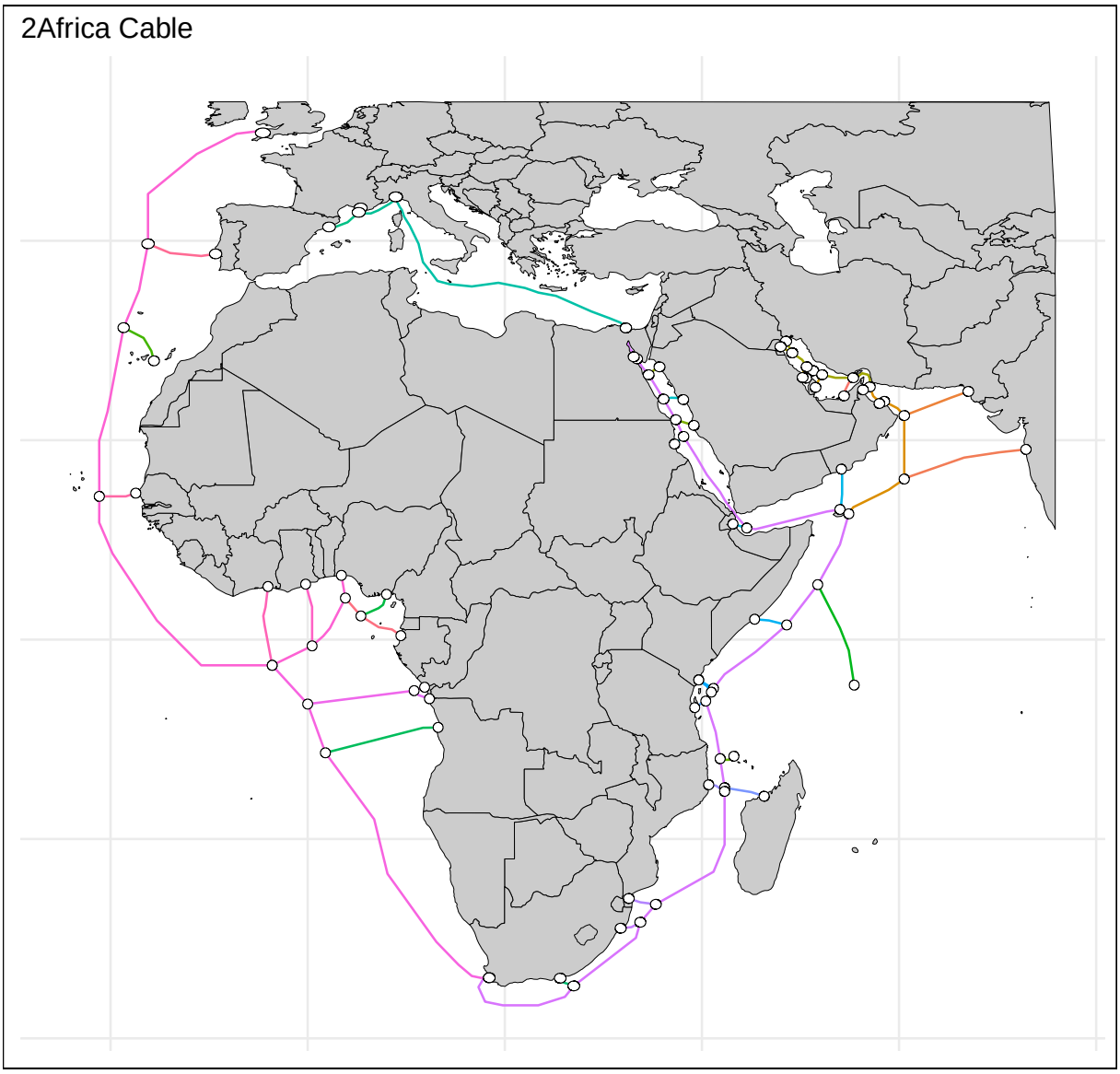
The original shape of cables present some challenges since it is not already provided in the form of edges and nodes. An example is the one presented in the following pictures. As one can observe, the geometry of each cable is not always subdivided correctly in single edges. Some edges do not stop at the nodes but continue longer. For this reason, in the following steps we extract all the intersections and subset the cable at each point. We will also proceed to the identify to which country/territory each node belongs. The rest of the analysis will continue on analyzing the subsea cable "2Africa", the longest subsea cable system in the world.

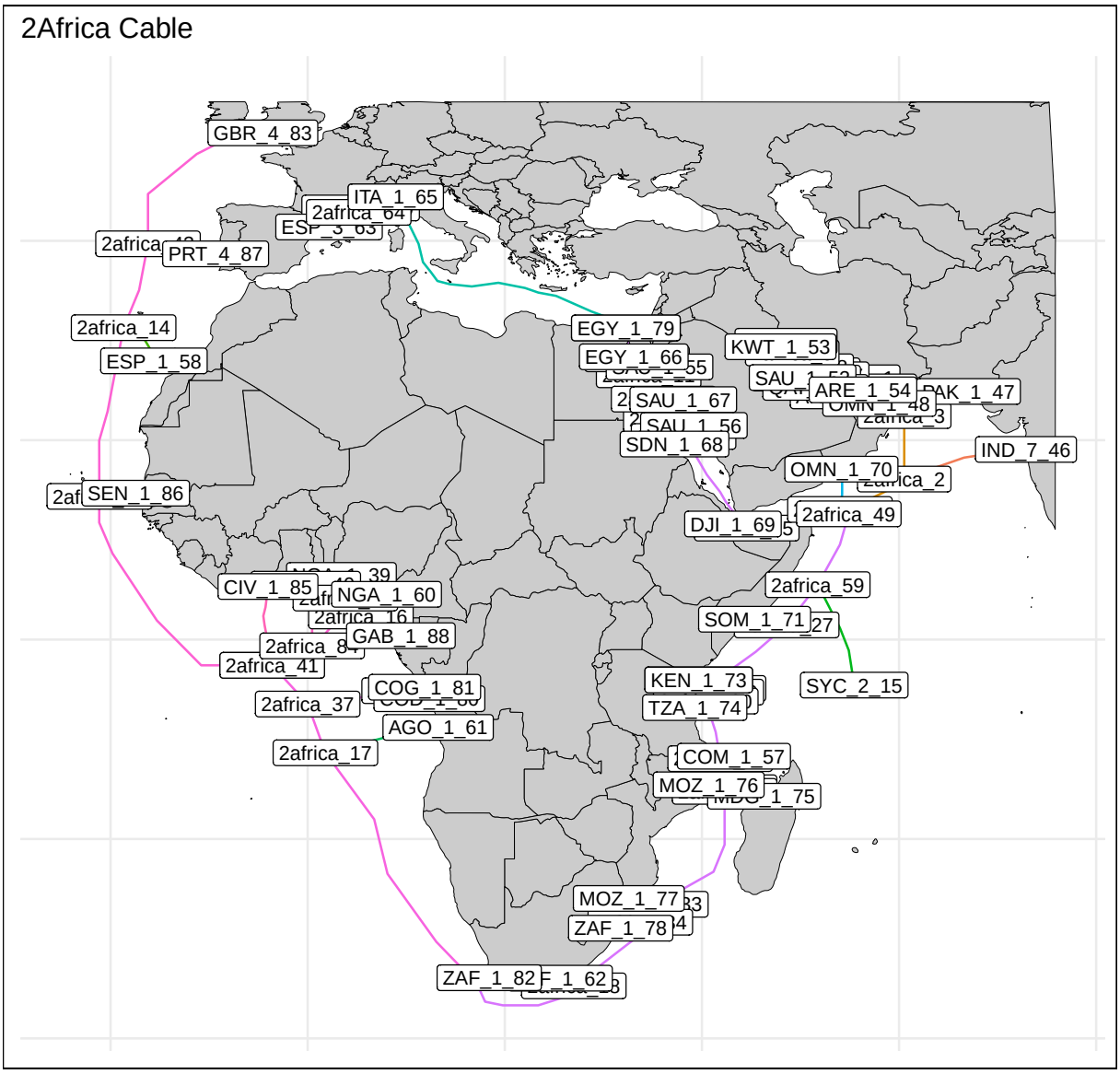

2Africa Cable
part
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44


Raman Cable
part
1
2
3
4
5

Extraction of Endpoints

Each part of each cable has one beginning and one end, we extract those points, store them, and use them to split the entire cable in single edges. We identify separately nodes that are on land and those that are on the sea by creating unique identifiers for each node. For those edges that are on land, we assign a name that is composed by the ID of the territory in which it lands and a second part that uniquely identify the landing point. This last aspect is necessary to consider these two landing points as separate.


2Africa Cable


2Africa Cable
GBR_4_83
ITA_1_65
2africa_64
ESP_3_63
2africa
PRT_4_87
2africa_14
ESP_1_58
EGY_1_79
EGY_1_66
KWT_1_53
SAU_1_67
ARE_1_54
SAU_1_56
SDN_1_68
IND_7_46
OMN_1_70
SEN_1_86
2africa_49
DJI_1_69
CIV_1_85
NGA_1_60
2africa_59
2africa_16
SOM_1_71
GAB_1_88
2africa_41
KEN_1_73
SYC_2_15
COG_1_81
2africa_37
TZA_1_74
AGO_1_61
2africa_17
COM_1_57
MOZ_1_76
MOZ_1_77
ZAF_1_78
ZAF_1_82

Splitting Cables

After having identified the nodes, we now proceed with the splitting. This is naturally the core part of our analysis. Since the code loops across parts, it is preferable to continue the explanation of the code through inline comments.

```
# Creating the empty list of all the edges of each cable
splitted_components  <- list()

# We refer to "parts" as the linestrings that are already provided as separate in
# the original dataset. For each of them, we will check if they cross any node and, if so, split them a

parts <- unlist(sub_cables_single_cable_decomposed$part)

# Looping through each part
for(single_part in parts){
  cable_section <- sub_cables_single_cable_decomposed %>%
    filter(part == single_part)

  # Selecting only those nodes that actually intersect the cable part
  intersecting_idx <- apply(st_intersects(intersections_points_geography,
                                          cable_section, sparse = FALSE), 1, any)

  actual_intersections <- intersections_points_geography[intersecting_idx, ]

  # Storing separately the intersection that we will use for splitting the cables
  actual_intersections_for_split <- actual_intersections

  # Nodes are not just points but 2D circles, this "buffer" around each node is
  # necessary since the geometries are not perfectly accurate and the split would
  # not work correctly otherwise. This polygons are circles of 5 Km in radius (see
  # previous chunk in when assigning "intersection_points"), and has a marginal effect
  # on the final length of the cable.

  # Note also that this implies that nodes that are within 5 km from the coast are
  # authomatically classified as land nodes. This is, again, to compensate the
  # imperfections of the original dataset.

  splitted <- st_split(cable_section, actual_intersections_for_split) %>%
    st_collection_extract("LINESTRING") %>%
    mutate(id = as.factor(1:nrow(.)))

  # Since we are splitting lines through circles, the result is that the
  # split creates sections of the cables that "lie" within the nodes.
  # We obviously want to exclude them so to consider only the edges that are
  # external to the nodes. In doing so we expand by 2 km the radius of
  # nodes and remove all the sections that lies ENTIRELY within these
  # these nodes of diameter 5+2 km

  actual_intersections_for_refine <- actual_intersections %>%
    st_transform(3857) %>%
    st_buffer(2000) %>%
    st_transform(4326) %>%
    st_wrap_dateline(options = c("WRAPDATELINE=YES"))
```

```r
  # Removing parts of cables that lie within the intersection point
  to_remove <- list()
  ids <- unlist(splitted$id)

  for(id_sect in ids){
    cable_part <- splitted %>% filter(id == id_sect)

    within_check <- rowSums(st_within(cable_part,
                                      actual_intersections_for_refine,
                                      sparse = FALSE)) > 0

    if(any(within_check)){
      to_remove[[as.character(id_sect)]] <- id_sect
    }
  }

  splitted <- splitted %>% filter(!id %in% unlist(to_remove))

  # To transform the cable geometry into a network oriented dataset we extract the
  # start and endpoint of each splitted part.

  splitted <- splitted %>%
    rowwise() %>%
    mutate(
      start_point = st_startpoint(geometry),
      end_point = st_endpoint(geometry))

  # Then, we compute the node that is closest to the start/endpoint in the graph,
  # which is naturally the one what has performed the split. We assign them to the
  # variables "start_id" and "end_id".

  splitted <- splitted %>% rowwise() %>%
    mutate(
      start_id = list(actual_intersections_for_split$node_id[order(st_distance(start_point, actual_inte
      end_id = list(actual_intersections_for_split$node_id[order(st_distance(end_point, actual_intersec

  splitted_components[[as.character(single_part)]] <- splitted
}

splitted_components <- do.call(bind_rows, splitted_components)

splitted_components <- splitted_components %>% mutate(section_id = paste(id, part, sep = "_"))


P5 <- ggplot() +
  geom_sf(data = splitted_components, aes(geometry = geometry, color = section_id),show.legend  =FALSE
  geom_polygon(data = world_map_cropped, aes(x = long, y = lat, group = group),
               fill = "gray80", color = "black", size = 0.1) +
  geom_sf(data = intersections_points_geography %>% st_buffer(50000), aes(geometry = geometry, fill = ty
  geom_sf_label(data = splitted_components %>% st_centroid(),
               aes(label =section_id), size=1.5, fill = "white") +
  scale_color_discrete() +
  theme_minimal() +
```

```
  labs(
    title = "2Africa Cable",
  ) +
  theme(axis.line = element_blank(),
        axis.text = element_blank(),
        axis.title = element_blank(),
        plot.background = element_rect(color = "black"))

P5
```

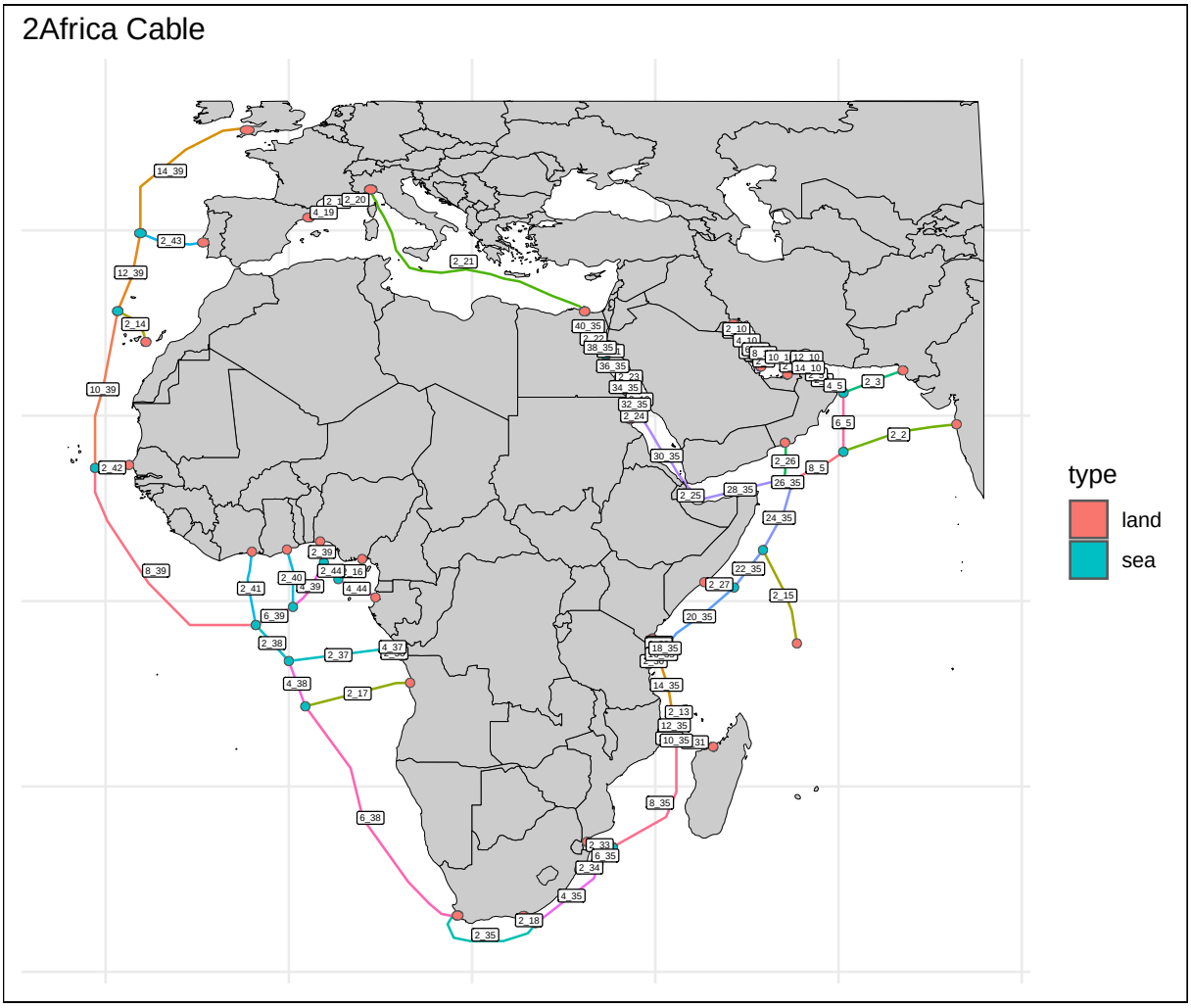

2Africa Cable
type
land
sea

Table 1: Cable Network Data

start_id	end_id	length
2africa_1	ARE_1_45	218423.88 [m]
2africa_2	IND_7_46	1346363.47 [m]
2africa_3	PAK_1_47	705598.99 [m]
2africa_4	OMN_1_48	51507.42 [m]
2africa_5	2africa_4	231027.18 [m]
2africa_4	2africa_3	255007.80 [m]
2africa_3	2africa_2	698578.48 [m]
2africa_2	2africa_49	719528.82 [m]
2africa_6	BHR_1_50	100334.07 [m]
2africa_7	QAT_1_51	148025.52 [m]

Table 2: Distance Matrix (first 10 rows and columns)

	2africa_1	2africa_2	2africa_3	2africa_4	2africa_5	2africa_6	2africa_7	2africa_8	2africa_9	IRQ_1_10
2africa_1	0.0	1483103.2	784524.8	529517.0	298489.8	400113.7	430912	9.68471485.58	677410.2	812368.2
2africa_2	1483103.2	0.0	698578.5	953586.3	1184613.4	1883216.9	1792232.9	2954588.82	2160513.4	2295471.5
2africa_3	784524.8	698578.5	0.0	255007.8	486035.0	1184638.5	1093654.4	5256010.35	1461934.9	1596893.0
2africa_4	529517.0	953586.3	255007.8	0.0	231027.2	929630.7	0838646.6	51001002.55	1206927.1	1341885.2
2africa_5	298489.8	1184613.4	486035.0	231027.2	0.0	698603.5	2607619.4	7769975.37	975900.0	1110858.0
2africa_6	400113.7	1883217.0	1184638.5	929630.7	698603.5	0.00	90984.05	71371.85	277296.4	412254.5
2africa_7	309129.7	1792232.9	1093654.4	838646.7	607619.5	90984.05	0.00	162355.90	368280.5	503238.5
2africa_8	471485.6	1954588.8	1256010.4	1001002.6	769975.4	71371.85	162355.90	0.00	205924.6	340882.7
2africa_9	677410.2	2160513.4	1461934.9	1206927.1	975900.0	277296.4	4368280.49	205924.59	0.0	134958.1
IRQ_1_10	812368.2	2295471.5	1596893.0	1341885.2	1110858.0	412254.5	0503238.55	340882.65	134958.1	0.0

Conversion to Network

To compute the shortest paths between nodes, it is necessary to convert the data into a undirected network, where the distance between nodes is represented by the length of the cable connecting them. We obtain a table as follows:

```
# Computing the distance between nodes
splitted_components <- splitted_components %>%
  mutate(length = st_length(geometry))

# Network-oriented dataset
cable_network <- splitted_components %>%
  select(start_id,  end_id, length) %>%
  st_drop_geometry() %>% unnest()

# Print as a kable table
cable_network %>%
  head(10) %>%
  kable(format = "latex", digits = 2, caption = "Cable Network Data") %>%
  kable_styling(full_width = TRUE, bootstrap_options = c("striped", "hover", "condensed", "responsive"))
```

A last manipulation require to restrict the distance matrix only to country-to-country observations. To do so, we use the way in which we constructed the IDs of the nodes. Those that refer to landpoints are in the form of "ITA_2_34", where the first three letters are the ISO3 code of the geopolitical belonging of territories, the second number identifies to which territory belong the observation (e.g., ITA_1 is Italy

mainland, while ITA_2 is Sardinia), finally the third number uniquely identify the landpoint in that territory (e.g., ITA_2_* are the IDs of all the landpoints in Sardinia, Italy). We therefore proceed by operating the following manipulations:

1. We remove all "sea" nodes by removing those nodes that are not in the form ISO3_*_*
2. We aggregate at territorial level. Namely, if a cable touches twice (or more) the same territory we group them and substitute in the distance matrix with a fictional node. The distance between this fictional node and the other external nodes is the lowest one among to the nodes in the group.

```
# Restricting the output to land cables (identified by those nodes with three
# capital letters and an underscore

pattern <- "^[A-Z]{3}_"

row_names <- rownames(distance_matrix)
col_names <- colnames(distance_matrix)

distance_matrix_filtered <- distance_matrix[row_names[str_detect(row_names, pattern)], col_names[str_de

get_nodes_with_same_beginning <- function(node_names) {
  common_ids <- sapply(node_names, function(name) {
    parts <- unlist(strsplit(name, "_"))
    paste(parts[1], parts[2], sep="_")
  })

  grouped_nodes <- split(node_names, common_ids)
  grouped_nodes <- grouped_nodes[sapply(grouped_nodes, length) > 1]

  return(grouped_nodes)
}
  distance_matrix_filtered <- distance_matrix_filtered %>% data.frame()

  node_names <- rownames(distance_matrix_filtered)
  grouped_nodes <- get_nodes_with_same_beginning(node_names)

  for (group_id in names(grouped_nodes)) {
    group <- grouped_nodes[[group_id]]

    # Compute minimum distances
    min_distances_row <- apply(distance_matrix_filtered[group, , drop=FALSE], 2, min)
    min_distances_col <- apply(distance_matrix_filtered[, group, drop=FALSE], 1, min)  %>% append(0)

    # Assigning the additional column
    distance_matrix_filtered <- distance_matrix_filtered %>%
      rbind(min_distances_row)

    rownames(distance_matrix_filtered)[nrow(distance_matrix_filtered)] <- group_id

    distance_matrix_filtered <- distance_matrix_filtered %>%
      cbind(min_distances_col)

    colnames(distance_matrix_filtered)[ncol(distance_matrix_filtered)] <- group_id
```

Table 3: Cable Network Data

	IRQ_1	SYC_2	FRA_14	ITA_1	GHA_1
IRQ_1	0	5068543	10016817.8	9594475.1	16126752
SYC_2	5068543	0	9055360.0	8633017.4	13448800
FRA_14	10016818	9055360	0.0	422342.6	20113569
ITA_1	9594475	8633017	422342.6	0.0	19691227
GHA_1	16126752	13448800	20113569.2	19691226.6	0

```
    # Removing the single columns
  distance_matrix_filtered <- distance_matrix_filtered[!rownames(distance_matrix_filtered) %in% group,
                                                       !colnames(distance_matrix_filtered) %in% group]



  }

  # Removing the extension at the end of IDS
  distance_matrix_filtered <- distance_matrix_filtered %>%
    rename_with(~ str_replace(.x,  "^([^_]+_[^_]+)_.*$", "\\1")) %>%
    mutate(rowname = str_replace(rownames(.),  "^([^_]+_[^_]+)_.*$", "\\1"))

  rownames(distance_matrix_filtered) <- distance_matrix_filtered$rowname

  distance_matrix_filtered <- distance_matrix_filtered %>% select(-rowname)

  distance_dataset <- distance_matrix_filtered %>%
  rownames_to_column("A") %>%
  pivot_longer(-A, names_to = "B", values_to = "distance") %>%
  mutate(cable = "2Africa") %>%
  relocate(cable, .before = distance) %>%
  filter(A < B)

distance_matrix_filtered[1:5,1:5] %>%
  kable(format = "latex", digits = 2, caption = "Cable Network Data") %>%
  kable_styling(full_width = TRUE, bootstrap_options = c("striped", "hover", "condensed", "responsive"))
```

Looping Across Cables

In this last section, we provide the code that loops across cables. Differently from the previous example, we introduce here an adjustment for those cables that cut through the dateline. Since the original dataset cuts in two the cables at the dateline, we reconnect them artificially.

```
cable_names <- unique(sub_cables$name)

distance_datasets <- list()

# Iterate over each cable
for (cable_name in cable_names) {
  sub_cables_single_cable <- sub_cables %>%
    filter(name == cable_name) %>%
    st_transform(4326)
```

```r
sub_cables_single_cable_decomposed <- sub_cables_single_cable %>%
  st_cast(to = "LINESTRING") %>%
  mutate(part = as.factor(1:nrow(.))) %>%
  relocate(part, .before = name)

start_pts <- st_sfc(st_startpoint(sub_cables_single_cable_decomposed$geometry),
                    crs = st_crs(sub_cables_single_cable_decomposed))

end_pts <- st_sfc(st_endpoint(sub_cables_single_cable_decomposed$geometry),
                  crs = st_crs(sub_cables_single_cable_decomposed))

start_sf <- st_sf(seg_id = sub_cables_single_cable_decomposed$id,
                  end_num = "A", geometry = start_pts)

end_sf <- st_sf(seg_id = sub_cables_single_cable_decomposed$id,
                end_num = "B", geometry = end_pts)

intersections_points <- start_sf %>%
  add_row(end_sf) %>%
  st_cast("POINT") %>%
  st_transform(3832) %>%
  st_buffer(5000) %>%
  st_transform(4326) %>%
  st_wrap_dateline(options = c("WRAPDATELINE=YES")) %>%
  mutate(node_id = paste0(seg_id, "_", row_number()))

intersections_points_geography <- intersections_points %>%
  st_join(world_boundaries_single_territories, join = st_intersects) %>%
  mutate(type = as.factor(ifelse(is.na(country_code_n), "sea", "land"))) %>%
  mutate(node_id = ifelse(is.na(country_code_n), node_id, paste0(country_code_n, "_", row(.)))) %>%
  st_wrap_dateline(options = c("WRAPDATELINE=YES"))

splitted_components <- list()

parts <- unlist(sub_cables_single_cable_decomposed$part)

for (single_part in parts) {
  cable_section <- sub_cables_single_cable_decomposed %>% filter(part == single_part)
  intersecting_idx <- apply(st_intersects(intersections_points_geography, cable_section, sparse = FALS
  actual_intersections <- intersections_points_geography[intersecting_idx, ]

  if (nrow(actual_intersections) == 0) {
    splitted_components[[as.character(single_part)]] <- cable_section
    next
  }

  actual_intersections_for_split <- actual_intersections

  actual_intersections_for_refine <- actual_intersections %>%
    st_transform(3832) %>%
    st_buffer(2000) %>%
    st_transform(4326) %>%
    st_wrap_dateline(options = c("WRAPDATELINE=YES"))
```

```
  splitted <- st_split(cable_section, actual_intersections_for_split) %>%
    st_collection_extract("LINESTRING") %>%
    mutate(id = as.factor(1:nrow(.)))

  to_remove <- list()
  ids <- unlist(splitted$id)

  for (id_sect in ids) {
    cable_part <- splitted %>% filter(id == id_sect)
    within_check <- rowSums(st_within(cable_part, actual_intersections_for_refine, sparse = FALSE)) >
    if (any(within_check)) {
      to_remove[[as.character(id_sect)]] <- id_sect
    }
  }

  splitted <- splitted %>% filter(!id %in% unlist(to_remove))

  splitted <- splitted %>%
    rowwise() %>%
    mutate(start_point = st_startpoint(geometry),
           end_point = st_endpoint(geometry))

  splitted <- splitted %>% rowwise() %>%
    mutate(start_id = list(actual_intersections_for_split$node_id[order(st_distance(start_point, actua
           end_id = list(actual_intersections_for_split$node_id[order(st_distance(end_point, actual_in

  splitted_components[[as.character(single_part)]] <- splitted
}

splitted_components <- do.call(bind_rows, splitted_components)

splitted_components <- splitted_components %>% mutate(section_id = paste(id, part, sep = "_"))
splitted_components <- splitted_components %>% mutate(length = st_length(geometry))

cable_network <- splitted_components %>%
    select(start_id, end_id, length) %>%
    st_drop_geometry() %>% as.data.frame()

# Adjusting nodes aroung 180 dayline

lon_threshold <- 1
lat_threshold <- 10

nodes_at_dateline <- intersections_points %>%
  st_centroid() %>%
  st_wrap_dateline(options = c("WRAPDATELINE=YES")) %>%
  mutate(coords = st_coordinates(geometry),
         lon = coords[,1],
         lat = coords[,2]) %>%
  filter(abs(abs(lon) - 180) < lon_threshold) %>%
  select(node_id, lon, lat) %>% st_drop_geometry()

if(length(nodes_at_dateline$node_id) != 0){
```

```
    close_pairs <- nodes_at_dateline %>%
        inner_join(nodes_at_dateline, by = character(), suffix = c("_1", "_2")) %>%
        filter(node_id_1 != node_id_2,
               abs(abs(lon_1) - abs(lon_2)) < lon_threshold,
               abs(lat_1 - lat_2) < lat_threshold) %>%
        distinct(node_id_1, node_id_2)

    nodes_at_dateline_df <- data.frame(
    start_id = close_pairs$node_id_1,
    end_id = close_pairs$node_id_2,
    length = 0
  )

    cable_network <- cable_network %>%
      rbind(nodes_at_dateline_df) %>%
      rename(from = start_id, to = end_id)
  }

  cable_network <- graph_from_data_frame(cable_network, directed = FALSE)

  distance_matrix <- distances(cable_network,
                               to = V(cable_network),
                               mode = "out", weights = E(cable_network)$length)

# Restricting the output to land cables (identified by those nodes with three
# capital letters and an underscore

pattern <- "^[A-Z]{3}_"

row_names <- rownames(distance_matrix)
col_names <- colnames(distance_matrix)

distance_matrix_filtered <- distance_matrix[row_names[str_detect(row_names, pattern)], col_names[str_de

get_nodes_with_same_beginning <- function(node_names) {
  common_ids <- sapply(node_names, function(name) {
    parts <- unlist(strsplit(name, "_"))
    paste(parts[1], parts[2], sep="_")
  })

  grouped_nodes <- split(node_names, common_ids)
  grouped_nodes <- grouped_nodes[sapply(grouped_nodes, length) > 1]

  return(grouped_nodes)
}
  distance_matrix_filtered <- distance_matrix_filtered %>% data.frame()

  # Skipping to the next cable if connects just one country
  if(nrow(distance_matrix_filtered) == 0){next}

  node_names <- rownames(distance_matrix_filtered)
  grouped_nodes <- get_nodes_with_same_beginning(node_names)
```

```r
  for (group_id in names(grouped_nodes)) {
    group <- grouped_nodes[[group_id]]

    # Compute minimum distances
    min_distances_row <- apply(distance_matrix_filtered[group, , drop=FALSE], 2, min)
    min_distances_col <- apply(distance_matrix_filtered[, group, drop=FALSE], 1, min)  %>% append(0)

    # Assigning the additional column
    distance_matrix_filtered <- distance_matrix_filtered %>%
      rbind(min_distances_row)

    rownames(distance_matrix_filtered)[nrow(distance_matrix_filtered)] <- group_id

    distance_matrix_filtered <- distance_matrix_filtered %>%
      cbind(min_distances_col)

    colnames(distance_matrix_filtered)[ncol(distance_matrix_filtered)] <- group_id

    # Removing the single columns
  distance_matrix_filtered <- distance_matrix_filtered[!rownames(distance_matrix_filtered) %in% group,
                                                       !colnames(distance_matrix_filtered) %in% group]


  }

  # Skipping to the next cable if connects just one country
  if(length(distance_matrix_filtered) == 1){
    if(distance_matrix_filtered == 0){next}}

  # Removing the extension at the end of IDS
  distance_matrix_filtered <- distance_matrix_filtered %>%
    rename_with(~ str_replace(.x,  "^([^_]+_[^_]+)_.*$", "\\1")) %>%
    mutate(rowname = str_replace(rownames(.),  "^([^_]+_[^_]+)_.*$", "\\1"))

  rownames(distance_matrix_filtered) <- distance_matrix_filtered$rowname

  distance_matrix_filtered <- distance_matrix_filtered %>% select(-rowname)

  distance_dataset <- distance_matrix_filtered %>%
  rownames_to_column("A") %>%
  pivot_longer(-A, names_to = "B", values_to = "distance") %>%
  mutate(cable = cable_name) %>%
  relocate(cable, .before = distance) %>%
  filter(A < B)

  distance_datasets[[cable_name]] <- distance_dataset
}

distance_datasets <- do.call(bind_rows, distance_datasets)

saveRDS(distance_dataset, file = "work_data/world_data_infrastructure/distance_dataset.rds")
```