# Political Factions

Subtitle

## Contents



## 1 Introduction

In this short analysis, we use the open data from the Chamber of the Deputies of the Italian Parliament to investigate the legislative co-sponsorship of the members of the Chamber of the Deputies. The analysis would cover the legislative process that ensued under Conte I and Conte II cabinets within the legislature XVIII of Italy.

The analysis aims to understand the presence of political factions underlying the legislative process in the Lower chamber of the Italian Parliament by employing social network analysis techniques. Moreover, we will analyse the cooperative behaviour of the Parliament parties computing the *intra-opposition party bill differentiation*, as shown by De Giorgi & Dias (2018).

Lastly, this project also strives to create a fully reproducible workflow documented, commented and hosted in a public Github repository.

## 2 Methods

The analysis is based on the research produced by De Giorgi and Dias; the study analysed the network created by the co-sponsorship of legislative acts during X and Y cabinets, in office, respectively, from 1900 to 1910 and 2016 to 2000.

The study does not provide the code used to produce the analysis nor define the software used. For these reasons, the analysis has been reproduced *ex nihilo* using R statistical computing language and its packages, combined with the provided documentation.

The data has been collected using `dati.camera.it` relying on a Virtuoso Endpoint via SPARQL language. The queries were prompted through `SPARQL` packages. The Social Network Analysis was held using `igraph`

package. Data processing and preparation was carried out with a selection of packages from the Tidyverse collections. As a general rule, the code was written using the Tidyverse verbs and syntax.

# 3 Technical overview

## 3.1 Package loading

```
library(here)
library(SPARQL)
library(tidyverse)
library(vroom)
library(stringr)
library(lubridate)
library(ggplot2)
library(igraph)
```

## 3.2 Data collection

The data retrieval consist of two main steps: firstly, we must declare the endpoint to scrape:

```
endpoint <- "http://dati.camera.it/sparql"
```

Since the SNA needs a list of vertices and a list of edges, we scraped the information of the deputies who took office during the selected cabinets and all the bills proposed. SPARQL queries use semantic triples, which are built as a set of three entities `subject-predicate-object`. To makes things more manageable, we firstly declare a query to get the deputies information. To restrict the result only to the 18th legislature, we used the triple `?atto ocd:rif_leg <http://dati.camera.it/ocd/legislatura.rdf/repubblica_18>`.

```
bio <- "
SELECT DISTINCT (CONCAT(?cognome,\" \",?nome) AS ?name) , ?partito, ?s_office, ?e_office
WHERE { ?persona a ocd:deputato;
        foaf:firstName ?nome;
        foaf:surname ?cognome .
          ?persona ocd:rif_mandatoCamera ?mandato .

        #adesione a partito
        ?mandato ocd:rif_deputato ?deputato .
        ?deputato ocd:aderisce ?l .
        ?l rdfs:label ?partito .
    OPTIONAL{?l ocd:startDate ?s_office.}
    OPTIONAL{?l ocd:endDate ?e_office.}

        #restict to 18esima legislatura
        ?mandato ocd:rif_leg <http://dati.camera.it/ocd/legislatura.rdf/repubblica_18> .}
ORDER BY ?name"
```

The query retrieves the deputy's name, the party, and the office's dates. The dates are pivotal since a deputy could change party during the mandate; these changes must be considered when building the social network edges.

Then we could query the database:

```
SPARQL(endpoint, bio)
```

Subsequently, we retrieved the bills taking the number of the bill, the date of the first proposal, the first signatory and the joint signatories. The construction of this query encountered a significant problem: the

endpoint could not provide more than 10000 results at once, and we had to use a `subquery-offset` method to bypass this limitation. To restrict the result only to the Conte I cabinet, we used the triple `?atto ocd:rif_governo <http://dati.camera.it/ocd/governo.rdf/g142>`.

```
query_main <- "
SELECT DISTINCT ?num ?date(CONCAT(?primo_cognome, \" \",?primo_nome) AS ?signatory)
(CONCAT(?altro_cognome,\" \",?altro_nome) AS ?joint_signatory)
WHERE {
 {
  SELECT ?num ?date ?primo_nome ?primo_cognome ?altro_nome ?altro_cognome
  WHERE {
  ?atto a ocd:atto;
          dc:identifier ?num;
          dc:date ?date;
          ocd:rif_leg <http://dati.camera.it/ocd/legislatura.rdf/repubblica_18> ;
          ocd:rif_governo <http://dati.camera.it/ocd/governo.rdf/g142> .

  ?atto ocd:primo_firmatario ?primo .
  ?primo  a ocd:deputato;
            foaf:firstName ?primo_nome;
            foaf:surname ?primo_cognome .

  ?atto ocd:altro_firmatario ?altro .
  ?altro  a ocd:deputato;
            foaf:firstName ?altro_nome;
            foaf:surname ?altro_cognome .
    }
    GROUP BY ?atto
    ORDER BY ?num ?primo_cognome ?altro_cognome
}
  }

LIMIT 10000
OFFSET"
```

Then, we defined the offset limits:

```
query_offset <- c("0", "5000", "10000", "15000",
                  "20000", "25000", "30000", "35000")
```

Lastly, we build a for loop to make consecutive calls to the database; a different offset limit is concatenated to the query for each call. This method permits retrieval of more than 10000 observations:

```
for (i in 1:length(query_offset)) {
  law <- str_c(query_main,
               query_offset[i],
               sep = " ")
  result_law <- SPARQL(endpoint, law)
  df_law <- rbind(df_law, result_law$results)
  Sys.sleep(2)
}
```

Since scraping large sets of data could be burdensome for the host, we set the offset to half of the limits (blocks of 5000 observations each), and after every call, the loop waits 2 seconds.

To retrieve the bills for Conte II cabinet, we used the same query structure, changing the "filter" triple to `?atto ocd:rif_governo <http://dati.camera.it/ocd/governo.rdf/g162` [from `g142` to `g162`].

Each result was exported into a CSV file after the retrieval:

```
write.csv(result1, here::here("data/deputies.csv"))
write.csv(result2, here::here("data/conte_i.csv"))
write.csv(result3, here::here("data/conte_ii.csv"))
```

`here` package provides a relative path to the top-level directory of the project, simplifying the referencing of the data regardless of the OS or the absolute path of the directories.

## 3.3 Data preparation

Since the resulting queries are not ready-made for the subsequent analysis, we must prepare the data. For this purpose, we could define functions for the needed purposes.

### 3.3.1 Deputies preparation

We define the function for preparing the deputies:

```
prep_deputies <- function(deputies, end_date = "2021-12-02") {

  deputies <- deputies %>%

  # removing office term included in the party name var party is provide as
  # "PartyX (1900-00-01-[...])" we need to remove what's after the first "("
    separate(col = partito,
             into = c("party", "trash"),
             sep = "\\(") %>%
    mutate(
      # arty column has space after the party name, then we must trim it!
      party = str_trim(party),

      # parsing offices data into a single interval format
      date = interval(start = ymd(s_office),
                      end = ymd(ifelse(test = is.na(deputies$e_office),
                                       #since the 18th leg is still in office
                                       #some MPs does not have ending date
                                       yes = 20220228,
                                       no = deputies$e_office)))) %>%

    # dropping working variables no more useful
    select(!c(trash, s_office, e_office)) %>%

    # drop deputies which have taken office *AFTER* Conte_ii cabinet
    filter(int_start(date) < lubridate::ymd(end_date))

  #reuturn value
  return(deputies)
}
```

The function's primary purpose is to deal with the query's problematic results. The query returned three columns:

- name of the deputy
- party with which
- start date
- end date

The major problem is the party column because it includes the party name and the terms of the mandate, written between round brackets. With `separate()`, we can split the party name from the terms, which are unused since we have it already. Then, we parse the start and end dates into an interval object.

### 3.3.2 Contributions preparation

```
prep_cabinet <- function(cabinet, cabinet_int, deputies) {
  `%!in%` <- Negate(`%in%`) # "not in" function declaration

  #define cabinet name
  name <- as.character(substitute(cabinet))


   # Due to the construction of dati.camera.it database it's easier preparing
   #data within R environment:
   cabinet <- cabinet %>%

     # removing duplicate rows
     distinct(.) %>%

     # Parsing dates
     mutate(date = ymd(date))

   # Extracting deputies and contributors
   cabinet_deputies <- deputies %>%
     filter(int_start(date) < int_end(cabinet_int)) %>%
     filter(!(int_end(date) < int_start(cabinet_int)))

   # Since some MPs, may have changed party or decayed in between
   # they must be coded as SWITCHER/DECAYED
   is_duplicate <- cabinet_deputies %>%

     # group by MPs names
     group_by(name) %>%

     # n as how many time a unique name appears
     summarise(n = n()) %>%

     # filter only the names who appear more than one time
     filter(n > 1) %>%

     # coding party as "SWITCHER/DECAYED"
     add_column(party = "SWITCHER/DECAYED") %>%

     #selecting only name an party to match the structure of cabinet_deputies
     select(name, party)


   cabinet_deputies <- cabinet_deputies %>%

     # remove switcher from cabinet_deputies with an anti joint fun
     anti_join(is_duplicate, by = "name") %>%

     #drop the date column, no more useful
```

```
    select(!(date)) %>%

    #bind the Switcher to cabinet_deputies
    bind_rows(., is_duplicate)

  #remove eventual orphan nodes
  cabinet_deputies <-  cabinet_deputies %>%
     filter(!(name %!in% c(cabinet$signatory,cabinet$joint_signatory)))

  # since there is MPs no included in the nodes df we must debug it!
  debug <- cabinet[which(!(cabinet$joint_signatory %in% cabinet_deputies$name)), ]
  debug <- unique(debug$joint_signatory)
  cabinet_deputies <- rbind(cabinet_deputies,
                            deputies[deputies$name %in% debug, 1:2])

  #assign the contributions DFs
  assign(name,
         cabinet,
         envir = parent.frame())

  #assign the deputies DFs
  assign(stringr::str_c(name, "_deputies"),
         cabinet_deputies,
         envir = parent.frame())
  }
```

The function consists of two primary purposes: first, removing the duplicates observation on the contributions dataframe. Furthermore, it deals with a significant issue: Italian members of Parliament can change the party during their mandate. This, cause the presence of duplicated names (vertices in the SNA) which must be removed.

The issue has been addressed by filtering for the deputies whose office mandate started *before* the cabinet end date and dropping the deputies whose office mandate started *after* the cabinet end date. Then, the deputies who appear more than once are saved in a temporary object, coding the `party` variables as "SWITCHER/DECAYED". Lastly, the duplicates deputies are removed from the `cabinet_deputies` dataframe using an `anti_join()`, and then they are appended again to the dataframe. This method permits to have solely unique observations in the `cabinet_deputies` dataframe. Hence, the observations that appeared more than once are now unique, with the party coded as "SWITCHER/DECAYED"

To remove eventual orphan nodes, the function removes the observations from `cabinet_deputies` dataframe, whose `name` is not present in the contributions dataframe, `cabinet`. In order to doing so, we used a user-defined operator: `%!in%`[1]. The operator returns the values of $x$ that do not match in $y$. Applied to our case using a negate filter, it filters out all the deputies who are not present in the contributors' dataframe[2].

The function addresses another issue, also; some bills have latecomers associated as joint signatories since the lengthy legislative procedure. Possibly, these latecomers have not taken office under the cabinet in analysis and must be removed. The solution is implemented in a slightly convoluted manner which must be addressed in future releases.

At its last, the function returns the resulting dataframes to the parent environment using the `assign()` verb The name of the returned objects is built upon the dataframe given as the `cabinet` argument to the function.

So:

[1] defined as `Negate(%in%)`

[2] `filter(!(name %!in% c(cabinet$signatory,cabinet$joint_signatory))`

```
# Deputies function
deputies <- prep_deputies(deputies)

# Conte_i cabinet dates
conte_i_date <- interval(ydm("2018-31-05"), ydm("2019-04-09"))

prep_cabinet(cabinet = conte_i, cabinet_int = conte_i_date, deputies = deputies)

# conte_ii cabinet dates
conte_ii_date <- interval(ydm("2019-04-09"), ydm("2021-12-02"))

prep_cabinet(cabinet = conte_ii, cabinet_int = conte_ii_date, deputies = deputies)


#' Since IV pol. group was founded only 10 days after Conte_ii starting date.
#' The function codes IV MPs as switcher.
#' Recognizing the weight of IV MPs we must recode them manually
iv <- deputies %>%
  filter(party == "ITALIA VIVA") %>%
  filter(int_start(date) == ymd("2019-09-19") &
        int_end(date) > int_end(conte_ii_date)) %>%
  select(!(date))

conte_ii_deputies$party <- ifelse(conte_ii_deputies$name %in% iv$name,
                                  "ITALIA VIVA",
                                  conte_ii_deputies$party)
```

#### 3.3.3 Italia Viva

IV challenge:

### 3.4 Social Network Analysis

For the SNA, we first create the edge and vertex dataframes using the defined functions:

## 4 Discussion

```
cosponsor <- function(contributions,
                      deputies, signatory = "signatory",
                      joint_signatory = "joint_signatory",
                      mp_name = "name" ) {
 csp <- contributions %>%
    left_join(deputies,
            by = c(signatory = mp_name )) %>%
    left_join(deputies,
              by = c(joint_signatory = mp_name),
              suffix = c("_main","_joint")) %>%
    group_by(num, party_main) %>%
    summarise(n= n(),
              n_joint = sum(party_joint == party_main)) %>%
    mutate(csp = (n_joint/n)) %>%
    group_by(party_main) %>%
    summarise(csp_avg = mean(csp))
```

```
 return(csp)
}
```

```
csp_i <-  cosponsor(contributions = conte_i,
                    deputies = conte_i_deputies)

csp_2 <-  cosponsor(contributions = conte_ii,
                      deputies = conte_ii_deputies)

csp <- csp_i %>%
  full_join(csp_2,
            by = c("party_main" = "party_main"),
            suffix = c("_i", "_ii")) %>%
  rename(conteI = csp_avg_i,
         conteII = csp_avg_ii,
         party = party_main) %>%
  mutate(across(2:3, round, 3))
```

Table 1: csp value

| party | conteI | conteII |
|---|---|---|
| FORZA ITALIA - BERLUSCONI PRESIDENTE | 0.930 | 0.896 |
| FRATELLI D'ITALIA | 0.948 | 0.916 |
| LEGA - SALVINI PREMIER | 0.985 | 0.977 |
| LIBERI E UGUALI | 0.827 | 0.481 |
| MISTO | 0.816 | 0.156 |
| MOVIMENTO 5 STELLE | 0.959 | 0.840 |
| PARTITO DEMOCRATICO | 0.949 | 0.873 |
| SWITCHER/DECAYED | 0.057 | 0.319 |
| ITALIA VIVA | NA | 0.797 |

## Word count

| Method | koRpus | stringi |
|---|---|---|
| Word count | 1129 | 1129 |
| Character count | 7025 | 7167 |
| Sentence count | 75 | Not available |
| Reading time | 5.6 minutes | 5.6 minutes |

## References

De Giorgi, E., & Dias, A. (2018). Standing apart together? Common traits of (new) challenger parties in the Italian parliament. *Italian Political Science*, *13*(2).