Exploring mobility dynamics in FVG - part 3: communities

Fabio Morea

introduction

Our research on mobility flows aims at partitioning the regional territory into smaller areas, striking a balance between granularity and practicality. Ideally, we want to identify a moderate number of areas that are neither excessively small nor overly large. For instance, having just one or two areas would be of little interest, while having over 50 areas would complicate interpretation and implementation. Therefore, we aim for a partition that yields a manageable number of areas, perhaps between 3 and 50, enabling us to gain meaningful insights into the interconnectedness of our territory while maintaining practicality for analysis and decision-making.

This challenge will be tackled through *network analysis*, specifically as a *community detection* problem.

The primary goal of this notebook is to establish a methodology for partitioning the set of locations into distinct *communities*. Additionally, it aims to conduct comparative analyses across various partitions, including those derived from data spanning different time intervals or originating from diverse groups of individuals.

The analysis is based on the network created in script 2, where mobility flows are encoded in a weighted undirected network. Here the nodes correspond to locations and the edges represent mobility flows. The weight of each edge represents the average daily movement of individuals between locations A and B within the specified time period.

Our ultimate goal is to divide our regional territory into smaller areas based on the mobility flows between different locations.

coding specifications

Network analysis is carried out with `igraph` package. Hierarchical community structure is analysed with `dendextend` package.

A formal approach

In network analysis, a **community** is a fundamental concept referring to subsets of nodes within a network that exhibit a higher degree of interconnectedness among themselves compared to the rest of the network.

A **partition** of the network is set of non-overlapping communities, such that each node belongs to precisely one community. Mathematically, let $G = (V, E)$ represent a network where V denotes the set of vertices (nodes) and E represents the set of edges (connections) between the vertices. A partition of V into k non-overlapping communities can be expressed as $[C_1, C_2, ..., C_k]$ such that $C_i \cap C_j = \emptyset$ and $\bigcup_{i=1}^{k} = G$.

Before exploring methods to identify an optimal partition of our network, it's important to address common issues:

1. **Trivially small (or large) communities**: When communities are either too big or too small they lack meaningful insights. However, single-node communities or small communities may be an intrinsic feature of the network.
2. **Non-valid partitions**: Algorithms may identify communities that have more external connections than internal ones, which is not consistent with the definition of community. Validity should be checked not only as a mean value for the whole network, but also for each indvidual community.
3. **Stochastic factors**: Some algorithms introduce randomness, leading to varying partitions with each run. Such variations may hinder interpretation, especially if the analysis is focused on single nodes (e.g. are locations A and B part of the same group?) and comparison of different partiotns over time.
4. **Dependency on node ordering**: Algorithms should yield consistent results regardless of node ordering, as networks are non-ordered mathematical objects.

Let's break down the requirements and the approach to solving this problem.

1. Non-overlapping and Union equals G: The partitions or communities identified within the network should not overlap, and collectively, they should cover the entire network.

2. Validity: Each partition should exhibit a higher density of connections within the community compared to connections with nodes outside the community.

3. Usefulness for analysis: The number of partitions (k) should strike a balance between being informative and not overwhelming. Having too few partitions (e.g., k=1, k=2, or k=3) might not provide enough granularity for analysis, while having too many partitions (k almost equal to N, where N is the total number of nodes) might not add significant information and could lead to complexity.

4. Well-distributed communities: While it's acceptable to have a giant community and several smaller ones, this should be justified by other configurations failing. Ideally, the communities should be well-distributed in terms of size and should capture different aspects or clusters within the network.

Given these criteria, the problem of identifying the best partition involves searching for configurations that satisfy these constraints and then ranking them based on a suitable index. This allows us to explore different numbers of communities and strike a balance between informativeness and simplicity.

Approach:

1. Community Detection Algorithms: Utilize community detection algorithms such as Walktrap to partition the network into communities. The result is a dendrogram that can be cut at different k ,originating alternative partitions (that are consistent to each other). This allows us to explore different numbers of communities and strike a balance between informativeness and simplicity. The next step is to check if results are *stable*, partitions are all *valid*, and *rank* them.

2. Sensitivity Analysis: Perform sensitivity analysis to assess the robustness of the chosen partition(s) to variations in parameters and algorithms. This ensures that the identified partition(s) are not overly dependent on specific choices. Check if results are stable with respect to network shuffling and steps WT(g,s). If unstable, use CCD and calculate uncertainty coefficient gamma.

3. Evaluation Metrics: Define an evaluation metric or index that captures the fuzziness of the partitions. Mixing Parameter of the wole network is a good choice: quantify the quality of the partitions based on cohesion within communities and separation between communities. Caculate also mixin parameter of each community: a partition is valid if ALL of its communities have a valid mixing parameter

4. Ranking: Rank the partitions based on the evaluation metric chosen. Select the partition(s) with the highest score, indicating the best balance between validity, usefulness, and distribution of communities.

By following this approach, we can systematically identify and rank partitions in the network G based on the specified constraints, ultimately selecting the most suitable partition(s) for analysis.

Load network and explore main features

Loading the data to a network g. NOTA non è pesato!!!!!!!!!!!!!!!!

```
file_name = "mobility_fvg_sample_01.graphml"
g = igraph::read_graph(file_name, format="graphml")
V(g)$str<-strength(g)
print(g)
```

```
IGRAPH f1b4ec4 UNW- 203 13487 --
+ attr: name (v/c), id (v/c), str (v/n), weight (e/n)
+ edges from f1b4ec4 (vertex names):
 [1] Aiello--Amaro      Aiello--Ampezzo    Aiello--Andreis    Aiello--Aquileia
 [5] Aiello--Arba       Aiello--ArtaTerme  Aiello--Attimis    Aiello--Aviano
 [9] Aiello--AzzanoX    Aiello--Bagnaria   Aiello--Basiliano  Aiello--Bertiolo
[13] Aiello--Bicinicco  Aiello--Brugnera   Aiello--Budoia     Aiello--Buja
[17] Aiello--Buttrio    Aiello--Camino     Aiello--CampTap    Aiello--Campoform
[21] Aiello--Caneva     Aiello--Capriva    Aiello--Carlino    Aiello--Casarsa
[25] Aiello--Cassacco   Aiello--Castions   Aiello--Cavasso    Aiello--Cervignano
[29] Aiello--Chions     Aiello--ChiopVisc  Aiello--Cividale   Aiello--Codroipo
+ ... omitted several edges
```

Plot the network using Longitide and Latitude as coordinates

```
# locs <- read_csv('./data/locations.csv') %>%
#   left_join(commdf) %>%
#     filter(!is.na(comm))
# lon_lat = cbind(locs$LON, locs$LAT)
#
# plot(g,vertex.label = NA, layout = lon_lat, vertex.size = sqrt(V(g)$str)/10)
```

Communities - a simple approach

as first test we identify communities using infomap. The result is a *igraph community object*

```
comms <- infomap.community(g)
#comms <- walktrap.community(g)
#comms <-  cluster_louvain(g,resolution = 1.2)
#comms <-  cluster_leiden(g,resolution = 1.2)
```

```
#comms <-  label.propagation.community(g)
g
```

```
IGRAPH f1b4ec4 UNW- 203 13487 --
+ attr: name (v/c), id (v/c), str (v/n), weight (e/n)
+ edges from f1b4ec4 (vertex names):
 [1] Aiello--Amaro      Aiello--Ampezzo    Aiello--Andreis    Aiello--Aquileia
 [5] Aiello--Arba       Aiello--ArtaTerme  Aiello--Attimis    Aiello--Aviano
 [9] Aiello--AzzanoX    Aiello--Bagnaria   Aiello--Basiliano  Aiello--Bertiolo
[13] Aiello--Bicinicco  Aiello--Brugnera   Aiello--Budoia     Aiello--Buja
[17] Aiello--Buttrio    Aiello--Camino     Aiello--CampTap    Aiello--Campoform
[21] Aiello--Caneva     Aiello--Capriva    Aiello--Carlino    Aiello--Casarsa
[25] Aiello--Cassacco   Aiello--Castions   Aiello--Cavasso    Aiello--Cervignano
[29] Aiello--Chions     Aiello--ChiopVisc  Aiello--Cividale   Aiello--Codroipo
+ ... omitted several edges
```

```
table(comms$membership)
```

```
 1  2  3  4  5  6  7  8
43 26 46 41 16 20  5  6
```

```
commdf <- data.frame(loc = comms$name, comm = comms$membership)
commdf %>% head(10)
```

```
         loc comm
1     Aiello    1
2      Amaro    2
3    Ampezzo    2
4    Andreis    3
5   Aquileia    1
6       Arba    3
7  ArtaTerme    2
8    Attimis    4
9     Aviano    3
10   AzzanoX    3
```

Results can be saved as a csv file, and passed on to other types of analysis.

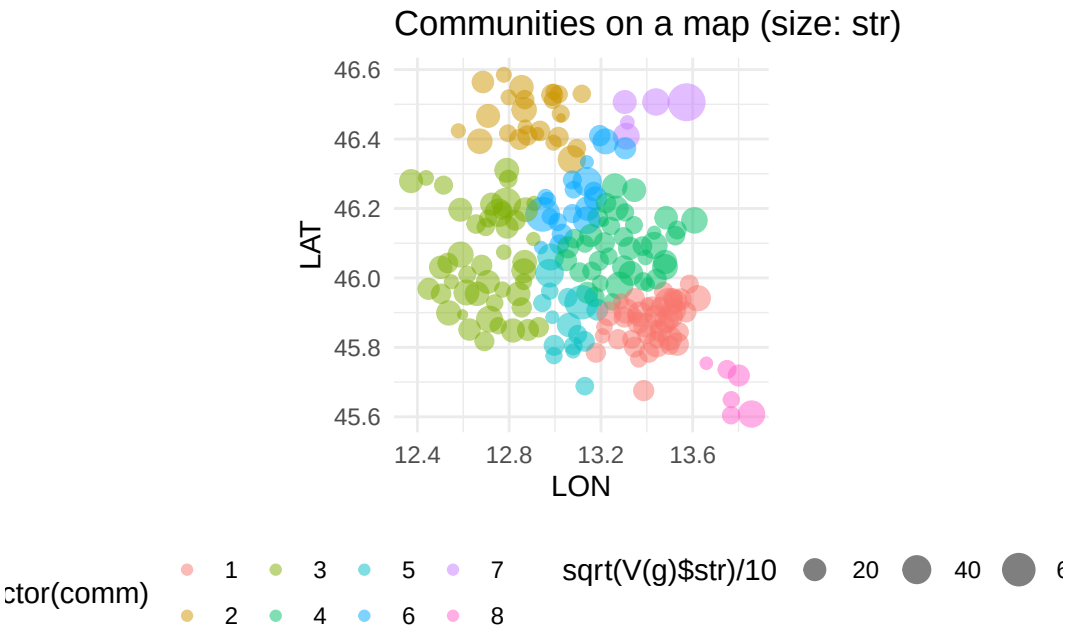
```
commdf %>% write_csv('communities.csv')
```

Example of use: plotting communities

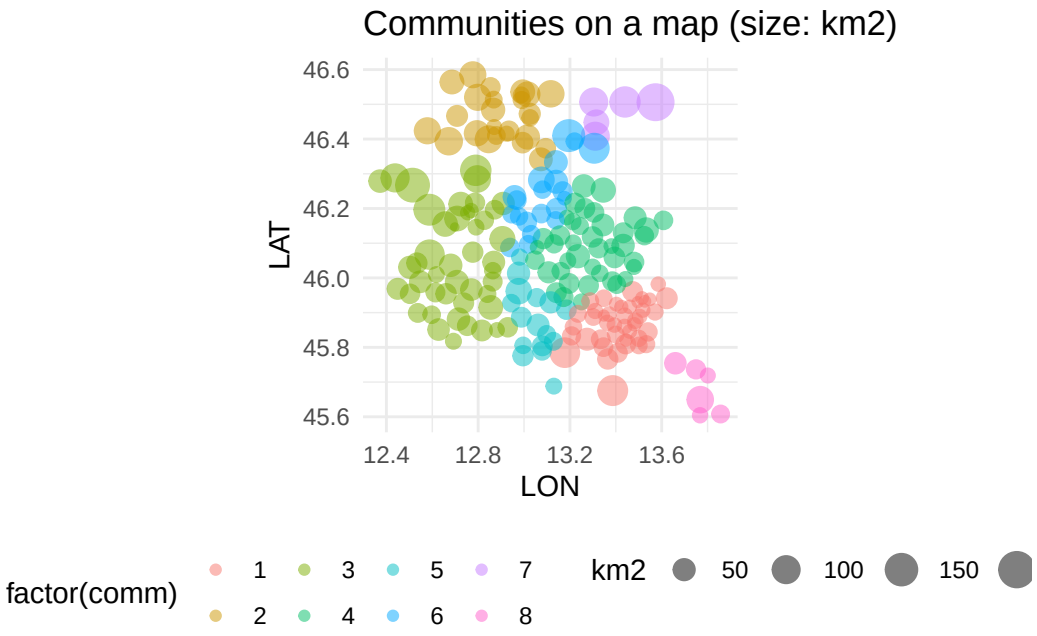
```
locs <- read_csv('./data/locations.csv') %>%
  left_join(commdf) %>%
    filter(!is.na(comm))
```

```
Rows: 203 Columns: 8
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (5): location, loc, prov, codISTAT, codCATASTALE
dbl (3): LAT, LON, km2

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
Joining with `by = join_by(loc)`
```

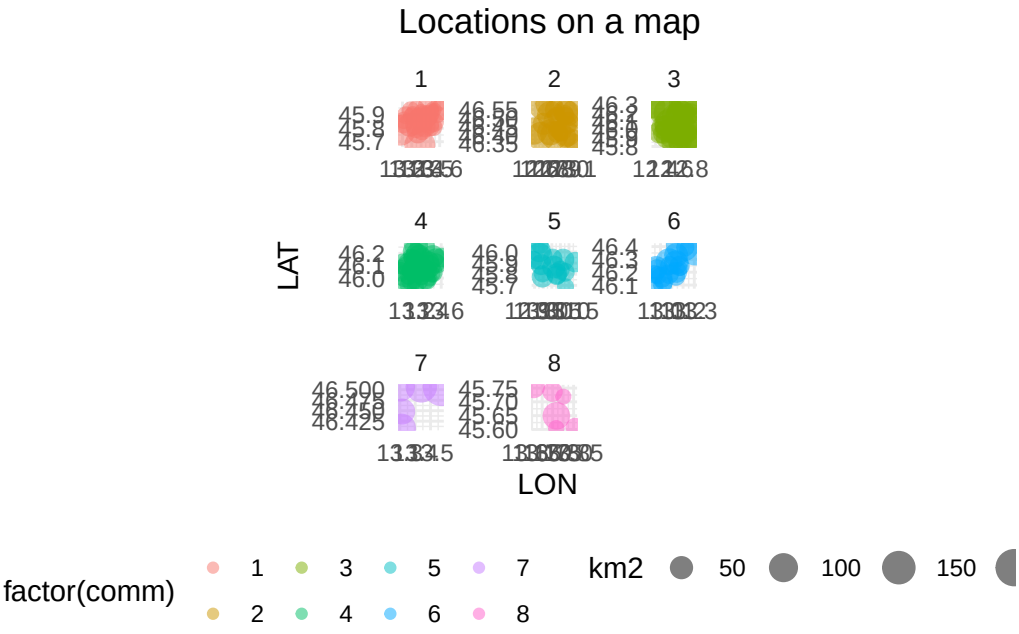
```
locs %>%
  ggplot(aes(x = LON, y = LAT)) +
  geom_point(aes(size = sqrt(V(g)$str)/10, color = factor(comm)), alpha = 0.5) +
  theme_minimal() +
theme(legend.position = 'bottom')+
  theme(aspect.ratio = 1)+
  ggtitle("Communities on a map (size: str)")
```




```
locs %>%
  ggplot(aes(x = LON, y = LAT)) +
  geom_point(aes(size = km2, color = factor(comm)), alpha = 0.5) +
  theme_minimal() +
theme(legend.position = 'bottom')+
  theme(aspect.ratio = 1)+
  ggtitle("Communities on a map (size: km2)")
```




```
locs %>%
  ggplot(aes(x = LON, y = LAT)) +
  geom_point(aes(size = km2, color = factor(comm)), alpha = 0.5) +
  #geom_text(aes(label = loc), nudge_y = 0.1) + # Add annotations
  theme_minimal() +
theme(legend.position = 'bottom')+
  theme(aspect.ratio = 1)+
  ggtitle("Locations on a map")+

  facet_wrap(~ comm, scales = 'free')
```



we can assign each community a name that recalls that of its strongest member. Plot communities coloured and separated using communities::layout_distance_comm

to do: avoid printig comm names; avoid names(mmm)<- V(g)$name

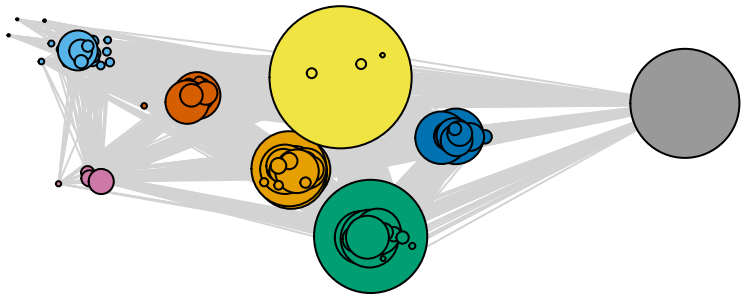
```
V(g)$community <- comms$membership
V(g)$clabels <- communities::comm_label_as_strongest(g, comms)
```

```
[1] "Monfalcone"
[1] "Tolmezzo"
[1] "Pordenone"
[1] "Udine"
[1] "Codroipo"
[1] "Gemona"
[1] "Tarvisio"
[1] "Trieste"
```

```
print(unique(V(g)$clabels))
```

```
[1] "C_Monfalcone" "C_Tolmezzo"   "C_Pordenone"  "C_Udine"      "C_Codroipo"
[6] "C_Gemona"     "C_Tarvisio"   "C_Trieste"
```

```
mmm <- comms$membership
names(mmm)<- V(g)$name
lo = layout.fruchterman.reingold(g,
     weights = communities::layout_distance_comm(g, mmm, eps=.1))

plot(g,
     layout = lo,
     vertex.label = NA,
     vertex.size = sqrt(V(g)$str)/20,
     vertex.color = V(g)$community,
     edge.color = 'lightgray',
     margin = -.1,
     asp = 0.35)
```



we can plot each community separately:

```
list_comms = unique(comms$membership)
for (c in list_comms){
  gcom <- igraph::induced.subgraph(g, vids = V(g)$community == c )
  plot(gcom,
     layout = layout_in_circle(gcom),
```

```
      vertex.label = V(gcom)$name,
      vertex.size = sqrt( V(gcom)$str )/5,
      edge.color = 'lightgray',
      edge.width = E(g)$weight/100,
      main = paste("Community ", V(gcom)$clabels[1]))
}
```

Community C_Tolmezzo

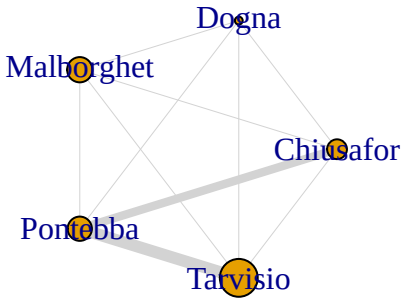
Community C_Pordenone

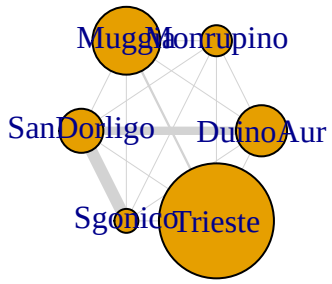
Community C_Udine

Community C_Codroipo

Community C_Gemona

Community C_Tarvisio





to do:

A network of communities

communities as squares, placed in the coordinates of the main city

to do: improve communities::make_community_network
so that weight and community names can be passed as parameters

```
E(g)$w <- E(g)$weight
V(g)$community <-V(g)$clabels
gc <- communities::make_community_network(g)
gc
```

```
IGRAPH f4c0b6f UNW- 8 64 --
+ attr: name (v/c), id (v/n), size (v/n), weight (e/n)
+ edges from f4c0b6f (vertex names):
 [1] C_Codroipo  --C_Codroipo   C_Codroipo  --C_Gemona
 [3] C_Codroipo  --C_Monfalcone C_Codroipo  --C_Pordenone
 [5] C_Codroipo  --C_Tarvisio   C_Codroipo  --C_Tolmezzo
 [7] C_Codroipo  --C_Trieste    C_Codroipo  --C_Udine
```

```
 [9] C_Codroipo  --C_Gemona     C_Gemona    --C_Gemona
[11] C_Gemona    --C_Monfalcone C_Gemona    --C_Pordenone
[13] C_Gemona    --C_Tarvisio   C_Gemona    --C_Tolmezzo
[15] C_Gemona    --C_Trieste    C_Gemona    --C_Udine
+ ... omitted several edges
```

```
plot(gc,
     layout = layout.circle(gc),
     vertex.shape = "square",
     vertex.color = as.factor(V(gc)$name),
     edge.width = log(E(gc)$weight/50),
     margin= -0.1)
```

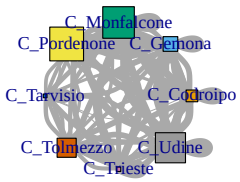


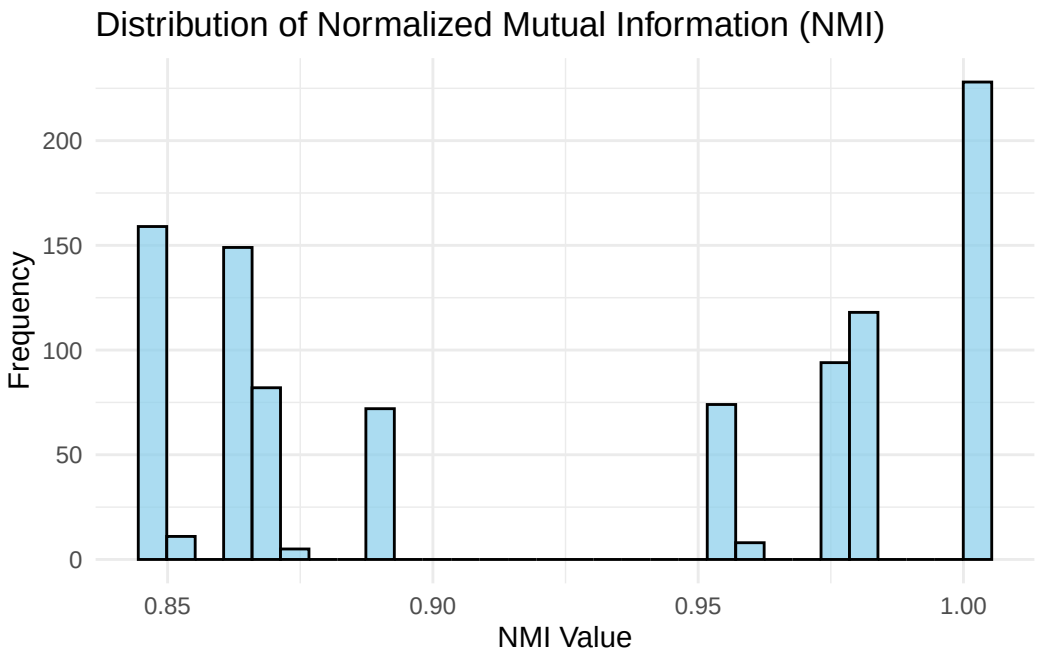
Figure 1: Line Plot 1

check stability

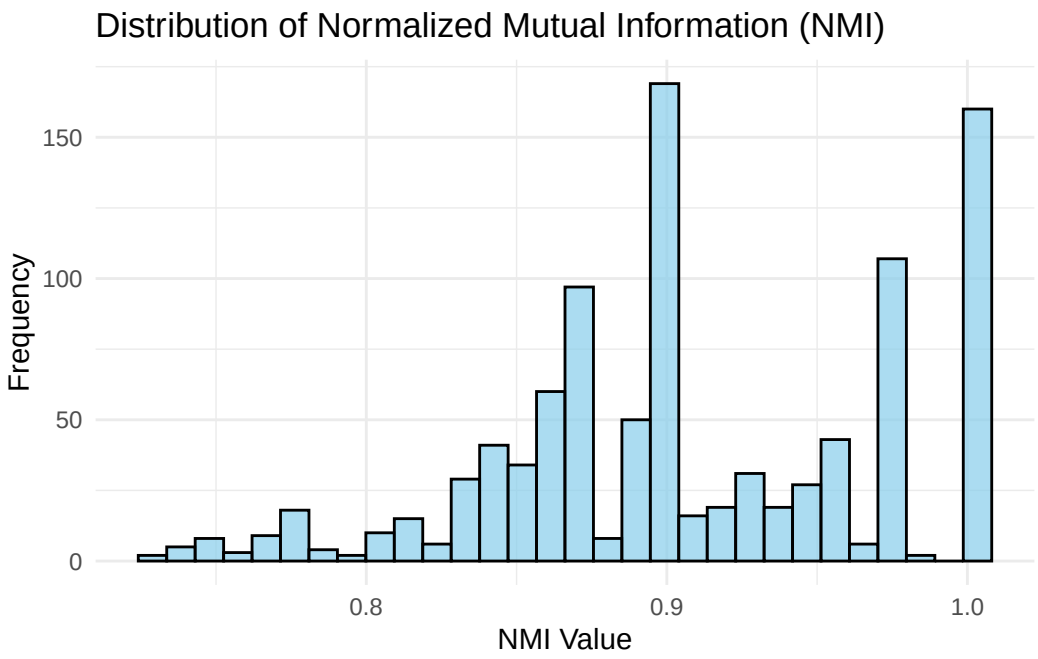
```
t = 100
M <- matrix(NA, nrow = vcount(g), ncol = t)
for (i in 1:t) {M[, i] <- igraph::infomap.community(g)$membership}

M1 <- matrix(NA, nrow = vcount(g), ncol = t)
for (i in 1:t) {M1[, i] <- igraph::cluster_louvain(g)$membership}
```

```
library(aricode)
# Bootstrap approach to compare pairs of columns
bootstrap_iterations <- 1000
bootstrap_stability <- function(bootstrap_iterations, M) {
```

```
  nmi_values <- replicate(bootstrap_iterations, {
    sampled_columns <- sample(ncol(M), replace = TRUE)
    igraph::compare(M[, sampled_columns[1]], M[, sampled_columns[2]], method = "nmi")
  })

  ggplot() +
    geom_histogram(aes(x = nmi_values), bins = 30, fill = "skyblue", color = "black", alph
    labs(title = "Distribution of Normalized Mutual Information (NMI)",
         x = "NMI Value",
         y = "Frequency") +
    theme_minimal()
}

bootstrap_stability(bootstrap_iterations = 1000, M)
```




```
bootstrap_stability(bootstrap_iterations = 1000, M1)
```



Infomap is more stable than louvain.

now check which nodes are more unstable

```
n = vcount(g)
D <- matrix(0, nrow = n, ncol = n)
rownames(D) <- V(g)$name

# Nested for loops to iterate over each pair of nodes
for (i in 1:(n - 1)) {
  for (j in (i + 1):n) {
    # Count how many times nodes i and j are in the same community across all iterations
    count <- sum(M[i,] == M[j,])
    # Store the count in the D matrix
    D[i, j] <- count
    D[j, i] <- count  # Since the pairs are symmetric
  }
}
D<- D/t
D[ D==0 ] <- NA
df <- data.frame(name = rownames(D),
                 s = round(rowMeans(D, na.rm = TRUE),2))
```

```r
df %>% filter(s==1) %>% print
```

```
                name s
Bertiolo     Bertiolo 1
Camino         Camino 1
Castions     Castions 1
Codroipo     Codroipo 1
DuinoAur     DuinoAur 1
Flaibano     Flaibano 1
Latisana     Latisana 1
Lignano       Lignano 1
Monrupino   Monrupino 1
Muggia         Muggia 1
Muzzana       Muzzana 1
Palazzolo   Palazzolo 1
Pocenia       Pocenia 1
Precenicco Precenicco 1
Rivignano   Rivignano 1
SanDorligo SanDorligo 1
Sedeglia     Sedeglia 1
Sgonico       Sgonico 1
Talmassons Talmassons 1
Trieste       Trieste 1
Varmo           Varmo 1
Ronchis       Ronchis 1
```

```r
df %>% filter(s>=0.5 & s <1) %>% print()
```

```
                  name    s
Aiello           Aiello 0.98
Amaro             Amaro 0.95
Ampezzo         Ampezzo 0.95
Andreis         Andreis 0.83
Aquileia       Aquileia 0.98
Arba               Arba 0.74
ArtaTerme     ArtaTerme 0.95
Attimis         Attimis 0.81
Aviano           Aviano 0.83
AzzanoX         AzzanoX 0.83
```

```
Bagnaria       Bagnaria 0.98
Basiliano     Basiliano 0.81
Bicinicco     Bicinicco 0.81
Bordano         Bordano 0.95
Brugnera       Brugnera 0.83
Budoia           Budoia 0.83
Buja               Buja 0.52
Buttrio         Buttrio 0.81
CampTap         CampTap 0.98
Campoform     Campoform 0.81
Caneva           Caneva 0.83
Capriva         Capriva 0.98
Carlino         Carlino 0.98
Casarsa         Casarsa 0.83
Cassacco       Cassacco 0.81
Castelno       Castelno 0.74
Cavasso         Cavasso 0.74
Cervignano   Cervignano 0.98
Chions           Chions 0.83
ChiopVisc     ChiopVisc 0.98
Chiusafor     Chiusafor 0.58
Cimolais       Cimolais 0.64
Cividale       Cividale 0.81
Claut             Claut 0.64
Colloredo     Colloredo 0.52
Comeglians   Comeglians 0.95
Cordenons     Cordenons 0.83
Cordovado     Cordovado 0.83
Cormons         Cormons 0.98
CornodiR       CornodiR 0.81
Coseano         Coseano 0.52
Dignano         Dignano 0.50
Doberdo         Doberdo 0.98
Dogna             Dogna 0.58
Dolegna         Dolegna 0.81
Drenchia       Drenchia 0.81
ErtoCasso     ErtoCasso 0.64
Faedis           Faedis 0.81
Fagagna         Fagagna 0.81
Fanna             Fanna 0.74
FarradIs       FarradIs 0.98
FiumeVen       FiumeVen 0.83
FiumicVV       FiumicVV 0.98
```

```
Fogliano       Fogliano 0.98
Fontanafr     Fontanafr 0.83
Forgaria       Forgaria 0.52
ForniAvol     ForniAvol 0.95
ForniSopra   ForniSopra 0.95
ForniSotto   ForniSotto 0.95
Frisanco       Frisanco 0.74
Gemona           Gemona 0.52
Gonars           Gonars 0.98
Gorizia         Gorizia 0.98
Gradisca       Gradisca 0.98
Grado             Grado 0.98
Lestizza       Lestizza 0.81
Lusevera       Lusevera 0.81
Magnano         Magnano 0.52
Majano           Majano 0.52
Malborghet   Malborghet 0.58
Maniago         Maniago 0.74
Manzano         Manzano 0.81
Mariano         Mariano 0.98
Martignac     Martignac 0.81
Medea             Medea 0.98
Meduno           Meduno 0.74
Mereto           Mereto 0.81
Moimacco       Moimacco 0.81
Monfalcone   Monfalcone 0.98
Montereale   Montereale 0.74
Morsano         Morsano 0.83
Mortegliano Mortegliano 0.81
Moruzzo         Moruzzo 0.81
Mossa             Mossa 0.98
Nimis             Nimis 0.81
Osoppo           Osoppo 0.52
Ovaro             Ovaro 0.95
Palmanova     Palmanova 0.98
Paluzza         Paluzza 0.95
PasianPr       PasianPr 0.81
Pasiano         Pasiano 0.83
Paularo         Paularo 0.95
PaviaUd         PaviaUd 0.81
Pinzano         Pinzano 0.50
Polcenigo     Polcenigo 0.83
Pontebba       Pontebba 0.58
```

```
Porcia           Porcia 0.83
Pordenone     Pordenone 0.83
Porpetto       Porpetto 0.98
Povoletto     Povoletto 0.81
Pozzuolo       Pozzuolo 0.81
Pradaman       Pradaman 0.81
PrataPord     PrataPord 0.83
PratoCarn     PratoCarn 0.95
Pravisdom     Pravisdom 0.83
Premariacco Premariacco 0.81
Preone           Preone 0.95
Pulfero         Pulfero 0.81
Ragogna         Ragogna 0.52
Ravascletto Ravascletto 0.95
Raveo             Raveo 0.95
ReanadR         ReanadR 0.81
Remanzacco   Remanzacco 0.81
Rigolato       Rigolato 0.95
RivedArc       RivedArc 0.52
Romans           Romans 0.98
RonchidL       RonchidL 0.98
Roveredo       Roveredo 0.83
Ruda               Ruda 0.98
SGiorNog       SGiorNog 0.98
SGiorRic       SGiorRic 0.74
SGiovanni     SGiovanni 0.81
SMartino       SMartino 0.74
SPier             SPier 0.98
SPietro         SPietro 0.81
SQuirino       SQuirino 0.83
SVitoTagl     SVitoTagl 0.83
SVitoTorre   SVitoTorre 0.98
Sacile           Sacile 0.83
Sagrado         Sagrado 0.98
Sappada         Sappada 0.95
Sauris           Sauris 0.95
Savogna         Savogna 0.98
Scanzian       Scanzian 0.98
Sdaniele       Sdaniele 0.52
Sequals         Sequals 0.74
SestoR           SestoR 0.83
Spilimber     Spilimber 0.74
Staranzano   Staranzano 0.98
```

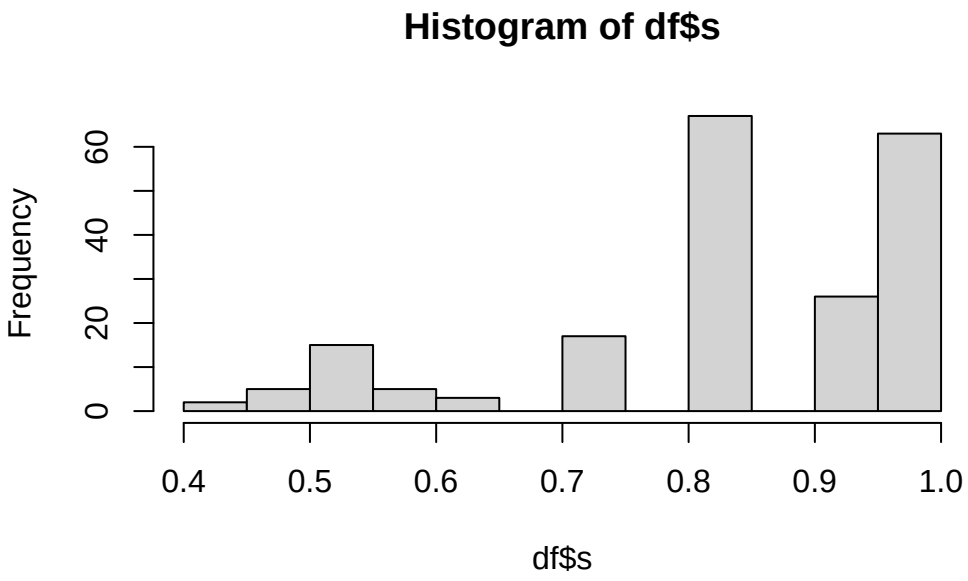
```
Taipana         Taipana 0.81
Tarcento       Tarcento 0.81
Tarvisio       Tarvisio 0.58
Tavagnacco   Tavagnacco 0.81
TerzoAq         TerzoAq 0.98
Tolmezzo       Tolmezzo 0.95
Torreano       Torreano 0.81
Torvisco       Torvisco 0.98
TramSotto     TramSotto 0.74
Trasaghis     Trasaghis 0.52
Travesio       Travesio 0.74
Tricesimo     Tricesimo 0.81
Udine             Udine 0.81
ValvArzene   ValvArzene 0.83
Venzone         Venzone 0.52
Verzegnis     Verzegnis 0.95
VillaSant     VillaSant 0.95
Villesse       Villesse 0.98
Visco             Visco 0.98
Vivaro           Vivaro 0.74
Zoppola         Zoppola 0.83
Zuglio           Zuglio 0.95
Artegna         Artegna 0.52
Enemonzo       Enemonzo 0.95
Sutrio           Sutrio 0.95
Turriaco       Turriaco 0.98
Vajont           Vajont 0.74
VitoAsio       VitoAsio 0.52
Cercivento   Cercivento 0.95
SVitoFaga     SVitoFaga 0.81
TramSopra     TramSopra 0.74
Prepotto       Prepotto 0.81
Moraro           Moraro 0.98
SLeonardo     SLeonardo 0.81
Slorenzo       Slorenzo 0.98
SFloriano     SFloriano 0.98
Lauco             Lauco 0.95
```

```
df %>% filter(s<0.5) %>% print()
```

```
              name    s
```

```
MoggioUd     MoggioUd 0.47
Resia           Resia 0.47
Resiutta     Resiutta 0.47
SMariaLL     SMariaLL 0.43
Trivignavo Trivignavo 0.43
```

```r
hist(df$s)
```




```r
# # Define function to fit 4-point beta distribution
# fit_beta_distribution <- function(data) {
#   require(VGAM)
#   # Fit beta distribution with 4 points
#   fit <- vglm(data ~ 1, betabinomial, trace = TRUE, crit = "coef")
#   return(fit)
# }
# # Identify names with s <= 0.3
# names_below_threshold <- df$name[df$s <= 0.5]
#
# # Initialize a list to store the column indices of D greater than 0 for each name
# columns_greater_than_zero <- list()
#
```

```
# # Iterate over each name below the threshold
# for (name in names_below_threshold) {
#   # Get the row index of the name
#   row_index <- which(rownames(D) == name)
#   # Get column indices where values are greater than 0
#   columns <- which(D[row_index, ] > 0)
#   # Store the result
#   DD <- D[ row_index, columns]
#   print(paste(name, mean(DD)))
#   print("Names of columns with values greater than 0:")
#   #print(rownames(D)[columns])
#
# }
#
#
#
# # Fit beta distribution to DD
# fit <- fit_beta_distribution(DD)
#
# # Print summary of the fitted distribution
# summary(fit)
```

check mixing parameter

```
edges_gc<-as_long_data_frame(gc) %>%
    select(from_name, to_name, weight) %>%
    mutate(intra_weight = if_else(from_name == to_name, 0,weight )) %>%
    mutate(self_weigth = if_else(from_name == to_name, weight, 0 )) %>%
    arrange(-weight)

# mixing parameter of the partition
mu = sum(edges_gc$intra_weight)/sum(edges_gc$weight)
print(edges_gc)
```

```
      from_name      to_name    weight intra_weight self_weigth
1   C_Pordenone  C_Pordenone 1017977.0          0.0   1017977.0
2       C_Udine      C_Udine  830050.5          0.0    830050.5
3  C_Monfalcone C_Monfalcone  578713.5          0.0    578713.5
4     C_Trieste    C_Trieste  271212.0          0.0    271212.0
5  C_Monfalcone      C_Udine  150914.0     150914.0         0.0
```

6	C_Codroipo	C_Codroipo	118502.5	0.0	118502.5
7	C_Gemona	C_Gemona	109206.0	0.0	109206.0
8	C_Monfalcone	C_Trieste	104780.0	104780.0	0.0
9	C_Gemona	C_Udine	98928.0	98928.0	0.0
10	C_Codroipo	C_Udine	87633.5	87633.5	0.0
11	C_Monfalcone	C_Udine	70045.5	70045.5	0.0
12	C_Tolmezzo	C_Tolmezzo	60789.0	0.0	60789.0
13	C_Codroipo	C_Udine	43081.0	43081.0	0.0
14	C_Codroipo	C_Pordenone	41587.5	41587.5	0.0
15	C_Pordenone	C_Udine	39475.0	39475.0	0.0
16	C_Gemona	C_Udine	37201.0	37201.0	0.0
17	C_Monfalcone	C_Trieste	34153.0	34153.0	0.0
18	C_Codroipo	C_Monfalcone	31161.5	31161.5	0.0
19	C_Codroipo	C_Monfalcone	24193.0	24193.0	0.0
20	C_Gemona	C_Pordenone	22875.5	22875.5	0.0
21	C_Codroipo	C_Pordenone	22472.0	22472.0	0.0
22	C_Pordenone	C_Udine	19845.0	19845.0	0.0
23	C_Trieste	C_Udine	16283.5	16283.5	0.0
24	C_Tolmezzo	C_Udine	13391.5	13391.5	0.0
25	C_Trieste	C_Udine	12572.5	12572.5	0.0
26	C_Tarvisio	C_Tarvisio	10987.5	0.0	10987.5
27	C_Monfalcone	C_Pordenone	10970.0	10970.0	0.0
28	C_Gemona	C_Tolmezzo	10870.5	10870.5	0.0
29	C_Gemona	C_Tolmezzo	9300.5	9300.5	0.0
30	C_Pordenone	C_Trieste	8735.5	8735.5	0.0
31	C_Monfalcone	C_Pordenone	8651.0	8651.0	0.0
32	C_Codroipo	C_Trieste	8548.0	8548.0	0.0
33	C_Tolmezzo	C_Udine	8286.0	8286.0	0.0
34	C_Gemona	C_Pordenone	7871.5	7871.5	0.0
35	C_Codroipo	C_Gemona	6681.5	6681.5	0.0
36	C_Tarvisio	C_Udine	5541.0	5541.0	0.0
37	C_Codroipo	C_Gemona	5458.0	5458.0	0.0
38	C_Gemona	C_Monfalcone	3910.5	3910.5	0.0
39	C_Gemona	C_Monfalcone	3642.5	3642.5	0.0
40	C_Gemona	C_Tarvisio	3602.0	3602.0	0.0
41	C_Gemona	C_Tarvisio	3478.5	3478.5	0.0
42	C_Tarvisio	C_Tolmezzo	2468.5	2468.5	0.0
43	C_Gemona	C_Trieste	2415.5	2415.5	0.0
44	C_Tarvisio	C_Udine	2331.5	2331.5	0.0
45	C_Monfalcone	C_Tolmezzo	2167.5	2167.5	0.0
46	C_Tolmezzo	C_Trieste	1938.0	1938.0	0.0
47	C_Pordenone	C_Tolmezzo	1762.0	1762.0	0.0
48	C_Codroipo	C_Tolmezzo	1744.5	1744.5	0.0

```
49   C_Tarvisio    C_Trieste    1570.5       1570.5         0.0
50 C_Monfalcone   C_Tarvisio    1566.5       1566.5         0.0
51  C_Pordenone    C_Trieste    1544.5       1544.5         0.0
52   C_Tarvisio   C_Tolmezzo    1435.0       1435.0         0.0
53  C_Pordenone   C_Tarvisio    1304.5       1304.5         0.0
54 C_Monfalcone   C_Tolmezzo    1114.5       1114.5         0.0
55  C_Pordenone   C_Tolmezzo    1017.0       1017.0         0.0
56   C_Codroipo   C_Tarvisio     787.5        787.5         0.0
57   C_Codroipo    C_Trieste     704.5        704.5         0.0
58   C_Codroipo   C_Tolmezzo     579.5        579.5         0.0
59   C_Tolmezzo    C_Trieste     546.0        546.0         0.0
60     C_Gemona    C_Trieste     527.0        527.0         0.0
61  C_Pordenone   C_Tarvisio     399.0        399.0         0.0
62 C_Monfalcone   C_Tarvisio     321.5        321.5         0.0
63   C_Tarvisio    C_Trieste     134.0        134.0         0.0
64   C_Codroipo   C_Tarvisio     120.5        120.5         0.0
```

```r
print(mu)
```

```
[1] 0.2510295
```

validate for each community: internal connections must be larger than external

```r
names <- unique(c(edges_gc$from_name, edges_gc$to_name))


 comm_df <- tibble(name = names) %>%
  mutate(total_weight = map_dbl(name,
      ~ sum(edges_gc$weight[edges_gc$from == .x | edges_gc$to == .x])))%>%
  mutate(intra_weight = map_dbl(name,
      ~ sum(edges_gc$intra_weight[edges_gc$from == .x | edges_gc$to == .x])))%>%
     mutate(self_weight = total_weight - intra_weight) %>%
     mutate(x = self_weight / intra_weight) %>%
     mutate(valid = (self_weight > intra_weight))%>%
  arrange(-total_weight)

 print(comm_df %>% filter(valid == TRUE))
```

```
# A tibble: 5 x 6
  name         total_weight intra_weight self_weight     x valid
```

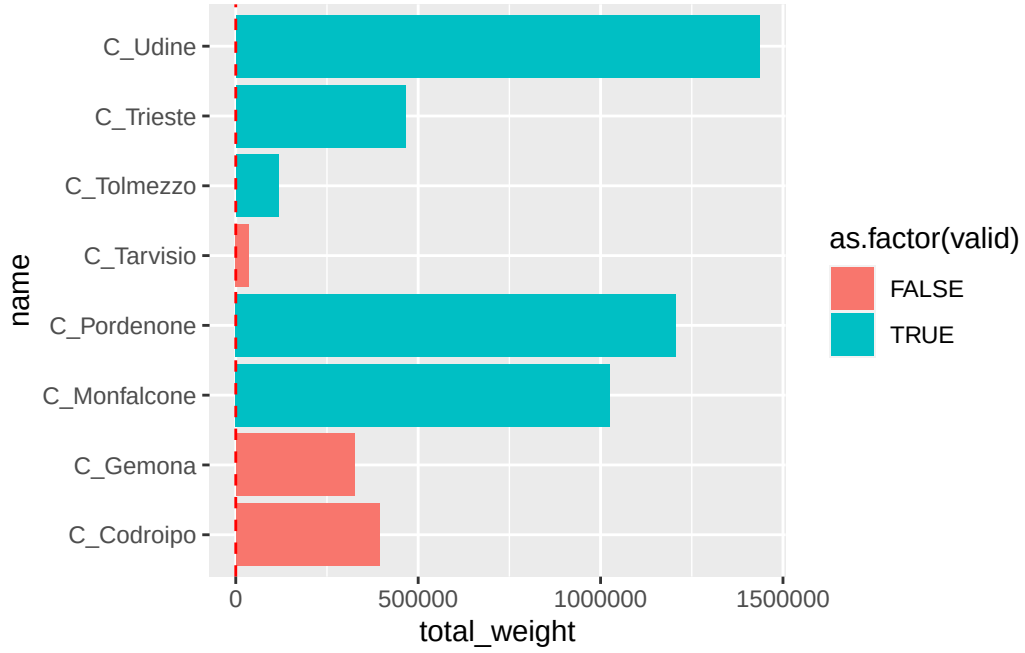
```
  <chr>               <dbl>        <dbl>       <dbl> <dbl> <lgl>
1 C_Udine          1435580.      605529      830050.  1.37 TRUE
2 C_Pordenone      1206487       188510     1017977   5.40 TRUE
3 C_Monfalcone     1026304.      447591      578714.  1.29 TRUE
4 C_Trieste         465664.      194452.     271212   1.39 TRUE
5 C_Tolmezzo        117410        56621       60789   1.07 TRUE
```

```
print(comm_df %>% filter(valid == FALSE))
```

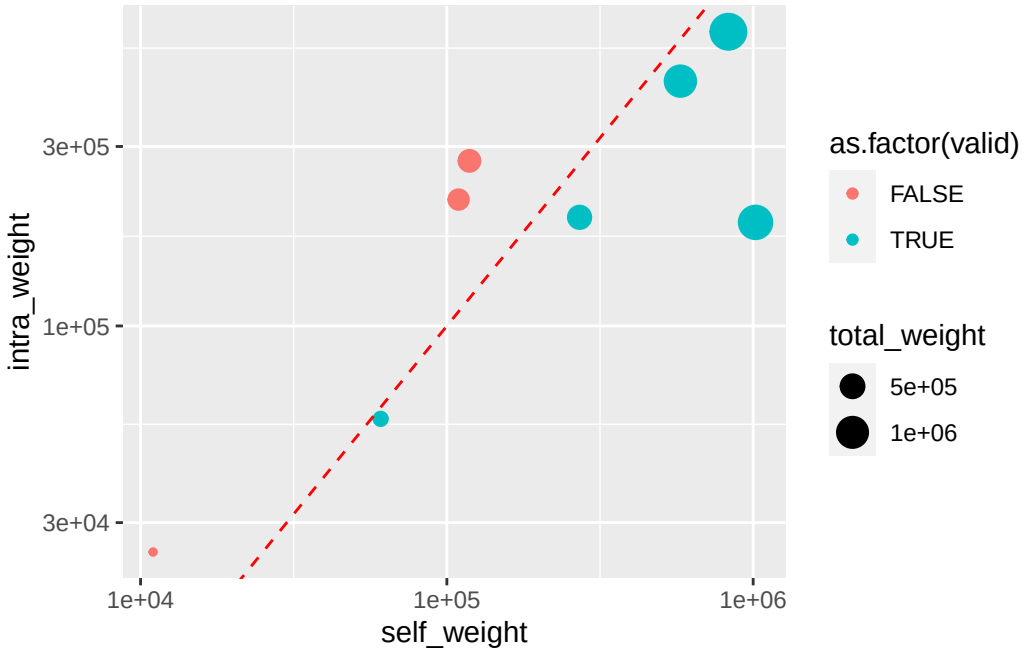
```
# A tibble: 3 x 6
  name       total_weight intra_weight self_weight     x valid
  <chr>             <dbl>        <dbl>       <dbl> <dbl> <lgl>
1 C_Codroipo      393255       274752.     118502. 0.431 FALSE
2 C_Gemona        325968.      216762.     109206  0.504 FALSE
3 C_Tarvisio       36048        25060.      10988. 0.438 FALSE
```

some communities are not valid! this is not a good partition.

```
comm_df %>% ggplot(aes(y = name, x = total_weight, fill = as.factor(valid)))+
    geom_col()+
    geom_vline(xintercept = 0.5, linetype = "dashed", color = 'red')
```



```
comm_df %>% ggplot(aes(x = self_weight, y = intra_weight, fill = as.factor(valid)))+
    geom_point(aes(size = total_weight, color = as.factor(valid)))+
    geom_abline(intercept = 0.0, slope = 1, linetype = "dashed", color = 'red')+
  scale_x_log10()+ scale_y_log10()
```



to do: - check stability over repeated trials - check input ordering bias