IUCN-GET: Diagrammatic assembly models

Example visualisation for freshwater biomes

JR Ferrer-Paris

This document provides an example of how to use and visualise the table summarising diagrammatic assembly model for several ecosystem functional groups (EFG) in R.

The IUCN Global Ecosytem Typology can be explored at: https://global-ecosystems.org

The data comes from an internal database and is not yet publicly available.

Set up

Load libraries

```
library(readr)
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
library(igraph)
```

```
Attaching package: 'igraph'

The following objects are masked from 'package:dplyr':

    as_data_frame, groups, union

The following objects are masked from 'package:stats':

    decompose, spectrum

The following object is masked from 'package:base':

    union
```

Read data

First locate the downloaded csv file.

Either using an absolute path:

```
input_csv_file <-
    "~/Downloads/dam_components_freshwater_results.csv"
```

Or use here to find the path to the file relative to the project folder:

```
here::i_am("examples/DAM-graph-freshwater.qmd")
```

```
here() starts at /Users/z3529065/proyectos/IUCN-GET/IUCN-GET-assembly-models
```

```
input_csv_file <-
    here::here("sandbox", "dam_components_freshwater_results.csv")
```

Now, I use the read_csv function from the readr package:

```
dam_data <- read_csv(input_csv_file)
```

```
Rows: 1023 Columns: 9
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (9): code, shortname, component_code, component_class, component_name, v...

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.
```

Explore the table

We can take a look at the data:

```
glimpse(dam_data)
```

```
Rows: 1,023
Columns: 9
$ code            <chr> "F1.1", "F1.5", "SF1.1", "SF1.2", "SF2.1", "FM1.1", "F~
$ shortname       <chr> "F1.1 Perm upland streams", "F1.5 Season lowland river~
$ component_code  <chr> "F1.1 A1", "F1.5 A1", "SF1.1 A9", "SF1.2 A1", "SF2.1 A~
$ component_class <chr> "Ambient environment", "Ambient environment", "Ambient~
$ component_name  <chr> "Catchment riparian vegetation", "Floodplains and estu~
$ variable        <chr> "adjacent ecosystems", "adjacent ecosystems", "adjacen~
$ characteristic  <chr> "catchment", NA, NA, NA, NA, NA, "seasonal", NA, NA, N~
$ value           <chr> "riparian vegetation", "floodplains", "surface waters"~
$ dam_version     <chr> "<v2.0", "<v2.0", "<v2.0", "<v2.0", "<v2.0", "<v2.0", ~
```

The column code refers to the code of the functional group (EFG), the shortname column includes the code and abbreviated name. Each row has a unique component_code used in the database to organise the data.

Components of the model are grouped in the following component_classes:

```
dam_data %>% select(component_class) %>% table
```

```
component_class
Ambient environment Biotic interactions Disturbance regimes   Ecological Traits
                219                  101                  26                 258
     Human activity              Process            Resources
                 33                  249                  137
```

The table includes the original name of the component in the model (`component_name`) and three columns that try to generalise the description of the component to make it more comparable between functional groups.

For example we can explore all components that have a `variable` called `adjacent ecosystems`:

```
dam_data %>%
    filter(variable %in% "adjacent ecosystems") %>%
    select(shortname,
        component_name,
        variable,
        characteristic,
        value) %>%
    knitr::kable()
```

shortname	component_name	variable	characteristic	value
F1.1 Perm upland streams	Catchment riparian vegetation	adjacent ecosystems	catchment	riparian vegetation
F1.5 Season lowland rivers	Floodplains and estuaries	adjacent ecosystems	NA	floodplains
SF1.1 Underground streams and pools	Surface waters	adjacent ecosystems	NA	surface waters
SF1.2 Groundwater ecosystems	Surface waters	adjacent ecosystems	NA	surface waters
SF2.1 Water pipes and canals	Surface water sources	adjacent ecosystems	NA	surface water sources

Create a graph

The table contains more than thousand rows, but we can gain some insight from visualising the relationships between functional groups and variables.

I use `igraph` to create a graph with all the data from the table:

- First I select the nodes (functional groups or EFGs and variables)
- Then I select the links from EFG to variables
- I put it all together in a graph object

```
nodes <- dam_data %>%
    transmute(id=code, name=code, class="EFG") %>%
    bind_rows(
        dam_data %>%
            transmute(id=variable, name=variable, class="vars")) %>%
    unique
```

```
links <- dam_data %>%
    transmute(from=code, to=variable)

nodes <- nodes %>% filter(id %in% links$from | id %in% links$to)
links <- links %>% filter(from %in% nodes$id & to %in% nodes$id)

g <- graph_from_data_frame(links,nodes,directed=F)
```

```
Warning in graph_from_data_frame(links, nodes, directed = F): In `d' `NA'
elements were replaced with string "NA"
```

```
Warning in graph_from_data_frame(links, nodes, directed = F): In `vertices[,1]'
`NA' elements were replaced with string "NA"
```

Modify attributes for plotting

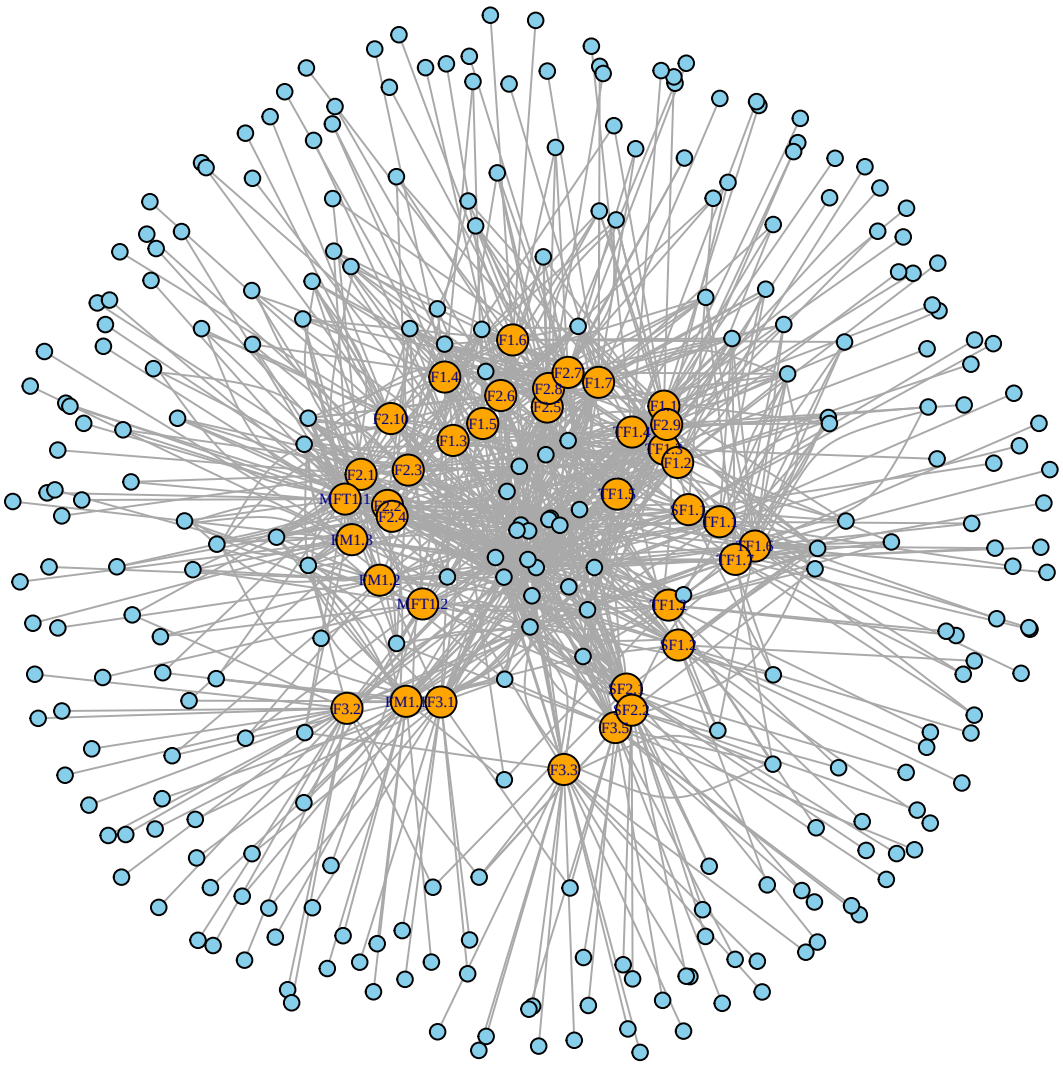
Now we change some attributes for visualisation

```
colrC <- c(EFG="orange",vars="skyblue")
V(g)$color<- colrC[V(g)$class]
V(g)$label <- ifelse(V(g)$class %in% c("EFG"), V(g)$name, NA)
#change arrow size and edge color:
E(g)$arrow.size <- .2
E(g)$edge.color <- "gray80"
V(g)$size <- 3
V(g)$size[V(g)$class %in% "EFG"] <- 6
```

Complete graph with all components

This is the plot of all EFGs and variables, I omitted the variable name for now. There are lots of variables that are used only once (blue circles in the periphery of the graph):

```
l <- layout_with_kk(g)
#l <- layout_with_fr(g)
plot(g, layout=l,vertex.label.cex=.5)
```



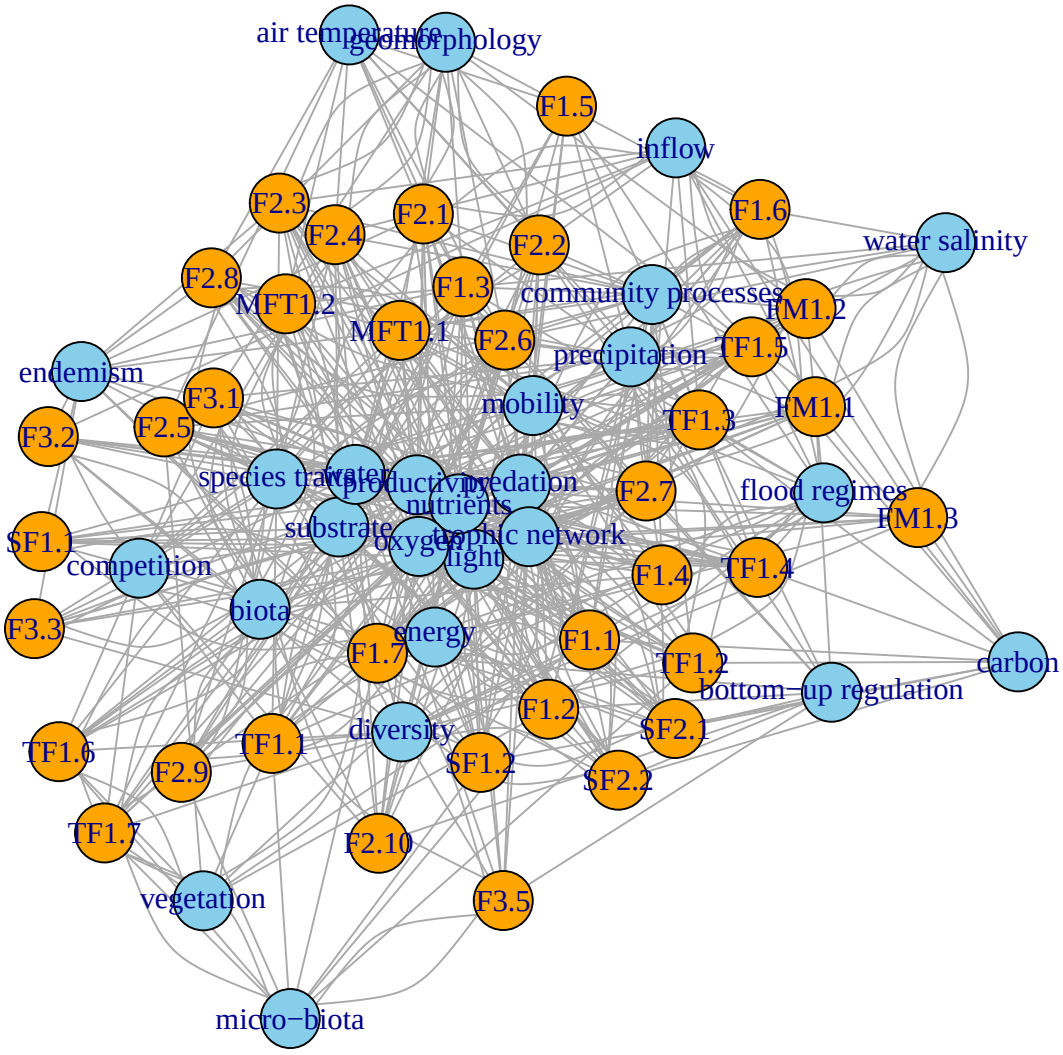
Examples of filtering

We can use the igraph functions to create a subgraph based on a selection of components, either variables or functional groups.

More frequent variables

For example, consider only variables connected to many functional groups, I will use a degree of 10 as a threshold, then I restore the names for the nodes and increase the node size:

```
sg <- subgraph(g, which(degree(g) > 10))
V(sg)$label <- V(sg)$name
V(sg)$size <- 12
plot(sg)
```

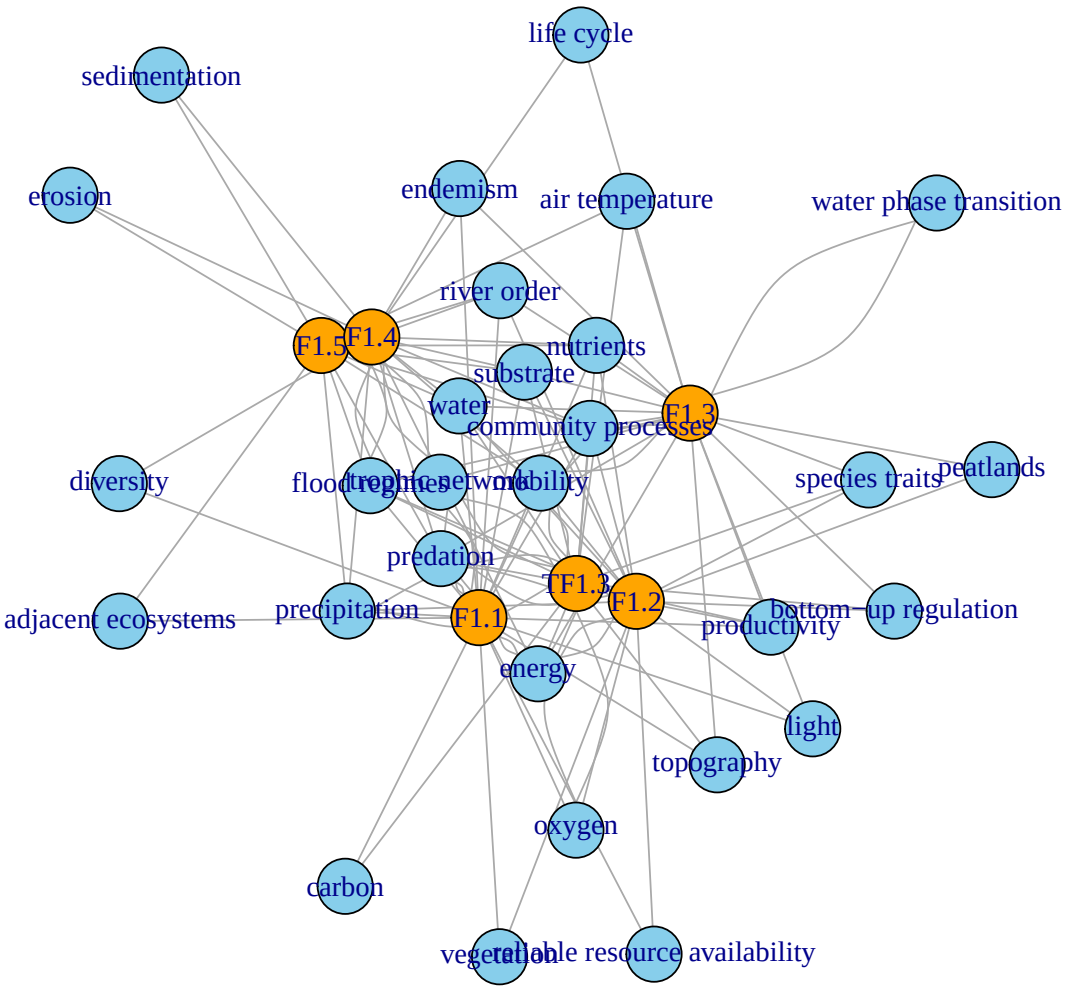


Selection of groups

Here another example where I focus on just a selection of groups. Notice I use here the subgraph function twice, first to select the selected groups and all variables and then to delete the disconnected nodes:

```
f1 <- c("F1.1","F1.2","F1.3","F1.4", "F1.5", "TF1.3")
sg <- subgraph(g, V(g)$name %in% f1 | V(g)$class  %in% "vars")
sg <- subgraph(sg,which(degree(sg) > 1))
V(sg)$label <- V(sg)$name
V(sg)$size <- 12

plot(sg)
```



Back to the table

Once we have filtered our graph to keep the nodes of interest, we can go back to the table and explore the information related to the selected nodes.

We use filter to select the nodes included in the subgraph, and then select the columns with the extra information we want to read:

```
filtered_table <- dam_data %>%
    filter(
        code %in% V(sg)$name,
        variable %in% V(sg)$name
        ) %>%
    select(code,component_class,component_name)
```

We can now explore this filtered table:

```
head(filtered_table) %>%
    knitr::kable()
```

code	component_class	component_name
F1.1	Ambient environment	Catchment riparian vegetation
F1.5	Ambient environment	Floodplains and estuaries

code	component_class	component_name
F1.3	Ambient environment	Cold winters
F1.5	Ambient environment	Warm temperatures
TF1.3	Ambient environment	local temperatures
F1.1	Ambient environment	High kinetic energy

Session Info

Details of the R session below:

```
sessionInfo()
```

```
R version 4.3.0 (2023-04-21)
Platform: aarch64-apple-darwin20 (64-bit)
Running under: macOS Ventura 13.4

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LA

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: Australia/Sydney
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] igraph_1.4.2 dplyr_1.1.2  readr_2.1.4

loaded via a namespace (and not attached):
 [1] crayon_1.5.2     vctrs_0.6.2      cli_3.6.1        knitr_1.42
 [5] rlang_1.1.1      xfun_0.39        generics_0.1.3   jsonlite_1.8.4
 [9] bit_4.0.5        glue_1.6.2       rprojroot_2.0.3  htmltools_0.5.5
[13] hms_1.1.3        fansi_1.0.4      rmarkdown_2.21   evaluate_0.21
[17] tibble_3.2.1     tzdb_0.4.0       fastmap_1.1.1    yaml_2.3.7
[21] lifecycle_1.0.3  compiler_4.3.0   pkgconfig_2.0.3  here_1.0.1
[25] digest_0.6.31    R6_2.5.1         tidyselect_1.2.0 utf8_1.2.3
[29] parallel_4.3.0   vroom_1.6.3      pillar_1.9.0     magrittr_2.0.3
[33] withr_2.5.0      bit64_4.0.5      tools_4.3.0
```