

The belief in irrigation as essential for food and water security

R code

Arnald Puy

Contents

1 Network analysis	2
1.1 Network metrics	9
1.2 Network plots	12
1.3 Uncertainties turned into facts	31
2 Proportion of paths ending up in no claim, no citation or modelling nodes	36
2.1 Network through time	39
3 Analysis of paths	59
3.1 “no claim” or “no citation” paths	59
3.2 Calculation of amplification	61
4 Both networks	68
4.1 Overlap between networks	68
4.2 Shared network	70
5 Study of Aquastat values	74
6 Uncertainty analysis	86
7 Session information	102

```

# PRELIMINARY FUNCTIONS #####
sensobol::load_packages(c("openxlsx", "data.table", "tidyverse", "bibliometrix",
                         "igraph", "ggraph", "cowplot", "tidygraph", "benchmarkme",
                         "parallel", "wesanderson", "scales", "countrycode",
                         "doParallel", "foreach", "sensobol", "sp", "raster", "ncdf4"))

# Create custom theme
theme_AP <- function() {
  theme_bw() +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          legend.background = element_rect(fill = "transparent",
                                             color = NA),
          legend.key = element_rect(fill = "transparent",
                                     color = NA),
          strip.background = element_rect(fill = "white"),
          legend.margin = margin(0.5, 0.1, 0.1, 0.1),
          legend.box.margin = margin(0.2,-4,-7,-7),
          plot.margin = margin(3, 4, 0, 4),
          legend.text = element_text(size = 8),
          axis.title = element_text(size = 10),
          axis.text.x = element_text(size = 7),
          axis.text.y = element_text(size = 7),
          axis.title.x = element_text(size = 7.3),
          axis.title.y = element_text(size = 7.3),
          strip.text.x = element_text(size = 7.4),
          strip.text.y = element_text(size = 7.4),
          legend.key.width = unit(0.4, "cm"),
          legend.key.height = unit(0.4, "cm"),
          legend.title = element_text(size = 7.8))
}

```

1 Network analysis

```

# CREATE VECTORS TO READ IN AND CLEAN THE DATASETS #####
tmp <- list()
names.files <- c("WORK", "NETWORK")
topics <- c("water", "food")
corpus <- c("abstract.corpus", "policy.corpus", "full.text.corpus")
cols_of_interest <- c("title", "author", "claim", "citation",
                      "document.type", "nature.claim")

# Paste all possible combinations of names -----

```

```

combs <- expand.grid(corpus = corpus, topics = topics, approach = names.files)
all.files <- paste(paste(paste(combs$corpus, combs$topics, sep = "."),
                           combs$approach, sep = "."),
                     "xlsx", sep = ".")  

# READ IN DATASETS AND TURN TO LOWERCAPS #####  

tmp <- list()  

for (i in 1:length(all.files)) {  

  tmp[[i]] <- data.table(read.xlsx(all.files[i]))  

  if (!str_detect(all.files[i], "NETWORK")) {  

    tmp[[i]][, title:= tolower(title)]  

  } else {  

    tmp[[i]][, (cols_of_interest):= lapply(.SD, tolower), .SDcols = (cols_of_interest)]  

  }
}  

names(tmp) <- all.files  

sub(".*\\\\.([^\n.]+)_.*", "\\1", all.files)  

## [1] "water" "water" "water" "food" "food" "food" "water" "water" "water"  

## [10] "food" "food" "food"  

# CLEAN AND MERGE DATASETS #####  

# Work datasets -----  

dataset.works <- all.files[str_detect(all.files, "_WORK")]
dataset.works.topics <- sub(".*\\\\.([^\n.]+)_.*", "\\1", dataset.works)  

tmp.works <- tmp[dataset.works]
names(tmp.works) <- dataset.works.topics
lapply(tmp.works, function(dt) dt[, .(doi, title, claim.in.text)]) %>%
  rbindlist(., idcol = "topic") %>%
  .[, .N, .(topic, claim.in.text)]  

##      topic claim.in.text     N
##      <char>          <char> <int>
## 1: water              F 1377
## 2: water             <NA> 159
## 3: water              T 2674
## 4: water    Paywalled     9

```

```

## 5: water      Russian      1
## 6: water      French       1
## 7: water      Indian       1
## 8: water      Ukrainian    1
## 9: water      Portuguese   1
## 10: water     T            2
## 11: food      <NA>        204
## 12: food      T            649
## 13: food      F            2875

# Network datasets ----

dataset.networks <- all.files[str_detect(all.files, "NETWORK")]
dataset.networks.topics <- sub(".*\\" .(^\\".)+_.*", "\\"1", dataset.networks)

tmp2 <- tmp[dataset.networks]
names(tmp2) <- dataset.networks.topics

network.dt <- rbindlist(tmp2, idcol = "topic") %>%
  .[, policy:= grepl("^policy", doi)] %>%
  .[, document.type:= trimws(document.type)] %>%
  .[, document.type:= tolower(document.type)]


# Retrieve year ----

network.dt[, year:= as.integer(sub(".* (\\"d{4})[a-z]?$", "\\"1", author))]

## Warning in eval(jsub, SDenv, parent.frame()): NAs introduced by coercion
# move policy to author ----

network.dt[, author:= ifelse(policy == TRUE, doi, author)]


# CHECK NUMBER OF FAO AQUASTAT CITES #####
#####

aquastat.cites <- network.dt[citation %like% "fao aquastat"] %>%
  .[, .N, .(citation, topic)]


aquastat.cites

##          citation topic     N
##          <char> <char> <int>
## 1: fao aquastat 2006  water    31
## 2: fao aquastat 2006  water    48
## 3: fao aquastat 2010  water    11
## 4: fao aquastat 2020  water     3
## 5: fao aquastat 2011  water     3
## 6: fao aquastat 2012  water     9
## 7: fao aquastat 2021  water     4

```

```

##   8: fao aquastat 2017 water      2
##   9: fao aquastat 2015 water      9
## 10: fao aquastat 2019 water      4
## 11: fao aquastat 2016 water     22
## 12: fao aquastat 2014 water      5
## 13: fao aquastat 2023 water      4
## 14: fao aquastat 2018 water      5
## 15: fao aquastat 2004 water      6
## 16: fao aquastat 2005 water      3
## 17:    fao aquastat water       1
## 18: fao aquastat 2003 water      2
## 19: fao aquastat 2013 water      4
## 20: fao aquastat 2008 water      1
## 21: fao aquastat 2022 water      1
## 22:    fao aquastat food        8
## 23: fao aquastat 2014 food        2
## 24: fao aquastat 2012 food        9
## 25: fao aquastat 2019 food        1
## 26: fao aquastat 2016 food        6
## 27: fao aquastat 2018 food        1
## 28: fao aquastat 2020 food        1
## 29: fao aquastat 2015 food        1
## 30: fao aquastat 2022 food        2
## 31: fao aquastat 2021 food        1
##           citation topic      N
oldest.aquastat.cite <- min(as.integer(sub(".* (\d{4})[a-z]?", "\\\\$1",
                                             aquastat.cites$citation)),
                                na.rm = TRUE)

```

Warning: NAs introduced by coercion

CHECK NUMBER OF FAOSTAT CITES

```

faostat.cites <- network.dt[citation %like% "faostat"] %>%
  .[, .N, .(citation, topic)]

```

faostat.cites

```

##               citation topic      N
##               <char> <char> <int>
##   1: faostat online service water      1
##   2:          faostat 2011 water      3
##   3:          faostat 2019 water      2
##   4:          faostat 2008 water      2
##   5:          faostat 2020 water      3
##   6:          faostat 2012 water      1
##   7:          faostat 2021 water      2
##   8:          faostat online water      1

```

```

##  9:          faostat  water      3
## 10:          faostat  water      5
## 11:          faostat 2023  water      1
## 12:          faostat 2012   food      2
## 13:          faostat 2011   food      1
## 14:          faostat 2019   food      2
## 15:          faostat 2002   food      2
## 16:          faostat 2016   food      2
## 17:          faostat 2003   food      1
## 18:          faostat 1999   food      1
## 19:          faostat   food      1

oldest.faostat.cite <- min(as.integer(sub(".* (\d{4})[a-z]?$", "\\\1",
                                         faostat.cites$citation)),
                           na.rm = TRUE)

## Warning: NAs introduced by coercion
# WRITE LOOKUP TABLE TO CHECK ALREADY RETRIEVED STUDIES #####
#####

lookup.dt <- network.dt[, .(doi, title, author, topic)] %>%
  .[order(title)] %>%
  unique()

lookup.dt[, .(number.rows = nrow(.SD)), topic]

##      topic number.rows
##      <char>      <int>
## 1:   food        903
## 2:  water       3671

# Export lookup tables -----
write.xlsx(lookup.dt, "lookup.dt.xlsx")
write.xlsx(lookup.dt[topic == "water"], "lookup.water.dt.xlsx")
write.xlsx(lookup.dt[topic == "food"], "lookup.food.dt.xlsx")

# Remove the year from mentions to FAO Aquastat -----
pattern <- "\b(?:19|20)\d{2}\b" # Matches years between 1900 and 2099

for (col in c("citation", "author")) {
  matches <- grep("fao aquastat\s+\d+$", network.dt[[col]], ignore.case = TRUE)
  network.dt[matches, (col) := gsub("\d+", "", network.dt[[col]][matches], perl = TRUE)]
  network.dt[, (col) := trimws(network.dt[[col]])]
}

# Remove the year from mentions to FAOSTAT -----
for (col in c("citation", "author")) {

```

```

matches <- grep("faostat\\s+\\d+", network.dt[[col]], ignore.case = TRUE)
network.dt[matches, (col) := gsub("\\d+", "", network.dt[[col]][matches], perl = TRUE)]
network.dt[, (col) := trimws(network.dt[[col]])]
}

# Rename columns ----

setnames(network.dt, c("author", "citation"), c("from", "to"))

# Rename category ----

network.dt[, category:= ifelse(!classification == "F", "Uncertain", "Fact")]

# Create copy and remove duplicated ----

network.dt.claim <- copy(network.dt)
network.dt.claim <- unique(network.dt.claim, by = c("from", "to", "document.type",
                                                     "nature.claim"))
cols_to_change <- colnames(network.dt)
network.dt.claim[, (cols_to_change):= lapply(.SD, trimws), .SDcols = (cols_to_change)]
network.dt.claim[, (cols_to_change):= lapply(.SD, str_squish), .SDcols = (cols_to_change)]

fwrite(network.dt.claim, "network.dt.claim.csv")

# Convert all to lower caps ----

network.dt <- network.dt[, .(from, to, year, document.type, nature.claim,
                           classification, category, topic)]
cols_to_change <- colnames(network.dt)
network.dt[, (cols_to_change):= lapply(.SD, trimws), .SDcols = (cols_to_change)]
network.dt[, (cols_to_change):= lapply(.SD, str_squish), .SDcols = (cols_to_change)]

# PLOT DESCRIPTIVE STATISTICS #####
total.rows <- network.dt[, .(number.rows = nrow(.SD)), topic]

# Check proportion of studies by nature of claim ----

network.dt.claim[, .N, .(nature.claim, topic)] %>%
  merge(., total.rows, by = "topic") %>%
  .[, fraction:= N / number.rows] %>%
  print()

## Key: <topic>
##      topic    nature.claim      N number.rows     fraction
##      <char>      <char> <int>      <int>      <num>
##  1: food      no citation    194       1038 0.186897881
##  2: food citation backup    617       1038 0.594412331

```

```

## 3: food      no claim    93      1038 0.089595376
## 4: food      <NA>      37      1038 0.035645472
## 5: food      modelling    2      1038 0.001926782
## 6: water citation backup 2751      4352 0.632123162
## 7: water modelling    20      4352 0.004595588
## 8: water no citation 973      4352 0.223575368
## 9: water      <NA>     116      4352 0.026654412
## 10: water no claim     439      4352 0.100873162

# Count document type by nature of claim ----

a <- network.dt[, .N, .(nature.claim, document.type, topic)] %>%
  merge(., total.rows, by = "topic") %>%
  .[, proportion:= N / number.rows] %>%
  na.omit() %>%
  ggplot(., aes(reorder(nature.claim, proportion), proportion)) +
  coord_flip() +
  geom_bar(stat = "identity") +
  facet_grid(topic~document.type) +
  scale_y_continuous(breaks = breaks_pretty(n = 2)) +
  labs(x = "", y = "Fraction") +
  theme_AP()

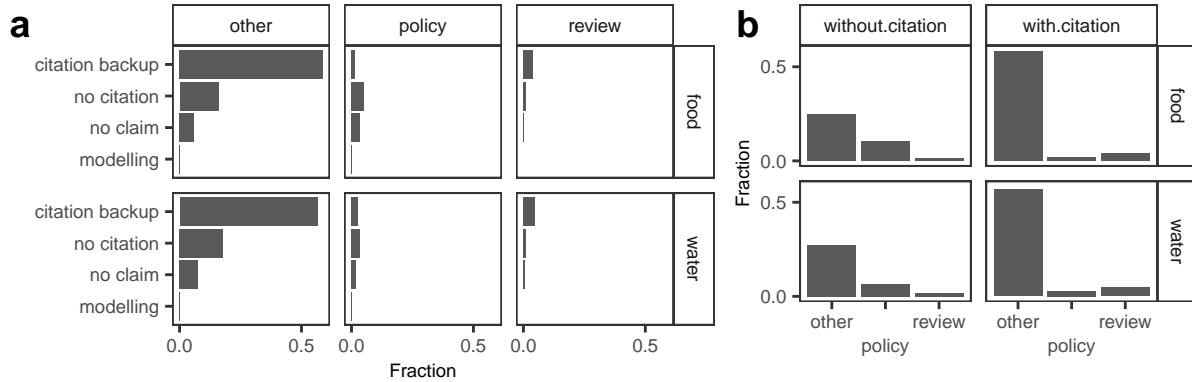
# Count how many documents make the claim and cite / do not cite,
# by document.type ----

b <- network.dt[, .(without.citation = sum(is.na(to)),
                     with.citation = .N - sum(is.na(to))), .(document.type, topic)] %>%
  melt(., measure.vars = c("without.citation", "with.citation")) %>%
  merge(., total.rows, by = "topic") %>%
  .[, proportion:= value / number.rows] %>%
  ggplot(., aes(document.type, proportion)) +
  geom_bar(stat = "identity") +
  scale_y_continuous(breaks = breaks_pretty(n = 2)) +
  scale_x_discrete(guide = guide_axis(n.dodge = 2)) +
  labs(x = "", y = "Fraction") +
  facet_grid(topic~variable) +
  theme_AP()

# merge ----

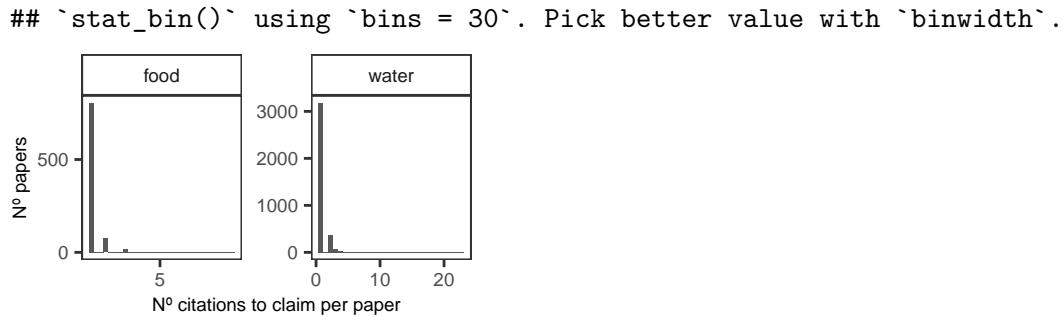
plot.claim <- plot_grid(a, b, ncol = 2, rel_widths = c(0.6, 0.4), labels = "auto")
plot.claim

```



```
# PLOT DISTRIBUTION OF CITATION SUPPORTING THE CLAIM #####
plot.supporting.claim <- network.dt[, .N, .(from, topic)] %>%
  .[order(-N)] %>%
  ggplot(., aes(N)) +
  geom_histogram() +
  facet_wrap(~topic, scale = "free") +
  scale_x_continuous(breaks = breaks_pretty(n = 3)) +
  scale_y_continuous(breaks = breaks_pretty(n = 3)) +
  theme_AP() +
  labs(x = "Nº citations to claim per paper", y = "Nº papers")

plot.supporting.claim
```



1.1 Network metrics

```
# CALCULATE NETWORK METRICS #####
# only complete cases ----

network.dt.complete <- network.dt[complete.cases(network.dt$to), ]
split.networks <- split(network.dt.complete, network.dt.complete$topic)

# Export-----
write.xlsx(network.dt.complete, "network.dt.complete.xlsx")

# Transform to graph -----
```

```

citation_graph <- lapply(split.networks, function(dt)
  graph_from_data_frame(d = dt, directed = TRUE))

## Warning in graph_from_data_frame(d = dt, directed = TRUE): In `d` `NA` elements
## were replaced with string "NA"

## Warning in graph_from_data_frame(d = dt, directed = TRUE): In `d` `NA` elements
## were replaced with string "NA"

# Calculate network metrics -----
lapply(citation_graph, function(x) edge_density(x))

## $food
## [1] 0.001132669
##
## $water
## [1] 0.0003277328

# Modularity:
# - c.1: Strong community structure, where nodes within groups are highly connected.
# - c. -1: Opposite of community structure, where nodes between groups are more connected.
# - c. 0: Indicates absence of community structure or anti-community structure in the network.
wtc <- lapply(citation_graph, function(x) cluster_walktrap(x))
lapply(wtc, function(x) modularity(x))

## $food
## [1] 0.9238484
##
## $water
## [1] 0.8739123

network_metrics <- lapply(citation_graph, function(x)
  data.table(node = V(x)$name,

    # Degree of a node: The number of connections or
    # edges linked to that node.
    # It represents how well-connected or central a
    # node is within the graph.
    degree = degree(x, mode = "in"),

    degree.out = degree(x, mode = "out"),

    # Betweenness centrality of a node: Measures the
    # extent to which a node lies on the shortest
    # paths between all pairs of other nodes in the graph.
    # Nodes with high betweenness centrality act as
    # bridges or intermediaries, facilitating
    # communication and information flow between other nodes.
  )
)

```

```

betweenness = betweenness(x),

# Closeness centrality of a node: Measures how
# close a node is to all other nodes in the graph,
# taking into account the length of the shortest paths.
# Nodes with high closeness centrality are able to
# efficiently communicate or interact with other
# nodes in the graph.
closeness = closeness(x),
pagerank = page_rank(x)$vector)
)

# Define the max number of rows
max.number <- 3

degree.nodes <- lapply(network_metrics, function(dt) dt[order(-degree)] [1:max.number])
degree.nodes.out <- lapply(network_metrics, function(dt) dt[order(-degree.out)] [1:max.number])
betweenness.nodes <- lapply(network_metrics, function(dt) dt[order(-betweenness)] [1:max.number])
pagerank.nodes <- lapply(network_metrics, function(dt) dt[order(-closeness)] [1:max.number])

degree.nodes

## $food
##           node degree degree.out betweenness closeness    pagerank
##           <char>  <num>      <num>      <num>      <num>      <num>
## 1:     fao aquastat     32          0          0       NaN 0.02961316
## 2:           fao 2002     18          0          0       NaN 0.01577372
## 3: morris et al 2003     16          0          0       NaN 0.01133942
##
## $water
##           node degree degree.out betweenness closeness    pagerank
##           <char>  <num>      <num>      <num>      <num>      <num>
## 1:     fao aquastat    178          0      0.0000       NaN 0.063812081
## 2: siebert et al 2010     67          3    367.8333      0.125 0.008105668
## 3:     fao 2011        66          1   138.5000      1.000 0.017543929
degree.nodes.out

## $food
##           node degree degree.out betweenness closeness    pagerank
##           <char>  <num>      <num>      <num>      <num>      <num>
## 1: taguta et al 2022      0          9          0 0.04761905 0.0007611628
## 2:     pei et al 2017      1          6          4 0.20000000 0.0008905604
## 3: kadigi et al 2004      0          6          0 0.10000000 0.0007611628
##
## $water
##           node degree degree.out betweenness closeness    pagerank
##           <char>  <num>      <num>      <num>      <num>      <num>

```

```

## 1:      wada 2015      0       23      0 0.02702703 0.0001773598
## 2: wada et al 2014a    1        9     10 0.09090909 0.0002075110
## 3: zarei et al 2021    0        9      0 0.06666667 0.0001773598

betweenness.nodes

## $food
##           node degree degree.out betweenness closeness
##           <char>  <num>      <num>      <num>      <num>
## 1: vorosmarty and sahagian 2000      5        3      15 0.3333333
## 2: siebert et al 2005            13        2      14 1.0000000
## 3: united nations 2009          10        2      11 1.0000000

## pagerank
##           <num>
## 1: 0.003996105
## 2: 0.008368667
## 3: 0.006614610
##
## $water
##           node degree degree.out betweenness closeness pagerank
##           <char>  <num>      <num>      <num>      <num>      <num>
## 1: siebert et al 2010      67        3 367.8333 0.1250000 0.008105668
## 2: doll 2009            12        1 200.0000 0.3333333 0.004984458
## 3: boretti and rosa 2019      12        5 179.5000 0.0400000 0.001630897

pagerank.nodes

## $food
##           node degree degree.out betweenness closeness pagerank
##           <char>  <num>      <num>      <num>      <num>      <num>
## 1: okorogbona et.al 2018      0        1        0      1 0.0007611628
## 2: du preez et al 2018      0        1        0      1 0.0007611628
## 3: meier et al 2018         3        1        3      1 0.0023786336
##
## $water
##           node degree degree.out betweenness closeness pagerank
##           <char>  <num>      <num>      <num>      <num>      <num>
## 1: sharma and irmak 2012      0        1        0      1 0.0001773598
## 2: world bank 2007            5        1       46      1 0.0052992873
## 3: brajovic et al 2015        0        1        0      1 0.0001773598

```

1.2 Network plots

```

# ADD FEATURES TO NODES #####
# Retrieve a vector with the node names -----
graph <- lapply(split.networks, function(nt)
  tidygraph::as_tbl_graph(nt, directed = TRUE))

```

```

## Warning in graph_from_data_frame(x, directed = directed): In `d` `NA` elements
## were replaced with string "NA"

## Warning in graph_from_data_frame(x, directed = directed): In `d` `NA` elements
## were replaced with string "NA"

vec.names <- lapply(graph, function(graph)
  graph %>%
    activate(nodes) %>%
    pull() %>%
    data.table(name = .))

# Merge with info from the network.dt -----
tmp.network <- split(network.dt, network.dt$topic)

vec.nature.claim <- list()

for(i in names(tmp.network)) {

  vec.nature.claim[[i]] <- merge(merge(vec.names[[i]], unique(tmp.network[[i]][, .(from, year,
    by.x = "name", by.y = "from", all.x = TRUE),
    unique(tmp.network[[i]][, .(from, document.type, classification,
    by.x = "name", by.y = "from", all.x = TRUE)
  }])

# Merge with the correct order -----
order_indices <- final.vec.nature.claim <- final.vec.document.type <-
  final.vec.year <- final.vec.classification <- final.vec.category <- list()

for (i in names(vec.names)) {

  order_indices[[i]] <- match(vec.names[[i]]$name, vec.nature.claim[[i]]$name)
  final.vec.nature.claim[[i]] <- vec.nature.claim[[i]][order_indices[[i]], ] %>%
    .[, nature.claim]
  final.vec.document.type[[i]] <- vec.nature.claim[[i]][order_indices[[i]], ] %>%
    .[, document.type]
  final.vec.year[[i]] <- vec.nature.claim[[i]][order_indices[[i]], ] %>%
    .[, year] %>%
    as.numeric()
  final.vec.classification[[i]] <- vec.nature.claim[[i]][order_indices[[i]], ] %>%
    .[, classification]
  final.vec.category[[i]] <- vec.nature.claim[[i]][order_indices[[i]], ] %>%
    .[, category]
}

```

```

# Attach to the graph -----
graph.final <- list()

for (i in names(graph)) {

  graph.final[[i]] <- graph[[i]] %>%
    activate(nodes) %>%
    mutate(nature.claim = final.vec.nature.claim[[i]],
           document.type = final.vec.document.type[[i]],
           year = final.vec.year[[i]],
           degree = network_metrics[[i]]$degree,
           classification = final.vec.classification[[i]],
           category = final.vec.category[[i]],
           degree.out = network_metrics[[i]]$degree.out,
           betweenness = network_metrics[[i]]$betweenness,
           pagerank = network_metrics[[i]]$pagerank)

}

for (i in names(graph.final)) {

  graph.final[[i]] <- graph.final[[i]] %>%
    activate(edges) %>%
    mutate(edge_color = .N()$nature.claim[to])
}

# EXPORT NODES AND EDGES #####
for (i in topics) {

  nodes <- graph.final[[i]] %>%
    activate(nodes) %>%
    data.frame() %>%
    data.table()

  edges <- graph.final[[i]] %>%
    activate(edges) %>%
    data.frame() %>%
    data.table()

  write.xlsx(nodes, paste(i, ".nodes.xlsx", sep = ""))
  write.xlsx(edges, paste(i, ".edges.xlsx", sep = ""))
}

```

```

# NUMBER OF NODES #####
lapply(graph.final, function(graph) V(graph))

## $food
## + 765/765 vertices, named, from 2a6cdb9:
## [1] okorogbona et.al 2018
## [2] du preez et al 2018
## [3] niu et al 2023
## [4] meier et al 2018
## [5] lobell et al 2006
## [6] rosa 2022
## [7] rolle et al 2021
## [8] mitchell et al 2018
## [9] wang et al 2012
## [10] hanjra and qureshi 2010
## + ... omitted several vertices
##
## $water
## + 2925/2925 vertices, named, from 2ce2977:
## [1] sharma and irmak 2012
## [2] doreau et al 2012
## [3] world water assessment programme 2009
## [4] world bank 2007
## [5] brajovic et al 2015
## [6] rivers et al 2015
## [7] kijne 2005
## [8] hafeez and khalid awan 2022
## [9] dunkelman et al 2017
## [10] nordin et al 2013
## + ... omitted several vertices

# NUMBER OF EDGES #####
lapply(graph.final, function(graph) ecount(graph))

## $food
## [1] 662
##
## $water
## [1] 2803

# SCALE-FREE PLOT #####
# Prepare data -----
dt.food <- graph.final[[1]] %>%
  activate(nodes) %>%
  data.frame() %>%

```

```

data.table() %>%
  .[, topic := "food"]

dt.water <- graph.final[[2]] %>%
  activate(nodes) %>%
  data.frame() %>%
  data.table() %>%
  .[, topic := "water"]

tmp <- rbind(dt.food, dt.water)

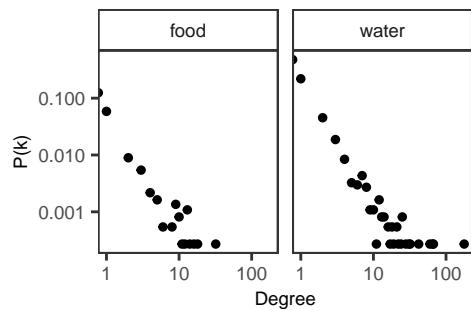
# Calculate the degree distribution -----
degree_distribution <- tmp[, .(P_k = .N / nrow(tmp)), .(degree, topic)]

plot.degree.distribution <- degree_distribution %>%
  ggplot(., aes(degree, P_k)) +
  geom_point(size = 1) +
  scale_y_log10() +
  scale_x_log10() +
  facet_wrap(~topic) +
  labs(x = "Degree", y = "P(k)") +
  theme_AP()

plot.degree.distribution

```

Warning in scale_x_log10(): log-10 transformation introduced infinite values.



```

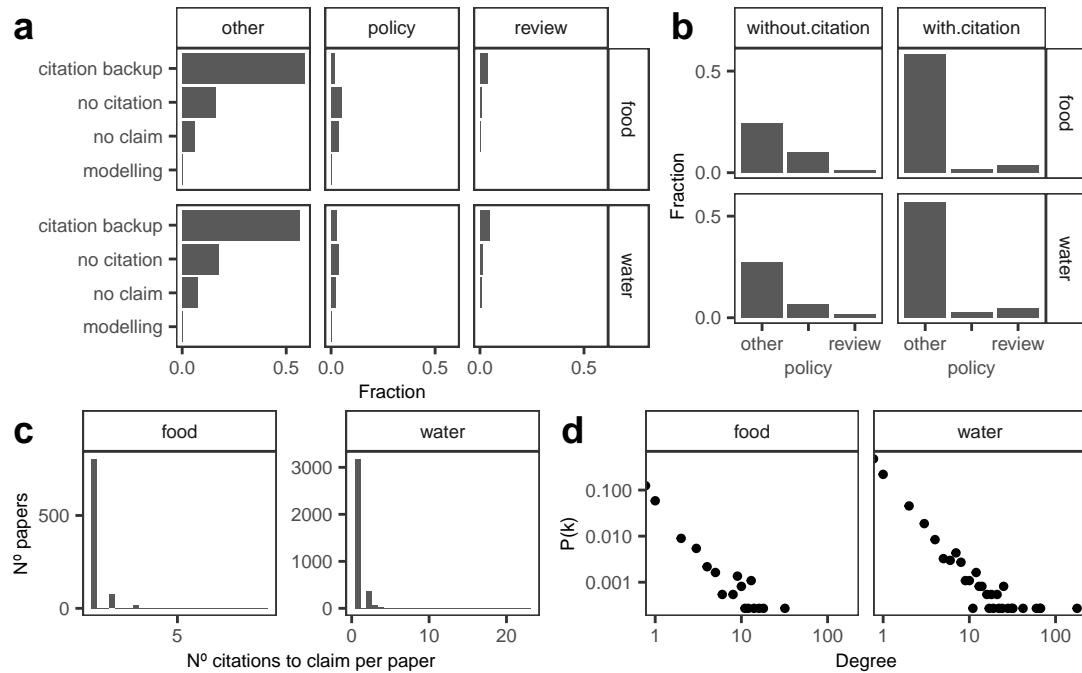
# MERGE DESCRIPTIVE PLOTS #####
bottom <- plot_grid(plot.supporting.claim, plot.degree.distribution, ncol = 2,
                     labels = c("c", "d"))

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning in scale_x_log10(): log-10 transformation introduced infinite values.

```

```
plot_grid(plot.claim, bottom, ncol = 1, rel_heights = c(0.6, 0.4))
```



```
# CALCULATE ALL POSSIBLE PATHS #####
# Define function -----
count_paths <- function(g) {

  # Extract the top 5 nodes with highest betweenness -----
  top_nodes <- g %>%
    activate(nodes) %>%
    top_n(5, betweenness) %>%
    pull(name)

  # Initialize counts -----
  total_paths_count <- 0
  top_nodes_paths_count <- 0

  g <- as.igraph(g)

  results <- foreach(i = V(g),
                     .combine = "c",
                     .packages = "igraph") %:%

  foreach(j = V(g),
         .combine = "c") %dopar% {
```

```

total_paths_pair <- 0
top_nodes_paths_pair <- 0

if (i != j) {

  # Calculate all possible paths ----

  paths <- all_simple_paths(g, from = i, to = j)
  total_paths_pair <- length(paths)

  # Check how many paths pass through the top betweenness nodes --

  for (path in paths) {

    if (any(names(path) %in% top_nodes)) {
      top_nodes_paths_pair <- top_nodes_paths_pair + 1

    }
  }
}

# Return the count of all paths and paths through top nodes ----

return(c(total_paths_pair, top_nodes_paths_pair))
}

# Aggregate results ----

total_paths_count <- sum(results[seq(1, length(results), by = 2)])
top_nodes_paths_count <- sum(results[seq(2, length(results), by = 2)])

return(c(total_paths_count, top_nodes_paths_count))

}

# Define parallel computing ----

cl <- makeCluster(floor(detectCores() * 0.75))
registerDoParallel(cl)

# Run the function ----

results.counts <- lapply(graph.final, function(graph)
  count_paths(graph))

# Stop the cluster ----

```

```

stopCluster(cl)

# SHOW TOTAL NUMBER OF PATHS AND PROPORTION OF PATHS PASSING
# THROUGH THE FIVE NODES WITH THE HIGHEST BETWEENNESS #####
#####

results.counts

## $food
## [1] 833 104
##
## $water
## [1] 5329 1375

lapply(results.counts, function(x) x[[2]] / x[[1]])

## $food
## [1] 0.1248499
##
## $water
## [1] 0.2580221

# PLOT NETWORK #####
#####

seed <- 1234
selected_colors <- c("darkblue", "lightgreen", "orange", "red", "grey")

# by nature of claim -----
# Label the nodes with highest degree -----

p1 <- p2 <- p3 <- p4 <- p5 <- list()

for(i in names(graph.final)) {

  set.seed(seed)

  p1[[i]] <- ggraph(graph.final[[i]], layout = "igraph", algorithm = "nicely") +
    geom_edge_link(arrow = arrow(length = unit(1.8, 'mm')),
                  end_cap = circle(1, "mm"),
                  aes(color = edge_color)) +
    scale_edge_color_manual(values = selected_colors, guide = "none") +
    geom_node_point(aes(color = nature.claim, size = degree)) +
    geom_node_text(aes(label = ifelse(degree >= min(degree.nodes[[i]]$degree), name, NA)),
                  repel = TRUE, size = 2.2) +
    labs(x = "", y = "") +
    scale_color_manual(name = "",
                      values = selected_colors) +
    theme_AP() +
    theme(axis.text.x = element_blank(),

```

```

        axis.ticks.x = element_blank(),
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        legend.position = "right")
}

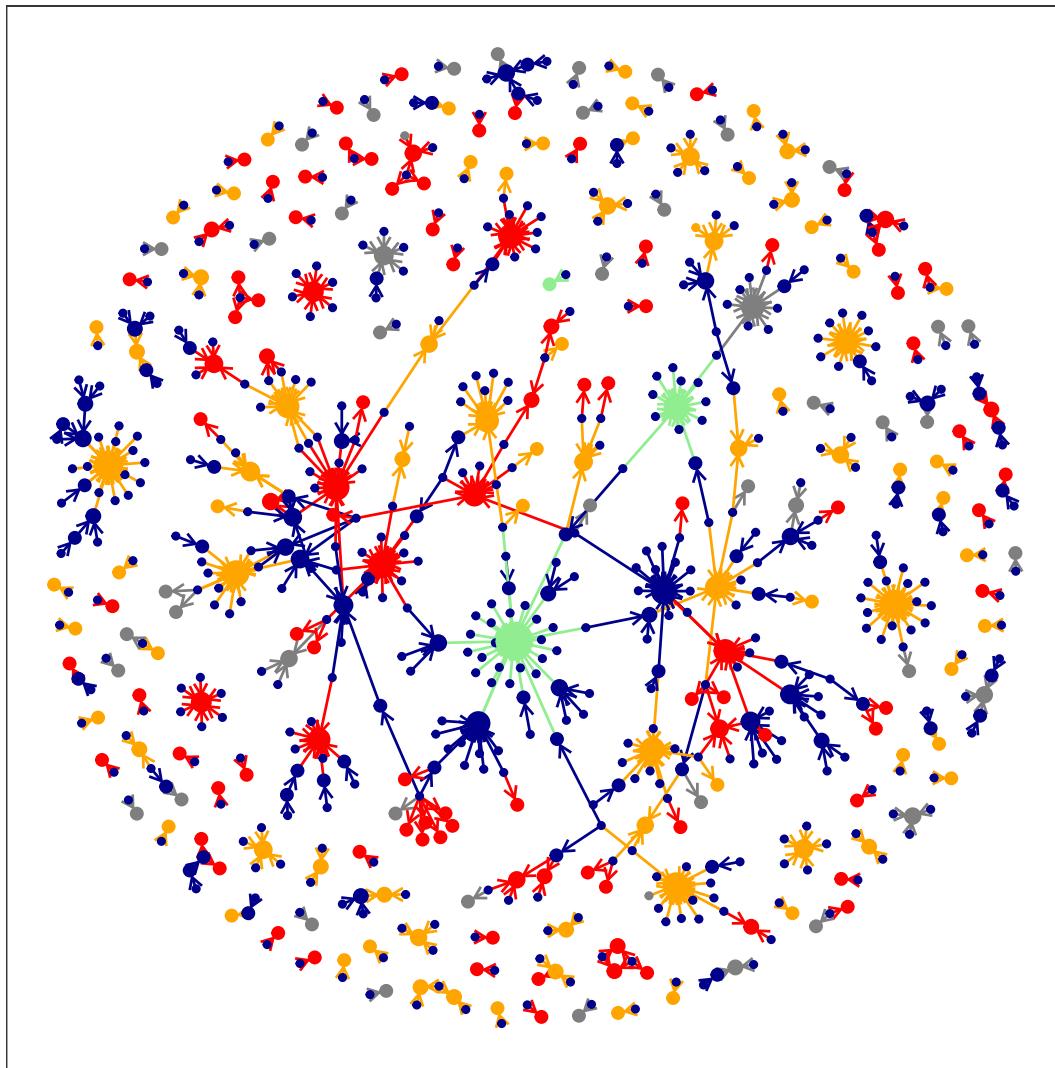
p1

## $food

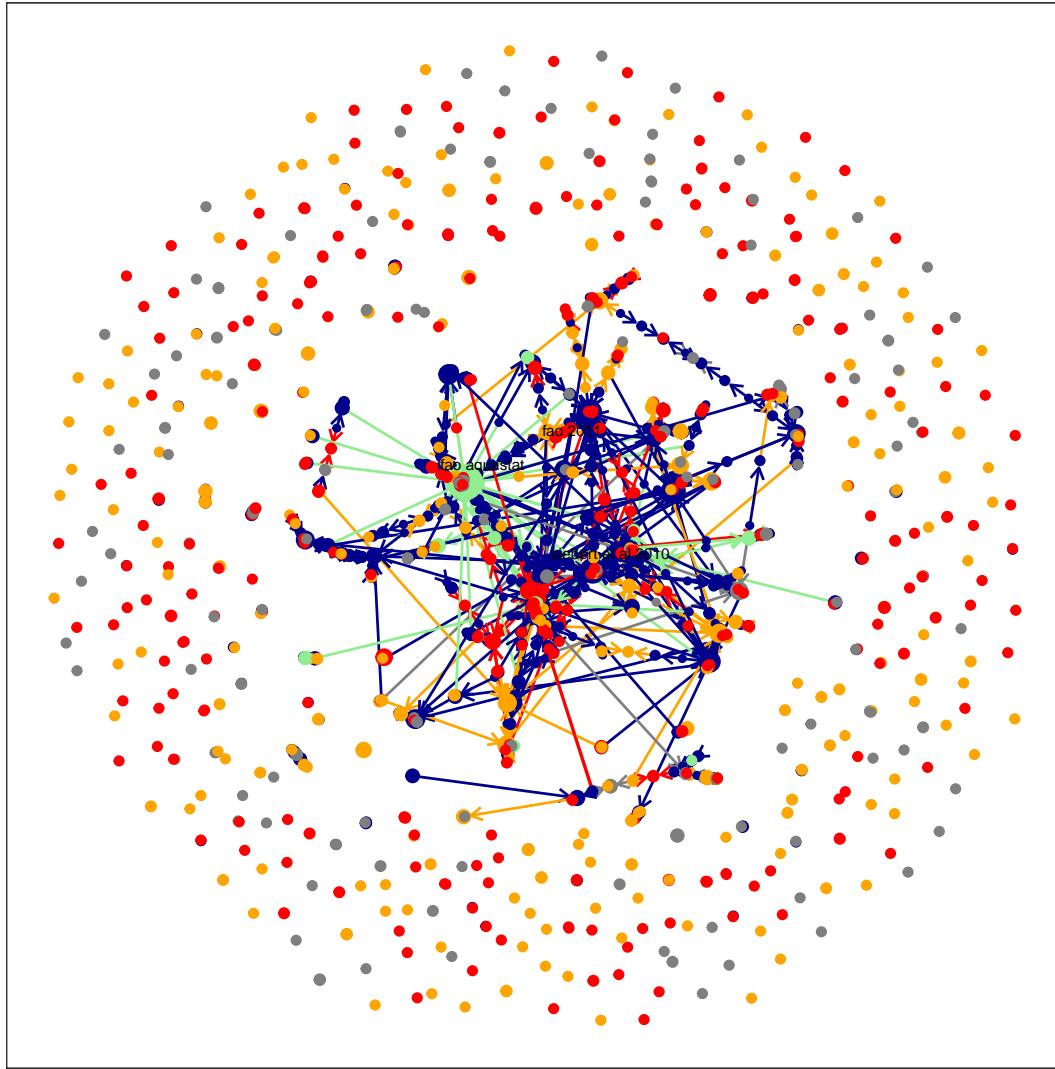
## Warning: Using the `size` aesthetic in this geom was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` in the `default_aes` field and elsewhere instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

## Warning: Removed 765 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

```



```
##  
## $water  
  
## Warning: Removed 2922 rows containing missing values or values outside the scale range  
## (`geom_text_repel()`).
```



```
for(i in names(graph.final)) {  
  
  set.seed(seed)  
  
  p5[[i]] <- ggraph(graph.final[[i]], layout = "stress") +  
    geom_edge_link(arrows = arrow(length = unit(1.8, 'mm')),  
                  end_cap = circle(1, "mm"),  
                  aes(color = edge_color)) +  
    scale_edge_color_manual(values = selected_colors, guide = "none") +  
    geom_node_point(aes(color = nature.claim, size = degree)) +  
    geom_node_text(aes(label = ifelse(nature.claim == "modelling", name, NA)),  
                  repel = TRUE, size = 2.2) +
```

```

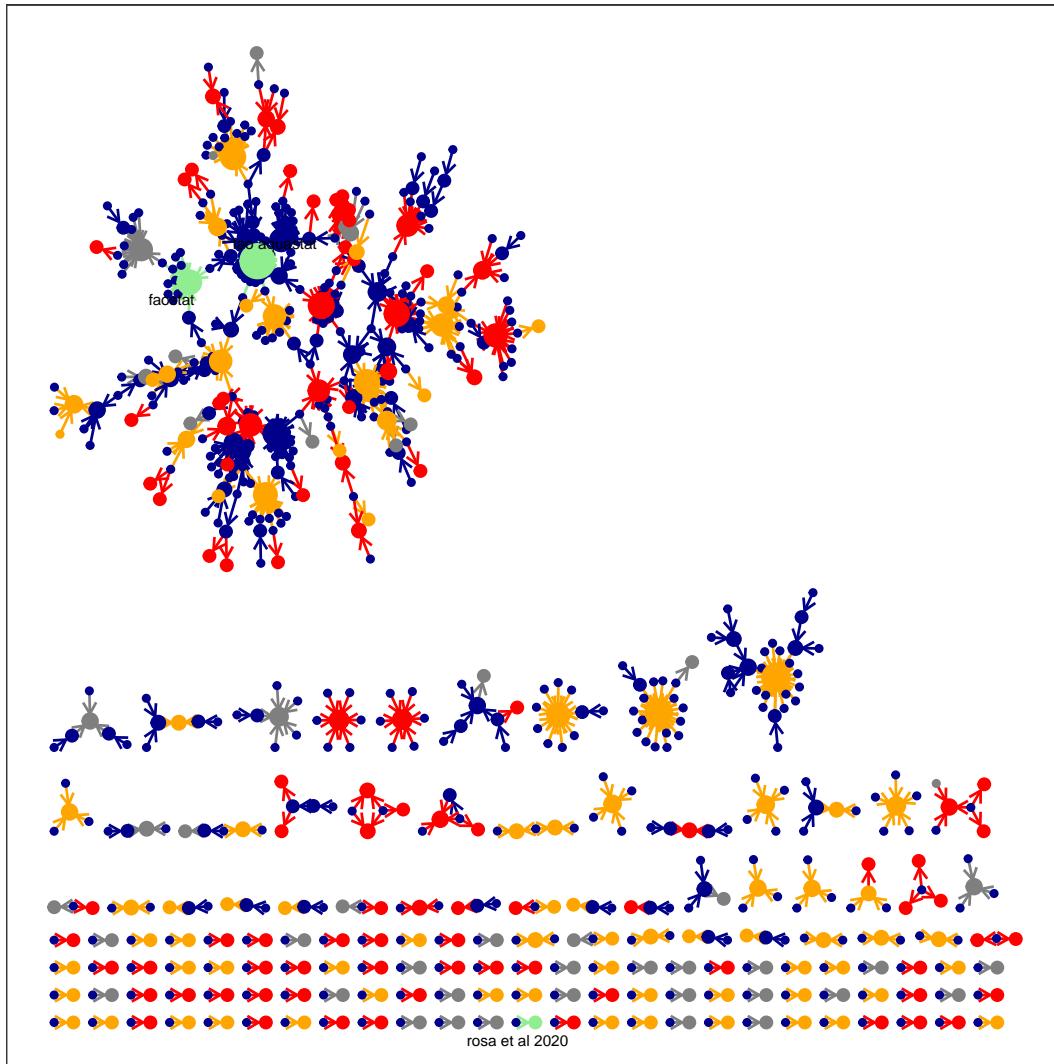
    labs(x = "", y = "") +
    scale_color_manual(name = "",
                       values = selected_colors) +
    theme_AP() +
    theme(axis.text.x = element_blank(),
          axis.ticks.x = element_blank(),
          axis.text.y = element_blank(),
          axis.ticks.y = element_blank(),
          legend.position = "right")
}

p5

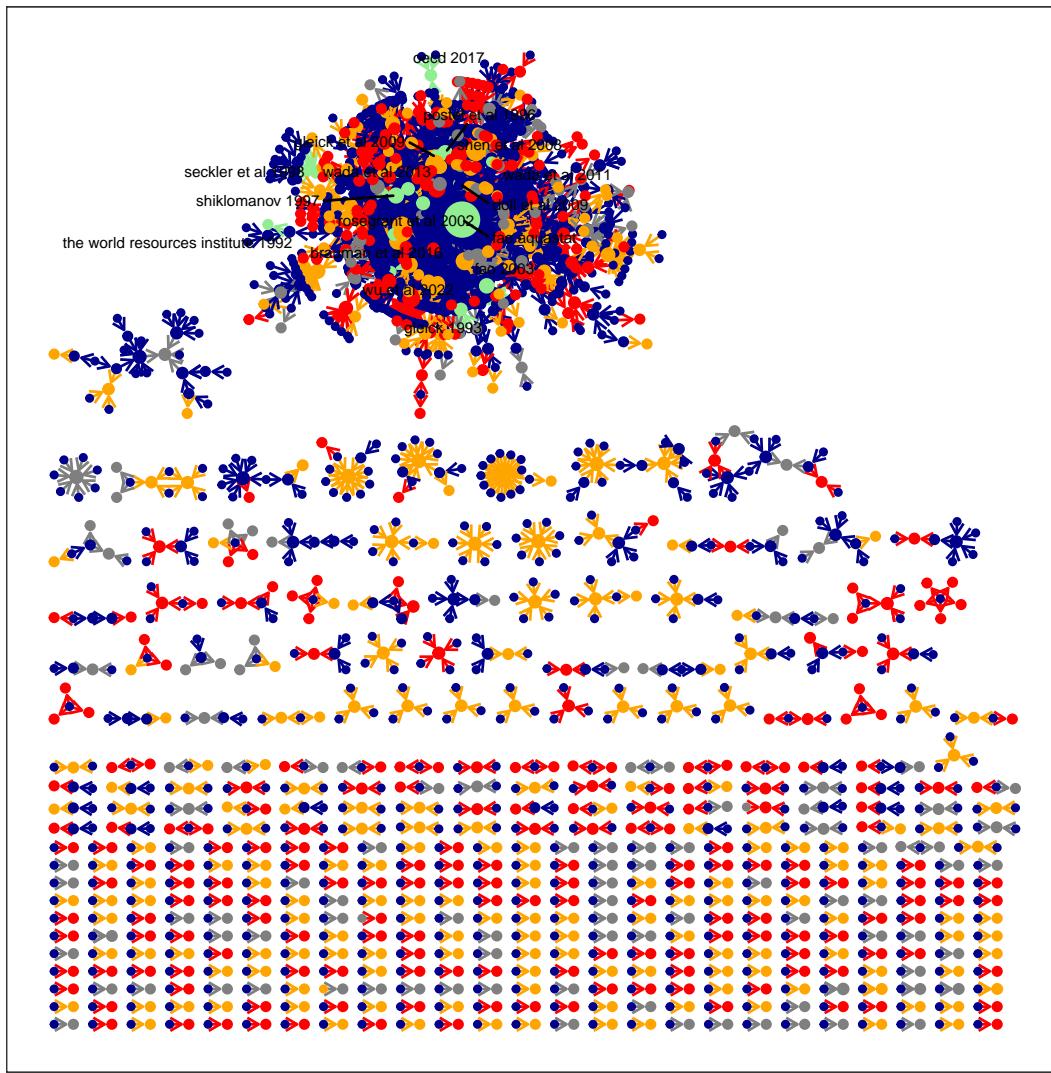
## $food

## Warning: Removed 762 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

```



```
##  
## $water  
  
## Warning: Removed 2909 rows containing missing values or values outside the scale range  
## (`geom_text_repel()`).
```



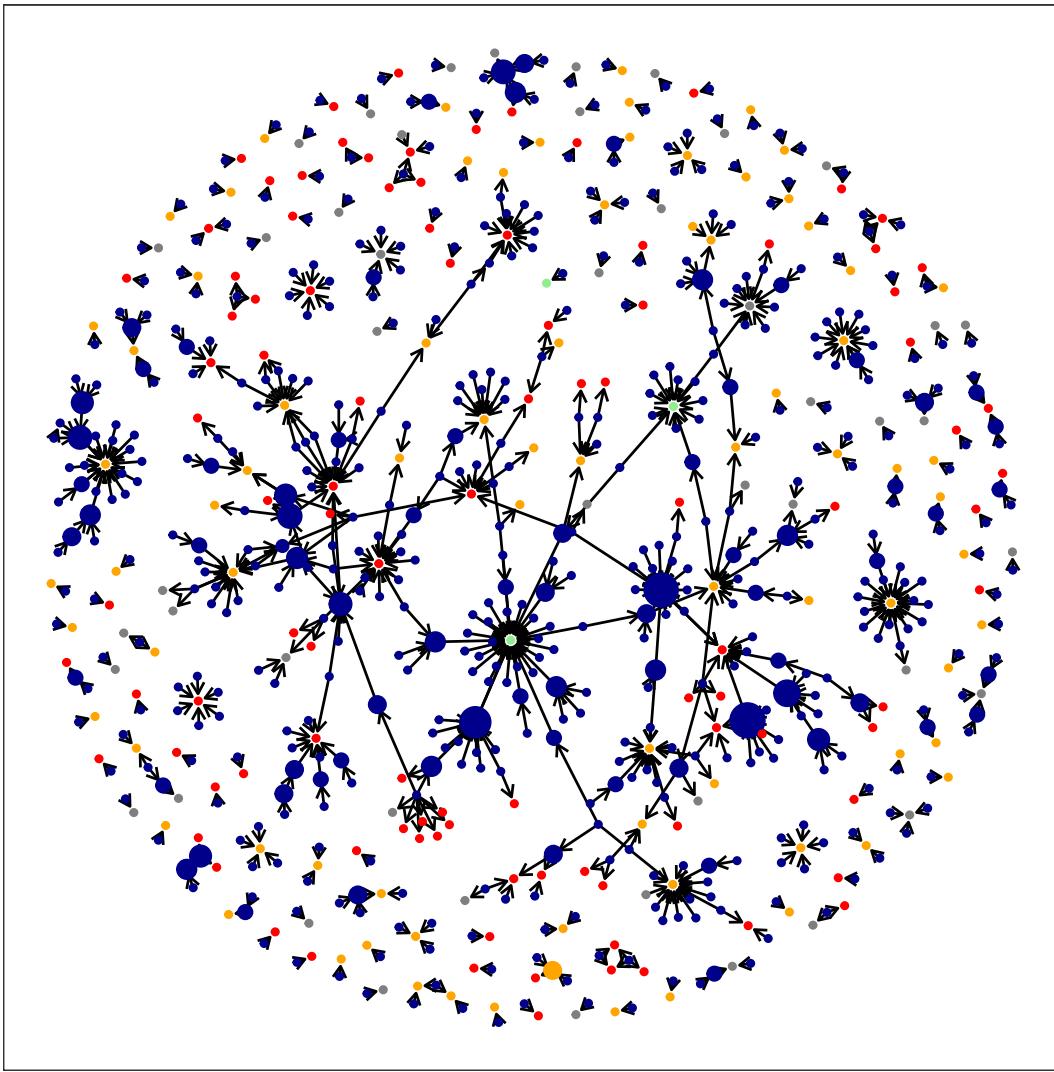
```
# Label the nodes with highest betweenness -----  
  
for (i in names(graph.final)) {  
  
  set.seed(seed)  
  
  p2[[i]] <- ggraph(graph.final[[i]], layout = "igraph", algorithm = "nicely") +  
    geom_edge_link(arrows = arrow(length = unit(1.8, 'mm')),  
                  end_cap = circle(1, "mm")) +  
    geom_node_point(aes(color = nature.claim, size = betweenness)) +  
    geom_node_text(aes(label = ifelse(betweenness >= min(betweenness.nodes[[i]]$betweenness),  
                        name, NA)),
```

```
repel = TRUE, size = 2.2) +
  labs(x = "", y = "") +
  scale_color_manual(name = "",
                      values = selected_colors) +
  theme_AP() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        legend.position = "right")
}

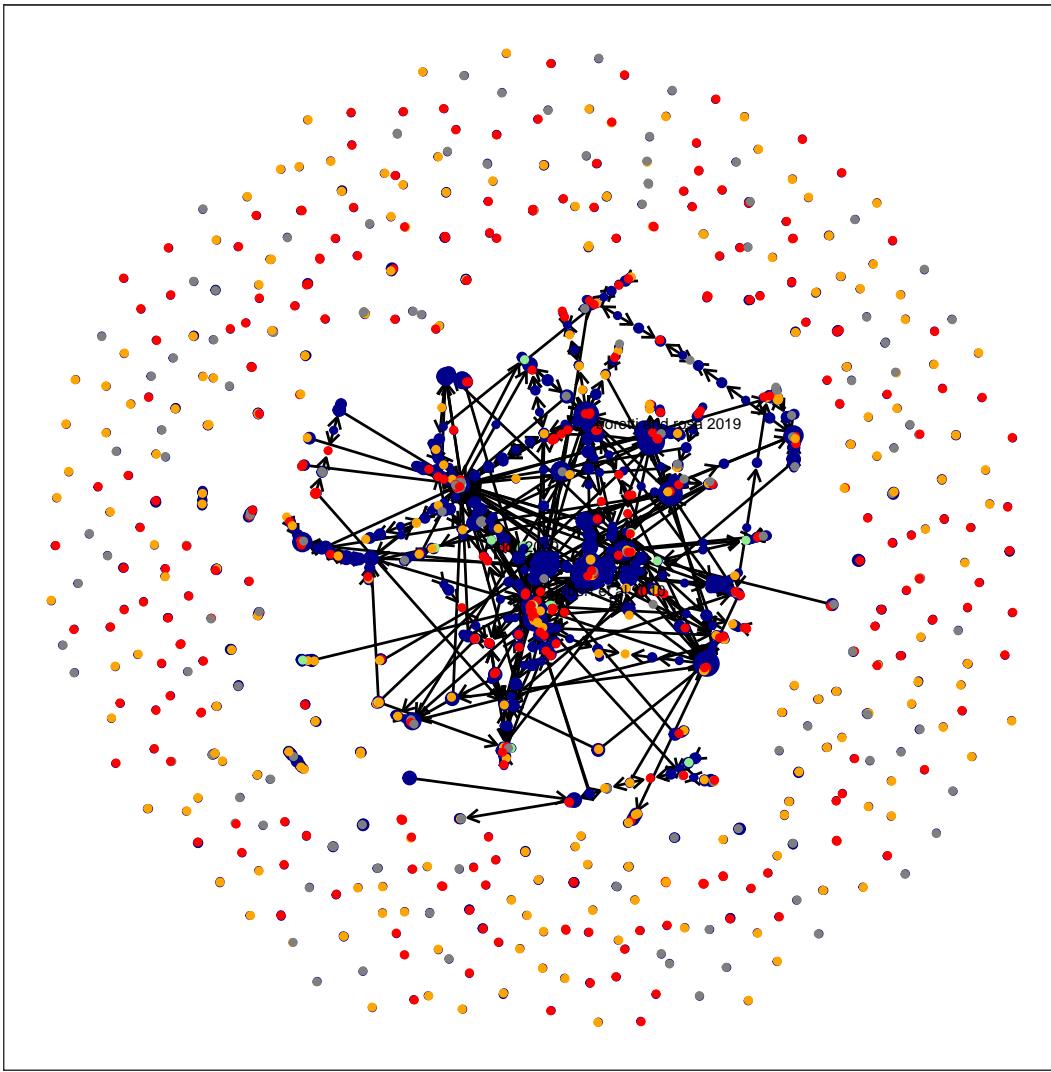
p2

## $food

## Warning: Removed 765 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).
```



```
##  
## $water  
  
## Warning: Removed 2922 rows containing missing values or values outside the scale range  
## (`geom_text_repel()`).
```



```
# by document.type-----
for (i in names(graph.final)) {
  set.seed(seed)

  p3[[i]] <- ggraph(graph.final[[i]], layout = "igraph", algorithm = "nicely") +
    geom_edge_link(arrows = arrow(length = unit(1.8, 'mm')),
                   end_cap = circle(1, "mm")) +
    geom_node_point(aes(color = document.type, size = degree)) +
    geom_node_text(aes(label = ifelse(degree >= min(degree.nodes[[i]]$degree), name, NA)),
                  repel = TRUE, size = 2.2) +
    labs(x = "", y = "") +
    scale_color_discrete(name = "") +
    theme_AP() +
    theme(axis.text.x = element_blank(),
          axis.ticks.x = element_blank(),
```

```

        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        legend.position = "right"
    }
}

```

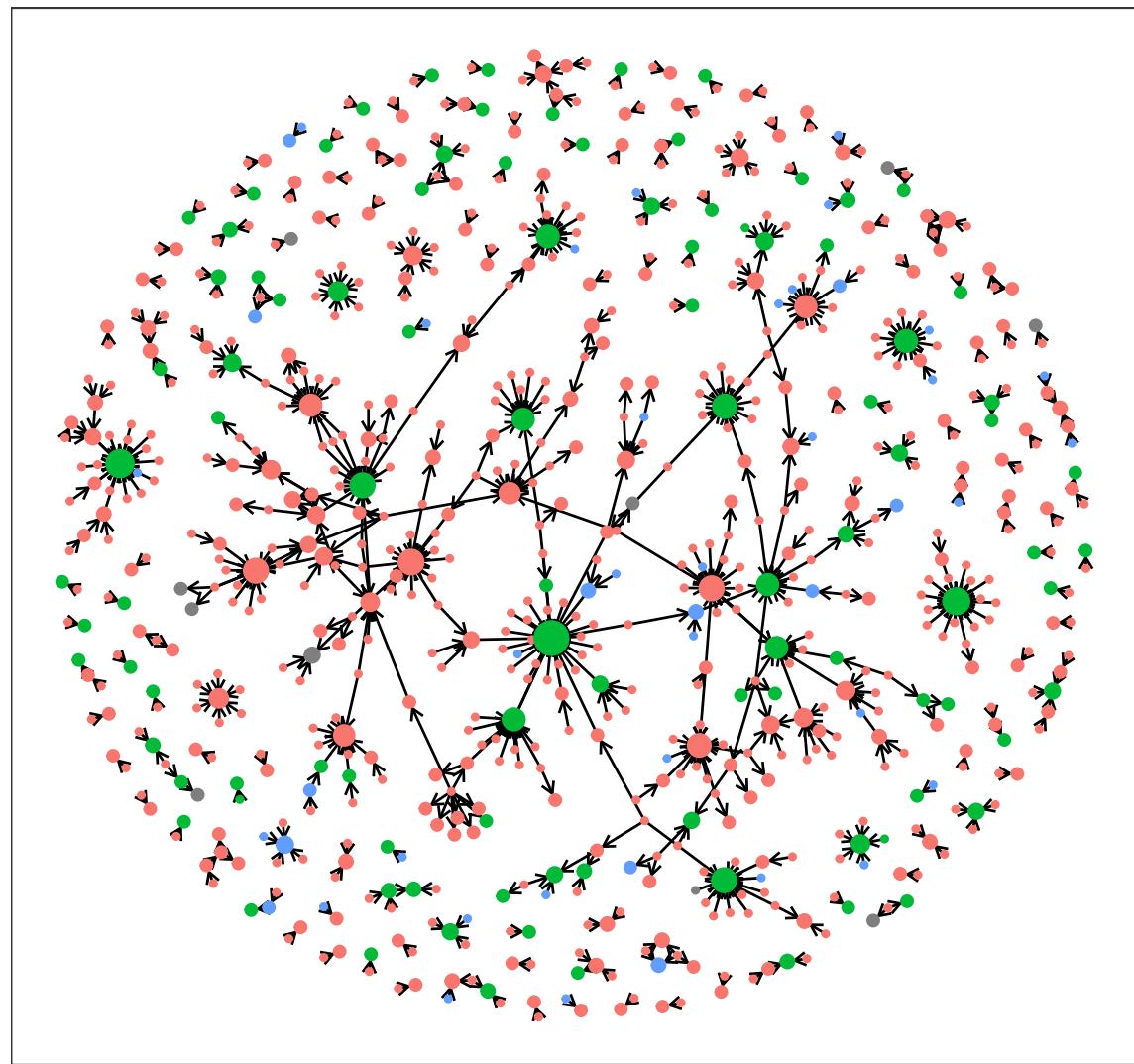
p3

\$food

```

## Warning: Removed 765 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

```



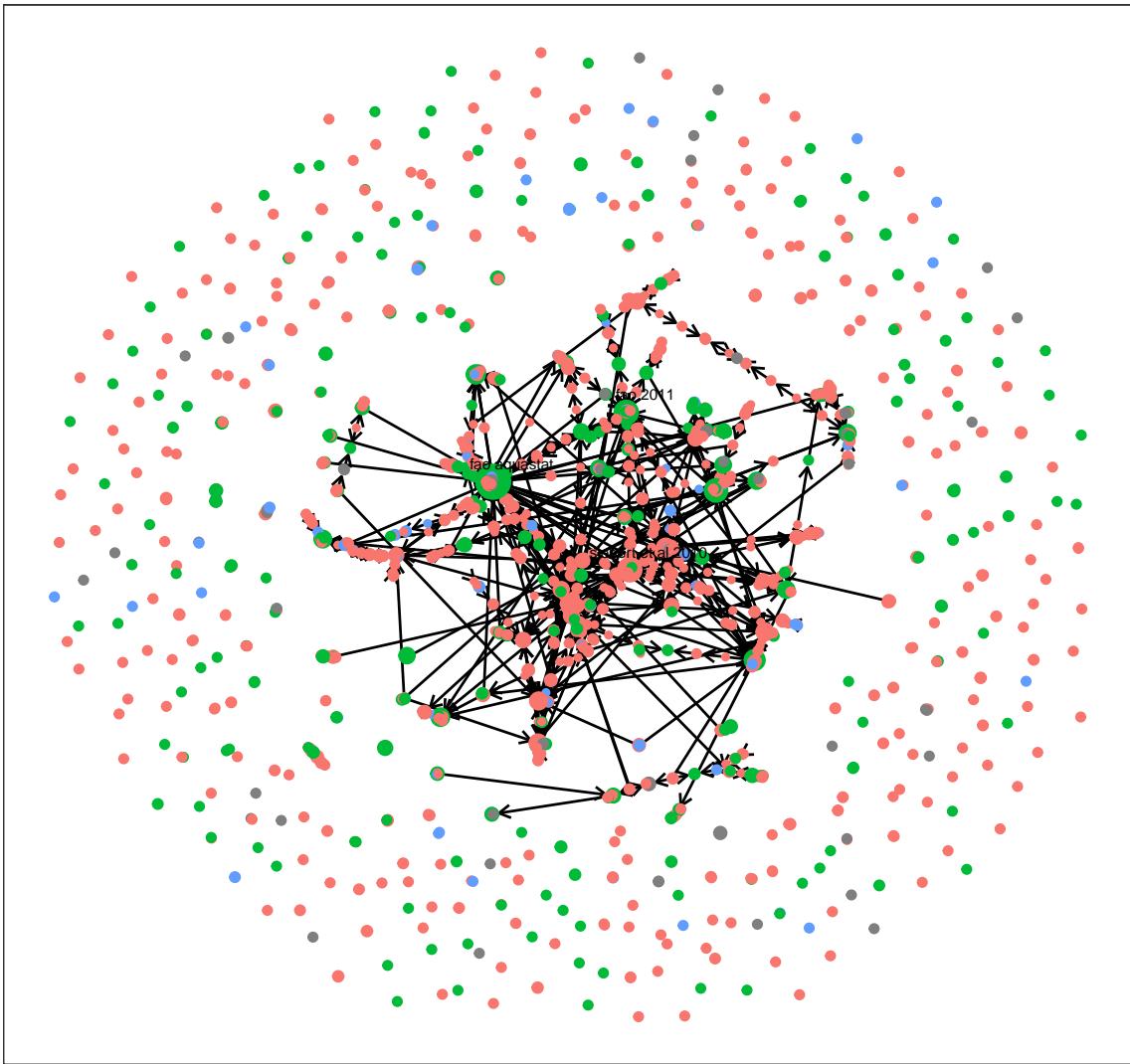
##

\$water

```

## Warning: Removed 2922 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

```



```
# Label nodes that are modelling exercises -----
for (i in names(graph.final)) {

  set.seed(seed)

  p4[[i]] <- ggraph(graph.final[[i]], layout = "igraph", algorithm = "nicely") +
    geom_edge_link(arrows = arrow(length = unit(1.8, 'mm')),
                   end_cap = circle(1, "mm")) +
    geom_node_point(aes(color = nature.claim)) +
    geom_node_text(aes(label = ifelse(nature.claim == "modelling", name, NA)),
                  repel = TRUE, size = 2.2) +
    labs(x = "", y = "") +
    scale_color_manual(name = "",
                      values = selected_colors) +
    theme_AP() +
    theme(axis.text.x = element_blank(),
```

```

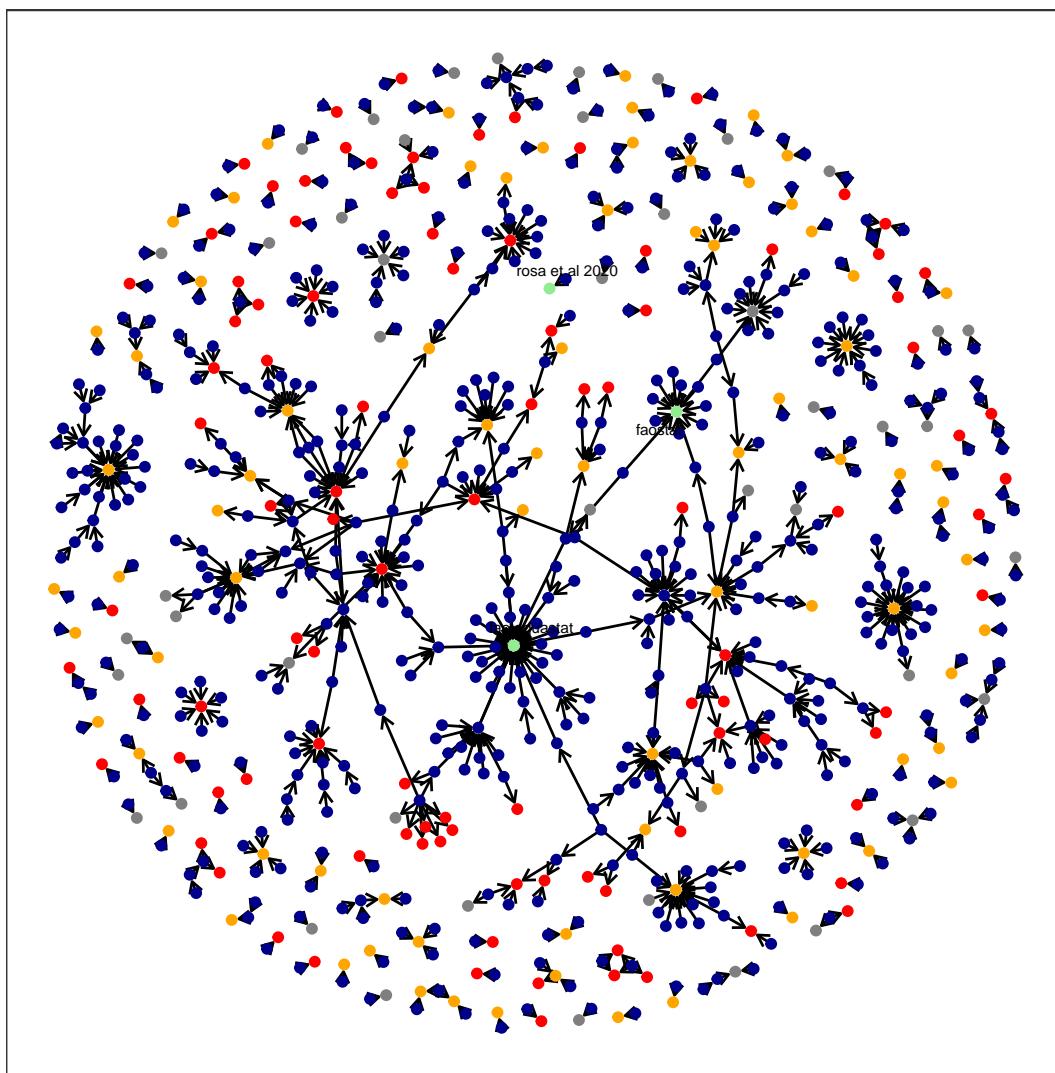
    axis.ticks.x = element_blank(),
    axis.text.y = element_blank(),
    axis.ticks.y = element_blank(),
    legend.position = "right")
}

p4

## $food

## Warning: Removed 762 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

```

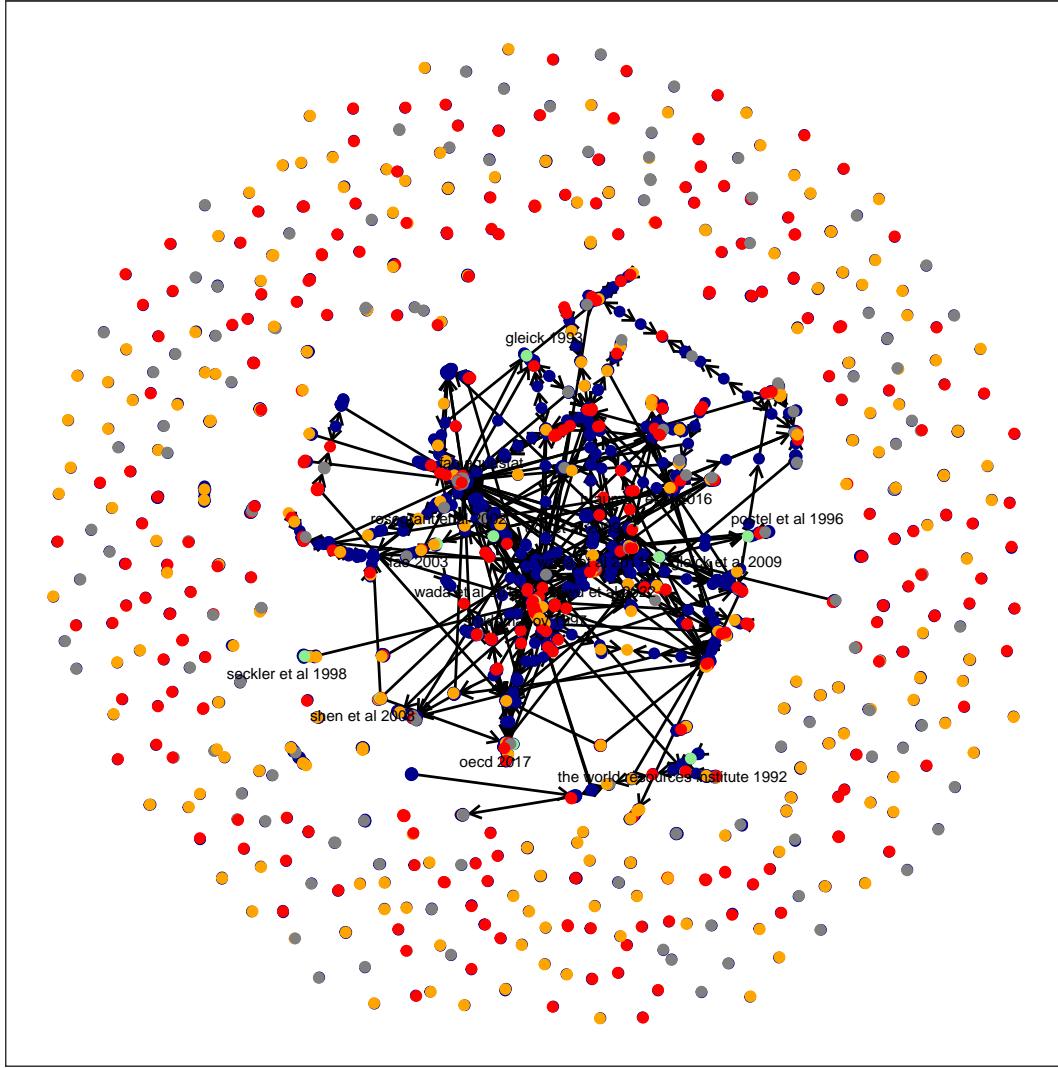


```

## 
## $water

## Warning: Removed 2909 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

```



```
# PLOT EVOLUTION NATURE CLAIM THROUGH TIME #####
out <- list()

for (i in names(graph.final)) {

  selected_colors <- c("darkblue", "lightgreen", "orange", "red", "grey")

  out[[i]] <- graph.final[[i]] %>%
    activate(nodes) %>%
    data.frame() %>%
    data.table() %>%
    .[, .N, .(year, nature.claim)] %>%
    ggplot(., aes(year, N, fill = nature.claim)) +
    scale_fill_manual(values = selected_colors, name = "") +
    geom_area() +
    scale_x_continuous(breaks = breaks_pretty(n = 3)) +
```

```

    theme_AP() +
    ggtitle(names(graph.final[i])) +
    theme(legend.position = "none",
          plot.title = element_text(size = 8))

}

legend <- get_legend(out[[2]] + theme(legend.position = "right"))

## Warning: Removed 4 rows containing non-finite outside the scale range
## (`stat_align()`).

## Warning in get_plot_component(plot, "guide-box"): Multiple components found;
## returning the first one. To return all, use `return_all = TRUE`.

bottom <- plot_grid(out[[1]] + labs(x = "Year", y = "Nº studies"),
                     out[[2]] + labs(x = "Year", y = ""))

```

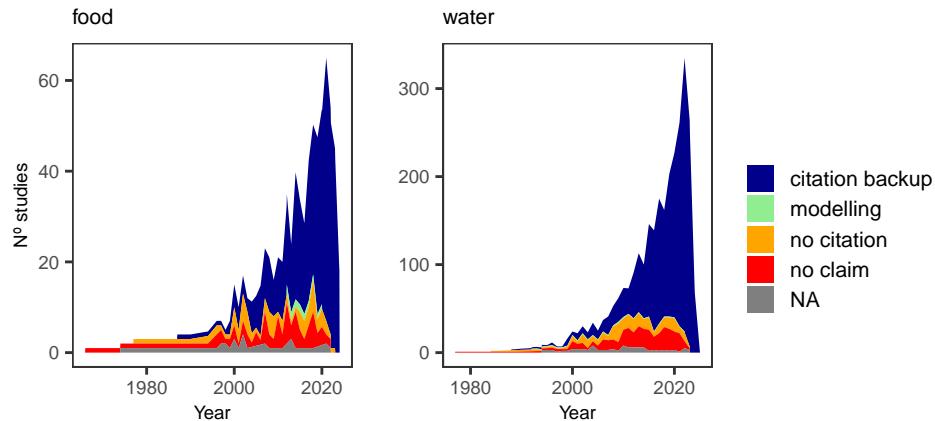
Warning: Removed 3 rows containing non-finite outside the scale range
(`stat_align()`).

Warning: Removed 4 rows containing non-finite outside the scale range
(`stat_align()`).

```

plot.network.claims <- plot_grid(bottom, legend, rel_widths = c(0.75, 0.25),
                                   ncol = 2)
plot.network.claims

```



1.3 Uncertainties turned into facts

```

# COUNT PROPORTION OF NODES THAT STATE AS FACT A CLAIM UTTERED AS UNCERTAIN #####
uncertainty_plot_fun <- function(graph) {

  # Extract name of all studies -----
  all.names <- graph %>%

```

```

activate(nodes) %>%
pull(name)

# Extract name of studies stating claim as fact ----

f.names <- graph %>%
  activate(nodes) %>%
  data.frame() %>%
  filter(classification == "F") %>%
  pull(name)

# Add names to edges ----

add.names.edges <- graph %>%
  activate(edges) %>%
  mutate(from.name = all.names[from],
         to.name = all.names[to])

# Calculate, for each study stating claim as fact, the studies it cites ----
out.classes <- lapply(f.names, function(x) {

  out_nodes <- add.names.edges %>%
    activate(edges) %>%
    filter(from.name == x) %>%
    pull(to.name)

})

# unlist names of studies cited by studies uttering claim as fact ----

di <- sort(unlist(out.classes))

# Extract only those that do not state claim as fact ----

nodes.no.fact <- graph %>%
  activate(nodes) %>%
  data.frame() %>%
  data.table() %>%
  .[name %in% di] %>%
  .[!classification == "F"] %>%
  .$name

name.edges <- add.names.edges %>%
  activate(edges) %>%
  data.frame() %>%
  filter(from.name %in% f.names & to.name %in% nodes.no.fact) %>%
  .[, c("from.name", "to.name")] %>%

```

```

c() %>%
  unlist() %>%
  unique()

output <- add.names.edges %>%
  activate(nodes) %>%
  filter(name %in% name.edges) %>%
  activate(edges) %>%
  filter(from.name %in% name.edges & to.name %in% name.edges)

return(output)
}

# PLOT GRAPH UNCERTAINTIES TURNED INTO FACTS #####
out <- lapply(graph.final, function(x) uncertainty_plot_fun(x))

p7 <- list()

for (i in names(out)) {

  set.seed(seed)

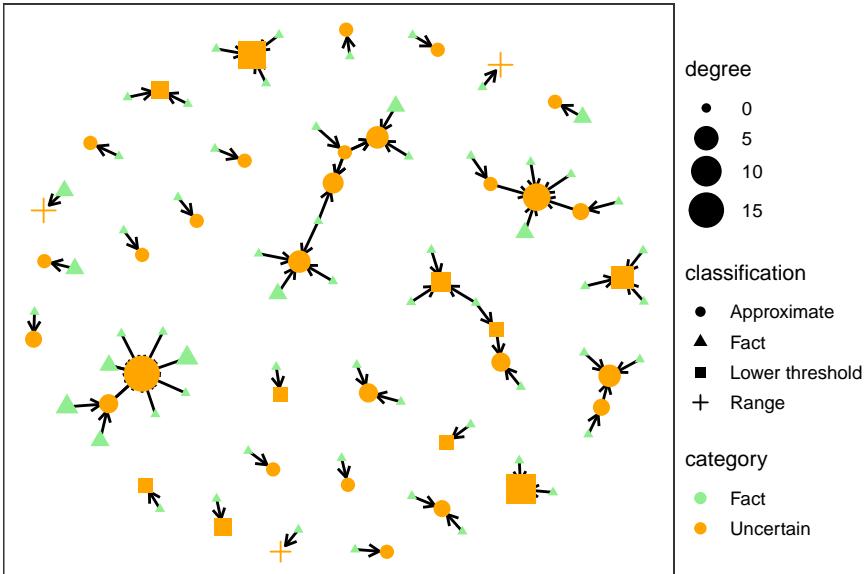
  p7[[i]] <- ggraph(out[[i]], layout = "igraph", algorithm = "nicely") +
    geom_edge_link(arrows = arrow(length = unit(1.6, 'mm')),
                   end_cap = circle(1, "mm")) +
    geom_node_point(aes(color = category, size = degree, shape = classification)) +
    scale_color_manual(values = c("lightgreen", "orange")) +
    scale_shape_discrete(labels = c("Approximate", "Fact", "Lower threshold", "Range", "Upper")),
    geom_node_text(aes(label = ifelse(degree >= min(degree.nodes[[i]]$degree), name, NA)),
                  repel = TRUE, size = 2.2) +
    labs(x = "", y = "") +
    theme_AP() +
    theme(axis.text.x = element_blank(),
          axis.ticks.x = element_blank(),
          axis.text.y = element_blank(),
          axis.ticks.y = element_blank(),
          legend.position = "right",
          legend.text = element_text(size = 7.2))
}

p7

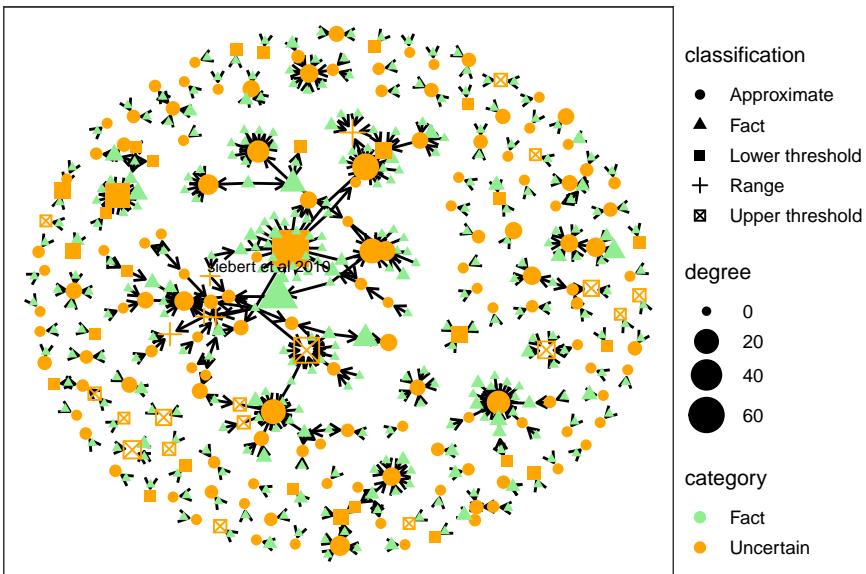
## $food

## Warning: Removed 99 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

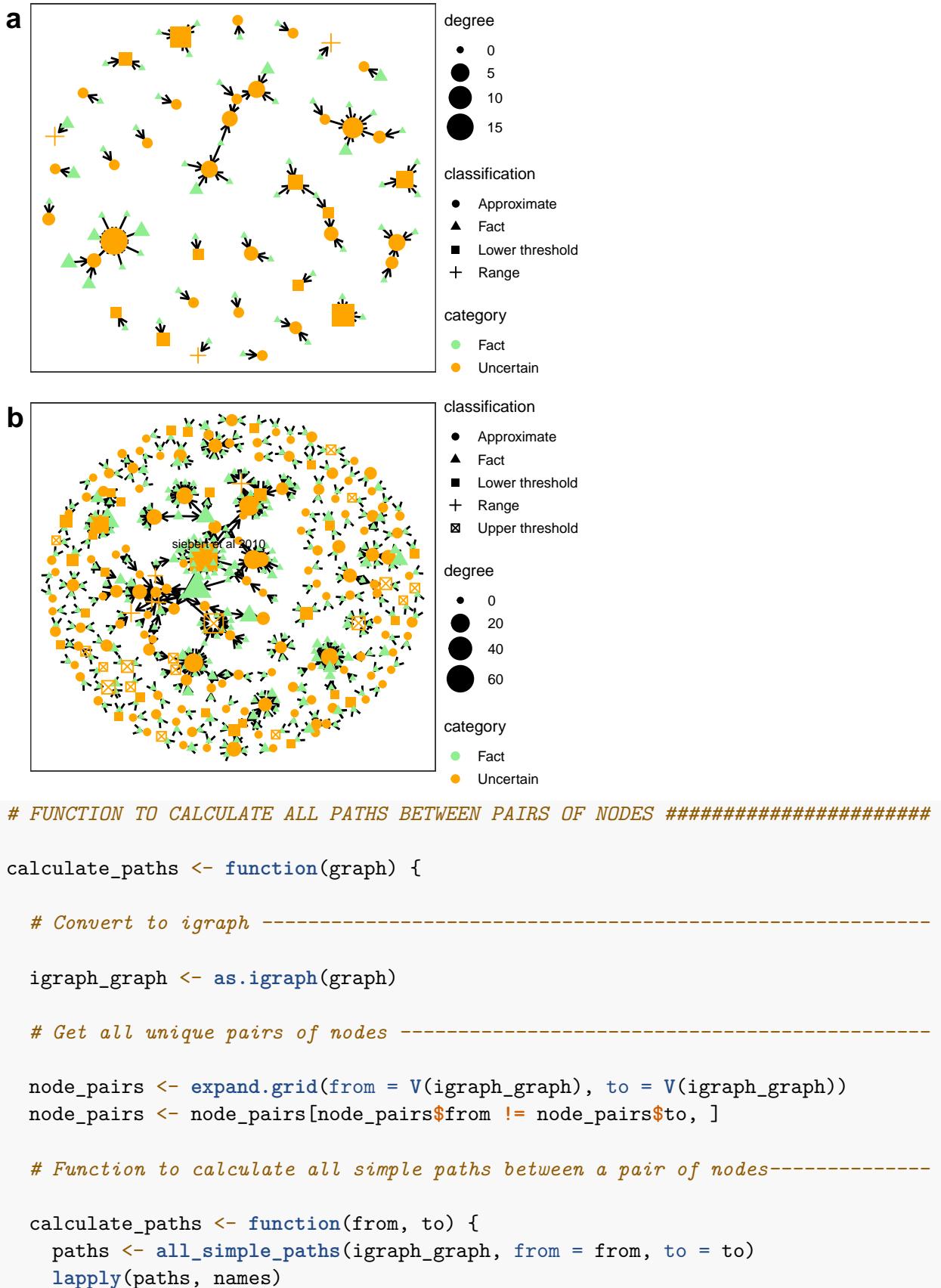
```



```
##  
## $water  
  
## Warning: Removed 496 rows containing missing values or values outside the scale range  
## (`geom_text_repel()`).
```



```
# MERGE PLOTS #####  
  
plot_grid(p7[[1]], p7[[2]], ncol = 1, labels = "auto")  
  
## Warning: Removed 99 rows containing missing values or values outside the scale range  
## (`geom_text_repel()`).  
  
## Warning: Removed 496 rows containing missing values or values outside the scale range  
## (`geom_text_repel()`).
```



```

}

# Apply the function to all node pairs and unnest the results-----
all_paths <- node_pairs %>%
  rowwise() %>%
  mutate(paths = list(calculate_paths(from, to))) %>%
  unnest(cols = c(paths))

out <- sum(sapply(all_paths$paths, function(x) length(x)))

return(out)

}

# CALCULATE ALL PATHS / PATHS TURNING HYPOTHESIS INTO FACTS #####
all.paths <- hypothesis.into.facts.paths <- list()

for (i in names(graph.final)) {

  all.paths[[i]] <- calculate_paths(graph.final[[i]])
  hypothesis.into.facts.paths[[i]] <- uncertainty_plot_fun(graph.final[[i]]) %>%
    calculate_paths()

}

# Print results: proportion of paths turning uncertainties into facts -----
for (i in names(all.paths)) {
  print(hypothesis.into.facts.paths[[i]] / all.paths[[i]])
}

## [1] 0.08695652
## [1] 0.08385644

```

2 Proportion of paths ending up in no claim, no citation or modelling nodes

```

# DEFINE FUNCTION #####
proportion_paths <- function(graph) {

  # Turn into data.frame -----
  end_nodes <- graph %>%

```

```

activate(nodes) %>%
filter(degree.out == 0) %>%
data.frame()

end_node_indices <- end_nodes$name

# Loop to store all paths to all end-nodes -----
all_paths <- list()

for (v in igraph::V(as.igraph(graph))) {

  paths_from_v <- igraph::all_simple_paths(as.igraph(graph),
                                             from = v,
                                             to = end_node_indices)

  if (length(paths_from_v) > 0) {

    all_paths <- c(all_paths, paths_from_v)
  }
}

# Extract the label of the last node in each path ----

end_labels <- sapply(all_paths, function(path) {

  last_node <- tail(path, 1)
  last_node_name <- V(as.igraph(graph))[last_node]$name

  graph %>%
    activate(nodes) %>%
    filter(name == last_node_name) %>%
    pull(nature.claim)
})

# Proportion of paths ending in "no citation", "no claim" and "modelling" ----

no_citation_paths <- sum(end_labels == "no citation", na.rm = TRUE)
no_claim_paths <- sum(end_labels == "no claim", na.rm = TRUE)
modelling_paths <- sum(end_labels == "modelling", na.rm = TRUE)
na_paths <- sum(is.na(end_labels))
total_paths <- length(end_labels)

proportion_no_citation <- no_citation_paths / total_paths
proportion_no_claim <- no_claim_paths / total_paths
proportion_modelling <- modelling_paths / total_paths
proportion_na <- na_paths / total_paths

```

```

# Wrap up for output -----
output <- data.table("no citation" = proportion_no_citation,
                      "no claim" = proportion_no_claim,
                      "modelling" = proportion_modelling,
                      "NA" = proportion_na)
return(output)

}

# RUN FUNCTION ##### #####
out <- lapply(graph.final, function(graph) proportion_paths(graph))
out

## $food
##   no citation  no claim modelling      NA
##           <num>     <num>     <num>     <num>
## 1:  0.3884786 0.3929099 0.09896603 0.1196455
##
## $water
##   no citation  no claim modelling      NA
##           <num>     <num>     <num>     <num>
## 1:  0.3043085 0.2717686 0.3097318 0.114191
# SUM PROPORTION NO CLAIM AND NO CITATION #####
lapply(out, function(x) x[, `no citation` + `no claim`])

## $food
## [1] 0.7813885
##
## $water
## [1] 0.5760771
# PLOT PROPORTION OF PATHS ENDING IN MODELLING, NO CLAIM AND NO CITATION #####
rbindlist(out, idcol = "belief") %>%
  melt(., measure.vars = colnames(.)[-1]) %>%
  ggplot(., aes(belief, value, fill = variable)) +
  geom_bar(stat = "identity",
            position = position_dodge(0.5)) +
  labs(x = "", y = "Fraction") +
  scale_fill_manual(values = c("orange", "red", "lightgreen", "grey"),
                    name = "") +
  theme_AP() +
  theme(legend.position = c(0.6, 0.9))

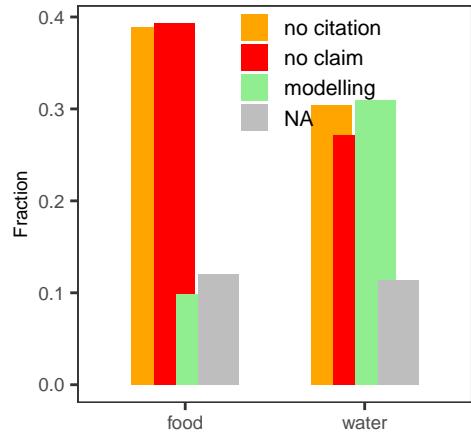
## Warning: A numeric `legend.position` argument in `theme()` was deprecated in ggplot2

```

```

## 3.5.0.
## i Please use the `legend.position.inside` argument of `theme()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

```



2.1 Network through time

```

# PLOT NETWORK THROUGH TIME #####
plot.years <- list()

for (i in c("water", "food")) {

  # Extract vector with names -----
  location_aquastat <- graph.final[[i]] %>%
    activate(nodes) %>%
    data.frame() %>%
    pull(name) %>%
    grep("aquastat", .)

  location_faostat <- graph.final[[i]] %>%
    activate(nodes) %>%
    data.frame() %>%
    pull(name) %>%
    grep("faostat", .)

  # Extract vector with years -----
  v_years <- graph.final[[i]] %>%
    activate(nodes) %>%
    data.frame() %>%
    pull(year)
}

```

```

# Substitute fao aquastat/faostat without year with the oldest citations ----

v_years[location_aquastat] <- oldest.aquastat.cite
v_years[location_faostat] <- oldest.faostat.cite

# Find NA values ----

na_indices <- is.na(v_years)
sum(na_indices)

# Generate random values to replace NA ----

random_values <- sample(2000:2020, sum(na_indices), replace = TRUE)

# Replace NA with random values ----

v_years[na_indices] <- random_values

# Define the coordinates-----

y_positions <- runif(length(v_years), min = -3, max = 3) # Random y-axis position
layout <- cbind(v_years, y_positions) # Use actual years for x-axi
layout_matrix <- as.matrix(layout)
colnames(layout_matrix) <- c("x", "y")

# PLOT NETWORK THROUGH TIME #####
# Set seed ----

set.seed(seed)

# Plot ----

plot.years[[i]] <- ggraph(graph.final[[i]], layout = layout_matrix, algorithm = "nicely") +
  geom_edge_link(arrows = arrow(length = unit(1.8, "mm")),
                 end_cap = circle(1, "mm"),
                 color = "grey",
                 alpha = 0.4) +
  geom_node_point(aes(color = nature.claim, size = degree)) +
  geom_node_text(aes(label = ifelse(nature.claim == "modelling", name, NA)),
                repel = TRUE, size = 2.5) +
  scale_color_manual(name = "",
                     values = selected_colors) +
  scale_x_continuous(name = "Year",
                     limits = range(v_years),
                     breaks = seq(min(v_years),
                                 max(v_years), by = 5)) +

```

```

    labs(x = "Year", y = "") +
    theme_AP() +
    theme(axis.text.y = element_blank(),
          axis.ticks.y = element_blank())

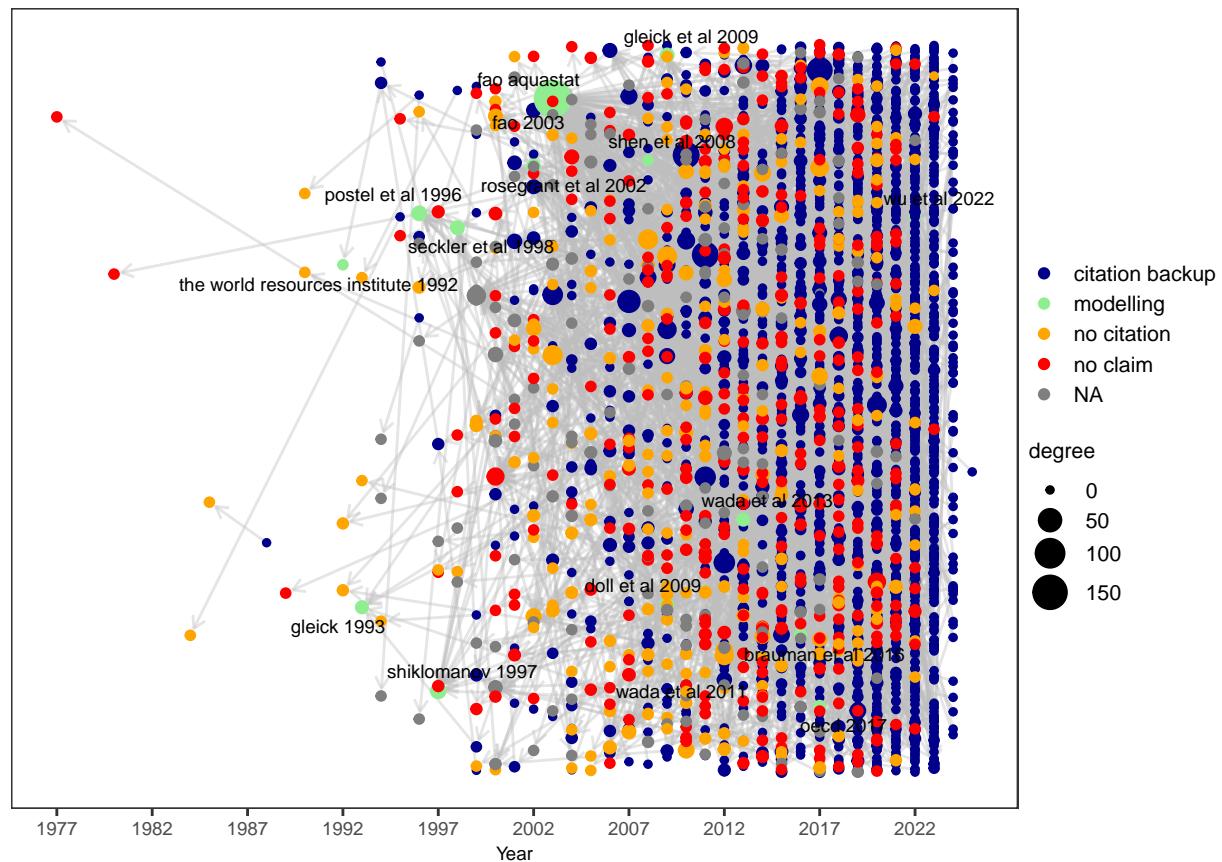
}

plot.years

## $water

## Warning: Removed 2909 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

```



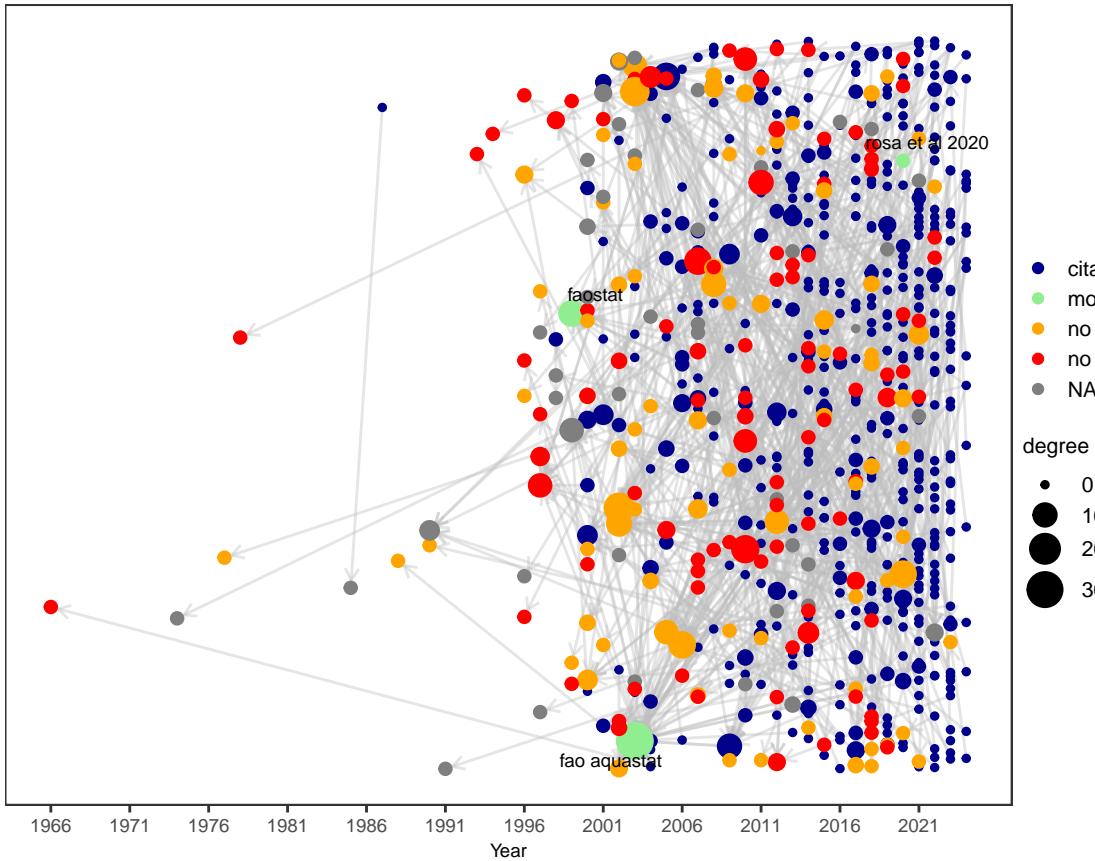
```

##

## $food

## Warning: Removed 762 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

```



```
# ANOTHER VISUALIZATION FOR YEARS BASED ON POLAR COORDINATES #####
#####

plot.years <- list()

for (i in c("water", "food")) {

  # Replace NA values in year with random samples from 2000 to 2020 -----
  g <- graph.final[[i]] %>%
    activate(nodes) %>%
    mutate(year = ifelse(is.na(year), sample(2000:2020, replace = TRUE), year)) %>%
    mutate(
      year_normalized = (year - min(year)) / (2024 - min(year)), # Normalize relative to 2024
      radius = year_normalized
    )

  # Assign the calculated positions -----
  g <- g %>%
    mutate(
      angle = seq(0, 2 * pi, length.out = n() + 1)[1:n()],
      x = radius * cos(angle),
      y = radius * sin(angle)
    )
}
```

```

)

# Determine the range of years -----
min_year <- min(g %>% pull(year))

# Dynamically determine the start year for the concentric circles -----
start_year <- floor(min_year / 10) * 10 # Round down to the nearest decade
end_year <- 2024 # Explicitly set the end year to 2024
year_intervals <- seq(start_year, end_year, by = 10)

# Create concentric circles every ten years -----
circle_data <- lapply(year_intervals, function(yr) {
  r <- (yr - min_year) / (end_year - min_year)
  tibble(
    x = r * cos(seq(0, 2 * pi, length.out = 100)),
    y = r * sin(seq(0, 2 * pi, length.out = 100)),
    year = yr,
    label_x = r, # Label position on the x-axis (angle = 0)
    label_y = 0 # Label position on the y-axis (angle = 0)
  )
}) %>% bind_rows()

# Remove duplicate labels -----
label_data <- circle_data %>%
  distinct(year, .keep_all = TRUE) # Keep only unique year labels

# Plot -----
# Set seed -----
set.seed(seed)

plot.years[[i]] <- ggraph(g, layout = "manual", x = x, y = y) +
  # Add concentric circles
  geom_edge_link(arrows = arrow(length = unit(1.8, "mm")),
                 end_cap = circle(1, "mm"),
                 alpha = 0.07,
                 aes(color = edge_color)) +
  scale_edge_color_manual(values = selected_colors, guide = "none") +
  geom_node_point(aes(color = nature.claim, size = degree)) +
  geom_node_text(aes(label = ifelse(nature.claim == "modelling", name, NA)),
                 repel = TRUE, size = 2.5) +
  scale_color_manual(name = "",
                     values = selected_colors) +
  geom_path(data = circle_data, aes(x = x, y = y, group = factor(year)),
            color = "black", linetype = "dashed") +
  # Add year labels

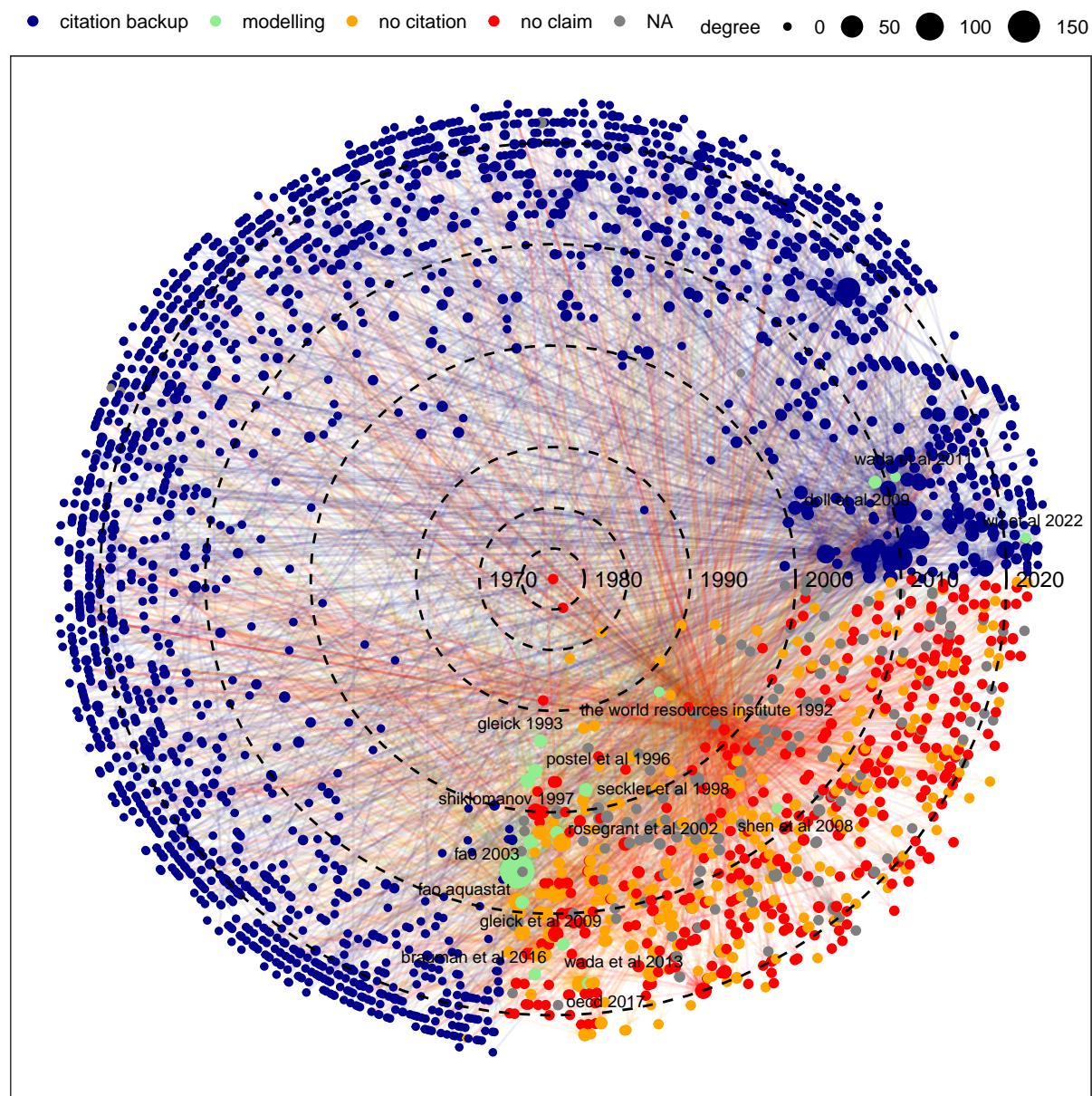
```

```
geom_text(data = label_data, aes(x = label_x, y = label_y, label = year),
           hjust = -0.2, vjust = 0.5, size = 3) +
  labs(x = "", y = "") +
  theme_AP() +
  theme(axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        legend.position = "top")
}

plot.years

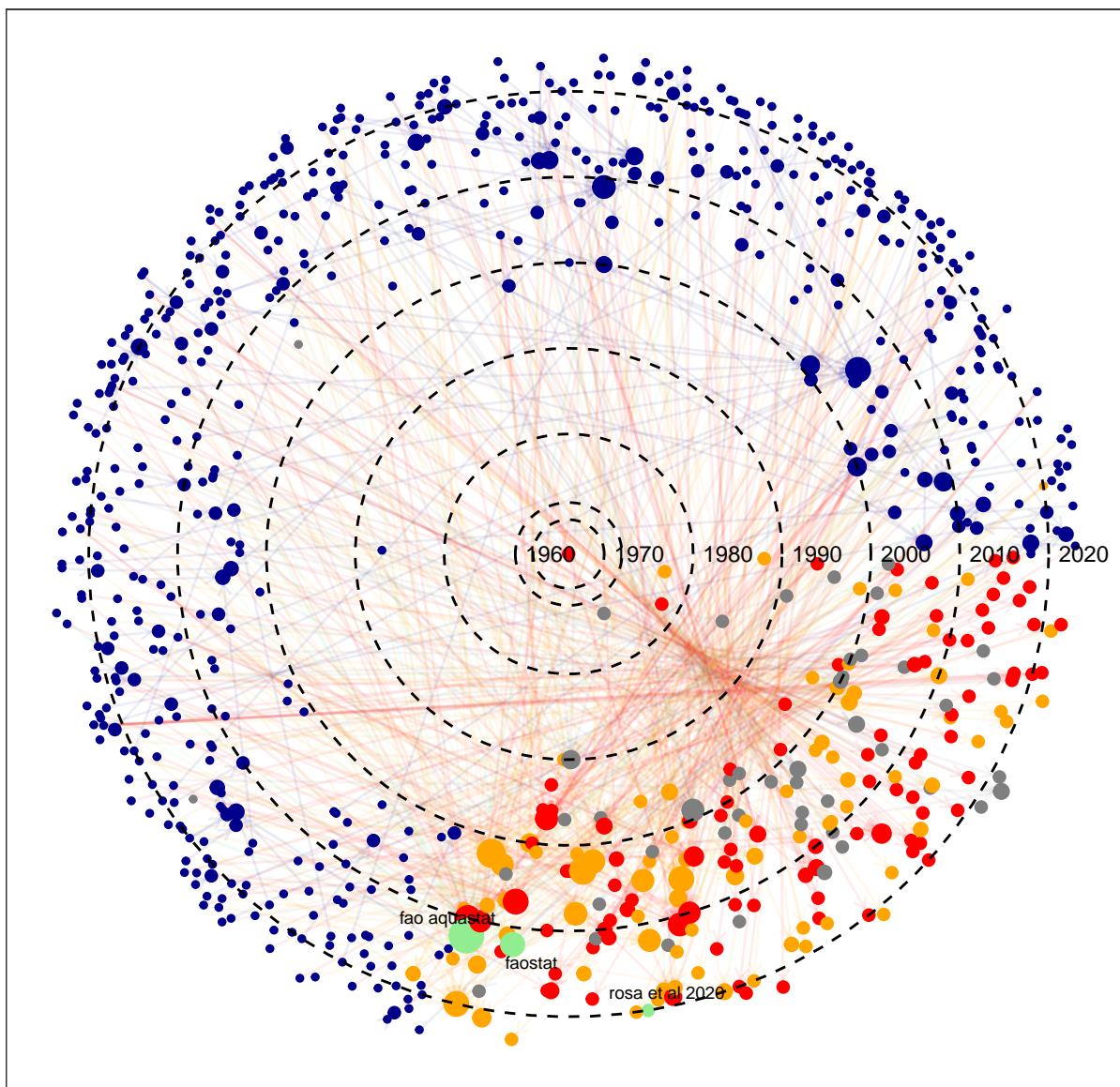
## $water

## Warning: Removed 2909 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).
```



```
##  
## $food  
  
## Warning: Removed 762 rows containing missing values or values outside the scale range  
## (`geom_text_repel()`).
```

● citation backup ● modelling ● no citation ● no claim ● NA degree ● 0 ● 10 ● 20 ● 30



```
# FUNCTION TO PLOT EVOLUTION OF NETWORK THROUGH TIME #####
network_through_time_fun <- function(graph, Year, seed) {

  # Extract all names -----
  all.names <- graph %>%
    activate(nodes) %>%
    pull(name)

  # Add names to edges -----
  add.names.edges <- graph %>%
    activate(edges) %>%
    
```

```

    mutate(from.name = all.names[from],
           to.name = all.names[to])

# Extract nodes by year -----
names.targeted <- add.names.edges %>%
  activate(edges) %>%
  filter(year < Year) %>%
  data.frame() %>%
  .[, c("from.name", "to.name")] %>%
  c() %>%
  unlist() %>%
  unique()

name.nodes <- add.names.edges %>%
  activate(nodes) %>%
  filter(name %in% names.targeted) %>%
  activate(edges) %>%
  filter(from.name %in% names.targeted & to.name %in% names.targeted)

set.seed(seed)

# Plot -----
out <- ggraph(name.nodes, layout = "igraph", algorithm = "nicely") +
  geom_edge_link(arrows = arrow(length = unit(1, 'mm')),
                 end_cap = circle(0.3, "mm")) +
  geom_node_point(aes(color = nature.claim, size = degree)) +
  geom_node_text(aes(label = ifelse(nature.claim == "modelling", name, NA)),
                 repel = TRUE, size = 2.2) +
  scale_color_manual(name = "",
                     values = selected_colors) +
  labs(x = "", y = "") +
  theme_AP() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        legend.position = "none")

return(out)
}

# DEFINE YEARS OF INTEREST #####
years.vector <- c(seq(2000, 2020, 10), 2024)

```

```

# RUN FUNCTION ##### #####
plots.through.time <- list()

for (i in names(graph.final)) {

  plots.through.time[[i]] <- lapply(years.vector, function(year)
    network_through_time_fun(graph = graph.final[[i]], Year = year, seed = seed) +
    ggtitle(year))
}

da <- list()

for (i in names(plots.through.time)) {

  for (j in 1:length(plots.through.time[[i]])) {

    da[[i]][[j]] <- plots.through.time[[i]][[j]] +
      geom_node_point(aes(color = nature.claim)) +
      theme(axis.text.x = element_blank(),
            axis.ticks.x = element_blank(),
            axis.text.y = element_blank(),
            axis.ticks.y = element_blank(),
            legend.position = "right")
  }
}

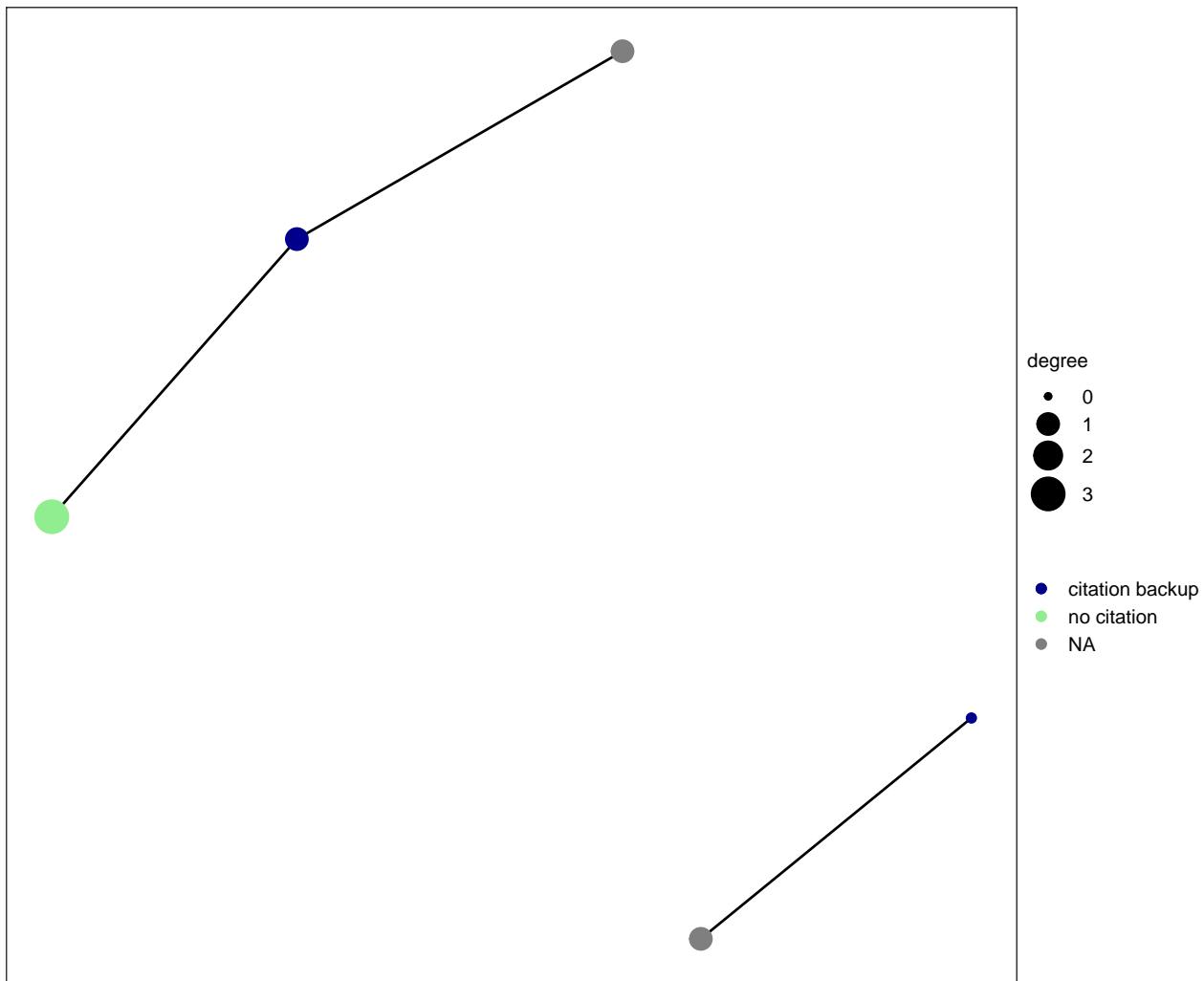
da

## $food
## $food[[1]]

## Warning: Removed 5 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

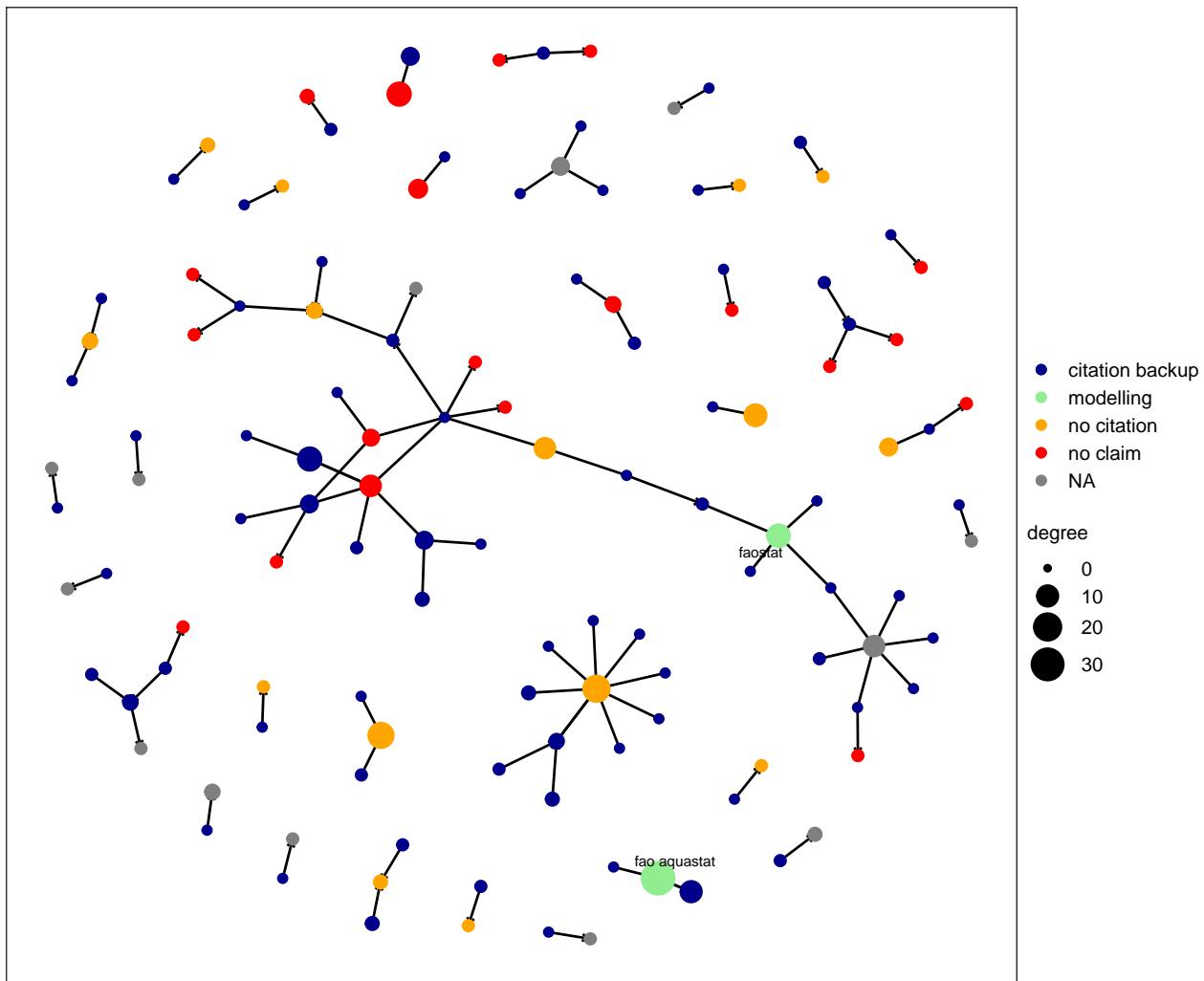
```

2000



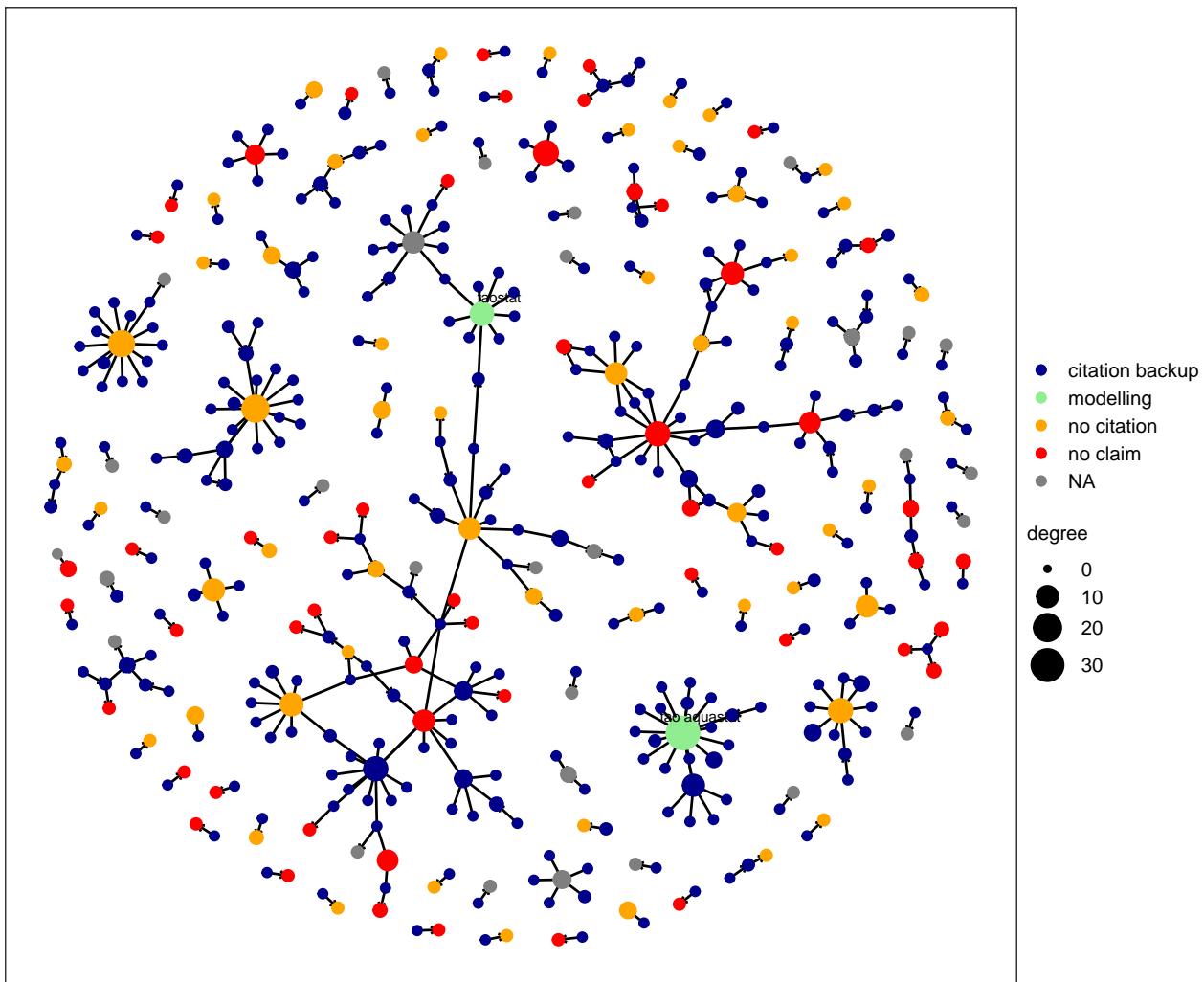
```
##  
## $food[[2]]  
  
## Warning: Removed 123 rows containing missing values or values outside the scale range  
## (`geom_text_repel()`).
```

2010



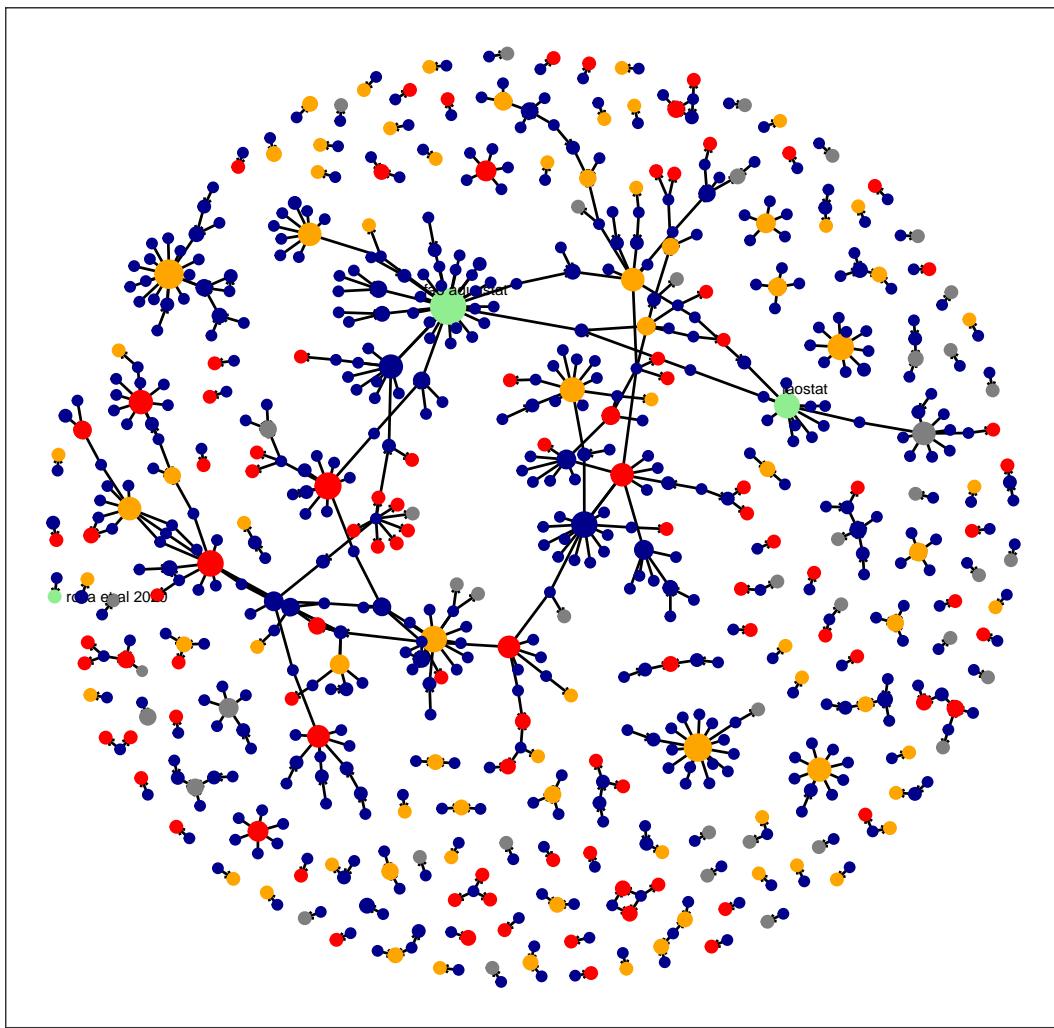
```
##  
## $food[[3]]  
  
## Warning: Removed 449 rows containing missing values or values outside the scale range  
## (`geom_text_repel()`).
```

2020



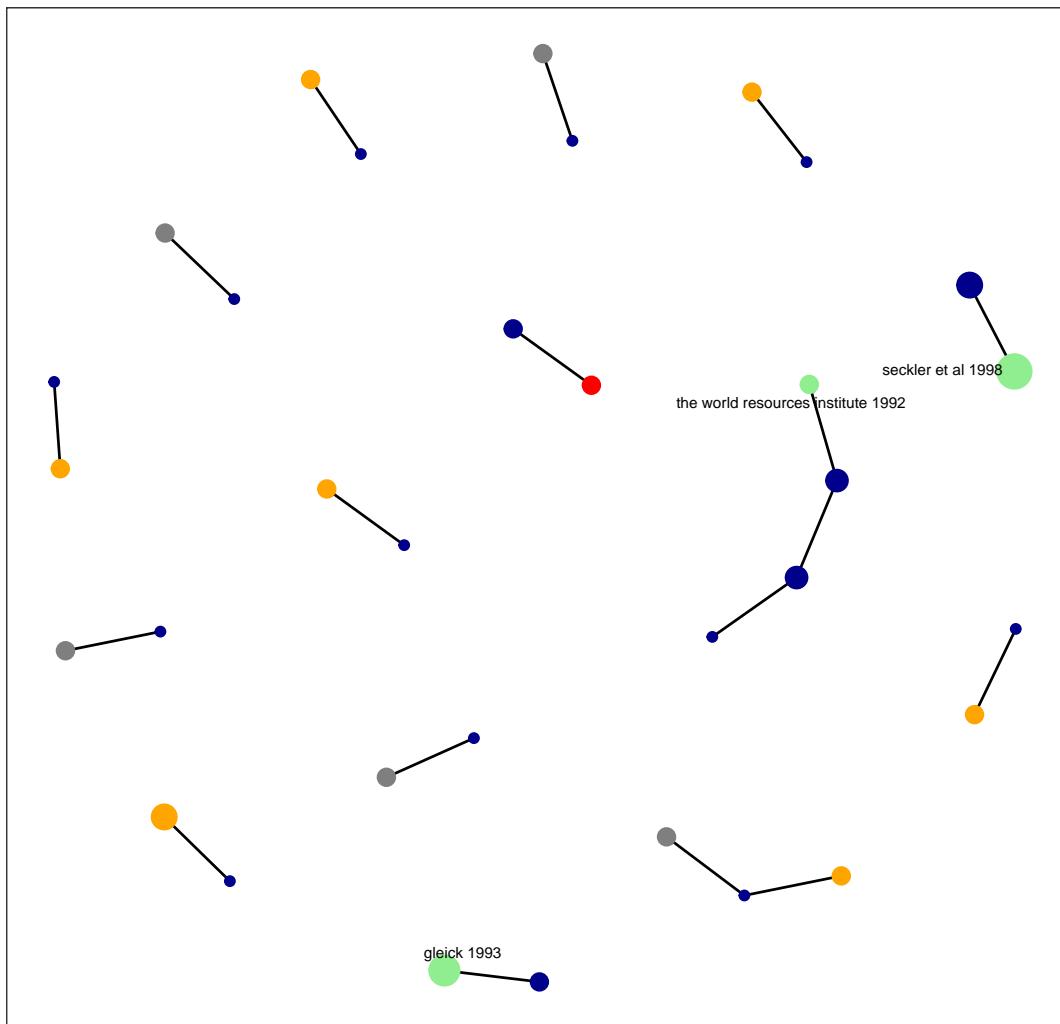
```
##  
## $food[[4]]  
  
## Warning: Removed 724 rows containing missing values or values outside the scale range  
## (`geom_text_repel()`).
```

2024



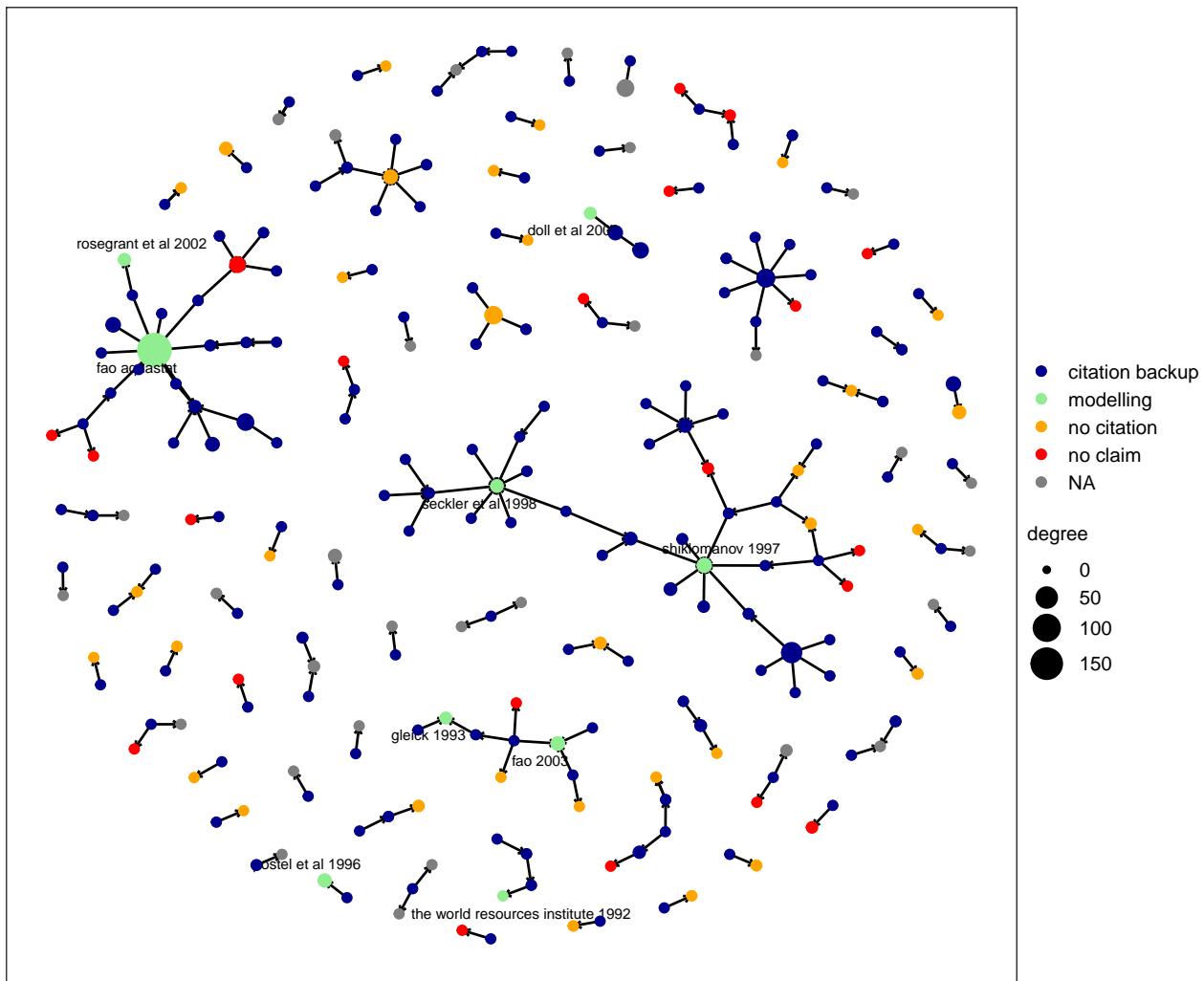
```
##  
##  
## $water  
## $water[[1]]  
  
## Warning: Removed 30 rows containing missing values or values outside the scale range  
## (`geom_text_repel()`).
```

2000



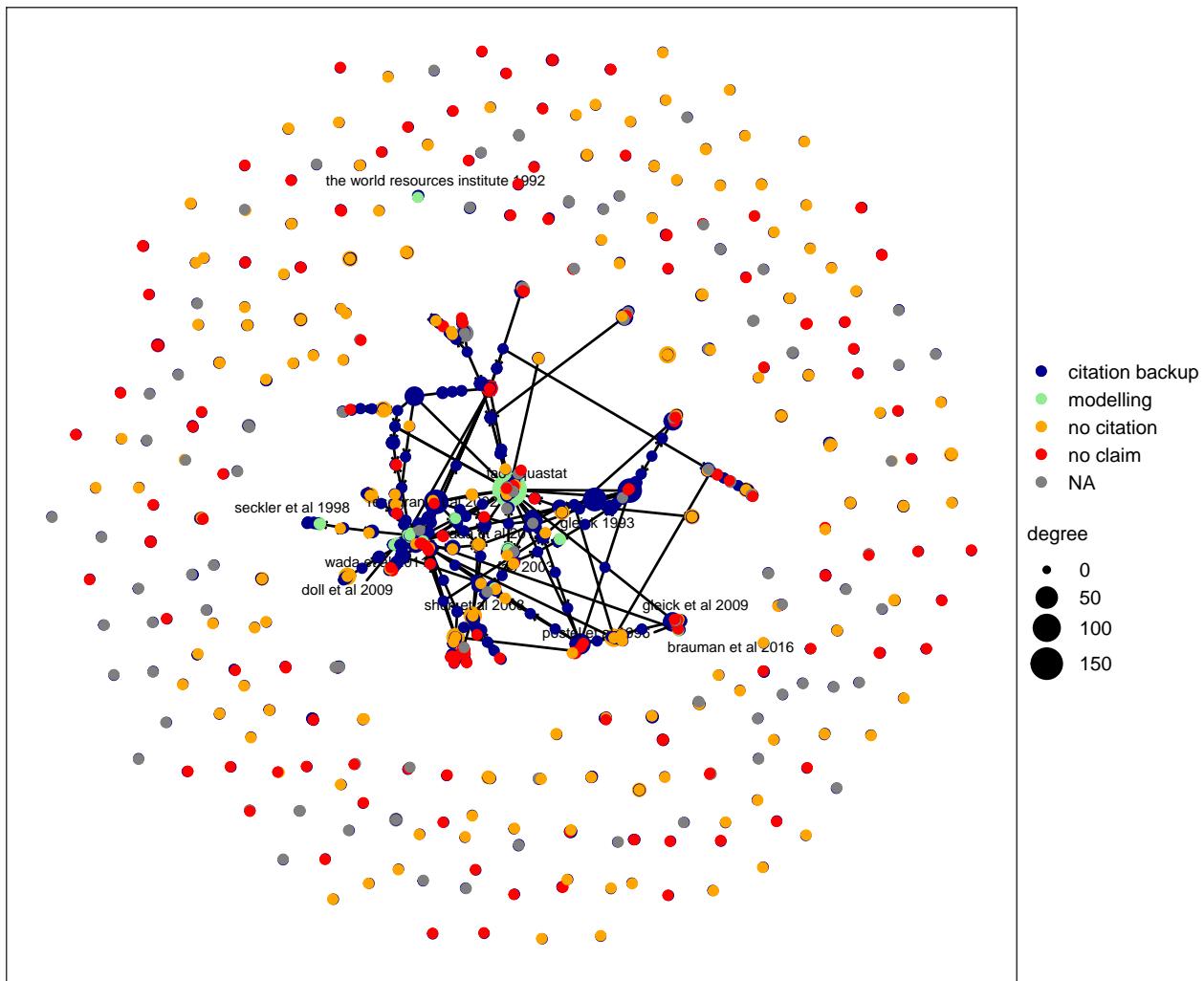
```
##  
## $water[[2]]  
  
## Warning: Removed 237 rows containing missing values or values outside the scale range  
## (`geom_text_repel()`).
```

2010



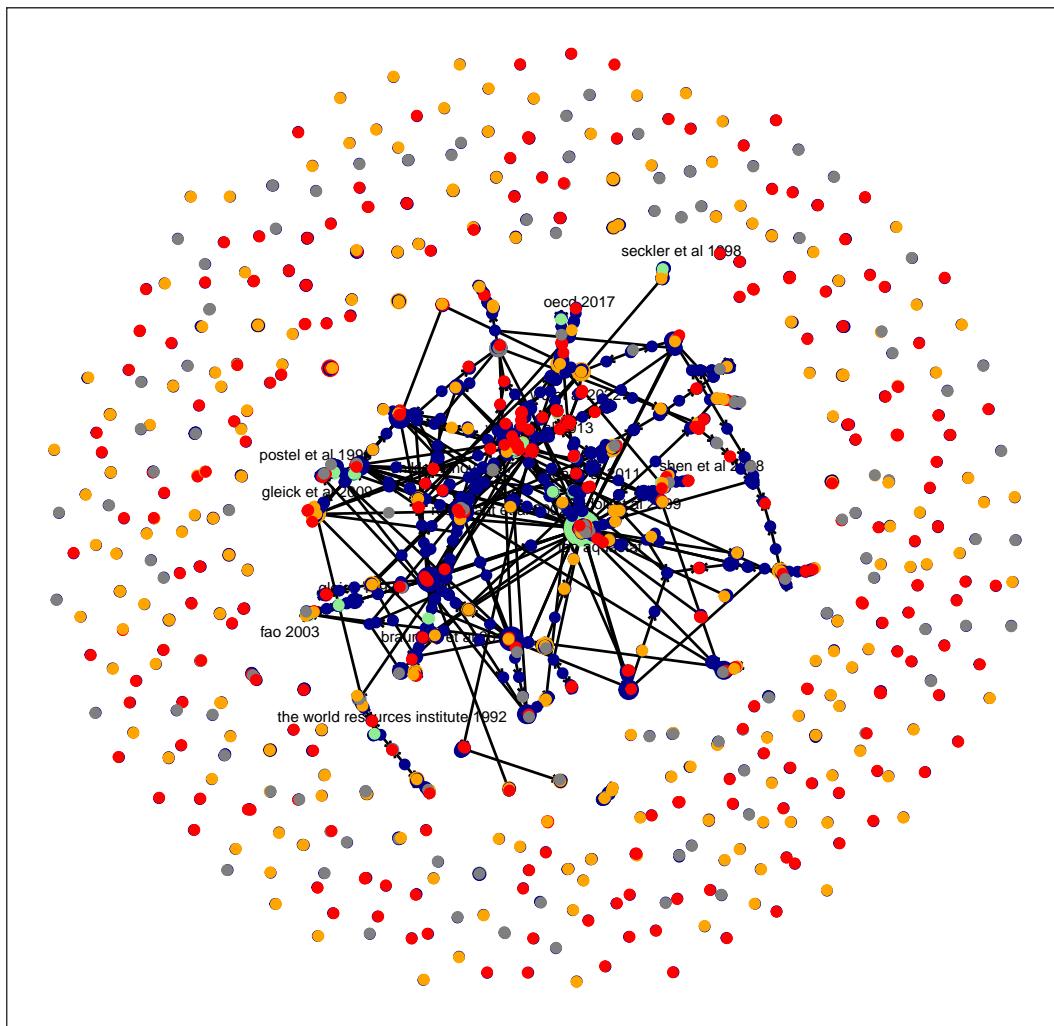
```
##  
## $water[[3]]  
  
## Warning: Removed 1476 rows containing missing values or values outside the scale range  
## (`geom_text_repel()`).  
  
## Warning: ggrepel: 1 unlabeled data points (too many overlaps). Consider  
## increasing max.overlaps
```

2020



```
##  
## $water[[4]]  
  
## Warning: Removed 2780 rows containing missing values or values outside the scale range  
## (`geom_text_repel()`).
```

2024



```
# PLOT #####
# Extract legend -----
legend.plot <- list()
for (i in names(plots.through.time)) {
  legend.plot[[i]] <- get_legend(plots.through.time[[i]][[length(plots.through.time[[i]])]] +
    theme(legend.position = "top"))
}
## Warning: Removed 724 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).
## Warning in get_plot_component(plot, "guide-box"): Multiple components found;
## returning the first one. To return all, use `return_all = TRUE` .
## Warning: Removed 2780 rows containing missing values or values outside the scale range
```

```

## (`geom_text_repel()`).

## Warning in get_plot_component(plot, "guide-box"): Multiple components found;
## returning the first one. To return all, use `return_all = TRUE`.

# Plot ----

bottom <- out.plot <- list()

for (i in names(plots.through.time)) {

  bottom[[i]] <- do.call(plot_grid, c(plots.through.time[[i]],
                                      nrow = floor(length(years.vector) / 2)))
  out.plot[[i]] <- plot_grid(legend.plot[[i]],
                            bottom[[i]], ncol = 1, rel_heights = c(0.1, 0.9))

}

## Warning: Removed 5 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

## Warning: Removed 123 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

## Warning: Removed 449 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

## Warning: Removed 724 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

## Warning: Removed 30 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

## Warning: Removed 237 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

## Warning: Removed 1476 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

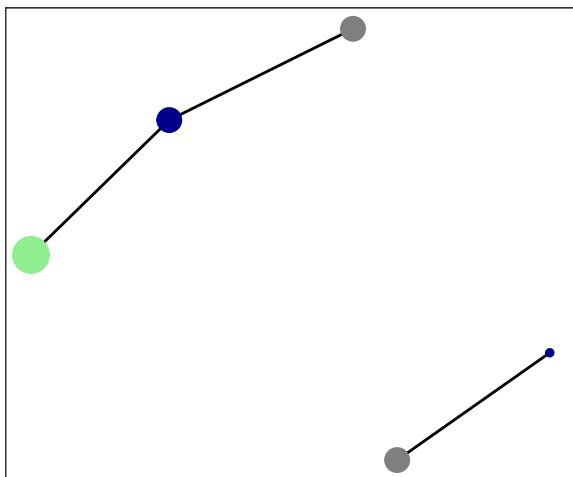
## Warning: Removed 2780 rows containing missing values or values outside the scale range
## (`geom_text_repel()`).

out.plot

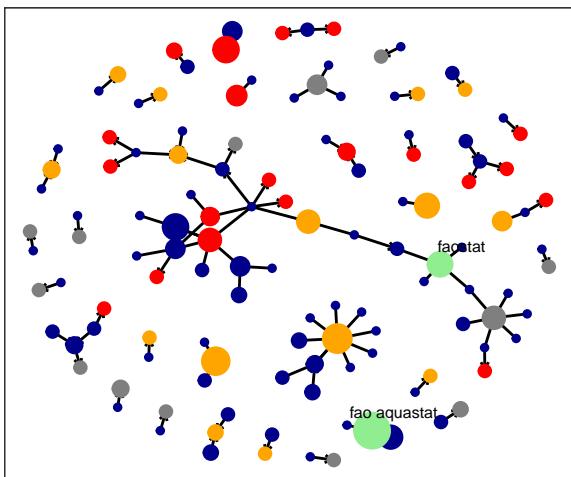
## $food

```

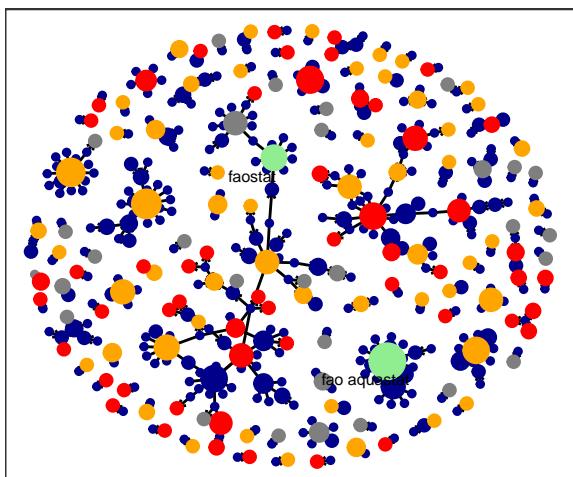
2000



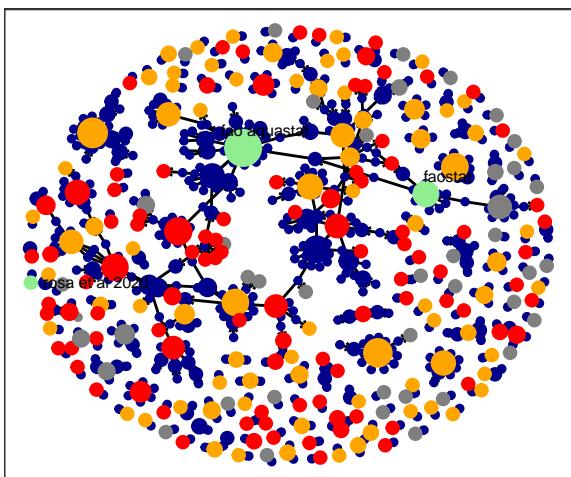
2010



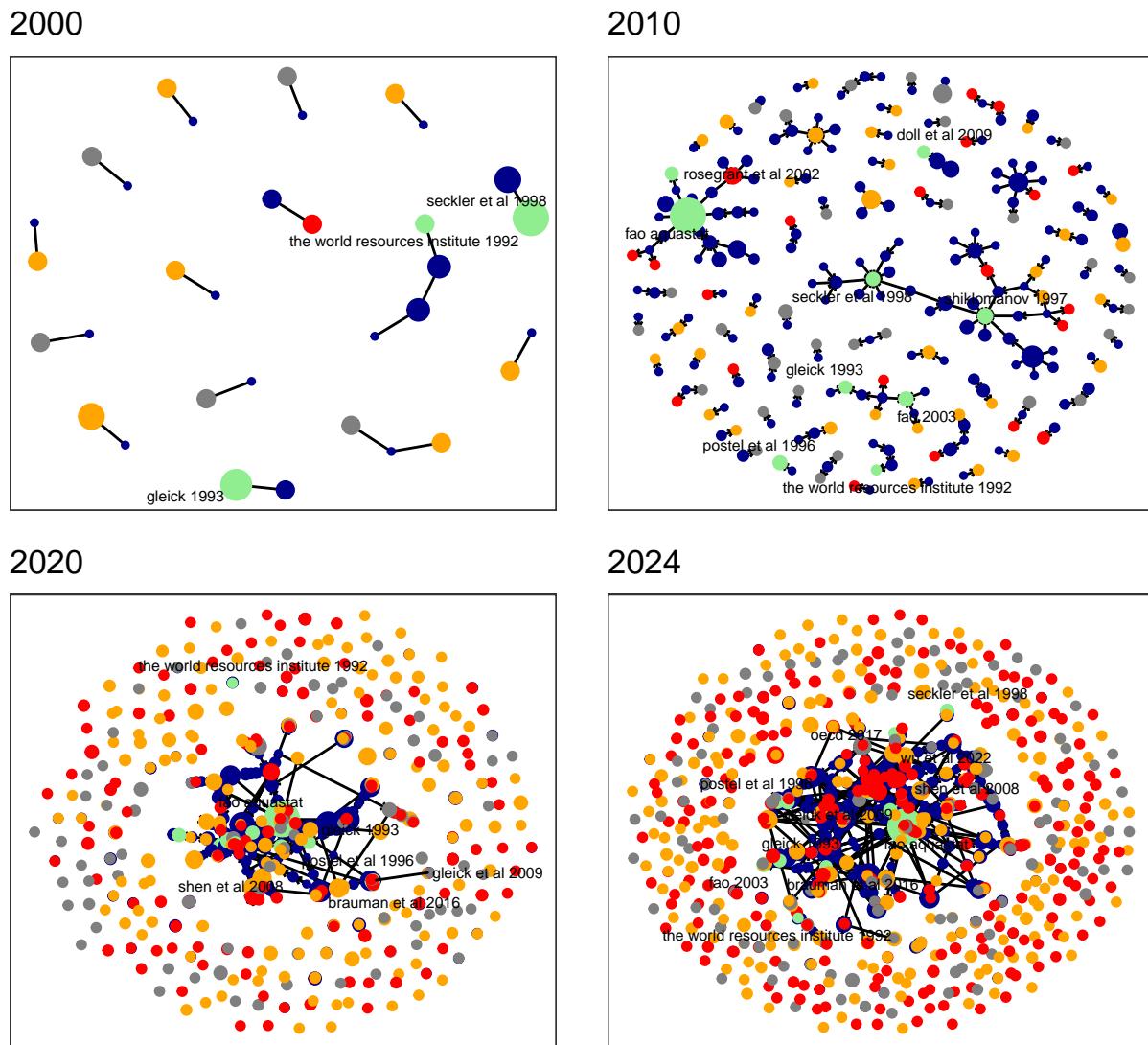
2020



2024



```
##  
## $water  
  
## Warning: ggrepel: 7 unlabeled data points (too many overlaps). Consider  
## increasing max.overlaps  
  
## Warning: ggrepel: 5 unlabeled data points (too many overlaps). Consider  
## increasing max.overlaps
```



3 Analysis of paths

3.1 “no claim” or “no citation” paths

```

# COUNT THE NUMBER OF NODES WITH PATHS ULTIMATELY LEADING TO NODES
# THAT DO NOT MAKE THE CITATION #####
# Function: loop through each node that do not make the claim to find all nodes
# connected to it -----
nodes_to_no_claim_node_fun <- function(g, terminal_nodes) {

  if (!is.igraph(g)) {
    g <- as.igraph(g)
  }
}

```

```

all_predecessors <- vector("list", length(terminal_nodes))

for (i in seq_along(terminal_nodes)) {

  terminal_node <- terminal_nodes[i]
  predecessors <- subcomponent(g, terminal_node, mode = "in")
  all_predecessors[[i]] <- predecessors
}

unique_predecessors <- unique(names(unlist(all_predecessors)))

return(unique_predecessors)
}

# CALCULATE

# Extract name of all nodes -----
all_nodes <- lapply(graph.final, function(graph)
  graph %>%
    activate(nodes) %>%
    pull(name))

# Extract name of nodes that do not make the claim -----

no.claim_nodes <- lapply(graph.final, function(graph)
  graph %>%
    activate(nodes) %>%
    filter(degree.out == 0 & nature.claim == "no claim") %>%
    pull(., "name"))

# Extract name of nodes that do not make the claim and those that make
# the claim but do not cite anybody -----

no.claim.and.no.citation.nodes <- lapply(graph.final, function(graph)
  graph %>%
    activate(nodes) %>%
    filter(degree.out == 0 & nature.claim == "no claim" | nature.claim == "no citation" ) %>%
    pull(., "name"))

# Run the function -----

tmp <- list()

for(i in names(graph.final)) {

  tmp[[i]] <- lapply(list(no.claim_nodes[[i]],

```

```

        no.claim.and.no.citation.nodes[[i]]), function(x)
    sort(nodes_to_no_claim_node_fun(graph.final[[i]], terminal_nodes = x)))
}

## Warning: `is.igraph()` was deprecated in igraph 2.0.0.
## i Please use `is_igraph()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

for(i in names(graph.final)) {
  names(tmp[[i]]) <- c("path ending in no claim",
                      "path ending in no claim or no citation")
}

# Calculate proportions -----
out <- list()

for(i in names(tmp)) {
  out[[i]] <- lapply(tmp[[i]], function(x) length(x) / length(all_nodes[[i]]))
}

out

## $food
## $food$path ending in no claim
## [1] 0.3947712
##
## $food$path ending in no claim or no citation
## [1] 0.7856209
##
## $water
## $water$path ending in no claim
## [1] 0.3582906
##
## $water$path ending in no claim or no citation
## [1] 0.6789744

```

3.2 Calculation of amplification

```

# CALCULATE AMPLIFICATION FUNCTION #####
# Prepare data -----
vec.topics <- c("food", "water")

```

```

graph <- modelling.papers <- list()

for (i in 1:length(vec.topics)) {

  graph[[i]] <- network.dt.complete[topic == vec.topics[[i]]] %>%
    graph_from_data_frame()

  modelling.papers[[i]] <- network.dt.complete[topic == vec.topics[[i]]] &
    nature.claim == "modelling"] %>%
  .[, from] %>%
  unique()
}

## Warning in graph_from_data_frame(.): In `d` `NA` elements were replaced with
## string "NA"

## Warning in graph_from_data_frame(.): In `d` `NA` elements were replaced with
## string "NA"

names(graph) <- vec.topics
names(modelling.papers) <- vec.topics

# DEFINE AMPLIFICATION FUNCTIONS #####
# Citation amplification function -----
citation_amplification_index <- function(graph, source_paper, modelling_papers) {

  # Get all simple paths from paper P -----
  all_paths <- all_simple_paths(graph, from = source_paper, mode = "out")

  # Filter out paths of length 1 that lead to modelling papers -----
  filtered_paths <- Filter(function(path) {

    if (length(path) > 2) {

      return(TRUE)

    } else {

      # If length is 2, check if terminal node is modelling paper -----

      return(!tail(path, n = 1) %in% modelling_papers)
    }
  }, all_paths)
}

```

```

# Calculate the number of paths -----
citation_amplification_index <- length(filtered_paths)

return(citation_amplification_index)
}

# Function to apply previous function to all papers -----
calculate_all_cai <- function(graph, modelling_papers) {

  papers <- V(graph)$name

  # Initialize an empty data.table to store results ----

  result_dt <- data.table(
    paper = character(),
    cai = numeric()
  )

  # Iterate over each paper ----

  for (paper in papers) {

    cai <- citation_amplification_index(graph, paper, modelling_papers)
    result_dt <- rbind(result_dt, data.table(paper = paper, cai = cai))
  }

  return(result_dt)
}

# CALCULATE AMPLIFICATION INDEX #####
# Calculate average amplification index of the networks -----
# (e.g., the number paths initiated by the average paper
# leading to studies that do not flow directly to "primary" data)

out <- list()

for(i in names(modelling.papers)) {

  out[[i]] <- calculate_all_cai(graph[[i]], modelling.papers[[i]])
}

# ARRANGE DATA #####
tmp <- rbindlist(out, idcol = "topic")

```

```

# SUMMARY STATISTICS #####
tmp[, mean(cai), topic]

##      topic      V1
##    <char>    <num>
## 1: food 1.088889
## 2: water 1.821880

# First 5 papers amplifying the most -----

tmp2 <- tmp[, .SD, topic] %>%
  .[order(-cai, topic)] %>%
  .[, head(.SD, 5), topic]

tmp2

##      topic          paper   cai
##    <char>        <char> <num>
## 1: water       wada 2015   39
## 2: water neysiani et al 2022   27
## 3: water       habtu 2024   23
## 4: water      abiyoje et al 2023   22
## 5: water      derossi et al 2024   22
## 6: food       taguta et al 2022   14
## 7: food       guo et al 2023a   12
## 8: food      kadigi et al 2004    8
## 9: food      xie and lark 2021    7
## 10: food      mao et al 2024    6

# PLOT DISTRIBUTION OF AMPLIFICATION INDICES #####

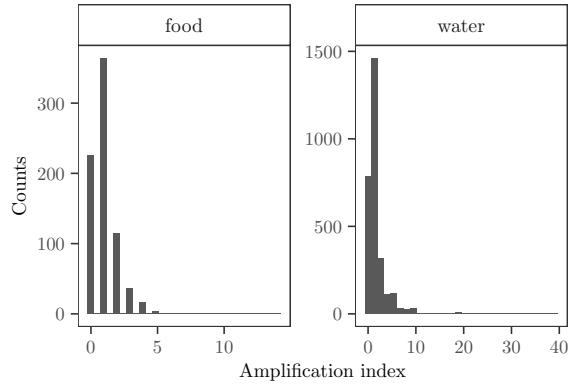
plot.amplification <- list()

# PLOT #####

plot.amplification <- ggplot(tmp, aes(cai)) +
  geom_histogram() +
  facet_wrap(~topic, scales = "free") +
  theme_AP() +
  labs(y = "Counts", x = "Amplification index")

plot.amplification

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
# PLOT THE NETWORK OF TOP AMPLIFYING PAPERS #####
# Define the starting nodes -----
vec.names.amplification <- tmp2[, slice_max(.SD, cai, n = 1), topic] %>%
  .[, paper]

# Reorder so the top node for food comes first -----
vec.names.amplification <- vec.names.amplification[order(vec.names.amplification)]

out <- list()

for (i in 1:length(vec.names.amplification)) {

  start_node <- vec.names.amplification[[i]]

  # Get all simple paths from the starting node to all reachable nodes
  paths <- all_simple_paths(graph[[i]], from = V(graph[[i]])[name == start_node])

  # Convert paths to a list of edge pairs for easy visualization
  edge_list <- lapply(paths, function(path) {
    edges <- as_edgelist(induced_subgraph(graph[[i]], path), names = TRUE)
    data.frame(from = edges[, 1], to = edges[, 2])
  })

  # Combine all path edges into a single data frame
  path_edges <- do.call(rbind, edge_list)

  #Get the unique nodes involved in the paths
  path_nodes <- unique(c(path_edges$from, path_edges$to))

  # Create the subgraph of all paths starting from the target node
  subgraph <- induced_subgraph(graph[[i]], vids = V(graph[[i]])[name %in% path_nodes])

  # Convert the subgraph to a tidygraph object
}
```

```

subgraph_tbl <- as_tbl_graph(subgraph)

# Retrieve a vector with the node names -----
vec.names <- subgraph_tbl %>%
  activate(nodes) %>%
  pull() %>%
  data.table(name = .)

nature.claim.vec <- graph.final[[i]] %>%
  activate(nodes) %>%
  data.frame() %>%
  data.table() %>%
  .[name %in% vec.names$name] %>%
  .[, nature.claim]

subgraph_tbl <- subgraph_tbl %>%
  activate(nodes) %>%
  mutate(nature.claim = nature.claim.vec)

# Plot the subgraph with ggraph

if (i == 1) {

  selected_colors <- c("darkblue", "orange", "red")

} else {

  selected_colors <- c("darkblue", "lightgreen", "orange", "red")

}

set.seed(123)

out[[i]] <- ggraph(subgraph_tbl, layout = "igraph", algorithm = "nicely") +
  geom_edge_link(arrows = arrow(length = unit(1.8, 'mm')),
                 end_cap = circle(1, "mm")) +
  geom_node_point(size = 1.5, aes(color = nature.claim)) +
  geom_node_text(aes(label = name), repel = TRUE, size = 2.2) +
  scale_color_manual(name = "",
                     values = selected_colors) +
  theme_AP() +
  labs(x = "", y = "") +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),

```

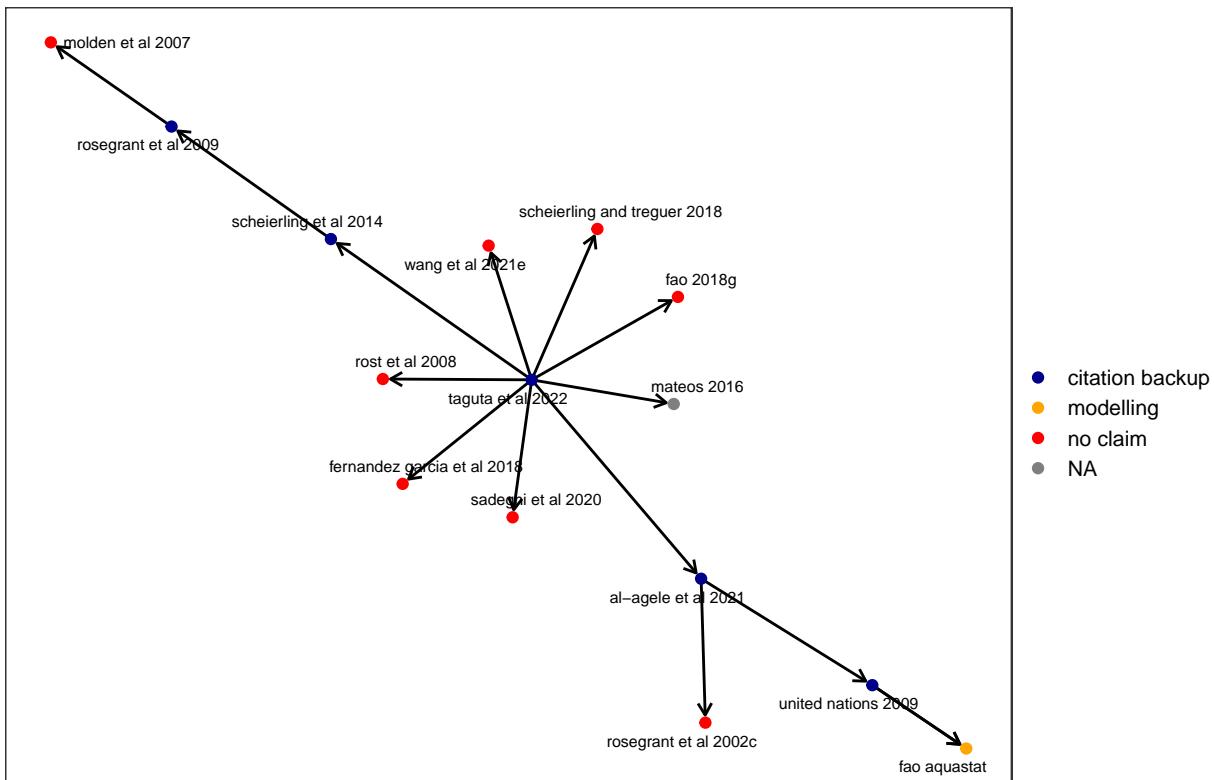
```

        legend.position = "right") +
      ggttitle(paste("Paths Activated by", start_node))
    }

out
```

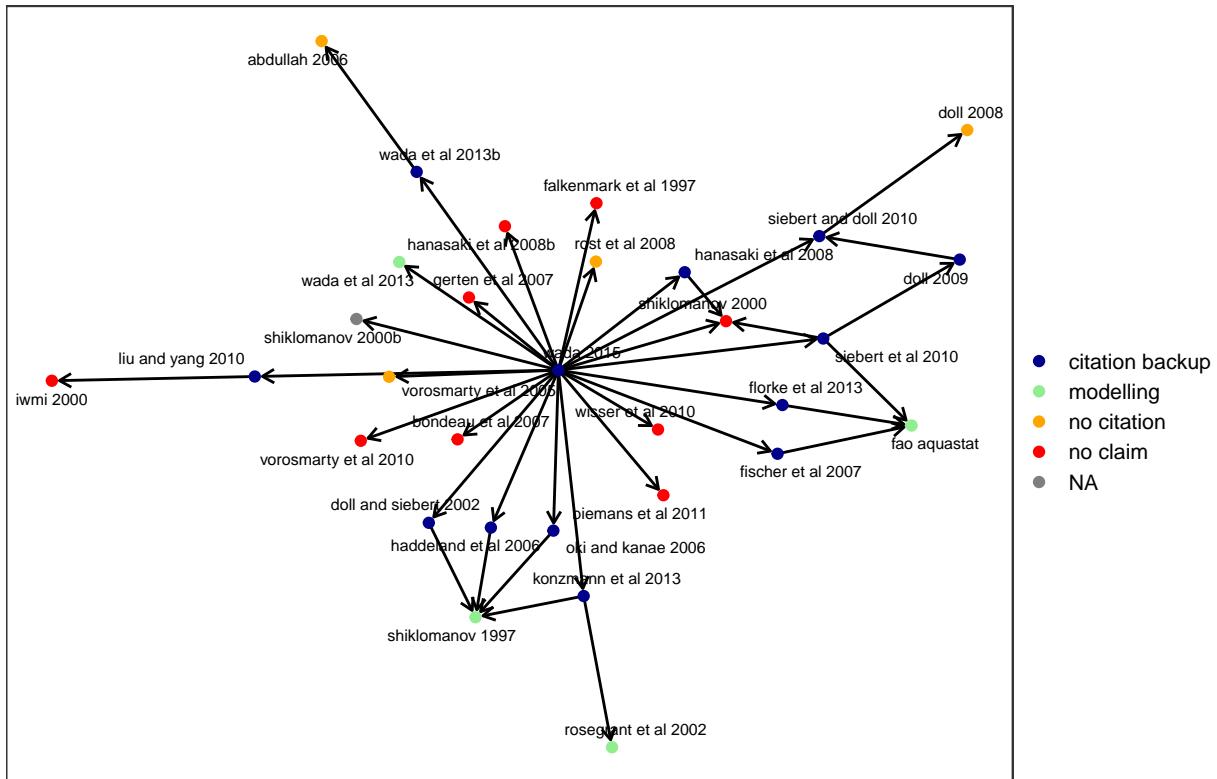
```
## [[1]]
```

Paths Activated by taguta et al 2022



```
##  
## [[2]]
```

Paths Activated by wada 2015



4 Both networks

4.1 Overlap between networks

```
# CHECK FULL NETWORK AND OVERLAP BETWEEN WATER AND FOOD NETWORK #####
# Prepare data -----
dd <- tidygraph::as_tbl_graph(network.dt.complete, directed = TRUE)

## Warning in graph_from_data_frame(x, directed = directed): In `d` `NA` elements
## were replaced with string "NA"
# Extract vector with names -----
all.names <- dd %>%
  activate(nodes) %>%
  pull(name)

# Retrieve names from water and food belief system -----

names.food <- network.dt.complete[topic == "food", to]
names.water <- network.dt.complete[topic == "water", to]
```

```

# Define intersections and differences -----
names.only.food <- setdiff(names.food, names.water)
names.only.water <- setdiff(names.water, names.food)
names.both <- intersect(names.water, names.food)

# New column defining whether nodes are in water, food or in both networks -----
final.graph <- dd %>%
  activate(nodes) %>%
  mutate(topic.final = ifelse(name %in% names.only.food, "food",
                               ifelse(name %in% names.only.water, "water",
                                      ifelse(name %in% names.both, "both", "uncited"))),
         topic.final = factor(topic.final, levels = c("food", "water", "both", "uncited")))

final.graph <- final.graph %>%
  activate(edges) %>%
  mutate(edge_color = .N()$topic.final[to])

# SOME STATS #####
dt.nodes <- final.graph %>%
  activate(nodes) %>%
  data.frame() %>%
  data.table()

# Fraction of network overlap -----
dt.nodes[, .N, topic.final] %>%
  .[, fraction:= N / nrow(dt.nodes)] %>%
  print

##   topic.final     N   fraction
##             <fctr> <int>     <num>
## 1:      uncited  2116  0.61103090
## 2:      water   1042  0.30089518
## 3:      both    125  0.03609587
## 4:      food    180  0.05197805

# PLOT MERGED NETWORK #####
selected.colors <- c("brown", "blue", "yellow", "grey")

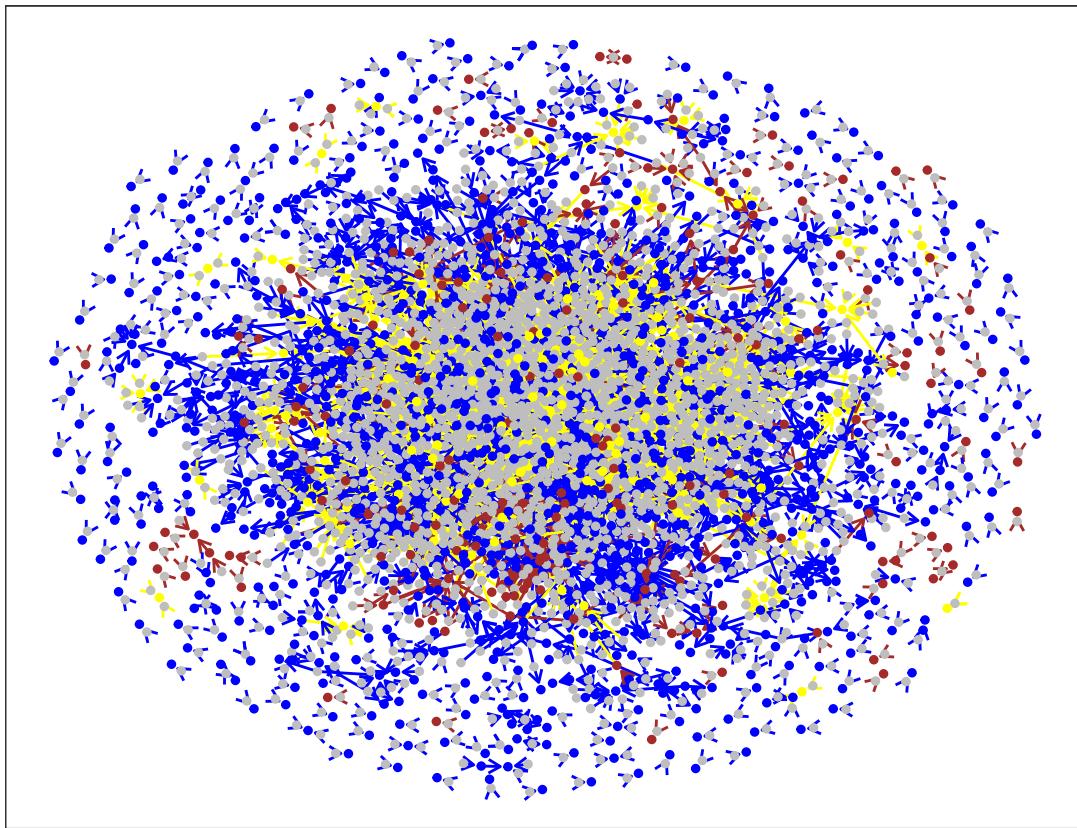
ggraph(final.graph, layout = "graphopt") +
  geom_edge_link(arrows = arrow(length = unit(1.8, 'mm')),
                 end_cap = circle(1, "mm"),
                 aes(color = edge_color)) +
  geom_node_point(aes(color = topic.final), size = 1) +

```

```

scale_edge_color_manual(values = selected.colors, guide = "none") +
scale_color_manual(name = "",
                   values = selected.colors) +
labs(x = "", y = "") +
theme_AP() +
theme(axis.text.x = element_blank(),
      axis.ticks.x = element_blank(),
      axis.text.y = element_blank(),
      axis.ticks.y = element_blank(),
      legend.position = "right")

```



4.2 Shared network

```

# PLOT ONLY THE NETWORK OF NODES BEING CITED FOR BOTH BELIEFS #####
# Prepare data -----
intersect.network <- network.dt.complete[to %in% names.both] %>%
  tidygraph::as_tbl_graph(., directed = TRUE)

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

```

```

intersect.graph <- network.dt.complete[to %in% names.both] %>%
  graph_from_data_frame(d = ., directed = TRUE)

## Warning in graph_from_data_frame(d = ., directed = TRUE): In `d` `NA` elements
## were replaced with string "NA"

# Calculate metrics ----

intersect.metrics <- data.table(node = V(intersect.graph)$name,
                                    degree = degree(intersect.graph, mode = "in"),
                                    degree.out = degree(intersect.graph, mode = "out"),
                                    betweenness = betweenness(intersect.graph),
                                    closeness = closeness(intersect.graph),
                                    pagerank = page_rank(intersect.graph)$vector)

degree.nodes <- intersect.metrics[order(-degree)][1:3]
degree.out.nodes <- intersect.metrics[order(-degree.out)][1:3]
betweenness.nodes <- intersect.metrics[order(-betweenness)][1:3]

# Retrieve a vector with the node names ----

vec.names <- intersect.network %>%
  activate(nodes) %>%
  pull() %>%
  data.table(name = .)

order <- match(vec.names$name, intersect.metrics$node)
tmp <- intersect.metrics[order]

intersect.graph.final <- intersect.network %>%
  activate(nodes) %>%
  mutate(degree = tmp$degree,
         degree.out = tmp$degree.out,
         betweenness = tmp$betweenness)

intersect.graph.final <- intersect.graph.final %>%
  activate(nodes) %>%
  mutate(topic = ifelse(name %in% names.both, "both beliefs", "citing"))

# PLOT #####
set.seed(12)

ggraph(intersect.graph.final, layout = "graphopt") +
  geom_edge_link(arrows = arrow(length = unit(1.8, 'mm')),
                 end_cap = circle(1, "mm"),
                 aes(color = topic),
                 alpha = 0.3) +

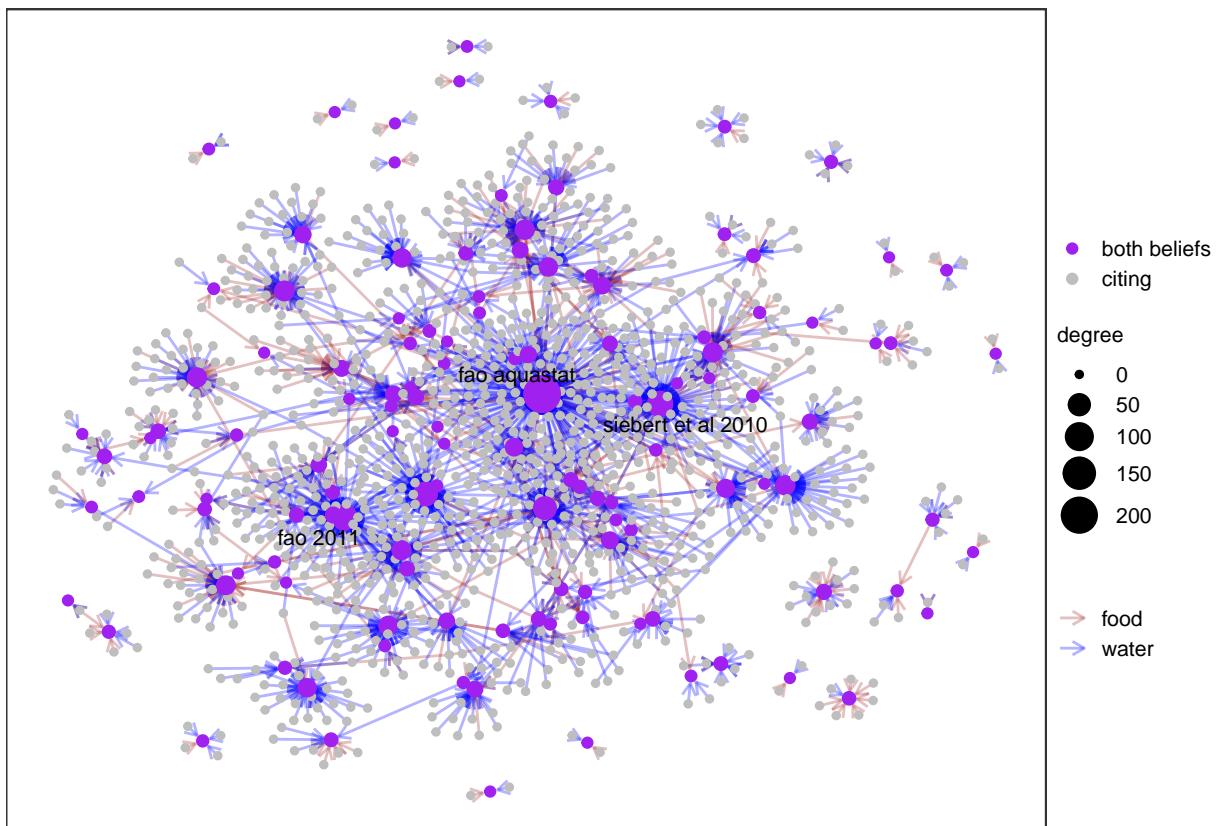
```

```

geom_node_point(aes(color = topic, size = degree)) +
  geom_node_text(aes(label = ifelse(degree >= min(degree.nodes$degree),
                                 name, NA)),
                repel = TRUE,
                size = 2.7) +
  scale_edge_color_manual(values = c("brown", "blue"),
                          name = "") +
  scale_color_manual(name = "",
                     values = c("purple", "grey")) +
  labs(x = "", y = "") +
  theme_AP() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        legend.position = "right")

```

Warning: Removed 1305 rows containing missing values or values outside the scale range
(`geom_text_repel()`).



```

set.seed(12)

ggraph(intersect.graph.final, layout = "graphopt") +
  geom_edge_link(arrows = arrow(length = unit(1.8, 'mm'))),

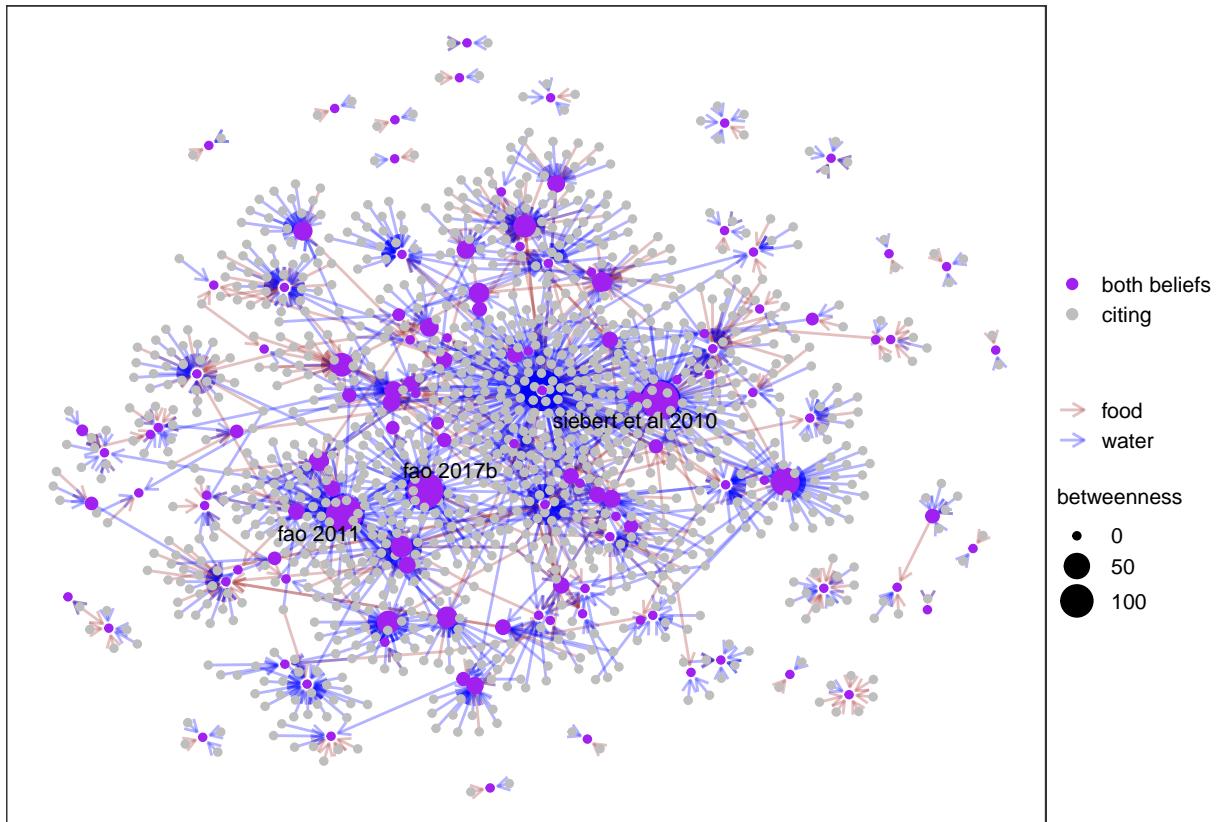
```

```

    end_cap = circle(1, "mm"),
    aes(color = topic),
    alpha = 0.3) +
  geom_node_point(aes(color = topic, size = betweenness)) +
  geom_node_text(aes(label = ifelse(betweenness >= min(betweenness.nodes$betweenness),
                                 name, NA)),
                 repel = TRUE,
                 size = 2.7) +
  scale_edge_color_manual(values = c("brown", "blue"),
                          name = "") +
  scale_color_manual(name = "",
                     values = c("purple", "grey")) +
  labs(x = "", y = "") +
  theme_AP() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        legend.position = "right")

```

Warning: Removed 1305 rows containing missing values or values outside the scale range
(`geom_text_repel()`).



5 Study of Aquastat values

```
# STUDY OF AQUASTAT PERCENTAGES #####
# Read in aquastat dataset ----

aquastat.dt <- read.xlsx("aquastat_dt.xlsx") %>%
  data.table() %>%
  .[Year == 2020] %>%
  setnames(., c("Value", "Area"), c("percentage", "country")) %>%
  .[, .(country, percentage)] %>%
  .[, data:= "aquastat 2020"] %>%
  .[, country:= countrycode(country, origin = "country.name", destination = "country.name")]

## Warning: Some values were not matched unambiguously: Australia and New Zealand
## Warning: Some strings were matched more than once, and therefore set to <NA> in the result:
aquastat.dt[, continent:= countrycode(country, origin = "country.name", destination = "continent")]

# Read in world resources institute dataset ----

wri <- fread("world_resources_institut_guide_to_the_global_environment_1994.csv") %>%
  .[order(country)] %>%
  .[, data:= "wri 1994"] %>%
  .[, country:= countrycode(country, origin = "country.name", destination = "country.name")]

## Warning: Some values were not matched unambiguously: , Cote d'lvoire
wri[, continent:= countrycode(country, origin = "country.name", destination = "continent")]

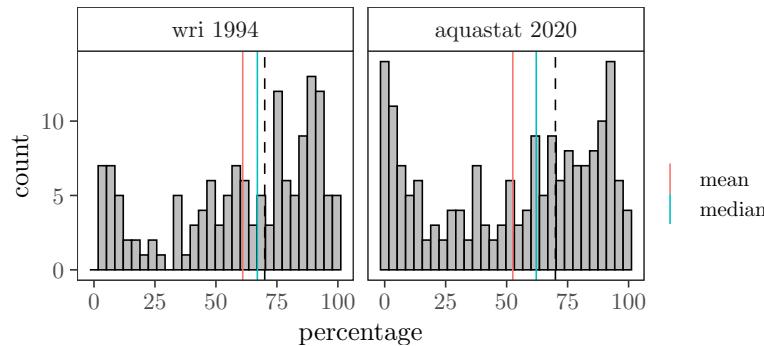
## Warning: Some values were not matched unambiguously: Czechoslovakia, Yugoslavia
# Compare distributions ----

dt.comparison <- rbind(aquastat.dt, wri) %>%
  .[, data:= factor(data, levels = c("wri 1994", "aquastat 2020"))]

dt.stats.comparison <- dt.comparison[, .(mean = mean(percentage, na.rm = TRUE),
                                         median = median(percentage, na.rm = TRUE)), data] %>%
  melt(., measure.vars = c("mean", "median"))

ggplot(dt.comparison, aes(percentage)) +
  geom_histogram(color = "black", fill = "grey") +
  facet_wrap(~data) +
  geom_vline(data = dt.stats.comparison, aes(xintercept = value, color = variable)) +
  scale_color_discrete(name = "") +
  geom_vline(xintercept = 70, lty = 2) +
  theme_AP()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 1 row containing non-finite outside the scale range
## (`stat_bin()`).
```



```
# At the country level -----
```

```
tmp <- aquastat.dt[wri, on = c("country", "continent")] %>%
  .[, .(country, continent, percentage, i.percentage)] %>%
  setnames(., c("percentage", "i.percentage"), c("aquastat 2020", "wri 1994")) %>%
  melt(., measure.vars = c("aquastat 2020", "wri 1994")) %>%
  .[, country := ifelse(country == "Trinidad & Tobago", "Trinidad and Tobago", country)] %>%
  na.omit() %>%
  split(., .$continent)

## Warning in melt.data.table(., measure.vars = c("aquastat 2020", "wri 1994")):
## 'measure.vars' [aquastat 2020, wri 1994] are not all of the same type. By order
## of hierarchy, the molten data value column will be of type 'double'. All
## measure variables not of type 'double' will be coerced too. Check DETAILS in
## ?melt.data.table for more on coercion.

out <- list()

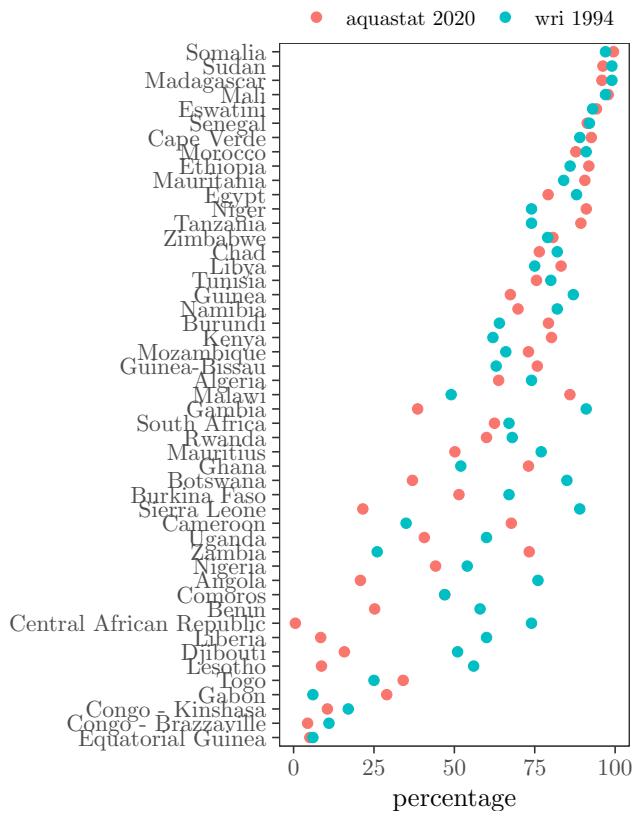
for(i in names(tmp)) {

  out[[i]] <- ggplot(tmp[[i]], aes(reorder(country, value),
                                    value, color = variable)) +
    coord_flip() +
    scale_color_discrete(name = "") +
    geom_point() +
    theme_AP() +
    theme(legend.position = "top") +
    labs(x = "", y = "percentage") +
    ggttitle(names(tmp[i]))
}

out

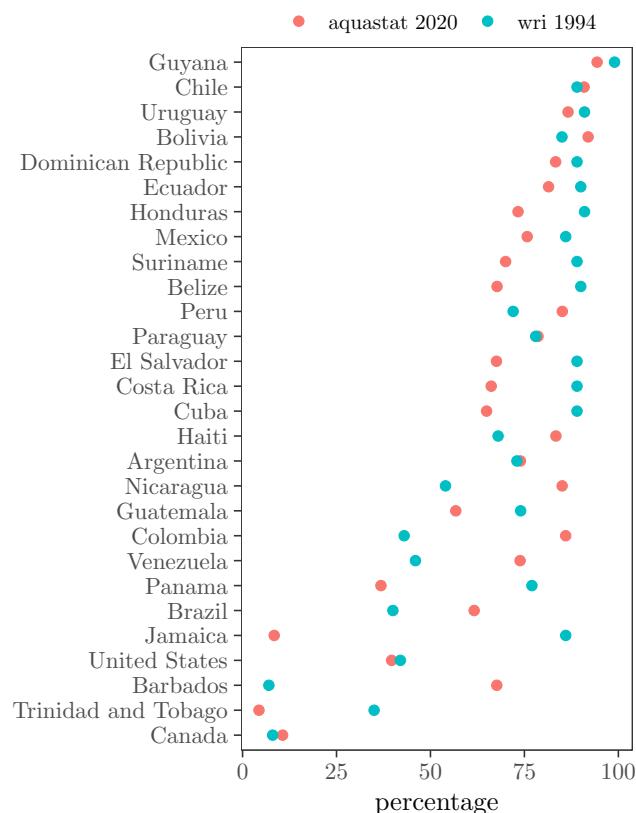
## $Africa
```

Africa



```
##  
## $Americas
```

Americas



```
##  
## $Asia
```

Asia



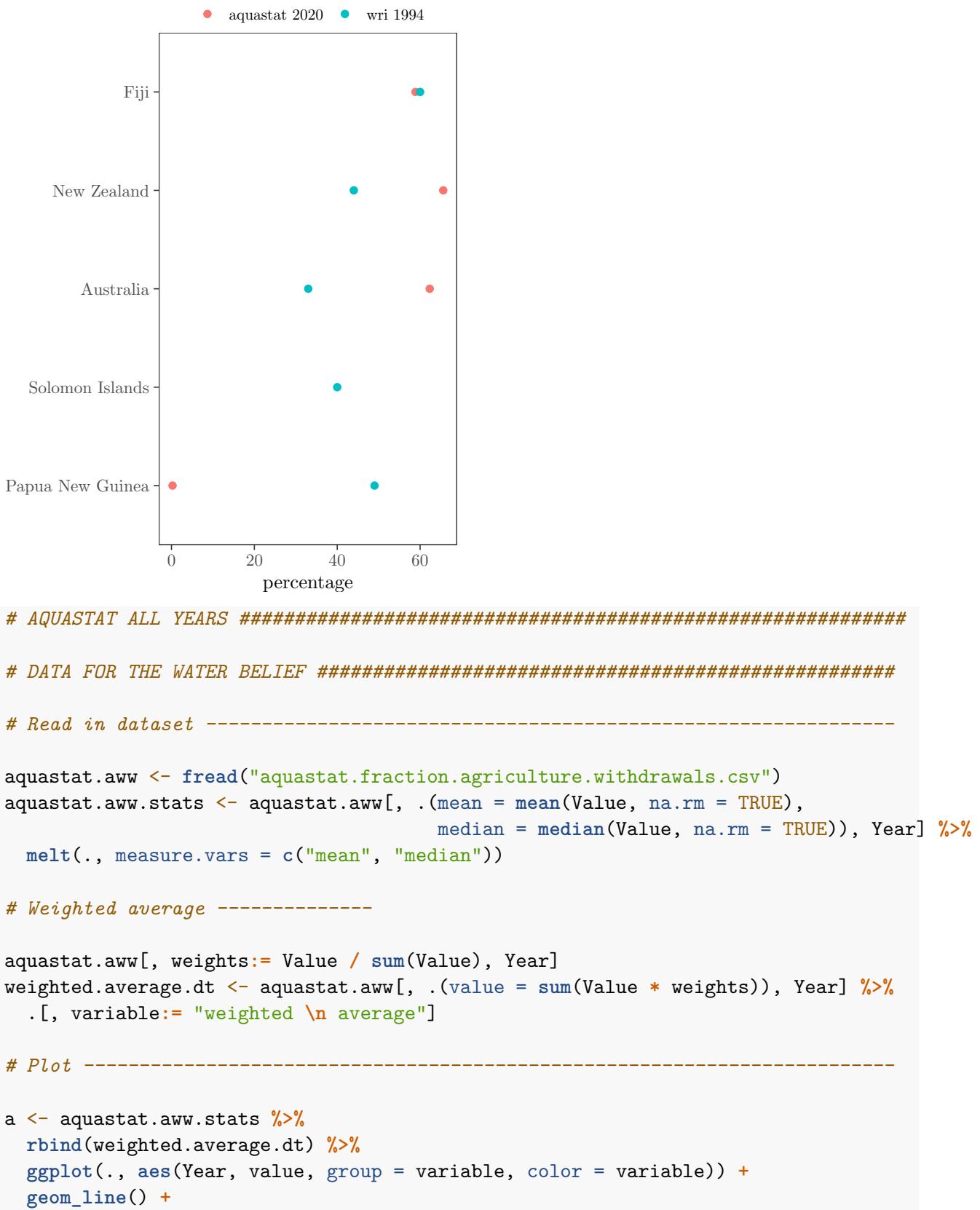
```
##  
## $Europe
```

Europe



```
##  
## $Oceania
```

Oceania

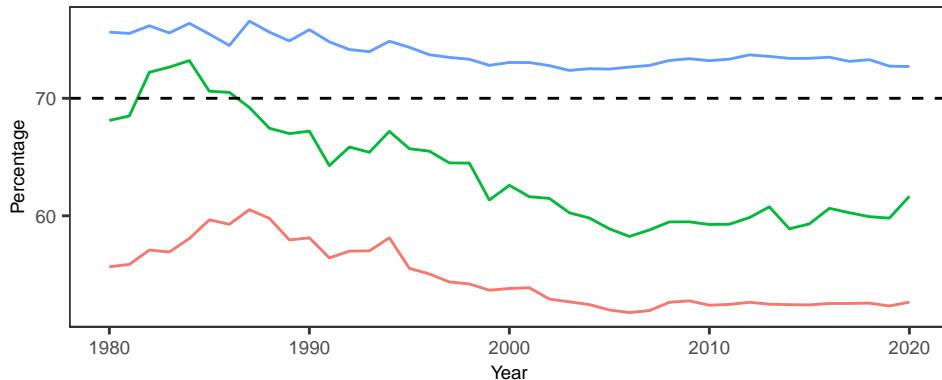


```

scale_color_discrete(name = "") +
geom_hline(yintercept = 70, lty = 2) +
theme_AP() +
labs(x = "Year", y = "Percentage") +
theme(legend.position = "none")

```

a

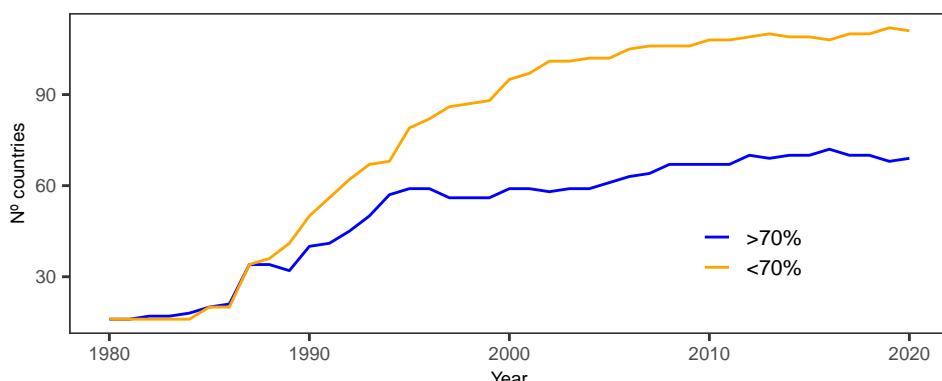


```

b <- aquastat.aww[, .(above.70 = sum(Value > 70),
                     below.70 = sum(Value < 70)), Year] %>%
  melt(., measure.vars = c("above.70", "below.70")) %>%
  ggplot(., aes(Year, value, color = variable)) +
  geom_line() +
  theme_AP() +
  scale_color_manual(name = "", labels = c(">70%", "<70%"),
                     values = c("blue", "orange")) +
  labs(x = "Year", y = "Nº countries") +
  theme(legend.position = c(0.78, 0.34))

```

b



Fraction of estimated and imputed values -----

```

n.countries <- aquastat.aww[, .(total.countries = .N), Year]
fraction.estimate <- aquastat.aww[, .N, .(Symbol, Year)] %>%

```

```

merge(., n.countries, by = "Year") %>%
  .[, fraction:= N / total.countries] %>%
  ggplot(., aes(Year, N, color = Symbol)) +
  geom_line() +
  labs(x = "Year", y = "Nº countries") +
  scale_color_discrete(name = "") +
  theme_AP() +
  theme(legend.position = c(0.85, 0.25),
        legend.text = element_text(size = 7))

dt.stats.year <- aquastat.aww.stats %>%
  rbind(weighted.average.dt) %>%
  .[Year == max(Year)]

water.histogram <- aquastat.aww[Year == max(Year)] %>%
  ggplot(., aes(Value)) +
  geom_histogram(color = "black", fill = "grey") +
  geom_vline(xintercept = 70, lty = 2) +
  geom_vline(data = dt.stats.year, aes(xintercept = value, color = variable)) +
  labs(x = "Percentage", y = "Nº countries") +
  theme_AP() +
  theme(legend.position = "none")

water.plots.aquastat <- plot_grid(a, water.histogram, b, fraction.estimate,
                                    ncol = 4, labels = c("b", ""))

```

`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

DATA FOR THE FOOD BELIEF

```

dt <- fread("aquastat.fraction.grain.irrigated.csv")

# Check the variable examined -----
unique(dt$Variable)

## [1] "% of total grain production irrigated"
# Calculate mean and median -----

```

```

aquastat.grain.stats <- dt[, .(mean = mean(Value),
                                median = median(Value),
                                N.countries = .N), Year] %>%
  melt(., measure.vars = c("mean", "median"))

# Calculated weighted average -----

```

```

dt[, weights:= Value / sum(Value), Year]
weighted.average.dt <- dt[, .(value = sum(Value * weights),

```

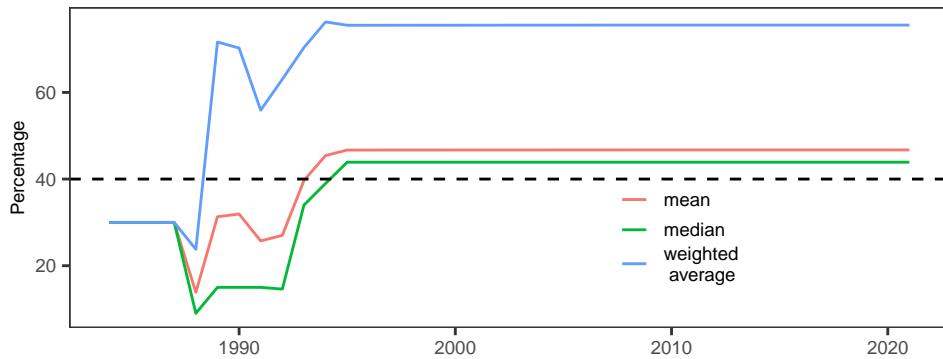
```

N.countries = .N), Year] %>%
.[, variable:= "weighted \n average"]

# Plot -----
a.crop <- aquastat.grain.stats %>%
  rbind(weighted.average.dt) %>%
  ggplot(., aes(Year, value, group = variable, color = variable)) +
  geom_line() +
  scale_color_discrete(name = "") +
  geom_hline(yintercept = 40, lty = 2) +
  theme_AP() +
  labs(x = "", y = "Percentage") +
  theme(legend.position = c(0.7, 0.38),
        legend.text = element_text(size = 7))

```

a.crop

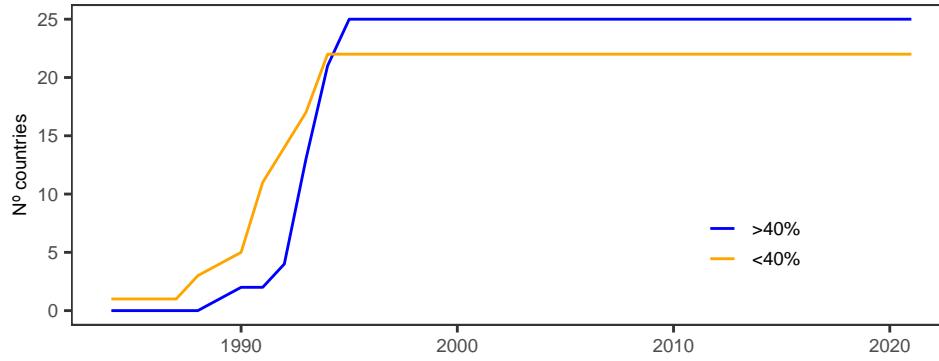


```

b.crop <- dt[, .(above.40 = sum(Value > 40),
               below.40 = sum(Value < 40)), Year] %>%
  melt(., measure.vars = c("above.40", "below.40")) %>%
  ggplot(., aes(Year, value, color = variable)) +
  geom_line() +
  theme_AP() +
  scale_color_manual(name = "", labels = c(">40%", "<40%"),
                     values = c("blue", "orange")) +
  labs(x = "", y = "Nº countries") +
  theme(legend.position = c(0.78, 0.34),
        legend.text = element_text(size = 7))

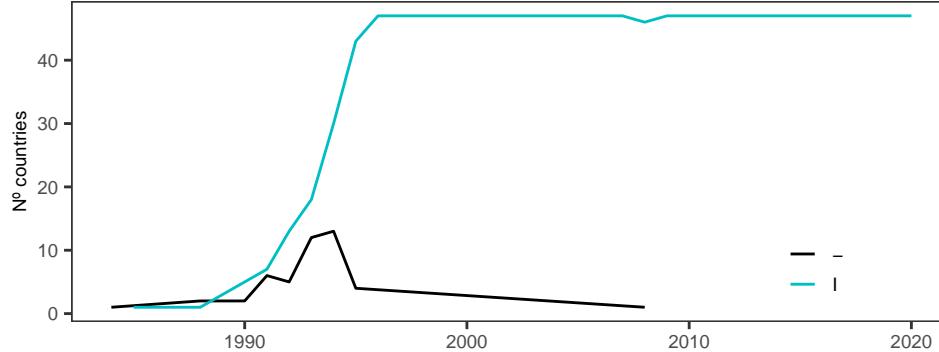
```

b.crop



```
n.countries.crop <- dt[, .(total.countries = .N), Year]
fraction.estimate <- dt[, .N, .(Symbol, Year)] %>%
  merge(., n.countries, by = "Year") %>%
  .[, fraction:= N / total.countries] %>%
  ggplot(., aes(Year, N, color = Symbol)) +
  geom_line() +
  labs(x = "", y = "Nº countries") +
  scale_color_manual(name = "", values = c("black", "#00BFC4")) +
  theme_AP() +
  theme(legend.position = c(0.85, 0.25),
        legend.text = element_text(size = 7))
```

fraction.estimate

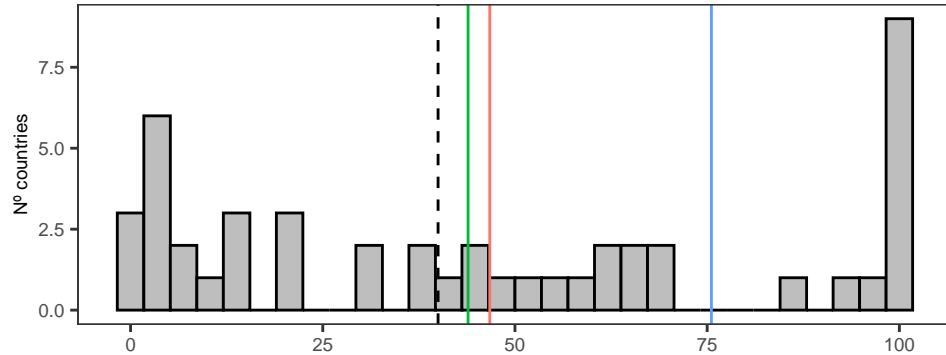


```
dt.stats.year.crops <- aquastat.grain.stats %>%
  rbind(weighted.average.dt) %>%
  .[Year == max(Year)]

crop.histogram <- dt[Year == max(Year)] %>%
  ggplot(., aes(Value)) +
  geom_histogram(color = "black", fill = "grey") +
  geom_vline(xintercept = 40, lty = 2) +
  geom_vline(data = dt.stats.year.crops, aes(xintercept = value, color = variable)) +
  labs(x = "", y = "Nº countries") +
  theme_AP() +
  theme(legend.position = "none")
```

```
crop.histogram
```

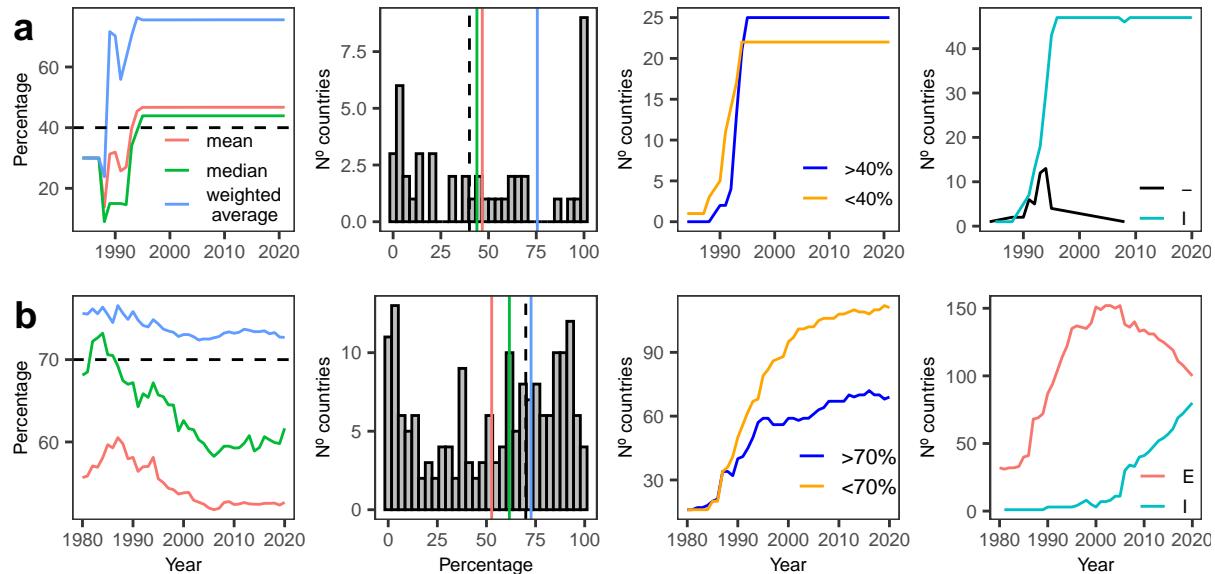
```
## `stat_bin()` using `bins = 30` . Pick better value with `binwidth` .
```



```
crop.plots.aquastat <- plot_grid(a.crop, crop.histogram, b.crop, fraction.estimate,  
                                labels = c("a", ""), ncol = 4)
```

```
## `stat_bin()` using `bins = 30` . Pick better value with `binwidth` .
```

```
plot_grid(crop.plots.aquastat, water.plots.aquastat, ncol = 1)
```



6 Uncertainty analysis

```
# DEFINE FUNCTIONS #####
# Transform columns to qunif with min and max reflecting uncertainty ----

transform_columns_fun <- function(mat, distr) {

  for (i in 1:nrow(distr)) {

    country <- distr$Country[i]

    min_val <- distr$min[i]
    max_val <- distr$max[i]

    mat[[country]] <- qunif(mat[[country]], min = min_val, max = max_val)
  }

  return(mat)
}

## Function to transform longitude and latitude to country.
# It is borrowed from Andy:
# https://stackoverflow.com/questions/14334970/convert-latitude-and-longitude-coordinates-to-c

coords2country = function(points) {

  countriesSP <- rworldmap::getMap(resolution = 'low')
  pointsSP = sp::SpatialPoints(points, proj4string=CRS(proj4string(countriesSP)))
  indices = sp::over(pointsSP, countriesSP)
  indices$ADMIN
  #indices$ISO3 # returns the ISO3 code
  #indices$continent # returns the continent (6 continent model)
  #indices$REGION # returns the continent (7 continent model)

}

# Function to retrieve .nc data from Huang et al 2018 ----

get_huang_fun <- function(nc_file, variable, year) {

  nc_data <- nc_open(nc_file)

  # Extract longitude, latitude, and water withdrawal data (withd_elec)
  lon <- ncvar_get(nc_data, "lon")
  lat <- ncvar_get(nc_data, "lat")
  withd_elec <- ncvar_get(nc_data, variable)
```

```

time <- ncvar_get(nc_data, "month")

# Close the NetCDF file after reading
nc_close(nc_data)

# Earth's radius in meters
earth_radius <- units::set_units(6371000, "m")

# Function to convert degrees to radians
deg_to_rad <- function(deg) {
  return(deg * pi / 180)
}

# Calculate the grid area (m2) for each grid cell
# Create a data.table with lon and lat values
dt <- data.table(lon = lon, lat = lat)

# Convert latitudes to radians for area calculation
dt[, lat_rad := deg_to_rad(lat)]

# Assume the grid cells have equal spacing (this should match your grid resolution)
# If you have grid cell dimensions, use them here. Otherwise, use a 0.5 degree grid as an ex
delta_lon <- deg_to_rad(0.5) # Assuming 0.5-degree grid spacing in longitude
delta_lat <- deg_to_rad(0.5) # Assuming 0.5-degree grid spacing in latitude

# Calculate the area of each grid cell (in square meters)
dt[, area_cell := as.numeric(earth_radius^2 * delta_lon * delta_lat * cos(lat_rad))]

# Generate the corresponding year and month for each time step
start_year <- 1971
start_month <- 1
n_months <- length(time)

# Generate year and month for each time step
years <- rep(start_year:(start_year + (n_months %% 12)), each = 12, length.out = n_months)
months <- rep(1:12, length.out = n_months)

# Convert withdrawal from mm/month to km3/month
for (month_idx in 1:n_months) {
  dt[[paste0("year_", years[month_idx], "_month_", months[month_idx])]] <- as.numeric((withdrawal[month_idx]))
}

# Select only columns that contain 'year_2010'
columns_selected <- grep(year, colnames(dt), value = TRUE)

# Create a new data.table with only the relevant columns for 2010
dt_2010 <- dt[, c("lon", "lat", columns_selected), with = FALSE]

```

```

countries <- coords2country(dt_2010)

dada <- cbind(countries, dt_2010) %>%
  melt(., measure.vars = columns_selected) %>%
  .[, sum(value, na.rm = TRUE), countries] %>%
  .[order(countries)] %>%
  .[, Country:= countrycode(countries, origin = "country.name", destination = "country.name")]

return(dada)
}

# Function to retrieve .nc4 files from ISIMIP -----
get_isimip_fun <- function(nc_file, variable, year, start_year) {

  nc_data <- nc_open(nc_file)

  # Extract longitude, latitude, and water withdrawal data
  lon <- ncvar_get(nc_data, "lon")
  lat <- ncvar_get(nc_data, "lat")
  aindww <- ncvar_get(nc_data, variable)
  time <- ncvar_get(nc_data, "time")

  # Close the NetCDF file after reading
  nc_close(nc_data)

  # Create a grid of lon and lat coordinates
  lon_lat_grid <- expand.grid(lon = lon, lat = lat)

  # Earth's radius in meters
  earth_radius <- units::set_units(6371000, "m")

  # Function to convert degrees to radians
  deg_to_rad <- function(deg) {
    return(deg * pi / 180)
  }

  # Calculate the grid area (m2) for each grid cell
  lon_lat_grid$lat_rad <- deg_to_rad(lon_lat_grid$lat)

  # Assume the grid cells have equal spacing (match your grid resolution)
  delta_lon <- deg_to_rad(0.5) # Assuming 0.5-degree grid spacing in longitude
  delta_lat <- deg_to_rad(0.5) # Assuming 0.5-degree grid spacing in latitude

  # Calculate the area of each grid cell (in square meters)
}

```

```

lon_lat_grid$area_cell <- as.numeric(earth_radius^2 * delta_lon * delta_lat * cos(lon_lat_gr

# Convert aindww from kg/m2/s to m3/month
# 1 kg of water = 1 liter = 1e-3 m3
# Multiply by the number of seconds in a month (30.44 days average)
seconds_per_month <- 30.44 * 24 * 3600 # Average month duration in seconds

# Generate the corresponding year and month for each time step
start_year <- start_year
n_months <- length(time)

years <- rep(start_year:(start_year + (n_months %% 12)), each = 12, length.out = n_months)
months <- rep(1:12, length.out = n_months)

# Convert withdrawal from kg/m2/s to km3/month for each grid cell
results <- list()
for (month_idx in 1:n_months) {
  aindww_month <- aindww[, , month_idx] # Extract the 2D slice (lon x lat) for the given month

  # Flatten the 2D data to match the grid
  aindww_month_flat <- as.numeric(indww_month)

  # Calculate water withdrawal in km3/month for each grid cell
  lon_lat_grid[[paste0("year_", years[month_idx], "_month_", months[month_idx])]] <-
    aindww_month_flat * 1e-3 * lon_lat_grid$area_cell * seconds_per_month / 1e9 # Convert to km3
}

# Select only columns that contain the selected year
columns_selected <- grep(year, colnames(lon_lat_grid), value = TRUE)

# Create a new data table with only the relevant columns for the specified year
dt_selected <- lon_lat_grid[, c("lon", "lat", columns_selected)] %>%
  na.omit()

countries <- coords2country(dt_selected)

result <- cbind(countries, dt_selected) %>%
  data.table() %>%
  melt(., measure.vars = columns_selected) %>%
  .[, sum(value, na.rm = TRUE), countries] %>%
  .[order(countries)] %>%
  .[, Country := countrycode(countries, origin = "country.name", destination = "country.name")]

return(result)
}

```

```

# DEFINE SETTINGS #####
N <- 2^20
type <- "QRN"
order <- "first"
matrices <- "A"

# UNCERTAINTY IN IRRIGATION #####
# Read dataset -----
gww <- fread("global_water_withdrawals.csv")
gww[, Country:= countrycode(Country, origin = "country.name", destination = "country.name")]

countries <- unique(gww$Country)

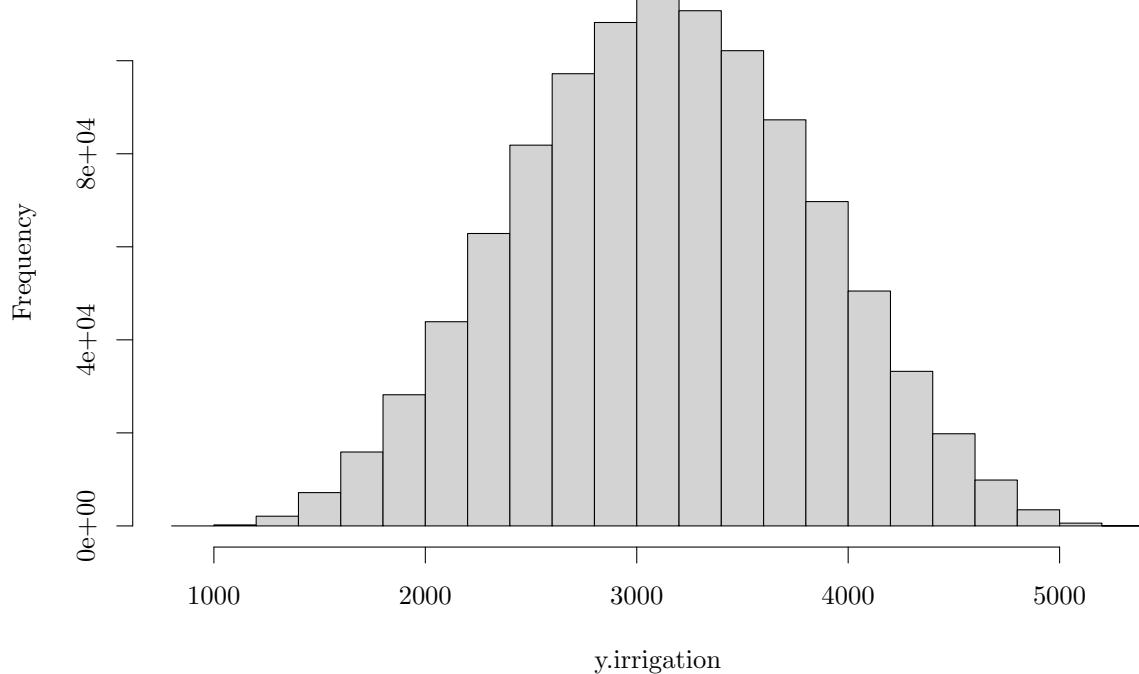
# Define sample matrix -----
params <- countries
mat <- data.table(sobol_matrices(matrices = matrices, N = N, params = params,
                                 order = order, type = type))

# Transform to appropriate distributions -----
mat <- transform_columns_fun(mat = mat, distr = gww)

# Calculate output vector -----
y.irrigation <- rowSums(mat)
hist(y.irrigation)

```

Histogram of y.irrigation



```
# DATASETS WITH INDUSTRY AND DOMESTIC VALUES #####
# Aquastat dataset -----
# https://data.apps.fao.org/aquastat/?lang=en&share=f-79eb812e-6cb9-406f-86f2-b7ee414c0ec6
aquastat.ind.dom <- fread("industrial Domestic_aquastat.csv") %>%
  .[, Country:= countrycode(Area, origin = "country.name", destination = "country.name")]

## Warning: Some values were not matched unambiguously: Australia and New Zealand, Central and
## Warning: Some strings were matched more than once, and therefore set to <NA> in the result:
aquastat.tmp <- aquastat.ind.dom[, .(Country, Variable, Value)] %>%
  .[Country %in% countries] %>%
  dcast(., Country ~ Variable, value.var = "Value") %>%
  setnames(., colnames(.)[2:3], c("Industry", "Domestic")) %>%
  .[order(Country)]

# Liu dataset -----
liu.dt <- data.table(read.xlsx("liu_dataset.xlsx")) %>%
  .[, Country:= countrycode(country, origin = "country.name", destination = "country.name")]
  .[Country %in% countries]

# Gleick dataset -----
```

```

gleick.dt <- data.table(read.xlsx("gleick_table.xlsx"))
colnames_numeric <- colnames(gleick.dt)[-c(1,2)]
gleick.dt <- gleick.dt[, (colnames_numeric) := lapply(.SD, as.numeric), .SDcols = (colnames_numeric)
  .[, total.irrigation:= (agricultural / 100) * total.withdrawal] %>%
  .[, total.industry:= (industrial / 100) * total.withdrawal] %>%
  .[, total.domestic:= (domestic / 100) * total.withdrawal] %>%
  .[, Country:= countrycode(country, origin = "country.name", destination = "country.name")] %>%
  .[Country %in% countries]

# INDUSTRY #####
# Khan et al 2023 dataset ----

path.projections <- "./files/khan_et_al_2023/industry"
list.of.files <- list.files(path.projections, pattern = "\\.csv$")
combinations <- lapply(list.of.files, function(x) strsplit(x, "_")[[1]][1:4]) %>%
  do.call(rbind, .) %>%
  data.frame()
colnames(combinations) <- c("SSP", "RCP", "Climate", "Use")

# Create parallel cluster ----

numCores <- detectCores() * 0.75
cl <- makeCluster(numCores)
registerDoParallel(cl)

# Run for loop ----

result <- foreach(i = 1:length(list.of.files),
  .combine = "rbind",
  .packages = c("data.table", "countrycode", "tidyverse",
    "sp", "rworldmap")) %dopar% {

  out <- fread(paste("./files/khan_et_al_2023/industry",
    list.of.files[i], sep = "/"))

  out <- out[, `:=` (SSP = combinations[i, 1],
    RCP = combinations[i, 2],
    Climate = combinations[i, 3],
    Use = combinations[i, 4])]

  Country <- coords2country(out[1:nrow(out), 2:3])

  df <- cbind(Country, out) %>%
    .[, Continent := countrycode(Country, origin = "country.name",
      destination = "continent")] %>%
    .[, Country:= countrycode(Country, origin = "country.name")]

```

```

            destination = "country.name")] %>
  .[, Dataset:= list.of.files[i]]

  df
}

# Stop the cluster after the computation -----
stopCluster(cl)

# Arrange Khan data -----
khan.industry <- result[, sum(`2010`), .(Country, Dataset)] %>%
  .[Country %in% countries] %>%
  .[, .(min = min(V1), max = max(V1)), Country] %>%
  .[order(Country)]

# Huang et al 2018 dataset -----
path.projections <- "./files/huang_et_al_2018/industrial"
list.of.files <- list.files(path.projections, pattern = "\\.nc$")
file <- paste("./files/huang_et_al_2018/industrial",
              list.of.files, sep = "/")
variables <- sub("\\..*", "", list.of.files)

# Retrieve data -----
out <- list()

for (i in 1:length(file)) {

  out[[i]] <- get_huang_fun(nc_file = file[[i]], variable = variables[[i]], year = "year_2010")
}

## Warning: Some values were not matched unambiguously: Siachen Glacier
## Warning: Some values were not matched unambiguously: Siachen Glacier
## Warning: Some values were not matched unambiguously: Siachen Glacier
huang.industry <- data.table(Country = out[[1]]$Country,
                               industrial.huang = out[[1]]$V1 +
                                 out[[2]]$V1 +
                                 out[[3]]$V1) %>%
  .[Country %in% countries]

# ISIMIP: Create vector with list of files -----

```

```

path <- "./files/isimip/industrial"
list.of.files <- list.files(path)
files.directory <- paste(path, list.of.files, sep = "/")
variable <- "aindww"
year <- "year_2010"
names.models <- lapply(list.of.files, function(x) sub("_.*", "", x))
start_year <- 1971

# Create parallel cluster -----
numCores <- detectCores() * 0.75
cl <- makeCluster(numCores)
registerDoParallel(cl)

# Run for loop -----
result <- foreach(i = 1:length(files.directory),
  .packages = c("data.table", "countrycode", "tidyverse",
    "sp", "rworldmap", "ncdf4")) %dopar% {

  start_year <- ifelse(names.models[i] == "cwatm" | names.models[i] == "h08",
    ifelse(names.models[i] == "miroc-integ-land", 1901, 1971))

  get_isimip_fun(nc_file = files.directory[i],
    variable = variable,
    year = year,
    start_year = start_year)
}

# Stop the cluster after the computation -----
stopCluster(cl)

# Arrange data -----
names(result) <- list.of.files

isimip.industry <- rbindlist(result, idcol = "ID") %>%
  .[, .(min = min(V1), max = max(V1)), Country] %>%
  .[Country %in% countries]

# Merge all industry datasets -----
industry.sample.matrix <- merge(khan.industry, aquastat.tmp[, .(Country, Industry)],
  by = "Country") %>%
  merge(., liu.dt[, .(Country, ind)], by = "Country") %>%

```

```

merge(., isimip.industry[, .(Country, min, max)], by = "Country") %>%
merge(., huang.industry[, .(Country, industrial.huang)], by = "Country") %>%
merge(., gleick.dt[, .(Country, total.industry)], by = "Country") %>%
melt(., measure.vars = colnames(.)[-1]) %>%
.[, .(min = min(value), max = max(value)), Country]

# Define sample matrix -----
params <- industry.sample.matrix$Country

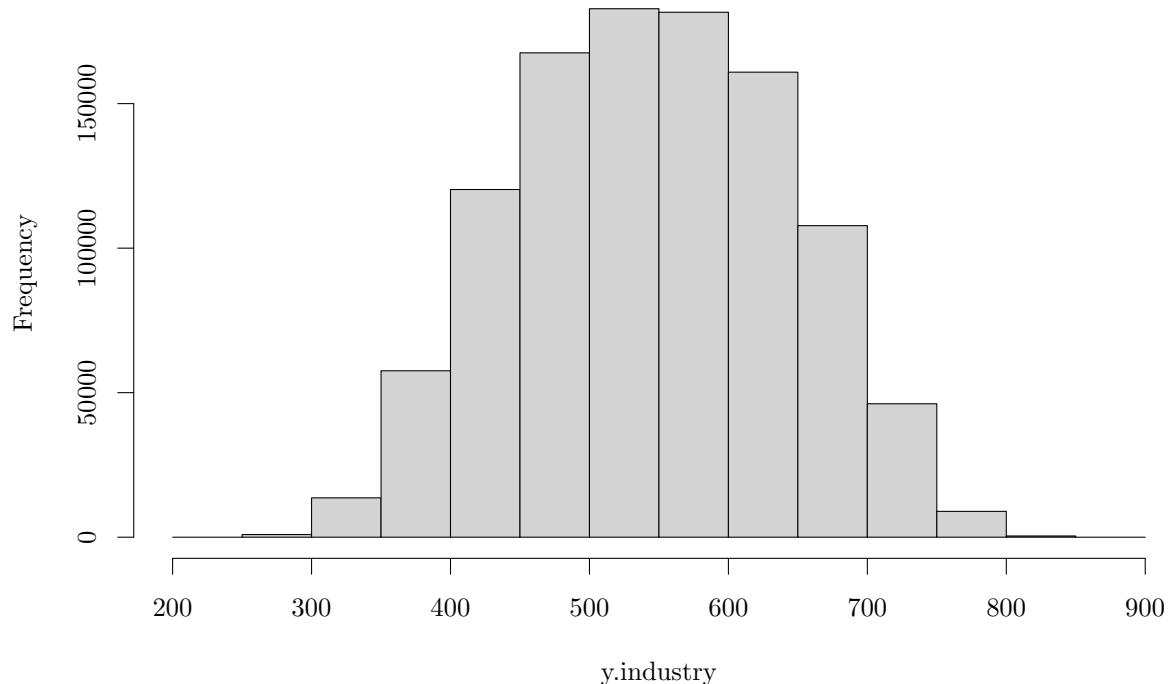
mat <- data.table(sobol_matrices(matrices = matrices, N = N, params = params,
                                 order = order, type = type))

# Transform to appropriate distributions -----
mat <- transform_columns_fun(mat = mat, distr = industry.sample.matrix)

# Calculate output vector -----
y.industry <- rowSums(mat, na.rm = TRUE)
hist(y.industry)

```

Histogram of y.industry



```
max(y.industry)
```

```
## [1] 851.5688
```

```

# DOMESTIC #####
# Khan et al 2023 dataset ----

path.projections <- "./files/khan_et_al_2023/domestic"
list.of.files <- list.files(path.projections, pattern = "\\.csv$")
combinations <- lapply(list.of.files, function(x) strsplit(x, "_")[[1]][1:4]) %>%
  do.call(rbind, .) %>%
  data.frame()
colnames(combinations) <- c("SSP", "RCP", "Climate", "Use")

# Create parallel cluster ----

numCores <- detectCores() * 0.75
cl <- makeCluster(numCores)
registerDoParallel(cl)

# Run for loop ----

result <- foreach(i = 1:length(list.of.files),
  .combine = "rbind",
  .packages = c("data.table", "countrycode", "tidyverse",
    "sp", "rworldmap")) %dopar% {

  out <- fread(paste("./files/khan_et_al_2023/domestic",
    list.of.files[i], sep = "/"))

  out <- out[, `:=` (SSP = combinations[i, 1],
    RCP = combinations[i, 2],
    Climate = combinations[i, 3],
    Use = combinations[i, 4])]

  Country <- coords2country(out[1:nrow(out), 2:3])

  df <- cbind(Country, out) %>%
    .[, Continent := countrycode(Country, origin = "country.name",
      destination = "continent")] %
    .[, Country := countrycode(Country, origin = "country.name",
      destination = "country.name")] %
    .[, Dataset := list.of.files[i]]

  df
}

# Stop the cluster after the computation ----

stopCluster(cl)

```

```

# Arrange Khan data -----
khan.domestic <- result[, sum(`2010`), .(Country, Dataset)] %>%
  .[Country %in% countries] %>%
  .[, .(min = min(V1), max = max(V1)), Country] %>%
  .[order(Country)]

# Huang et al 2018 dataset -----
path.projections <- "./files/huang_et_al_2018/domestic"
list.of.files <- list.files(path.projections, pattern = "\\.nc$")
file <- paste(path.projections,
              list.of.files, sep = "/")
variables <- sub("\\..*", "", list.of.files)

# Retrieve data -----
huang.domestic <- get_huang_fun(nc_file = file, variable = variables, year = "year_2010") %>%
  .[Country %in% countries]

## Warning: Some values were not matched unambiguously: Siachen Glacier

# ISIMIP: Create vector with list of files -----
path <- "./files/isimip/domestic"
list.of.files <- list.files(path)
files.directory <- paste(path, list.of.files, sep = "/")
variable <- "adomww"
year <- "year_2010"
names.models <- lapply(list.of.files, function(x) sub("_.*", "", x))
start_year <- 1971

# Create parallel cluster -----
numCores <- detectCores() * 0.75
cl <- makeCluster(numCores)
registerDoParallel(cl)

# Run for loop -----
result <- foreach(i = 1:length(files.directory),
  .packages = c("data.table", "countrycode", "tidyverse",
    "sp", "rworldmap", "ncdf4")) %dopar% {

  start_year <- ifelse(names.models[i] == "miroc-integ-land" | names.models[i] == "cwatm", 1901, 1971)

```

```

        get_isimip_fun(nc_file = files.directory[i],
                        variable = variable,
                        year = year,
                        start_year = start_year)
    }

# Stop the cluster after the computation -----
stopCluster(cl)

# Arrange data -----
names(result) <- list.of.files

isimip.domestic <- rbindlist(result, idcol = "ID") %>%
  .[, .(min = min(V1), max = max(V1)), Country] %>%
  .[Country %in% countries]

# Merge all domestic datasets -----
domestic.sample.matrix <- merge(khan.domestic,
                                 aquastat.tmp[, .(Country, Domestic)],
                                 by = "Country") %>%
  merge(., huang.domestic[, .(Country, V1)], by = "Country") %>%
  merge(., liu.dt[, .(Country, muni)], by = "Country") %>%
  merge(., isimip.domestic[, .(Country, min, max)], by = "Country") %>%
  merge(., gleick.dt[, .(Country, total.domestic)], by = "Country") %>%
  melt(., measure.vars = colnames(.)[-1]) %>%
  .[, .(min = min(value), max = max(value)), Country] %>%
  na.omit()

# Define sample matrix -----
params <- domestic.sample.matrix$Country

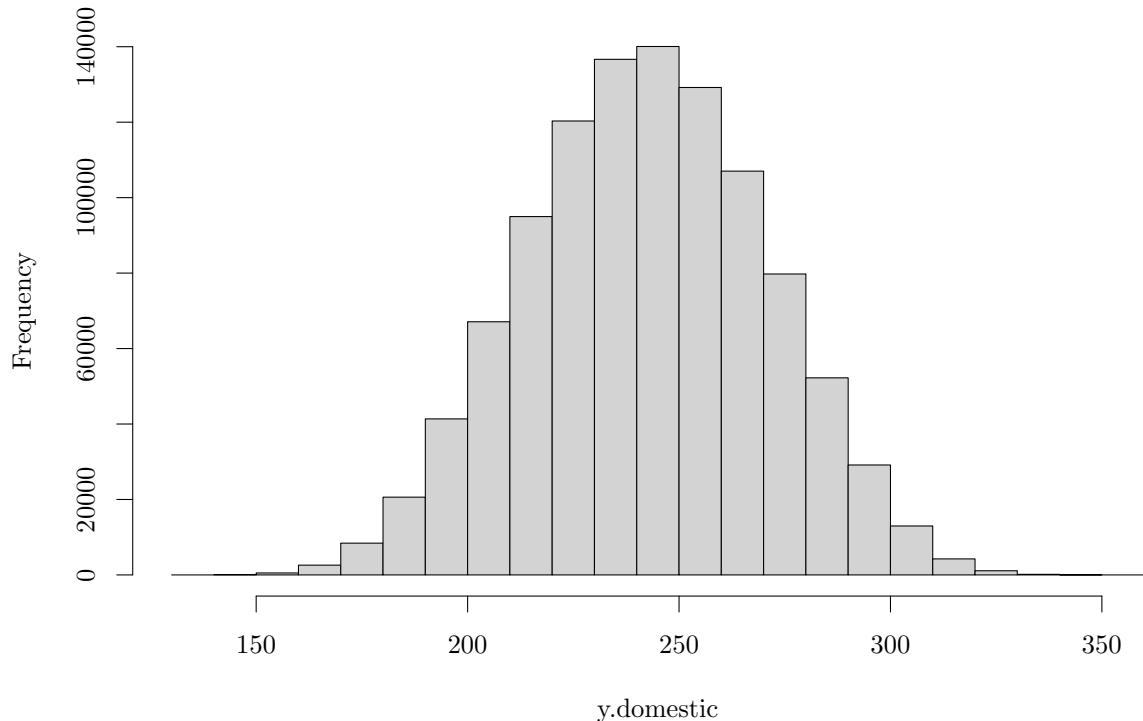
mat <- data.table(sobol_matrices(matrices = matrices, N = N, params = params,
                                  order = order, type = type))

# Transform to appropriate distributions -----
mat <- transform_columns_fun(mat = mat, distr = domestic.sample.matrix)

# Calculate output vector -----
y.domestic<- rowSums(mat, na.rm = TRUE)
hist(y.domestic)

```

Histogram of y.domestic



```
# CALCULATE PERCENTAGES #####
percentages.dt <- data.table(irrigation = y.irrigation,
                               industry = y.industry,
                               domestic = y.domestic) %>%
  .[, total:= rowSums(.SD)] %>%
  .[, percent.irrig:= (irrigation / total) * 100] %>%
  .[, percent.indus:= (industry / total) * 100] %>%
  .[, percent.domestic:= (domestic / total) * 100]

# Stats -----
percentages.dt[, .(min = min(percent.irrig), max = max(percent.irrig))]

##           min      max
##       <num>    <num>
## 1: 51.35991 91.0266

quantile(percentages.dt$percent.irrig, probs = c(0.025, 0.975))

##      2.5%     97.5%
## 69.04671 86.30175

# Plot -----
vector.sectors <- c("irrigation", "industry", "domestic")
```

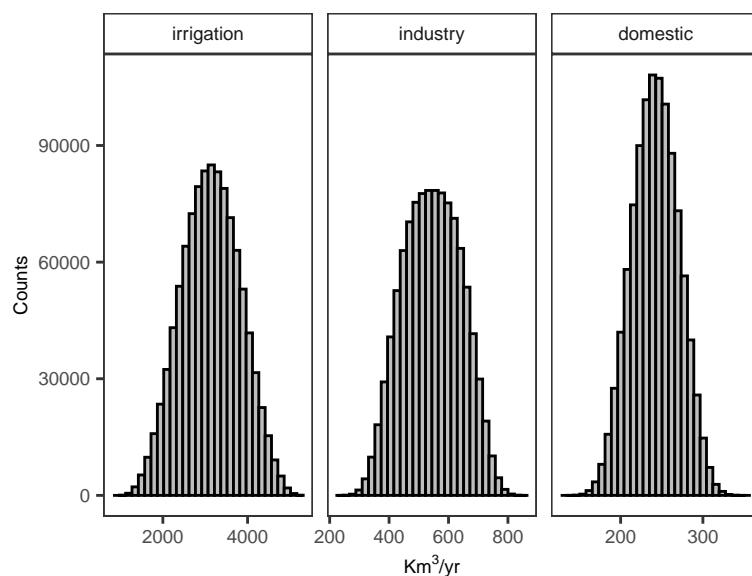
```

sectors.plot <- melt(percentages.dt,
                      measure.vars = vector.sectors) %>%
  ggplot(., aes(value)) +
  geom_histogram(color = "black", fill = "grey") +
  facet_wrap(~variable, scale = "free_x", ncol = 4) +
  scale_x_continuous(breaks = breaks_pretty(n = 3)) +
  labs(x = bquote("Km"^3 * "/yr"), y = "Counts") +
  theme_AP()

sectors.plot

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

```



```

total.plot <- percentages.dt %>%
  ggplot(., aes(total)) +
  geom_histogram(color = "black", fill = "grey") +
  scale_x_continuous(breaks = breaks_pretty(n = 3)) +
  labs(x = bquote("Km"^3 * "/yr"), y = "Counts") +
  theme_AP()

percentages.plot <- percentages.dt[, .(percent.irrig, percent.indus, percent.domestic)] %>%
  setnames(., colnames(.), vector.sectors) %>%
  melt(., measure.vars = vector.sectors) %>%
  ggplot(., aes(value, fill = variable)) +
  labs(x = "Water withdrawal (%)", y = "Count") +
  scale_fill_discrete(name = "") +
  geom_histogram(alpha = 0.6, position = "identity") +
  geom_vline(xintercept = 70, lty = 2) +
  theme_AP() +
  theme(legend.position = c(0.45, 0.85))

## Warning: A numeric `legend.position` argument in `theme()` was deprecated in ggplot2

```

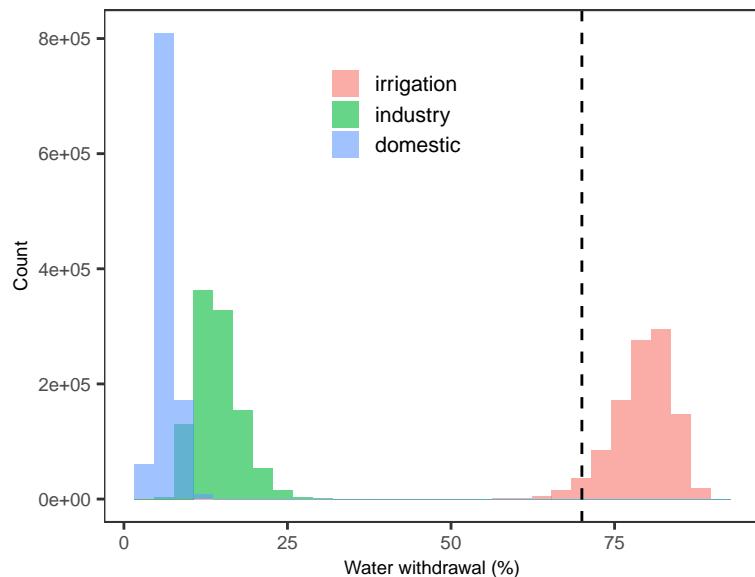
```

## 3.5.0.
## i Please use the `legend.position.inside` argument of `theme()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

percentages.plot

```

`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

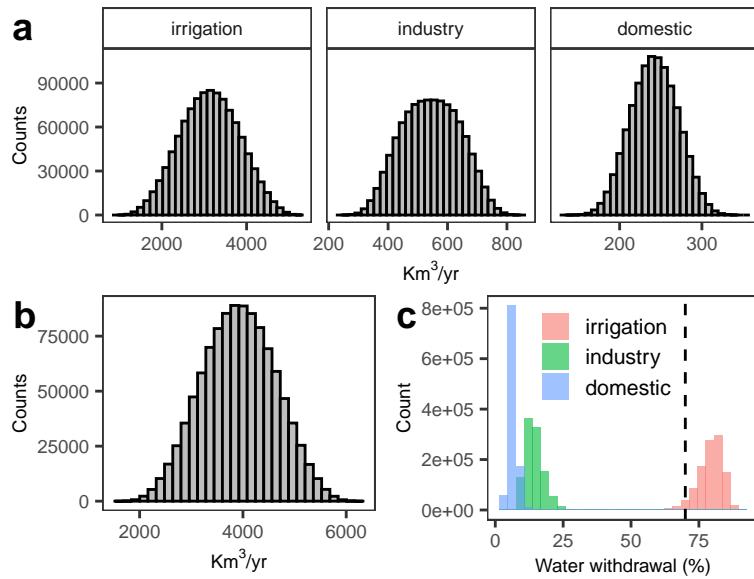


```
bottom <- plot_grid(total.plot, percentages.plot, ncol = 2, labels = c("b", "c"))
```

`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
 ## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

```
plot_grid(sectors.plot, bottom, ncol = 1, labels = c("a", ""))
```

`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.



7 Session information

```
# SESSION INFORMATION #####
sessionInfo()

## R version 4.3.3 (2024-02-29)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Sonoma 14.2.1
##
## 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;
##
## 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: Europe/London
## tzcode source: internal
##
## attached base packages:
## [1] parallel stats      graphics grDevices utils      datasets methods
## [8] base
##
## other attached packages:
## [1] doParallel_1.0.17 iterators_1.0.14 foreach_1.5.2   countrycode_1.5.0
## [5] scales_1.3.0      wesanderson_0.3.6 benchmarkme_1.0.8 tidygraph_1.3.0
## [9] cowplot_1.1.3     ggraph_2.1.0      igraph_2.0.3    bibliometrix_4.0.1
## [13] lubridate_1.9.2   forcats_1.0.0     stringr_1.5.1   dplyr_1.1.4
## [17] purrrr_1.0.2     readr_2.1.4      tidyr_1.3.1    tibble_3.2.1
```

```

## [21] ggplot2_3.5.1      tidyverse_2.0.0      data.table_1.14.99 openxlsx_4.2.5.2
##
## loaded via a namespace (and not attached):
## [1] gridExtra_2.3          readxl_1.4.2        rlang_1.1.4
## [4] magrittr_2.0.3         tidytext_0.4.1       compiler_4.3.3
## [7] vctrs_0.6.5           pkgconfig_2.0.3     fastmap_1.1.1
## [10] ellipsis_0.3.2        utf8_1.2.4          promises_1.2.0.1
## [13] rmarkdown_2.21         tzdb_0.3.0          xfun_0.39
## [16] jsonlite_1.8.4        flashClust_1.01-2   SnowballC_0.7.1
## [19] later_1.3.0          tweenr_2.0.2         cluster_2.1.6
## [22] R6_2.5.1              stringi_1.8.3       RColorBrewer_1.1-3
## [25] cellranger_1.1.0     estimability_1.4.1  Rcpp_1.0.12
## [28] knitr_1.42            filehash_2.4-5       httpuv_1.6.9
## [31] rentrez_1.2.3         Matrix_1.6-5         timechange_0.2.0
## [34] tidyselect_1.2.0       rstudioapi_0.15.0    stringdist_0.9.10
## [37] pubmedR_0.0.3         yaml_2.3.7          viridis_0.6.4
## [40] codetools_0.2-19      lattice_0.22-5      plyr_1.8.8
## [43] shiny_1.7.4            withr_3.0.0         benchmarkmeData_1.0.4
## [46] coda_0.19-4           evaluate_0.20       polyclip_1.10-6
## [49] zip_2.3.0              pillar_1.9.0         janeaustenr_1.0.0
## [52] DT_0.27                plotly_4.10.1       generics_0.1.3
## [55] hms_1.1.3              munsell_0.5.1       xtable_1.8-4
## [58] leaps_3.1              glue_1.7.0          tikzDevice_0.12.4
## [61] emmeans_1.8.5          scatterplot3d_0.3-43 lazyeval_0.2.2
## [64] tools_4.3.3             tokenizers_0.3.0     mvtnorm_1.1-3
## [67] graphlayouts_1.1.1     XML_3.99-0.14       grid_4.3.3
## [70] rscopus_0.6.6          colorspace_2.1-0     dimensionsR_0.0.3
## [73] ggforce_0.4.1          bibliometrixData_0.3.0 cli_3.6.3
## [76] fansi_1.0.6             viridisLite_0.4.2    gtable_0.3.5
## [79] digest_0.6.34          ggrepel_0.9.5       FactoMineR_2.8
## [82] htmlwidgets_1.6.2       farver_2.1.2         htmltools_0.5.5
## [85] factoextra_1.0.7       lifecycle_1.0.4     httr_1.4.5
## [88] multcompView_0.1-9     mime_0.12           MASS_7.3-60.0.1

## Return the machine CPU -----
cat("Machine:      "); print(get_cpu()$model_name)

## Machine:
## [1] "Apple M1 Max"

## Return number of true cores -----
cat("Num cores:    "); print(detectCores(logical = FALSE))

## Num cores:
## [1] 10

```

```
## Return number of threads -----  
cat("Num threads: "); print(detectCores(logical = FALSE))  
  
## Num threads:  
## [1] 10
```