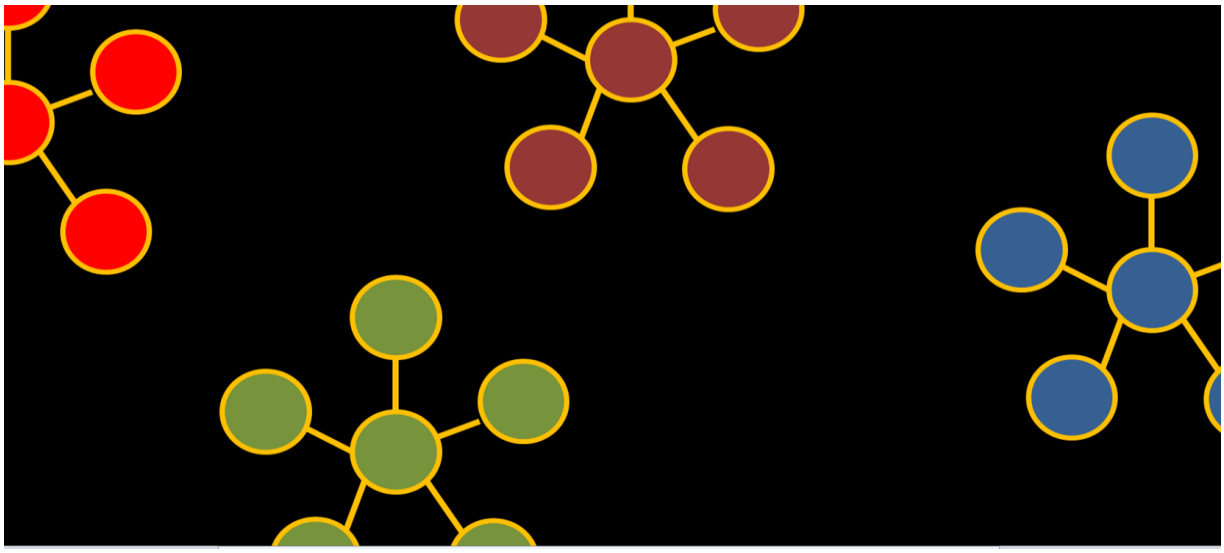


Getting started with GloBI to create networks

Daily, millions of species are interacting. When species occurences and their interactions are reported by the wider public, a valuable treasure of data is created which helps to understand how ecosystems are functioning worldwide. In the light of global change, it is important to not only study species extinctions but also extinctions of interactions due to unbalanced range shifting of interacting species (Valiente-Banuet et al., 2014). Moreover, the impact of invasive species on biodiversity can be estimated by their interaction network.

An interaction network consists of species (circles or nodes) which are connected by lines representing an interaction (e.g. preys on, pollinates, etc.). Within a network, species can be either directly or indirectly linked. In this blogpost, I wrote out a workflow to create the interaction network of *Vespa velutina* in Belgium, containing both direct and indirect interactions, based on data available on GloBI and the Belgian species cube.

GloBI indexes species interaction data. It also offers the possibility to visualise species interactions. However, if you want to dive into the world of GloBI to explore its data for research, it is worth the effort to learn how to work with the complete GloBI database. Here, I offer a starting guide (set up with the help of Jorrit Poelen) on how to access the complete GloBI database to create networks. The flow presented below is largely based on interactIAS, a Jupyter notebook created by Quentin Groom. This work was supported by Action CA17122 Increasing understanding of alien species through citizen science (Alien-CSI), supported by COST (European Cooperation in Science and Technology www.cost.eu) through a virtual short term scientific mission.

Install & setup WSL

The fastest way to perform data wrangling on large datafiles is using the Linux command line. If you have a Windows laptop (like me) the best way to get started is by installing WSL (Windows subsystem for Linux) following this instruction guide. I choose to install Ubuntu as Linux distribution.

After installation, open the Linux distribution by the start Menu: a Linux terminal will appear. The first time you open this terminal you will be asked to create a Linux username and password.

Update and upgrade your repository

```
sudo apt update
sudo apt upgrade
```

Now, use the pwd command to find out your current working directory.

```
pwd
```

You can create a new folder by:

```
mkdir /home/username/newfolder
```

Define this new folder as your working directory:

```
cd /home/username/newfolder
```

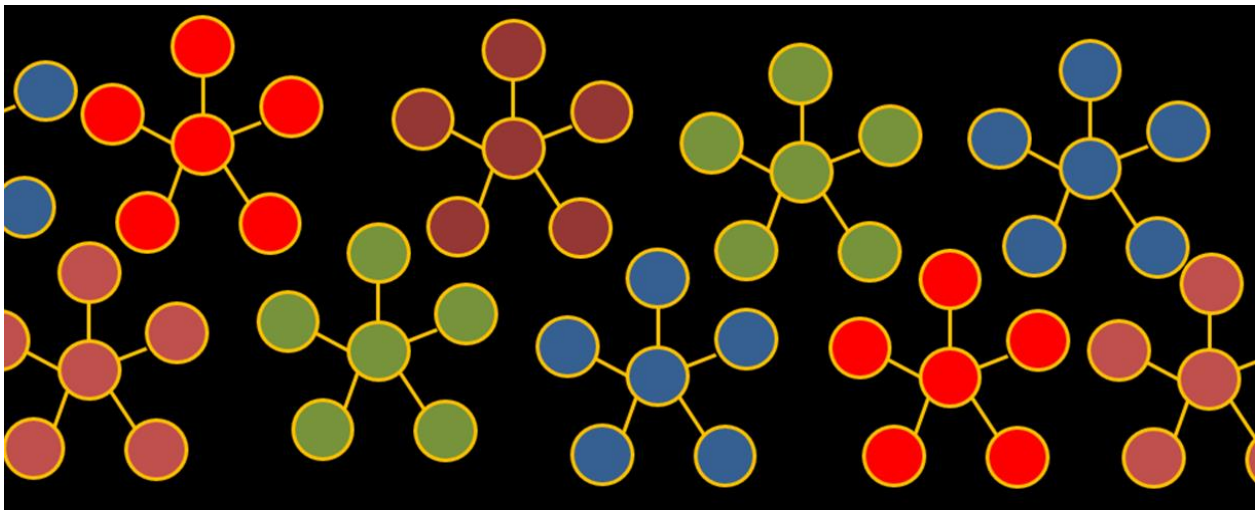
Data filtering from entire GloBi database

Now, go to the GloBI website and download latest stable version of the entire GloBI database. The database of GloBI is available in many formats. For the following, you are advised to download a stable (citable) version of the database in tsv format. Place the download into the working directory after downloading.

Check the content of this file by printing the first line:

```
cat interactions.tsv.gz | gunzip | head -n 1
```

In this example, we are interested in creating a network for *Vespa velutina*, containing both direct and indirect interactions.



Getting direct interactions from GloBI

To obtain the direct interactions, we now search the database for all lines containing *Vespa velutina*. This outcome is saved into a the file vespa_velutina_interactions.tsv by:

```
zgrep "Vespa velutina" interactions.tsv.gz > vespa_velutina_interactions.tsv
```

Herein, zgrep allows searching within zipped files. As such, a file does not not need to be unpacked beforehand. In this example, we are lucky as *Vespa velutina* has no synonyms. In case your species of interest has multiple synonyms, the above code should be adapted to filter interactions.tsv.gz for all synonyms.

Explore the file vespa_velutina_interactions.tsv by printing the first 10 lines of the dataframe within the terminal:

```
cat vespa_velutina_interactions.tsv | head
```

Check the number of lines within vespa_velutina_interactions.tsv by:

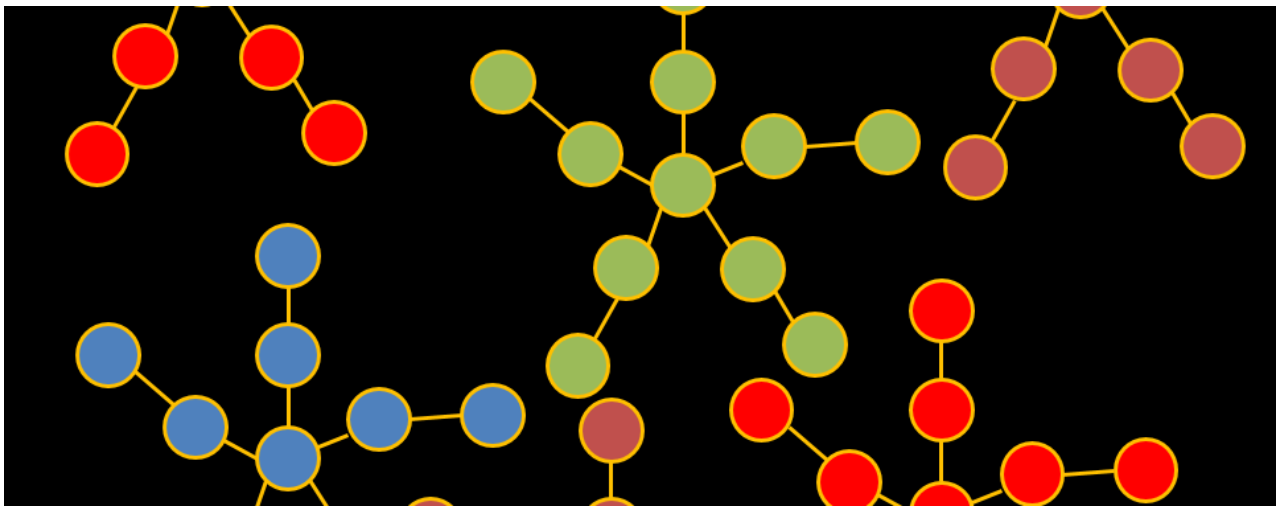
```
cat vespa_velutina_interactions.tsv | wc -l
```

The file vespa_velutina_interactions.tsv contains all direct interactions of *Vespa velutina* within the GloBI database. We clean this final file before importing in R by only saving the following columns and deleting any duplicate rows:

- column 2: taxonids of the source species
- column 3: taxon name
- column 4: taxonomic level of source species
- column 8: mapped source species name
- column 20: phylum name source species
- column 22: kingdom name source species
- column 39: interaction type
- column 42: taxonids of the target species

- column 43: taxon name
- column 44: taxonomic level of target species
- column 48: mapped target species name
- column 60: phylum name target species
- column 62: kingdom name target species

```
cat vespa_velutina_interactions.tsv| cut -f2,3,4,8,20,22,39,42,43,44,48,60,62
| sort | uniq -c | sort -nr > vespa_velutina_interactions-light.tsv
```



Getting indirect interactions from GloBI

Before we continue with filtering the indirect interactions of *Vespa velutina* from GloBI, it is important to highlight that an interaction always consists out of a source species, an interaction type and a target species. In the file vespa_velutina_interactions.tsv, *Vespa velutina* might occur both as source or target species. Now, we want to list all unique source and target species with which *Vespa velutina* is interacting. To do this, we filter all unique species names from column 8 and 48 within the file vespa_velutina_interactions.tsv. Both columns refer to the mapped species name of the source species and target species, respectively.

```
cat vespa_velutina_interactions.tsv| cut -f8 | sort | uniq > vespa_velutina_s
ources.tsv
cat vespa_velutina_interactions.tsv| cut -f48 | sort | uniq > vespa_velutina_
targets.tsv
```

It is important to manually remove the first row of the files in case it represents an empty line.

The indirect interactions of *Vespa velutina* are selected from GloBI by looping over each of these species within both files and writing out all interactions containing these species into an output file (secundary_interactions_sources.tsv and secundary_interactions_targets.tsv, respectively)

```
while read line; do zgrep "$line" interactions.tsv.gz >> secondary_interactio
ns_sources.tsv; done <vespa_velutina_sources.tsv
while read line; do zgrep "$line" interactions.tsv.gz >> secondary_interactio
ns_targets.tsv; done <vespa_velutina_targets.tsv
```

Again, we clean up both output files by only selecting particular columns (see above) and deleting duplicate rows.

```
cat secondary_interactions_sources.tsv| cut -f2,3,4,8,20,22,39,42,43,44,48,60
,62 | sort | uniq -c | sort -nr > secondary_interactions_sources_light.tsv
cat secondary_interactions_targets.tsv| cut -f2,3,4,8,20,22,39,42,43,44,48,60
,62 | sort | uniq -c | sort -nr > secondary_interactions_targets_light.tsv
```

For completeness, I mention that there is a list of refuted interactions available on GloBI. Interactions within this list contain errors and should therefore be excluded from your network. This is not illustrated here.

© Gilles San Martin

Fine-tuning of network within R

Now, we will fine-tune these interactions into a network in R.

First, load all necessary libraries.

```
library(dplyr)
library(stringr)
library(tidyr)
library(rglobi)
library(tidyverse)
library(purrr)
```

Read in all three files containing interactions (primary interactions and secondary interactions of sources and targets) and bind these together in one dataframe.

```
header <- c('sourceTaxonIDs',
            'sourceTaxonName',
            'sourceTaxonLevel',
            'sourceSpeciesName',
            'sourcePhylum',
            'sourceKingdom',
            'interactionType',
            'targetTaxonIDs',
            'targetTaxonName',
            'targetTaxonLevel',
            'targetSpeciesName',
            'targetPhylum',
            'targetKingdom')

#reading in GLOBI output
interactions_sources <- read.csv("secondary_interactions_sources_light.tsv",
  sep = "\t",
  quote="",
  header=FALSE,
  col.names=header)

interactions_targets <- read.csv("secondary_interactions_targets_light.tsv",
  sep = "\t",
  quote="",
  header=FALSE,
  col.names=header)

primary_interactions <- read.csv("vespa_velutina_interactions_light.tsv",
  sep = "\t",
  quote="",
  header=FALSE,
  col.names=header)

raw_interactions <- rbind(interactions_sources,
                          interactions_targets,
                          primary_interactions)
```

Check the type of interactions occuring in your dataset.

```
unique(raw_interactions$interactionType)
```

Define the interactions that are of interest.

```
interactions_to_include <- c("hasHost",
                             "eats",
                             "pathogenOf",
                             "interactsWith",
                             "parasiteOf",
                             "endoparasiteOf",
                             "ectoparasiteOf",
                             "visitsFlowersOf",
                             "preysOn",
                             "visits",
                             "endoparasitoidOf",
                             "mutualistOf",
                             "pollinates",
                             "parasitoidOf",
                             "guestOf",
                             "kills",
                             "ectoParasitoid")
```

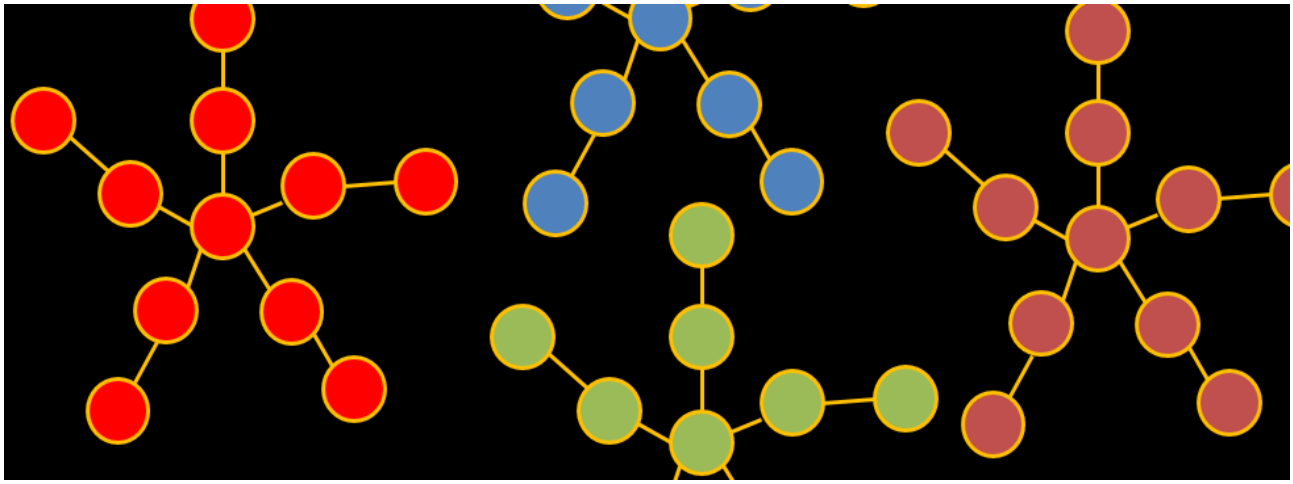
Now, further process the dataframe by: - selecting interactions that are of interest - selecting rows in which species name of source and target are defined (notice we hereby only include taxons at species level in the network) - selecting particular columns - replacing any interaction type or kingdom with a terminology of choice - removing any duplicate rows

```
interactionsCleaned <- raw_interactions %>%
 filter(interactionType %in% interactions_to_include)%>%
 filter(sourceSpeciesName!="")%>%
 filter(targetSpeciesName!="")%>%
 select(sourceSpeciesName,
        sourcePhylum,
        sourceKingdom,
        interactionType,
        targetSpeciesName,
        targetPhylum,
        targetKingdom)%>%
 mutate(interactionType = str_replace(interactionType, "kills", "preyson"))%>
%
 mutate(sourceKingdom=str_replace(sourceKingdom, 'Metazoa', 'Animalia'))%>%
 mutate(targetKingdom=str_replace(targetKingdom, 'Metazoa', 'Animalia'))%>%
 distinct()
```

Which species occur in this network? Export these within the file all_species_network.csv.

```
all_species <- sort(
            unique(
              c(interactionsCleaned$sourceSpeciesName,
                interactionsCleaned$targetSpeciesName)
              )
            )

 write.table(all_species,
             'all_species_network.csv',
              row.names=FALSE,
              col.names=FALSE,
              quote=FALSE)
```



Intermezzo: taxonomic alignment with nomer

In order to create the Belgian network of *Vespa velutina*, we will only incorporate species that have been observed in Belgium since 2000 according to GBIF. To do this, we use nomer to obtain the GBIF speciesKey of each species and then link this to the speciesKeys mentioned within the Belgian occurence cube. There are multiple ways to obtain the GBIF taxonKey of a list of species (such as the function name_backbone from rgbif) but for increased speed I here demonstrate how to apply nomer.

Nomer is available as a linux package. So we return to the Linux command line to perform the following steps. First update and upgrade the repository before you install nomer:

```
sudo apt update
sudo apt upgrade
```

Install curl if not yet installed. This package allows you to download nomer from github.

```
sudo apt install curl
```

Download Nomer locally. Check for the most recent versions of nomer here and adapt url-link below.:

```
curl -L https://github.com/globalbioticinteractions/nomer/releases/download/0
.4.8/nomer.deb > nomer.deb
```

Install Nomer and its dependencies:

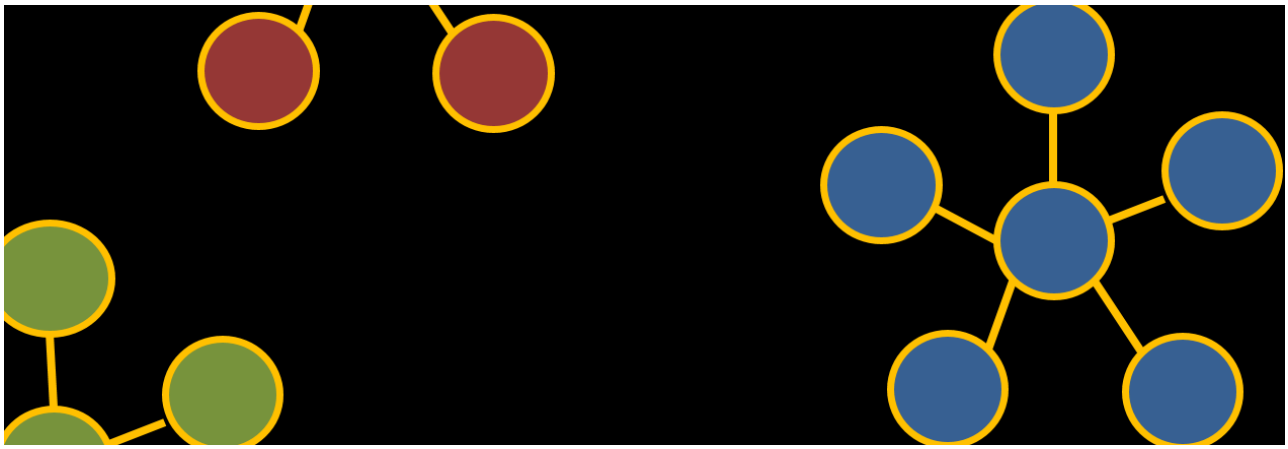
```
sudo apt install ./nomer.deb
```

Download taxonomic backbone of GBIF for nomer locally. Check for the most recent versions of nomer here and adapt url-link below.

```
curl -L https://github.com/globalbioticinteractions/nomer/releases/download/0
.4.7/gbif_mapdb.zip > ~/.cache/nomer/gbif_mapdb.zip
```

Now, copy the file all_species_network.csv into your working repository (see explanation above on how to define your working repository in Linux). Add a tab in front of each species name, a necessity for running nomer, and lookup the GBIF taxonomic backbone for each species. Save outcome in all_species_network_gbif.tsv.

```
cat all_species_network.csv |sed "s/^/\t/g" | nomer append gbif> all_species_
network_gbif.tsv
```



Finalizing network

Now we open the output from Nomer in R:

```
headerNames <- c('V1','speciesName','relation','taxonKey', 'GBIFspeciesName',
'author','taxonLevel','V8','taxonomy','GBIFtaxonomy','taxonomyTaxonlevel', 'V
12', 'url')

all_species_network_gbif <- read.csv("all_species_network_gbif.tsv",
                                     sep = "\t",
                                     quote="",
                                     header=FALSE,
                                     col.names = headerNames)
```

It is important to list the names that are not found by Nomer and check for possible errors.

```
not_found_nomer <- all_species_network_gbif %>% filter(relation=='NONE')
write.csv(not_found_nomer, 'not_found_nomer.csv')
```

To create the network, we are only interested in the taxonKeys per species returned by Nomer. We separate the column taxonKey into two new columns: taxonomy and taxonKey based on the separator ':'. As such the value 'GBIF:1311477' is separated into 'GBIF' and '1311477'. We only maintain distinct rows and delete all names that are not recognized by Nomer.

```
speciesNetwork <- all_species_network_gbif %>%
  select(speciesName, taxonKey)%>%
  separate(taxonKey, c('taxonomy', 'taxonKey'),sep=":")%>%
  distinct%>%
  filter(!is.na(taxonKey))
```

Import the species cube for Belgium after downloading it into your R working directory. Also, we define the year from which observations in the cube are considered relevant.

```
year <- 2000

cube_BE <- read_csv('be_species_cube.csv')%>%
  filter(year>=2000)
```

Which species from the network occur in the Belgian species cube?

```
speciesNetworkCubeBE <- speciesNetwork %>% filter(
  taxonKey%in%cube_BE$speciesKey)
```

What are the primary interactions of *Vespa velutina*? To answer this question we look up all interactions having *Vespa velutina* as source and a species within the Belgian cube as target (PartI), and all interactions having a species within the cube as source and *Vespa velutina* as target (PartII). Together, these represent the primary interactions of *Vespa velutina* in Belgium, according to GloBI and the Belgian species cube.

```
primaryInteractionsPartI <- interactionsCleaned %>%
  filter(sourceSpeciesName == "Vespa velutina")%>%
  filter(targetSpeciesName%in%speciesNetworkCubeBE$speciesName)

primaryInteractionsPartII<- interactionsCleaned %>%
  filter(sourceSpeciesName%in%speciesNetworkCubeBE$speciesName)%>%
  filter(targetSpeciesName == "Vespa velutina")
```

What are the primary species?

```
primary_species<- unique(c(primaryInteractionsPartI$targetSpeciesName,
                           primaryInteractionsPartII$sourceSpeciesName))
```

What are the secondary interactions of *Vespa velutina*? To answer this question we look up all interactions having a primary species as source and a species within the Belgian cube as target (PartI), and all interactions having a species within the cube as source and a primary

species as target (PartII). Together, these represent the secondary interactions of *Vespa velutina* in Belgium, according to GloBI and the Belgian species cube.

```
secondaryInteractionsPartI <- interactionsCleaned %>%
  filter(sourceSpeciesName%in%primary_species)%>%
  filter(targetSpeciesName%in%speciesNetworkCubeBE$speciesName)

secondaryInteractionsPartII<- interactionsCleaned %>%
  filter(sourceSpeciesName%in%speciesNetworkCubeBE$speciesName)%>%
  filter(targetSpeciesName%in%primary_species)
```

What are the secondary species? Note that species can be primary and secondary, therefore primary species are deleted from this list.

```
secondary_species<- unique(c(secondaryInteractionsPartI$targetSpeciesName,
                             secondaryInteractionsPartII$sourceSpeciesName))

secondary_species <- secondary_species[!(secondary_species%in%primary_species
)]
```

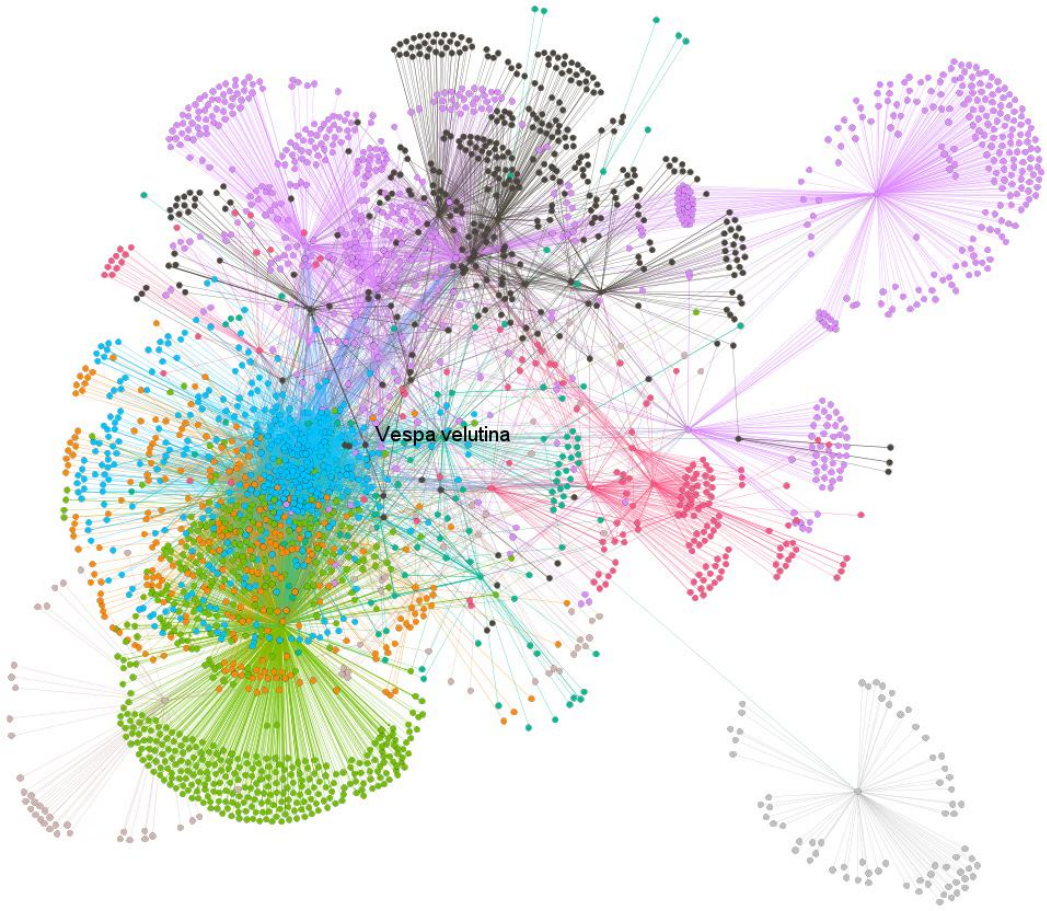
Bind all interactions together and export your network.

```
PrimSecInteractions <- rbind(primaryInteractionsPartI,
                             primaryInteractionsPartII,
                             secondaryInteractionsPartI,
                             secondaryInteractionsPartII)%>%
  select(sourceSpeciesName, interactionType, targetSpeciesName)%>%
  rename(source=sourceSpeciesName,
         interaction=interactionType,
         target=targetSpeciesName)%>%
  distinct()

 write.csv(PrimSecInteractions, 'edges.csv', row.names=FALSE)
```

Network visualisation in Gephi

There are many options to visualise your network but Gephi is certainly a good candidate. You can find more information on how to create networks in Gephi here. When we visualise the network obtained above we get the following figure:



How and what to cite when creating networks?

Do not forget to refer to the original datasets and sources of the interactions in your network. Their info is described in columns 87 to 91 in the file interactions.tsv.gz.

For instance, the original datasets of the primary interactions described in the file vespa_velutina_interactions.tsv can be found by:

```
cat vespa-velutina-interactions.tsv | cut -f88,89,90,91 | uniq
```

As an example, the reference of each individual interaction in the file vespa_velutina_interactions.tsv can be found by:

```
cat vespa-velutina-interactions.tsv | cut -f87 | uniq
```

Do also cite the version of GloBI that you downloaded: An overview is available here.

The sources of nomer can be found by:

```
nomer properties| grep preston
```

The version of the GBIF taxonomic backbone that is applied by nomer can be found by:

```
nomer properties| grep gbif
```

Refer to this blogpost by https://doi.org/10.5281/zenodo.7576207.

Jasmijn Hillaert (Research Institute for Nature and Forest, Belgium).

Any comments or issues can be posted here or send by email. Any updates of the presented code are available at https://doi.org/10.5281/zenodo.7576207.