rmangal - R access to hosted MANGAL API

The `mangal` project is a data specification and API, desgined to facilitate the retrieval, archival, and re-use of data on ecological interactions.

An overview of the mangal format

The database (and underlying data format) is built around the idea that networks required meta-data to be fully understood. Instead of presenting them as 0/1 matrices, the `mangal` data format uses relations between objects as small as a possible. So as to understand the different elements, let's connect to a server implementing the API:

```
netdb <- mangalapi("http://localhost:8000")
```

The `netdb` object has all the necessary informations for the `rmangal` package to work (and is the first argument of many functions). Let's look at the names of this object:

```
names(netdb)
```

```
##  [1] "base"        "trail"       "dataset"     "environment" "interaction"
##  [6] "item"        "network"     "population"  "reference"   "taxa"
## [11] "trait"       "user"
```

Note that if you are logged-in at this point (see the vignette about *Contributing data*), `netdb` will gain the attributes `auth` and `me`, so let's disregard these. Each element in this array corresponds to a *resource*, *i.e.* a type of object you can interact with. The data specification is a description of (i) the content of each field, and (ii) how it should be formatted. As no one is supposed to remember the whole data specification, a function called `whatIs` will give you a brief overview of what each field is supposed to mean:

```
whatIs(netdb, "taxa")
```

```
##         field                                        help    type  null
## 1        bold             The BOLD identifier of the taxa integer  TRUE
## 2 description             A short description of the taxa  string  TRUE
## 3        gbif             The GBIF identifier of the taxa integer  TRUE
## 5        itis             The ITIS identifier of the taxa integer  TRUE
## 6        name             The scientific name of the taxa  string FALSE
## 7        ncbi    The NCBI Taxonomy identifier of the taxa integer  TRUE
## 9  vernacular The vernacular name of the taxa, in English  string FALSE
##   unique values
## 1   TRUE
## 2  FALSE
## 3   TRUE
## 5   TRUE
## 6   TRUE
## 7   TRUE
## 9  FALSE
```

The `rmangal` package will return you objects as `lists`, and (in case you want to contribute data), will expect objects in the same format. In the `data.frame` returned by `whatIs`, there are all the informations to understand the objects that are returned. If you are not interested in contributing data, you will most likely be OK with the first two columns: the name of the field, and what it means. For example, this data frame gives you all the meta-data associated with a taxa. The `null` column will also tell you which fields are facultative, and which are mandatory.

Let's compare this output with a `taxa` object pulled from the database (we'll see each element of the data structure and how to access it just after):

```
getTaxa(netdb, 1)
```

```
## $bold
## NULL
##
## $description
## NULL
##
## $gbif
## NULL
##
## $id
## [1] "1"
##
## $itis
## NULL
##
## $name
## [1] "Urospermum picrioides"
##
## $ncbi
## NULL
##
## $owner
## [1] "tpoisot"
##
## $vernacular
## [1] ""
```

Describing nodes

Nodes in the networks can be of type `taxa`, `population`, and `item`. Calling `whatIs` on `item` or `population` will show that `population`s are linked to a `taxa`, and that `item`s are linked to a population. Here are the reasons why.

Describing interactions

Meta-data

Pulling data from the database

The data can be accessed with either `get*` or `list*`. A function starting with `get` will retrieve a *single* record, identified by its `id`. A function starting by `list` will return *all* records of a given type (options to filter will be added in future releases). All the functions follow the same naming convention: either `get` or `list`, and the name of the resource (`taxa`, `reference`) with its first letter capitalized. So getting a list of all networks with their name, unique identifier, and number of interactions, is as simple as

```
head(ldply(listNetwork(netdb), summarize, id = id, name = name, n_int = length(interactions)
```

```
## 11 object(s) found
```

```
##   id   name n_int
## 1  1 BAT1CA    52
## 2  2 BAT2CA   113
## 3  3 FRA1OP    46
## 4  4 FRA2OP    55
## 5  5 MED1CA    60
## 6  6 MED2CA   117
```

If we want to have a look at the first network, we simply need to write

```
(net1 <- getNetwork(netdb, 1))
```

```
## $date
## NULL
##
## $description
## NULL
##
## $environment
## list()
##
## $id
## [1] "1"
##
## $interactions
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14"
## [15] "15" "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28"
## [29] "29" "30" "31" "32" "33" "34" "35" "36" "37" "38" "39" "40" "41" "42"
## [43] "43" "44" "45" "46" "47" "48" "49" "50" "51" "52"
##
## $latitude
## NULL
##
## $longitude
## NULL
##
## $metaweb
## [1] FALSE
##
## $name
## [1] "BAT1CA"
##
## $owner
## [1] "tpoisot"
```

And similarly, the content of the first interaction of this network is

```
(int1 <- getInteraction(netdb, net1$interactions[1]))
```

```
## $description
## NULL
##
## $ecotype
## [1] "pollination"
##
## $id
## [1] "1"
```

```
##
## $item_from
## NULL
##
## $item_to
## NULL
##
## $owner
## [1] "tpoisot"
##
## $pop_from
## NULL
##
## $pop_to
## NULL
##
## $strength_f
## [1] 0.1667
##
## $strength_t
## NULL
##
## $taxa_from
## [1] "33"
##
## $taxa_to
## [1] "1"
##
## $units_f
## [1] "visits per minute"
##
## $units_t
## NULL
```

To get a sense of what each property mean, you simply need to call `whatIs(netdb, 'interaction')`.

Example: plotting a network

With this information in hand, getting a full network with all taxa information is simply a matter of following each interaction down to the taxa level, and putting this together in a single object. It's easy, but tedious. The `network_as_graph` function will take care of it automatically, and pull a network a an `igraph` object:

```
G <- network_as_graph(netdb, 1)
A <- get.adjacency(G, sparse = FALSE)
A <- A[-which(rowSums(A) == 0), -which(colSums(A) == 0)]
visweb(A)
```

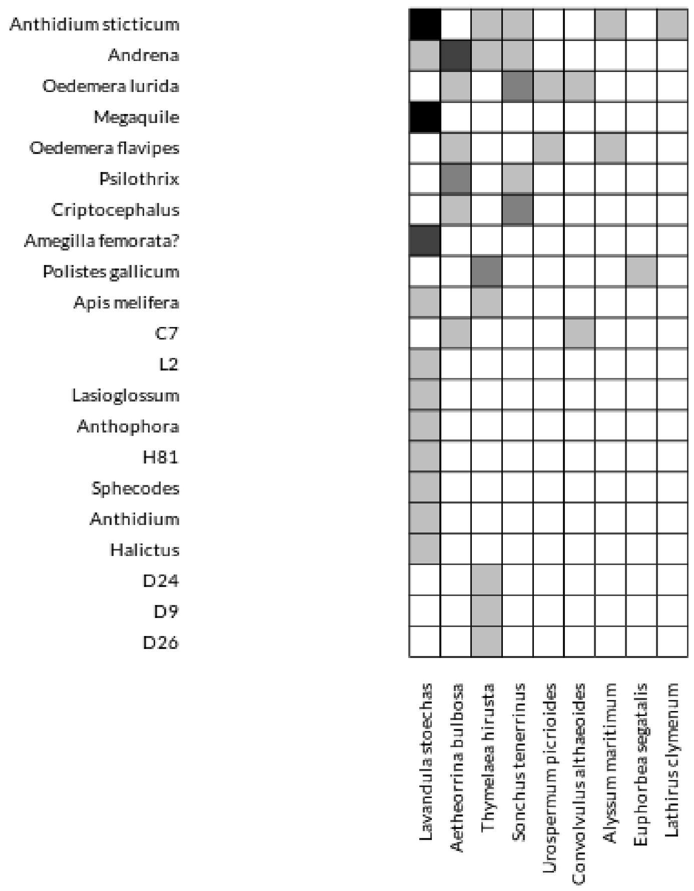

Figure 1: A visualisation of the network.