Christmas_Island_Boolean_approach

21 June 2019

We use the full Town network as an example to illustrate the Boolean approach.

1. Define the species interaction network

We define the Christmas Island Town network by the species present, the interactions between them, and the signs of these interactions. The function initialise_foodweb returns a DiagrammeR graph obect, storing information of the network structure (nodes and edges) as dataframes (i.e. NDF: node data frame, and EDF: edge data frame). The network can be quickly plotted using function *render_graph()* from package *DiagrammeR* for a quick check of network structures. For a neat plot, use funciton *foodweb_neat_plot()*.

```
spp_list = c(
  'cat',
  'rat',
  'crab',
  'goshawk',
  'hawkOwl',
  'tropicBird',
  'flyingFox',
  'feralChicken',
  'kestrel',
  'groundCultivars',
  'canopyCultivars',
  'diurnalInsectResources',
  'nocturnalInsectResources'
)

positive_edges_list = list(
    'cat'= c('tropicBird', 'flyingFox', 'diurnalInsectResources', 'feralChicken', 'rat'),
    'rat'= c('tropicBird', 'groundCultivars', 'canopyCultivars', 'diurnalInsectResources', 'nocturnalIns
    'crab'= c('groundCultivars'),
    'goshawk'= c('rat', 'tropicBird', 'diurnalInsectResources', 'feralChicken'),
    'hawkOwl'= c('rat', 'nocturnalInsectResources'),
    'flyingFox'= c('canopyCultivars'),
    'feralChicken'= c('diurnalInsectResources'),
    'kestrel'= c('rat', 'diurnalInsectResources')
)

negative_edges_list = list(
    'cat'= c('cat'),
    'rat'= c('rat', 'cat', 'crab', 'goshawk', 'hawkOwl', 'kestrel'),
    'crab'= c('crab'),
    'goshawk'= c('goshawk'),
    'hawkOwl'= c('hawkOwl'),
    'tropicBird'= c('tropicBird', 'cat', 'rat', 'goshawk'),
    'flyingFox'= c('flyingFox', 'cat'),
    'feralChicken'= c('feralChicken', 'cat', 'rat', 'goshawk'),
    'kestrel'= c('kestrel'),
    'groundCultivars'= c('groundCultivars', 'crab', 'rat'),
    'canopyCultivars'= c('canopyCultivars', 'flyingFox', 'rat'),
```

```
    'diurnalInsectResources'= c('diurnalInsectResources', 'cat', 'rat', 'goshawk', 'feralChicken', 'kes
    'nocturnalInsectResources'= c('nocturnalInsectResources', 'rat', 'hawkOwl')
)

unmonitored_spp_list = c(
    'crab',
    'kestrel',
    'groundCultivars',
    'canopyCultivars',
    'diurnalInsectResources',
    'nocturnalInsectResources')

unmonitored_spp_list_sim = c(
    'cat',
    'rat',
    'feralChicken',
    'crab',
    'kestrel',
    'groundCultivars',
    'canopyCultivars',
    'diurnalInsectResources',
    'nocturnalInsectResources')

control_list = c('cat','rat')
```

Initialise the full Town network.

```
full_web <- initialise_foodweb(positive_edges_list, negative_edges_list)
```

Using the function *qualitative_community_matrix*, we can convert the interaction network into a qualitative community matrix (Mq), and get two named vectors *labelToIndex* and *indexToLabel* to map species labels (i.e. names) to indices of matrix and vice versa.

```
output <- qualitative_community_matrix(full_web)
Mq <- output$Mq
Mq
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
##  [1,]   -1    0    0    0    0   -1    0    0    0     0     0    -1     0
##  [2,]    0   -1    0    1    1    1    0    0    0     0     0     1     1
##  [3,]    0    0   -1    0    0    0    0    1    0     0     0     0     0
##  [4,]    0   -1    0   -1   -1    0   -1    0    0    -1     0    -1     0
##  [5,]    0   -1    0    1   -1    0   -1    0    0     0     0    -1     0
##  [6,]    1   -1    0    0    0   -1    0    0    0     0     0     0     0
##  [7,]    0    0    0    1    1    0   -1    0    0     0     0     1     1
##  [8,]    0    0   -1    0    0    0    0   -1    0     0     0    -1     0
##  [9,]    0    0    0    0    0    0    0    0   -1     0     1     1     0
## [10,]    0    0    0    1    0    0    0    0    0    -1     0     1     0
## [11,]    0    0    0    0    0    0    0    0   -1     0    -1    -1     0
## [12,]    1   -1   -1    1    1    0   -1    1   -1    -1     1    -1     1
## [13,]    0   -1    0    0    0    0   -1    0    0     0     0    -1    -1
```

```
labelToIndex <- output$labelToIndex
indexToLabel <- output$indexToLabel

unname(indexToLabel[10])
```

```
## [1] "kestrel"
```

```
unname(labelToIndex["kestrel"])
```

```
## [1] 10
```

Get following vectors: - *spp_list_idx* represents indices of species in *spp_list* in order.. - *control_list_idx* represents indices of pest species in *control_list* in order. - *monitored_spp_list* is a vector of species being monitored in the Boolean approach. - *monitored_spp_list_idx* represents indices of species in *monitored_spp_list*.

```
spp_list_idx <- unname(labelToIndex[spp_list])
control_list_idx <- unname(labelToIndex[control_list])

monitored_spp_list <- spp_list[!spp_list %in% unmonitored_spp_list]
monitored_spp_list_idx <- unname(labelToIndex[monitored_spp_list])
```

2.The Boolean approach: parameter sweep on the full web

Storing and comparing string vectors of species-response combinations ('pos' and 'neg') can be extremly time comsuming in large while loop in R. Thus, we store the species-reponses calculated from each matrice as 1s and 0s corresponding to 'pos' and 'neg',and then concatenate this string of 1s and 0s as a binary digit. Lastly, the binary digit is converted as an integer and stored in a vector *collectedRespInt*.

```
set.seed(178)

search_terminator = 0.5
t_max = 1e06 # Here we use 1e6 for illustration, which takes about 3mins.
# t_max = 1e8 # ~ 6hours to run

k = 1
t = 0
t_last_updated = 0

collectedResponses = list()
collectedRespInt = NULL

sz <- dim(Mq)
n <- length(Mq)

start_time <- Sys.time()

while ((((t-t_last_updated) < search_terminator*t) | (t < 1000)) & (t < t_max)){

    valid <- FALSE

    while (!valid) {

      # find a random community matrix that is stable
      maxEig = 1

      while (maxEig > 0) {

          M = matrix(runif(n), sz[1], sz[2]) * Mq
          maxEig <- max(Re(eigen(M, symmetric=FALSE, only.values=TRUE)$values))
```

```
      }

      # Now have a valid stable matrix, find the sensitivity matrix
      Sq <- -solve(M)

      # check validation criteria:  the pest respond negatively to management
      control_easy_cat = (Sq[labelToIndex['cat'],labelToIndex['cat']] > 0)
      control_easy_rat = (Sq[labelToIndex['rat'],labelToIndex['rat']] > 0)
      valid <- all(control_easy_cat, control_easy_rat)
    }

    # Now have a valid stable community matrix

    response <- ifelse(-Sq[spp_list_idx, control_list_idx] < 0, 0, 1)

    respInt <- strtoi(str_c(response, collapse = ""), base = 2L)

    existComb <- is.element(respInt, collectedRespInt)

    if (!existComb) {

        collectedRespInt[k] <- respInt

        k = k + 1
        t_last_updated = t
    }

    t = t + 1
}

end_time <- Sys.time()
time_elapsed = end_time - start_time
print(time_elapsed)
```

```
## Time difference of 2.791105 mins
```

```
t_last_updated
```

```
## [1] 994945
```

```
t
```

```
## [1] 1e+06
```

```
length(collectedRespInt)
```

```
## [1] 7395
```

The integers respresenting possible species-response combinations are now stored in *collectedRespInt*. Now convert the *collectedRespInt* into a dataframe, where species responses are denoted as 1 (positive response) and 0 (negative response). Write the model responses to a csv file.

```
colnames <- unlist(lapply(control_list, function(x) paste0(x, "_", spp_list)))

observedResponse <-
    append((2**length(colnames))-1, collectedRespInt) %>%
    intToBin(.) %>%   # convert the integer back into binary strings
```

```
    str_split(., "") %>%  # chop strings to 0s and 1s
    do.call(rbind, .) %>%
    as.data.frame() %>%
    setNames(., colnames) %>%
    slice(., -1)

write.csv(observedResponse, file = "town_search_full_10e6.csv", row.names=FALSE)
```

Read in the csv file of observed species-response combinations that we saved in advance for the full run of the parameter-sweep (number of matrices produced: 10^8).

```
observedResponse_df <- read.csv("town_search_full_10e8.csv", head = TRUE)
nrow(observedResponse_df)
```

```
## [1] 8930
```

```
# head(observedResponse_df) # Check the dataframe
```

3. Boolean analysis and draw implication networks

3.1. Cat management only

First, performing the Boolean analysis for cat management only.

We write a list of desired responses for this management. The *desiredResponses_c* list is then converted into a Boolean mask *desiredResponsesMask_c*, and passed to the function *getUnobservedInts2*.

Function *getUnobservedInts2* finds the complement of the set of observed responses (i.e. unobserved response combinations).The function first concatenate each observed response combination into a binary string and then convert this binary string into an integer. Then the complement corresponds to the list of integers that are missing from the full set of integers.

```
allResponse <- colnames(observedResponse_df)
# allResponse # check species response

str4true = 'pos'
str4flase = 'neg'

desiredResponses_c = c(
  'cat_tropicBird',
  'cat_goshawk',
  'cat_feralChicken',
  'cat_hawkOwl',
  'cat_rat',
  'cat_flyingFox')

boolLen_c <- length(desiredResponses_c)

desiredResponsesMask_c <- match(desiredResponses_c, allResponse)

unobservedInts_c <- getUnobservedInts2(observedResponse_df, desiredResponsesMask_c, boolLen_c)
length(unobservedInts_c)
```

```
## [1] 22
```

The function *getUnobservedBooldf* turns the list of integers *unobservedInts_c*, corresponding to unobserved species responses, back into a dataframe in Boolean expression (1s and 0s). The function *getUnobservedBooldf*

also add an new column *unob* which is a vector of 1s (the dataframe therefore becomes a truth table) to allow funciton *logicopt()* in Package *LogicOpt* to perform Boolean minimization on the dataframe.

```
unobservedBooldf_c <- getUnobservedBooldf(unobservedInts_c, desiredResponses_c)
```

The function*logicopt()* in Package *LogicOpt* is used to perform the Boolean minimisation.

```
start_time <- Sys.time()

opt_c <- logicopt(unobservedBooldf_c, boolLen_c, 1, mode="espresso") # optimize the truth table
optEqn_c <- tt2eqn(opt_c[[1]], boolLen_c, 1) # show the optimized equations

end_time <- Sys.time()
time_elapsed = end_time - start_time
print(time_elapsed)
```

```
## Time difference of 0.01196504 secs
```

The function *getPCUList* converts the optimized equations (*optEqn_c*) into a list of strings (PCUs) for further analysis.

```
optEqn_c # check the optimized equations
```

```
## [1] "unob = cat_goshawk*cat_feralchicken*cat_rat + cat_tropicbird*cat_goshawk*cat_rat + cat_rat*cat_
```

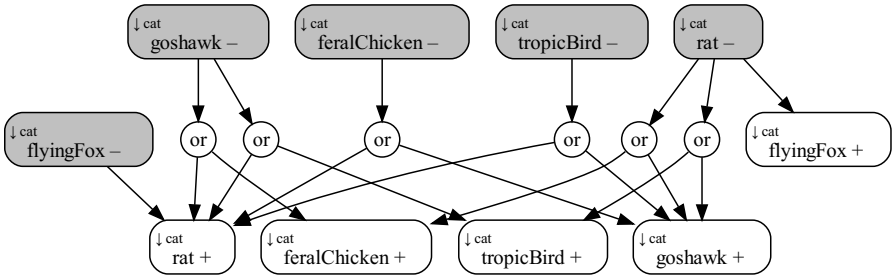
```
PCUList_c <- getPCUList(optEqn_c, str4true, str4flase, desiredResponses_c)
PCUList_c
```

```
## [[1]]
## [1] "negcat_rat"       "negcat_flyingFox"
##
## [[2]]
## [1] "negcat_goshawk"      "negcat_feralChicken" "negcat_rat"
##
## [[3]]
## [1] "negcat_tropicBird" "negcat_goshawk"    "negcat_rat"
```

PCULists are used as inputs to the functions *get_edgelist_singleAnte* and *get_edgelist_certainAnte*, which get an edgelist in a single-antecedent form or a certain-antecedent form. Then the edgelist will be passed to *draw_implication_network* function to plot the corresponding implication network.

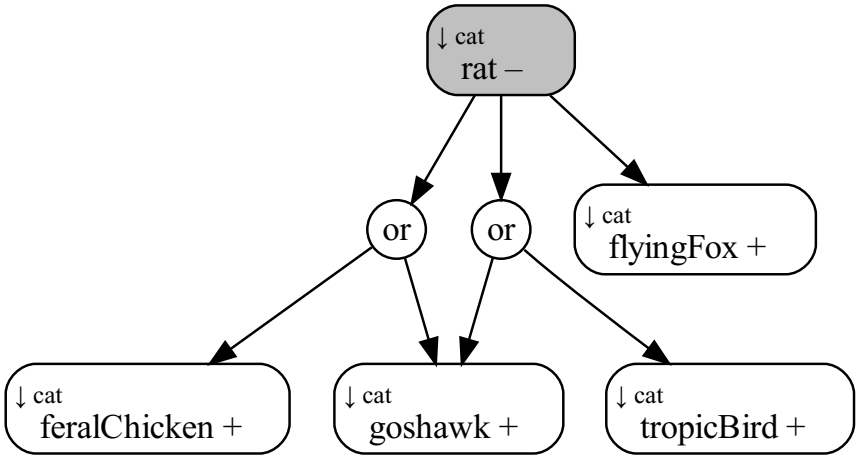
In the following attempt, we can see that species-responses are contingent upon the response of rat population to cat management.

```
edgelist_c1 <- get_edgelist_singleAnte(PCUList_c)
draw_implication_network(edgelist_c1)
```



Above network can be simplified as follows. It can be seen that if cat management has a negative effect on the rat population, certain positive outcomes are guaranteed; while if cat management has a positive effect on the rat population, we found no shorter implication rules.

```
alwaysAnteList_c1 = c('negcat_rat')
edgelist_c2 <- get_edgelist_certainAnte(PCUList_c, alwaysAnteList_c1)
draw_implication_network(edgelist_c2)
```



3.2 Rat management only

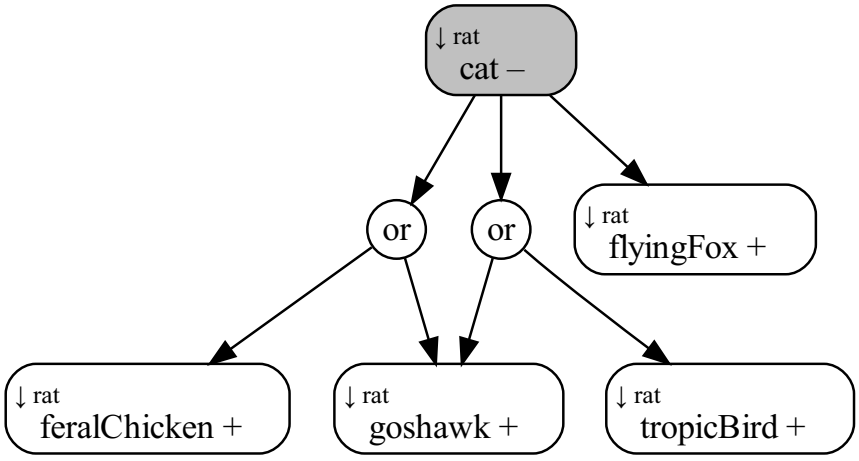
Similar to above procedure, performing the Boolean analysis for rat management only. Write a list of desired responses for this management.Get the PCUList.

```
desiredResponses_r = c(
  'rat_tropicBird',
  'rat_goshawk',
  'rat_feralChicken',
  'rat_hawkOwl',
  'rat_cat',
  'rat_flyingFox')

boolLen_r <- length(desiredResponses_r)
desiredResponsesMask_r <- match(desiredResponses_r, allResponse)

unobservedInts_r <- getUnobservedInts2(observedResponse_df, desiredResponsesMask_r, boolLen_r)

unobservedBooldf_r <- getUnobservedBooldf(unobservedInts_r, desiredResponses_r)

opt_r <- logicopt(unobservedBooldf_r, boolLen_r, 1, mode="espresso")
optEqn_r <- tt2eqn(opt_r[[1]], boolLen_r, 1)

PCUList_r <- getPCUList(optEqn_r, str4true, str4flase, desiredResponses_r)
PCUList_r
```

```
## [[1]]
## [1] "negrat_cat"       "negrat_flyingFox"
##
## [[2]]
## [1] "negrat_goshawk"      "negrat_feralChicken" "negrat_cat"
##
## [[3]]
## [1] "negrat_tropicBird" "negrat_goshawk"    "negrat_cat"
```

We find that species-responses to rat management are contingent upon the response of cat population to rat management. If rat management has a negative effect on the cat population, certain positive outcomes are guaranteed; while if rat management has a positive effect on the rat population, we found no shorter implication rules.

```
alwaysAnteList_r1 = c('negrat_cat')
edgelist_r1 <- get_edgelist_certainAnte(PCUList_r, alwaysAnteList_r1)
draw_implication_network(edgelist_r1)
```



3.3 Combined cat and rat management

Similar to above procedure, performing the Boolean analysis for combined cat and rat management. Write a list of desired responses for this management. Get the PCUList.

```
allResponse <- colnames(observedResponse_df)
# allResponse # check species response

str4true = 'pos'
str4flase = 'neg'

desiredResponses_cr = c(
  'cat_tropicBird',
  'cat_goshawk',
  'cat_feralChicken',
  'cat_hawkOwl',
  'cat_rat',
  'cat_flyingFox',
  'rat_tropicBird',
  'rat_goshawk',
  'rat_feralChicken',
  'rat_hawkOwl',
  'rat_cat',
  'rat_flyingFox')

boolLen_cr <- length(desiredResponses_cr)
desiredResponsesMask_cr <- match(desiredResponses_cr, allResponse)

unobservedInts_cr <- getUnobservedInts2(observedResponse_df, desiredResponsesMask_cr, boolLen_cr)

unobservedBooldf_cr <- getUnobservedBooldf(unobservedInts_cr, desiredResponses_cr)
```

```
opt_cr <- logicopt(unobservedBooldf_cr, boolLen_cr, 1, mode="espresso")
optEqn_cr <- tt2eqn(opt_cr[[1]], boolLen_cr, 1)

PCUList_cr <- getPCUList(optEqn_cr, str4true, str4flase, desiredResponses_cr)
# PCUList_cr  # check the PCUList
```

The PCUList is rather long and implication rules for the combined cat and rat management can be complicated. Thus We provide another document for truncating and simplifying complicated implication networks for the combined management. See *Christmas_Island_complicated_implication_networks*.