faoswsAupus: A package replicating the logic of the AUPUS statistical working system

Joshua M. Browning, Michael. C. J. Kao
Food and Agriculture Organization
of the United Nations

Abstract

This vignette provides a detailed description of the usage of functions in the **faoswsAupus** package.

Keywords: AUPUS, Standardization.

1. Data Initialization

First, we need to load the relevant packages:

```
library(faoswsAupus)
library(faosws)
library(igraph)
library(data.table)
```

This package comes with many functions for replicating the AUPUS and standardization procedures as well as several useful datasets. The first we will use is called 'US', and it contains an example of the data that is used within this package. There is also a variable called usAupusParam, and that variable contains information about the US dataset.

```
is(US)

## [1] "list"   "vector"

names(US)

## [1] "aupusData"          "inputData"          "ratioData"
## [4] "shareData"          "balanceElementData" "itemInfoData"
## [7] "populationData"     "extractionRateData"

sapply(US, class)

##      aupusData    inputData    ratioData    shareData    balanceElementData
## [1,] "data.table" "data.table" "data.table" "data.table" "data.table"
## [2,] "data.frame" "data.frame" "data.frame" "data.frame" "data.frame"
##      itemInfoData populationData extractionRateData
## [1,] "data.table" "data.table"   "data.table"
## [2,] "data.frame" "data.frame"   "data.frame"
```

```
sapply(US, dim)

##      aupusData inputData ratioData shareData balanceElementData itemInfoData
## [1,]        15        10        15        10                  15            3
## [2,]        75         6        37         5                   4            3
##      populationData extractionRateData
## [1,]              5                 15
## [2,]              6                  5

is(usAupusParam)

## [1] "list"   "vector"

names(usAupusParam)

## [1] "areaCode"    "itemCode"    "elementCode" "year"        "keyNames"
```

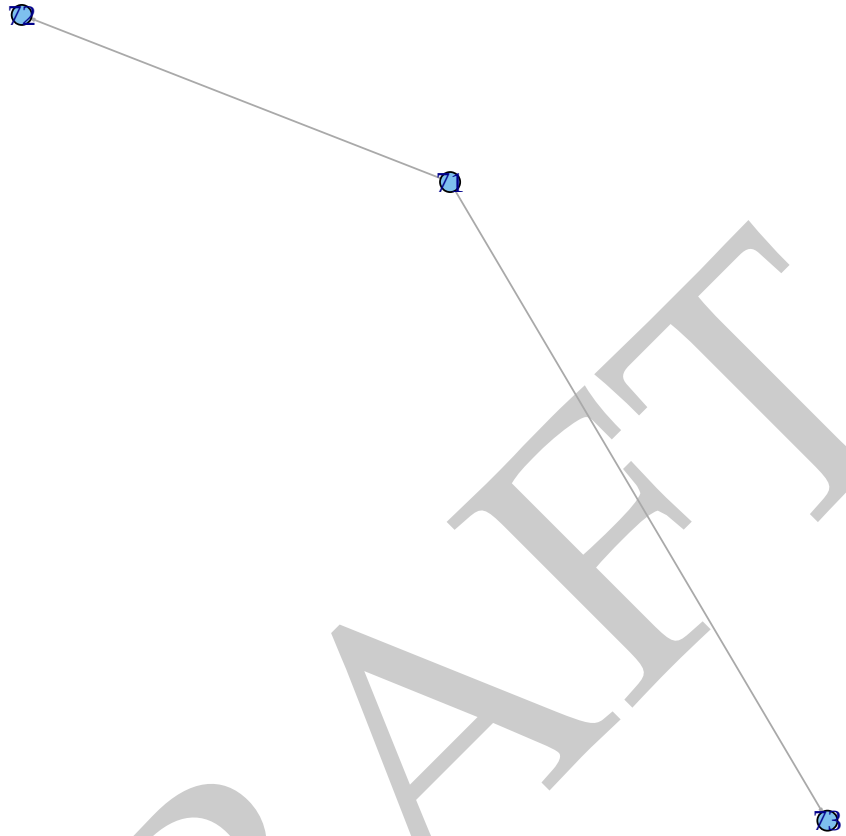
Note: usually these values would be generated in the following manner:

```
GetTestEnvironment(
    baseUrl = "https://hqlprswsas1.hq.un.fao.org:8181/sws",
    token = "66984d62-6add-4ad4-bdf3-5d8538bb2b70")
usAupusParam = getAupusParameter(areaCode = "231", assignGlobal = FALSE,
                                 yearsToUse = 2009:2013)
US = getAupusDataset(aupusParam = usAupusParam)
```

The GetTestEnvironment function from the faosws package sets up the right variables in R for querying the SWS database, and the specific token I used provides the necessary information for the AUPUS dataset. The parameters and data are then pulled from the SWS. However, for this vignette, we'll just use the data that already exists in this package.

We see that the US dataset contains 8 data.tables. Let's look at a subset of this data to more easily understand what we're working with. First, let's ensure we grab a meaningful subset of data. The plotCommodityTree function takes the shareData dataset and generates a plot for how the commodities are related to one another.

```
plotCommodityTree(US$shareData)
```



For simplicity, let's look at only commodities 71, 72, and 73.

```
US = subsetAupus(aupusData = US, itemKeys = c(71, 72, 73),
                 aupusParam = usAupusParam)
```

Now, we wish to work with this data in a network framework, and the package contains a few functions to generate that framework:

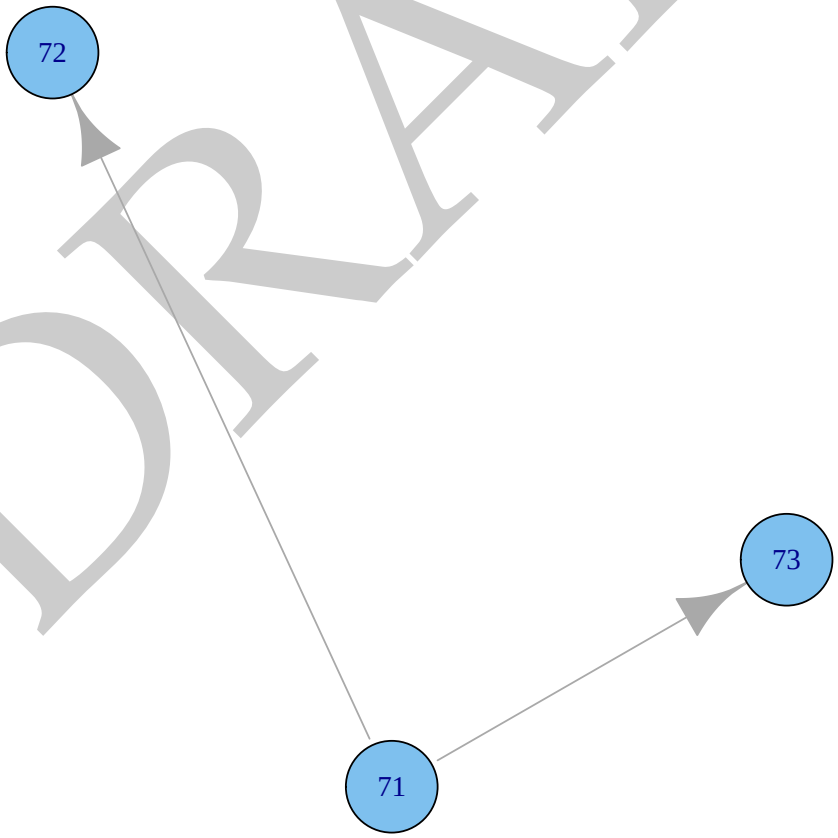
```
aupusNetwork = suaToNetworkRepresentation(dataList = US,
                                          aupusParam = usAupusParam)
names(aupusNetwork)

## [1] "nodes" "edges"

sapply(aupusNetwork, class)

##       nodes         edges
```

```
## [1,] "data.table" "data.table"
## [2,] "data.frame" "data.frame"

sapply(aupusNetwork, dim)

##      nodes edges
## [1,]    15    10
## [2,]   114     9
```

We see that now the data has been condensed down into two objects: nodes and edges. The nodes dataset has 15 rows (5 years times 3 commodities). There are only 10 rows in the edges dataset, as there are only 2 edges times 5 years. The nodes dataset is essentially the merged aupusData, itemInfoData, ratioData, balanceElementData, and populationData from US, while the edges dataset contains the datasets shareData, extractionRateData, and inputData from US. Here's our reduced network visualization:

```
plotCommodityTree(US$shareData, edge.arrow.size = 2, vertex.size = 25)
aupusNetwork$nodes[, .(geographicAreaFS, timePointYearsSP, measuredItemFS)]

##     geographicAreaFS timePointYearsSP measuredItemFS
##  1:              231             2009            71
##  2:              231             2010            71
##  3:              231             2011            71
##  4:              231             2012            71
##  5:              231             2013            71
##  6:              231             2009            72
##  7:              231             2010            72
##  8:              231             2011            72
##  9:              231             2012            72
## 10:              231             2013            72
## 11:              231             2009            73
## 12:              231             2010            73
## 13:              231             2011            73
## 14:              231             2012            73
## 15:              231             2013            73

colnames(aupusNetwork$nodes)[1:10]

##  [1] "geographicAreaFS"                "timePointYearsSP"
##  [3] "measuredItemFS"                  "Value_measuredElementFS_11"
##  [5] "flagFaostat_measuredElementFS_11" "Value_measuredElementFS_21"
##  [7] "flagFaostat_measuredElementFS_21" "Value_measuredElementFS_31"
##  [9] "flagFaostat_measuredElementFS_31" "Value_measuredElementFS_41"

aupusNetwork$edges[, .(geographicAreaFS, timePointYearsSP,
                       measuredItemParentFS, measuredItemChildFS)]

##     geographicAreaFS timePointYearsSP measuredItemParentFS measuredItemChildFS
##  1:              231             2009                  71                 72
##  2:              231             2010                  71                 72
```

```
##  3:              231             2011                 71                 72
##  4:              231             2012                 71                 72
##  5:              231             2013                 71                 72
##  6:              231             2009                 71                 73
##  7:              231             2010                 71                 73
##  8:              231             2011                 71                 73
##  9:              231             2012                 71                 73
## 10:              231             2013                 71                 73

colnames(aupusNetwork$edges)

## [1] "geographicAreaFS"       "measuredItemParentFS"   "measuredItemChildFS"
## [4] "timePointYearsSP"       "Value_share"            "Value_extraction"
## [7] "flagFaostat_extraction" "Value_input"            "flagFaostat_input"
```

2. AUPUS

The entire AUPUS procedure is encapsulated in one function: Aupus. However, let's look at some of the functions called within this main function to understand how the process works. First, we have to set up some of the things as done by the Aupus function:

```
nodes = aupusNetwork$nodes
edges = aupusNetwork$edges
nodes = coerceColumnTypes(aupusParam = usAupusParam, data = nodes)
edges = coerceColumnTypes(aupusParam = usAupusParam, data = edges)
from = usAupusParam$keyNames$itemParentName
to = usAupusParam$keyNames$itemChildName
processingLevelData = edges[, findProcessingLevel(.SD, from = from,
    to = to, aupusParam = usAupusParam),
    by = c(usAupusParam$keyNames$areaName, usAupusParam$keyNames$yearName)]
setkeyv(processingLevelData, key(nodes))
invisible(nodes[processingLevelData, `:=`(processingLevel, i.processingLevel)])
invisible(nodes[is.na(processingLevel), processingLevel := 0])
nodes[, c(key(nodes), "processingLevel"), with = FALSE]

##     geographicAreaFS measuredItemFS timePointYearsSP processingLevel
##  1:              231             71             2009               0
##  2:              231             71             2010               0
##  3:              231             71             2011               0
##  4:              231             71             2012               0
##  5:              231             71             2013               0
##  6:              231             72             2009               1
##  7:              231             72             2010               1
##  8:              231             72             2011               1
##  9:              231             72             2012               1
## 10:              231             72             2013               1
## 11:              231             73             2009               1
## 12:              231             73             2010               1
## 13:              231             73             2011               1
## 14:              231             73             2012               1
## 15:              231             73             2013               1
```

The processing level function above uses functions from the **igraph** package to determine the "processing level." This value indicates the order in which a particular node is processed: nodes at level 0 have no inputs/dependencies/children, and thus they can be processed immediately. Once level 0 nodes have been processed, we have the data necessary for processing their parents, and these are nodes with processing level 1. Aggregation continues until all processing levels have been processed.

Our example is simple: 72 and 73 are children of 71, and so 71 must first be processed before we process 72 and 73. Thus, 71 is processingLevel = 0 and 72 and 73 have processingLevel = 1.

At each level, there are three main processes that are performed:

1. The main AUPUS module is ran on each node. This module computes each individual element following the logic of the old system.

2. The edges of the graph are updated.

3. The inputs from processing are updated.

2.1. Main AUPUS Module

The main function here is calculateAupusElements. This function calls all of the individual element calculation functions. Each element function has it's own calculation, and is documented within it's help page. However, we'll show an example for a few of the functions. First, though, note that we must subset the AUPUS data: we only want to process the commodities at the lowest processing level right now:

```
toProcess = nodes[processingLevel == 0, ]
```

Ok, now let's compute element 11:

```
toProcess[, Value_measuredElementFS_11]

## [1] 12000 23000 20000 20000 20000

toProcess[, Value_measuredElementFS_161]

## [1] NA NA NA NA NA

toProcess[, flagFaostat_measuredElementFS_11]

## [1] ""  ""  ""  "T" "T"

calculateEle11(data = toProcess, aupusParam = usAupusParam)

## integer(0)

toProcess[, Value_measuredElementFS_11]

## [1] 12000 23000 20000 20000 20000

toProcess[, flagFaostat_measuredElementFS_11]

## [1] ""  ""  ""  "T" "T"
```

Element 11 is called "Initial Existence" and thus it is set to "Final Existence" (element 161) from the previous year, if that value exists and if element 11 is currently missing. In this case, all of element 161's values are missing and thus no updating occurs. Let's continue processing some elements:

```
calculateEle21(data = toProcess, aupusParam = usAupusParam)

## integer(0)

calculateEle41(data = toProcess, aupusParam = usAupusParam)

## integer(0)
```

```
calculateEle51(data = toProcess, aupusParam = usAupusParam)

## integer(0)

calculateEle314151(data = toProcess, aupusParam = usAupusParam)

## [[1]]
## integer(0)
##
## [[2]]
## integer(0)

calculateEle63(data = toProcess, aupusParam = usAupusParam)

## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## integer(0)

calculateEle71(data = toProcess, aupusParam = usAupusParam)

## [[1]]
## integer(0)
##
## [[2]]
## integer(0)

calculateEle93(data = toProcess, aupusParam = usAupusParam)

## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## integer(0)

calculateTotalSupply(data = toProcess, aupusParam = usAupusParam)
tail(colnames(toProcess))

## [1] "Value_balanceElement" "Value_population_11"  "Value_population_21"
## [4] "processingLevel"      "newSummation"         "TOTAL_SUPPLY"

toProcess$TOTAL_SUPPLY

## [1] 297711 309126 331715 347403 365818
```

Each of the calculateEleXX functions above calculates updated values for each element and returns a vector of list of vectors with indices. These indices represent the rows that were updated in the computation of this element. The last function, calculateTotalSupply, returns nothing but adds an additional column to the toProcess data.table. This column, TO-

TAL_SUPPLY, represents the total supply for this commodity. Now, we can proceed with computing other elements. Some elements, such as 101, 111, 121, 131, and 141 use total supply to fill in their values:

```
calculateEle101(stotal = "TOTAL_SUPPLY", data = toProcess,
                aupusParam = usAupusParam)

## integer(0)

calculateEle111(stotal = "TOTAL_SUPPLY", data = toProcess,
                aupusParam = usAupusParam)

## [[1]]
## integer(0)
##
## [[2]]
## integer(0)

calculateEle121(stotal = "TOTAL_SUPPLY", data = toProcess,
                aupusParam = usAupusParam)

## integer(0)

calculateEle131(stotal = "TOTAL_SUPPLY", data = toProcess,
                aupusParam = usAupusParam)

## integer(0)

calculateEle141(stotal = "TOTAL_SUPPLY", data = toProcess,
                aupusParam = usAupusParam)

## [1] 1 2 3 4 5

calculateEle144(population11Num = "Value_population_11",
                data = toProcess, aupusParam = usAupusParam)

## integer(0)

calculateEle151(stotal = "TOTAL_SUPPLY", data = toProcess,
                aupusParam = usAupusParam)

## integer(0)

calculateEle161(data = toProcess, aupusParam = usAupusParam)

## integer(0)

calculateEle171(data = toProcess, aupusParam = usAupusParam)

## integer(0)

calculateEle174(population11Num = "Value_population_11",
                                  data = toProcess,
                                  aupusParam = usAupusParam)

## integer(0)
```

Once we've computed element 141, we can compute nutritive values: calories, proteins, and fats. These are based on ratios provided in the database (to convert quantity into these values).

```
calculateTotalNutritive(ratioNum = "Ratio_measuredElementFS_261",
                            elementNum = 261, data = toProcess,
                            aupusParam = usAupusParam)

## integer(0)

calculateDailyNutritive(population11Num = "Value_population_11",
                        population21Num = "Value_population_21",
                        dailyElement = 264, totalElement = 261,
                        data = toProcess, aupusParam = usAupusParam)

## integer(0)

calculateTotalNutritive(ratioNum = "Ratio_measuredElementFS_271",
                            elementNum = 271, data = toProcess,
                            aupusParam = usAupusParam)

## integer(0)

calculateDailyNutritive(population11Num = "Value_population_11",
                        population21Num = "Value_population_21",
                        dailyElement = 274, totalElement = 271,
                        data = toProcess, aupusParam = usAupusParam)

## integer(0)

calculateTotalNutritive(ratioNum = "Ratio_measuredElementFS_281",
                            elementNum = 281, data = toProcess,
                            aupusParam = usAupusParam)

## integer(0)

calculateDailyNutritive(population11Num = "Value_population_11",
                        population21Num = "Value_population_21",
                        dailyElement = 284, totalElement = 281,
                        data = toProcess, aupusParam = usAupusParam)

## integer(0)
```

We now need to calculate two remaining elements (541 and 546, containing final and total demand) and then we can balance.

```
calculateEle541(data = toProcess, aupusParam = usAupusParam)

## integer(0)

calculateEle546(data = toProcess, aupusParam = usAupusParam)
```

```
## integer(0)

calculateTotalUtilization(data = toProcess, aupusParam = usAupusParam)
calculateBalance(supply = "TOTAL_SUPPLY", utilization = "TOTAL_UTILIZATION",
                 data = toProcess, aupusParam = usAupusParam)

## integer(0)

tail(colnames(toProcess))

## [1] "Value_measuredElementFS_542" "Value_measuredElementFS_543"
## [3] "Value_measuredElementFS_544" "Value_measuredElementFS_545"
## [5] "TOTAL_UTILIZATION"           "BALANCE"
```

Now, we have the TOTAL_SUPPLY, TOTAL_UTILIZATION, and BALANCE values.

2.2. Update Edges

The second step in the AUPUS procedure is to update the edges of the commodity network. In this step, we're updating the extraction rates and the input from processing values on the edges data.table with the values from the nodes table. Note: the 131 element represents the input from processing value, and the 41 element has extraction rates.

```
toProcess[, c("timePointYearsSP", "Value_measuredElementFS_131",
              "Value_measuredElementFS_41"), with = F]

##    timePointYearsSP Value_measuredElementFS_131 Value_measuredElementFS_41
## 1:             2009                       83800                    17418.12
## 2:             2010                       83800                    17601.64
## 3:             2011                       84000                    16408.66
## 4:             2012                       84000                    17574.18
## 5:             2013                       84000                    17314.47

edges[, c("timePointYearsSP", "measuredItemChildFS", "Value_share",
          "Value_input", "Value_extraction"), with = FALSE]

##     timePointYearsSP measuredItemChildFS Value_share Value_input
##  1:             2009                  72         100       83800
##  2:             2010                  72         100       83800
##  3:             2011                  72         100       84000
##  4:             2012                  72         100       84000
##  5:             2013                  72         100       84000
##  6:             2009                  73         100       83800
##  7:             2010                  73         100       83800
##  8:             2011                  73         100       84000
##  9:             2012                  73         100       84000
## 10:             2013                  73         100       84000
##     Value_extraction
##  1:             8000
##  2:             8000
##  3:             8000
```

```
##  4:             8000
##  5:             8000
##  6:             1600
##  7:             1600
##  8:             1600
##  9:             1600
## 10:             1600
```

```
updateEdges(nodes = toProcess, edges = edges, aupusParam = usAupusParam)
edges[, c("timePointYearsSP", "measuredItemChildFS", "Value_share",
          "Value_input", "Value_extraction"), with = FALSE]
```

```
##     timePointYearsSP measuredItemChildFS Value_share Value_input
##  1:             2009                  72         100       83800
##  2:             2010                  72         100       83800
##  3:             2011                  72         100       84000
##  4:             2012                  72         100       84000
##  5:             2013                  72         100       84000
##  6:             2009                  73         100       83800
##  7:             2010                  73         100       83800
##  8:             2011                  73         100       84000
##  9:             2012                  73         100       84000
## 10:             2013                  73         100       84000
##     Value_extraction
##  1:         17418.12
##  2:         17601.64
##  3:         16408.66
##  4:         17574.18
##  5:         17314.47
##  6:         17418.12
##  7:         17601.64
##  8:         16408.66
##  9:         17574.18
## 10:         17314.47
```

The Value_input column of edges is updated (in theory) to reflect the amount of the parent commodity flowing to the child. In this case, the original values for inputs from processing were already correct, and so nothing is changed with them. However, the extraction rates are different, and those values are changed on the edges table (updated to reflect the nodes table).

2.3. Update Inputs from Processing

In this step, the data from the edges data.table is passed to the nodes data.table at the next processing level.

```
nodesNextLevel = nodes[processingLevel == 1, ]
nodesNextLevel[, c("timePointYearsSP", "measuredItemFS",
                   "Value_measuredElementFS_31"), with = FALSE]
```

```
##     timePointYearsSP measuredItemFS Value_measuredElementFS_31
##  1:             2009            72                       83800
```

```
##  2:              2010              72                         83800
##  3:              2011              72                         84000
##  4:              2012              72                         84000
##  5:              2013              72                         84000
##  6:              2009              73                         83800
##  7:              2010              73                         83800
##  8:              2011              73                         84000
##  9:              2012              73                         84000
## 10:              2013              73                         84000
```

```
updateInputFromProcessing(nodes = nodesNextLevel,
                          edges = edges,
                          aupusParam = usAupusParam)
nodesNextLevel[, c("timePointYearsSP", "measuredItemFS",
                   "Value_measuredElementFS_31"), with = FALSE]
```

```
##     timePointYearsSP measuredItemFS Value_measuredElementFS_31
##  1:              2009              72                         83800
##  2:              2010              72                         83800
##  3:              2011              72                         84000
##  4:              2012              72                         84000
##  5:              2013              72                         84000
##  6:              2009              73                         83800
##  7:              2010              73                         83800
##  8:              2011              73                         84000
##  9:              2012              73                         84000
## 10:              2013              73                         84000
```

This entire section (i.e. the whole AUPUS procedure) can be run by calling the function Aupus. This function will also compute the processing levels and iterate through each of them.

3. Standardization

Standardization refers to the process of aggregating multiple commodities up to one representative commodity. For example, wheat can be reported directly, or derivatives of wheat, such as wheat flour, can also be reported. To get a simpler view of the food balance sheet, we can "standardize" commodities up to their parent commodities. This allows us to reduce the size of the food balance sheet and make it easier to understand/analyze, but still retains all of the food information available.

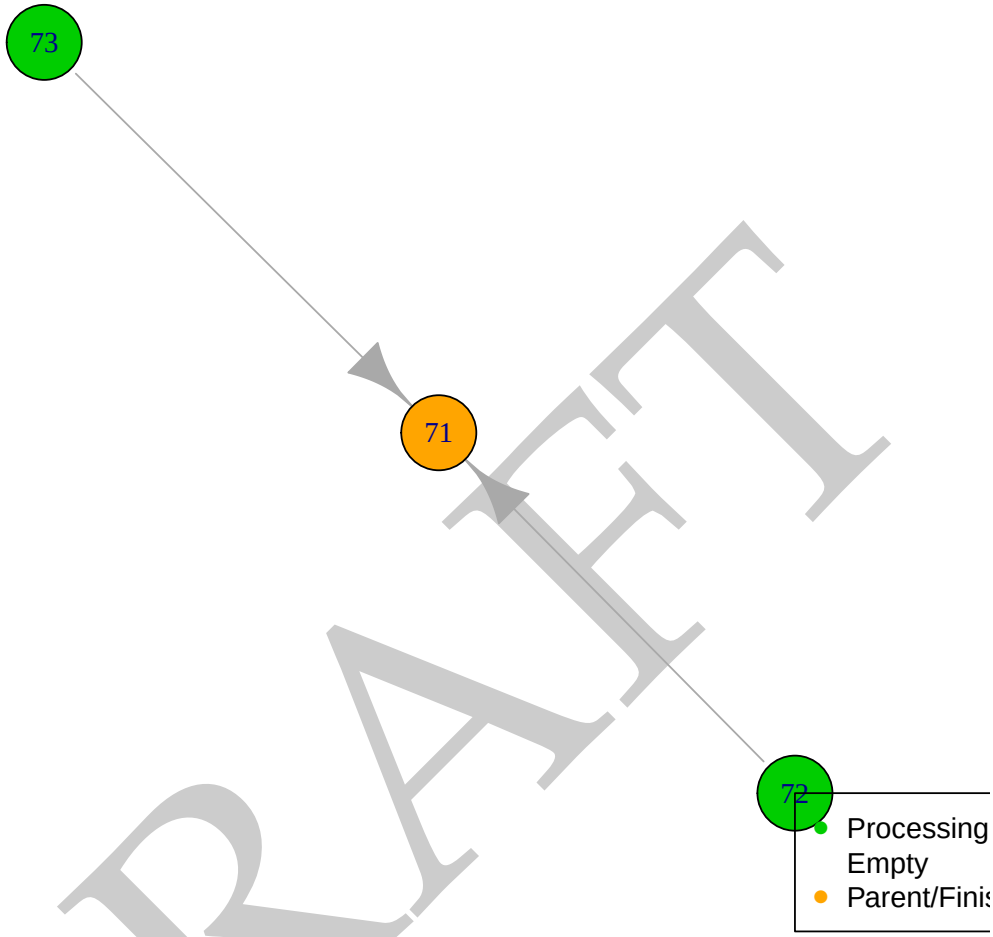
Now, from the previous sections, we have an object called "updatedAupusNetwork" which contains a data.table called "nodes" and a data.table called "edges." We can convert this object into a graph object using the constructStandardizationGraph function. We also pass a character vector containing the names of the FBS element variables we're interested in.

```
FBSelements =
    c("Value_measuredElementFS_51", "Value_measuredElementFS_61",
      "Value_measuredElementFS_91", "Value_measuredElementFS_101",
      "Value_measuredElementFS_111", "Value_measuredElementFS_121",
      "Value_measuredElementFS_141", "Value_measuredElementFS_151")
standardizationGraph =
```

```
    constructStandardizationGraph(aupusNetwork = aupusNetwork,
                                  standardizeElement = FBSelements,
                                  aupusParam = usAupusParam)
is(standardizationGraph)

## [1] "list"   "vector"

sapply(standardizationGraph, class)

##     2009     2010     2011     2012     2013
## "igraph" "igraph" "igraph" "igraph" "igraph"
```


Let's now begin the standardization procedure:

```
standardization(standardizationGraph[[1]], standardizeElement = FBSelements,
                plot = TRUE, aupusParam = usAupusParam,
                vertex.size = 20, edge.arrow.size = 2, vertex.label.cex = 1)

## Continue processing?
##    measuredItemFS Value_measuredElementFS_51 Value_measuredElementFS_61
## 1:             71                   223816.4                   121263.7
##    Value_measuredElementFS_91 Value_measuredElementFS_101
## 1:                   5618.022                    44497.73
##    Value_measuredElementFS_111 Value_measuredElementFS_121
## 1:                       76200                           0
##    Value_measuredElementFS_141 Value_measuredElementFS_151
## 1:                    38048.31                           0
```



To understand more clearly what happened, let's examine element 51 (which represents the quantity of produced goods).

```
vertex.attributes(standardizationGraph[[1]])[1:2]

## $name
## [1] "71" "72" "73"
##
## $Value_measuredElementFS_51
## [1] 177630  67040  13408

edge.attributes(standardizationGraph[[1]])[c(3, 4)]

## $Value_share
## [1] 100 100
##
## $Value_extraction
## [1] 17418.12 17418.12
```

```
E(standardizationGraph[[1]])

## Edge sequence:
##
## [1] 72 -> 71
## [2] 73 -> 71
```

So, items 72 and 73 will be standardized to item 71 (based on the edges). The current value for element 51, item 71 is 177,630. All 67,040 units from item 72 can be standardized/converted to item 71. This is because the Value_share is 100, implying 100% of item 72 came from 71 (and not other commodities as well). The extraction rate values presented are out of a base of 10,000, so for 71 to 72 the actual extraction rate is 8,000/10,000 = 80%. Thus, we can manually calculate the value for element 51, item 71:

```
177630 +                          # The initial value
    67040 * 10000/8000 * 100/100 + # Standardizing element 72
    13408 * 10000/1600 * 100/100   # Standardizing element 73

## [1] 345230
```

This value matches what we computed using the standardization function above. The standardization function, however, performs the above operations for each commodity and element type, and it performs the procedure multiple times (if necessary) to standardize multiple levels back to one main commodity. The fbsStandardization function performs this standardization for each graph and returns the result as a data.table.

```
fbsStandardization(graph = standardizationGraph,
                   standardizeElement = FBSelements,
                   plot = FALSE, aupusParam = usAupusParam)

##    measuredItemFS Value_measuredElementFS_51 Value_measuredElementFS_61
## 1:             71                   223816.4                   121263.7
## 2:             71                   234464.8                   121621.6
## 3:             71                   209834.8                   172235.9
## 4:             71                   222263.5                   172155.6
## 5:             71                   241366.8                   172172.6
##    Value_measuredElementFS_91 Value_measuredElementFS_101
## 1:                   5618.022                    44497.73
## 2:                   6664.610                   100687.47
## 3:                   4754.304                    78983.80
## 4:                   4706.732                    85647.58
## 5:                   4716.778                    93762.29
##    Value_measuredElementFS_111 Value_measuredElementFS_121
## 1:                        76200                           0
## 2:                        76200                           0
## 3:                        76200                           0
## 4:                        85000                           0
## 5:                        86000                           0
##    Value_measuredElementFS_141 Value_measuredElementFS_151 timePointYearsSP
## 1:                     38048.31                           0             2009
## 2:                     36619.31                           0             2010
```

```
## 3:                41447.63                0                2011
## 4:                38698.82                0                2012
## 5:                39279.28                0                2013
```

Affiliation:

Joshua M. Browning and Michael. C. J. Kao
Economics and Social Statistics Division (ESS)
Economic and Social Development Department (ES)
Food and Agriculture Organization of the United Nations (FAO)
Viale delle Terme di Caracalla 00153 Rome, Italy
E-mail: joshua.browning@fao.org, michael.kao@fao.org
URL: https://svn.fao.org/projects/SWS/RModules/faoswsAupus/