Working with Two-Mode Networks in "multiplex"

Antonio Rivero Ostoic
CESU, University of San Simón

January 29, 2025

Social networks are configurations defined by a collection of relationships among collective actors. In terms of set theory, a relation is an ordered pair such as (x, y) that refers to a directed linkage from an element x to an element y, where $x \in X$ and $y \in Y$ called the "domain" and "codomain" of the relation. The context of a binary relation R is the overall relation set that results from the Cartesian product of the domain and codomain or $X \times Y$ of all ordered pairs (x, y) where R is a subset of the context.

Usually, a social network refers to a domain with a set of relations on such domain, which is the generic term used to name the social entities in the system, and in such a case the system of relations is said to be a one-mode network. However, when the domain and the codomain in the system are not equal, there are two sets of entities that describe the entire social configuration, which is known as affiliation, bipartite, or else as a two-mode network.

1 Galois representation

In terms of Formal Concept Analysis, the domain and codomain of a two-mode network are characterized respectively as a set of objects G, and a set of attributes M. A formal context is obtained with an incident relation $I \subseteq G \times M$ between these sets, and this triple is typically represented as a data table.

```
> # fruits data set with attributes
> frt <- data.frame(yellow = c(0,1,0,0,1,0,0,0), green = c(0,0,1,0,0,0,0,1), red = c(1,0,0,1,0,0,0,0),
+                    orange = c(0,0,0,0,0,1,1,0), apple = c(1,1,1,1,0,0,0,0), citrus = c(0,0,0,0,1,1,1,1))
> # fruit varieties as objects
> rownames(frt) <- c("PinkLady","GrannySmith","GoldenDelicious","RedDelicious","Lemon","Orange","Mandarin","Lime")
> frt
```

	yellow	green	red	orange	apple	citrus
PinkLady	0	0	1	0	1	0
GrannySmith	1	0	0	0	1	0
GoldenDelicious	0	1	0	0	1	0
RedDelicious	0	0	1	0	1	0
Lemon	1	0	0	0	0	1
Orange	0	0	0	1	0	1
Mandarin	0	0	0	1	0	1
Lime	0	1	0	0	0	1

Galois derivations

The formal concept of a formal context is a pair of sets of objects A and attributes B that is maximally contained on each other. A *Galois derivation* between G and M is defined for any subsets $A \subseteq G$ and $B \subseteq M$ by

$$A' = m \in M \mid (g, m) \in I \quad \text{(for all } g \in A\text{)}$$
$$B' = g \in G \mid (g, m) \in I \quad \text{(for all } m \in B\text{)}$$

where A and B are said to be the extent and intent of the formal concept respectively, whereas A' is the set of attributes common to all the objects in the intent and B' the set of objects possessing the attributes in the extent.

With package **multiplex**, it is possible to perform an algebraic analysis of two-mode networks with the function `galois()` to produce Galois derivations between objects and attributes. This command creates an adjunction between the two sets partially ordered by inclusion, and we obtain the complete list of concepts of the context, which can be assigned into an object with the class named "`Galois`" and "`full`."

```
> # load package
> library("multiplex")

> # Galois derivations between fruits and their attributes
> galois(frt)


$sep
[1] ", "

$gc
$yellow
[1] "GrannySmith, Lemon"

$green
[1] "GoldenDelicious, Lime"

$`apple, red`
[1] "PinkLady, RedDelicious"

$`citrus, orange`
[1] "Mandarin, Orange"

$apple
[1] "GoldenDelicious, GrannySmith, PinkLady, RedDelicious"

$citrus
[1] "Lemon, Lime, Mandarin, Orange"

$`apple, citrus, green, orange, red, yellow`
character(0)

$`apple, yellow`
[1] "GrannySmith"

$`citrus, yellow`
[1] "Lemon"

$`apple, green`
[1] "GoldenDelicious"
```

```
$`citrus, green`
[1] "Lime"


[[12]]
[1] "GoldenDelicious, GrannySmith, Lemon, Lime, Mandarin, Orange, PinkLady, RedDelicious"


attr(,"class")
[1] "Galois" "full"
```

Reduced labeling of concepts

It is also possible to condense the labeling of the objects and attributes in the Galois derivations with
option `"reduced"` as argument in `labeling` of `galois()` function. Below is another Galois representation
of the fruits dataset with two domains by using the R native pipe.

```
> # Galois derivation with a reduced labeling
> frt |> galois(labeling = "reduced")


$sep
[1] ", "

$gc
$reduc
$reduc$yellow
character(0)

$reduc$green
character(0)

$reduc$red
[1] "PinkLady, RedDelicious"

$reduc$orange
[1] "Mandarin, Orange"

$reduc$apple
character(0)

$reduc$citrus
character(0)

$reduc[[7]]
character(0)

$reduc[[8]]
[1] "GrannySmith"

$reduc[[9]]
[1] "Lemon"

$reduc[[10]]
[1] "GoldenDelicious"

$reduc[[11]]
[1] "Lime"

$reduc[[12]]
character(0)
```

The class object of the above representation is hidden, and this is since the full labeling is useful for the construction of the hierarchy of concepts. That is the reason why the whole structure of the 12 formal concepts given by the Galois derivation is kept in the output.

```
> # structure of Galois derivation with full labeling
> frt |>
+   galois(labeling = "reduced") |>
+   getElement("gc") |>
+   getElement("full") |>
+   str()


List of 12
 $ yellow                               : chr "GrannySmith, Lemon"
 $ green                                : chr "GoldenDelicious, Lime"
 $ apple, red                           : chr "PinkLady, RedDelicious"
 $ citrus, orange                       : chr "Mandarin, Orange"
 $ apple                                : chr "GoldenDelicious, GrannySmith, PinkLady, RedDelicious"
 $ citrus                               : chr "Lemon, Lime, Mandarin, Orange"
 $ apple, citrus, green, orange, red, yellow: chr(0)
 $ apple, yellow                        : chr "GrannySmith"
 $ citrus, yellow                       : chr "Lemon"
 $ apple, green                         : chr "GoldenDelicious"
 $ citrus, green                        : chr "Lime"
 $                                      : chr "GoldenDelicious, GrannySmith, Lemon, Lime, Mandarin, Orange, PinkLady, RedDelicious"
 - attr(*, "class")= chr [1:2] "Galois" "full"
```

Partial ordering of formal concepts

A hierarchy of the concepts is given by the relation subconcept–superconcept

$$(A, B) \leq (A_2, B_2) \quad \Leftrightarrow \quad A_1 \subseteq A_2 \qquad (\Leftrightarrow \quad B_1 \subseteq B_2)$$

For an ordering of concepts in a formal context, function `partial.order()` has the `"galois"` option in the `type` argument where the hierarchy of the concepts is constructed. In this case, even though the concepts have the "reduced" option, it is the "full" labeling of the formal concepts that is the base of the ordering among these concepts that can be designated in different ways.

```
> # partial ordering of the formal concepts with established labels
> pogdc <- frt |>
+   galois(labeling = "reduced") |>
+   partial.order(type = "galois", lbs = paste("c", seq_len(12), sep = ""))
> pogdc


    c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12
c1   1  0  0  0  0  0  0  0  0   0   0   1
c2   0  1  0  0  0  0  0  0  0   0   0   1
c3   0  0  1  0  1  0  0  0  0   0   0   1
c4   0  0  0  1  0  1  0  0  0   0   0   1
c5   0  0  0  0  1  0  0  0  0   0   0   1
c6   0  0  0  0  0  1  0  0  0   0   0   1
c7   1  1  1  1  1  1  1  1  1   1   1   1
c8   1  0  0  0  1  0  0  1  0   0   0   1
c9   1  0  0  0  0  1  0  0  1   0   0   1
c10  0  1  0  0  1  0  0  0  0   1   0   1
c11  0  1  0  0  0  1  0  0  0   0   1   1
c12  0  0  0  0  0  0  0  0  0   0   0   1
attr(,"class")
[1] "Partial.Order" "galois"         ", "
```

Object `pogdc` records the partial order of the concepts with a a customized labeling of the concepts. The partial order table shows that all concepts are included in concept 12, whereas concept 7 is included in the rest of the concepts. As a result, these concepts constitute the maxima and the minima elements of a complete lattice that provides for all the formal concepts of the context. From the outputs given with the Galois derivation of this context, these concepts correspond to the set of objects and the set of attributes, which are entirely abridged in the reduced formal context.

Concept diagram of the context

The concept diagram of the formal context is a system of concepts partially ordered where the greatest lower bound of the meet and the least upper bound of the join are defined as

$$\bigwedge_{t \in T} \ (A_t, B_t) \ = \ \left(\bigcap_{t \in T} A_t, \ \left(\bigcup_{t \in T} B_t \right)'' \right)$$
$$\bigvee_{t \in T} \ (A_t, B_t) \ = \ \left(\left(\bigcup_{t \in T} A_t \right)'', \ \bigcap_{t \in T} B_t \right)$$

To plot this type of concept or lattice diagram with the labeling corresponding to the reduced context.

```
> # plot concept lattice diagram
> suppressPackageStartupMessages(require("Rgraphviz", quietly = TRUE))
> frt |> galois(labeling = "reduced") |>
+   partial.order(type = "galois") |>
+   diagram(type = "concept", main = "Fruits & Colors", fsize = 17, fcol = "red", col.main = "blue")
```
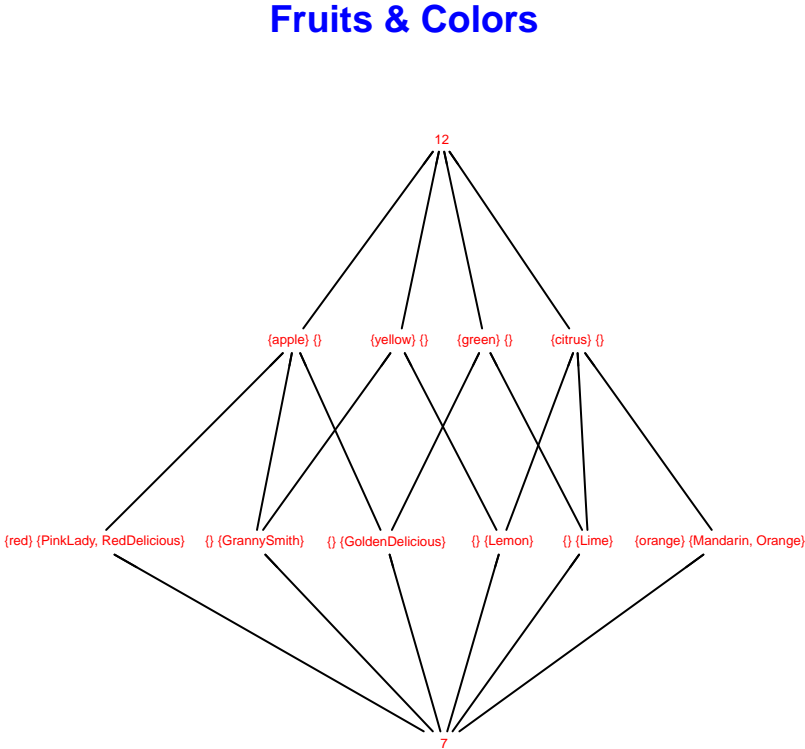

Figure 1: Concept Lattice of fruits and color characteristics

Since this is a reduced representation of the context, both objects and attributes are only given just once. Besides, labels are placed instead of the nodes rather than next to them as the typical representation of formal context. Moreover, in case that a concept does not have a label, which happens in most reduced contexts, then the number of the concept is placed as the node.

2 Diagram levels & Order Filters

The construction of the concept diagram of the context allows us to have additional information about the network relational structure. One part is concerned with the inclusion levels in the lattice structure, and another aspect deals with downsets and upsets, which are formed from all the lower and greater bounds of an element in the lattice diagram. Next, we take a brief look at the suitable functions to get such information.

Levels in the lattice diagram

Mainly when dealing with large diagrams, it can be difficult to distinguish the different heights in the lattice and the elements belonging to each level. Function `diagram.levels()` allows us to count with such information, and we illustrate this routine with the entry `pogdc` that represents the partial order of the concepts corresponding to the fruits data set.

```
> # diagram levels of partial order
> require("Rgraphviz", quietly = TRUE, warn.conflicts = FALSE)
> frt |>
+    galois(labeling = "reduced") |>
+    partial.order(type = "galois") |>
+    diagram.levels()


$`2`
[1] "{yellow} {}" "{green} {}"  "{apple} {}"  "{citrus} {}"

$`3`
[1] "{red} {PinkLady, RedDelicious}" "{orange} {Mandarin, Orange}"    "{} {GrannySmith}"
[4] "{} {Lemon}"                     "{} {GoldenDelicious}"           "{} {Lime}"

$`4`
[1] "7"

$`1`
[1] "12"
```

Hence, concepts 7 and 12 make a class of their own, whereas the rest of the concepts belong either to class 2 or to class 3.

By setting `perm` to `TRUE`, we obtain the different classes in the lattice structure in a convenient way, and also a permuted partial order table according to the clustering. Recall that `pogdc` used in function `diagram.levels()` records the formal concepts with a customized labeling.

```
> # diagram levels with permutation
> if( require("Rgraphviz", quietly = TRUE, warn.conflicts = FALSE)) {
+ diagram.levels(pogdc, perm = TRUE) }
```

```
$cls
   2  2  3  3  2  2  4  3  3   3   3   1
 1 c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12

$clu
 [1] 2 2 3 3 2 2 4 3 3 3 3 1

$perm
    c12 c1 c2 c5 c6 c3 c4 c8 c9 c10 c11 c7
c12   1  0  0  0  0  0  0  0  0   0   0  0
c1    1  1  0  0  0  0  0  0  0   0   0  0
c2    1  0  1  0  0  0  0  0  0   0   0  0
c5    1  0  0  1  0  0  0  0  0   0   0  0
c6    1  0  0  0  1  0  0  0  0   0   0  0
c3    1  0  0  1  0  1  0  0  0   0   0  0
c4    1  0  0  0  1  0  1  0  0   0   0  0
c8    1  1  0  1  0  0  0  1  0   0   0  0
c9    1  1  0  0  1  0  0  0  1   0   0  0
c10   1  0  1  1  0  0  0  0  0   1   0  0
c11   1  0  1  0  1  0  0  0  0   0   1  0
c7    1  1  1  1  1  1  1  1  1   1   1  1
```

Order Filters and Order Ideals

Implications among objects and attributes in an arbitrary partially ordered set representing context are revealed by subsets in the order structure.

Let $(P, \leq)$ be an ordered set, and a, b are elements in P.

A non-empty subset U [resp. D] of P is an upset [resp. downset] called a *order filter* [resp. *order ideal*] if, for all $a \in P$ and $b \in U$ [resp. D]

$$b \leq a \quad \text{implies} \quad a \in U \qquad [\text{ resp. } a \leq b \quad \text{implies} \quad a \in D]$$

For a particular element $x \in P$, the upset $\uparrow x$ formed for all the upper bounds of x is called a *principal order filter* generated by x. Dually, $\downarrow x$ is a *principal order ideal* with all the lower bounds of x. Order filters and order ideals not coinciding with P are called *proper*.

To illustrate these concepts, we apply the function `fltr()` to the third element of the partial order represented by `pogd` that results in a proper principal order filter for this formal concept with labels.

```
> # first assign the partial order of the reduced context
> pogd <- frt |> galois(labeling = "reduced") |>
+   partial.order(type = "galois")


> # principal order filter of the third concept
> fltr(3, pogd)


$`3`
[1] "{red} {PinkLady, RedDelicious}"

$`5`
[1] "{apple} {}"

$`12`
[1] "12"
```

We get the same result when introducing the one or more of the names of this concept.

```
> # principal order filter of the concept with these labels
> fltr(c("red", "RedDelicious"), pogd)


$`3`
[1] "{red} {PinkLady, RedDelicious}"

$`5`
[1] "{apple} {}"

$`12`
[1] "12"
```

Or alternatively we combine elements from different concepts to obtain other types of order filters in the concept diagram of the context.

```
> # order filter of two concepts
> fltr(c("Lemon", "Lime"), pogd)


$`9`
[1] "{} {Lemon}"

$`11`
[1] "{} {Lime}"

$`1`
[1] "{yellow} {}"

$`2`
[1] "{green} {}"

$`6`
[1] "{citrus} {}"

$`12`
[1] "12"
```

Order ideals and principal order ideals are obtained similarly with this function if the argument `ideal` is set to `TRUE`.

```
> # order ideal of two concepts
> fltr(c(9, 11), pogd, ideal = TRUE)


$`9`
[1] "{} {Lemon}"

$`11`
[1] "{} {Lime}"

$`7`
[1] "7"
```

3 Bipartite graphs

Two-mode network are depicted through *bipartite graphs*, where the entities in one set only can relate to the elements placed in the other set. To plot network structure, **multiplex** has a reverse dependence on package **multigraph** for the visualization of multiple networks, and also for bipartite graphs representing affiliation or two-mode networks. Hence, function `bmgraph()` serves to plot the graph of the `frt` dataset with the default layout option that is `"bip"`.

```
> # load "multigraph" package and plot bipartite graph
> suppressPackageStartupMessages(library("multigraph", quietly = TRUE))
> multigraph::bmgraph(frt, pch = 16:15, lwd = 2, fsize = 6)
```
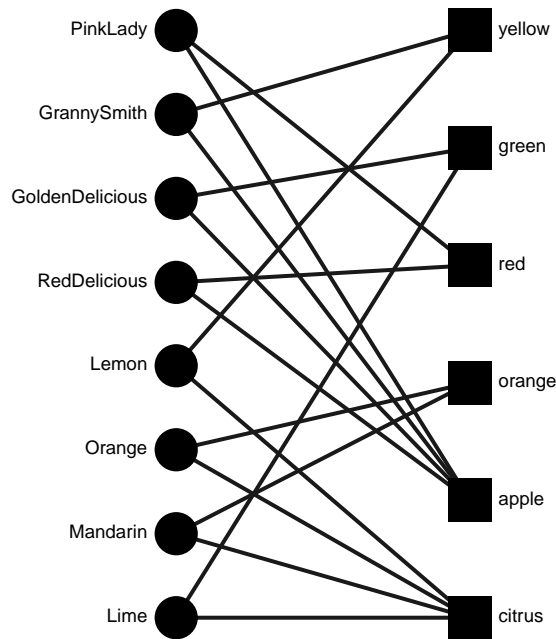

Figure 2: Bipartite graph of fruits with attributes

Bipartite graph with a binomial projection

Another possibility to depict a two-mode network as a "classic" bipartite graph is to apply a force-directed layout to the binomial projection of this two-mode data set. The plot below is made with arguments in function `bmgraph()` for changing the aspect of the vertices and to make the graph clock-wise rotated.

```
> # plot proyection of bipartite graph
> require(multigraph)
> bmgraph(frt, layout = "force", seed = 1, cex = 3, fsize = 7, vcol = 8, pch = 16:15, lwd = 2, rot = 15)
```
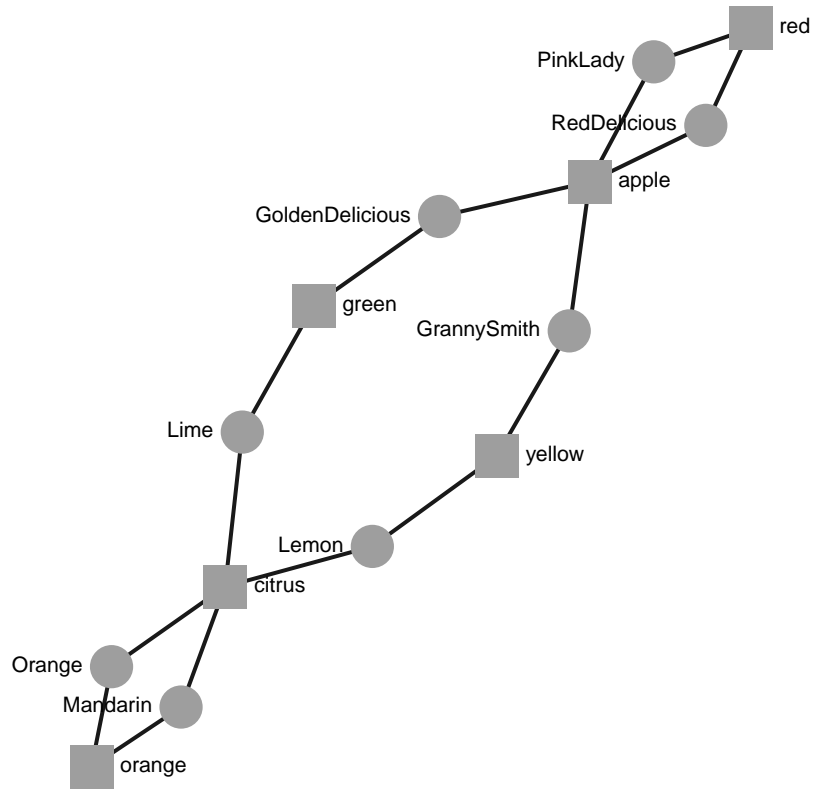
Figure 3: Bipartite graph of fruit attributes with a force-directed layout

References

[1] Ganter, B and Wille R *Formal Concept Analysis – Mathematical Foundations*. Springer. 1996.

[2] Hansen KD, Gentry J, Long L, Gentleman R, Falcon S, Hahne F and Sarkar D **Rgraphviz**: *Provides plotting capabilities for R graph objects*. R package v 2.24.0

[3] Ostoic, JAR **multiplex**: *Algebraic Tools for the Analysis of Multiple Social Networks*. R package v 3.8-3

[4] Ostoic, JAR **multigraph**: *Plot and Manipulate Multigraphs*. R package v 0.99-4