Literary Data: Some Approaches

Andrew Goldstone
http://www.rci.rutgers.edu/~ag978/litdata

April 23, 2015. Network basics.

```r
# from data("flo", package="network")
flo <- read.csv("network-intro/padgett-florence.csv",
                row.names=1)
flo <- as.matrix(flo)   # pretty much the same
```

- Which Florentine clans are linked by marriage?
- Guadagni is linked to Albizzi, Bischeri, Lamberteschi…
- Medici is linked to Acciaiuoli, Albizzi, Barbadori, …

formally

A (simple) graph G is a set V (the vertices) together with a set E of two-element subsets of V (the edges).

example: Florentine marriages

V {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}

E {{1,9}, {2,6}, {2,7}, {2,9}, {3,5}, {3,9}, {4,7}, {4,11}, {4,15}, {5,11}, {5,15}, {7,8}, {7,16}, {9,13}, {9,14}, {9,16}, {10,14}, {11,15}, {13,15}, {13,16}}

adjacency matrix

- number families in alphabetical order
- we can represent these ties with a matrix `flo`
- `flo[i, j] == 1` iff families `i` and `j` intermarried

```
             Acciaiuoli Albizzi Barbadori
Guadagni              0       1         0
Lamberteschi          0       0         0
Medici                1       1         1
```

- how many edges?

adjacency matrix

- number families in alphabetical order
- we can represent these ties with a matrix `flo`
- `flo[i, j] == 1` iff families `i` and `j` intermarried

```
              Acciaiuoli Albizzi Barbadori
Guadagni               0       1         0
Lamberteschi           0       0         0
Medici                 1       1         1
```

- how many edges?

```r
sum(flo) / 2  # why over 2?
```

```
[1] 20
```

`t(A)` transpose of a matrix A ($A^T_{ij} = A_{ji}$)

- What do we know about `t(flo)`?

(parenthesis on directed graphs)

A directed graph G is a set V together with a set E of ordered pairs of distinct elements of V. In general the corresponding adjacency matrix is not symmetric ($A_{ij} \neq A_{ji}$).

- models asymmetric relations (i.e. most of them)

degree

- how many families is a given family connected to?

```
rowSums(flo)
```

```
  Acciaiuoli      Albizzi    Barbadori
           1            3            2
    Bischeri   Castellani       Ginori
           3            3            1
    Guadagni Lamberteschi       Medici
           4            1            6
       Pazzi      Peruzzi        Pucci
           1            3            0
     Ridolfi     Salviati      Strozzi
           3            2            4
  Tornabuoni
           3
```

degree distribution

degree distribution

```
table(rowSums(flo))
```

```
0 1 2 3 4 6
1 4 2 6 2 1
```

a data structure

```
library("igraph")
```

```
flg <- graph.adjacency(flo, mode="undirected")
class(flg)
```

```
[1] "igraph"
```

```r
V(flg)
```

```
Vertex sequence:
 [1] "Acciaiuoli"   "Albizzi"
 [3] "Barbadori"    "Bischeri"
 [5] "Castellani"   "Ginori"
 [7] "Guadagni"     "Lamberteschi"
 [9] "Medici"       "Pazzi"
[11] "Peruzzi"      "Pucci"
[13] "Ridolfi"      "Salviati"
[15] "Strozzi"      "Tornabuoni"
```

```r
vcount(flg)
```

```
[1] 16
```

```r
degree(flg)
```

```
  Acciaiuoli      Albizzi    Barbadori
           1            3            2
    Bischeri   Castellani       Ginori
           3            3            1
    Guadagni Lamberteschi       Medici
           4            1            6
       Pazzi      Peruzzi        Pucci
           1            3            0
     Ridolfi     Salviati      Strozzi
           3            2            4
  Tornabuoni
           3
```

```r
ecount(flg)
```

```
[1] 20
```

```r
E(flg)[1:5]   # and 15 more....
```

```
Edge sequence:

[1] Medici     -- Acciaiuoli
[2] Ginori     -- Albizzi
[3] Guadagni   -- Albizzi
[4] Medici     -- Albizzi
[5] Castellani -- Barbadori
```

alternate representations: edge list

```r
flg_edges <- get.edgelist(flg)
head(flg_edges)
```

```
     [,1]         [,2]
[1,] "Acciaiuoli" "Medici"
[2,] "Albizzi"    "Ginori"
[3,] "Albizzi"    "Guadagni"
[4,] "Albizzi"    "Medici"
[5,] "Barbadori"  "Castellani"
[6,] "Barbadori"  "Medici"
```

alternate representations: adjacency list

```r
flg_adjlist <- get.adjlist(flg)
head(flg_adjlist)
```

```
$Acciaiuoli
[1] 9

$Albizzi
[1] 6 7 9

$Barbadori
[1] 5 9

$Bischeri
[1]  7 11 15

$Castellani
[1]  3 11 15

$Ginori
[1] 2
```

aside on side-effect graphics

- base R graphical parameters:

```
old_par <- par()
par(bg="gray10", fg="white")
```

- `igraph` plotting:

```
old_par_igraph <- igraph.options()
igraph.options(vertex.label.color="white", vertex.color=NA,
               vertex.frame.color="white")
```

```
plot(flg)   # easy---a little *too* easy
```

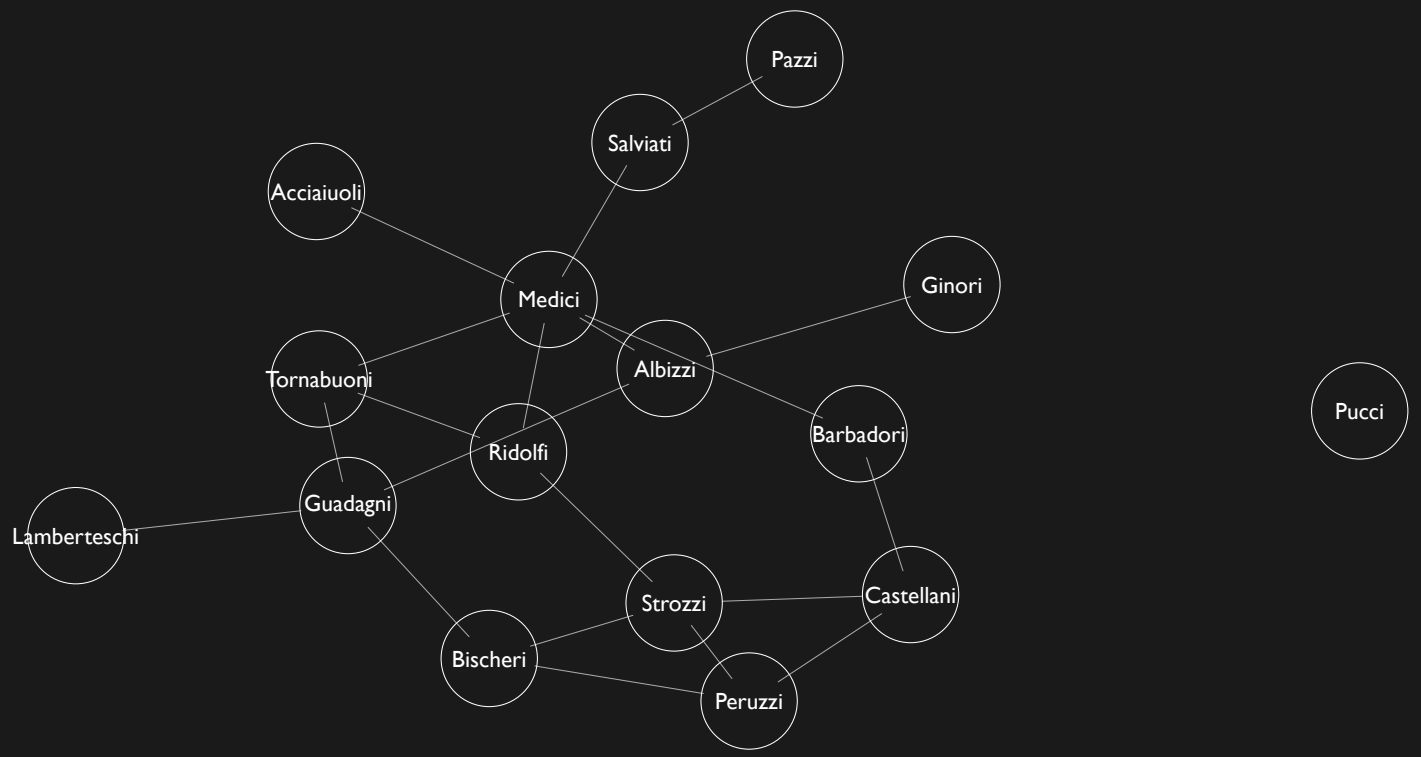


Figure 1: Our first network visualization

```
plot(flg)
```

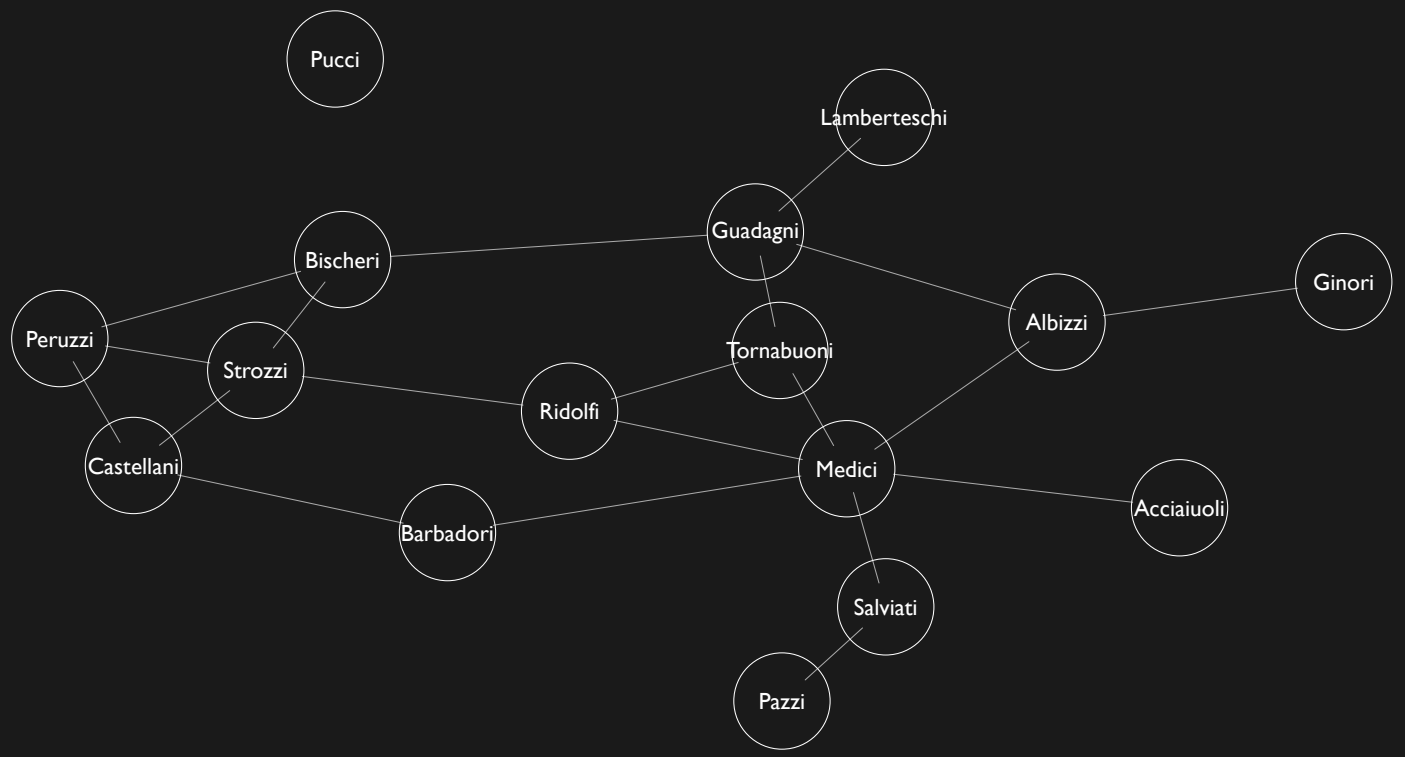


Figure 2: The same, again?

```
plot(flg, edge.curved=T)
```

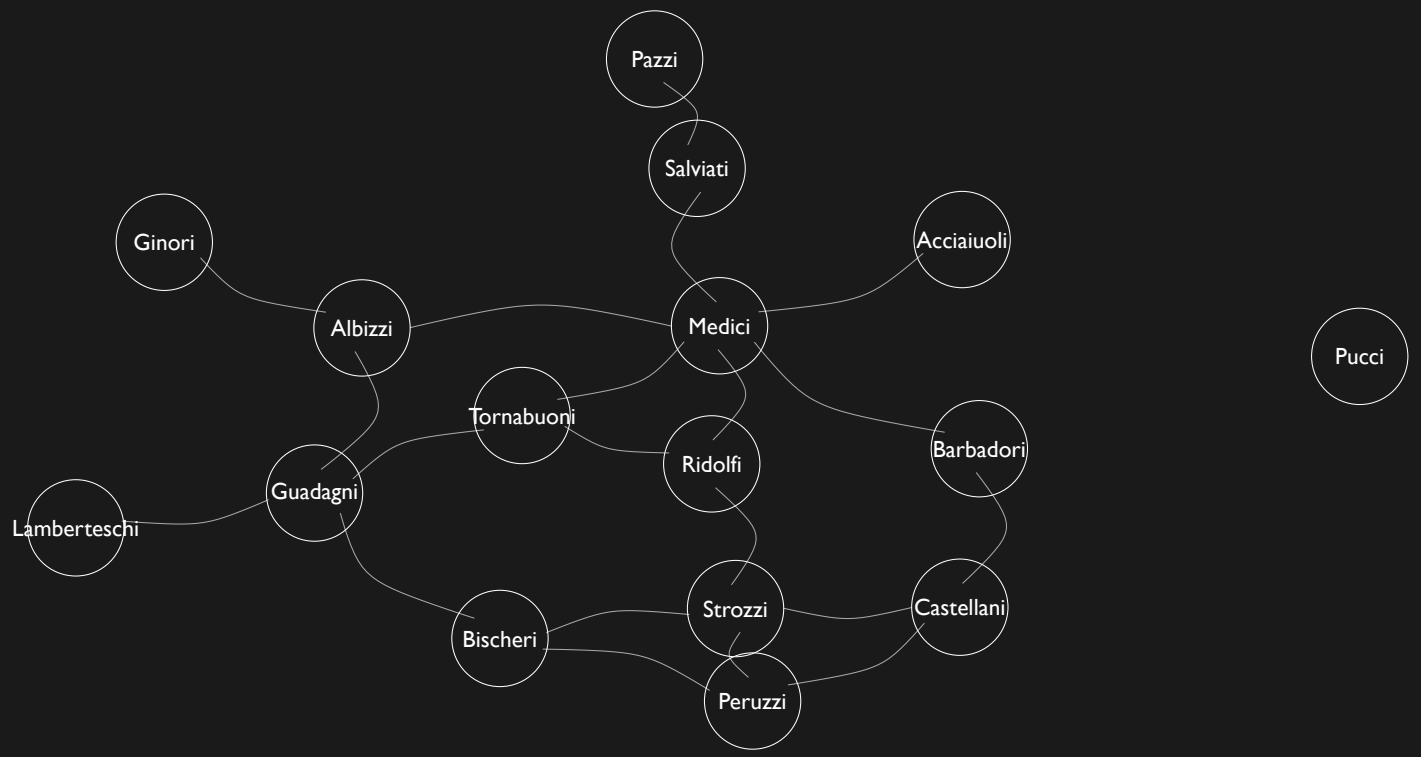


Figure 3: Uh…

```
plot(flg, layout=layout.fruchterman.reingold)
```

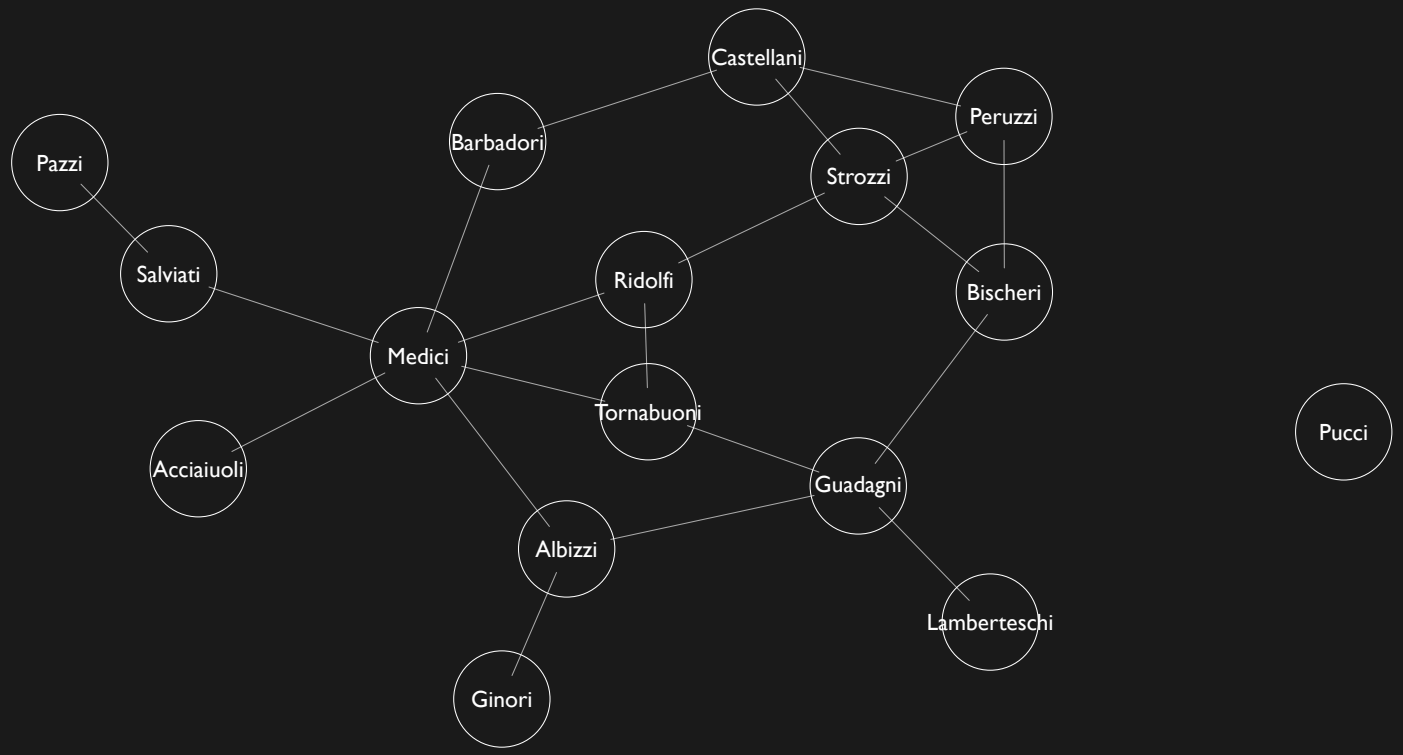


Figure 4: Still the same

```
plot(flg, layout=layout.circle)
```

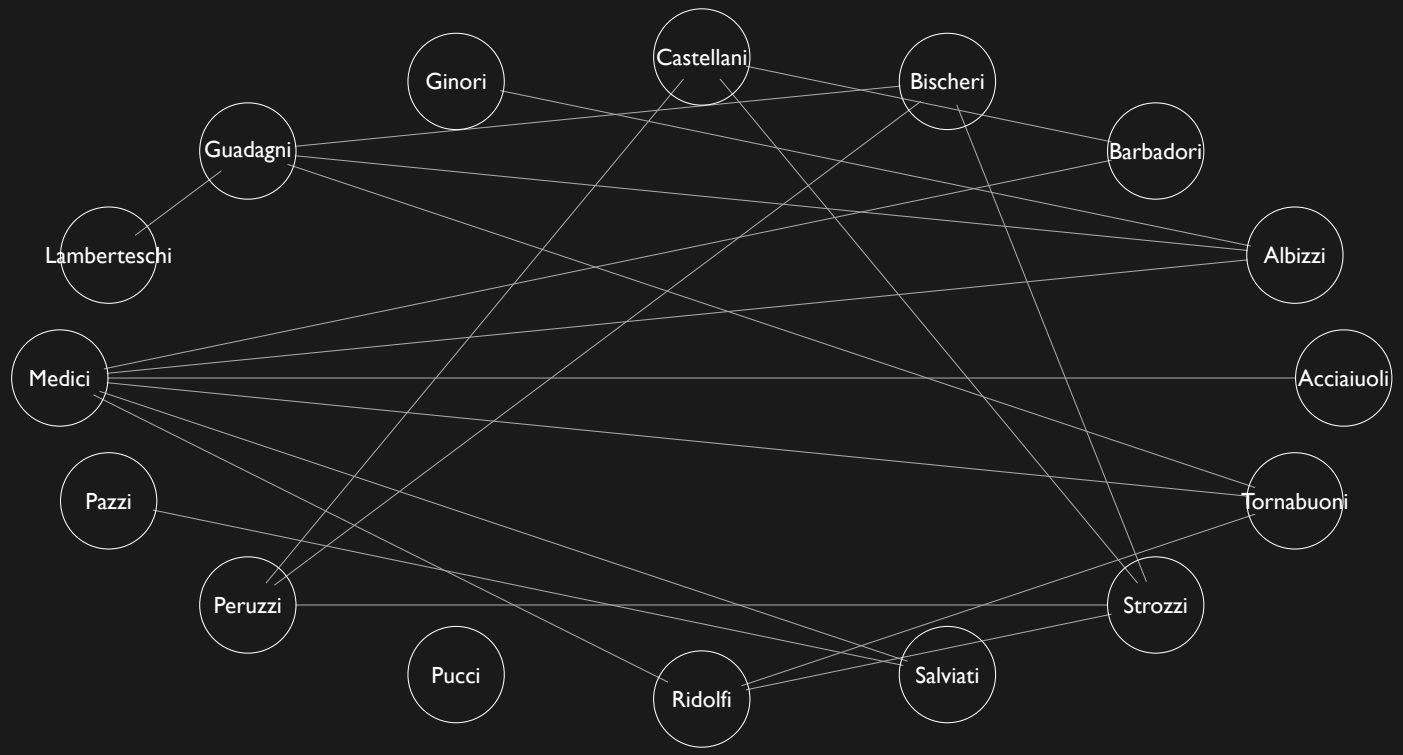


Figure 5: Not not not different

```
plot(flg, layout=layout.random)
```

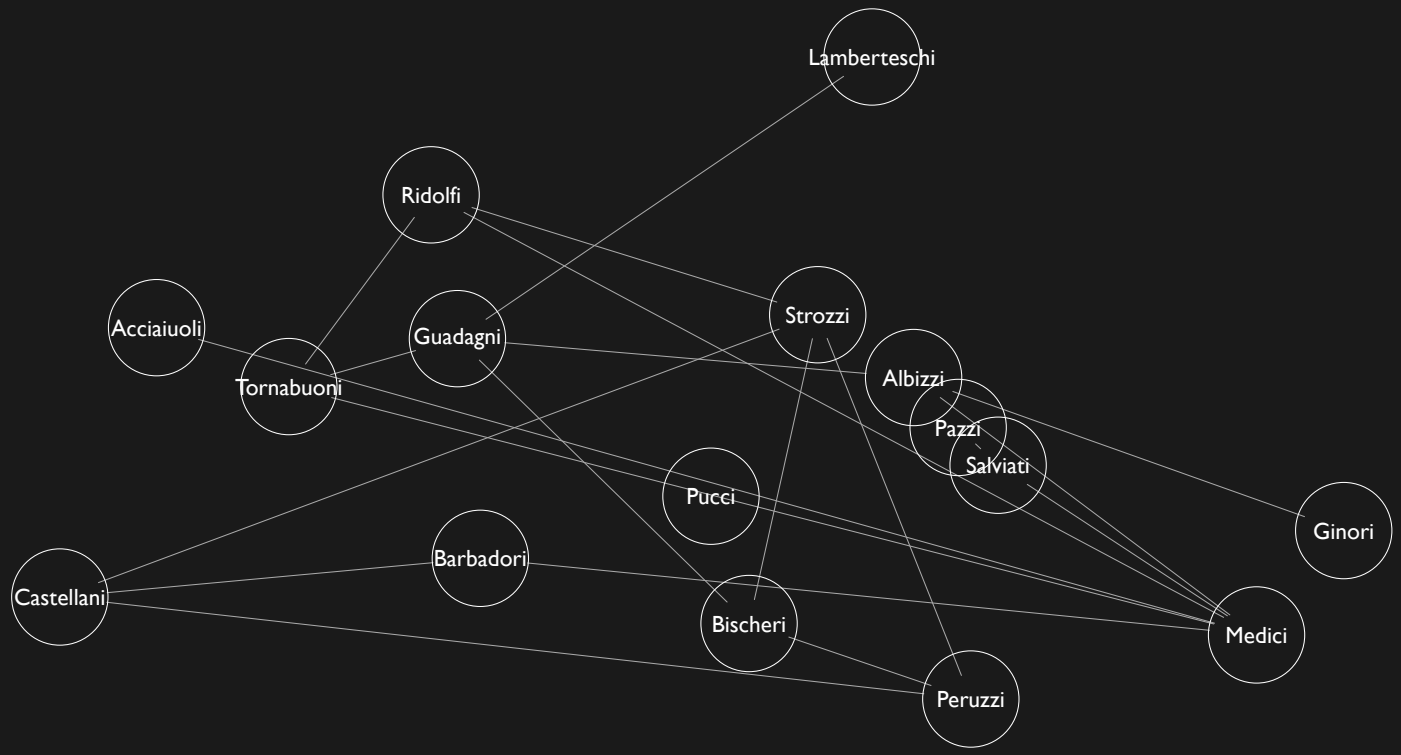


Figure 6: The idea of eternal return

```r
set.seed(297)   # otherwise non-deterministic
# plot igraph object g with xyz layout
plot(g, layout=layout.xyz)
help("plot.igraph")
help("igraph.plotting")
```

a “stat”

```r
coords <- layout.kamada.kawai(flg)
head(coords)  # 2 columns of coordinates
```

```
           [,1]        [,2]
[1,]  0.7265483  1.31148122
[2,]  1.4642171 -0.91076618
[3,] -0.4090781  0.58689322
[4,] -0.6464960 -1.91466136
[5,] -1.4483593 -0.09440367
[6,]  2.5137724 -1.22933776
```

the grammar of the network graphic

```
vs <- data_frame(family=V(flg)$name,
          x=coords[ , 1],
          y=coords[ , 2])
edges <- get.edgelist(flg, names=F)
es <- data_frame(x1=coords[edges[ , 1], 1],
                 y1=coords[edges[ , 1], 2],
                 x2=coords[edges[ , 2], 1],
                 y2=coords[edges[ , 2], 2])
p <- ggplot(vs, aes(x, y)) +
    geom_point(size=15, shape=1, color="white") +
    geom_text(aes(label=family), color="white")
p <- p + geom_segment(data=es,
                      aes(x=x1, y=y1, xend=x2, yend=y2),
                      color="white")
```

```
p + plot_theme()
```

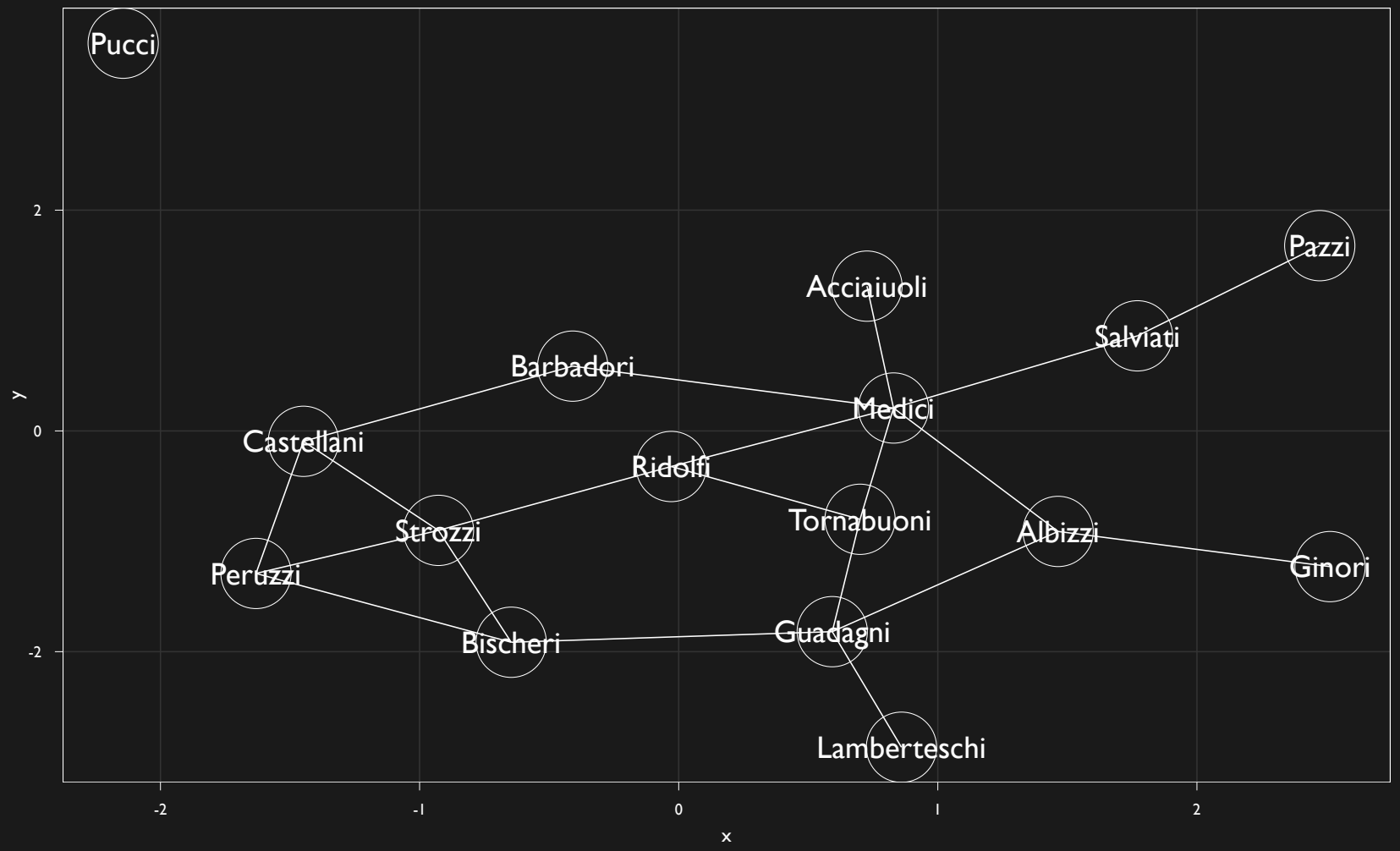


Figure 7: The same, still, but with ggplot

or back to the adjacency matrix

```
heat_p <- get.adjacency(flg, sparse=F) %>%
    as.data.frame() %>%            # matrix to frame
    mutate(ego=rownames(.)) %>%    # family names as a column
    gather("alter", "weight", -ego) %>%
    ggplot(aes(ego, alter)) + geom_tile(aes(fill=weight)) +
    xlab("") + ylab("")
```

- `geom_tile`: fill in squares at `x`, `y` aesthetics
- with color given by `fill` aesthetic
- and blank where there's no data
  (sometimes useful, sometimes better to fill in zeroes)

```
heat_p + plot_theme() + theme(legend.position="none")
```

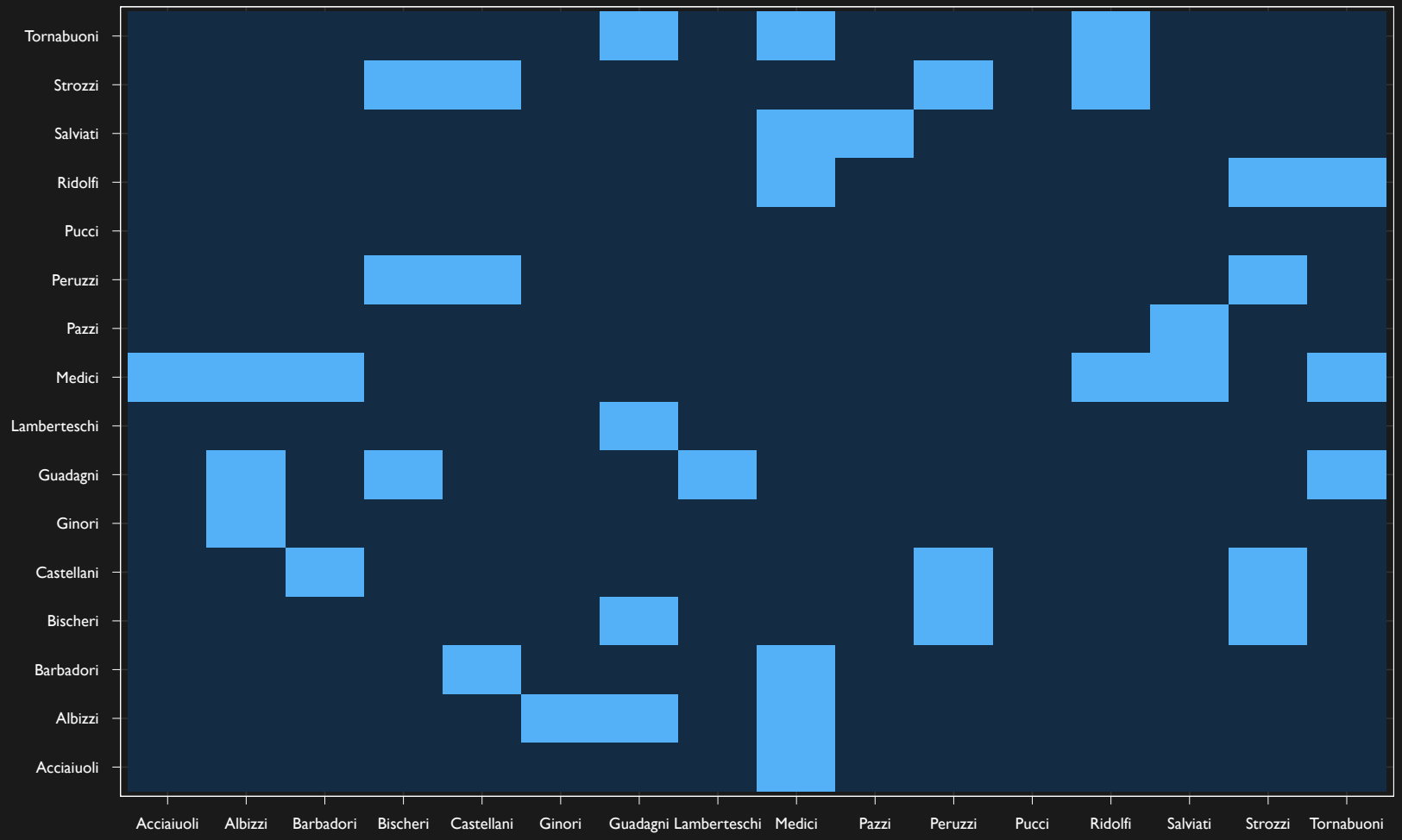


Figure 8: The adjacency matrix, visually. Note the symmetry

neighbors

```r
neighbors(flg, "Medici")
```

```
[1]  1  2  3 13 14 16
```

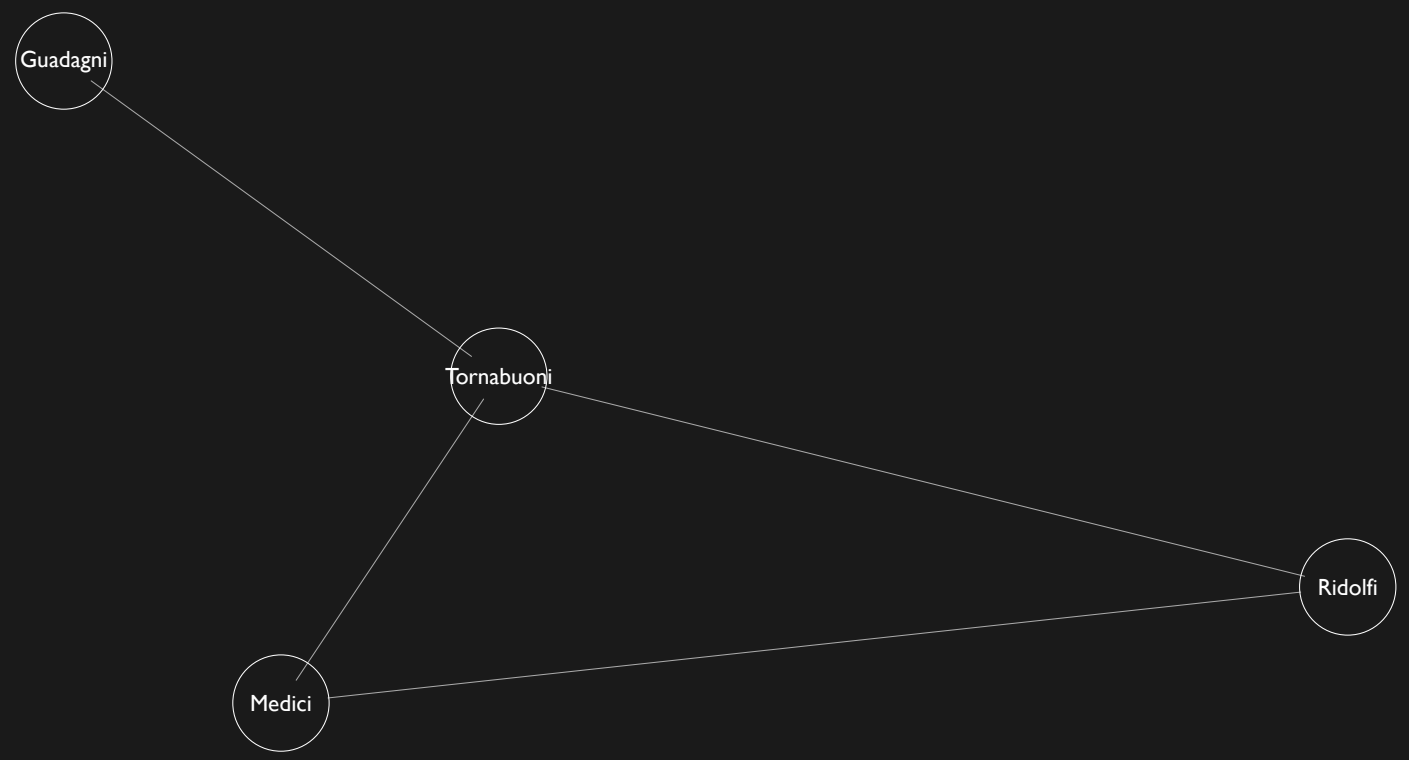
```r
V(flg)[neighbors(flg, "Medici")]
```

```
Vertex sequence:
[1] "Acciaiuoli" "Albizzi"    "Barbadori"
[4] "Ridolfi"    "Salviati"   "Tornabuoni"
```

- `incident(g, v)` for edges of `g` touching a vertex `v`

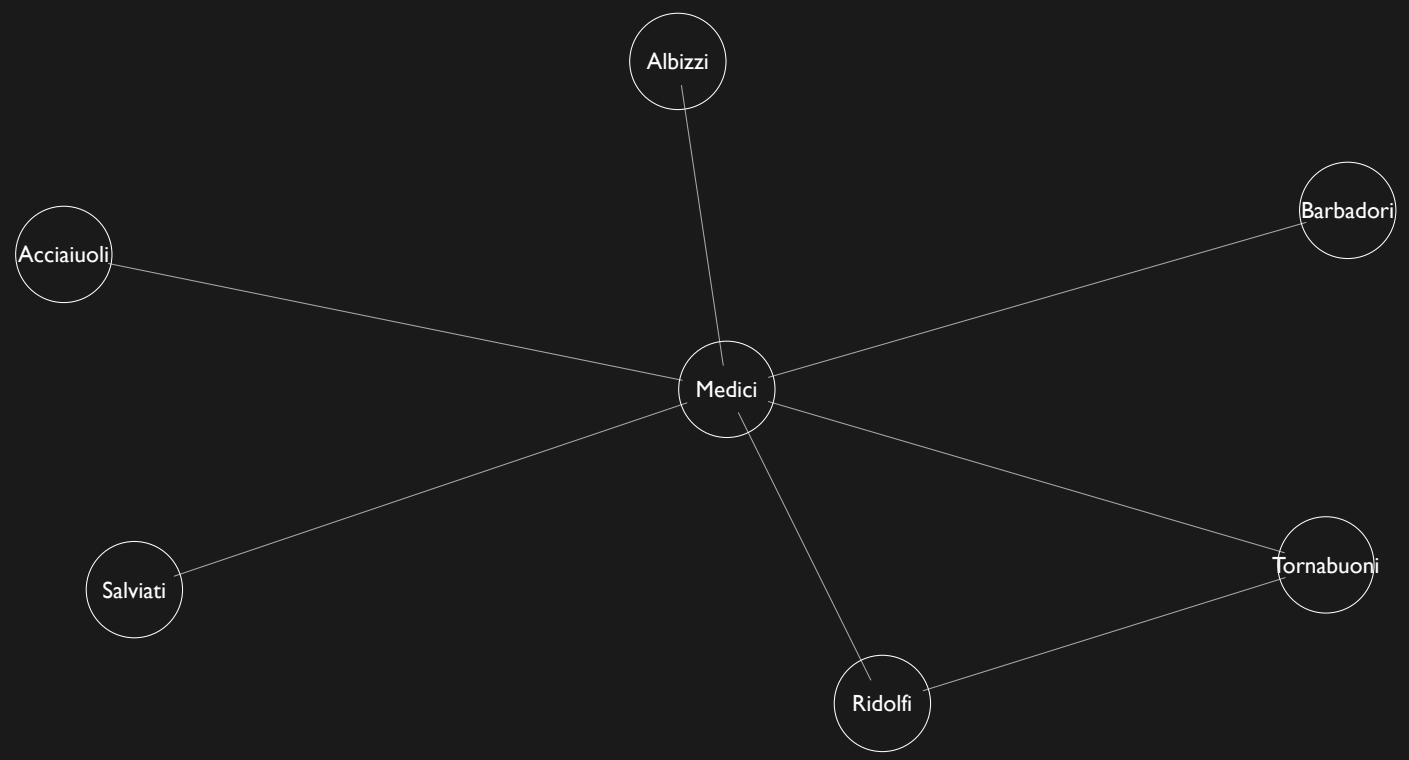
subgraph

```
sub_flg <- induced.subgraph(flg, c("Medici", "Ridolfi",
    "Tornabuoni", "Guadagni"))
plot(sub_flg)
```



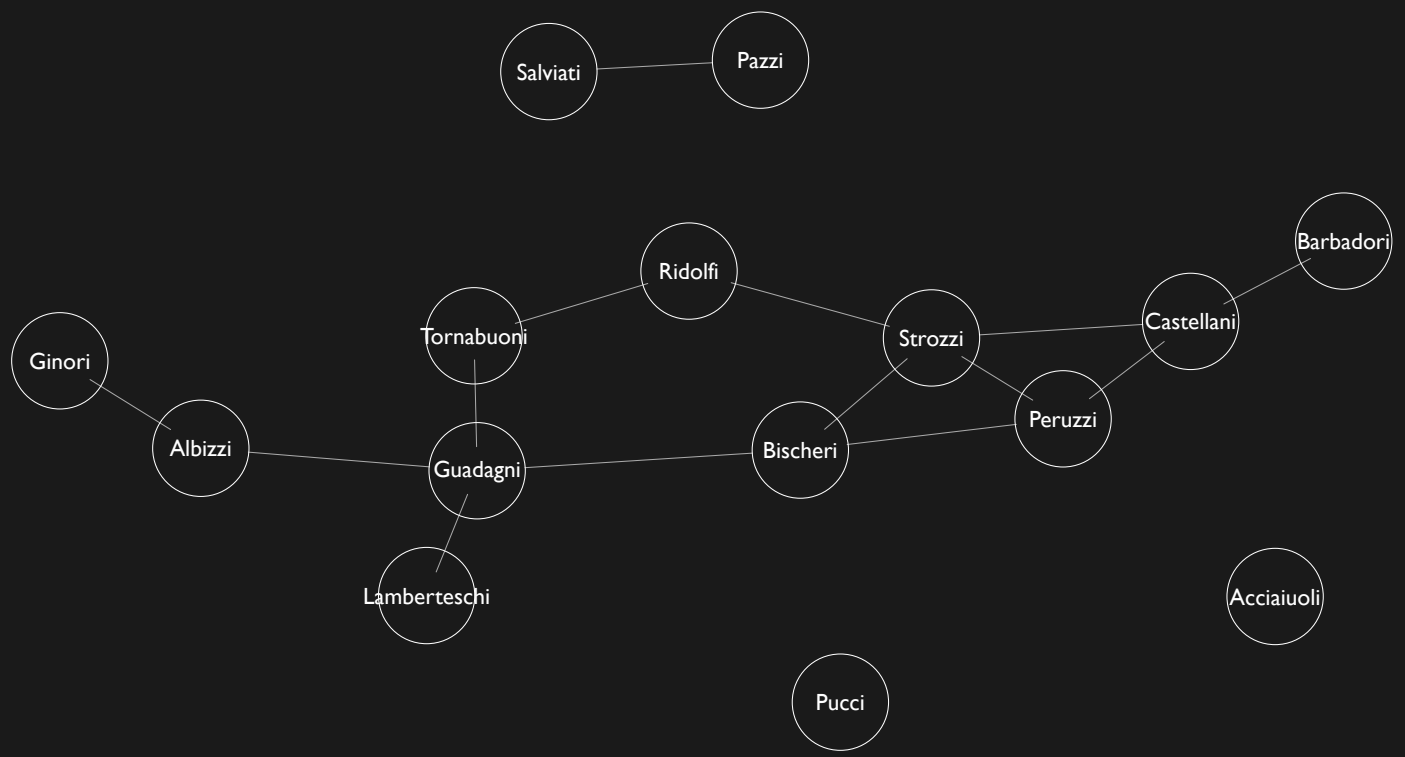
neighborhood subgraph

```
# returns a list even for one nbhd
graph.neighborhood(flg, order=1, "Medici")[[1]] %>%
    plot()
```



subtraction

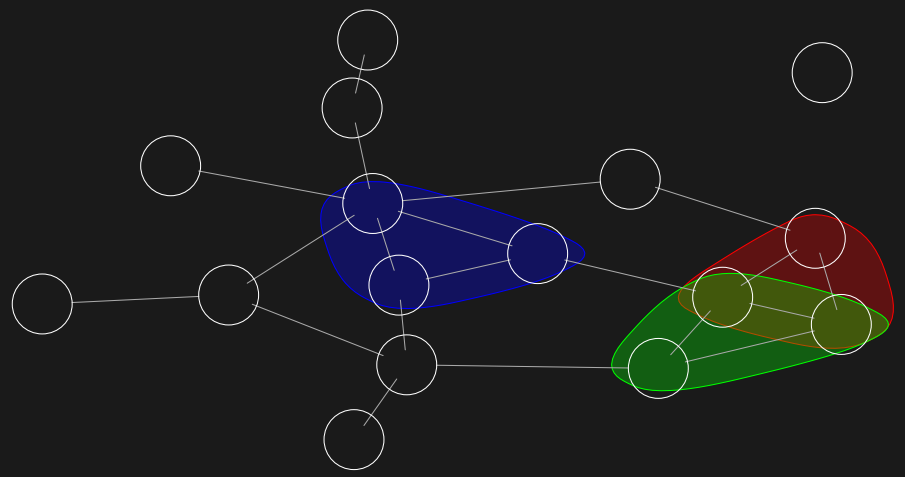
```
medicis_gone <- flg - vertex("Medici")
plot(medicis_gone)
```



cliques

```
cliques(flg)           # all complete subgraphs
largest.cliques(flg)   # list of vertex numbers
```

```
flg_clq <- largest.cliques(flg)
plot(flg, mark.groups=flg_clq, vertex.label=NA)
```



weighted edges

assign a number $A_{ij} \geq 0$ to each relation between vertices

bipartite networks

```r
# from https://github.com/kjhealy/revere
boston_aff <- read.csv("network-intro/PaulRevereAppD.csv",
                       row.names=1)
boston_aff[1:3, 1:3]
```

```
              StAndrewsLodge LoyalNine
Adams.John                 0         0
Adams.Samuel               0         0
Allen.Dr                   0         0
              NorthCaucus
Adams.John              1
Adams.Samuel            1
Allen.Dr                1
```

```r
dim(boston_aff)   # hmm...
```

```
[1] 254   7
```

- `boston_aff` is an *affiliation* matrix but not an adjacency matrix

duality of persons and groups

If A_{ij} is the affiliation (0 or 1) of person i in group j, then the number of groups two persons a and b share is

$$
\begin{aligned}
P_{ab} &= \sum_j A_{aj} A_{bj} \\
&= \sum_j A_{aj} A^T_{jb}
\end{aligned}
$$

That is $P = AA^T$ (and the group adjacency matrix is $G = A^T A$).

```
boston_aff <- as.matrix(boston_aff)
boston <- graph.adjacency(boston_aff %*% t(boston_aff),
                          mode="undirected", weighted=T,
                          diag=F)
```

```r
degree(boston, "Revere.Paul")
```

```
Revere.Paul
        245
```

```r
graph.strength(boston, "Revere.Paul")
```

```
Revere.Paul
        283
```

```r
degree(boston, "Revere.Paul")
```

```
Revere.Paul
        245
```

```r
graph.strength(boston, "Revere.Paul")
```

```
Revere.Paul
        283
```

```r
revere_nbhd <- neighbors(boston, "Revere.Paul")
length(revere_nbhd)
```

```
[1] 245
```

```r
revere_links <- incident(boston, "Revere.Paul")
sum(E(boston)[revere_links]$weight)   # attributes with $
```

```
[1] 283
```

```r
V(boston)["Revere.Paul"]$color <- "orange"
plot(boston, vertex.size=5, vertex.label=NA)
```

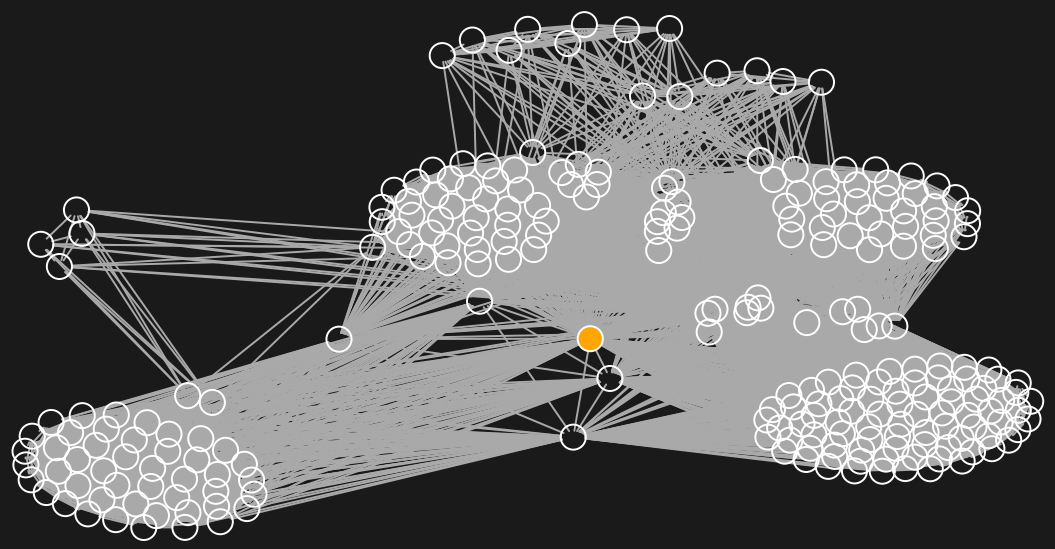


Figure 14: Listen, my vertices, and you shall hear

The betweenness centrality is

$$c_B(v) = \sum_{i \neq j \neq v \in V} \frac{\sigma(i, j|v)}{\sigma(i, j)}$$

where $\sigma(i, j)$ is the length of the shortest path from i to j and $\sigma(i, j|v)$ is the same, restricted to paths through v.

(There are many other centrality measures.)

```r
sort(degree(flg), decreasing=T)[3]
```

```
Strozzi
      4
```

```r
betweenness(flg, c("Medici", "Guadagni", "Strozzi"))
```

```
   Medici  Guadagni   Strozzi
47.500000 23.166667  9.333333
```

```
plot(flg, vertex.size=2 * sqrt(betweenness(flg)))
```

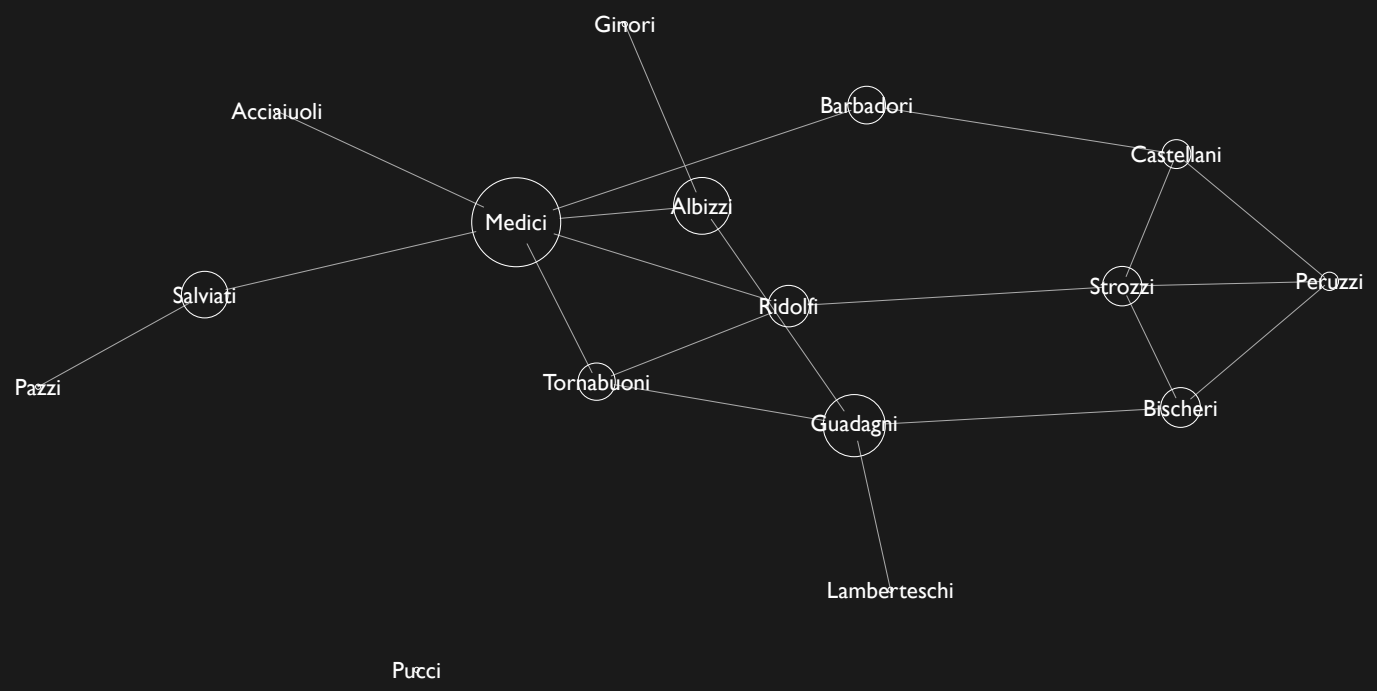


Figure 15: Circles are scaled to vertex betweenness centrality

funtimes

- in `igraph` weighted graphs $\sigma(i, j)$ is the smallest *sum* of edge weights along paths from i to j
- (not…necessarily…what you expect)

```
sort(betweenness(boston, weights=1 / E(boston)$weight),
     decreasing=T)[1:5]
```

```
     Revere.Paul    Warren.Joseph
        5511.024         2966.858
    Urann.Thomas Barber.Nathaniel
        2117.149         1894.898
    Chase.Thomas
        1346.857
```

let's get bibliographic

```
po <- read.table("network-intro/poetry_meta.tsv", sep="\t",
                 quote="", header=T, comment="",
                 stringsAsFactors=F, encoding="UTF-8")
colnames(po)
```

```
[1] "creator" "title"   "genre"   "volume"
[5] "issue"   "pubdate"
```

```
nrow(po)
```

```
[1] 4357
```

a handy function

```r
editors <- c("Ficke", "Moody", "Pound", "Lorimer")
combn(editors, 2) # matrix of all pairs
```

```
     [,1]    [,2]    [,3]      [,4]
[1,] "Ficke" "Ficke" "Ficke"   "Moody"
[2,] "Moody" "Pound" "Lorimer" "Pound"
     [,5]      [,6]
[1,] "Moody"   "Pound"
[2,] "Lorimer" "Lorimer"
```

more dplyr utilities

`distinct`	like `unique` but for data frames (by row)
`first`	`first(x) == x[1]`
`last`	`last(x) == x[length(x)]`
`data_frame`	like `data.frame` but assumes `stringsAsFactors=F`

edge list

```
po_pairs <- po %>%
    filter(genre == "poetry") %>%
    select(volume, issue, creator) %>%
    distinct() %>% # count each poet once per issue
    group_by(creator) %>%
    filter(n() > 5) %>%   # threshold
    group_by(volume, issue) %>% # could use pubdate instead
    filter(n() > 1) %>% # drop singletons
    do({ # split, apply, combine
        copubs <- combn(.$creator, 2)
        data_frame(poet1=copubs[1, ],
                   poet2=copubs[2, ])
    })
```

```
head(po_pairs[ , c("poet1", "poet2")]) %>% print_tabular()
```

poet1	poet2
Ficke, Arthur Davison	Pound, Ezra
Ficke, Arthur Davison	Conkling, Grace Hazard
Pound, Ezra	Conkling, Grace Hazard
Aldington, Richard	Monroe, Harriet
Yeats, William Butler	Corbin, Alice
Ficke, Arthur Davison	Bynner, Witter

direction

- Is there any difference between

```
poet1  poet2
    X      Y
```

and

```
poet1  poet2
    Y      X
```

?

unordering (hard way)

```
po_pairs %>%
    mutate(u1=ifelse(poet1 < poet2, poet1, poet2),
           u2=ifelse(poet1 < poet2, poet2, poet1)) %>%
    select(u1, u2)
```

- also memory-greedy (double space)

unordering (still hard)

```
po_pairs %>%
    mutate(id=1:n()) %>%
    gather("colnum", "poet", -volume, -issue, -id) %>%
    group_by(volume, issue, id) %>%
    arrange(poet) %>%
    mutate(colnum=c("col1", "col2")) %>%
    spread(colnum, poet)
```

- fortunately, someone else is going to solve this particular problem

weighted or unweighted?

- Did they ever appear together?
- *or* How many times did they appear together?
  - Should appearing four times count twice as much as appearing twice?

the software does the work

```
# chuck blank names (should go back to the data)
po_copub <- ungroup(po_pairs) %>%
    filter(poet1 != "", poet2 != "") %>%
    select(poet1, poet2) %>%
    as.matrix() %>%
    graph.edgelist(directed=F)
```

- but `po_copub` is not a simple graph! `igraph` can help:

```
E(po_copub)$weight <- count.multiple(po_copub)
po_copub <- simplify(po_copub)
```

degree distribution, un/weighted

```
stack(list(unweighted=degree(po_copub), # see Teetor, 5.5
           weighted=graph.strength(po_copub))) %>%
    ggplot(aes(values)) +
        geom_bar(binwidth=4, color="white") +
        facet_wrap(~ ind) + plot_theme() + xlab("degree")
```

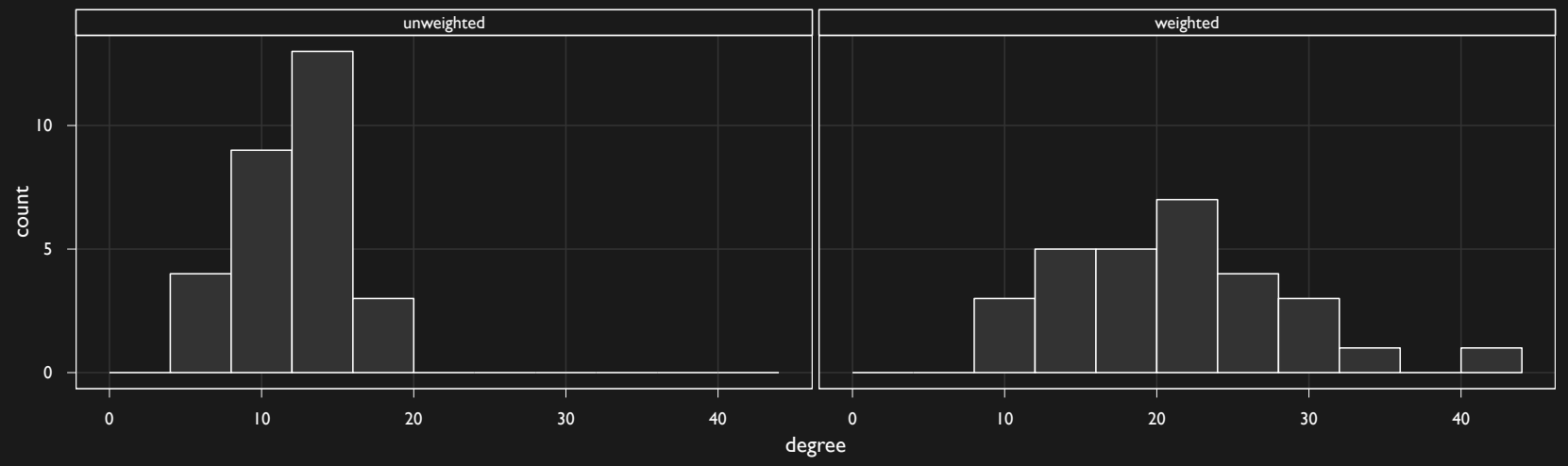


Figure 16: Degree distribution of *Poetry* copublication

```
plot(po_copub, vertex.label=NA, vertex.size=5)
```

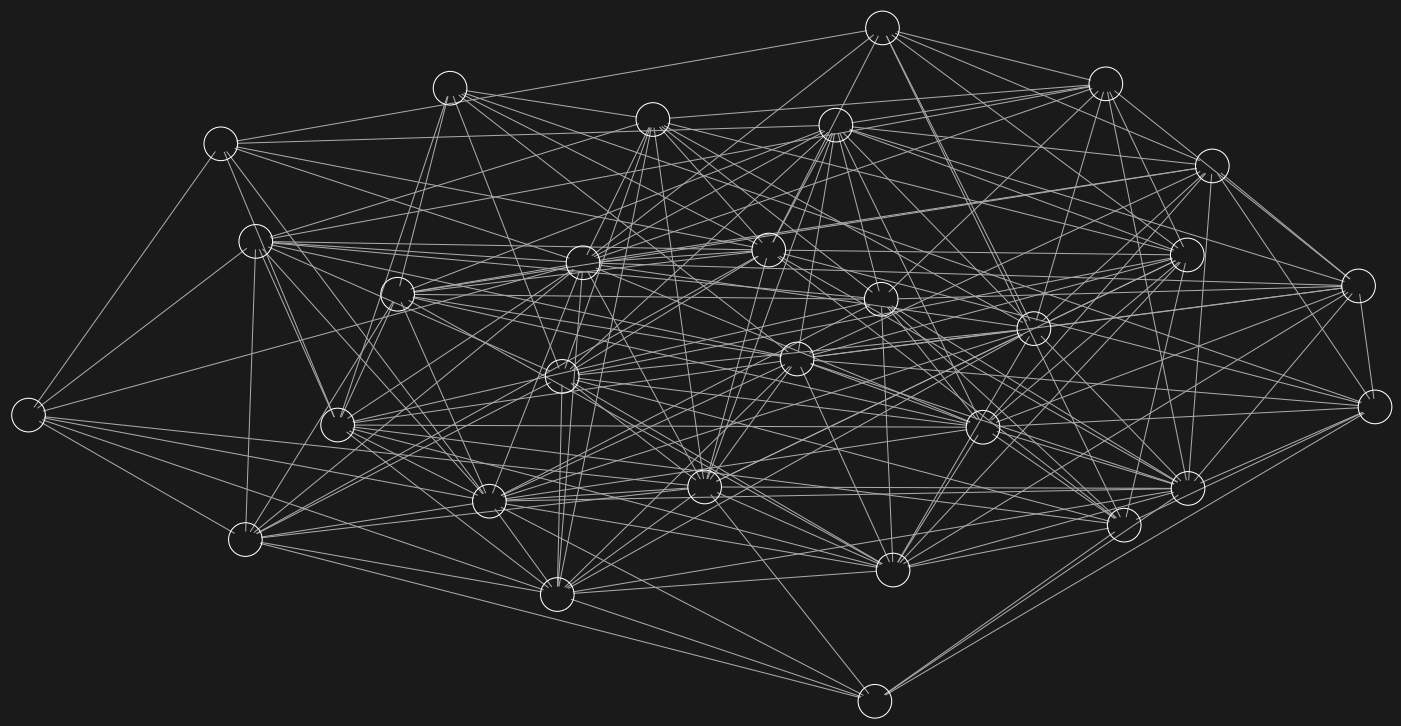


Figure 17: Hairball

house standbys (?)

```
V(po_copub)[largest.cliques(po_copub)[[1]]]
```

```
Vertex sequence:
[1] "Stevens, Wallace"
[2] "Sandburg, Carl"
[3] "Bodenheim, Maxwell"
[4] "Aldington, Richard"
[5] "Corbin, Alice"
[6] "Driscoll, Louise"
[7] "Lowell, Amy"
```

one more bibliographic network

```
authors <- read.table("network-intro/c20_authors.tsv",
                      header=T, quote="", sep="\t",
                      as.is=T)
aug <- authors %>%
    filter(pubtype == "book") %>%
    select(id, author) %>%
    distinct() %>%
    group_by(author) %>%
    filter(n() > 5) %>%   # threshold
    group_by(id) %>%
    filter(n() > 1) %>% # drop singletons
    do({
        auths <- combn(.$author, 2)
        data_frame(ego=auths[1, ],
                   alter=auths[2, ])
    })
```

```r
aug <- ungroup(aug) %>%
    select(ego, alter) %>%
    as.matrix() %>%
    graph.edgelist(directed=F)
E(aug)$weight <- count.multiple(aug)
aug <- simplify(aug)
# Kleinberg's authority score (Google-esque)
sort(authority.score(aug)$vector, decreasing=T)[1:6]
```

```
          Eliot, T. S.           Pound, Ezra
             1.0000000             0.9012067
          Joyce, James Yeats, William Butler
             0.4644893             0.4135240
       Woolf, Virginia      Stevens, Wallace
             0.4032480             0.3177726
```

btw

```
devtools::install_github("gastonstat/arcdiagram")
library("arcdiagram")
```

```
arcplot(get.edgelist(flg), horizontal=F,
        col.arcs="white", col.labels="white")
```

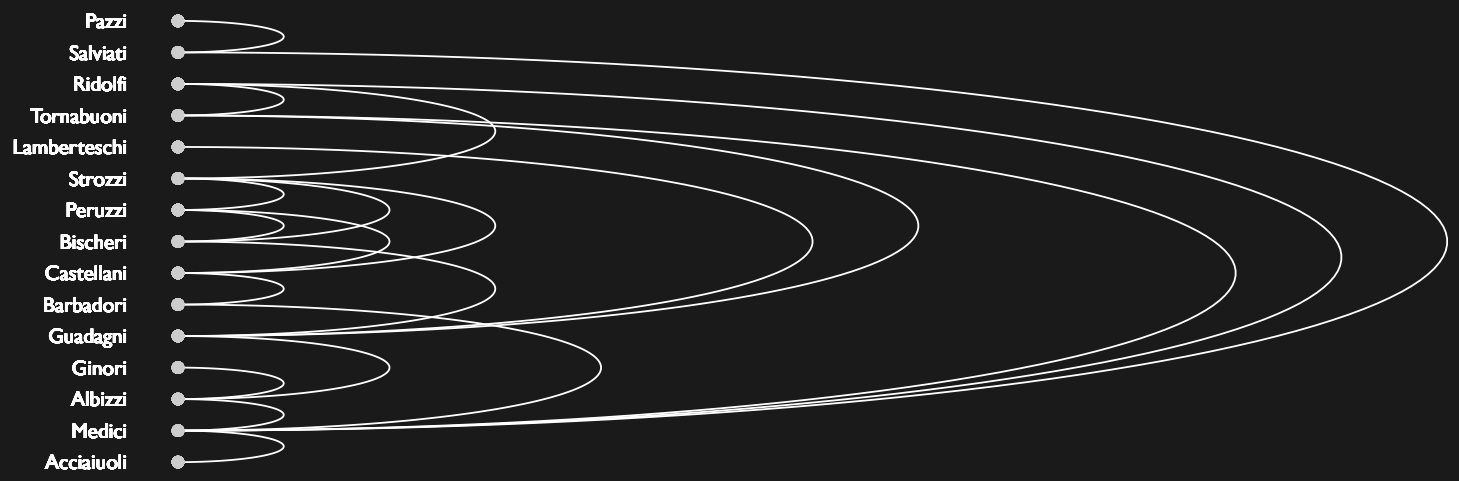


Figure 18: Arc diagrams *are* possible in R

igraph manual++

Kolaczyk, Eric D., and Gábor Csárdi. *Statistical Analysis of Network Data with R*. New York: Springer, 2014. DOI: 10.1007/978-1-4939-0983-4.

chaos of network software

`tnet`	builds on `igraph` with more for weighted networks
`statnet`	R package family (includes `sna`, `network`, `ergm`)
`intergraph`	conversions among the packages
`GGally`	`ggplot` extensions, including for networks
Gephi	Graphical monstrosity, widely used, crashes like the dickens