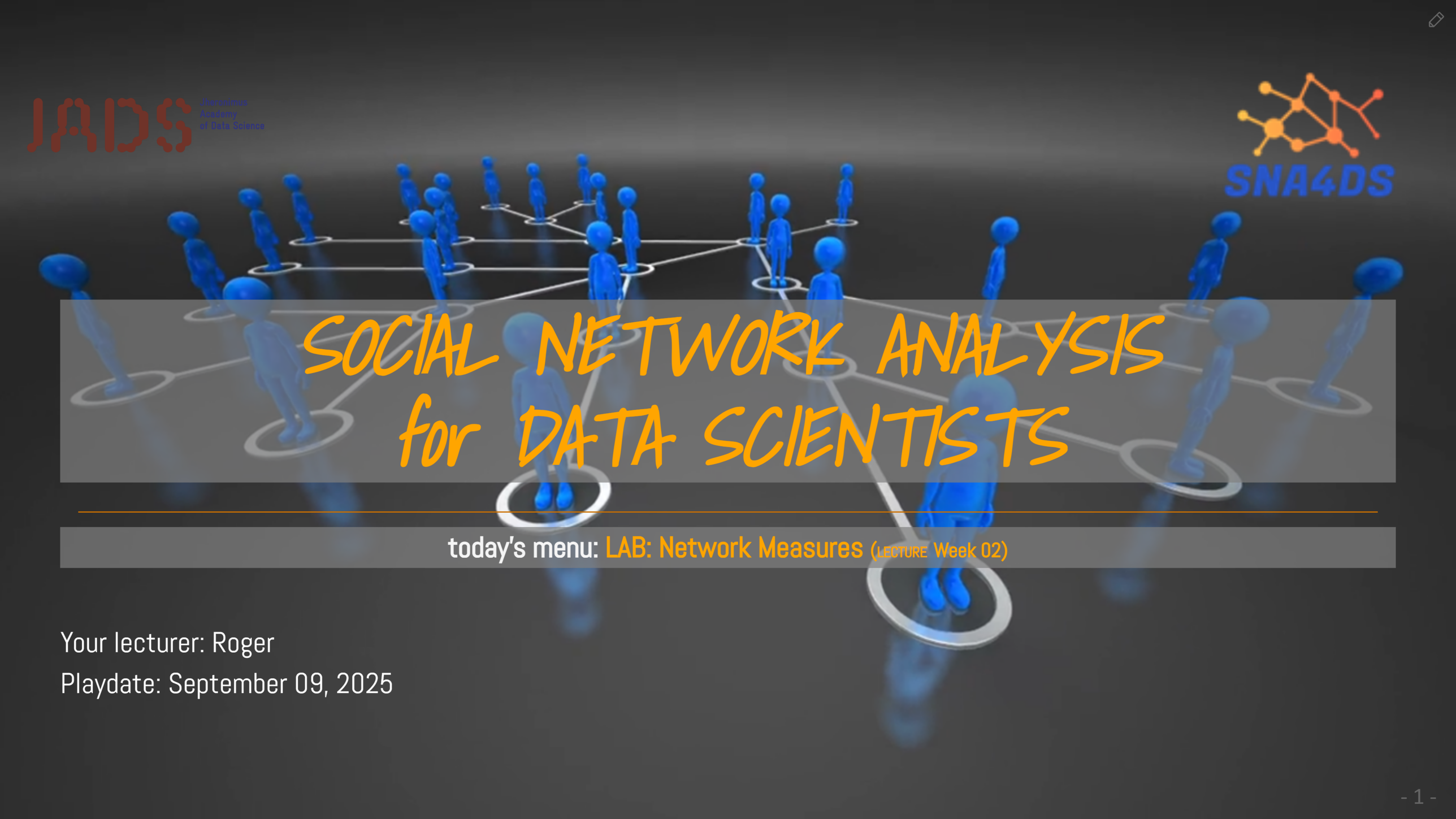


SOCIAL NETWORK ANALYSIS for DATA SCIENTISTS

today's menu: LAB: Network Measures (LECTURE Week 02)

Your lecturer: Roger

Playdate: September 09, 2025

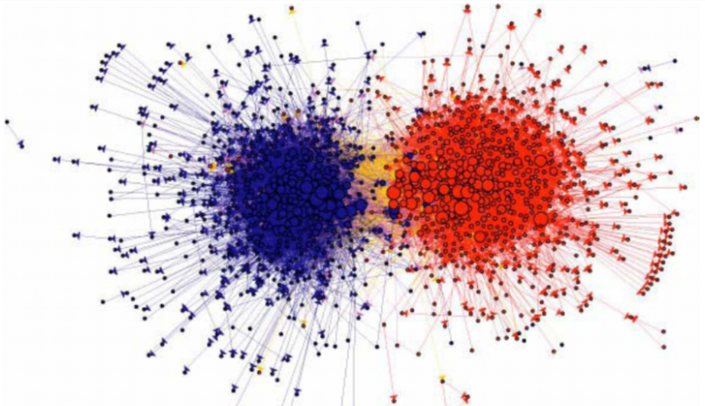
Today's menu

1. Homeplay
2. Bridging
3. Quiz time!
4. Pagerank

The dataset

```
data(blogosphere, package = "SNA4DSData")
snafun::print(blogosphere)
```

```
IGRAPH 3254f1f D--- 1490 19090 --
+ attr: id (v/n), label (v/c), value (v/n), source (v/c), party (v/c),
| color (v/c), within (e/l), within_republican (e/l), within_democrat
| (e/l)
+ edges from 3254f1f:
 [1]  267->1394  267-> 483  267->1051  904->1479  904-> 919  904->1045
 [7]  904->1330  904->1108  904-> 995  904-> 963  904->1000  904->1192
[13]  904->1067  904->1270  904->1437  904->1037  903-> 855  903->1008
[19]  982->1429  982-> 952  982-> 892  982->1479  982-> 956  982->1306
[25]  982-> 810  982->1437  982->1112  982->1051  982-> 826  982->1478
[31]  982-> 979  982->1255 1167-> 963 1167-> 802 1167-> 842 1167->1408
+ ... omitted several edges
```



Some descriptives

```
# mean distance
snafun::g_mean_distance(blog1)
```

```
[1] 3.390184
```

```
# diameter
snafun::g_diameter(blog1)
```

```
[1] 9
```

```
# dyad census
snafun::count_dyads(blog1)
```

```
 Mutual Asymmetric    Null 
   2307      14408 1092590 
```

```
# reciprocity
snafun::g_reciprocity(blog1)
```

Transitivity

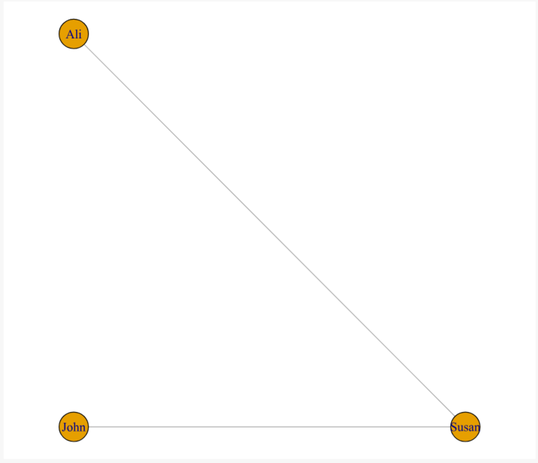
A friend of a friend is also my friend

This is *pure transitivity*, but it often doesn't happen that strongly.

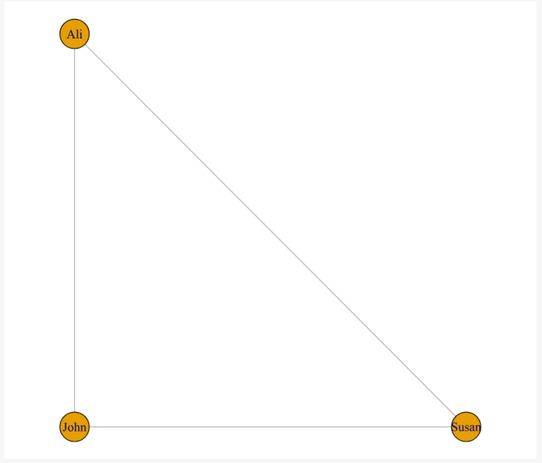
Often more accurate:

> If John is friends with Susan and Susan is friends with Ali, then it is *more likely* that Ali and John are friends than not.

If this third edge is there, we say that Ali and John *close the triad*.

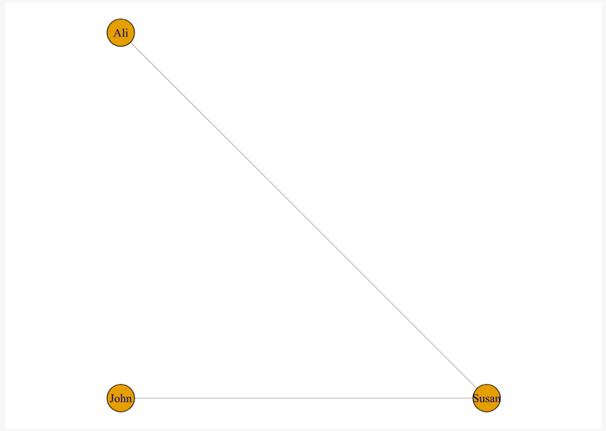


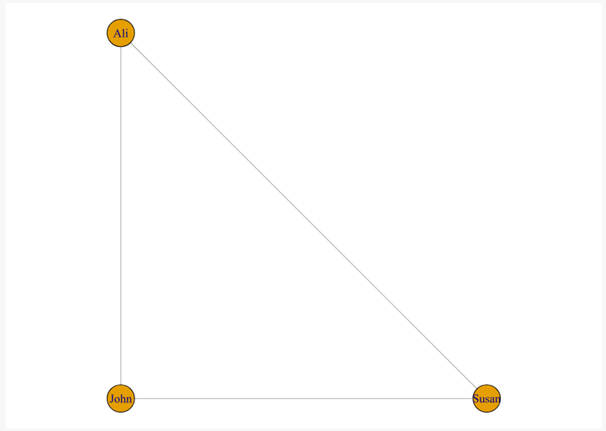
Open triad



Closed triad

Transitivity





- John -- Susan -- Ali form a path of length 2
- A closed triad forms a cycle of length 3
- we look at each set of 3 vertices that have a path of length 2 and count the proportion of them that form the full cycle (ie. that are *closed*),

$$Transitivity = \frac{no.closedtriads}{no.pathsoflength2}$$

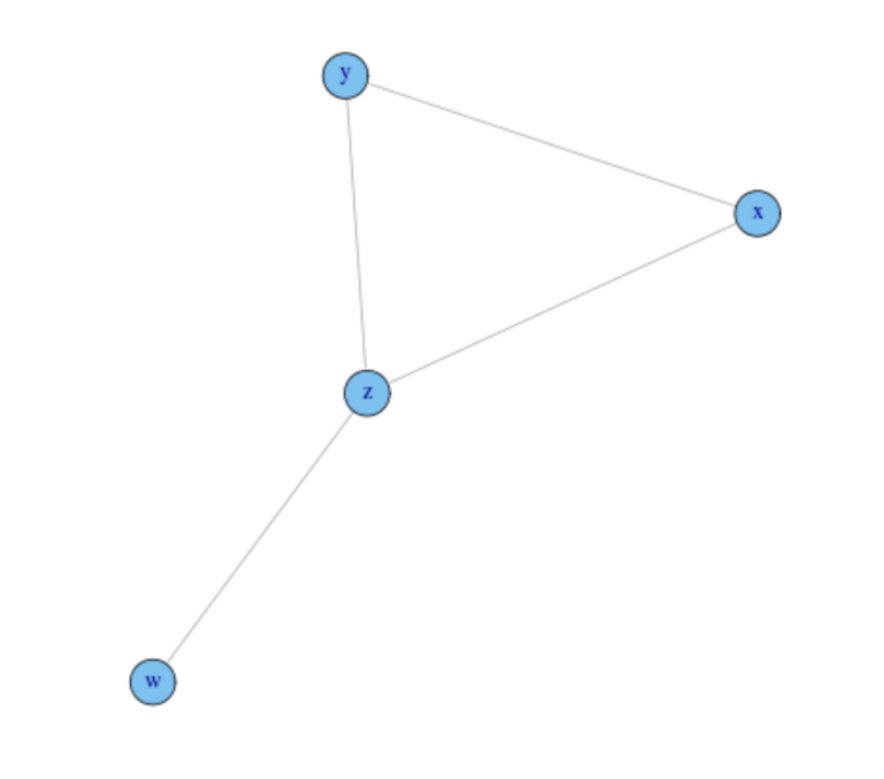
(so: the proportion of friends-of-my-friend who are **also** my friend)

This is often also called the "clustering coefficient".

Transitivity = 1 implies perfect transitivity
Transitivity = 0 implies no closed triads

Transitivity

Look at this network. What is the transitivity in this network?



Transitivity = 0.6

Triads with a two-path in them:

- y-x-z: closed
- y-z-x: closed
- x-y-z: closed
- w-z-y: open
- w-z-x: open

So, 3 out of 5 (= 60%) are closed.

```
snafun::create_manual_graph(w--z--y--x--z) |> snafun::g_transitivity()
```

```
[1] 0.6
```

Transitivity

Recall that

```
snafun::g_transitivity(blog1)
```

```
[1] 0.2395381
```

What does this tell us about the network?

Transitivity

In *directed* networks, we need to take into account in which direction the friendship flows.

However, it is common to consider directed ties *as being undirected* when determining transitivity.

`snafun::g_transitivity` and `igraph::transitivity` consider every graph to be *undirected*.

`sna::gtrans` allows you to choose between directed or undirected.

We will come back to transitivity in directed graphs in the second half of the course when discussing ERGM's.

walktrap communities

Back to the blogosphere. Is the walktrap a meaningful approach for this network?

How many communities did you find? Do you find that surprisingly few, surprisingly many, as-expected?

➔ The walktrap algorithm tends to return more communities than alternative algorithms. Often, you want to discard small walktrap communities.

Isolates

Toss them isolates!

How many communities did you find for the "pruned" network? Do you find that surprisingly few, surprisingly many, as-expected?

```
blog0 <- snafun::remove_isolates(blog1)
walk0 <- snafun::extract_comm_walktrap(blog0)
igraph::membership(walk0) |> table()
```

```
  1   2   3   4   5   6   7   8   9  10  11  12
560   4 640   2   2   4   2   2   2   2   2   2
```

```
igraph::modularity(walk0)
```

```
[1] 0.4302334
```

```
igraph::sizes(walk0) |> length()
```

```
[1] 12
```

Today's menu

Homeplay

Homeplay bonus

Bonus question

assignment | mixing matrix | table

Continue with this smaller network.

1. Add the group membership (which you get with `igraph::membership`) as a vertex attribute.
2. Run `snafun::make_mixingmatrix(graph_without_isolates, "community")` and interpret the result. Does it make sense? What does this tell you about the boundaries around the communities?.
3. Extract all vertex attributes using `snafun::extract_all_vertex_attributes` and explore whether it is indeed the case that like-minded blogs tend cluster together inside the communities you found.

Bonus question

assignment | mixing matrix | table

```
blog0 <- snafun::add_vertex_attributes(blog0, "community", value = igraph::membership(walk0))
```

```
# mixing matrix
snafun::make_mixingmatrix(blog0, "community")
```

```
     to
from    1    2    3    4    5    6    7    8    9   10   11   12
  1  8339    1  633    0    1    0    0    0    0    2    1    0
  2     2    3    3    0    0    0    0    0    0    0    0    0
  3   675    0 9338    0    0    1    0    1    1    0    0    0
  4     1    0    1    1    0    0    0    0    0    0    0    0
  5     1    0    0    0    1    0    0    0    0    0    0    0
  6     0    0    0    0    0    5    0    0    0    0    0    0
  7     1    0    0    0    0    0    1    0    0    0    0    0
  8     0    0    0    0    0    0    0    1    0    0    0    0
  9     0    0    0    0    0    0    0    0    1    0    0    0
```

Bonus question

assignment | mixing matrix | table

```
# get all attributes from the network object
all_attr <- snafun::extract_all_vertex_attributes(blog0)

table(all_attr$community, all_attr$party)
```

```
    conservative liberal
 1            23     537
 2             1       3
 3           604      36
 4             2       0
 5             0       2
 6             4       0
 7             0       2
 8             0       2
 9             2       0
 10            0       2
```

Today's menu

Homeplay

Homeplay bonus

Bridging

Who bridges between the communities?

Let's find the blogs that refer the most to blogsites in the "opposite" community.

How could we do this?

One strategy:

- find those edges that cross boundaries
- find which vertices are part of those edges
- look at the most active ones

Let us first limit ourselves to the two main communities, since the others are so small.

How could we do this?

Approach 1 | Approach 2

1. identify the vertices that are in communities 1 or 3
2. take the subset of the network that only includes these vertices (and the edges between them)--this is the so-called ***induced subgraph***.

```
# TRUE / FALSE: is a vertex in 1 or 3
in_1_3 <- snafun::extract_vertex_attribute(blog0, "community") %in% c(1, 3)
small_net <- snafun::extract_subgraph(blog0, v_to_keep = in_1_3)
snafun::print(small_net)
```

```
IGRAPH efc1180 D--- 1200 18985 --
+ attr: id (v/n), label (v/c), value (v/n), source (v/c), party (v/c),
| color (v/c), community (v/n)
+ edges from efc1180:
 [1] 1->  21 1->  48 1->  70 1-> 123 1-> 245 1-> 278 1-> 330 1-> 366 1-> 439
[10] 1-> 485 1-> 486 1-> 488 1-> 505 1-> 987 1->1154 2->   1 2->  16 2->  22
[19] 2->  26 2->  46 2->  48 2->  82 2-> 116 2-> 122 2-> 123 2-> 127 2-> 137
[28] 2-> 142 2-> 160 2-> 182 2-> 189 2-> 245 2-> 252 2-> 275 2-> 294 2-> 296
[37] 2-> 328 2-> 341 2-> 362 2-> 366 2-> 375 2-> 416 2-> 417 2-> 428 2-> 430
```

Let us first limit ourselves to the two main communities, since the others are so small.

How could we do this?

Approach 1 | Approach 2

1. identify the vertices that are *not* in communities 1 or 3
2. remove those vertices from the network (and the orphan edges)

```
# TRUE / FALSE: is a vertex in 1 or 3
not_in_1_3 <- !in_1_3
small_net2 <- snafun::remove_vertices(blog0, v = not_in_1_3)
snafun::print(small_net2)
```

```
IGRAPH efc2b26 D--- 1200 18985 --
+ attr: id (v/n), label (v/c), value (v/n), source (v/c), party (v/c),
| color (v/c), community (v/n)
+ edges from efc2b26:
 [1] 1->  21 1->  48 1->  70 1-> 123 1-> 245 1-> 278 1-> 330 1-> 366 1-> 439
[10] 1-> 485 1-> 486 1-> 488 1-> 505 1-> 987 1->1154 2->   1 2->  16 2->  22
[19] 2->  26 2->  46 2->  48 2->  82 2-> 116 2-> 122 2-> 123 2-> 127 2-> 137
[28] 2-> 142 2-> 160 2-> 182 2-> 189 2-> 245 2-> 252 2-> 275 2-> 294 2-> 296
[37] 2-> 328 2-> 341 2-> 362 2-> 366 2-> 375 2-> 416 2-> 417 2-> 428 2-> 430
[46] 2-> 431 2-> 432 2-> 436 2-> 474 2-> 485 2-> 486 2-> 488 2-> 494 2-> 505
```

Now that we only have the vertices from communities 1 and 3 left, how do we find the most active connectors?

➔ There is a function called `igraph::crossing` that indicates for every edge whether it connects two communities (`TRUE` / `FALSE`).

(you find this by going through the `igraph` help or by browsing the cheatsheet.)

Approach | R code | Gimme details!

One approach:

- identify all "crossing" edges
- for each edge, identify the sender
- create a table of the number of times each sender occurs
- done!

Approach | R code | Gimme details!

```
small_walk <- snafun::extract_comm_walktrap(small_net)
crossing <- igraph::crossing(small_walk, small_net) |> which()

senders <- igraph::ends(small_net, crossing)[, 1]

sort(table(senders), decreasing = TRUE)
```

```
senders
 362 1158   12  513  224  553  821  625  120  252  411 1146  209  292  394  479
  25   20   18   17   16   16   16   15   13   13   13   13   12   12   12   12
 508  579  726  878  467  657 1057   99  363  458  190  426  684 1177  174  295
  12   12   12   12   11   11   11   10   10   10    9    9    9    9    8    8
 348  463  719  904 1103   28   74   97  108  486  563  727  831   19   95  284
   8    8    8    8    8    7    7    7    7    7    7    7    7    6    6    6
 299  318  355  391  569  593  598  734  956 1054 1085 1113 1121   20  121  160
   6    6    6    6    6    6    6    6    6    6    6    6    6    5    5    5
 165  184  193  333  356  370  401  417  471  488  552  559  629  678  970 1029
```

Approach R code **Gimme details!**

```
table_all <- sort(table(senders), decreasing = TRUE) |> names()  # table column names

snafun::extract_all_vertex_attributes(small_net)[table_all, c("label", "party", "community")]
```

Show 10 entries Search:

id	label	party	community
362	obsidianwings.blogs.com	liberal	1
1158	truthprobe.blogspot.com	conservative	3
12	aintnobaddude.com	liberal	1
513	thetalkingdog.com	liberal	1
224	iwantmycountryback.org	liberal	3
553	washingtonmonthly.com	liberal	1
821	instapundit.com	conservative	3
625	balloon-juice.com	conservative	3

Showing 1 to 10 of 420 entries Previous 1 2 3 4 5 … 42 Next

Today's menu

Homeplay

Homeplay bonus

Bridging

Quiz

Facility location analysis

Situation 1

Consider the location of a *fire station*. When a fire occurs, a firetruck should be able to arrive to any potential fire location as soon as possible.

Which centrality measure finds you an optimal location (with respect to this single objective)?

www.menti.com / code: 49197653

Facility location analysis

This is called a **minimax location problem**.

This is captured by *eccentricity*.

Facility location analysis

Situation 2

Find a spot for a shopping mall, such that the entire pool of customers have a minimal distance to the mall.

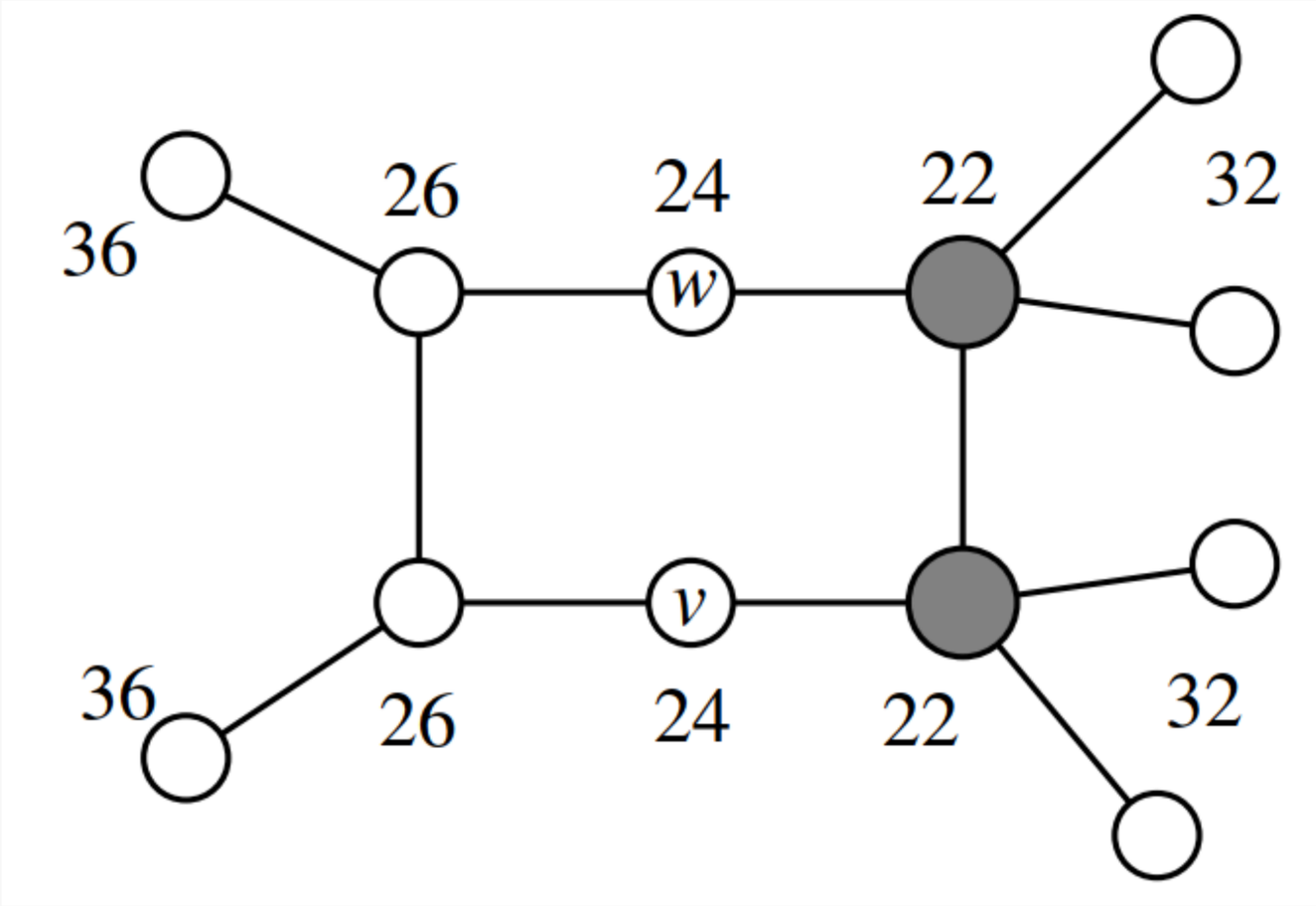
This increases access for the general population and minimizes fuel consumption of customers traveling to the mall.

Which centrality measure finds you an optimal location (with respect to this single objective)?

Facility location analysis

This is called a **minisum location problem**.

This is captured by *closeness centrality*.



Total distances in this network.

Note that nodes *v* and *w* have lower eccentricity.

Facility location analysis

Situation 3

Consider the placement of a billboard. The best placement is that point in the city where most cars will pass by.

For today, assume, for simplicity, that all locations are equally attractive as (final) locations for travelers.

Which centrality measure finds you an optimal location (with respect to this single objective)?

Facility location analysis

This is called an **absolute value** problem.

This is captured by *stress centrality*.

Stress centrality captures the vertex that is most visited, assuming travel is preferred by shortest routes.

Today's menu

Homeplay

Homeplay bonus

Bridging

Quiz

Pagerank

PageRank: alternative approach

In the tutorial, you calculated PageRank centrality based on diffusion of importance between webpages (the vertices), through their links (=the edges).

An alternative approach to calculate the importance of webpages is by *simulating the behavior* of websurfers.

1. start at a random webpage
2. click to the next page, following the outgoing links of the page (fully randomly)
3. continue clicking to the next pages for a while, then start over from step 1

Then:

PageRank = proportion of times a specific page is visited out of all pages visited.

PageRank: alternative approach

How to include damping:

1. start at a random page (all pages have the same probability)
2. with probability *damping*, follow a link to the next page (randomly picking one of the outgoing links from the current page)
3. with probability 1 - *damping*, jump to a random page in the network
4. PageRank = proportion of times you visit each page
5. repeat from step 1 until convergence

This simulates the behavior of a surfer getting "bored" with her current pages after a while and going to a different site for new inspiration/fun.

PageRank: alternative approach

Let's simulate

1. Download the file **PageRank_SNA4DS.html** to your computer, from the modules section on our Canvas page (modules, this week).
2. Save it to a location on your computer and open it from there.

PageRank: alternative approach

Start a simulation by FIRST clicking **setup** (to create a network) and then **step** (for a single step), or **go** (for continuous running). You can change the speed of the simulation using the speed slider on top.

Exercise:

- Compare the "random-surfer" and "diffusion" methods: **random surfer** is the method just explained, **diffusion** is the method explained in the tutorial.
    - How do their results compare?
    - Which one is faster?
- Play with the *damping factor* to how it affects the process and outcome.
    - Try at least values 0, .85, 1.
- Look at the tutorial networks ("Tutorial network" and "Tutorial network 2"), but also try the other networks. What do you notice?

10:00

PageRank: alternative approach

Some things to notice:

- In the “Example 1” network, five of the pages have no inbound links. Did you notice the effect?
- The diffusion model is a lot faster than the random surfer method. (but it has an advantage computationally for really large networks)
- When damping factor = 1, the methods do not always converge to the same results.
- When damping factor = 0, all pages get equal PageRank