Installing/loading the required libraries

Set the variable `INSTALL` to `TRUE` to install the libraries, set to `FALSE` once this has been done.

In [1]:
```
INSTALL = FALSE
if (INSTALL) {
    install.packages(c("dplyr", "readr", "sqldf", "igraph", "ggraph", "raster",
} else {
    library(dplyr)
    library(readr)
    library(igraph)
    library(ggraph)
    library(raster)
}
```

```
Attaching package: ‘dplyr’


The following objects are masked from ‘package:stats’:

    filter, lag


The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union


Attaching package: ‘igraph’


The following objects are masked from ‘package:dplyr’:

    as_data_frame, groups, union


The following objects are masked from ‘package:stats’:

    decompose, spectrum


The following object is masked from ‘package:base’:

    union


Loading required package: ggplot2

Loading required package: sp


Attaching package: ‘sp’
```

```
The following object is masked from ‘package:ggraph’:

    geometry


Attaching package: ‘raster’


The following object is masked from ‘package:dplyr’:

    select
```

Load the data

We load two data files: `air_travel_MB.csv` contains the network description and `airport_codes_CAN.csv` , which has some information about all airfields in Canada.

The data in air_travel_MB is pre-processed. It represents an extreme simplification of a much more complete dataset (Rows: 2,757,430; Columns: 19) giving, for a given time period, the total number of travellers between 3734 airports worldwide along trips including up to 5 intermediate stops.

What I did is to keep only the airports in Manitoba. Any airport outside of Manitoba is set as `RoW` (rest of the world). Also, the existing multi-leg trips within Manitoba are "decomposed": if the data had 10 people flying, say, YWG⟶YQD⟶YTH (Winnipeg to The Pas to Thompson), these would appear as 10 people flying from YWG to YQD and 10 people flying from YQD to YTH.

Flights to and from RoW are decomposed the same way as flights within Manitoba, increasing the number of trips between RoW and YWG. Indeed, 5 passengers flying RoW⟶YWG⟶YQD and 5 passengers flying RoW⟶YWG⟶YBR (Brandon) will show as 10 passengers flying RoW to YWG, together with 5 passengers flying YWG⟶YQD and 5 passengers flying YWG⟶YBR.

In [2]:
```
air_travel_MB <- read_delim("air_travel_MB.csv",
                     delim = ",", escape_double = FALSE,
                              trim_ws = TRUE)
airport_codes_MB <- read_csv("airport_codes_CAN.csv") %>%
  filter(ISO_Region == "CA-MB")
head(air_travel_MB)
head(airport_codes_MB)
```

```
Rows: 108 Columns: 3
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (2): orig, dest
dbl (1): vol


ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this messag
e.
```

```
Rows: 2792 Columns: 18

── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (12): Identifier, Type, Name, ISO_Country_Code, ISO_Region, Municipality...
dbl  (4): Airport_ID, Latitude, Longitude, Elevation_Ft
lgl  (2): Continent, Is_Scheduled_Service


ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this messag
e.
```

A tibble: 6 × 3

orig	dest	vol
<chr>	<chr>	<dbl>
RoW	YBR	2002
RoW	YTH	18
RoW	YWG	157323
RoW	YYQ	1474
XLB	RoW	766
XLB	XTL	114

Airport_ID	Identifier	Type	Name	Latitude	Longitude	Elevation_Ft	Continent	ISO_Country_C
<dbl>	<chr>	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<lgl>	<c
35391	CA-0006	closed	RCAF Station Carberry	49.87210	-99.39730	NA	NA	
39668	CA-0022	closed	Anama Bay-Dauphin River Airport	51.96267	-98.13684	NA	NA	
39677	CA-0031	closed	Austin Airport	49.93333	-98.91666	NA	NA	
39686	CA-0040	closed	Beausejour Airport	50.13697	-96.20822	NA	NA	
39694	CA-0048	closed	Bethany Airport	50.35000	-99.75000	NA	NA	
39696	CA-0050	closed	Bird Airport	56.50000	-94.21667	NA	NA	

Data wrangling remark

Note that there is an alternative to `dplyr` (the package that allows us to use the pipe `%>%`); it is indeed possible to treat data frames as SQL tables and run SQL queries on them. This may make

more sense to some of you. For illustration, to create the table of movements, I used the following command (which can run, since it is not destructive of what we have already done).

In [3]:
```
query =
"SELECT orig,dest,SUM(vol) AS vol
FROM air_travel_MB
GROUP BY orig,dest
ORDER BY orig,dest"

air_travel_MB = sqldf::sqldf(query)
```

Finish preparing the data: we want to record the latitude and longitudes for all the airports in the `air_travel_MB` data set. Note that we give `RoW` a location: we will show it as an additional vertex in the graph. (I picked somewhere close to but not in Manitoba.)

In [4]:
```
IATA_codes_MB_in_data = data.frame(IATA = sort(unique(union(air_travel_MB$orig,
                                                            air_travel_MB$dest))
lon = c()
lat = c()
for (i in 1:dim(IATA_codes_MB_in_data)[1]) {
  idx = which(airport_codes_MB$IATA_Code == IATA_codes_MB_in_data$IATA[i])
  if (length(idx) == 0) {
    lon = c(lon, -90)
    lat = c(lat, 49.5)
  } else {
    lon = c(lon, airport_codes_MB$Longitude[idx])
    lat = c(lat, airport_codes_MB$Latitude[idx])
  }
}
IATA_codes_MB_in_data$lon = lon
IATA_codes_MB_in_data$lat = lat

head(IATA_codes_MB_in_data)
```

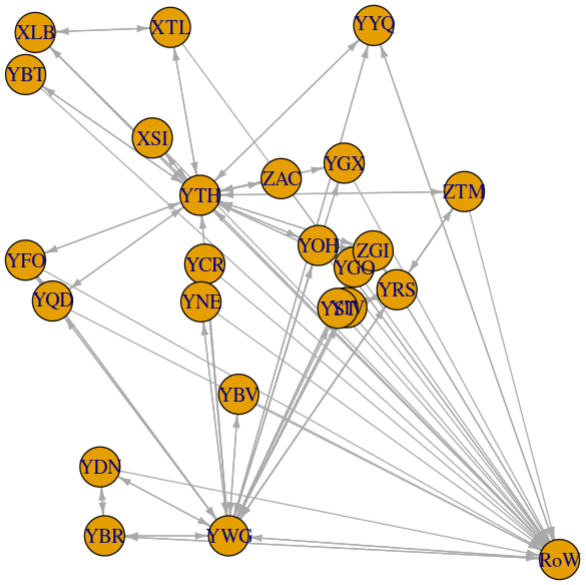
A data.frame: 6 × 3

	IATA	lon	lat
	<chr>	<dbl>	<dbl>
1	RoW	-90.0000	49.5000
2	XLB	-101.4690	58.6175
3	XSI	-98.9072	56.7928
4	XTL	-98.5122	58.7061
5	YBR	-99.9519	49.9100
6	YBT	-101.6790	57.8894

We are now in a position to make the graph and plot it in simple form. We add the volume as the weight of the arcs.

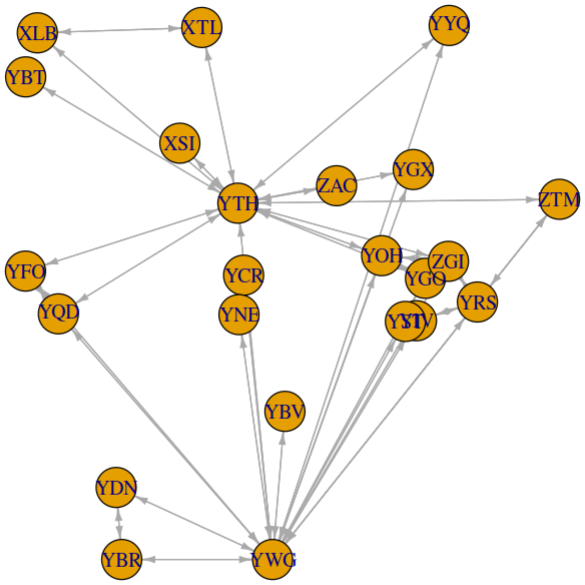
In [5]:
```
G = graph_from_data_frame(air_travel_MB)
V(G)$x = lon
V(G)$y = lat
```

```
plot(G, edge.arrow.size = 0.5, edge.arrow.width = 0.75)
```



We also make a subgraph without `RoW` to focus on the properties really specific to Manitoba.

In [6]:
```
G_MB = induced_subgraph(G, V(G)$name[V(G)$name != "RoW"])
plot(G_MB, edge.arrow.size = 0.5, edge.arrow.width = 0.75)
```



Making nicer plots

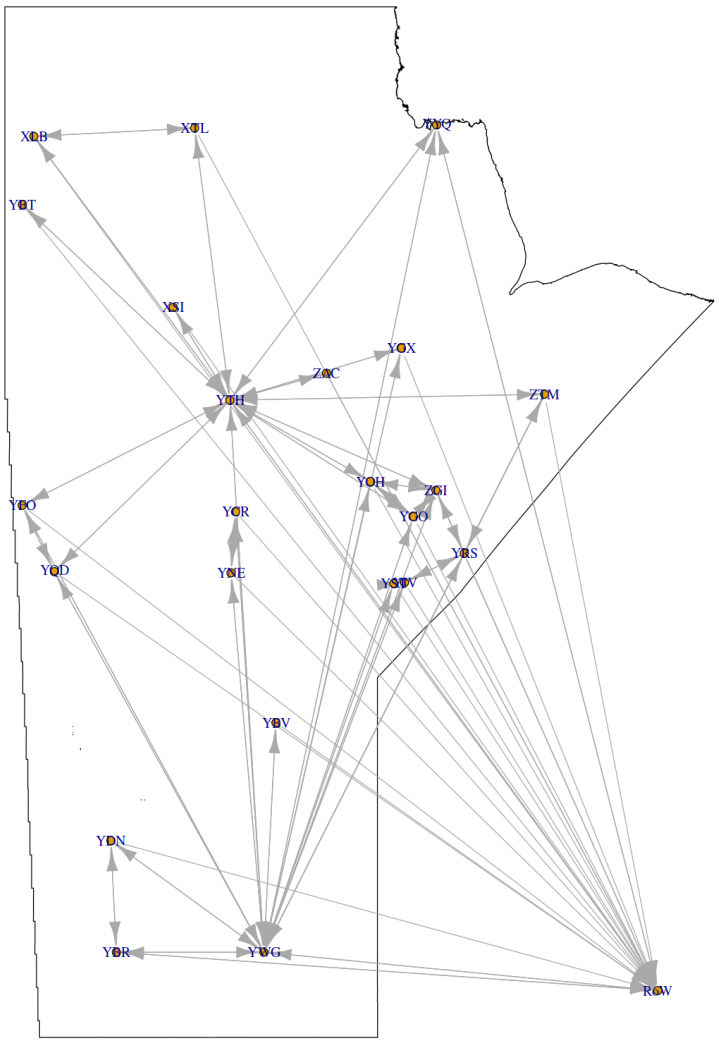
Before we explore the properties of the graph, let us spend a little time on visualisation. This is not to be neglected: nice plots help more than you think.

The next piece of code does the following:

- Redefine the output size (does it once for all, need to modify if need be)
- Uses the library `raster` to read in the shape file for Canada
- Selects Manitoba from the Canada shape file
- Plots Manitoba
- Plots the graph

In [7]:

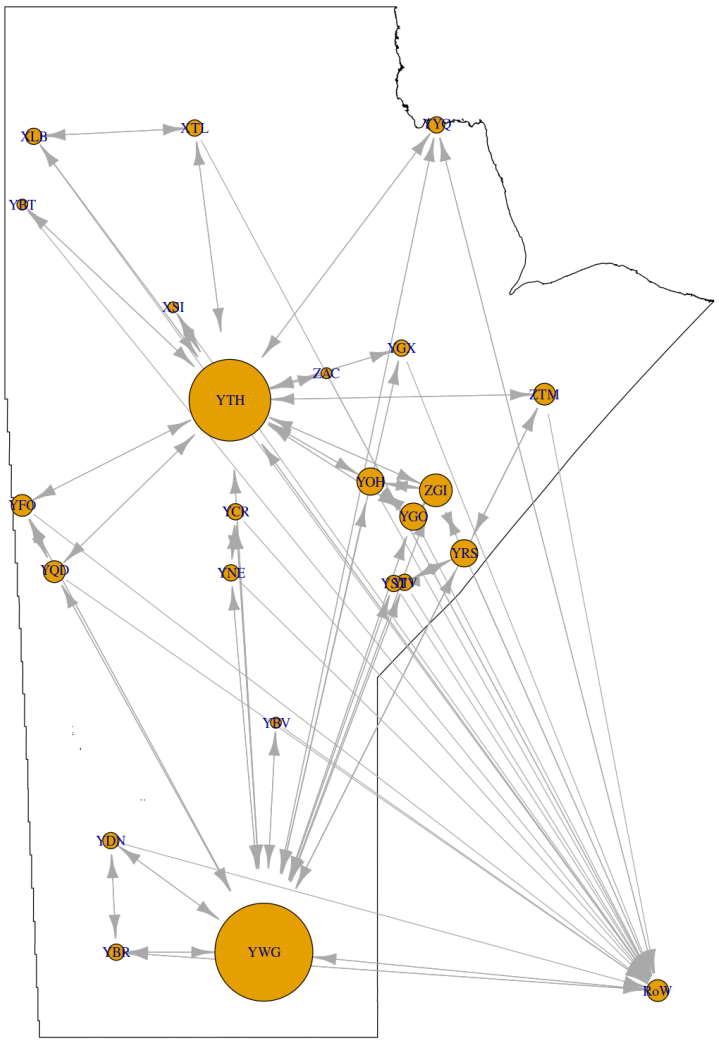
```
options(repr.plot.width=15, repr.plot.height=15)
Canada <- getData('GADM', country='CAN', level=1)
MB <- Canada[Canada$NAME_1 == "Manitoba",]
plot(MB)
plot(G, add = TRUE, rescale = FALSE,
    edge.arrow.size = 0.5, edge.arrow.width = 0.75)
```



Let us add the degree information as the size of the vertices.

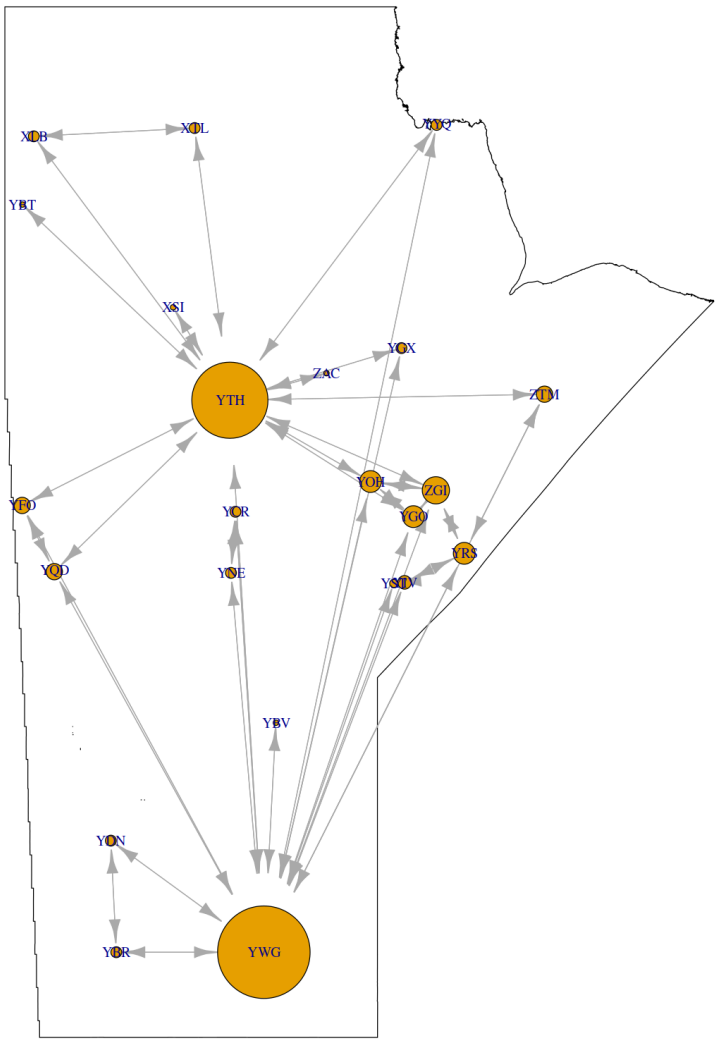
In [8]:

```
plot(MB)
V(G)$size = degree(G, mode = "out")*10
plot(G, add = TRUE, rescale = FALSE,
    edge.arrow.size = 0.5, edge.arrow.width = 0.75)
```



In [9]:

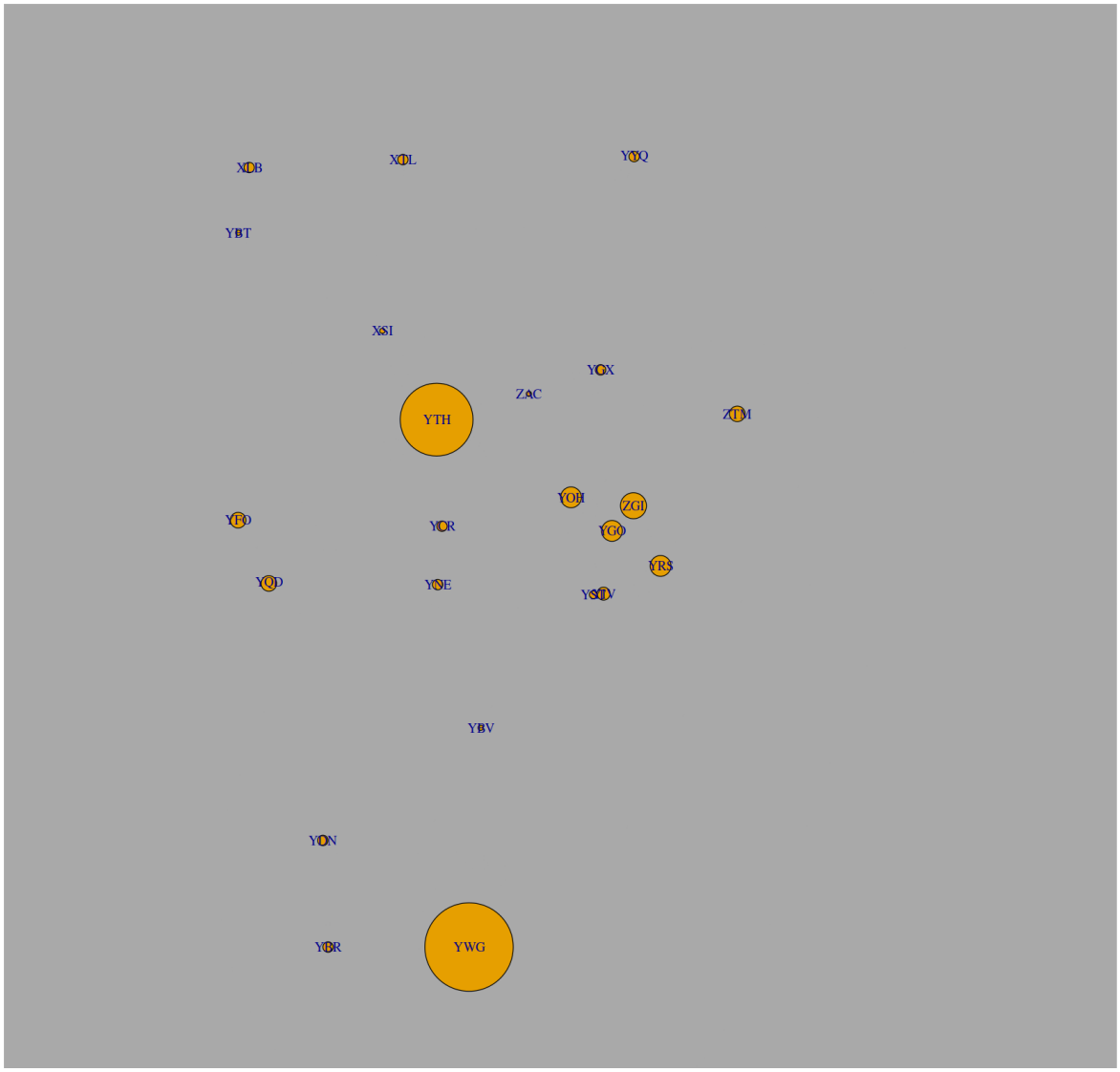
```
plot(MB)
V(G_MB)$size = degree(G_MB)*5
plot(G_MB, add = TRUE, rescale = FALSE,
    edge.arrow.size = 0.5, edge.arrow.width = 0.75)
```



Now let us take a look at the arcs. Maybe change their thickness to reflect the volumes along the arcs?

In [10]:

```
plot(MB)
V(G_MB)$size = degree(G_MB)*5
plot(G_MB, add = TRUE, rescale = FALSE, edge.width = E(G_MB)$vol,
    edge.arrow.size = 0.5, edge.arrow.width = 0.75)
```



My guess: some of the arcs are so massive that this is all we can see. So let us renormalise the values of the weights. First, what do they look like now?

In [11]:
```
sort(E(G)$vol)
```

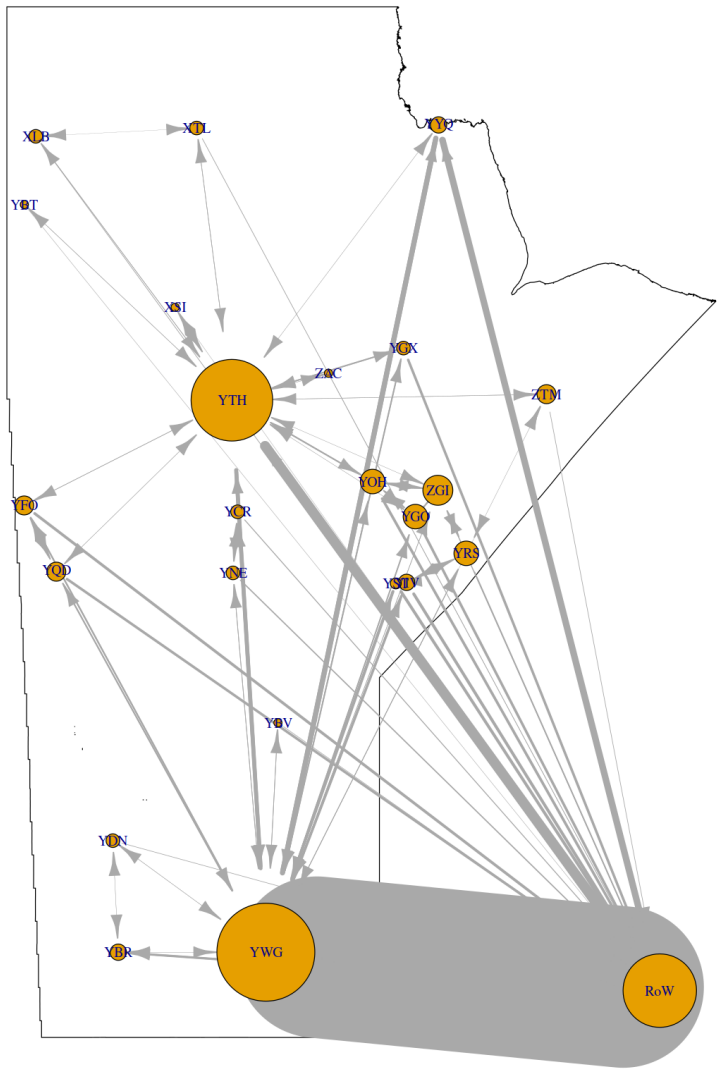
18 · 47 · 48 · 48 · 50 · 98 · 99 · 114 · 116 · 174 · 176 · 208 · 208 · 212 · 242 · 276 · 278 · 279 · 288 · 288 · 288 · 292 · 292 · 309 · 337 · 337 · 343 · 344 · 364 · 366 · 368 · 369 · 372 · 383 · 399 · 401 · 401 · 406 · 447 · 447 · 449 · 451 · 451 · 462 · 529 · 565 · 567 · 582 · 592 · 644 · 650 · 652 · 652 · 682 · 684 · 694 · 696 · 701 · 702 · 713 · 714 · 758 · 766 · 780 · 806 · 848 · 854 · 1036 · 1050 · 1167 · 1172 · 1242 · 1270 · 1279 · 1342 · 1349 · 1394 · 1462 · 1464 · 1474 · 1485 · 1543 · 1565 · 1607 · 1638 · 1750 · 1857 · 2002 · 2230 · 2518 · 2582 · 2877 · 2899 · 2910 · 2924 · 3037 · 3157 · 3355 · 3355 · 3365 · 4246 · 4571 · 5770 · 5789 · 7430 · 11464 · 157323 · 183256

So, yes, clearly a bit too much variation. If we scale linearly by multiplying by a constant, we will most likely lose the information on the small weights. Just to check that we are intuiting correctly,

though, let us try once doing this.

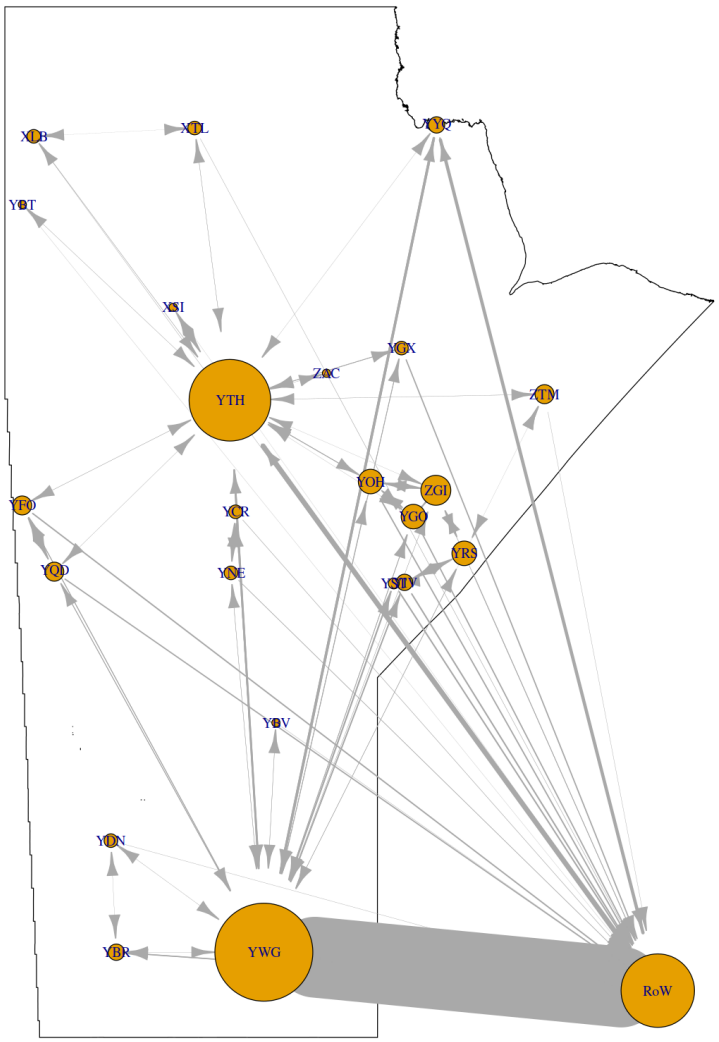
In [12]:

```
plot(MB)
V(G)$size = degree(G)*5
plot(G, add = TRUE, rescale = FALSE, edge.width = E(G)$vol*0.001,
    edge.arrow.size = 0.5, edge.arrow.width = 0.75)
```



In [13]:

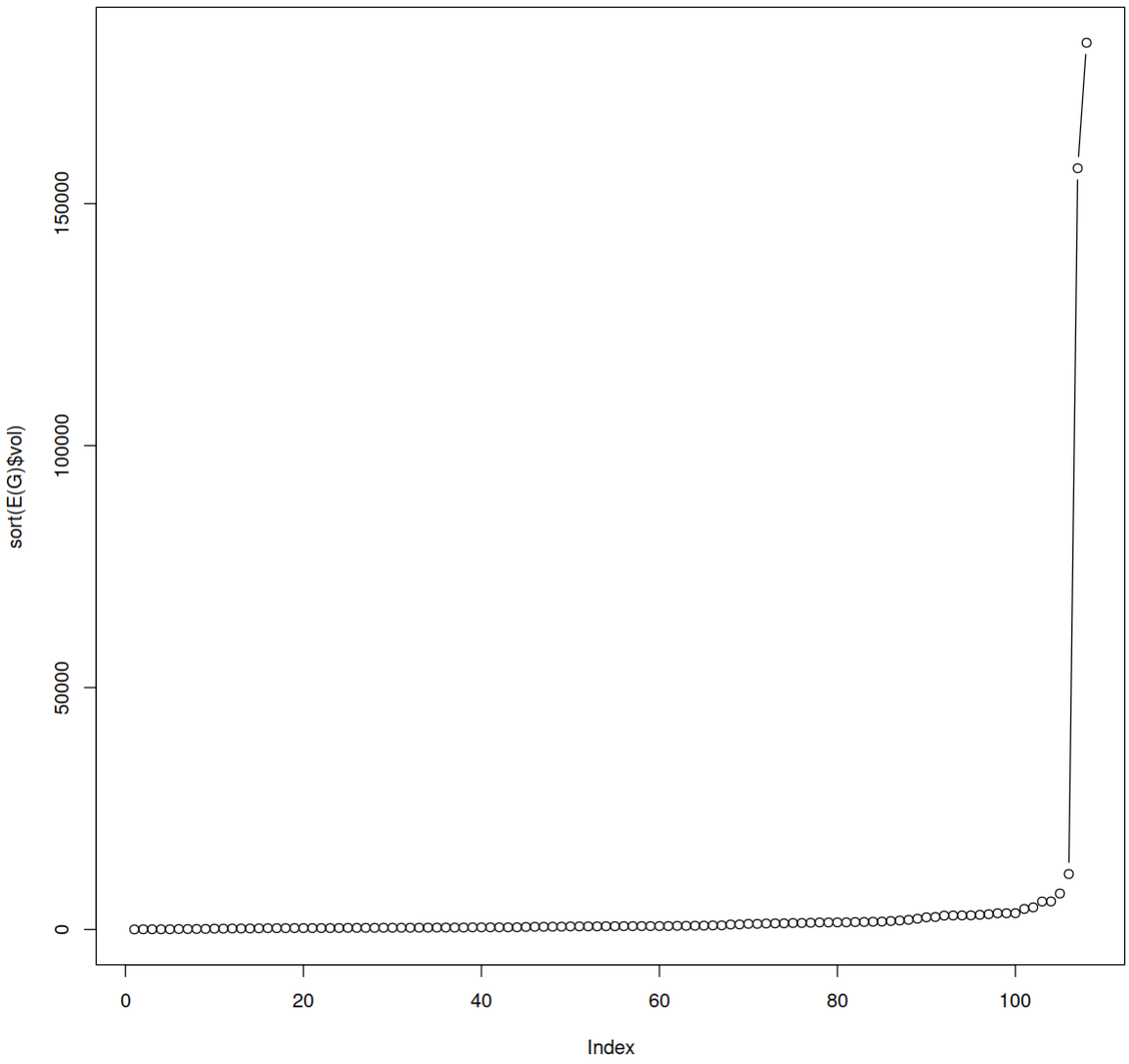
```
plot(MB)
V(G)$size = degree(G)*5
plot(G, add = TRUE, rescale = FALSE, edge.width = E(G)$vol*0.0005,
    edge.arrow.size = 0.5, edge.arrow.width = 0.75)
```



What we suspected, indeed: to make the arcs with the largest volume not too large, we had to essentially get rid of the weight of the arcs with small volumes. So what we want to do is to process the values through a function that would make the smaller values larger relative to the larger ones or the larger values closer to the smaller one, then scale the result. For this, it is good to look at what the values look like.

In [14]:

```
options(repr.plot.width=10, repr.plot.height=10)
plot(sort(E(G)$vol), type = "b")
```



One way around this could be to use a so-called sigmoid function. This is a function of the following form:

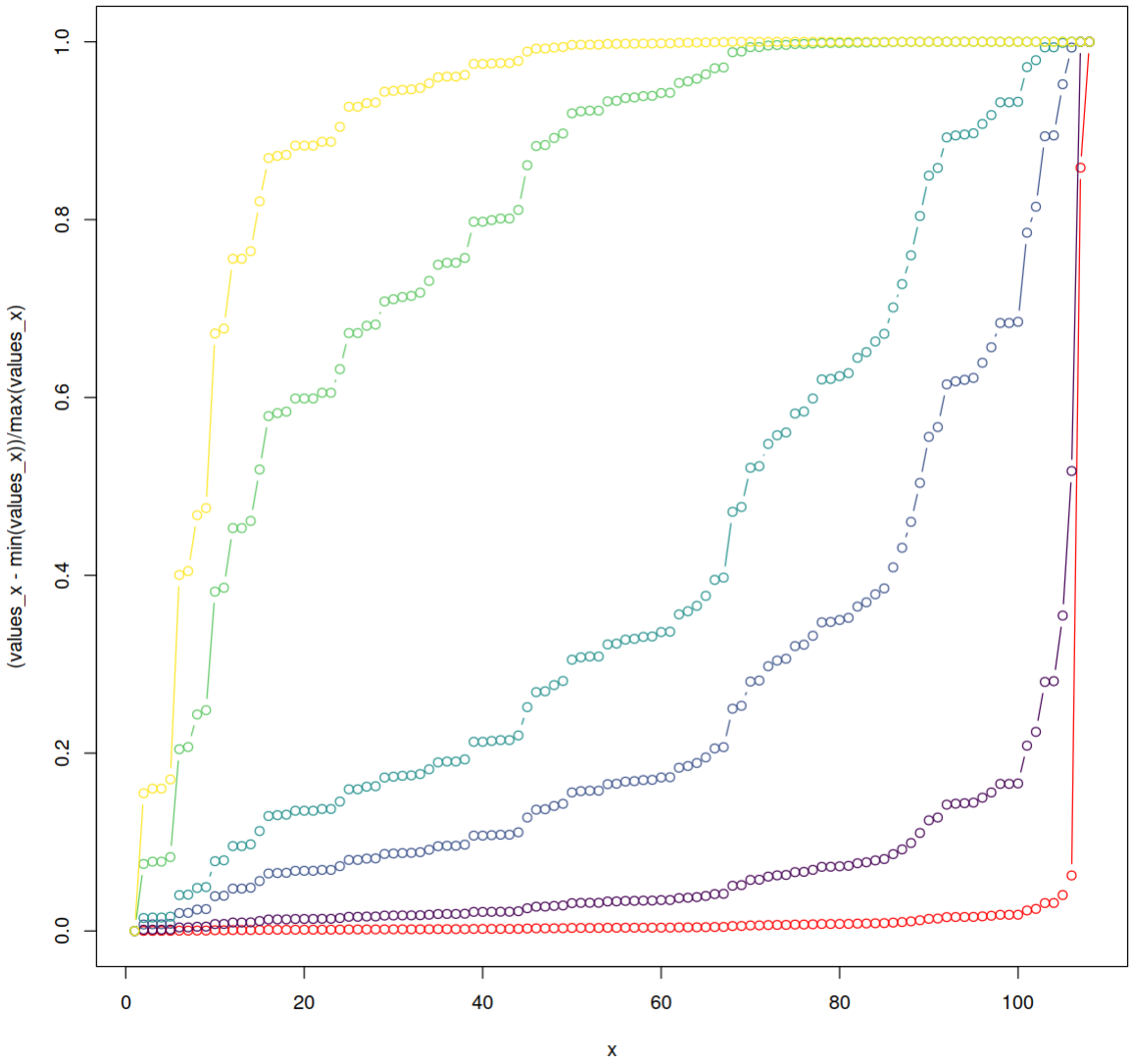
$$f(x) = \frac{1}{1+e^{-cx}},$$

where $\mathbb{R} \ni c > 0$ is a parameter. This function has the following properties: $\lim_{x\to-\infty} f(x) = 0$ and $\lim_{x\to\infty} f(x) = 1$. By playing with the value of c, one can set the beahviour of f. Let us see how different values of c influence the value of the points. (The `viridis` palette used starts with yellow and goes towards green and then purple.)

In [15]:

```
sigmoid = function(x, c) {
    return(1/(1+exp(-c*x)))
}

values_c = c(0.0001, 0.0005, 0.001, 0.005, 0.01)
x = 1:length(E(G))
values_x = sort(E(G)$vol)
my_palette = viridis::viridis(length(values_c))
```

```
# First, we plot the "data" (scaled to the [0,1] range)
plot(x, (values_x-min(values_x))/max(values_x),
     type = "b", col = "red",
     ylim = (range(values_x)-min(values_x))/max(values_x))
for (c in values_c) {
    idx = which(c == values_c) # Find which position in the values_c vector, for
    tmp = sigmoid(values_x, c) # Let us rescale the result to between 0 and 1
    tmp = tmp-min(tmp)
    tmp = tmp/max(tmp)
    lines(x, tmp, type = "b", col = my_palette[idx])
}
```

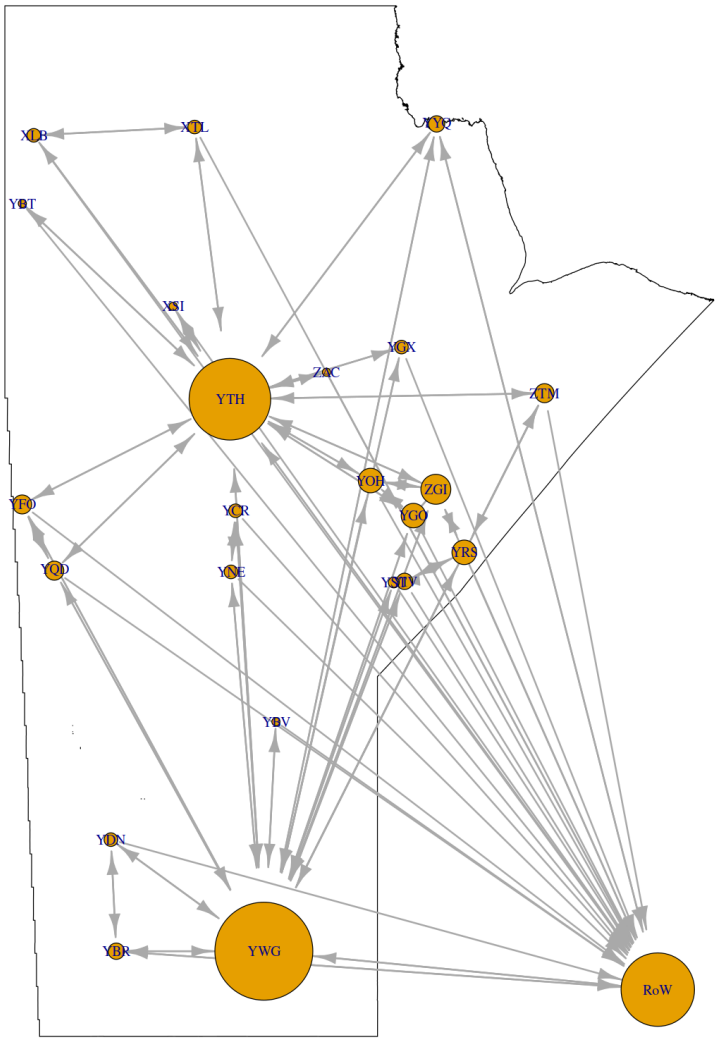


That last one looks quite nice. We most likely will have to multiply by a factor, so we compute the values in advance.

In [16]:

```
options(repr.plot.width=15, repr.plot.height=15)
edge_widths = 2*sigmoid(E(G)$vol, 0.1)

plot(MB)
V(G)$size = degree(G)*5
```

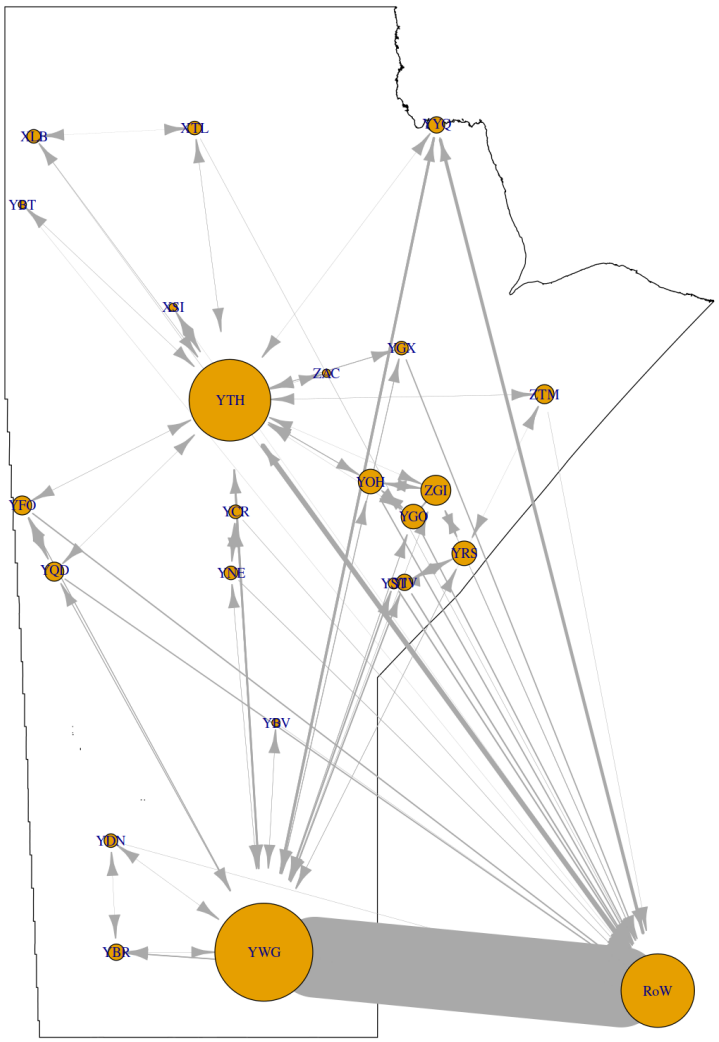
```
plot(G, add = TRUE, rescale = FALSE,
     edge.width = edge_widths, type = "b",
    edge.arrow.size = 0.5, edge.arrow.width = 0.75)
```



Well, after all this, I am not convinced we achieved between than we had done above but just a linear scaling. So when we want a a good plot, we'll come back to that solution.

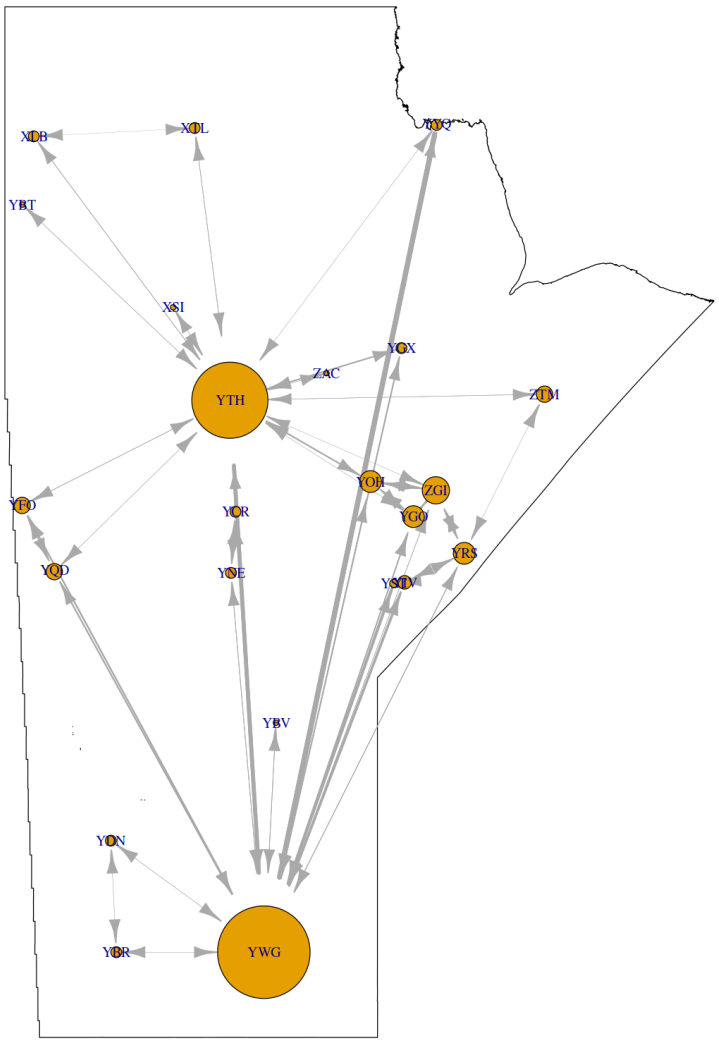
In [17]:

```
plot(MB)
V(G)$size = degree(G)*5
plot(G, add = TRUE, rescale = FALSE, edge.width = E(G)$vol*0.0005,
    edge.arrow.size = 0.5, edge.arrow.width = 0.75)
```



In [18]:

```
plot(MB)
V(G_MB)$size = degree(G_MB)*5
plot(G_MB, add = TRUE, rescale = FALSE,
     edge.width = E(G_MB)$vol*0.001,
    edge.arrow.size = 0.5, edge.arrow.width = 0.75)
```



Eccentricities and related measures

In [19]:
```
distances(G_MB)
```

A matrix: 23 × 23 of type dbl

	XLB	XSI	XTL	YBR	YBT	YBV	YCR	YDN	YFO	YGO	⋯	YOH	YQD	YRS	YST	YTH	Y
XLB	0	2	1	3	2	3	3	3	2	2	⋯	2	2	3	3	1	
XSI	2	0	2	3	2	3	3	3	2	2	⋯	2	2	3	3	1	
XTL	1	2	0	3	2	3	3	3	2	2	⋯	2	2	3	3	1	
YBR	3	3	3	0	3	2	2	1	2	2	⋯	2	2	2	2	2	
YBT	2	2	2	3	0	3	3	3	2	2	⋯	2	2	3	3	1	
YBV	3	3	3	2	3	0	2	2	2	2	⋯	2	2	2	2	2	
YCR	3	3	3	2	3	2	0	2	2	2	⋯	2	2	2	2	2	

	XLB	XSI	XTL	YBR	YBT	YBV	YCR	YDN	YFO	YGO	⋯	YOH	YQD	YRS	YST	YTH	Y
YDN	3	3	3	1	3	2	2	0	2	2	⋯	2	2	2	2	2	
YFO	2	2	2	2	2	2	2	2	0	2	⋯	2	1	2	2	1	
YGO	2	2	2	2	2	2	2	2	2	0	⋯	1	2	2	2	1	
YGX	2	2	2	2	2	2	2	2	2	2	⋯	2	2	2	2	1	
YIV	3	3	3	2	3	2	2	2	2	2	⋯	2	2	1	1	2	
YNE	3	3	3	2	3	2	1	2	2	2	⋯	2	2	2	2	2	
YOH	2	2	2	2	2	2	2	2	2	1	⋯	0	2	2	2	1	
YQD	2	2	2	2	2	2	2	2	1	2	⋯	2	0	2	2	1	
YRS	3	3	3	2	3	2	2	2	2	2	⋯	2	2	0	2	2	
YST	3	3	3	2	3	2	2	2	2	2	⋯	2	2	2	0	2	
YTH	1	1	1	2	1	2	2	2	1	1	⋯	1	1	2	2	0	
YWG	2	2	2	1	2	1	1	1	1	1	⋯	1	1	1	1	1	
YYQ	2	2	2	2	2	2	2	2	2	2	⋯	2	2	2	2	1	
ZAC	2	2	2	3	2	3	3	3	2	2	⋯	2	2	3	3	1	
ZGI	2	2	2	2	2	2	2	2	2	1	⋯	1	2	1	2	1	
ZTM	2	2	2	2	2	2	2	2	2	2	⋯	2	2	1	2	1	

In [20]:

```
distances(G_MB, mode = "out")
```

A matrix: 23 × 23 of type dbl

	XLB	XSI	XTL	YBR	YBT	YBV	YCR	YDN	YFO	YGO	⋯	YOH	YQD	YRS	YST	YTH	Y
XLB	0	2	1	3	2	3	3	3	2	2	⋯	2	2	3	3	1	
XSI	2	0	2	3	2	3	3	3	2	2	⋯	2	2	3	3	1	
XTL	1	2	0	3	2	3	3	3	2	2	⋯	2	2	3	3	1	
YBR	3	3	3	0	3	2	2	1	2	2	⋯	2	2	2	2	2	
YBT	2	2	2	3	0	3	3	3	2	2	⋯	2	2	3	3	1	
YBV	3	3	3	2	3	0	2	2	2	2	⋯	2	2	2	2	2	
YCR	3	3	3	2	3	2	0	2	2	2	⋯	2	2	2	2	2	
YDN	3	3	3	1	3	2	2	0	2	2	⋯	2	2	2	2	2	
YFO	2	2	2	2	2	2	2	2	0	2	⋯	2	1	2	2	1	
YGO	2	2	2	2	2	2	2	2	2	0	⋯	1	2	2	2	1	
YGX	2	2	2	2	2	2	2	2	2	2	⋯	2	2	2	2	1	
YIV	3	3	3	2	3	2	2	2	2	2	⋯	2	2	1	2	2	
YNE	3	3	3	2	3	2	1	2	2	2	⋯	2	2	2	2	2	
YOH	2	2	2	2	2	2	2	2	2	1	⋯	0	2	2	2	1	
YQD	2	2	2	2	2	2	2	2	1	2	⋯	2	0	2	2	1	

	XLB	XSI	XTL	YBR	YBT	YBV	YCR	YDN	YFO	YGO	⋯	YOH	YQD	YRS	YST	YTH	Y
YRS	3	3	3	2	3	2	2	2	2	2	⋯	2	2	0	2	2	
YST	3	3	3	2	3	2	2	2	2	2	⋯	2	2	2	0	2	
YTH	1	1	1	2	1	2	2	2	1	1	⋯	1	1	2	2	0	
YWG	2	2	2	1	2	1	1	1	1	1	⋯	1	1	1	1	1	
YYQ	2	2	2	2	2	2	2	2	2	2	⋯	2	2	2	2	1	
ZAC	2	2	2	3	2	3	3	3	2	2	⋯	2	2	3	3	1	
ZGI	2	2	2	2	2	2	2	2	2	1	⋯	1	2	1	2	1	
ZTM	2	2	2	2	2	2	2	2	2	2	⋯	2	2	1	2	1	

In [21]:

```
eccentricity(G_MB)
eccentricity(G_MB, mode = "in")
eccentricity(G_MB, mode = "out")
writeLines("Radius (all/in/out):")
radius(G_MB)
radius(G_MB, mode = "in")
radius(G_MB, mode = "out")
writeLines("Diameter (all/in/out:")
diameter(G_MB)
max(distances(G_MB, mode = "in"))
max(distances(G_MB, mode = "out"))
```

XLB: 3 **XSI:** 3 **XTL:** 3 **YBR:** 3 **YBT:** 3 **YBV:** 3 **YCR:** 3 **YDN:** 3 **YFO:** 2 **YGO:** 2 **YGX:** 2 **YIV:** 3 **YNE:** 3 **YOH:** 2 **YQD:** 2 **YRS:** 3 **YST:** 3 **YTH:** 2 **YWG:** 2 **YYQ:** 2 **ZAC:** 3 **ZGI:** 2 **ZTM:** 2

XLB: 3 **XSI:** 3 **XTL:** 3 **YBR:** 3 **YBT:** 3 **YBV:** 3 **YCR:** 3 **YDN:** 3 **YFO:** 2 **YGO:** 2 **YGX:** 2 **YIV:** 3 **YNE:** 3 **YOH:** 2 **YQD:** 2 **YRS:** 3 **YST:** 3 **YTH:** 2 **YWG:** 2 **YYQ:** 2 **ZAC:** 3 **ZGI:** 2 **ZTM:** 2

XLB: 3 **XSI:** 3 **XTL:** 3 **YBR:** 3 **YBT:** 3 **YBV:** 3 **YCR:** 3 **YDN:** 3 **YFO:** 2 **YGO:** 2 **YGX:** 2 **YIV:** 3 **YNE:** 3 **YOH:** 2 **YQD:** 2 **YRS:** 3 **YST:** 3 **YTH:** 2 **YWG:** 2 **YYQ:** 2 **ZAC:** 3 **ZGI:** 2 **ZTM:** 2

```
Radius (all/in/out):
```

2

2

2

```
Diameter (all/in/out:
```

3

3

3

Central points, centre, periphery

For simplicity, I only show the out-degree case. And just to show how easily it's done: let's colour the vertices in the centre red and those in the periphery blue.

In [22]:

```
e_out = eccentricity(G_MB, mode = "out")
writeLines("Central points:")
```

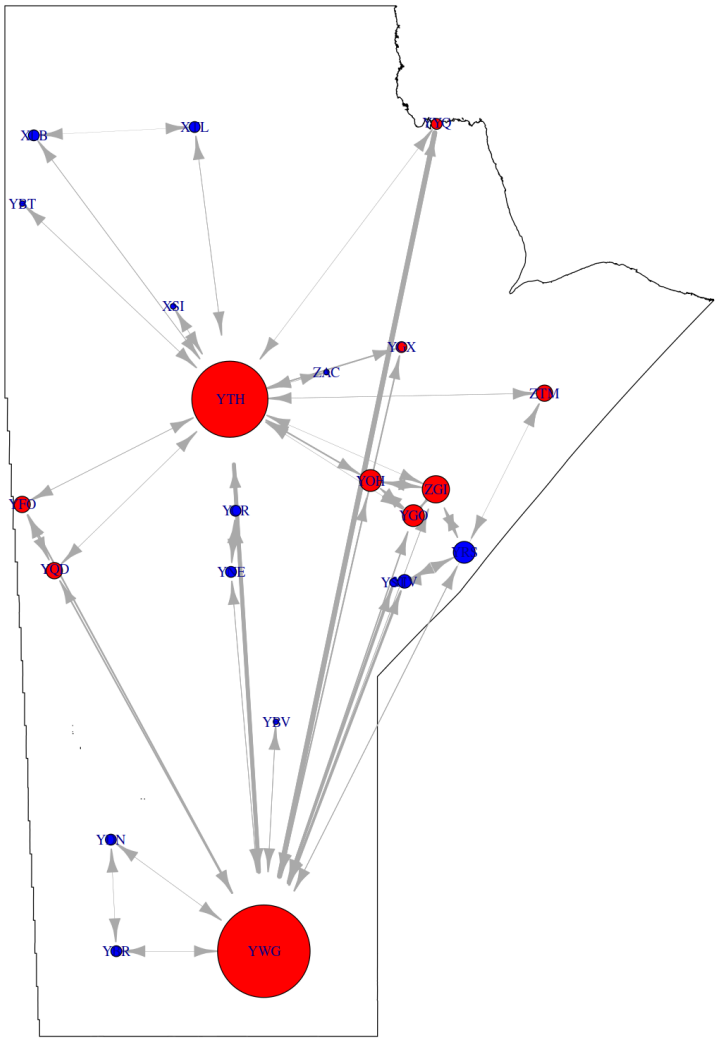
```
V(G_MB)[which(e_out == min(e_out))]$name
V(G_MB)[which(e_out == min(e_out))]$color = "red"
writeLines("Periphery:")
V(G_MB)[which(e_out == max(e_out))]$name
V(G_MB)[which(e_out == max(e_out))]$color = "blue"

plot(MB)
V(G_MB)$size = degree(G_MB)*5
plot(G_MB, add = TRUE, rescale = FALSE,
     edge.width = E(G_MB)$vol*0.001,
    edge.arrow.size = 0.5, edge.arrow.width = 0.75)
```

```
Central points:
```

'YFO' · 'YGO' · 'YGX' · 'YOH' · 'YQD' · 'YTH' · 'YWG' · 'YYQ' · 'ZGI' · 'ZTM'

```
Periphery:
```

'XLB' · 'XSI' · 'XTL' · 'YBR' · 'YBT' · 'YBV' · 'YCR' · 'YDN' · 'YIV' · 'YNE' · 'YRS' · 'YST' · 'ZAC'



Girth

Recall that `girth` in `igraph` assumes the graph is undirected.

In [23]:
```
girth(G)
girth(G_MB)
```

```
$girth
[1] 3

$circle
+ 3/24 vertices, named, from d17519f:
[1] XLB RoW XTL
$girth
[1] 3

$circle
+ 3/23 vertices, named, from a72bb96:
[1] XTL XLB YTH
```

Density

What fraction of possible arcs/edges are present?

In [24]:
```
edge_density(G)
edge_density(G_MB)
```

0.195652173913043

0.160079051383399

Connectedness and related

Connectedness

In [25]:
```
is.connected(G_MB, "weak")
is.connected(G_MB, "strong")
```

TRUE

TRUE

So there is a single strongly connected component.

Articulation points

The function assumes the graph is undirected, even in the directed case.

In [26]:
```
articulation_points(G_MB)
```

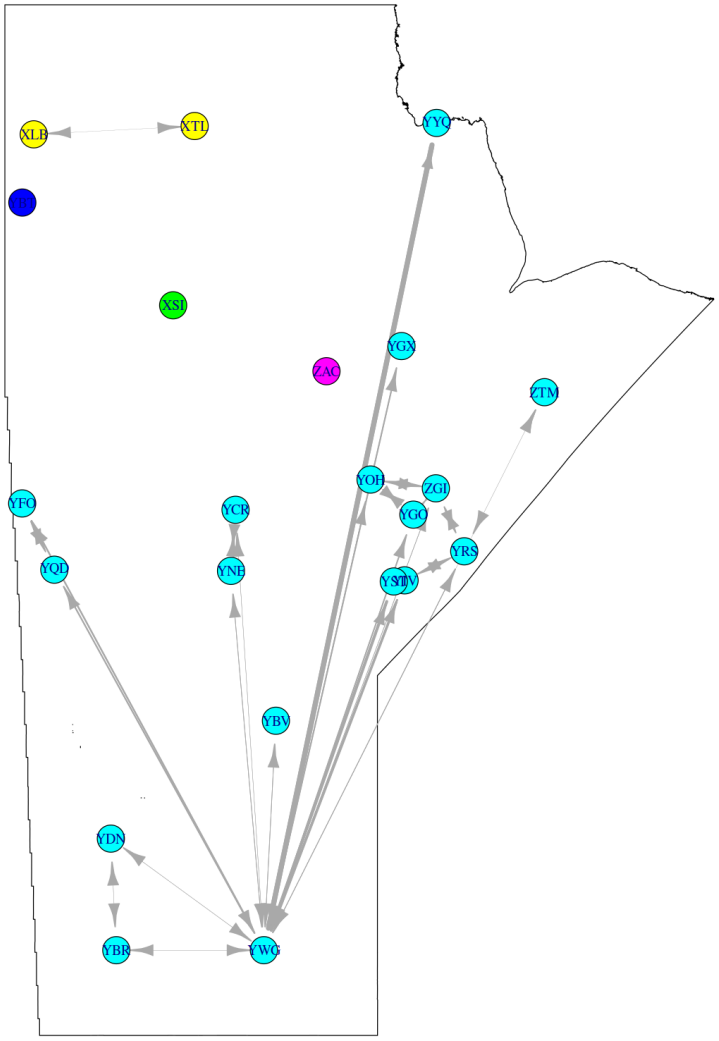
```
+ 2/23 vertices, named, from a72bb96:
[1] YTH YWG
```

So removing Winnipeg or Thompson disconnects the graph.

In [27]:

```
G_disconnected = induced_subgraph(G_MB, V(G_MB)$name[V(G_MB)$name != "YTH"])
is_connected(G_disconnected)

comps <- components(G_disconnected)$membership
colbar <- rainbow(max(comps)+1)
V(G_disconnected)$color <- colbar[comps+1]

plot(MB)
V(G_disconnected)$size = rep(50,length(V(G_disconnected)))
plot(G_disconnected, add = TRUE, rescale = FALSE,
     edge.width = E(G_disconnected)$vol*0.001,
    edge.arrow.size = 0.5, edge.arrow.width = 0.75)
```

FALSE

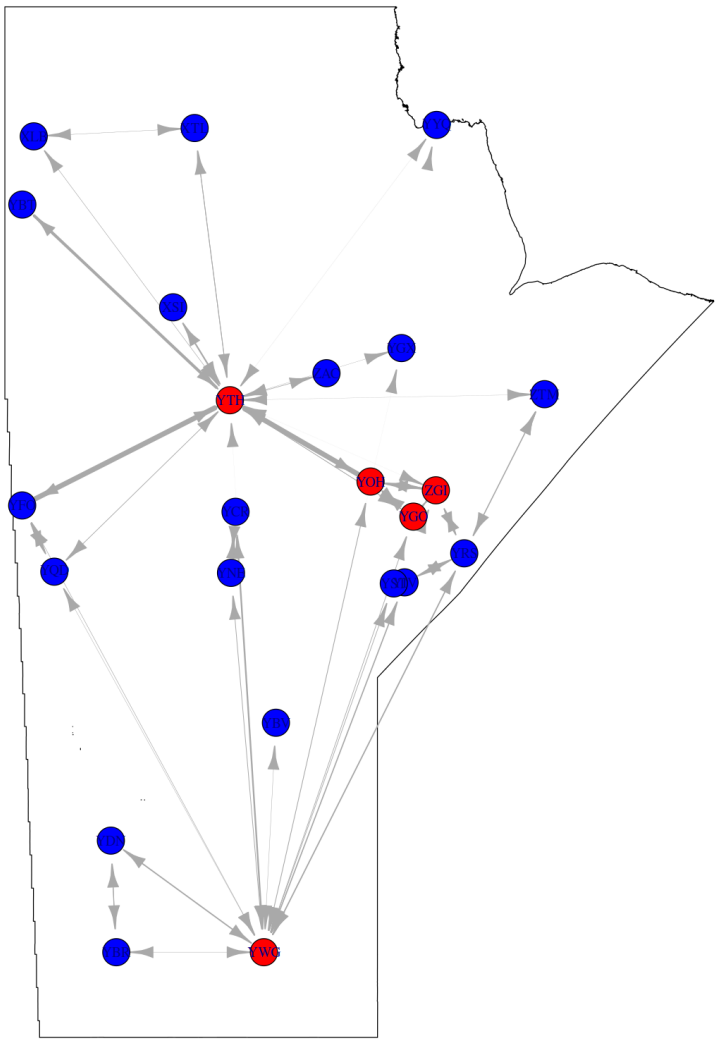


Cliques

In [28]:

```
lc = largest_cliques(G_MB)
lc
vert_largest_clique = lc[[1]]

V(G_MB)$color = "blue"
V(G_MB)[vert_largest_clique]$color = "red"

plot(MB)
V(G_MB)$size = rep(50,length(V(G_MB)))
plot(G_MB, add = TRUE, rescale = FALSE,
     edge.width = E(G_disconnected)$vol*0.001,
    edge.arrow.size = 0.5, edge.arrow.width = 0.75)
```

```
Warning message in largest_cliques(G_MB):
“At cliques.c:1125 :directionality of edges is ignored for directed graphs”
[[1]]
+ 5/23 vertices, named, from a72bb96:
[1] YWG YTH YGO YOH ZGI
```



Centrality

Degree-centrality

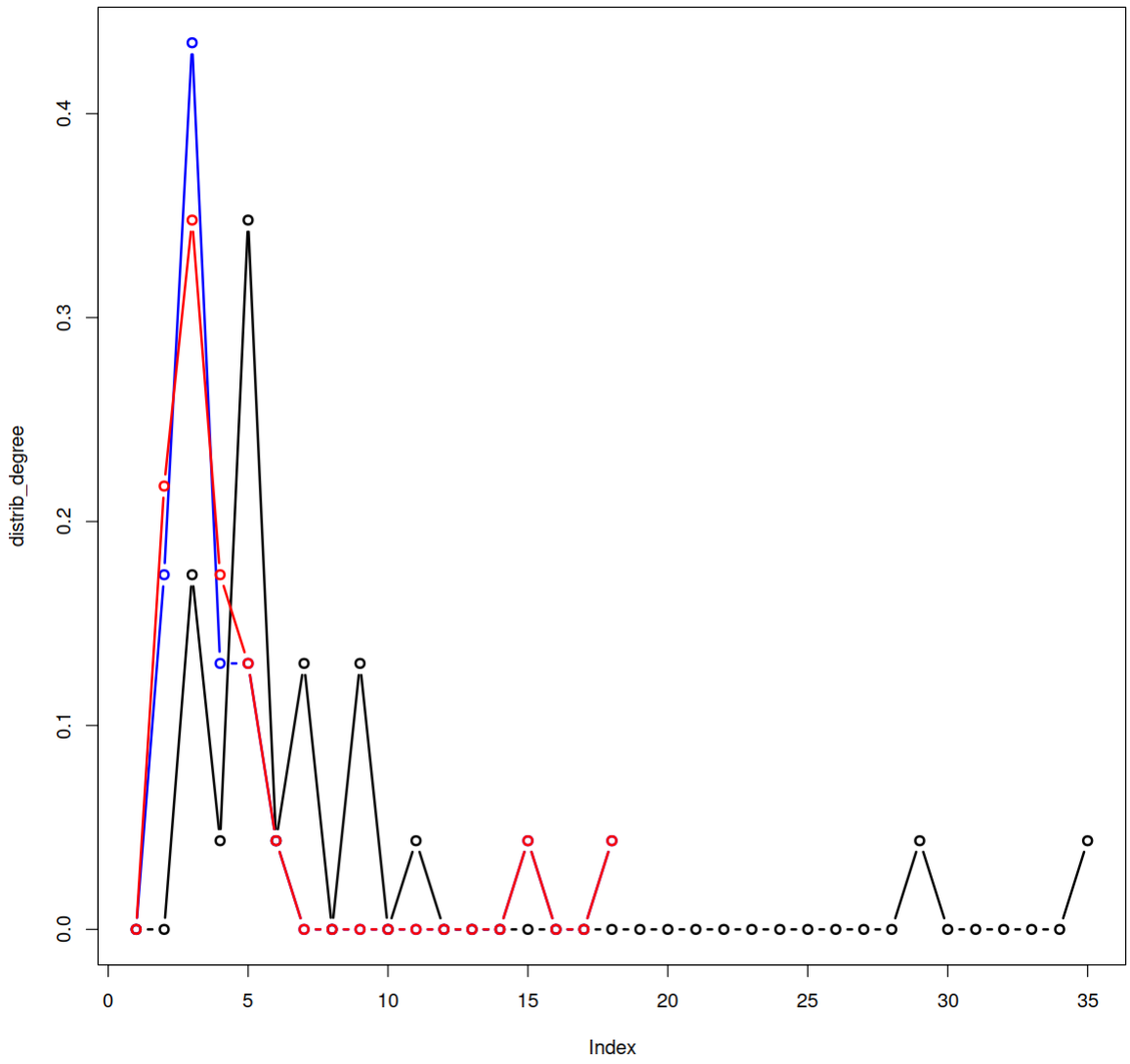
In [29]:
```
degree(G_MB, mode = "out")
degree(G_MB, mode = "in")
degree(G_MB)
distrib_outdegree = degree_distribution(G, mode = "out")
distrib_indegree = degree_distribution(G, mode = "in")
distrib_degree = degree_distribution(G)
```

XLB: 2 **XSI:** 1 **XTL:** 2 **YBR:** 2 **YBT:** 1 **YBV:** 1 **YCR:** 2 **YDN:** 2 **YFO:** 3 **YGO:** 4 **YGX:** 2 **YIV:** 2 **YNE:** 2 **YOH:** 4 **YQD:** 3 **YRS:** 4 **YST:** 2 **YTH:** 14 **YWG:** 17 **YYQ:** 2 **ZAC:** 1 **ZGI:** 5 **ZTM:** 3

XLB: 2 **XSI:** 1 **XTL:** 2 **YBR:** 2 **YBT:** 1 **YBV:** 1 **YCR:** 2 **YDN:** 2 **YFO:** 3 **YGO:** 4 **YGX:** 2 **YIV:** 3 **YNE:** 2 **YOH:** 4 **YQD:** 3 **YRS:** 4 **YST:** 1 **YTH:** 14 **YWG:** 17 **YYQ:** 2 **ZAC:** 1 **ZGI:** 5 **ZTM:** 3

XLB: 4 **XSI:** 2 **XTL:** 4 **YBR:** 4 **YBT:** 2 **YBV:** 2 **YCR:** 4 **YDN:** 4 **YFO:** 6 **YGO:** 8 **YGX:** 4 **YIV:** 5 **YNE:** 4 **YOH:** 8 **YQD:** 6 **YRS:** 8 **YST:** 3 **YTH:** 28 **YWG:** 34 **YYQ:** 4 **ZAC:** 2 **ZGI:** 10 **ZTM:** 6

In [30]:
```
options(repr.plot.width=10, repr.plot.height=10)

distrib_outdegree = degree_distribution(G_MB, mode = "out")
distrib_indegree = degree_distribution(G_MB, mode = "in")
distrib_degree = degree_distribution(G_MB)

y_max = max(max(distrib_indegree, distrib_outdegree), distrib_degree)
plot(distrib_degree, type = "b", lwd = 2, col = "black", ylim = c(0, y_max))
points(distrib_outdegree, type = "b", lwd = 2, col = "blue")
points(distrib_indegree, type = "b", lwd = 2, col = "red")
```



K-nearest neighbours

For info, the return values of the function `knn` are

- `knn` : A numeric vector giving the average nearest neighbor degree for all vertices in vids.
- `knnk` : A numeric vector, its length is the maximum (total) vertex degree in the graph. The first element is the average nearest neighbor degree of vertices with degree one, etc.

In [31]:
```
knn(G_MB, mode = "all")
knn(G_MB, mode = "in")
knn(G_MB, mode = "out")
```

$knn **XLB:** 16 **XSI:** 28 **XTL:** 16 **YBR:** 19 **YBT:** 28 **YBV:** 34 **YCR:** 19 **YDN:** 19 **YFO:** 22.6666666666667 **YGO:** 20 **YGX:** 31 **YIV:** 17.4 **YNE:** 19 **YOH:** 20 **YQD:** 22.6666666666667 **YRS:** 13.75 **YST:** 24.3333333333333 **YTH:**

7.14285714285714 **YWG:** 6.70588235294118 **YYQ:** 31 **ZAC:** 28 **ZGI:** 17.2 **ZTM:** 23.3333333333333

$knnk NaN · 29.5 · 24.3333333333333 · 21.25 · 17.4 · 22.8888888888889 · NaN · 17.9166666666667 · NaN · 17.2 · NaN · NaN · NaN · NaN · NaN · NaN · NaN · NaN · NaN · NaN · NaN · NaN · NaN · NaN · NaN · NaN · NaN · 7.14285714285714 · NaN · NaN · NaN · NaN · NaN · 6.70588235294118

$knn **XLB:** 16 **XSI:** 28 **XTL:** 16 **YBR:** 19 **YBT:** 28 **YBV:** 34 **YCR:** 19 **YDN:** 19 **YFO:** 22.6666666666667 **YGO:** 20 **YGX:** 31 **YIV:** 15 **YNE:** 19 **YOH:** 20 **YQD:** 22.6666666666667 **YRS:** 13.75 **YST:** 34 **YTH:** 7.14285714285714 **YWG:** 6.70588235294118 **YYQ:** 31 **ZAC:** 28 **ZGI:** 17.2 **ZTM:** 23.3333333333333

$knnk 30.4 · 21.25 · 20.9166666666667 · 17.9166666666667 · 17.2 · NaN · NaN · NaN · NaN · NaN · NaN · NaN · NaN · 7.14285714285714 · NaN · NaN · 6.70588235294118

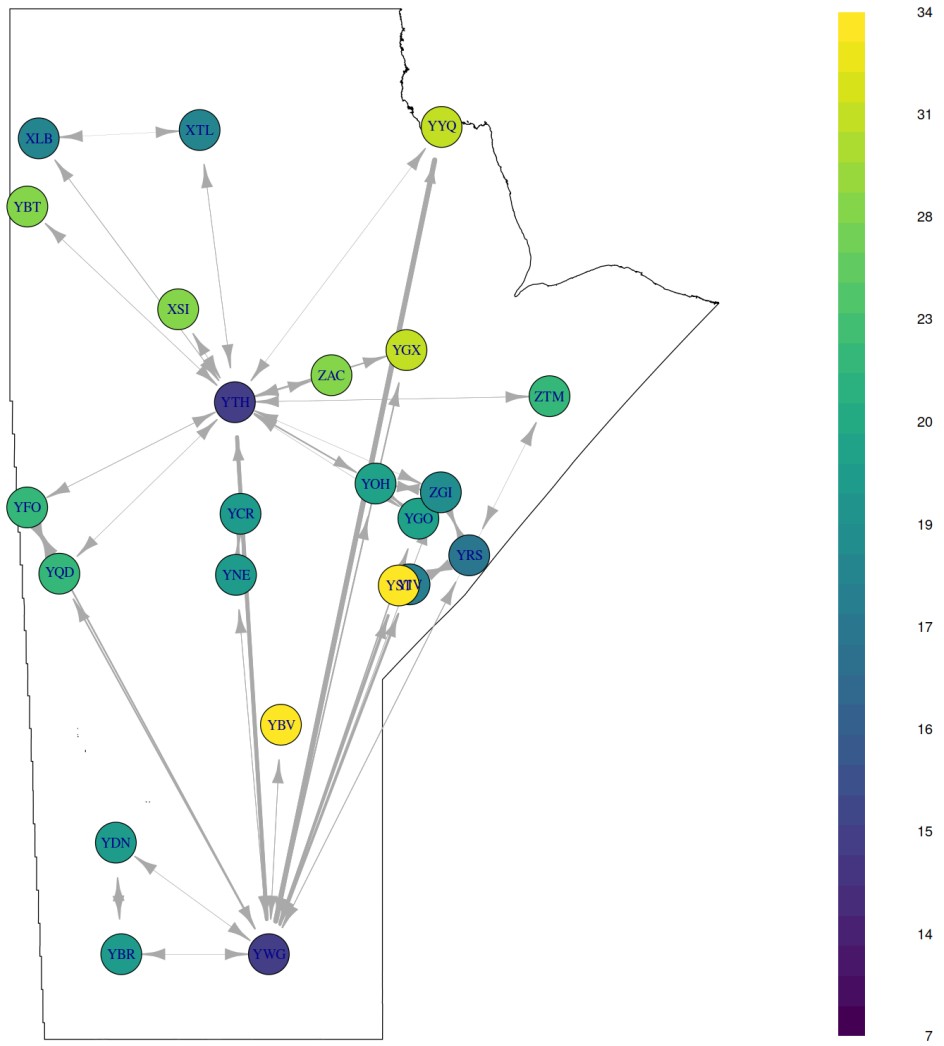
$knn **XLB:** 16 **XSI:** 28 **XTL:** 16 **YBR:** 19 **YBT:** 28 **YBV:** 34 **YCR:** 19 **YDN:** 19 **YFO:** 22.6666666666667 **YGO:** 20 **YGX:** 31 **YIV:** 21 **YNE:** 19 **YOH:** 20 **YQD:** 22.6666666666667 **YRS:** 13.75 **YST:** 19.5 **YTH:** 7.14285714285714 **YWG:** 6.70588235294118 **YYQ:** 31 **ZAC:** 28 **ZGI:** 17.2 **ZTM:** 23.3333333333333

$knnk 29.5 · 21.05 · 22.8888888888889 · 17.9166666666667 · 17.2 · NaN · NaN · NaN · NaN · NaN · NaN · NaN · NaN · 7.14285714285714 · NaN · NaN · 6.70588235294118

Let us plot the values in the graph, say, for the in-knn.

In [32]:

```
options(repr.plot.width=15, repr.plot.height=15)

knn_in = knn(G_MB, mode = "in")$knn
knn_in_bar = sort(unique(round(knn_in)))
colbar <- viridis::viridis(max(ceiling(knn_in)))
V(G_MB)$color <- colbar[round(knn_in)]

plot(MB)
V(G_MB)$size = rep(75,length(V(G_MB)))
plot(G_MB, add = TRUE, rescale = FALSE,
     edge.width = E(G_MB)$vol*0.001,
     edge.arrow.size = 0.5,
     edge.arrow.width = 0.75)
DescTools::ColorLegend(x = "right", col = colbar,
                       width = 0.5, inset = 0.1,
                       labels = sprintf("%d", knn_in_bar))
```



Coreness

In [33]:

```
coreness(G_MB, "all")
coreness(G_MB, "in")
coreness(G_MB, "out")
```

XLB: 4 **XSI:** 2 **XTL:** 4 **YBR:** 4 **YBT:** 2 **YBV:** 2 **YCR:** 4 **YDN:** 4 **YFO:** 6 **YGO:** 8 **YGX:** 4 **YIV:** 4 **YNE:** 4 **YOH:** 8 **YQD:** 6 **YRS:** 6 **YST:** 3 **YTH:** 8 **YWG:** 8 **YYQ:** 4 **ZAC:** 2 **ZGI:** 8 **ZTM:** 6

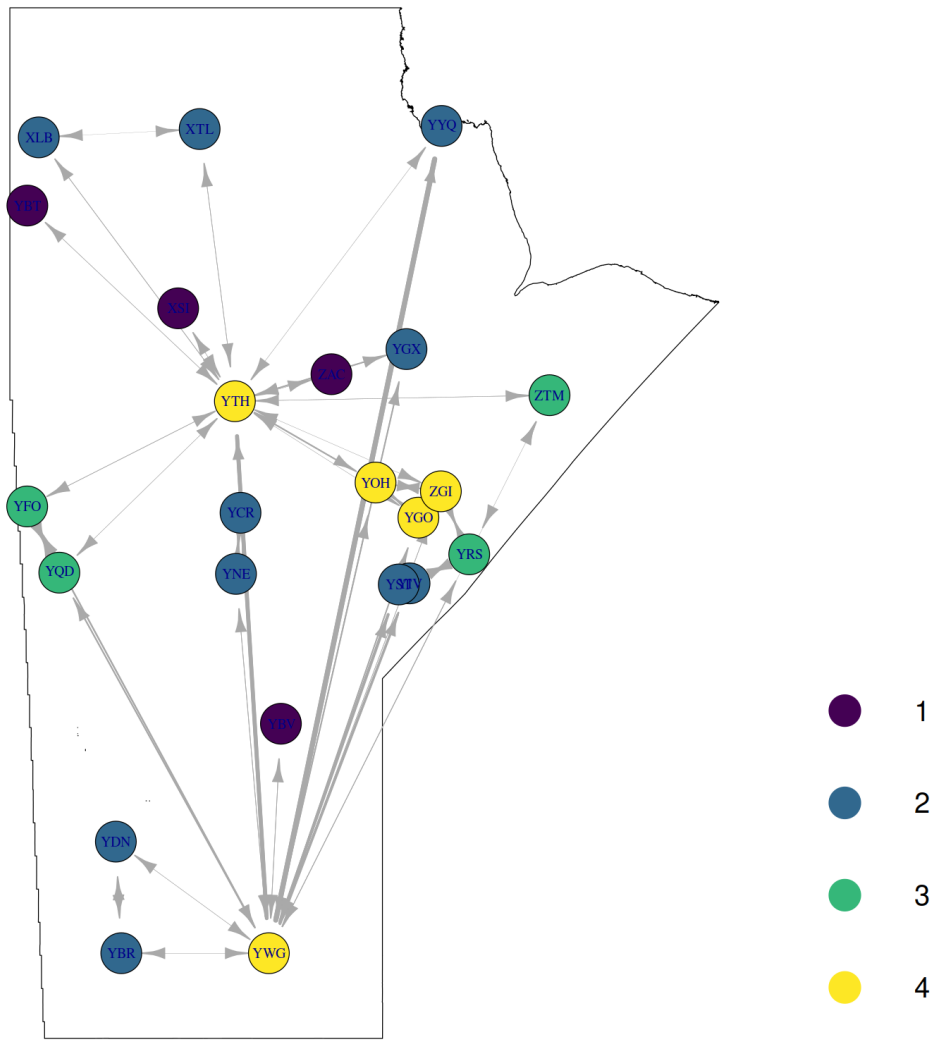
XLB: 2 **XSI:** 1 **XTL:** 2 **YBR:** 2 **YBT:** 1 **YBV:** 1 **YCR:** 2 **YDN:** 2 **YFO:** 3 **YGO:** 4 **YGX:** 2 **YIV:** 2 **YNE:** 2 **YOH:** 4 **YQD:** 3 **YRS:** 3 **YST:** 1 **YTH:** 4 **YWG:** 4 **YYQ:** 2 **ZAC:** 1 **ZGI:** 4 **ZTM:** 3

XLB: 2 **XSI:** 1 **XTL:** 2 **YBR:** 2 **YBT:** 1 **YBV:** 1 **YCR:** 2 **YDN:** 2 **YFO:** 3 **YGO:** 4 **YGX:** 2 **YIV:** 2 **YNE:** 2 **YOH:** 4 **YQD:** 3 **YRS:** 3 **YST:** 2 **YTH:** 4 **YWG:** 4 **YYQ:** 2 **ZAC:** 1 **ZGI:** 4 **ZTM:** 3

Let us look at the out-coreness, for instance, and colour the vertices as a function of that.

In [34]:

```
cores = coreness(G_MB, "out")
colbar <- viridis::viridis(max(cores))
V(G_MB)$color <- colbar[cores]

plot(MB)
V(G_MB)$size = rep(75,length(V(G_MB)))
plot(G_MB, add = TRUE, rescale = FALSE,
     edge.width = E(G_MB)$vol*0.001,
    edge.arrow.size = 0.5, edge.arrow.width = 0.75)
legend("bottomright",
       legend = sprintf("%d", sort(unique(cores))),
       pch = rep(19, length(unique(cores))),
       pt.cex = rep(5, length(unique(cores))),
       cex = 2,
       box.lty=0,
       col = colbar[sort(unique(cores))])
```



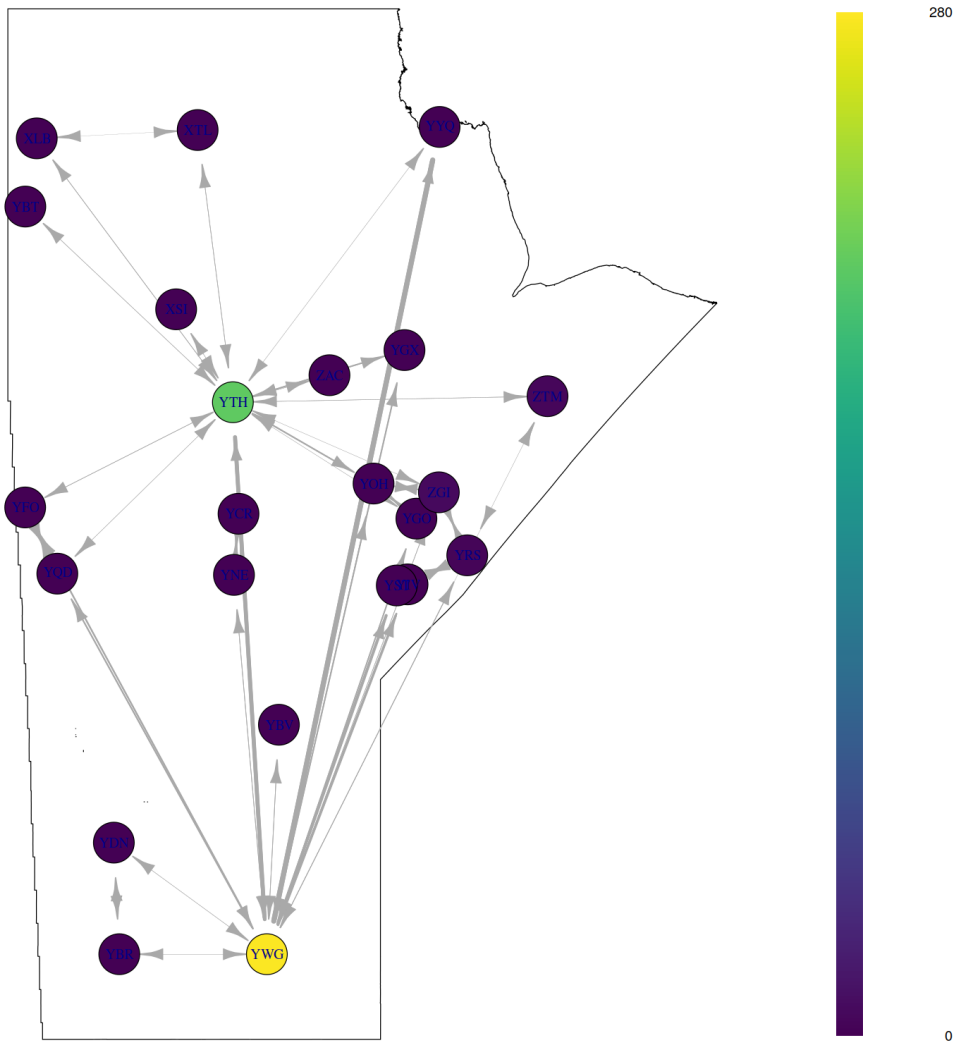
Betweenness

In [35]:
```
betweenness(G)
betweenness(G_MB)
```

RoW: 63.4166666666667 **XLB:** 0 **XSI:** 0 **XTL:** 0 **YBR:** 2.16666666666667 **YBT:** 0 **YBV:** 0 **YCR:** 0 **YDN:** 0 **YFO:** 0 **YGO:** 0 **YGX:** 0 **YIV:** 0.5 **YNE:** 0 **YOH:** 0 **YQD:** 0 **YRS:** 2.66666666666667 **YST:** 0 **YTH:** 191.916666666667 **YWG:** 250.166666666667 **YYQ:** 0 **ZAC:** 0 **ZGI:** 5.08333333333333 **ZTM:** 3.08333333333333

XLB: 0 **XSI:** 0 **XTL:** 0 **YBR:** 0 **YBT:** 0 **YBV:** 0 **YCR:** 0 **YDN:** 0 **YFO:** 0 **YGO:** 0 **YGX:** 0 **YIV:** 0.5 **YNE:** 0 **YOH:** 0 **YQD:** 0 **YRS:** 2.66666666666667 **YST:** 0 **YTH:** 211.666666666667 **YWG:** 280.166666666667 **YYQ:** 0 **ZAC:** 0 **ZGI:** 6 **ZTM:** 4

In [36]:
```
betweenness_MB = betweenness(G_MB)
# shift numbers by 1 because the second command fails when there are zeros..
colbar <- viridis::viridis(max(ceiling(betweenness_MB))+1)
V(G_MB)$color <- colbar[round(betweenness_MB)+1]
# For the label of the colour bar
colbar_str = sprintf("%d", range(unique(round(betweenness_MB))))

plot(MB)
V(G_MB)$size = rep(75,length(V(G_MB)))
plot(G_MB, add = TRUE, rescale = FALSE,
     edge.width = E(G_MB)$vol*0.001,
     edge.arrow.size = 0.5,
     edge.arrow.width = 0.75)
DescTools::ColorLegend(x = "right", col = colbar,
                       width = 0.5, inset = 0.1,
                       labels = colbar_str)
```



Closeness

First, the unweighted case, i.e., using the geodesic distance.

In [37]:
```
closeness(G_MB, mode = "all")
closeness(G_MB, mode = "in")
closeness(G_MB, mode = "out")
```

XLB: 0.02 **XSI:** 0.0196078431372549 **XTL:** 0.02 **YBR:** 0.0212765957446809 **YBT:** 0.0196078431372549 **YBV:** 0.0208333333333333 **YCR:** 0.0212765957446809 **YDN:** 0.0212765957446809 **YFO:** 0.024390243902439 **YGO:** 0.025 **YGX:** 0.0238095238095238 **YIV:** 0.0217391304347826 **YNE:** 0.0212765957446809 **YOH:** 0.025 **YQD:** 0.024390243902439 **YRS:** 0.0222222222222222 **YST:** 0.0212765957446809 **YTH:** 0.0333333333333333 **YWG:**

0.037037037037037 **YYQ:** 0.0238095238095238 **ZAC:** 0.0196078431372549 **ZGI:** 0.0256410256410256 **ZTM:** 0.024390243902439

XLB: 0.02 **XSI:** 0.0196078431372549 **XTL:** 0.02 **YBR:** 0.0212765957446809 **YBT:** 0.0196078431372549 **YBV:** 0.0208333333333333 **YCR:** 0.0212765957446809 **YDN:** 0.0212765957446809 **YFO:** 0.024390243902439 **YGO:** 0.025 **YGX:** 0.0238095238095238 **YIV:** 0.0217391304347826 **YNE:** 0.0212765957446809 **YOH:** 0.025 **YQD:** 0.024390243902439 **YRS:** 0.0222222222222222 **YST:** 0.0208333333333333 **YTH:** 0.0333333333333333 **YWG:** 0.037037037037037 **YYQ:** 0.0238095238095238 **ZAC:** 0.0196078431372549 **ZGI:** 0.0256410256410256 **ZTM:** 0.024390243902439

XLB: 0.02 **XSI:** 0.0196078431372549 **XTL:** 0.02 **YBR:** 0.0212765957446809 **YBT:** 0.0196078431372549 **YBV:** 0.0208333333333333 **YCR:** 0.0212765957446809 **YDN:** 0.0212765957446809 **YFO:** 0.024390243902439 **YGO:** 0.025 **YGX:** 0.0238095238095238 **YIV:** 0.0212765957446809 **YNE:** 0.0212765957446809 **YOH:** 0.025 **YQD:** 0.024390243902439 **YRS:** 0.0222222222222222 **YST:** 0.0212765957446809 **YTH:** 0.0333333333333333 **YWG:** 0.037037037037037 **YYQ:** 0.0238095238095238 **ZAC:** 0.0196078431372549 **ZGI:** 0.0256410256410256 **ZTM:** 0.024390243902439

We can also compute closeness using weighted arcs, i.e., using the travel volumes. Geodesic distance shown for comparison. We save the results for future manipulation.

In [38]:
```
# Unweighted case
cl_MB_U = closeness(G_MB, mode = "out")
cl_MB_U
# Weighted case
cl_MB_W = closeness(G_MB, mode = "out", weights = E(G_MB)$vol)
cl_MB_W
```

XLB: 0.02 **XSI:** 0.0196078431372549 **XTL:** 0.02 **YBR:** 0.0212765957446809 **YBT:** 0.0196078431372549 **YBV:** 0.0208333333333333 **YCR:** 0.0212765957446809 **YDN:** 0.0212765957446809 **YFO:** 0.024390243902439 **YGO:** 0.025 **YGX:** 0.0238095238095238 **YIV:** 0.0212765957446809 **YNE:** 0.0212765957446809 **YOH:** 0.025 **YQD:** 0.024390243902439 **YRS:** 0.0222222222222222 **YST:** 0.0212765957446809 **YTH:** 0.0333333333333333 **YWG:** 0.037037037037037 **YYQ:** 0.0238095238095238 **ZAC:** 0.0196078431372549 **ZGI:** 0.0256410256410256 **ZTM:** 0.024390243902439

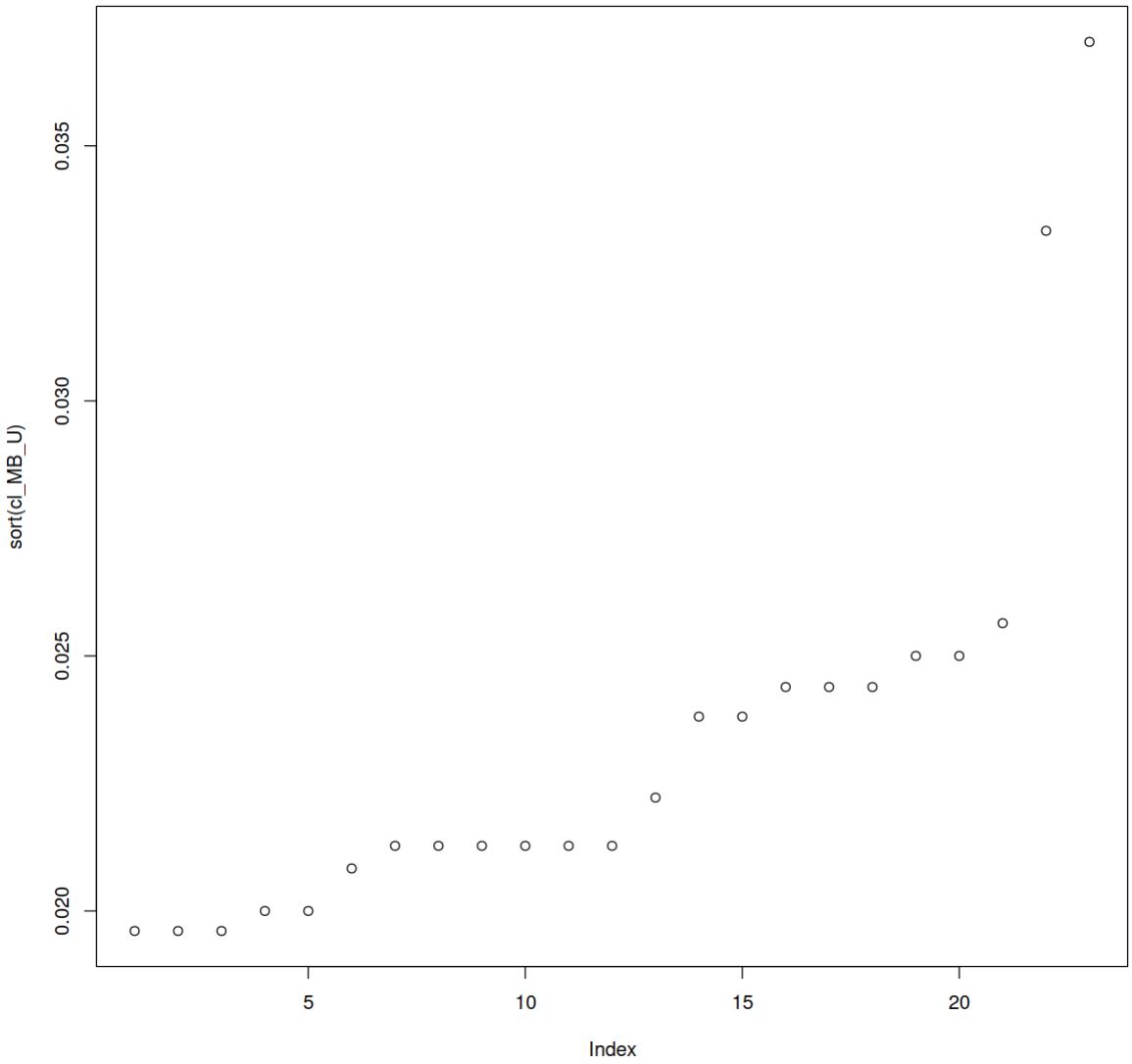
XLB: 3.55606130649692e-05 **XSI:** 4.17188151856487e-05 **XTL:** 3.78845279587816e-05 **YBR:** 4.0256028340244e-05 **YBT:** 3.98914951332376e-05 **YBV:** 2.92817194225645e-05 **YCR:** 3.76293508936971e-05 **YDN:** 3.72689326177698e-05 **YFO:** 3.60516259283294e-05 **YGO:** 5.86063412061185e-05 **YGX:** 2.42072137496974e-05 **YIV:** 3.33555703802535e-05 **YNE:** 2.98632264229827e-05 **YOH:** 6.46663217796172e-05 **YQD:** 4.28320555103439e-05 **YRS:** 5.35188654000535e-05 **YST:** 2.24749404414078e-05 **YTH:** 6.413134098634e-05 **YWG:** 5.21893429361724e-05 **YYQ:** 4.67573759760602e-05 **ZAC:** 4.3813529617946e-05 **ZGI:** 6.56857593273778e-05 **ZTM:** 5.54938956714761e-05

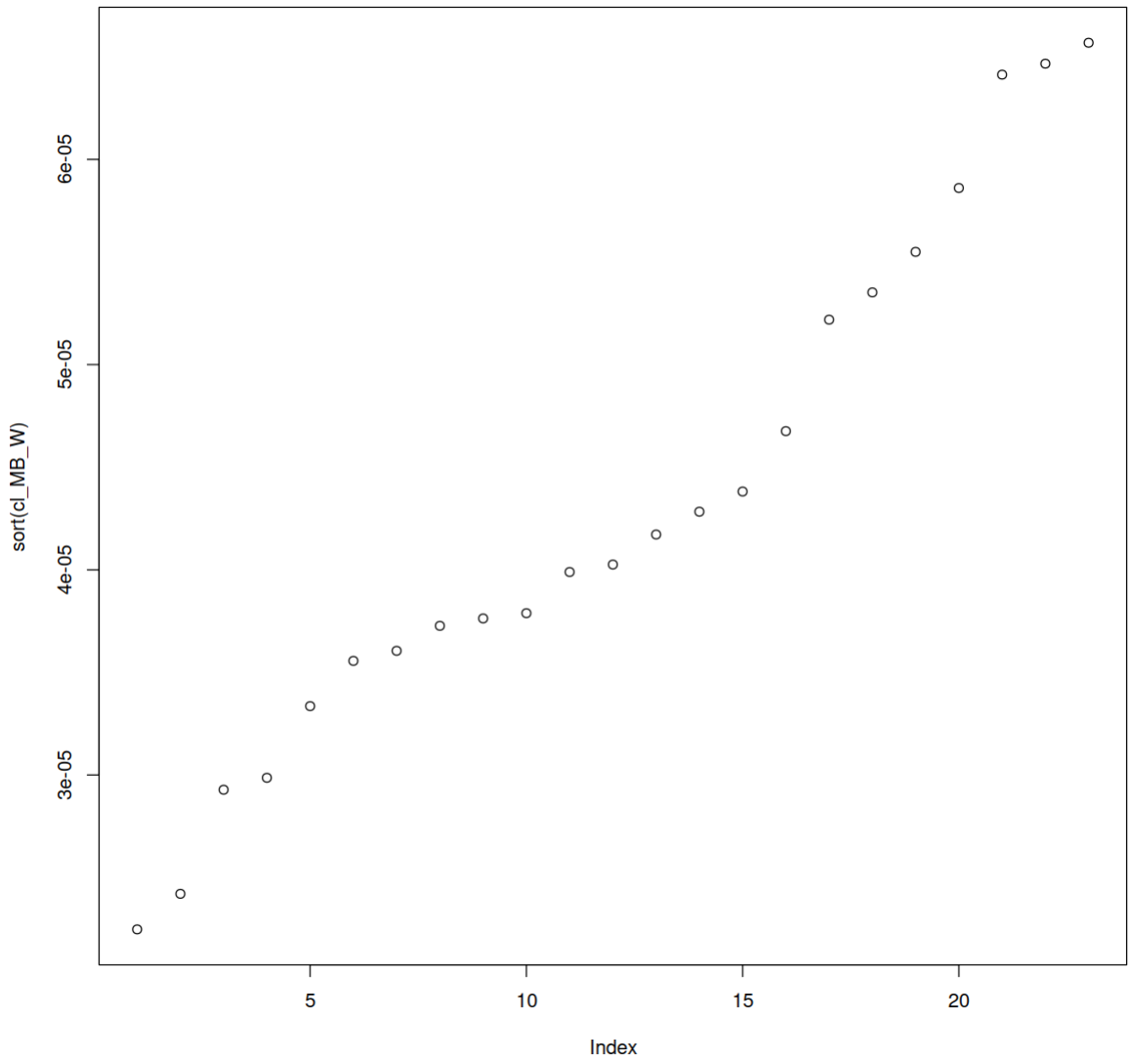
To prepare for plots, let us look at the ranges of values.

In [39]:
```
options(repr.plot.width=10, repr.plot.height=10)

range(cl_MB_U)
plot(sort(cl_MB_U))
range(cl_MB_W)
plot(sort(cl_MB_W))
```

0.0196078431372549 · 0.037037037037037

2.24749404414078e-05 · 6.56857593273778e-05





So if we multiply the first values by 10,000 and the second ones by 1e7, we should get good ranges of variation.

In [40]:
```
range(cl_MB_U * 1e4)
range(cl_MB_W * 1e7)
```

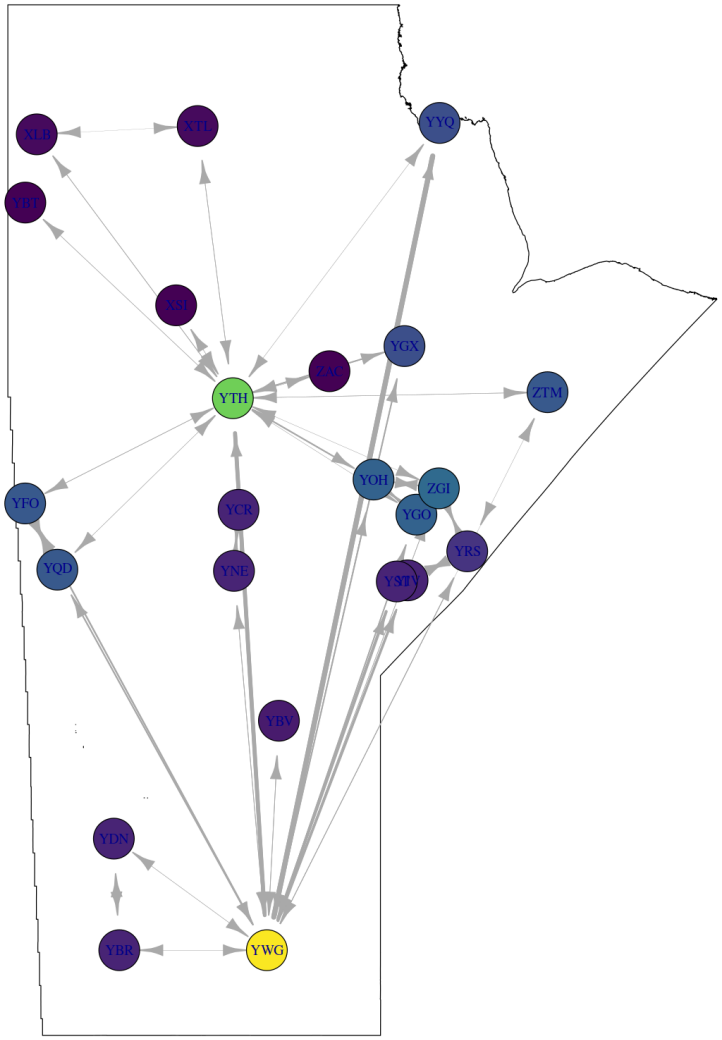
196.078431372549 · 370.37037037037

224.749404414078 · 656.857593273778

Indeed, so that's what we do.

In [41]:
```
options(repr.plot.width=15, repr.plot.height=15)

cl_MB_U_tmp = cl_MB_U * 1e4
Delta_cl = ceiling(max(cl_MB_U_tmp)-min(cl_MB_U_tmp))
colbar <- viridis::viridis(Delta_cl+1)
V(G_MB)$color <- colbar[round(cl_MB_U_tmp-min(cl_MB_U_tmp))+1]
# For the label of the colour bar
colbar_str = sprintf("%f", range(cl_MB_U_tmp))
```

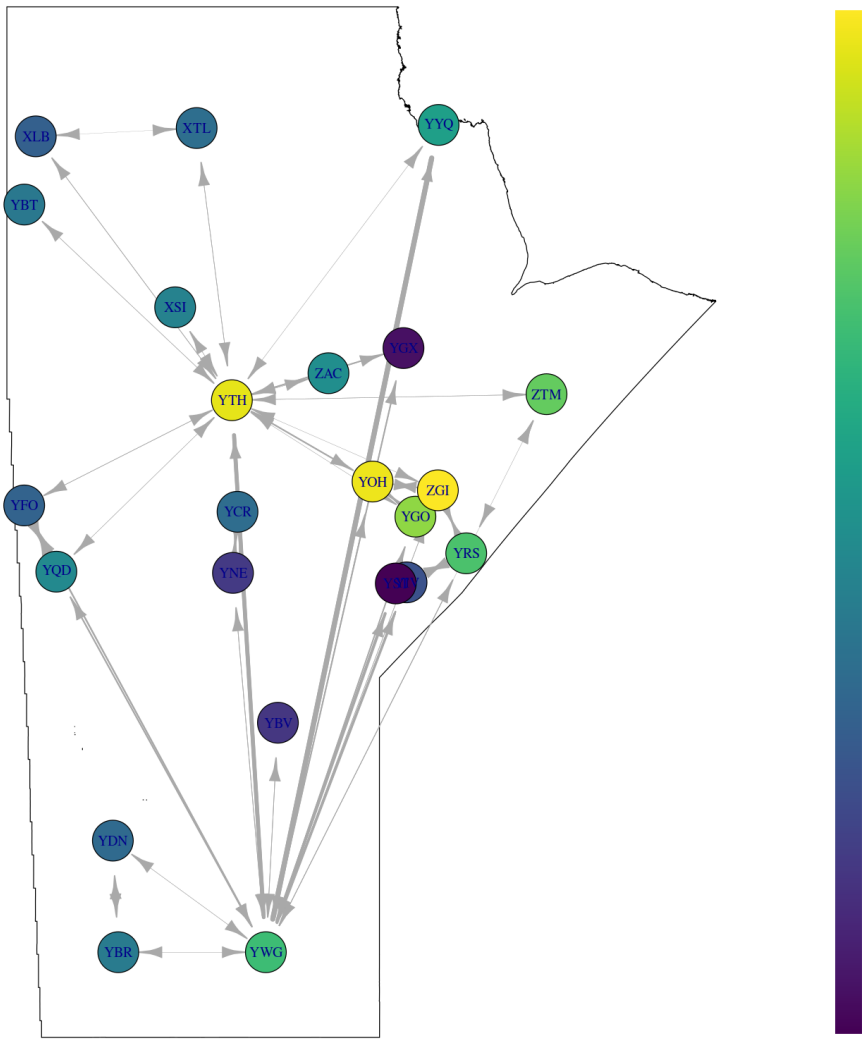
```
plot(MB)
V(G_MB)$size = rep(75,length(V(G_MB)))
plot(G_MB, add = TRUE, rescale = FALSE,
     edge.width = E(G_MB)$vol*0.001,
     edge.arrow.size = 0.5,
     edge.arrow.width = 0.75)
DescTools::ColorLegend(x = "right", col = colbar,
                       width = 0.5, inset = 0.1,
                       labels = colbar_str)
```



In [42]:

```
cl_MB_W_tmp = cl_MB_W * 1e7
Delta_cl = ceiling(max(cl_MB_W_tmp)-min(cl_MB_W_tmp))
colbar <- viridis::viridis(Delta_cl+1)
V(G_MB)$color <- colbar[round(cl_MB_W_tmp-min(cl_MB_W_tmp))+1]
# For the label of the colour bar
colbar_str = sprintf("%f", range(cl_MB_W_tmp))

plot(MB)
```

```
V(G_MB)$size = rep(75,length(V(G_MB)))
plot(G_MB, add = TRUE, rescale = FALSE,
     edge.width = E(G_MB)$vol*0.001,
     edge.arrow.size = 0.5,
     edge.arrow.width = 0.75)
DescTools::ColorLegend(x = "right", col = colbar,
                       width = 0.5, inset = 0.1,
                       labels = colbar_str)
```

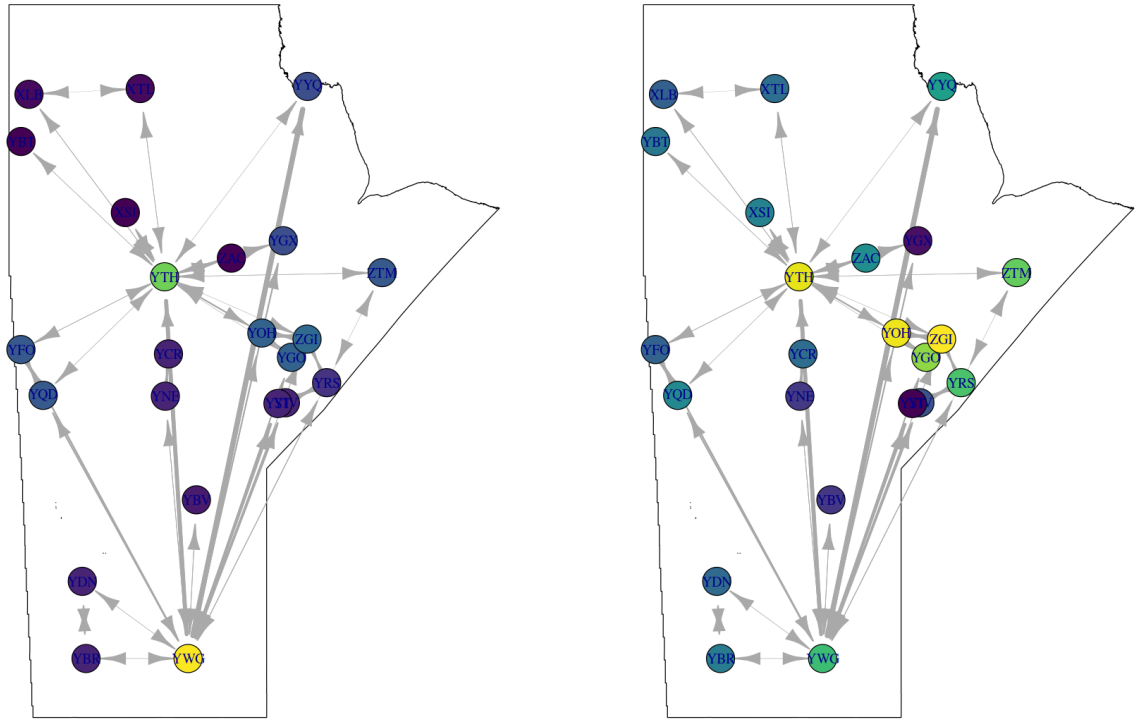


One interesting remark here: closeness radically changes when considering the unweighted and weighted version. Just to confirm this, let us plot the two graphs side by side.

In [43]:

```
par(mfrow=c(1,2))

# Unweighted MB only
Delta_cl = ceiling(max(cl_MB_U_tmp)-min(cl_MB_U_tmp))
colbar <- viridis::viridis(Delta_cl+1)
V(G_MB)$color <- colbar[round(cl_MB_U_tmp-min(cl_MB_U_tmp))+1]
# For the label of the colour bar
```

```
colbar_str = sprintf("%f", range(cl_MB_U_tmp))
plot(MB)
V(G_MB)$size = rep(75,length(V(G_MB)))
plot(G_MB, add = TRUE, rescale = FALSE,
     edge.width = E(G_MB)$vol*0.001,
     edge.arrow.size = 0.5,
     edge.arrow.width = 0.75)

# Weighted MB only
Delta_cl = ceiling(max(cl_MB_W_tmp)-min(cl_MB_W_tmp))
colbar <- viridis::viridis(Delta_cl+1)
V(G_MB)$color <- colbar[round(cl_MB_W_tmp-min(cl_MB_W_tmp))+1]
# For the label of the colour bar
colbar_str = sprintf("%f", range(cl_MB_W_tmp))
plot(MB)
V(G_MB)$size = rep(75,length(V(G_MB)))
plot(G_MB, add = TRUE, rescale = FALSE,
     edge.width = E(G_MB)$vol*0.001,
     edge.arrow.size = 0.5,
     edge.arrow.width = 0.75)
```



So Thompson (YTH) is has higher closeness centrality than Winnipeg (YWG) when passenger volumes are considered. Does the situation change if we recall that Winnipeg is quite connected to the rest of the world? We prepare the computations and the plots as we had done earlier.

In [44]:

```
# Unweighted case
cl_MB_RoW_U = closeness(G, mode = "out")
range(cl_MB_RoW_U)
# Weighted case
cl_MB_RoW_W = closeness(G, mode = "out", weights = E(G)$vol)
range(cl_MB_RoW_W)
```

0.0196078431372549 · 0.0357142857142857

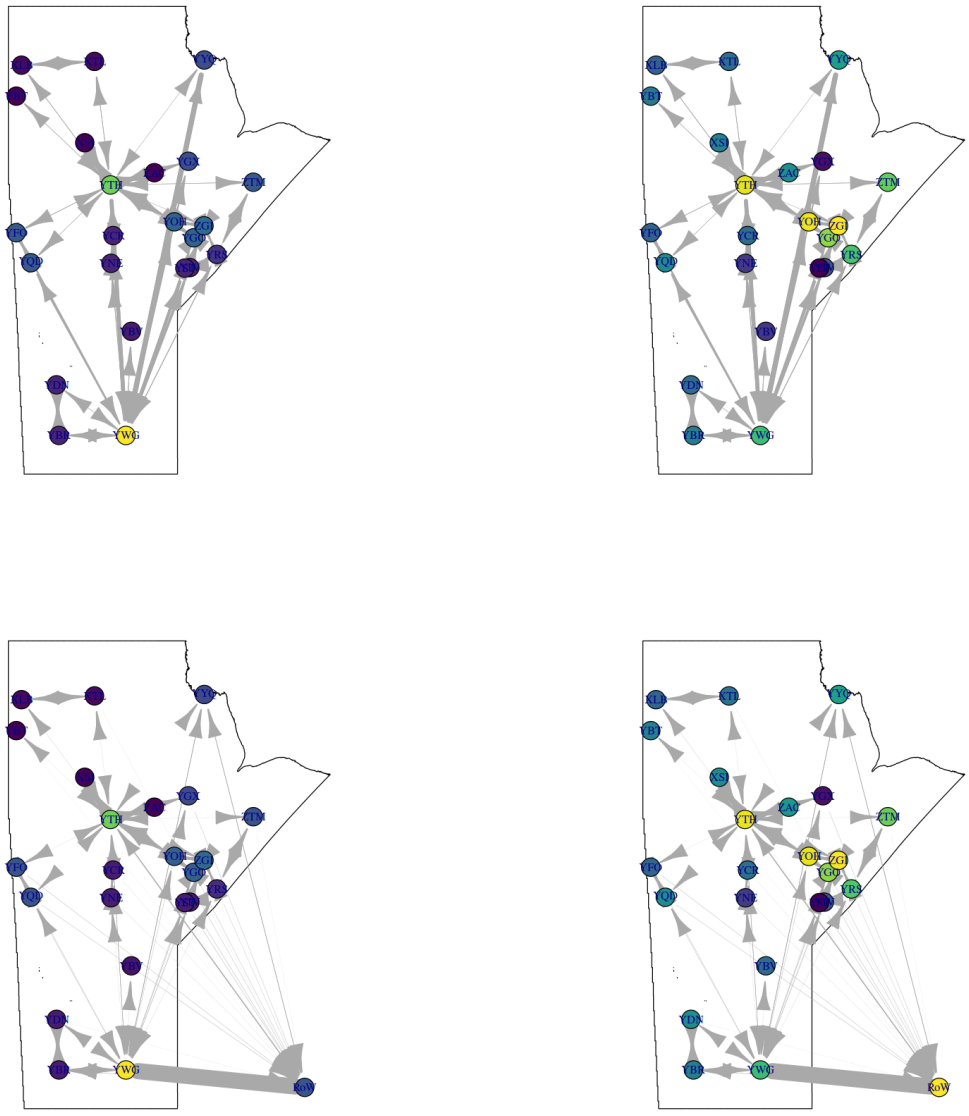
2.14486412285782e-05 · 6.24726682076591e-05

Ranges are quite similar to those earlier, so we can probably just do the same as before. We will plot the 4 figures this time. First row is as before, second row is with rest of the world information

thrown in.

In [45]:

```
par(mfrow=c(2,2))

cl_MB_RoW_U_tmp = cl_MB_RoW_U * 1e4
cl_MB_RoW_W_tmp = cl_MB_RoW_W * 1e7

# Unweighted MB only
Delta_cl = ceiling(max(cl_MB_U_tmp)-min(cl_MB_U_tmp))
colbar <- viridis::viridis(Delta_cl+1)
V(G_MB)$color <- colbar[round(cl_MB_U_tmp-min(cl_MB_U_tmp))+1]
# For the label of the colour bar
colbar_str = sprintf("%f", range(cl_MB_U_tmp))
plot(MB)
V(G_MB)$size = rep(75,length(V(G_MB)))
plot(G_MB, add = TRUE, rescale = FALSE,
     edge.width = E(G_MB)$vol*0.001,
     edge.arrow.size = 0.5,
     edge.arrow.width = 0.75)

# Weighted MB only
Delta_cl = ceiling(max(cl_MB_W_tmp)-min(cl_MB_W_tmp))
colbar <- viridis::viridis(Delta_cl+1)
V(G_MB)$color <- colbar[round(cl_MB_W_tmp-min(cl_MB_W_tmp))+1]
# For the label of the colour bar
colbar_str = sprintf("%f", range(cl_MB_W_tmp))
plot(MB)
V(G_MB)$size = rep(75,length(V(G_MB)))
plot(G_MB, add = TRUE, rescale = FALSE,
     edge.width = E(G_MB)$vol*0.001,
     edge.arrow.size = 0.5,
     edge.arrow.width = 0.75)

# Unweighted MB with RoW
Delta_cl = ceiling(max(cl_MB_RoW_U_tmp)-min(cl_MB_RoW_U_tmp))
colbar <- viridis::viridis(Delta_cl+1)
V(G)$color <- colbar[round(cl_MB_RoW_U_tmp-min(cl_MB_RoW_U_tmp))+1]
# For the label of the colour bar
colbar_str = sprintf("%f", range(cl_MB_RoW_U_tmp))
plot(MB)
V(G)$size = rep(75,length(V(G)))
plot(G, add = TRUE, rescale = FALSE,
     edge.width = E(G)$vol*0.0001,
     edge.arrow.size = 0.5,
     edge.arrow.width = 0.75)

# Weighted MB with RoW
Delta_cl = ceiling(max(cl_MB_RoW_W_tmp)-min(cl_MB_RoW_W_tmp))
colbar <- viridis::viridis(Delta_cl+1)
V(G)$color <- colbar[round(cl_MB_RoW_W_tmp-min(cl_MB_RoW_W_tmp))+1]
# For the label of the colour bar
colbar_str = sprintf("%f", range(cl_MB_RoW_W_tmp))
plot(MB)
V(G)$size = rep(75,length(V(G)))
plot(G, add = TRUE, rescale = FALSE,
     edge.width = E(G)$vol*0.0001,
     edge.arrow.size = 0.5,
     edge.arrow.width = 0.75)
```



Actually, things do not change, except that RoW becomes important as well.