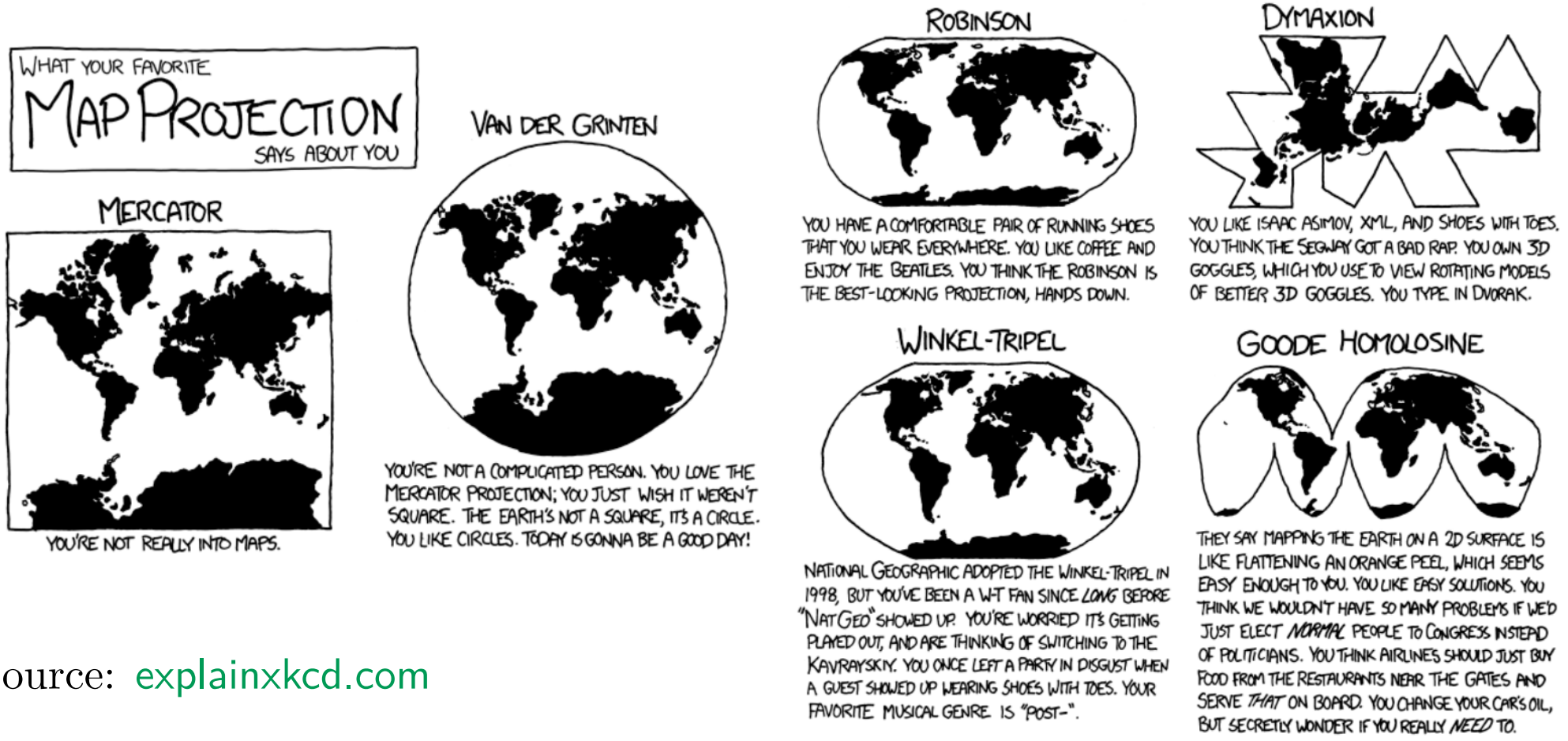
# Maps and Projections

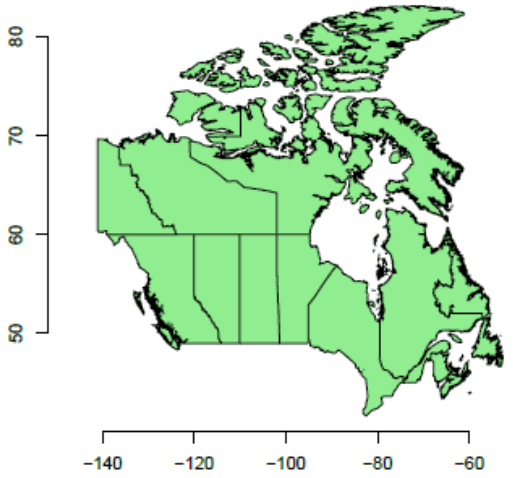


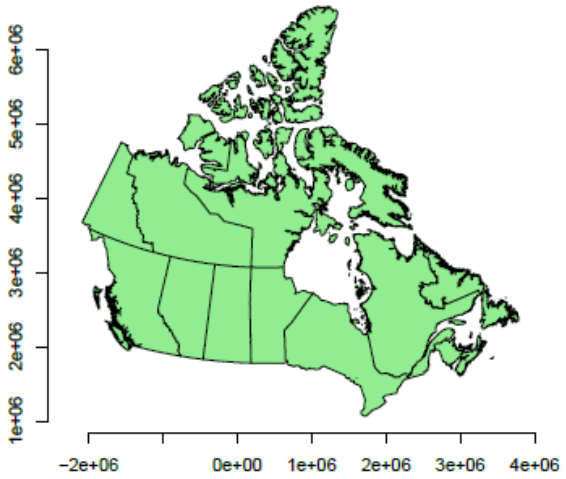

Source: explainxkcd.com

## Maps and Projections

```
1 library(rgdal)
2 library(maptools)
3 Proj <- CRS("+proj=longlat +datum=WGS84")
4 CAN_shp <- readShapePoly("CAN_adm1.shp",
      verbose=TRUE, proj4string=Proj)
5 plot(CAN_shp)

6 new_CAN_shp <- spTransform(CAN_shp, CRS("+
      init=epsg:26978"))
7 plot(new_CAN_shp)
```



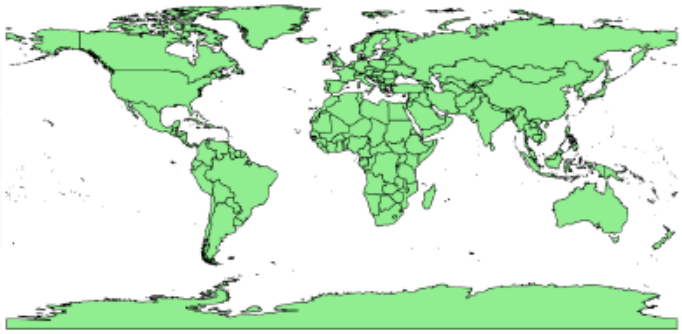



# Maps and Projections

```
library(sp)
library(rworldmap)
data(countriesLow)

crs.WGS84 <- CRS("+proj=longlat +datum=WGS84
    ")
countries.WGS84 <- spTransform(countriesLow,
     crs.WGS84)
plot(countries.WGS84,col="light green")

crs.laea <- CRS("+proj=laea +lat_0=52 +lon_
    0=10 +x_0=4321000 +y_0=3210000 +ellps=
    GRS80 +units=m +no_defs")
countries.laea <- spTransform(countriesLow,
    crs.laea)
plot(countries.laea,col="light green")
```



## Maps and Points

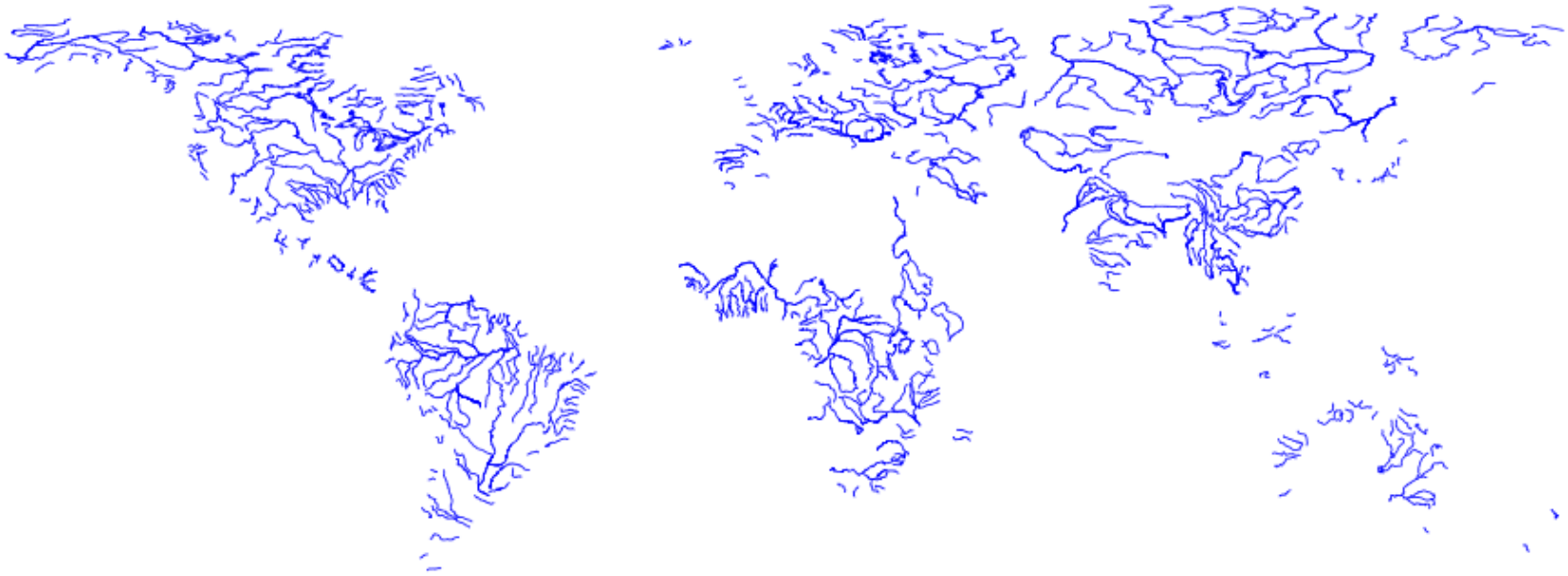
```
library(maps)
data("world.cities")
world.cities2 = world.cities[world.cities$pop>100000,]
plot(world.cities2$lon,world.cities2$lat,pch=19,cex=.7,axes=FALSE,
    xlab="",ylab="")
```

## Maps and Lines

```
loc="https://raw.githubusercontent.com/jjrom/hydre/master/mapserver/
    geodata/rivers/GRDC_687_rivers"
ext=c(".dbf",".prj",".sbn",".sbx",".shp",".shp.xml",".shx")
dir.create('data/rivieres')
for(i in 1:length(ext)){
 loc1=paste(loc,ext[i],"?raw=true",sep="")
 loc2=paste("data/rivieres/GRDC_687_rivers",ext[i],sep="")
 getdata(loc1,loc2)
}
```

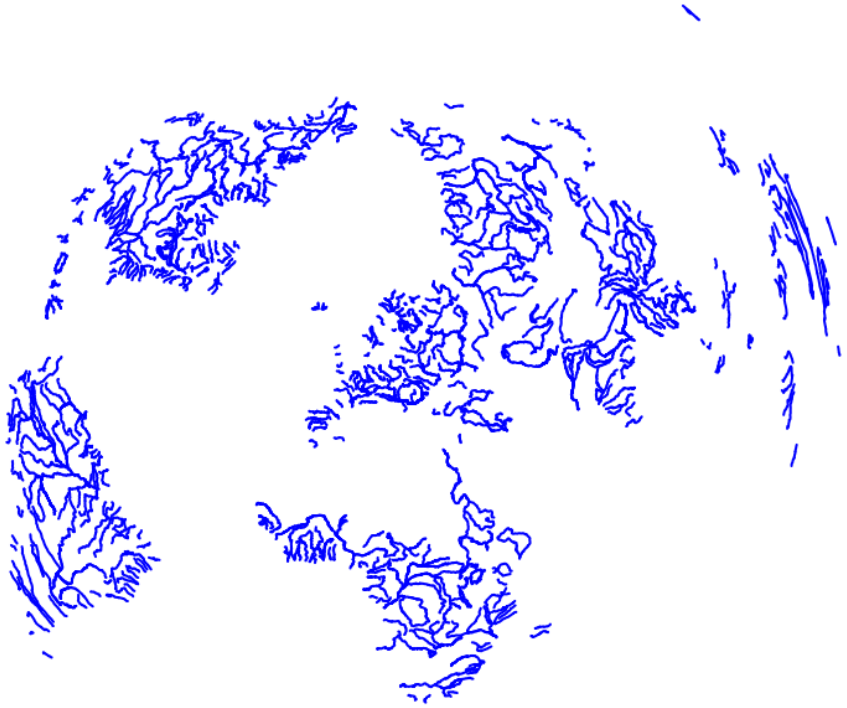
## Maps and Lines

```
library("sp")
library("rgdal")
shape <- readShapeLines("GRDC_687_rivers.shp")
plot(shape,col="blue")
```



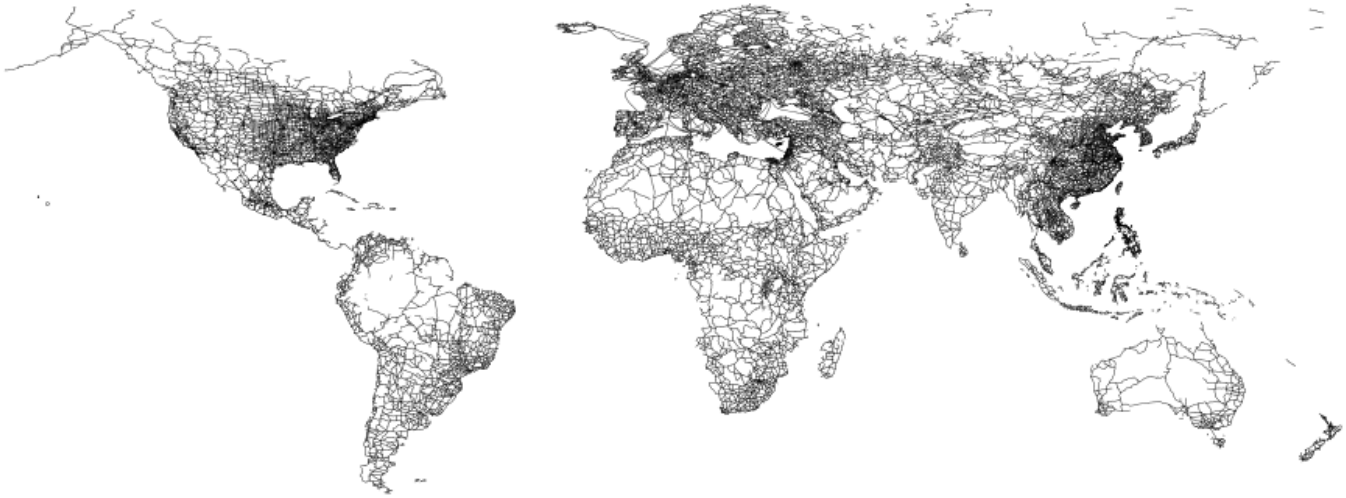
## Maps and Lines

```
crs.laea <- CRS("+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_
    0=3210000 +ellps=GRS80 +units=m +no_defs")
shape.nouveau = spTransform(shape,crs.laea)
plot(shape.nouveau,col="blue")
```



## Maps and Lines

```
getdatazip("http://www.naturalearthdata.com/http//www.
    naturalearthdata.com/download/10m/cultural/ne_10m_roads.zip", "
    data/routes.zip", 'data/routes/')
shape <- readShapeLines("data/routes/ne_10m_roads.shp")
plot(shape,pch=19,lwd=.7)
```



## Maps and Polygons

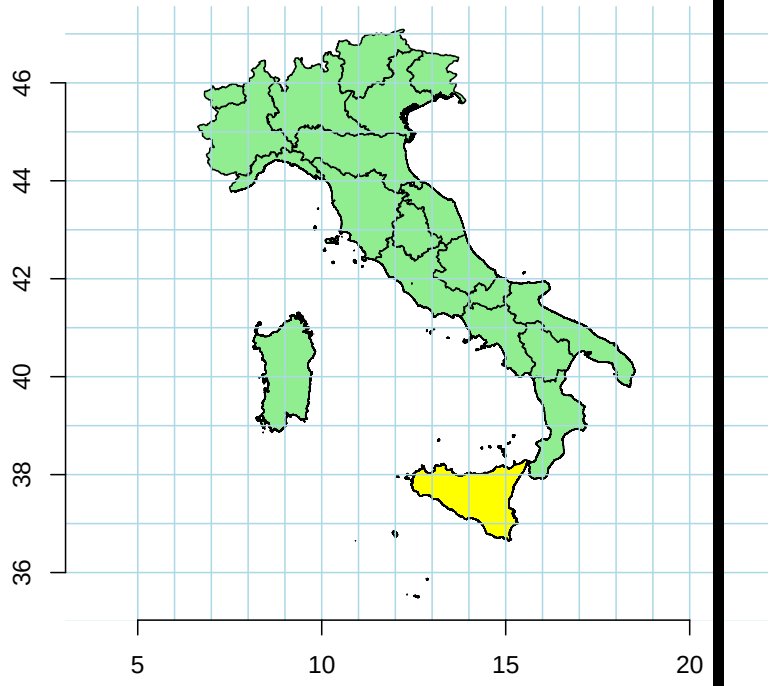
```
download.file("http://freakonometrics.free.
    fr/Italy.zip",destfile="Italy.zip")
library(rgdal)
ita1<-readOGR(dsn="/Italy",layer="ITA_adm1",
    verbose=FALSE)
plot(ita1,col="light green")
```

## Maps and Polygons

```
> ita1@data[,"NAME_1"][1:6]
[1]   Abruzzo   Apulia   Basilicata   Calabria
[5] Campania Emilia-Romagna
pos<-which(ita1@data[,"NAME_1"]=="Sicily")
Poly_Sicily<-ita1[pos,]
plot(ita1,col="light green")
plot(Poly_Sicily,col="yellow",add=TRUE)
```

## Maps and Polygons

```
plot(ita1,col="light green")
plot(Poly_Sicily,col="yellow",add=TRUE)
abline(v=5:20,col="light blue")
abline(h=35:50,col="light blue")
axis(1)
axis(2)
```



## Maps and Polygons

```
pos<-which(ita1@data[,"NAME_1"] %in% c("
    Abruzzo","Apulia","Basilicata","Calabria
    ","Campania","Lazio","Molise","Sardegna"
    ,"Sicily"))
ita_south <- ita1[pos,]
ita_north <- ita1[-pos,]
plot(ita1)
plot(ita_south,col="red",add=TRUE)
plot(ita_north,col="blue",add=TRUE)
```

## Maps and Polygons

Here we have only used colors, but it is also possible to merge the polygons together,

```
library(rgeos)
ita_s<-gUnionCascaded(ita_south)
ita_n<-gUnionCascaded(ita_north)
plot(ita1)
plot(ita_s,col="red",add=TRUE)
plot(ita_n,col="blue",add=TRUE)
```

## Maps and Polygons

On polygons, it is also possible to visualize centroids,

```
plot(ita1,col="light green")
plot(Poly_Sicily,col="yellow",add=TRUE)
> gCentroid(Poly_Sicily,byid=TRUE)
SpatialPoints:
          x        y
14 14.14668 37.58842
Coordinate Reference System (CRS) arguments:
+proj=longlat +ellps=WGS84
+datum=WGS84 +no_defs +towgs84=0,0,0
points(gCentroid(Poly_Sicily,byid=TRUE),pch
    =19,col="red")
```

## Maps and Polygons or

```
G <- as.data.frame(gCentroid(ita1,byid=TRUE)
    )
plot(ita1,col="light green")
text(G$x,G$y,1:20)
```

## Maps and Polygons

Consider two trajectories, caracterized by a series of knots (from the centroïds list)

```
c1 <-G[c(17,20,6,16,8,12,2),]
c2 <- G[c(13,11,1,5,3,4),]
```

We can convert those segments into SpatialLines

```
L1<-SpatialLines(list(Lines(list(Line(c1)),"Line1")))
L2<-SpatialLines(list(Lines(list(Line(c2)),"Line2")))
```

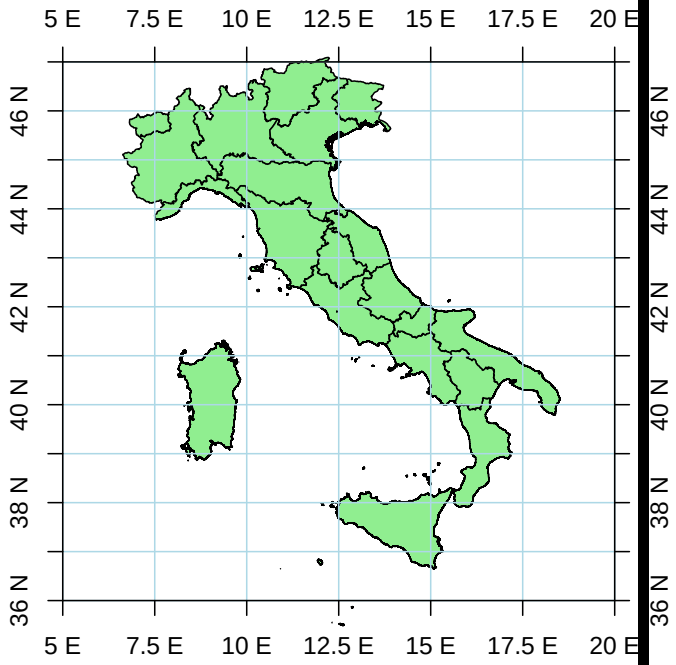
To make sure that we can add those lines on the map, use

```
L1@proj4string<-ita1@proj4string
L2@proj4string<-ita1@proj4string
cross_road <-gIntersection(L1,L2)
> cross_road@coords
SpatialPoints:
          x        y
1 11.06947 44.03287
1 14.20034 41.74879
Coordinate Reference System (CRS) arguments:
+proj=longlat +ellps=WGS84
+datum=WGS84 +no_defs +towgs84=0,0,0
plot(ita1,col="light green")
plot(L1,col="red",lwd=2,add=TRUE)
plot(L2,col="blue",lwd=2,add=TRUE)
plot(cross_road,pch=19,cex=1.5,add=TRUE)
```

## Maps and Polygons

To add elements on maps, consider, e.g.

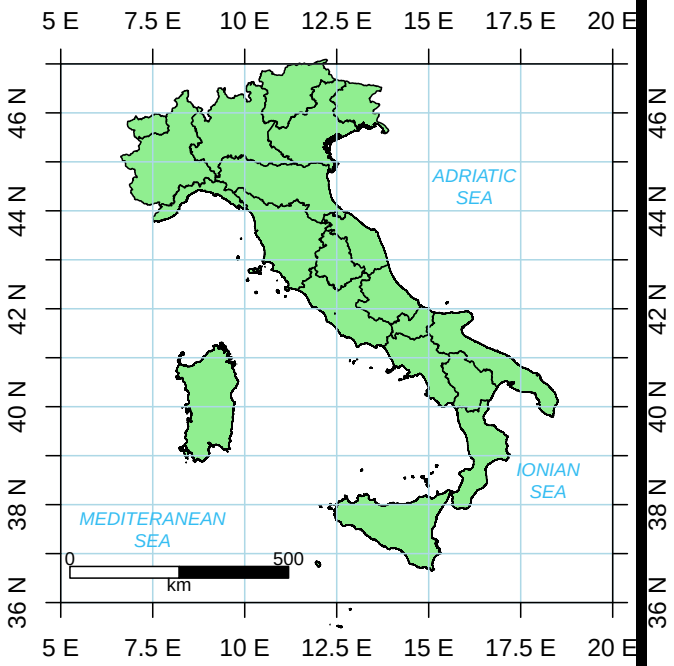
```
plot(ita1,col="light green")
grat <- border_plot(sp=ita1,WE=seq(5,20,by
    =2.5),NS=seq(36,47))
plot(grat,col="light blue",add=TRUE)
labs<-paste(seq(5,20,by=2.5)," E",sep="")
axis(side=1,pos=36,at=seq(5,20,by=2.5),
labels=labs)
axis(side=3,pos=47,at=seq(5,20,by=2.5),
labels=labs)
labs<-paste(seq(36,47)," N",sep="")
axis(side=2,pos=5,at=seq(36,47),
labels=labs)
axis(side=4,pos=20,at=seq(36,47),
labels=labs)
```



## Maps and Polygons

To add elements on maps, consider, e.g.

```
sea<-data.frame(Name=c("MEDITERANEAN\nSEA","
    ADRIATIC\nSEA","IONIAN\nSEA"),
Longitude=c(7.5,16.25,18.25),Latitude=c
    (37.5,44.5,38.5))
text(sea$Longitude,sea$Latitude,sea$Name,
cex=0.75,col="#34bdf2",font=3)
library(raster)
scalebar(d=500,xy=c(5.25,36.5),,type="bar",
    below="km",
lwd=4,divs=2,col="black",cex=0.75,lonlat=
    TRUE)
```



## Maps and Polygons

```
library(sp)
library(raster)
download.file("http://biogeo.ucdavis.edu/
    data/gadm2.8/rds/GBR_adm2.rds","GBR_adm2
    .rds")
UK=readRDS("GBR_adm2.rds")
UK@data[159,"HASC_2"]="GB.NR"
plot(UK, xlim = c(-4,-2), ylim = c(50, 59),
    main="UK areas")
```

## Maps and Polygons

One can add Ireland on the left part

```
download.file("http://biogeo.ucdavis.edu/
    data/gadm2.8/rds/IRL_adm0.rds","IRL_adm0
    .rds")
IRL=readRDS("IRL_adm0.rds")
plot(IRL,add=TRUE)
```

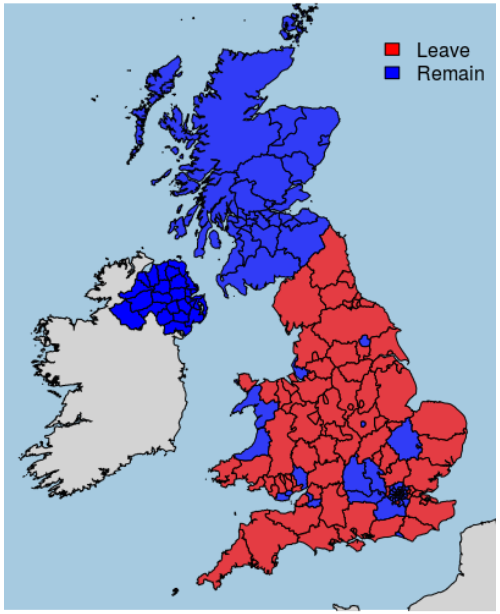
## Maps and Polygons

and France, in the lower right corner

```
download.file("http://biogeo.ucdavis.edu/data/gadm2.8/rds/FRA_adm0.rds","FRA_adm0.rds")
FR=readRDS("FRA_adm0.rds")
plot(FR,add=TRUE)
loc="https://f.hypotheses.org/wp-content/blogs.dir/253/files/2016/12/EU-referendum-result-data.csv"
referendum=read.csv(loc,header=TRUE,dec=".",sep=",",stringsAsFactors = FALSE)
referendum=referendum[c(3,6,13,14)]
library(plyr)
referendum=ddply(referendum,.(Region,HASC_code),summarise,Remain=sum(Remain),Leave=sum(Leave))
```

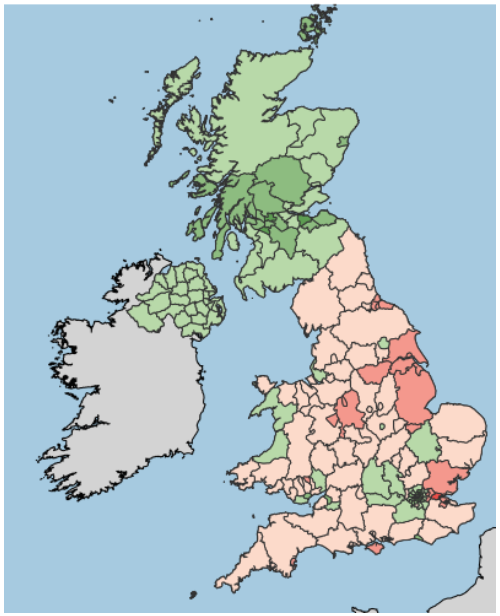
## Maps and Polygons

```
row.names(referendum)=seq(1,nrow(referendum),1)
leave_or_remain=cbind(referendum,"Brexit?"=0)
leave_or_remain[,"Brexit?"]=ifelse(leave_or_remain$
    Remain<leave_or_remain$Leave,rgb(1,0,0,.7),rgb
    (0,0,1,.7))
map_data=data.frame(UK@data)
map_data=cbind(map_data,"Brexit"=0)
for (i in 1:nrow(map_data)){
if(map_data[i,"NAME_1"]=="Northern Ireland"){
map_data[i,"Brexit"]="blue"}else{
map_data[i,"Brexit"]=as.character(leave_or_remain[
    leave_or_remain$HASC_code==map_data$HASC_2[i],'
    Brexit?'])}}
plot(UK, col = map_data$Brexit, border = "gray1",
    xlim = c(-4,-2), ylim = c(50, 59), main="How the
    UK has voted?", bg="#A6CAE0")
plot(IRL, col = "lightgrey", border = "gray1",add=
    TRUE)
```



## Maps and Polygons

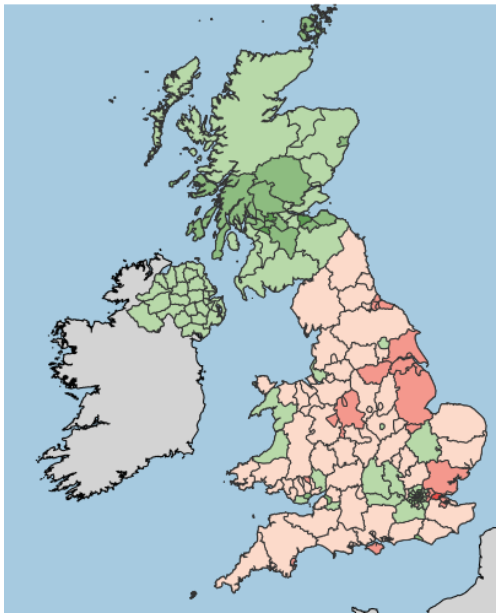
```
library(cartography)
cols <- carto.pal(pal1 = "red.pal",n1 = 5,pal2="
    green.pal",n2=5)
map_data=data.frame(UK@data)
map_data=cbind(map_data,"Percentage_Remain"=0)
for (i in 1:nrow(map_data)){
if(map_data[i,"NAME_1"]=="Northern Ireland"){
map_data[i,"Percentage_Remain"]=55.78}else{
    map_data[i,"Percentage_Remain"]=100*leave_or_
    remain[leave_or_remain$HASC_code==map_data$HASC_
    2[i],"Remain"]/(leave_or_remain[leave_or_remain$
    HASC_code==map_data$HASC_2[i],"Remain"]+leave_or_
    remain[leave_or_remain$HASC_code==map_data$HASC_
    2[i],"Leave"])}}
```



(... to be continued)

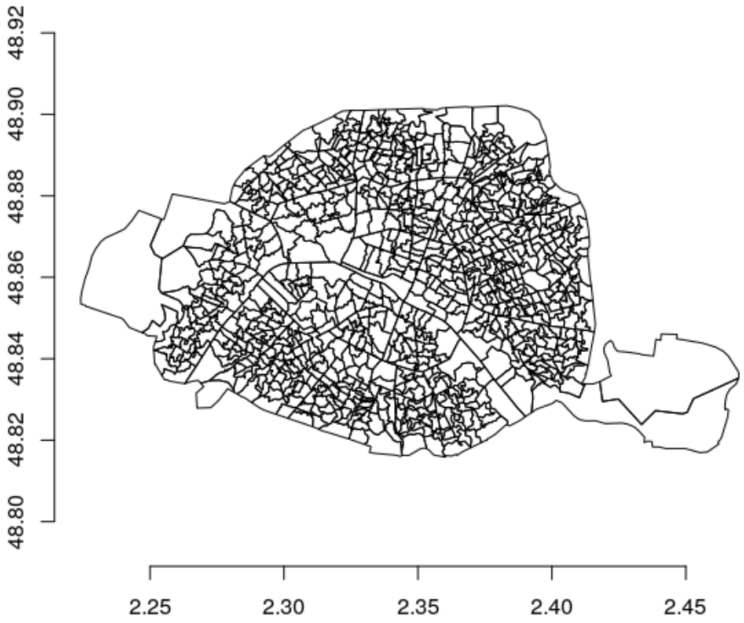
## Maps and Polygons

```
plot(UK, col = "grey", border = "gray1", xlim = c
    (-4,-2), ylim = c(50, 59),bg="#A6CAE0")
plot(IRL, col = "lightgrey", border = "gray1",add=
    TRUE)
plot(FR, col = "lightgrey", border = "gray1",add=
    TRUE)
choroLayer(spdf = UK,
df = map_data,
var = "Percentage_Remain",
breaks = seq(0,100,10),
col = cols,
legend.pos = "topright",
legend.title.txt = "",
legend.values.rnd = 2,
add = TRUE)
```



## Maps and Projections

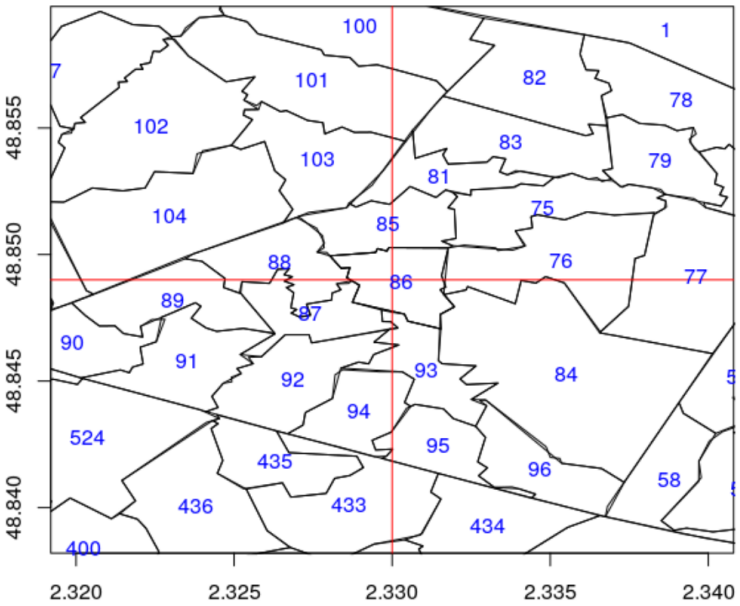
```
library (maptools)
library (rgdal)
library (PBSmapping)
download.file("http://freakonometrics.free.
    fr/paris-cartelec.zip","paris-cartelec.
    zip")
unzip("paris-cartelec.zip")
paris <- readShapeSpatial ( "paris-cartelec.
    shp" )
proj4string ( paris ) <- CRS( "+init=epsg
    :2154" )
paris <- spTransform(paris , CRS( "+proj=
    longlat +ellps=GRS80" ) )
plot ( paris )
```



## Maps and Projections

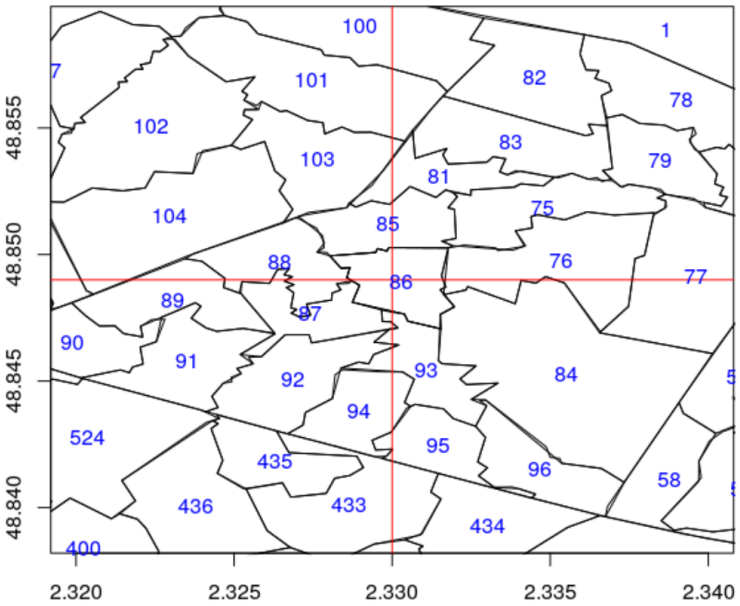
```
point=c(2.33,48.849)
plot(point,xlim=point[1]+c(-.01,.01),ylim=
    point[2]+c(-.01,.01))
plot(paris,add=TRUE)

library(rgeos)
idx=unique(poly_paris$PID)
for(i in idx){
ct=centroid(as.matrix(poly_paris[poly_paris$
    PID==i,c("X","Y")]))
text(ct[1],ct[2],i,col="blue")}
```



## Maps and Projections

```
which.poly <- function ( point ) {
 idx=unique(poly_paris$PID)
 idx_valid=NULL
 for ( i in idx ) {
 pip=point.in.polygon (point[ 1 ] , point[ 2
     ] ,
 poly_paris [ poly_paris$PID==i , "X" ] ,
 poly_paris [ poly_paris$PID==i , "Y" ] )
 if ( pip >0) idx_valid=c ( idx_valid , i )
    }
 return ( idx_valid ) }
```




Hence, here

```
> which.poly(point)
[1] 86
```

## Using Google Maps and Open Street Maps

```
library(ggmap)
library(geosphere)

> (Hotel <- geocode("8 sinaran drive singapore"))
Information from URL : http://maps.googleapis.com/maps/api/geocode/
    json?address=8+sinaran+drive+singapore&sensor=false
Google Maps API Terms of Service : http://developers.google.com/maps/
    terms
       lon      lat
1 103.845 1.320279
> (Airport <- geocode("singapore airport"))
Information from URL : http://maps.googleapis.com/maps/api/geocode/
    json?address=singapore+airport&sensor=false
Google Maps API Terms of Service : http://developers.google.com/maps/
    terms
       lon      lat
1 103.9915 1.36442
```

## Using Google Maps and Open Street Maps

The distance, in km, is obtained using the Haversine formula wikipedia.org

```
> distHaversine(Airport,Hotel, r=6378.137)
[1] 17.03514
```

while the driving distance is

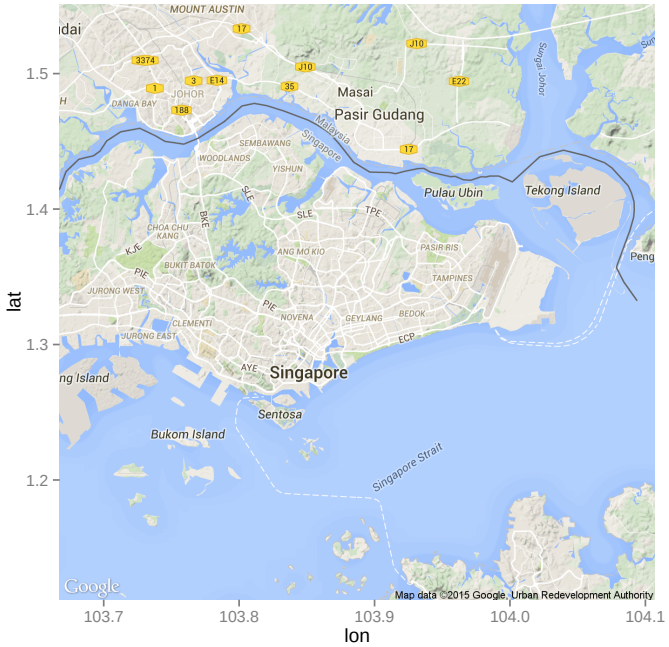
```
> mapdist(as.numeric(Hotel),as.numeric(Airport))
                                  from                                    to
      m      km     miles
1 8 Sinaran Dr, Singapore 307470 80 Airport Blvd, Singapore 819642
    19627 19.627 12.19622
  seconds  minutes     hours
1    1220 20.33333 0.3388889
```

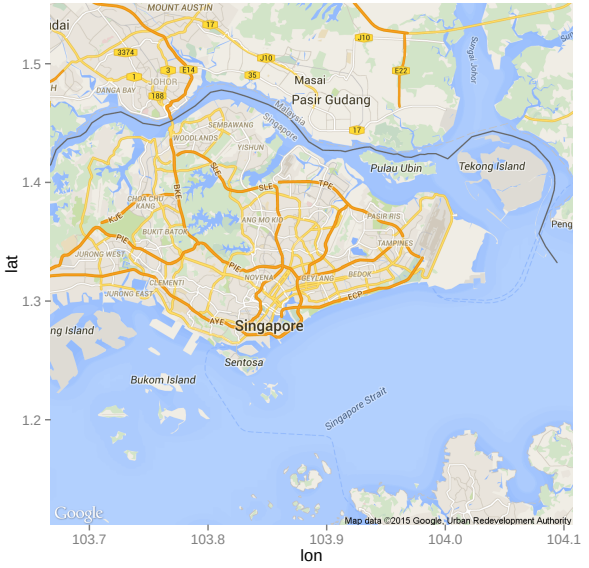
# Background Maps

```
Loc=data.frame(rbind(Airport, Hotel))
library(ggmap)
library(RgoogleMaps)
CenterOfMap <- as.list(apply(Loc,2,mean))
SG <- get_map(c(lon=CenterOfMap$lon, lat=
    CenterOfMap$lat),zoom = 11, maptype = "
    terrain", source = "google")
SGMap <- ggmap(SG)
SGMap
```

or

```
SGMap <- get_map(c(lon=CenterOfMap$lon, lat=
    CenterOfMap$lat),zoom = 11, maptype = "
    roadmap")
ggmap(SGMap)
```



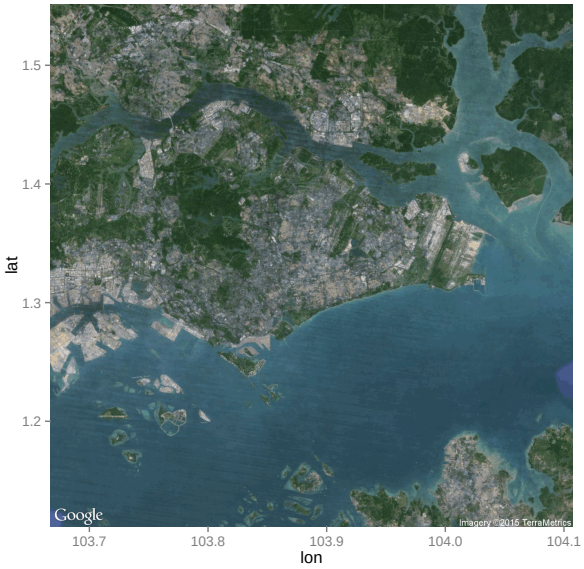
## Background Maps

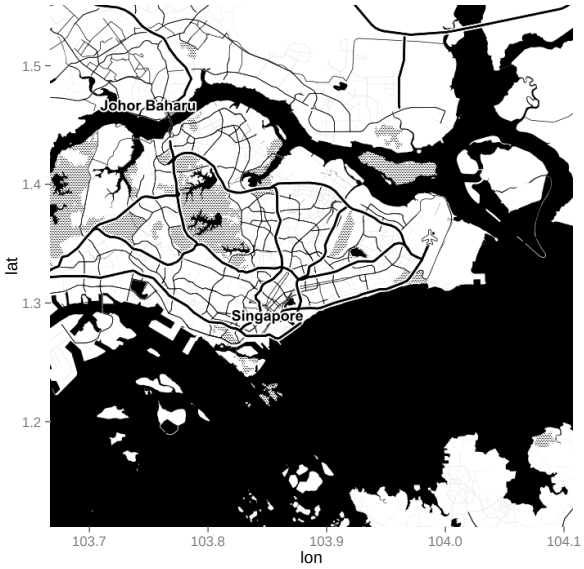
Note that those are ggplot2 maps, see ggmapCheatsheet

```
SGMap <- get_map(c(lon=CenterOfMap$lon, lat=
    CenterOfMap$lat),zoom = 11, maptype = "
    satellite")
ggmap(SGMap)
```

or

```
SGMap <- get_map(c(lon=CenterOfMap$lon, lat=
    CenterOfMap$lat),zoom = 11, maptype = "
    toner", source = "stamen")
ggmap(SGMap)
```



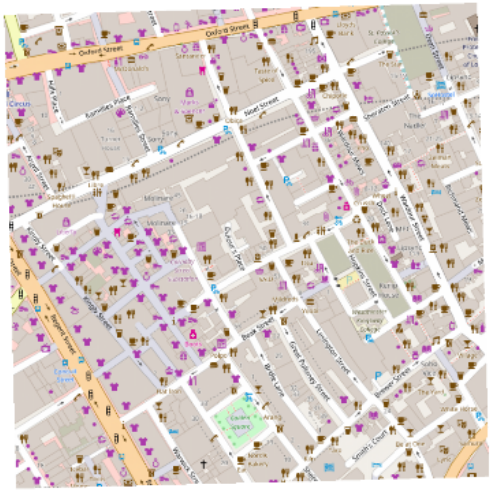



## Background Maps

```
library(OpenStreetMap)
map <- openmap(c(lat= 51.516,   lon= -.141),
     c(lat= 51.511,   lon= -.133))
map <- openproj(map, projection = "+init=
    epsg:27700")
plot(map)
```

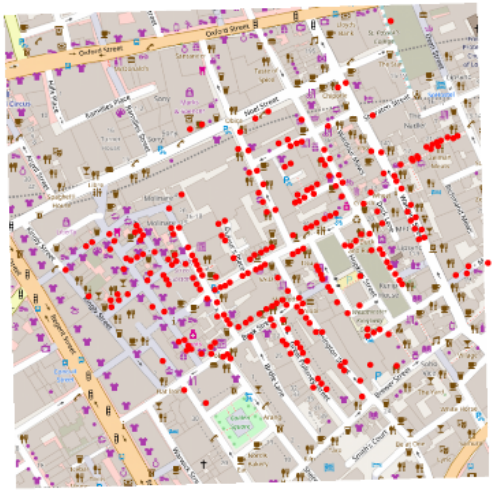
## Background Maps

```
library(OpenStreetMap,verbose=FALSE,quietly=
    TRUE,warn.conflicts=FALSE)
map <- openmap(c(lat= 51.516, lon= -.141), c
    (lat= 51.511, lon= -.133))
map <- openproj(map, projection = "+init=
    epsg:27700")
plot(map)
loc="http://freakonometrics.free.fr/cholera.
    zip"
download.file(loc,destfile="cholera.zip")
unzip("cholera.zip", exdir="./")
library(maptools,verbose=FALSE,quietly=TRUE,
    warn.conflicts=FALSE)
deaths<-readShapePoints("Cholera_Deaths")
```



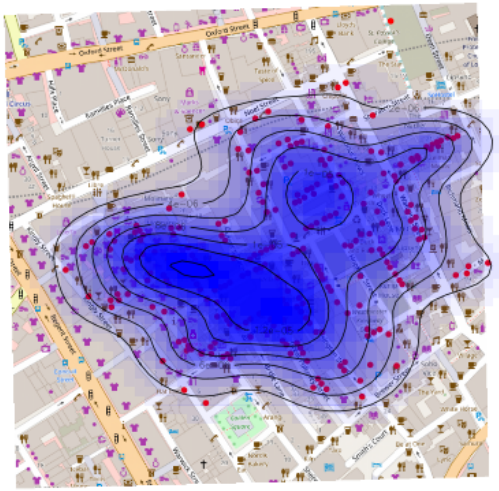
## Background Maps

```
head(deaths@coords)
##   coords.x1 coords.x2
## 0    529309    181031
## 1    529312    181025
## 2    529314    181020
## 3    529317    181014
## 4    529321    181008
## 5    529337    181006
plot(map)
points(deaths@coords,col="red", pch=19, cex
    =.7 )
```



## Background Maps

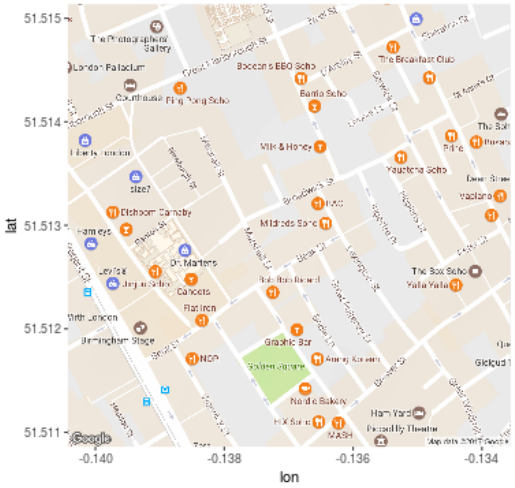
```
X = deaths@coords
library(KernSmooth)
kde2d = bkde2D(X, bandwidth=c(bw.ucv(X[,1]),
    bw.ucv(X[,2])))
library(grDevices,verbose=FALSE,quietly=TRUE
    ,warn.conflicts=FALSE)
clrs = colorRampPalette(c(rgb(0,0,1,0), rgb
    (0,0,1,1)), alpha = TRUE)(20)
plot(map)
points(deaths@coords,col="red", pch=19, cex
    =.7 )
image(x=kde2d$x1, y=kde2d$x2,z=kde2d$fhat,
    add=TRUE,col=clrs)
contour(x=kde2d$x1, y=kde2d$x2,z=kde2d$fhat,
     add=TRUE)
```



## Background Maps

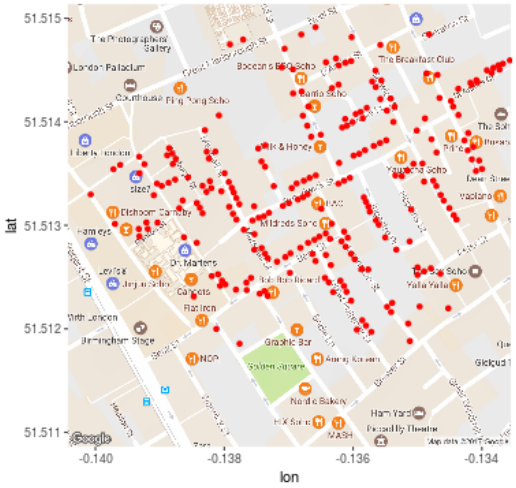
or one can use a ggplot2 visualization with Google Maps,

```
library(ggmap,verbose=FALSE,quietly=TRUE,
    warn.conflicts=FALSE)
get_london = get_map(c(-.137,51.513), zoom
    =17)
london = ggmap(get_london)
london
```



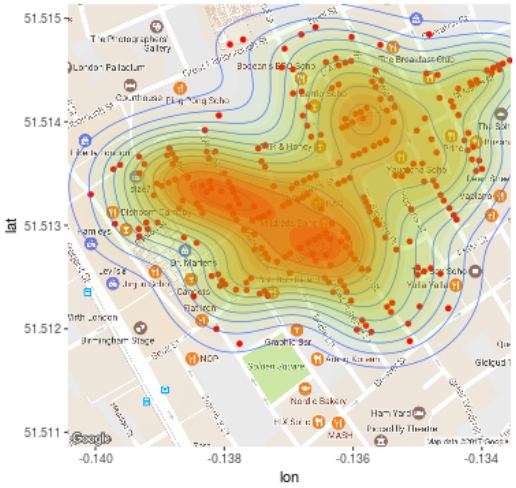
## Background Maps

```
df_deaths <- data.frame(X)
library(sp,verbose=FALSE,quietly=TRUE,warn.
    conflicts=FALSE)
library(rgdal,verbose=FALSE,quietly=TRUE,
    warn.conflicts=FALSE)
coordinates(df_deaths)=~coords.x1+coords.x2
proj4string(df_deaths)=CRS("+init=epsg:27700
    ")
df_deaths = spTransform(df_deaths,CRS("+proj
    =longlat +datum=WGS84"))
london + geom_point(aes(x=coords.x1,y=coords
    .x2),data=data.frame(df_deaths@coords),
    col="red")
```

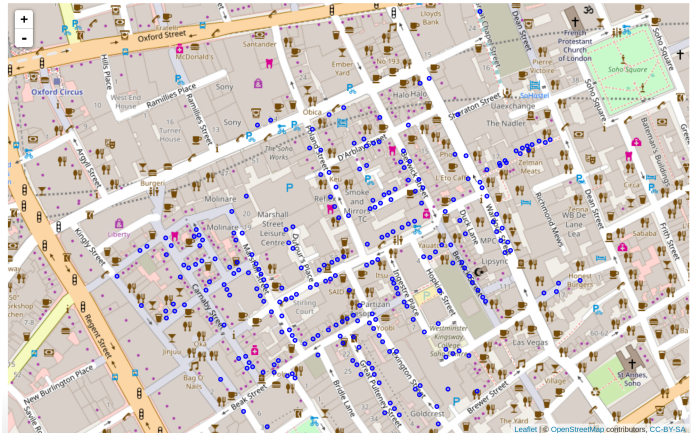


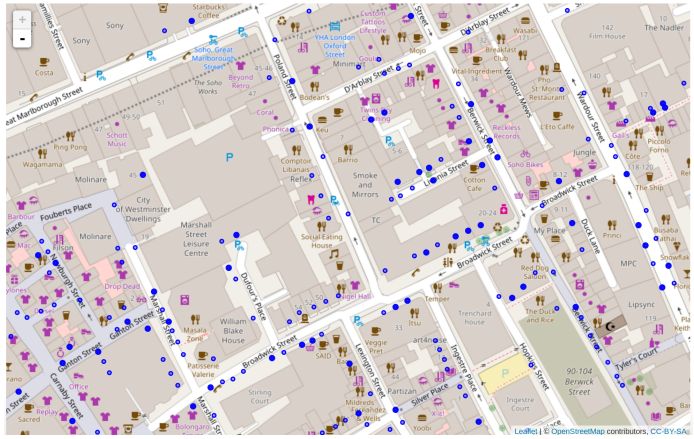
## Background Maps

```
1 london = london + geom_point(aes(x=coords.x1
    , y=coords.x2),
2 data=data.frame(df_deaths@coords),col="red")
     + geom_density2d(data = data.frame(df_
    deaths@coords), aes(x = coords.x1, y=
    coords.x2), size = 0.3) + stat_density2d
    (data = data.frame(df_deaths@coords),
    aes(x = coords.x1, y=coords.x2), size =
    0.01, bins = 16, geom = "polygon") +
    scale_fill_gradient(low = "green", high
    = "red", guide = FALSE) + scale_alpha(
    range = c(0, 0.3), guide = FALSE)
3 london
```



```
require(leaflet)
deaths <- readShapePoints("Cholera_Deaths")
df_deaths <- data.frame(deaths@coords)
coordinates(df_deaths)=~coords.x1+coords.x2
proj4string(df_deaths)=CRS("+init=epsg:27700")
df_deaths=spTransform(df_deaths,CRS("+proj=
    longlat +datum=WGS84"))
df=data.frame(df_deaths@coords)
lng=df$coords.x1
lat=df$coords.x2
m = leaflet()%>% addTiles()
m = m %>% fitBounds(-.141, 51.511, -.133,
    51.516)
rd=.5; op=.8; clr="blue"
m = leaflet() %>% addTiles()
m = m %>% addCircles(lng,lat, radius = rd,
    opacity=op,col=clr)
```



## Post Office Problem

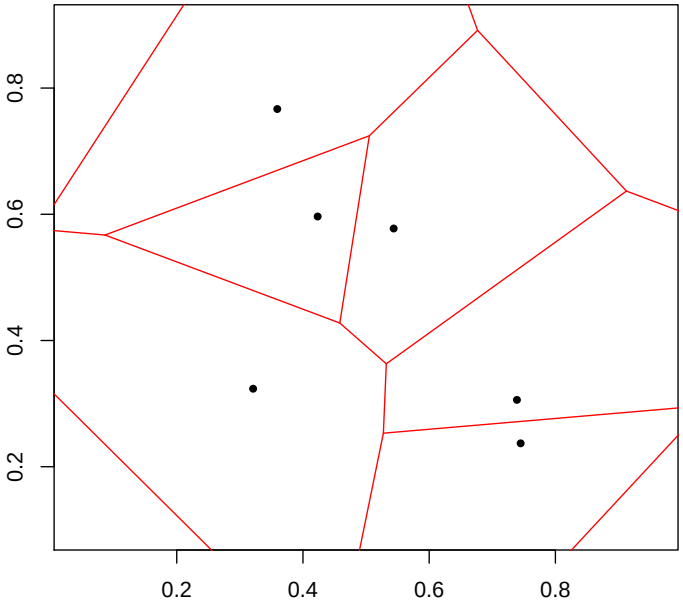
Consider two points $\boldsymbol{a}$ and $\boldsymbol{b}$ and let $B(\boldsymbol{a},\boldsymbol{b})$ denote the bissector, and let $H(\boldsymbol{a},\boldsymbol{b})$ denote the (closed) halfspace that is bounded by $B(\boldsymbol{a},\boldsymbol{b})$ that contains $\boldsymbol{a}$.

Given $n$ points $\mathcal{P}=\{\boldsymbol{p}_1,\cdots,\boldsymbol{p}_n\}$, the Voronoi cell $V_{\mathcal{P}}(i)$ of point $\boldsymbol{p}_i$ is defined as

$$V_{\mathcal{P}}(i)=\left\{\boldsymbol{x},\|\boldsymbol{x}-\boldsymbol{p}_i\|\leq\|\boldsymbol{x}-\boldsymbol{p}\|,\forall\boldsymbol{p}\in\mathcal{P}\right\}$$

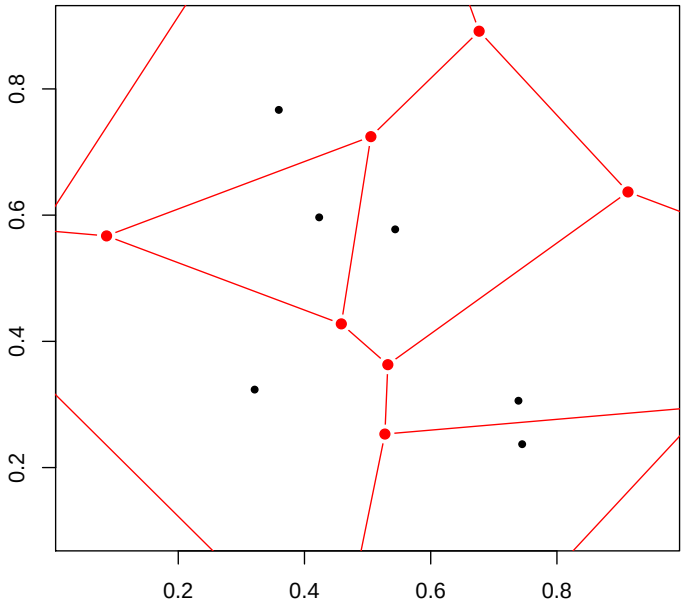
Observe that $V_{\mathcal{P}}(i)=\bigcap_{j\neq i}H(\boldsymbol{p}_i,\boldsymbol{p}_j)$

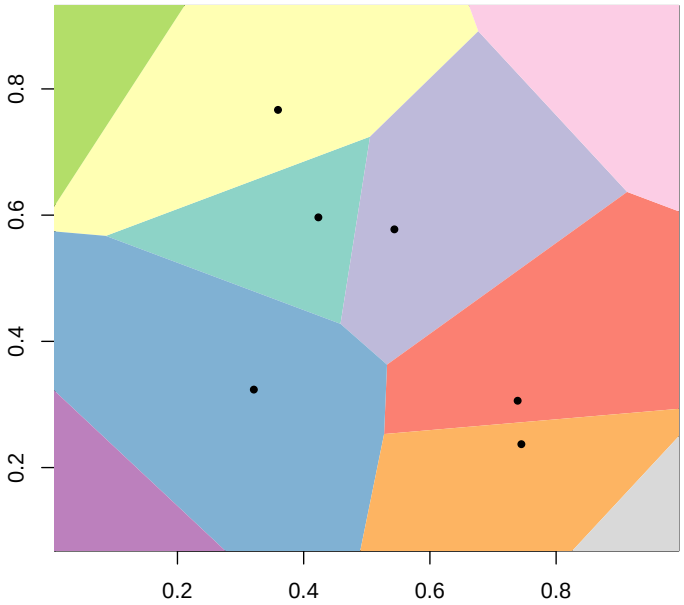
One can easily prove that $V_{\mathcal{P}}(i)\neq\emptyset$ is a convex set. Let $VD(\mathcal{P})$ denote the Voronoi diagram is the subdivision of the plane induced by Voronoi cells.



# Post Office Problem

```
x <- rnorm(50, 0, 1.5); y <- rnorm(50, 0, 1)
library(deldir)
vtess <- deldir(x, y)
plot(x, y, pch=20, col="black", cex=1)
plot(vtess, wlines="tess", wpoints="none", number=FALSE, add=TRUE)
```



## Post Office Problem

Let $VV(\mathcal{P})$, $VE(\mathcal{P})$ and $VR(\mathcal{P})$ denote respectively the set of vertices, edges and regions (faces).

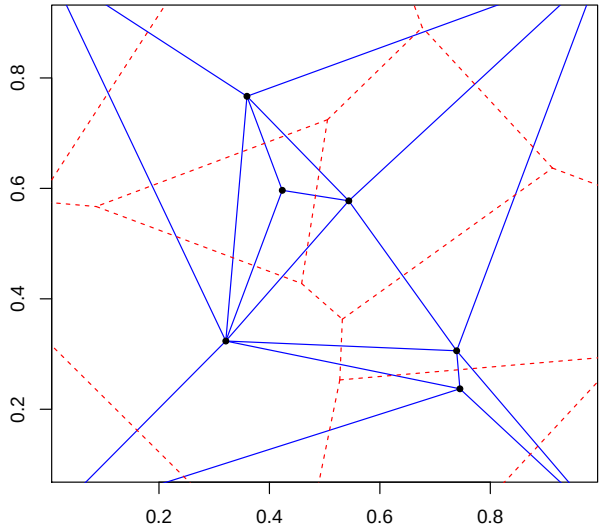
For every vertice $v \in VV(\mathcal{P})$, $v$ is the common intersection of at least three edges from $VE(\mathcal{P})$ (exactly three, generally).

For $n \geq 3$, the number of vertices in the Voronoi diagram of a set of $n$ point sites in the plane is at most $2n - 5$ and the number of edges is at most $2n - 6$

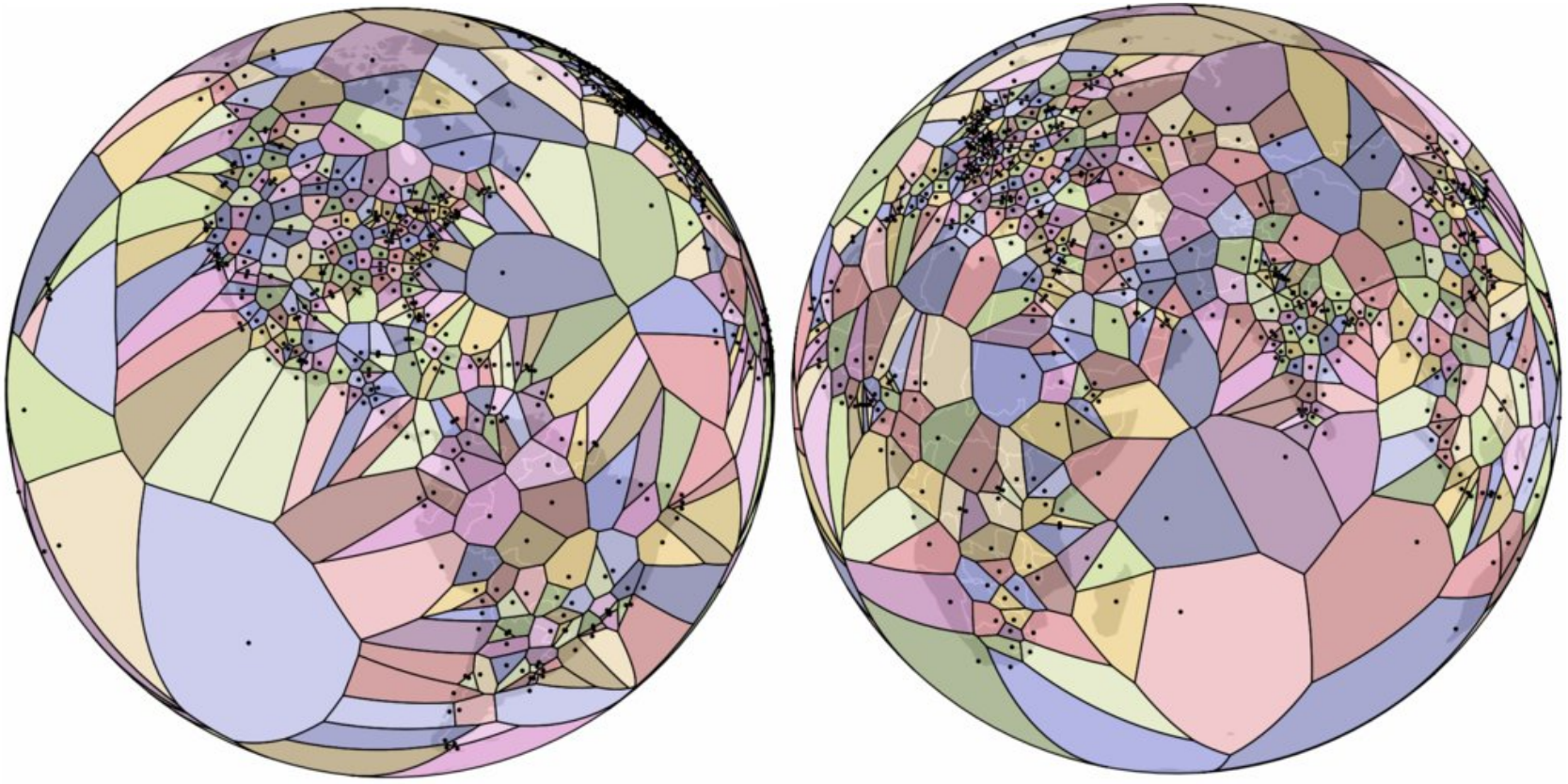
## Post Office Problem

**Proposition** The straight-line dual of $VD(\mathcal{P})$ for a set of points $\mathcal{P}$ is the unique Delaunay triangulation.

```
plot(x, y, type="n", asp=1,xlab="",ylab="")
plot(vtess, wlines="triang", wpoints="none",
     number=FALSE, add=TRUE, lty=1,col="blue
    ")
points(x, y, pch=20, col="black", cex=1)
```

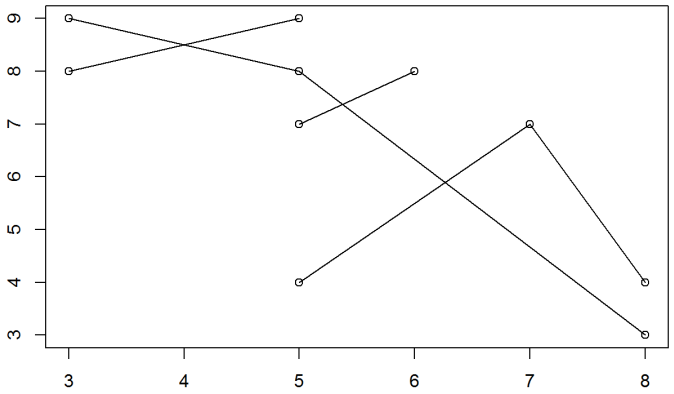


# Voronoi Regions for Airports



## Voronoi Sets with Networks

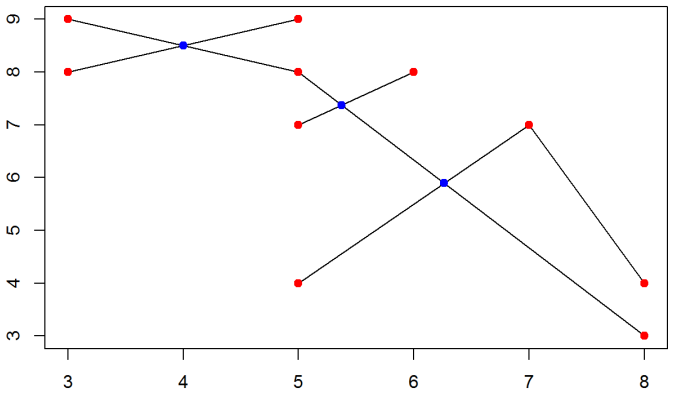
```
dd <- data.frame(Id=c(1,1,1,2,2,2,3,3,4,4),
                 X=c(3,5,8,5,7,8,3,5,5,6),
                 Y=c(9,8,3,4,7,4,8,9,7,8))
plot(dd$X,dd$Y)
for(i in unique(dd[,1])) lines(dd$X[dd$Id==i
    ],dd$Y[dd$Id==i])
library(shapefiles)
library(geosphere)
library(sp)
library(rgeos)
```



## Voronoi Sets with Networks

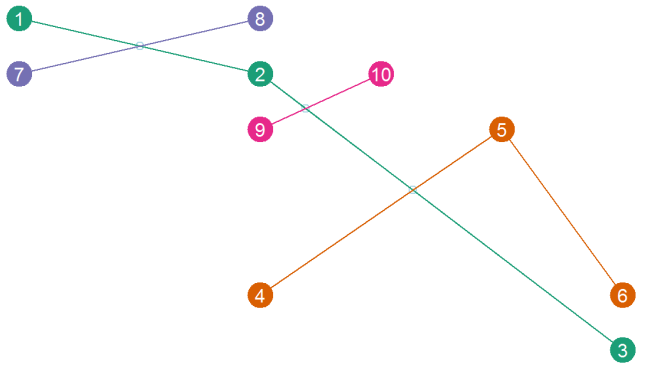
```
source("http://freakonometrics.free.fr/
    extendshp.R")
dd2 <-  extend_shp(dd)
plot(dd$X,dd$Y,col="white")
for(i in unique(dd[,1])) lines(dd$X[dd$Id==i
    ],dd$Y[dd$Id==i])
points(dd2[["knots"]]$X,dd2[["knots"]]$Y,pch
    =19,col="blue")
points(dd$X,dd$Y,pch=19,col="red")

library(RColorBrewer)
darkcols <- brewer.pal(8, "Dark2")
```



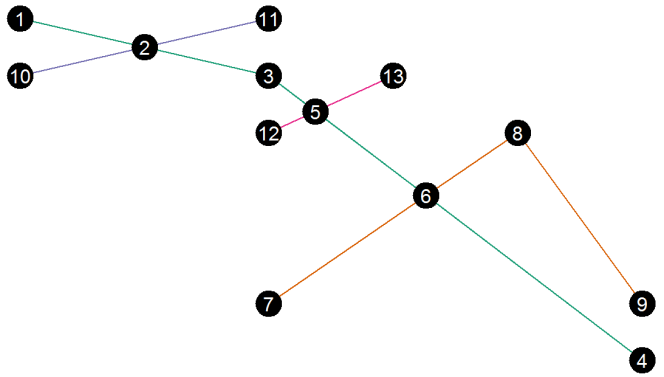
## Voronoi Sets with Networks

```
plot(dd2[["shp"]]$X,dd2[["shp"]]$Y,col="light blue",xlab="",ylab="",axes=FALSE)
for(i in 1:nrow(dd2$connected)) segments(dd2[["knots"]]$X[dd2[["connected"]]$knot1[i]],dd2[["knots"]]$Y[dd2[["connected"]]$knot1[i]], dd2[["knots"]]$X[dd2[["connected"]]$knot2[i]], dd2[["knots"]]$Y[dd2[["connected"]]$knot2[i]], col=darkcols[dd2[["connected"]]$Id[i]])
points(dd$X,dd$Y,pch=19,col=darkcols[dd$Id],cex=3)
text(dd$X,dd$Y,1:nrow(dd),col="white")
```



## Voronoi Sets with Networks

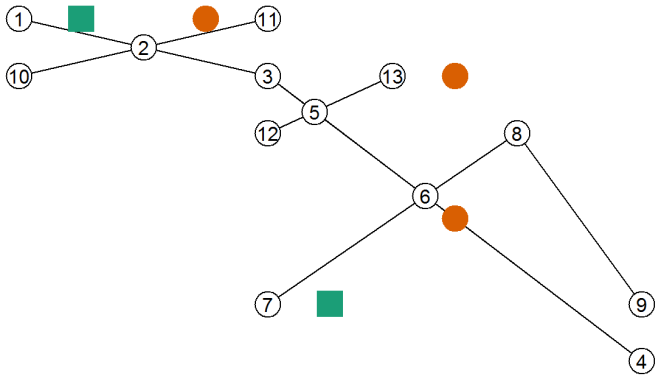
```
plot(dd2[["shp"]]$X,dd2[["shp"]]$Y,col="white",xlab="",ylab="",axes=FALSE)
for(i in 1:(nrow(dd2$connected)/2)) segments(dd2[["knots"]]$X[dd2[["connected"]]$knot1[i]],dd2[["knots"]]$Y[dd2[["connected"]]$knot1[i]], dd2[["knots"]]$X[dd2[["connected"]]$knot2[i]], dd2[["knots"]]$Y[dd2[["connected"]]$knot2[i]], col=darkcols[dd2[["connected"]]$Id[i]])
points(dd2[["knots"]]$X,dd2[["knots"]]$Y,pch=19,col="black",cex=3)
text(dd2[["knots"]]$X,dd2[["knots"]]$Y,dd2[["knots"]]$NO,col="white")
```



## Voronoi Sets with Networks

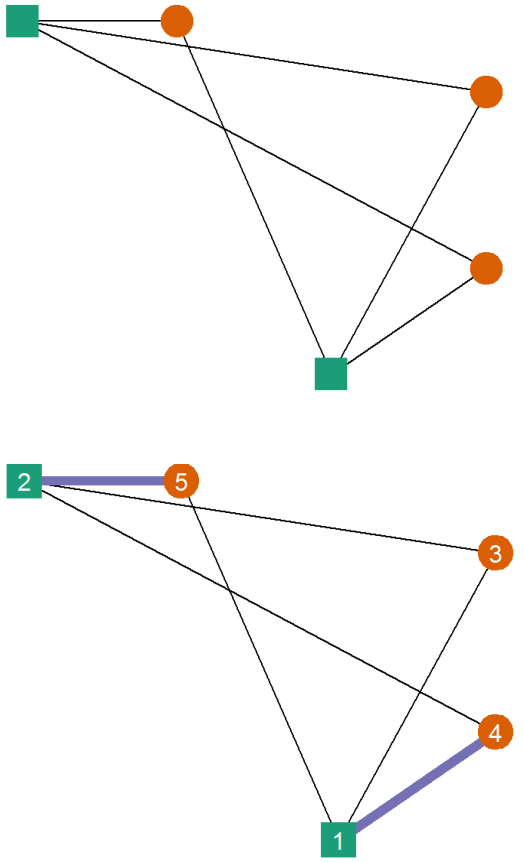
```
agents=data.frame(X=c(5.5,3.5,6.5,6.5,4.5), Y=c(4,9,8,5.5,9), Type=as
    .factor(c("Demand","Demand","Supply","Supply","Supply")))
plot(dd2[["knots"]]$X,dd2[["knots"]]$Y,pch=19,col="white",cex=3)
for(i in 1:(nrow(dd2$connected)/2)) segments(dd2[["knots"]]$X[dd2[["
    connected"]]$knot1[i]],dd2[["knots"]]$Y[dd2[["connected"]]$knot1[i
    ]], dd2[["knots"]]$X[dd2[["connected"]]$knot2[i]], dd2[["knots"]]$
    Y[dd2[["connected"]]$knot2[i]])
```

```
text(dd2[["knots"]]$X,dd2[["knots"]]$Y,dd2[[
    "knots"]]$NO,col="black")
points(agents$X,agents$Y,pch=forme[agents$
    Type],col=darkcols[agents$Type],cex=3)
```



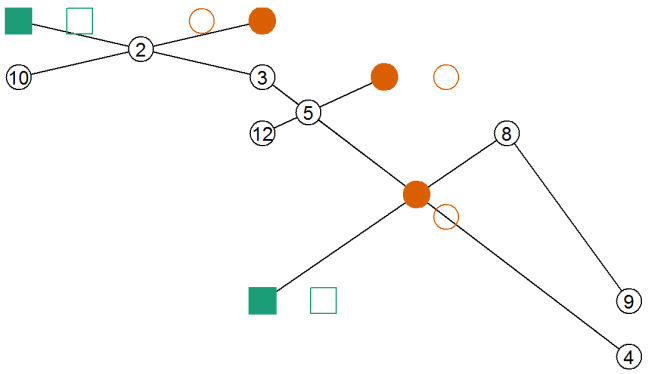
```
close_supply <- function(coord,knots=agents[agents$Type=="Supply",]){
    knots[which.min(distHaversine( coord[,c("X","Y")], knots[,c("X","Y
    ")], r =6378.137)),]}
```

```
list_demand <- which(agents$Type=="Demand")
plot(agents$X,agents$Y,pch=forme[agents$Type],col=
    darkcols[agents$Type],cex=3.2)
for(i in which(agents$Type=="Demand")) segments(
    agents[i,"X"],agents[i,"Y"],agents[agents$Type
    =="Supply","X"],agents[agents$Type=="Supply","
    Y"])
for(i in list_demand){
supply=unlist(close_supply(agents[i,]))
segments(agents[i,"X"],agents[i,"Y"],supply["X"],
    supply["Y"],col=darkcols[3],lwd=6)}
text(agents$X,agents$Y,1:nrow(agents),pch=forme[
    agents$Type],col="white")
```



## Voronoi Sets with Networks

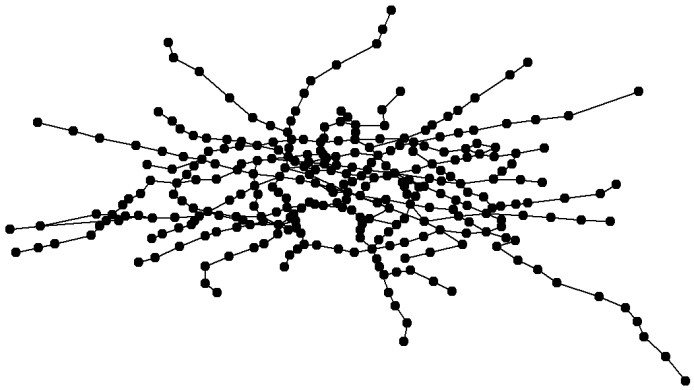
```
close_knot=function(coord,knots=dd2$knots){
  knots[which.min(distHaversine( coord[,c("X","Y")] ,  knots[,c("X","Y")] , r=6378.137)),]
}
shift_agents=agents
shift_agents$Xk=shift_agents$X
shift_agents$Yk=shift_agents$Y
shift_agents$NO=0
for(i in 1:nrow(shift_agents)) shift_agents[i,c("Xk","Yk","NO")]=close_knot(agents[i,])
```



## Voronoi Sets with Networks

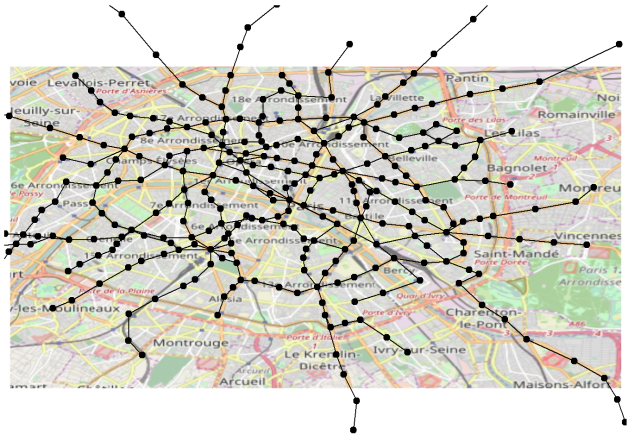
```
download.file("http://freakonometrics.free.fr/arcs.csv","arcs.csv")
download.file("http://freakonometrics.free.fr/nodes.csv","nodes.csv")
arcs = read.csv(file = "arcs.csv",sep=";", header=FALSE)
nodes = read.csv("nodes.csv",sep=";", header=FALSE)
subarcs=arcs[(arcs$V1<303)&(arcs$V2<303),]
plot(nodes[1:302,c("V3","V4")],pch=19,xlab="",ylab="",axes=FALSE)
```

```
for(i in 1:nrow(arcs)) segments(nodes[
    subarcs$V1[i],"V3"], nodes[subarcs$V1[i
    ],"V4"], nodes[subarcs$V2[i],"V3"],
    nodes[subarcs$V2[i],"V4"])
```



## Voronoi Sets with Networks

```
library(OpenStreetMap)
map=openmap(c(lat=48.9,lon=2.26),c(lat=48.8,lon
    =2.45))
map=openproj(map,projection="+init=epsg:4326")
plot(map)
points(nodes[1:302,c("V3","V4")],pch=19)
for(i in 1:nrow(arcs)) segments(nodes[subarcs$
    V1[i],"V3"], nodes[subarcs$V1[i],"V4"],
    nodes[subarcs$V2[i],"V3"], nodes[subarcs$V2
    [i],"V4"])
```

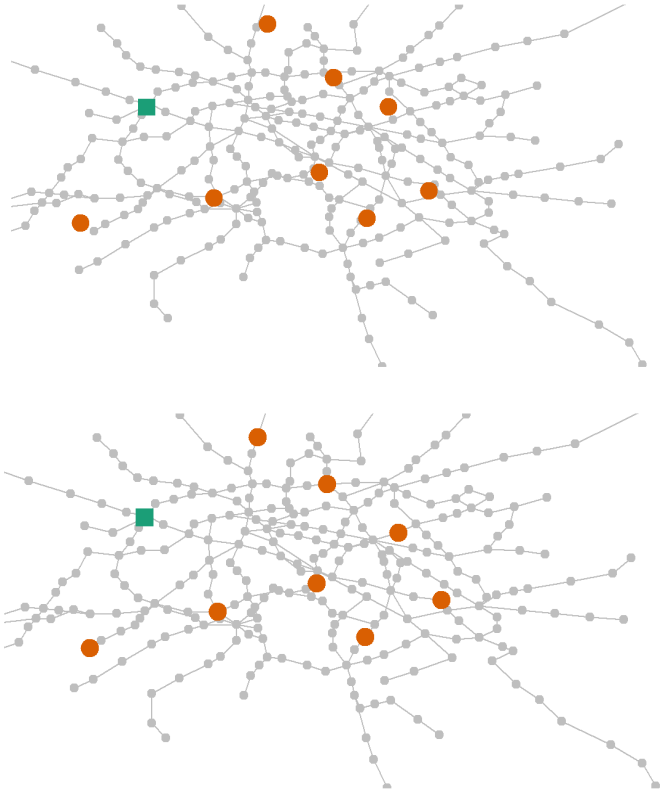


## Voronoi Sets with Networks

```
adresses <- c("47 boulevard de l'hopital Paris", "20 rue leblanc
    Paris", "2 rue ambroise pare paris", "1 place parvis notre dame
    paris", "149 rue de sevres paris", "1 avenue claude vellafaux
    paris", "184 faubourg saint antoine paris", "46 rue henri huchard
    paris")
adresses <- c("place etoile paris")
library(ggmap)
sick <- lapply(adresses,geocode)
hospital <- lapply(adresses,geocode)
```

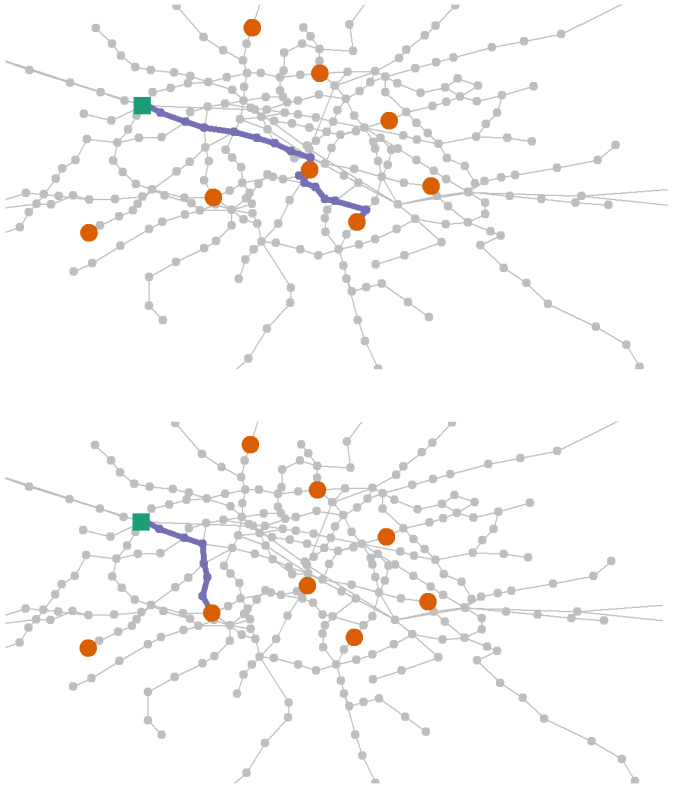
## Voronoi Sets with Networks

```
plot(nodes[1:302,c("V3","V4")],pch=19,xlim=c
    (2.26,2.45),ylim=c(48.8,48.9),xlab="",
    ylab="",col="grey",axes=FALSE)
points(unlist(hospital)[names(unlist(
    hospital))=="lon"],unlist(hospital)[
    names(unlist(hospital))=="lat"],
        pch=forme[2],col=darkcols[2],cex=2)
points(unlist(sick)[names(unlist(sick))=="
    lon"],unlist(sick)[names(unlist(sick))==
    "lat"],
        pch=forme[1],col=darkcols[1],cex=2)
```



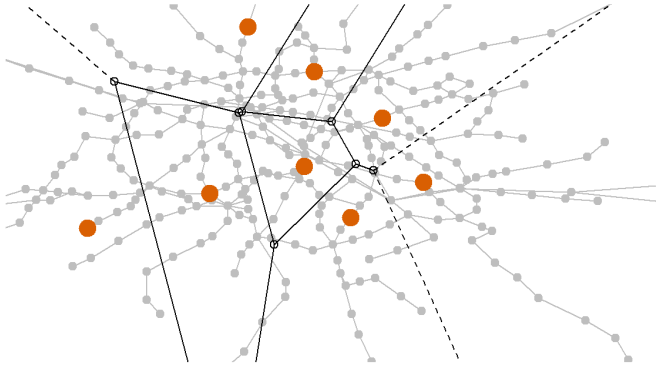
## Voronoi Sets with Networks

```
newhospital=data.frame(X=rep(NA,length(
    hospital)),Y=rep(NA,length(hospital)),NO
    =rep(NA,length(hospital)))
for(i in 1:length(hospital)) newhospital[i
    ,]=as.numeric(close_knot(hospital[[i]]))
newsick=data.frame(X=rep(NA,length(sick)),Y=
    rep(NA,length(sick)),NO=rep(NA,length(
    sick)))
for(i in 1:length(sick)) newsick[i,]=as.
    numeric(close_knot(sick[[i]]))
library(igraph)
g <- make_graph(edges = as.vector(rbind(arcs
    [,1],arcs[,2])),directed=FALSE)
sp <- shortest_paths(g, newsick[1,"NO"],
    newhospital[1,"NO"], weights= arcs[,3])
pth <- as.numeric(sp$vpath[[1]])
```



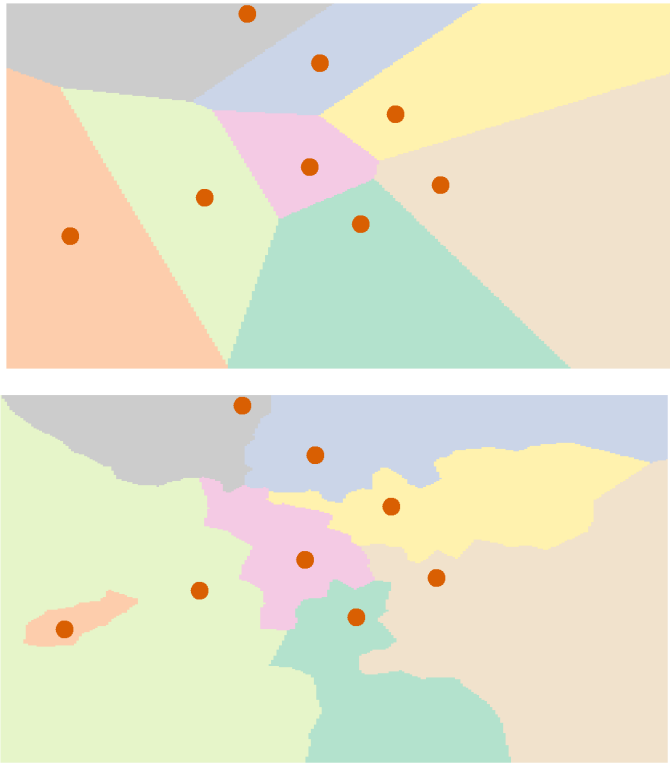
## Voronoi Sets with Networks

```
plot(nodes[,c("V3","V4")],pch=19,xlim=c
    (2.26,2.45),ylim=c(48.8,48.9),xlab="",
    ylab="",col="grey",axes=FALSE)
for(i in 1:nrow(arcs)) segments(nodes[arcs$
    V1[i],"V3"], nodes[arcs$V1[i],"V4"],
    nodes[arcs$V2[i],"V3"], nodes[arcs$V2[i
    ],"V4"], col="grey")
points(newhospital[,"X"],newhospital[,"Y"],
    pch=forme[2],col=darkcols[2],cex=2)
library(tripack)
loc=agents[agents$Type=="Supply",c("X","Y")]
names(loc)=c("x","y")
m=voronoi.mosaic(loc)
plot(m,add=TRUE)
```



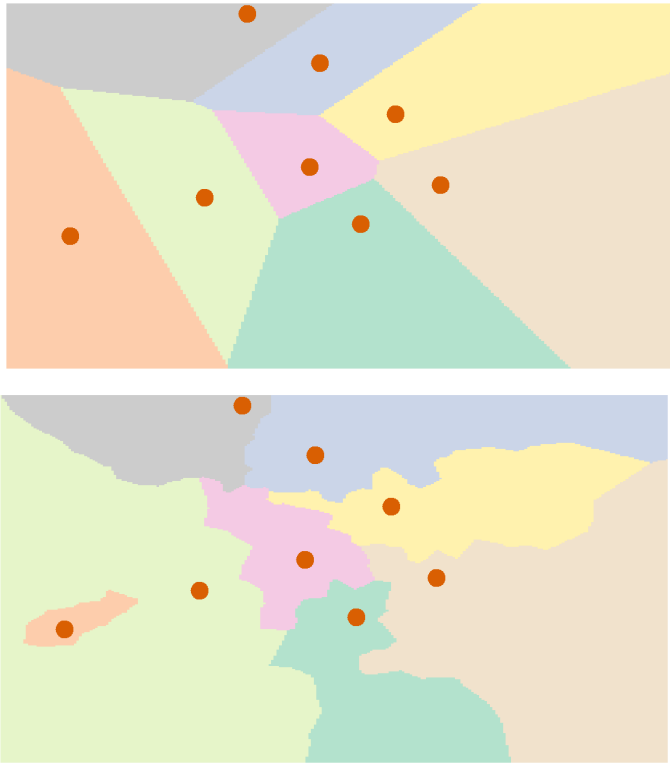
## Voronoi Sets with Networks

```
affect1=function(x,y){ which.min(
    distHaversine( c(x,y) ,  loc[,c("x","y")
    ] , r =6378.137))}
vx=seq(2.26,2.45,length=251)
vy=seq(48.8,48.9,length=251)
vz1=matrix(NA,251,251)
for(i in 1:251){ for(j in 1:251){
    vz1[i,j]=affect1(vx[i],vy[j])}}
lightcols <- brewer.pal(8, "Pastel2")
image(vx,vy,vz1,xlim=c(2.26,2.45),ylim=c
    (48.8,48.9),xlab="",ylab="",axes=FALSE,
    col=lightcols)
points(newhospital[,"X"],newhospital[,"Y"],
    pch=forme[2],col=darkcols[2],cex=2)
```



## Voronoi Sets with Networks

```
affect2=function(x,y){
pt=as.numeric(close_knot(data.frame(lon=x,
    lat=y)))
  for(j in 1:sum(agents$Type=="Supply")){
    d[j]=distances(g, pt[3], newhospital[j,"
    NO"], weights= arcs[,3])  }
which.min(d)}
vz2=matrix(NA,251,251)
for(i in 1:251){
  for(j in 1:251){
    vz2[i,j]=affect2(vx[i],vy[j])  } }
```



## Sports Ranking (NBA, NHL, etc)

| team | wins | losses | to play | against | | | |
|---|---|---|---|---|---|---|---|
| $i$ | $w_i$ | $l_i$ | $r_i$ | A | B | C | D |
| A | 83 | 71 | 8 | - | 1 | 6 | 1 |
| B | 80 | 79 | 3 | 1 | - | 0 | 2 |
| C | 78 | 78 | 6 | 6 | 0 | - | 0 |
| D | 77 | 82 | 3 | 1 | 2 | 0 | - |

Which teams have a chance of finishing season with most wins?

D eliminated since it can finish with at most 80 wins, (A already has 83)

**Proposition** If $w_i + r_i < w_j$ for some $j$, then team $i$ is eliminated

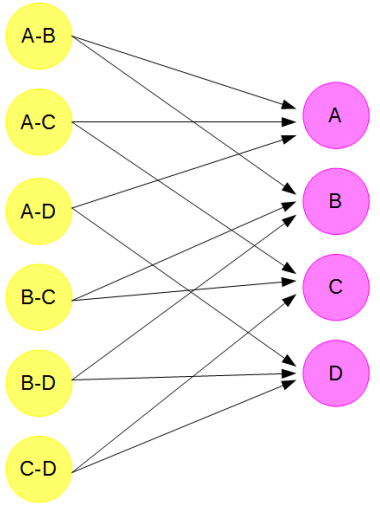
B can win 83, but will be eliminated...

C can win only if A loses all its game

## Sports Ranking (NBA, NHL, etc)

Is a (specific) team eliminated ? see Schwartz (1966) Possible Winners in Partially Completed Tournaments

Idea: use max-flow theory...
Let $r_{ij}$ denote the number of remaining games between $i$ and $j$
Consider a bipartite graph with games nodes on one side, team nodes on the other side




**Theorem** : Team $i$ not eliminated if and only if max flow saturates all arcs leaving source.

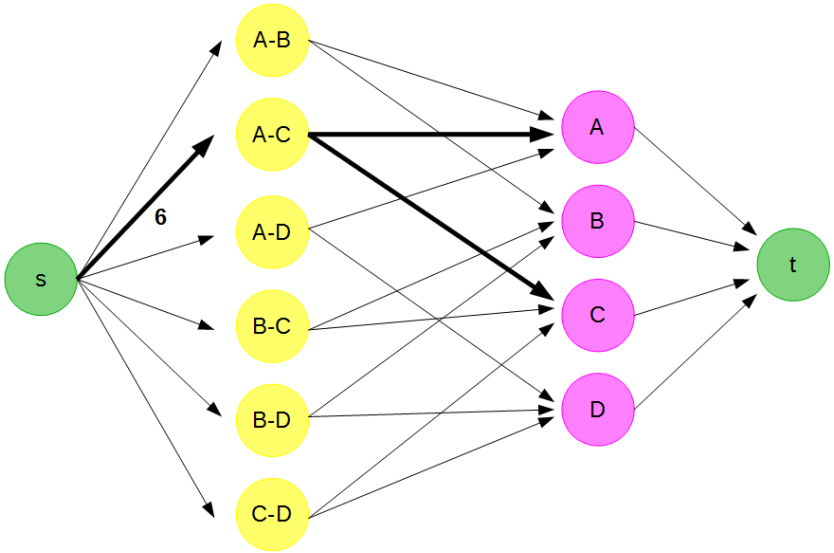
## Sports Ranking (NBA, NHL, etc)

Let $R$ denote a subset of teams, then set

$$w(R) = \sum_{i \in R} w_i \text{ and } r(R) = \frac{1}{2} \sum_{i,j \in R} r_{ij}$$

Define $a(R) = \dfrac{w(R) + r(R)}{|R|}$

If $a(R) > w_i + r_i$ then $i$ is eliminated by a team in $R$.




**Theorem** : Team $i$ is eliminated iff $\exists R$ that eliminates $i$.

## Sports Ranking (NBA, NHL, etc)

**Property** : If team $j$ is eliminated and $w_i + r_i \leq w_j + r_j$, then $i$ is also eliminated.

Proof : if $j$ is eliminated, there is $R$ such that $a(R) > w_j + r_j \geq w_i + r_i$

Either $i \notin R$, the $R$ eliminates $i$, or $i \in R$, and then $R \backslash \{i\}$ eliminates $i$.

Thus, one can proof the following result

**Theorem** : There is a threshold $T$ such that $w_i + r_i \leq T$ if and only if $i$ is eliminated.

Thus, the goal is to find the smallest value $T$ for which max-flow saturates all source arcs

## Modeling Congestion in Networks

In networks, we've seen how to go from $u$ to $v$ using the shortest path. But in real applications, one should also consider congestion, as in Galichon (2016).

For a given node $v$ and a given edge $e$, consider the node-incidence matrix $N$ defined as

$$N_{e,v} = \begin{cases} +1 \text{ if e is out of u} \\ \text{-1 if e is into u} \\ 0 \text{ otherise} \end{cases}$$

Conservation equation - Kirchoff's law - states that

$$b_v = \sum_{u:(u,v)\in E} F_{u,v} - \sum_{u:(v,u)\in E} F_{v,u}$$

which can also be written $\boldsymbol{NF} = \boldsymbol{b}$, i.e.

$$\sum_{e\in E} N_{u,e}F_e = b_u \text{ for all } u \in V.$$

Feasible flows are defined as $\boldsymbol{F}$ such that $\boldsymbol{F} \geq \boldsymbol{0}$ and $\boldsymbol{NF} = \boldsymbol{b}$.

Minimizing the total cost of transport under a feasibility constraint means that we solve the min-flow problem

$$\min_{\boldsymbol{F}} \left\{ \sum_{(u,v)\in E} F_{u,v} k_{u,v} \right\} \text{ subject to } \boldsymbol{F} \geq \boldsymbol{0} \text{ and } \boldsymbol{NF} = \boldsymbol{b}.$$

This primal problem has dual problem - also called max-cut problem

$$\max_{\boldsymbol{w}} \left\{ \sum_{v\in V} w_v b_b \right\} \text{ subject to } w_v - w_u \leq k_{u,v}.$$

From a social planner perspective, consider the following program

$$\min \{\mathcal{W}(\pi)\} \text{ s.t. } \boldsymbol{F} \geq \boldsymbol{0} \text{ and } N\boldsymbol{F} = b$$

for some total transportation cost function $\mathcal{W}$. Classically, consider a separable

function

$$\mathcal{W}(\boldsymbol{F}) = \sum_{(u,v)\in E} K_{u,v}(F_{u,v})$$

for some real valued functions $K_{u,v}$.

Congestions can be captured with convex functions $K_{u,v}$

If $\mathcal{W}$ is a convex function, the primal value of the optimal transportation problem on the network

$$\min\{\mathcal{W}(\boldsymbol{F})\} \text{ s.t. } \boldsymbol{F} \geq \boldsymbol{0} \text{ and } N\boldsymbol{F} = b$$

coincides with its dual value,

$$\max\left\{\sum_{v\in V} w_v b_b - \mathcal{W}^\star(\boldsymbol{w}^\mathsf{T}\boldsymbol{N})\right\}$$

where $\boldsymbol{w}^\mathsf{T}\boldsymbol{N}$ is the matrix with entries $w_j - w_i$, and where $\mathcal{W}^\star$ is the convex

conjugate function of $\mathcal{W}$, in the sense that

$$\mathcal{W}^\star(x) = \sup_{\boldsymbol{F}} \left\{ \sum_{(u,v)\in E} F_{u,v}x_{u,v} - \mathcal{W}(\boldsymbol{F}) \right\}.$$

Note that in the limiting case where $\mathcal{W}$ is linear, i.e.

$$\mathcal{W}(\boldsymbol{F}) = \sum_{(u,v)\in E} k_{u,v}F_{u,v},$$

we recovert the min-flow / max-cut theorem.

Dial's Model, from Dial (1971). Consider the case where

$$\mathcal{W}(\boldsymbol{F}) = \sum_{(u,v)\in E} k_{u,v}F_{u,v} + \sigma \sum_{(u,v)\in E} F_{u,v}\log[F_{u,v}]$$

then there is a vector $\boldsymbol{w}$ such that the optimal flow satisfies

$$F_{u,v} = \exp\left(\frac{-k_{u,v} + w_v - w_u - 1}{\sigma}\right)$$

see Galichon (2016) for a comprehensive proof.

All other things equal, all transitions are possible, but less costly transitions will be more likely than others.

From an individual perspective, with a so-called selfish routing problem willing to go from the source $s$ to the sink $t$, we add $\delta b$ to the network, that yield to an increment $\delta \boldsymbol{F}$ on the flow.

On the edge $(u, v)$, additional transportation cost $\delta F_{u,v}$ is added, which is a function of the saturation of the network, $k_{i,j}(F_{u,v})$.

$\boldsymbol{F}$ is a Wardrop equilibrium (from Wardrop (1952)) if it solves

$$\min\left\{\sum_{(u,v)\in E} K_{u,v}(F_{u,v})\right\} \text{ s.t. } N\boldsymbol{F}=b$$

where $K_{u,v}$ is a primitive of $k_{u,v}$ ($k_{u,v}=K'_{u,v}$).

The proof is based on Kuhn & Tucker conditions (see wikipedia) on a local version of the problem (which will be linear). Note that an equilibrium flow $\boldsymbol{F}$ is

not necessarily optimal. The optimal flow solves

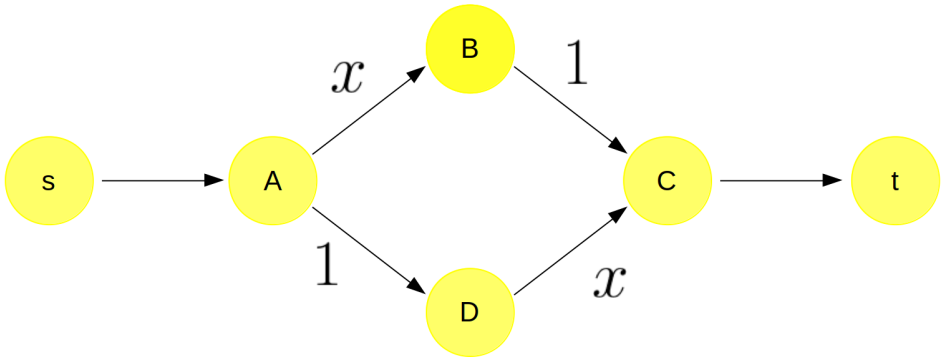
$$\min\left\{\sum_{(u,v)\in E} k_{u,v}(F_{u,v})\right\} \text{ s.t. } N\boldsymbol{F}=b$$

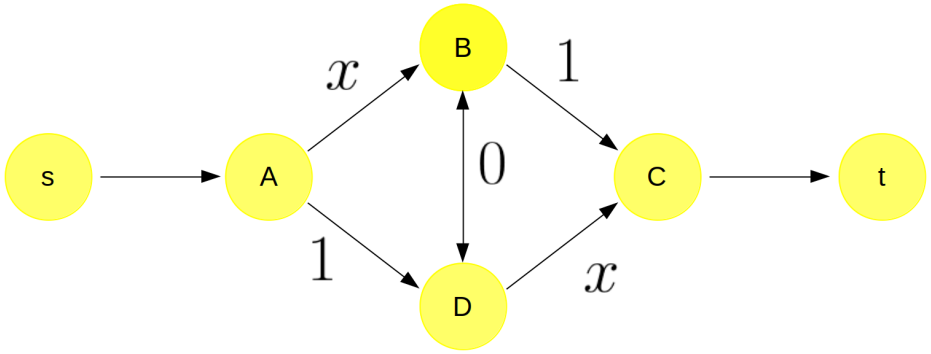
which is different when $k$'s are not linear.

The difference between optimal $\boldsymbol{F}$ and equilibrium $\boldsymbol{F}$ is sometimes called price of anarchy, see Koutsoupias & Christodoulou (1999).

There might also be strange paradoxes, such as the popular Braess paradox (from Braess (1968)).

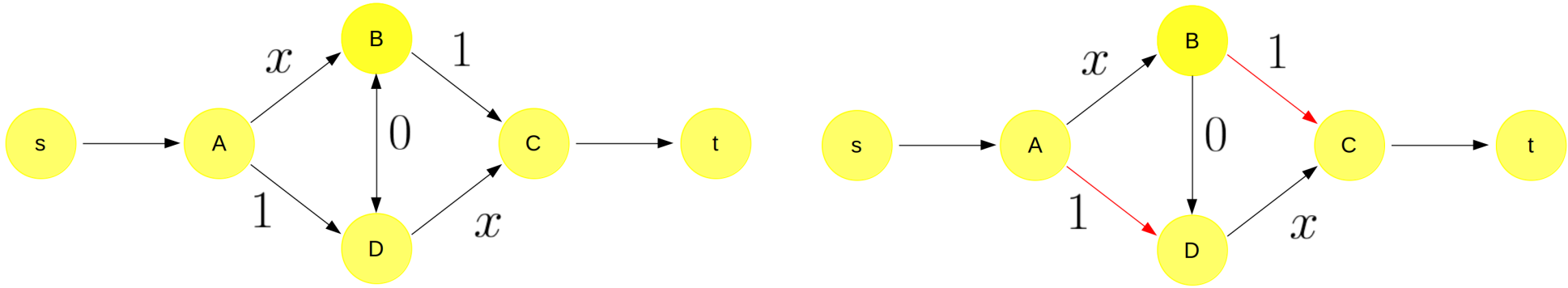
Consider the following networks, with the following costs (either $x$ or 1)



On the one on the left, the unique equilibrium is to split the flow in two paths, $\{s, \mathsf{A}, \mathsf{B}, \mathsf{C}, t\}$ and $\{s, \mathsf{A}, \mathsf{D}, \mathsf{C}, t\}$ and the cost per infinitesimal unit is $3/2$, on both ways. Total cost is here $3/2$ which is the optimal value.

On the one on the right, an additiona road is considered, the edge $(\mathsf{B}, \mathsf{D})$. That road does not change the optimal flow. But it does affect the equilibrium, since $\{s, \mathsf{A}, \mathsf{B}, \mathsf{D}, \mathsf{C}, t\}$ is now a shortest path, and the total cost is now 2 : this new road create congestion !




Conversly, one can prove that closing a road might improve the traffic...