Reproducible Example - Betweenness Accessibility

This notebook presents a documented and reproducible example of the betweenness accessibility approach (see companion paper).

Preliminaries

Begin by clearing the workspace and loading all necessary packages.

```r
rm(list=ls())
```

```r
library(sp)
library(rgdal)
library(plyr)
library(spdep)
library(maptools)
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.5.3
```

```r
library(reshape) #for melt
library(igraph)
library(spatstat)
```

```
## Warning: package 'spatstat' was built under R version 3.5.3
```

```
## Warning: package 'spatstat.data' was built under R version 3.5.3
```

```r
library(doParallel)
library(tmap)
```

This example uses a sample road network. This is a small extract of a much larger network file that was obtained from Census of Canada:

```r
ntw<-readOGR(dsn = ".",
             layer = "Simple Network_shp",
             verbose = TRUE,
             encoding = "latin1")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: ###deleted##
## It has 4 fields
## Integer64 fields read as strings:  ID
```

The network is an object of class SpatialLinesDataFrame, and it is projected:

```r
summary(ntw)
```

```
## Object of class SpatialLinesDataFrame
## Coordinates:
##         min       max
## x -79.87486 -79.84999
## y  43.25013  43.26550
## Is projected: FALSE
## proj4string : [+proj=longlat +ellps=GRS80 +no_defs]
## Data attributes:
##        ID        LENGTH            DIR    CLASS
##  1      :1   Min.   :0.2631   Min.   :0   20:8
##  10     :1   1st Qu.:0.4635   1st Qu.:0   23:6
##  11     :1   Median :0.6273   Median :0
##  12     :1   Mean   :0.6482   Mean   :0
##  13     :1   3rd Qu.:0.8145   3rd Qu.:0
##  14     :1   Max.   :1.1817   Max.   :0
##  (Other):8
```

Data Preparation

The network as obtained from the Census does not have speeds on links. Instead it has rank codes and class codes that describe the links. These attributes can be used to impute speed. For instance, a link of class 10 is a highway, whereas a link of class 23 is a local street. Speed values can be imputed based on the class of the link and speed limits obtained from the road classification system for Toronto:

```
# Initialize speeds
ntw$speed <- 0

# Impute speeds based on class
ntw$speed[which(ntw$CLASS %in% c(10:12))] <- 90
ntw$speed[which(ntw$CLASS %in% c(20,21))] <- 60
ntw$speed[which(ntw$CLASS %in% c(22,25))] <- 50
ntw$speed[which(ntw$CLASS %in% c(23,29,80))] <- 40
ntw$speed[is.na(ntw$CLASS)] <- 40
```

The next step is to extract the nodes of the network. Links can be composed of multiple line segments, which means that not every point in a link is necessarily a node. For this reason, it is important to distinguish between the starting and ending nodes of a link and other points.

This process begins by copying the network to a new object:

```
osm <- ntw
```

The nodes will be the coordinates of the lines in the network. The following extracts the coordinates for all points:

```
a <- lapply(slot(osm, "lines"), function(x) lapply(slot(x, "Lines"), function(y) slot(y, "coords")))
```

Then, we can determine the number of points per line:

```
aa <- lapply(a,function(x) length(unlist(x))/2)
```

This gives the coordinates of the points in a list, as well as the number of points in each link in the network. Based on this it is possible to extract the coordinates as lists of matrices:

```
d <- lapply(a, '[[', 1) #extract coords as list with sub matrices
```

Create an index as a list. The index is for the total number of links in the network (as a list to use `lapply` later on):

```
index <- as.list(1:length(d))
```

Retrieve the coordinates of the starting nodes. These are the coordinates of the first point in every link. Notice that `sapply` will return an array instead of a list. Transpose so that the two columns are the coordinates and each row is a node:

```r
d2 <- t(sapply(d, function (x) x[1,]))  #coords of starting nodes
```

Next, retrieve the coordinates of the ending nodes. These are the coordinates of the last point in every link:

```r
d3 <- t(sapply(index, function(x) d[[x]][aa[[x]],])) #coords of ending nodes
```

Store the nodes as dataframes:

```r
start_c <- data.frame(long = d2[,1], lat = d2[,2])
end_c <- data.frame(long = d3[,1], lat = d3[,2])
```

Convert the nodes to a `SpatialPoints` object and copy the projection from the network:

```r
nodes_st <- SpatialPoints(start_c)
nodes_end <- SpatialPoints(end_c)
nodes_st@proj4string <- ntw@proj4string
nodes_end@proj4string <- ntw@proj4string
```

The coordinates of the nodes can be appended to `osm` (which until now is still identical to `ntw`)

```r
# Starting nodes
osm$long_st <- nodes_st$long
osm$lat_st<-nodes_st$lat

#Ending nodes
osm$long_end <- nodes_end$long
osm$lat_end <- nodes_end$lat
```

Notice that many of the nodes are actually duplicates, since most starting points for links are also ending points of other links. Remove the duplicates (the original set of nodes has 28 points):

```r
nodes <- spRbind(nodes_st, nodes_end)
nodes <- remove.duplicates(nodes, zero = 0.0)
```

After removing duplicates, nodes now has 12.

Add an ID column to match with links, and the coordinates can be added to the data slot of the `SpatialPointsDataFrame`:

```r
nodes$node_ID <- c(1:length(nodes))
nodes$node_long <- nodes$long
nodes$node_lat <- nodes$lat
```

Match the IDs of the nodes to the links. This can bw done by using `dplyr` join functions, with multi-criteria keys:

```r
junk <- left_join(osm@data, nodes@data, by = c("long_st" = "node_long", "lat_st" = "node_lat"))
osm$node_ID_st <- junk$node_ID
junk <- left_join(osm@data, nodes@data, by = c("long_end" = "node_long", "lat_end" = "node_lat"))
osm$node_ID_end <- junk$node_ID
```

The network has now been linked to the node IDs. Now we add the length of the links and travel time, based on the imputed speeds as discussed above:

```r
osm$length <- SpatialLinesLengths(osm, longlat = TRUE)# in km
osm$fftt <- 3600*osm$length/osm$speed   # free flow travel time in sec.
```

Convert to a graph

Once that the network data have been prepared, the `igraph` package is used for analysis. For this, the trasportation network needs to be converted into a graphical network object, which includes the free flow travel time variable as an attribute:

```
edges <- graph.data.frame(d = osm@data[,c("node_ID_st","node_ID_end","fftt")], directed = FALSE)
```

Also, create a graphical network object with weights. The weights are the free flow travel times on the links:

```
edges_w  <- edges
E(edges_w)$weight<-osm$fftt
```

Zonal system

The next step is to load the shape file with the zoning system. This file is a small extract of Census geography, concretely Dissemination Areas (DAs). The source file was obtained from here:

```
zones <- readOGR(dsn =  ".",layer = "Simple Zoning_shp", verbose = TRUE, encoding = "latin1")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: ###deleted##
 ## with 9 features
## It has 10 fields
## Integer64 fields read as strings:  ID POPULATION EMPLOYMENT
```

```
zones$POPULATION <- as.numeric(as.vector(zones$POPULATION))
zones$EMPLOYMENT <- as.numeric(as.vector(zones$EMPLOYMENT))
```

Check that `zones` and `osm` use the same projection:

```
zones@proj4string
```

```
## CRS arguments: +proj=longlat +ellps=GRS80 +no_defs
```

```
osm@proj4string
```

```
## CRS arguments: +proj=longlat +ellps=GRS80 +no_defs
```

Ok.

Verify that the zones include information on population and employment at the zones:

```
summary(zones@data)
```

```
##        ID          AREA           COLORING           TRACT     CMA
##  928029 :1   Min.   :0.3154   Min.   :0.000   5370035.00:1   537:9
##  937544 :1   1st Qu.:0.3698   1st Qu.:1.000   5370036.00:1
##  948699 :1   Median :0.4447   Median :2.000   5370037.00:1
##  948719 :1   Mean   :0.4673   Mean   :1.889   5370048.00:1
##  959416 :1   3rd Qu.:0.5705   3rd Qu.:3.000   5370049.00:1
##  959434 :1   Max.   :0.5893   Max.   :4.000   5370050.00:1
##  (Other):3                                    (Other)   :3
##  PROVINCE       NAME    ABBREV   POPULATION     EMPLOYMENT
##  00:9     0035.00:1   ON:9   Min.   :1658   Min.   : 750
##           0036.00:1          1st Qu.:2464   1st Qu.:1250
##           0037.00:1          Median :3243   Median :1250
##           0048.00:1          Mean   :2942   Mean   :2942
##           0049.00:1          3rd Qu.:3395   3rd Qu.:4082
##           0050.00:1          Max.   :4341   Max.   :8000
##           (Other):3
```

Accessibility analysis is conducted at the level of zones, but requires the network, so we need to link the zones and network. To do this, we need to obtain the centroids of the zones:

```
centroids <- SpatialPointsDataFrame(coordinates(zones), data=zones@data)
```

Next, we overlay the nodes with the zones to append the zone IDs to the nodes. Note that the zones (DAs) are used for this:

```
overlay <- over(nodes, zones)
nodes$TRACT <- overlay$TRACT
```

Identify the nodes that are closest to a zonal centroid:

```
centroid_node <- nncross(as.ppp(centroids), as.ppp(nodes), what = "which", k = 1)
```

Add a factor to indicate whether a node is a centroid node:

```
nodes$centroid_node <- FALSE
nodes$centroid_node[centroid_node] <- TRUE
```

Subset the centroid nodes:

```
centroid_nodes <- subset(nodes, nodes$centroid_node == TRUE)
```

Append the zonal data to `centroid_nodes`:

```
centroid_nodes@data <- left_join(centroid_nodes@data, zones@data, by = "TRACT")
```

Betweenness accessibility

Now that a network that includes the zonal attributes is prepared, betweenness accessibility can be implemented.

To do this, the shortest paths between nodes need to be found and characterized; the objective is to identify the nodes that are in each shortest path. Test this by finding the shortest paths from one node to all:

```
sps <- get.shortest.paths(graph = edges_w,
                          from = centroid_nodes$node_ID[1],
                          to = centroid_nodes$node_ID,
                          mode = "out",
                          output = "epath") #epath to obtain the sequence of edges that enter in each path
```

This implements the calculations of shortest paths using a parallel routine:

```
registerDoParallel(cores = 4)
n_folds <- length(centroid_nodes)

epaths <- foreach(k = icount(n_folds), .packages='igraph') %dopar% {
  result <- get.shortest.paths(graph = edges_w,
                          from = centroid_nodes$node_ID[k],
                          to = centroid_nodes$node_ID,
                          mode = "out",
                          output = "epath")
  return(result)
}
```

This bit of code will match the links to their edge identifiers (stored in object `edges_w`). The parallel implementation is with `foreach` over the origin nodes, whereas the list apply `lapply` is done over the paths for each origin node:

```
list_links_sequence <- foreach(k = icount(n_folds), .packages='igraph') %dopar% {
  result <- lapply(X = as.list(1:length(epaths[[k]]$epath)), function(X) match(epaths[[k]]$epath[[X]],E
  return(result)
}
```

Next, calculate the cost for each OD pair according to the shortest paths:

```
OD_fftt <- distances(edges_w, v = centroid_nodes$node_ID,
                    to = centroid_nodes$node_ID,
                    mode = "out")
```

The list with the sequence of links needs to be melted, but before melting all the integer (0)s must be changed to NAs:

```
for (k in 1:n_folds){
  idx <- lapply(list_links_sequence[[k]], length) == 0
  list_links_sequence[[k]][idx] <- NA
}
```

Melt:

```
f <- melt(list_links_sequence)
```

Then summarize the links, where `hits` is the number of times that a link is part of a shortest path, and `zones` is the zone of origin:

```
links_sum <- ddply(f, "value", summarise, hits=length(L2), zones=list(L2))
```

Remove NAs from the dataframe `links_sum`:

```
links_sum <- filter(links_sum, !value %in% c(NA))
```

Next, append the `hits` (i.e., number of times that a link is part of a shortest path) to the network object `osm`:

```
osm$hits <- 0
osm$hits[links_sum$value] <- links_sum$hits
```

After this a cost matrix a cost matrix is created. First, calculate the travel time and length of each path:

```
# calculate the length and fftt per pair of OD
paths_fftt<-list()
paths_length<-list()
#for (i in 1:) {
for (i in 1:length(list_links_sequence)){
  #print(i)
  paths_length[[i]]<-lapply(X=as.list(1:length(list_links_sequence)), function(X){
    length=sum(osm$length[list_links_sequence[[i]][[X]]])
    return(length)
    }
    )
  paths_fftt[[i]]<-mclapply(X=as.list(1:length(list_links_sequence)), function(X) {
    fftt=sum(osm$fftt[list_links_sequence[[i]][[X]]])
    return(fftt)
    }
  )
}
```

This is the cost matrix (free flow travel time):

```
# create O-D matrices with length and tt
od_tt<-data.frame(matrix(NA, nrow = nrow(zones), ncol = nrow(zones)))
colnames(od_tt)<-zones$TRACT
rownames(od_tt)<-zones$TRACT
od_tt<-t(sapply(X=as.list(1:nrow(od_tt)),function(X) (unlist(paths_fftt[[X]])))) # must transpose it
```

Obtain an impedance matrix, in this case using a made up distance-decay function:

```
impedance_fftt <- exp(-0.05 * OD_fftt) - diag(length(zones)) #comment this to remove self-potential
```

Calculate accessibility to employment. This is the impedance matrix multiplied by employment at the destination. Store as a matrix to make it easier to decompose accessibility for origin-destination pairs later.

IMPORTANT: Note that the matrix is sorted by the node IDs, NOT the tract IDs. Therefore, this calculation makes use of the data in `centroid_nodes` rather than `zones`:

```
acc_empl_mat <- impedance_fftt * matrix(rep(centroid_nodes$EMPLOYMENT, length(zones)),
                               ncol = length(zones), nrow = length(zones), byrow = TRUE)
```

The sum over the rows is the accessibility for the centroid of origin. Match to the zonal ID before joining to zones:

```
acc_empl <- data.frame(TRACT = centroid_nodes$TRACT, acc_empl = rowSums(acc_empl_mat))
```

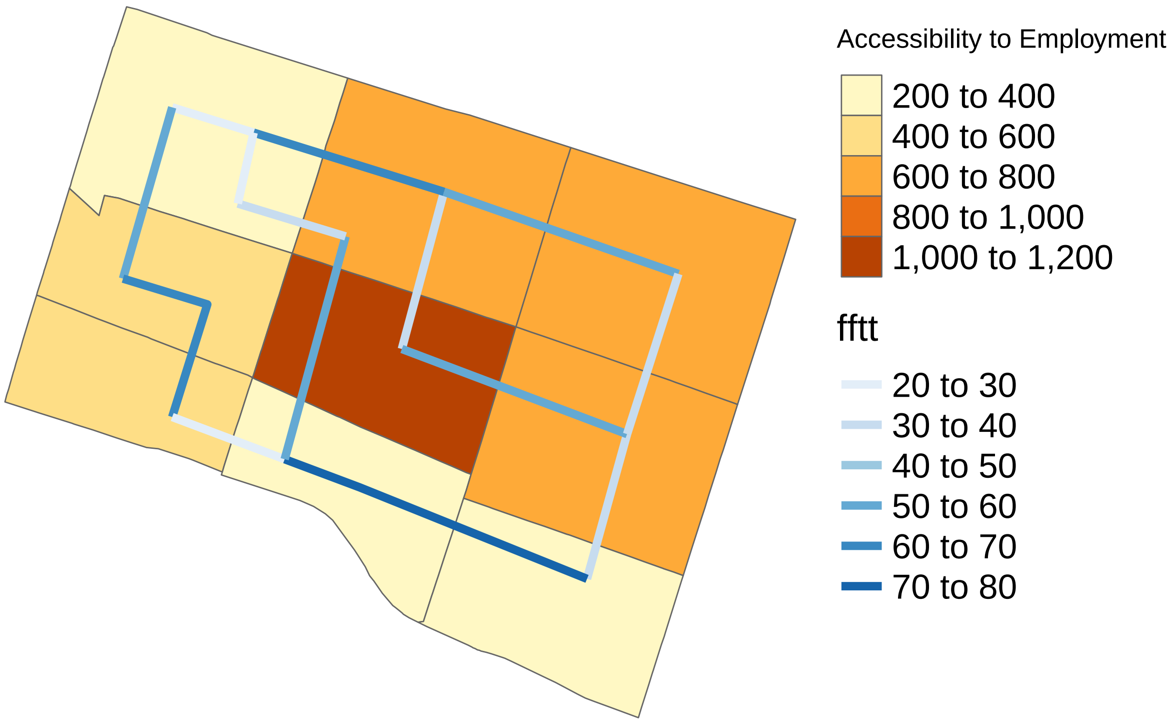
Join to `zones`:

```
zones@data <- left_join(zones@data, acc_empl)
```

```
## Joining, by = "TRACT"
```

Plot accessibility and free flow travel time on the network:

```
tm_shape(zones) + tm_borders() +
  tm_fill("acc_empl", title = "Accessibility to Employment") +
  tm_shape(osm) +  tm_lines(col = "fftt", title.col = "fftt",
  palette = "Blues", scale = 1, legend.lwd.show = FALSE,lwd=6)+
  tm_layout(frame = F, legend.text.size = 1.5, legend.title.size = 2,scale=0.7,legend.outside = T)
```



Next, if we divide the accessibility per origin-destination pair by the total accessibility of the corresponding zone we obtain the contribution of accessibility of each pair to the total accessibility of the zone:

```
acc_empl_mat_st <- acc_empl_mat / matrix(rep(rowSums(acc_empl_mat), length(zones)),
                                  ncol = length(zones), nrow = length(zones), byrow = FALSE)
```

This can be multiplied by population at the zone of origin in order to obtain number of people that are pushed from the origin to every destination, based on the proportion of accessibility that the destination generates for the origin:

```
pop_btw_mat <- acc_empl_mat_st * matrix(rep(centroid_nodes$POPULATION, length(zones)), ncol = length(zo
```

Push the values of population to the links. Initialize a vector of betweenness accessibility (`btw_acc`) with as many entries as there are links (i.e., the length of `osm`):

```
btw_acc_mat <- matrix(0, nrow = length(osm), ncol = length(zones))
```

Then, loop through the list of links in `list_link_sequence`. This is a list of lists; the first list is for the first origin, the second list for the second origin, the third list for the third origin, etc. Within each of these lists, the first list is for the first *destination*, the second list is for the second *destination*, etc. Then, the elements of the list are the numbers of the links on the path between origin i and destination j. (Note that there will be NAs when there is no path between i and j and also between i and i). Since the betweenness accessibility vector was initialized with zeros, in a loop the values will be updated as the sum of the previous betweenness accessibility, plus the share of the population that passes through the link in the way between origin i and destination j:

```
for (i in 1:length(zones)){
  for (j in 1:length(zones)) {
    if (!is.na(sum(unlist(list_links_sequence[[i]][j])))){
      btw_acc_mat[unlist(list_links_sequence[[i]][j]), i] <- btw_acc_mat[unlist(list_links_sequence[[i]]
    }
  }
}
colnames(btw_acc_mat) <- paste("btw_acc_node_", centroid_nodes$node_ID, sep = "")
```

Calculate the total betweenness-accessibility for links:

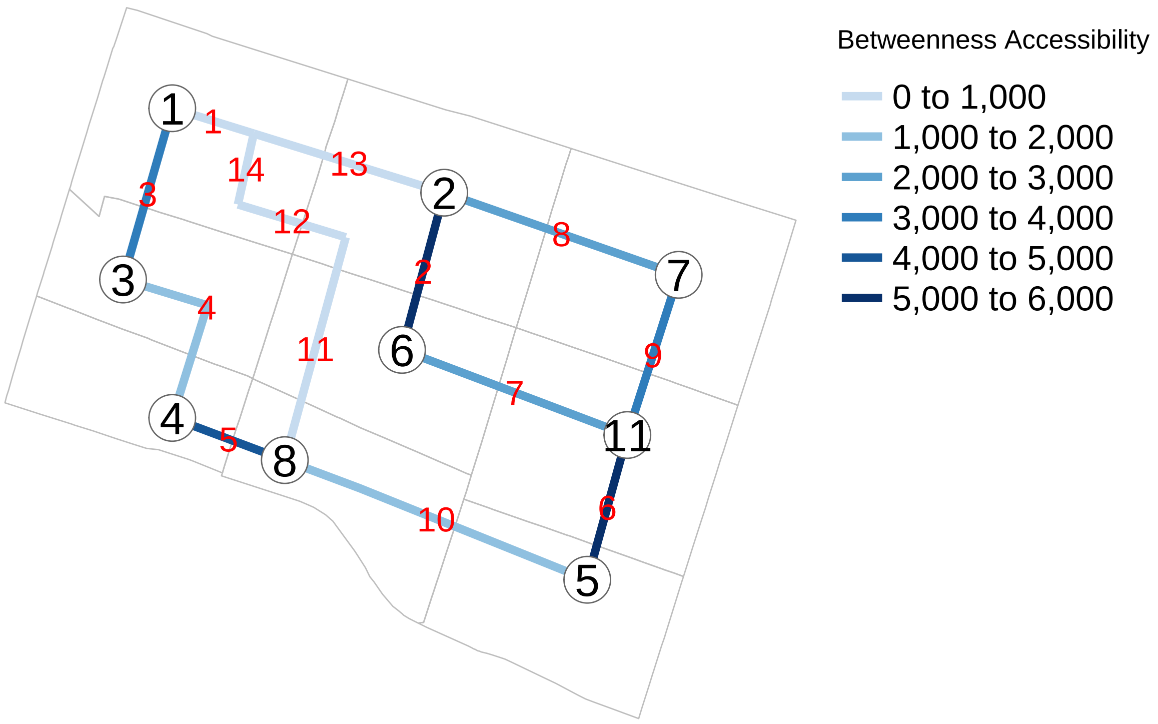
```
osm$btw_acc <- rowSums(btw_acc_mat)
```

Append the results to `osm` but round for ease of visualization later on:

```
osm@data <- cbind(osm@data, round(btw_acc_mat, digits = 2))
```

Plot betweenness-accessibility:

```
tm_shape(zones) +
  tm_borders(col="grey") +
  tm_shape(osm) +
  tm_lines(lwd = 6, title.col = "Betweenness Accessibility",
           col = "btw_acc", palette= c('#c6dbef','#6baed6','#2171b5','#08306b'), scale = 1) +
  tm_text("ID", col = "red",size=1.5) +
  tm_shape(centroid_nodes) +  tm_bubbles(size = 3, col = "white")  + tm_text("node_ID",size = 2)+
  tm_layout(frame = F, legend.text.size = 1.5, legend.title.size = 2,scale=0.7,legend.outside = T)
```



Betweenness accessibility is an attribute of the links and is the total number of people allocated to the link that are pushed from an origin to its accessible destinations.

Manual Validation

Now let's do this "by hand".

Plot the shortest path threes for every origin. First identify the shortest path tree from every origin:

```r
shortest_path_trees <- list() #initialize a list with the unique links of each shortest path three
spt <- data.frame() #initialize a dataframe to store the travel time values on the shortest path trees

for (i in 1:length(zones)){
  shortest_path_trees[[i]] <- na.omit(unique(unlist(list_links_sequence[[i]])))
  spt[unlist(shortest_path_trees[[i]]), i] <- 1 #osm$fftt[unlist(shortest_path_trees[[i]])]
  colnames(spt)[i] <- paste("SP_node_", centroid_nodes$node_ID[i], sep = "")
}
```

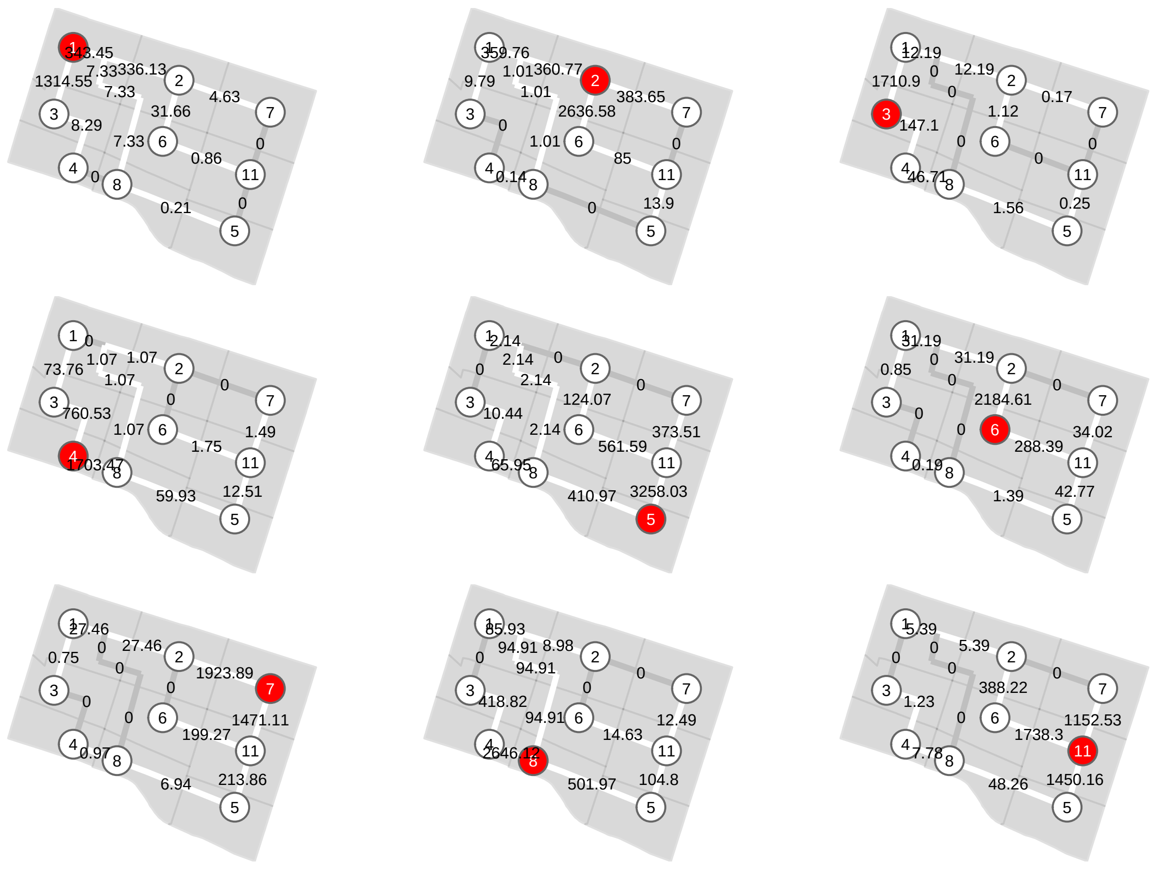
Append the shortest path trees to the links object `osm`:

```r
osm@data <- cbind(osm@data, spt)
```

Plot with faceting:

```r
for (i in centroid_nodes$node_ID) {
temp <- tm_shape(zones) + tm_polygons(col = "#d9d9d9",border.alpha = 0.2) +
  tm_shape(osm) +
  tm_lines(paste("SP_node_",i,sep=""), palette=c("white","#737373"), lwd = 3,scale=1) +
  tm_layout(frame = FALSE, inner.margins = 0, outer.margins = 0) +
  tm_shape(centroid_nodes[-which(centroid_nodes$node_ID==i),]) +
  tm_bubbles(size = 1, col = "white",scale=1) +
  tm_text("node_ID",col = "black", size = 0.5,scale=1)+
  tm_shape(centroid_nodes[which(centroid_nodes$node_ID==i),]) +
  tm_bubbles(size = 1, col = "red",scale=1) +
  tm_text("node_ID",col = "white", size = 0.5,scale=1)+
  tm_shape(osm) +
  tm_text(paste("btw_acc_node_",i,sep=""), col = "black", size = 0.5, overwrite.lines=TRUE,scale=1) +
  tm_legend(show = FALSE)
assign(paste("sp",i,sep=""),temp)
}
```

```r
suppressWarnings(tmap_arrange(sp1, sp2, sp3, sp4, sp5, sp6, sp7, sp8, sp11, ncol = 3))
```



Compare against the population “pushed” from each origin node to each destination node:

```
pop_btw_mat
```

```
##              1            2            3            4            5
## 1     0.000000   299.839145 1306.2539791    8.2918993    0.2056402
## 2   349.977187     0.000000    9.7868797    0.1406778   13.9046429
## 3  1698.707017    10.903946    0.0000000  100.3936835    1.3034142
## 4    73.764461     1.072179  686.7658473    0.0000000   47.4170893
## 5     2.141666   124.065830   10.4384342   55.5118227    0.0000000
## 6    30.338021  2153.424155    0.8483826    0.1934246   41.3840650
## 7    26.717309  1896.422281    0.7471318    0.9671285  206.9214900
## 8    85.929987     8.980654  418.8209401 2227.2989815  397.1692272
## 11    5.393426   382.830927    1.2321038    6.5523553 1401.9059041
##              6            7            8           11
## 1    30.797470    4.6306498     7.123118    0.8580993
## 2  2551.577538  383.6504171     0.868930   71.0937268
## 3     1.119980    0.1683981    45.148636    0.2549242
## 4     1.746758    1.4911679  1642.468575    9.2739223
## 5   437.526822  373.5068463   342.882041 2322.9265384
## 6     0.000000   34.0226141     1.194733  211.5946039
## 7   199.271984    0.0000000     5.973700 1057.9789752
## 8    14.630978   12.4901382     0.000000   77.6790940
## 11 1350.080025 1152.5330716    40.472188    0.0000000
```

Notice how the population on the links matches the values allocated to each origin destination pair. All is good!

IMPORTANT Please note that the sum of the accessibility-betweenness measure for the system, although it is measured in population, does not equal the population in the system. This is because there will be duplication. For instance, in the example above, the population that is pushed from node of origin 1 to destination node 7 appears in every link on the way to node 7.