reilly-m5

Nolan Reilly

11/23/2020

Matrix Representation of a Graph

- The below code will generate a random adjacency matrix. This function can scale efficiently to ~ 500 columns and rows on my machine:

```
createAdjMatrix = function(n){                # n = number of columns and rows
  d = runif(n*n)                              # generate random data from 0-1
  d[d < .8] = NA                              # 20% chance of adjacency
  d = matrix(d, nrow = n)                     # form data into a square matrix
  diag(d) = NA                                # make diagonal into NAs
  d[upper.tri(d)] = t(d)[upper.tri(d)]        # make upper & lower triangles adjacent
  return(d)                                      # i.e distance 1->3 is the same as 3->1
}
testmatrix = createAdjMatrix(5)
kable(testmatrix, digits = 2, caption = 'Adjacency Matrix')
```

Table 1: Adjacency Matrix

NA	NA	NA	0.90	NA
NA	NA	NA	0.92	NA
NA	NA	NA	NA	0.98
0.9	0.92	NA	NA	0.95
NA	NA	0.98	0.95	NA

- The next step in the workbook is to translate the above adjacency matrix into an adjacency list representation:

```
AdjMatrix2List <- function(d){                # d = adjacency matrix, use above function
  x = vector()                                # initialize empty vector to store results
  for (i in 1:nrow(d)){                       # loop through row
    for (j in 1:ncol(d)){                     # loop over each column value in row
      if(!is.na(d[i,j])){                     # skip over NAs/values without adjacency
        x = c(x, i, j, d[i,j])                # appending the row, column, & value
      }
    }
  }
  labs = c('head', 'tail', 'weight')          # column names for results matrix
  ds = matrix(x, ncol = 3, byrow = TRUE)      # shape result vector into a 3 column matrix
  colnames(x = ds) = labs                     # add column labels
  return(ds)
}
```

```
#testmatrix = createAdjMatrix(100)            # create matrix with n rows/columns
kable(AdjMatrix2List(testmatrix),
      digits = 2,
      caption = 'Adjacency List Representation')
```

Table 2: Adjacency List Representation

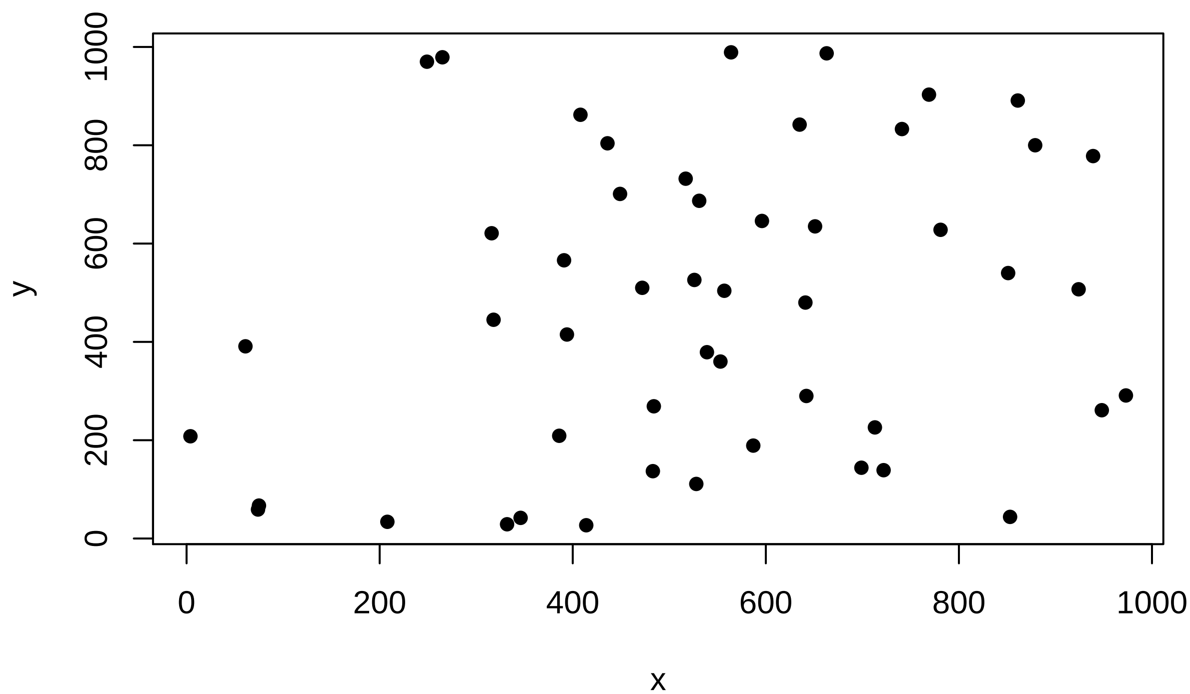
head	tail	weight
1	4	0.90
2	4	0.92
3	5	0.98
4	1	0.90
4	2	0.92
4	5	0.95
5	3	0.98
5	4	0.95

Euclidean Minimum Spanning Tree

In this exercise, I will create a Euclidean Minimum Spanning Tree (E-MST) on a set of random (X, Y) coordinates. One can think of each of these coordinates as locations for a facility where we want to build and visualize a road network that connects all of these facilities at minimum cost.

- I will first create a set of 50 random X,Y coordinates to represent 50 facilities that we want to efficiently connect (minimum cost).

```
n <- 50                                       # number of points to generate
x <- round(runif(n)*1000)
y <- round(runif(n)*1000)
plot(x,y,pch=16)
```



* The following code generates a complete graph of the distances between all 50 facilities (generated above). This is a complete graph because it calculates the distance between every single facility - resulting in an adjacency matrix that is full, except for 0's along the diagonal (when a facility is paired with itself).

```
hold = vector()                                    # initialize empty vector to store results

for (i in 1:n){                                    # set point 1
  for (j in 1:n){                                  # set point 2
    dist = sqrt(((y[j] - y[i])**2) +               # calculate euclidean distance
                  ((x[j] - x[i])**2))
    hold = c(hold, dist)                           # append distance to results vector
  }
}

matrixlabs = paste('(',x,',',y,')',sep ='')       # generate coordinate labels
holdresults = matrix(hold, ncol = n,               # shape results into square matrix
                     byrow = TRUE,
                     dimnames = list(matrixlabs,matrixlabs))

kable(head(holdresults[,1:5]),
      digits = 2, caption = 'Complete Adjacency Matrix')
```

Table 3: Complete Adjacency Matrix

	(564,989)	(394,415)	(642,290)	(517,732)	(851,540)
(564,989)	0.00	598.65	703.34	261.26	532.89
(394,415)	598.65	0.00	277.72	340.03	473.79

	(564,989)	(394,415)	(642,290)	(517,732)	(851,540)
(642,290)	703.34	277.72	0.00	459.34	325.85
(517,732)	261.26	340.03	459.34	0.00	385.25
(851,540)	532.89	473.79	325.85	385.25	0.00
(386,209)	800.05	206.16	268.51	539.16	570.78

- I will now use the AdjMatrix2List function (created above) to transform this complete adjacency matrix into an adjacency list:

```
ds = AdjMatrix2List(holdresults)
kable(head(ds), caption = 'Complete Adjacency List')
```

Table 4: Complete Adjacency List

head	tail	weight
1	1	0.0000
1	2	598.6451
1	3	703.3385
1	4	261.2623
1	5	532.8884
1	6	800.0525

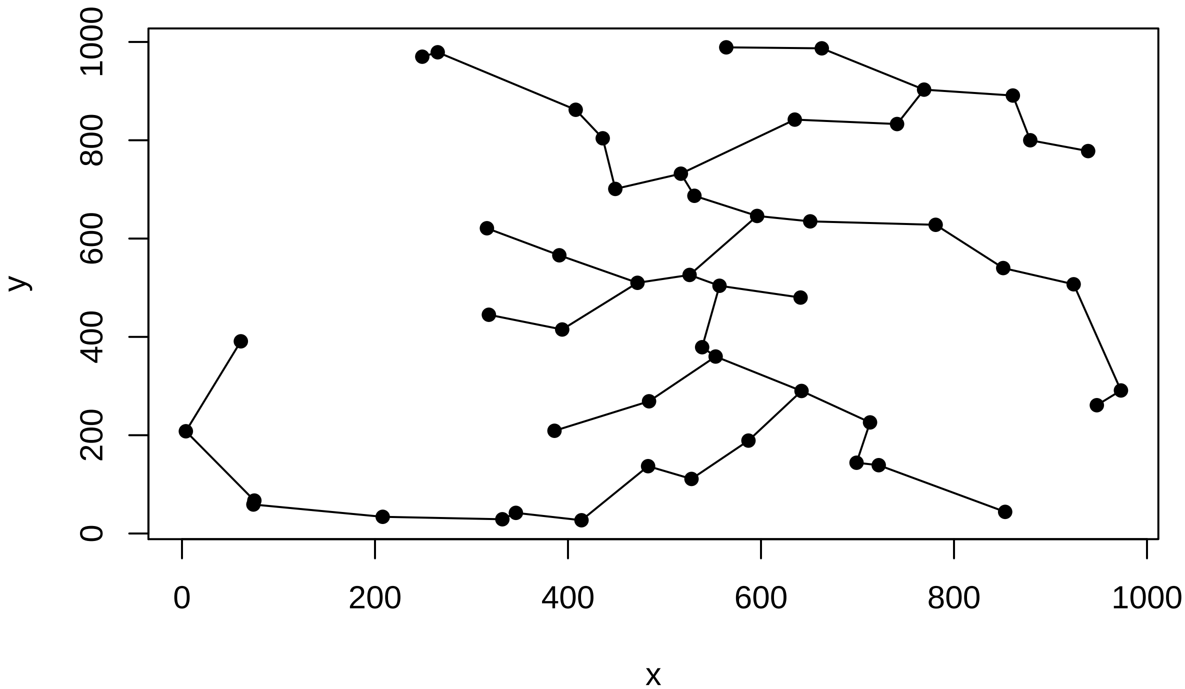
- This format now allows me to calculate the minimum spanning tree (the most efficient path between the facilities). This will be calculated using the Dijkstra-Prim algorithm that is found in the 'optrees' package.

```
ds.mst = msTreePrim(nodes = 1:n,                    # calculate minimum spanning tree
                    arcs = ds)                      # arcs = adjacency list created above
route = ds.mst$tree.arcs                            # pull the route (from-to) from results
head(route)                                         # ept1 = starting node | ept2 = ending node
```

```
##      ept1 ept2    weight
## [1,]    1   32  99.02020
## [2,]   32   50 135.24792
## [3,]   50   25  75.39231
## [4,]   50   44  92.77931
## [5,]   44   21  92.76314
## [6,]   21   16  63.90618
```

- This information can be used to graphically represent the most efficient route through the facilities:

```
plot.mst = function(route){
  segments(x[route[,1]], y[route[,1]],x[route[,2]],y[route[,2]])
}
plot(x,y,pch=16)
plot.mst(route)
```


0
200
400
600
800
1000
y
0
200
400
600
800
1000
x