Discrete Optimization

Cheryl Ngo

September 29, 2019

2 Matrix Representation

We can use Cmat2ArcList to convert the adjacency matrix d to an adjacency list which works when n is small.

```
n=1000
d=runif(n*n)
d[d<.8]=NA
d=matrix(d,nrow=n,ncol=n) #reshape the vector
diag(d)=NA
d[upper.tri(d)]=t(d)[upper.tri(d)] #undirected graphs are symmetric

#ds2=Cmat2ArcList(nodes=seq(1,n,1),Cmat=d,directed=FALSE)
```

The following function, AdjMatrix2List produces, from matrix d, the sparse matrix ds.

```
#Create adjacency list function and run for d
AdjMatrix2List=function(d){
  ds=matrix(,nrow=0,ncol=3)
  colnames(ds)=c("head","tail","weight")
  for (i in 1:nrow(d)){
    for (j in i:nrow(d)){
      if (is.na(d[i,j])==FALSE){
        ds=rbind(ds,c(i,j,d[i,j]))
      }
    }
  }
  return(ds)
}
```

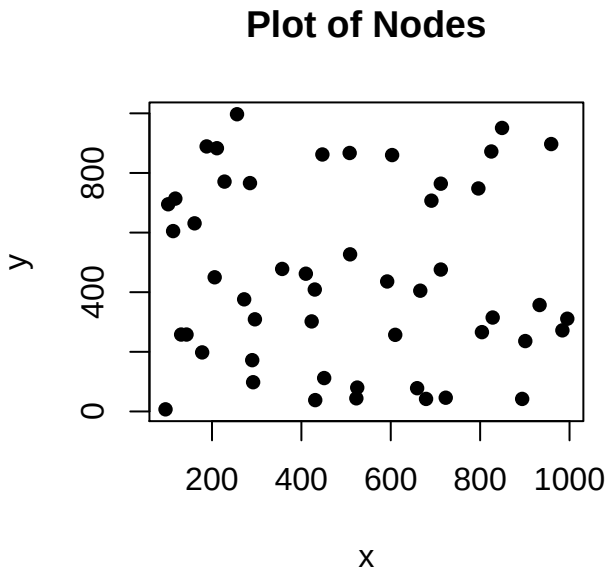
```
head(ds)
```

```
##      head tail    weight
## [1,]    1   12 0.9971975
## [2,]    1   16 0.8001409
## [3,]    1   28 0.9206426
## [4,]    1   38 0.8435542
## [5,]    1   41 0.8971563
## [6,]    1   42 0.9636896
```

3 Euclidean Minimum Spanning Tree

Visualize a set of 50 random locations:

```
n=50
x=round(runif(n)*1000)
y=round(runif(n)*1000)
plot(x,y,pch=16,main="Plot of Nodes")
```



3.1 Adjacency matrix (d)

The following function, AdjMatrix3, creates an adjacency matrix with distances as the edge weights.

```
#Create adjacency matrix with distances
AdjMatrix3=function(x,y){
  d=matrix(,nrow=n,ncol=n)
  for (i in 1:length(x)){
    for (j in i:length(x)){
      dist=sqrt((y[j]-y[i])^2+(x[j]-x[i])^2)
      d[i,j]=dist
    }
  }
  return(d)
}
```

3.2 Adjacency List (*ds*)

Using the same function from problem 2, we create an adjacency list *ds*.

```
ds=AdjMatrix2List(d) #ds=Cmat2ArcList(nodes=seq(1,n,1),Cmat=d,directed=FALSE)
head(ds)
```

```
##      head tail   weight
## [1,]    1    1   0.0000
## [2,]    1    2 374.3007
## [3,]    1    3 604.6288
## [4,]    1    4 692.2904
## [5,]    1    5  40.5216
## [6,]    1    6 600.4706
```

3.3 Minimum Spanning Tree

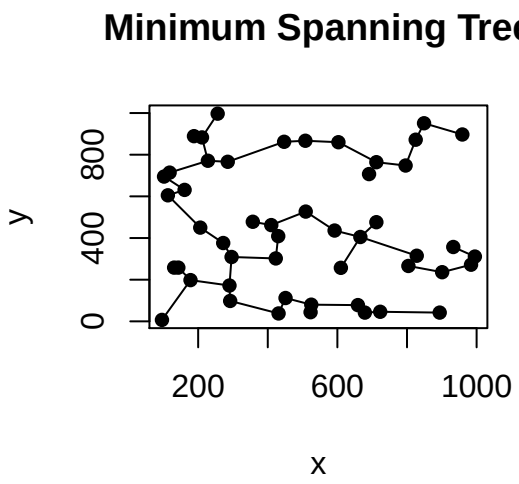
We can calculate the minimum spanning tree of matrix d using Kruskal's algorithm with msTreeKruskal.

```
ds.mst=msTreeKruskal(1:n,ds)
#head(ds.mst)
```

3.4 Plot Minimum Spanning Tree

Plot the minimum spanning tree of matrix d:

```
plot.mst=function(arcList,x,y){
  i=ds.mst$tree.arcs[,1]
  j=ds.mst$tree.arcs[,2]
  segments(x[i],y[i],x[j],y[j])
}
plot(x,y,pch=16,main="Minimum Spanning Tree")
plot.mst(ds.mst$tree.arcs,x,y)
```



4 Hostile Agents

1. This problem can be represented by at least one undirected graph where the nodes are agents and the edges are weighted with the probability that the message is intercepted. The solution to this problem is to find the minimum spanning tree for each connected graph resulting in a minimum spanning forest if there are disconnected components.
2. The inputs could be structed as an adjacency matrix where each row/column specifies a pair of agents connected by an edge and the maxtrix value represents the probability that a message between them will be intercepted.
3. I would choose Kruskal's algorithm to solve the problem because our graph is likely to be sparse because we would have a minimal number of agents with a minimal number of connections to other agents.
4. The computational efficiency of Kruskal's algorithm is O(N log N).

5 Project Scheduling

5.1 1693 Analytics

Create the node and edge representations:

```
s.labels=c('a','b','c','d','e','f','g','h','i','j')
s.nodes=c(90,15,5,20,21,25,14,28,30,45)
```

5.4 Earliest Start Times (ES)

```
nodes=c(1,2,3,4,5,6,7,8,9,10)
duration=c(1,2,-90,1,6,-90,1,9,-90,2,3,-15,6,7,-25,
     9,10,-30,3,7,-5,7,4,-14,4,8,-20,4,5,-20,
     4,10,-20)
duration=matrix(unlist(duration),ncol=3,byrow=TRUE)
```

```
est.negative=getShortestPathTree(nodes,duration,algorithm="Bellman-Ford",show.graph=FALSE)
```

```
##
##  Shortest path tree
##  Algorithm: Bellman-Ford
##  Stages:  9 | Time:  0
##  -----------------------------
##       head     tail    weight
##          1        2       -90
##          1        6       -90
##          1        9       -90
##          2        3       -15
##          4        5       -20
##          4        8       -20
##          4       10       -20
##          6        7       -25
##          7        4       -14
##  -----------------------------
```

```
##                    Total = -384
##
##  Distances from source:
##  ------------------------------
##     source     node  distance
##          1        2       -90
##          1        3      -105
##          1        4      -129
##          1        5      -149
##          1        6       -90
##          1        7      -115
##          1        8      -149
##          1        9       -90
##          1       10      -149
##  ------------------------------
```

```
# est.negative=getShortestPathTree(nodes,duration,algorithm="Bellman-Ford",
# show.graph=TRUE,show.distances=TRUE)
est=est.negative$distances*-1
```

The earliest start times for the project are a : 0, b : 90, c : 105, d : 129, e : 149, f : 90, g : 115, h : 149, i : 90, j : 149

5.6 Earliest Finish Times

The earliest finish times for the project are a : 90, b : 105, c : 110, d : 149, e : 170, f : 115, g : 129, h : 177, i : 120, j : 194.

5.5 Earliest Overall Project Completion Time

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following object is masked from 'package:igraph':
##
##     %--%

## The following object is masked from 'package:base':
##
##     date
```

```
maxef=max(unlist(solution[2,]))
ect=as.Date("2019-11-1")+maxef
```

The earliest overall completion time for the project is 2020-05-13.

5.7 Latest Finish Times

```
nodes2=c(1,2,3,4,5,6,7,8,9,10,11)
duration2=c(2,1,-19,6,1,-25,9,1,-30,3,2,-5,7,6,-14,
     10,9,-45,7,3,-14,4,7,-20,8,4,-28,5,4,-21,
     10,4,-45,11,8,0,11,5,0,11,10,0)
duration2=matrix(unlist(duration2),ncol=3,byrow=TRUE)
lft.negative=getShortestPathTree(nodes2,duration2,algorithm="Bellman-Ford",
                                 source.node = 11,show.graph=FALSE)
```

```
##
##  Shortest path tree
##  Algorithm: Bellman-Ford
##  Stages:  10 | Time:  0.01
##  ------------------------------
##       head     tail    weight
##          3        2        -5
##          4        7       -20
##          6        1       -25
##          7        3       -14
##          7        6       -14
##         10        4       -45
##         10        9       -45
##         11        5         0
##         11        8         0
##         11       10         0
##  ------------------------------
##                    Total = -168
##
##  Distances from source:
##  ------------------------------
##     source     node  distance
##         11        1      -104
##         11        2       -84
##         11        3       -79
##         11        4       -45
##         11        5         0
##         11        6       -79
##         11        7       -65
##         11        8         0
##         11        9       -45
##         11       10         0
##  ------------------------------
```

```
# lft.negative=getShortestPathTree(nodes2,duration2,algorithm="Bellman-Ford",
# source.node=11,show.graph=TRUE,show.distances=TRUE)
# lft=est.negative$distances*-1
solution=rbind(solution,lft.negative$distances[1:10]+maxef)
rownames(solution)[3]="Latest Finish Times"
```

The latest finish times for the project are a : 90, b : 110, c : 115, d : 149, e : 194, f : 115, g : 129, h : 194, i : 149, j : 194.

5.8 Latest Start Times

```
lst=c()
for (i in 1:length(s.nodes)){
  lst[i]=solution[[3,i]]-s.nodes[i]
}
solution=rbind(solution,lst)
rownames(solution)[4]="Latest Start Times"
```

The latest start times for the project are a : 0, b : 95, c : 110, d : 129, e : 173, f : 90, g : 115, h : 166, i : 119, j : 149.

5.9 Slack

```
slack=c()
for (i in 1:length(s.nodes)){
  slack[i]=solution[[3,i]]-solution[[2,i]]
}
solution=rbind(solution,slack)
rownames(solution)[5]="Slack"
```

1. The tasks with scheduling flexibility are b, c, e, h, i
2. The tasks on the critical path are a, d, f, g, j

5.10 Gantt Chart (Extra)

```
# s.desc=c("Prototype","Purchase","Manufacture","Revision",
#          "Initial","Training","Input","Sales","Pre-prod",
#          "Post-prod")
# barplot(rbind(unlist(solution[3,])-s.nodes,s.nodes),horiz=TRUE,col=c("white","gray"),
# border=0,names.arg = s.desc,las=1)
```

```
# library(timevis) # Install this first
# gnt = data.frame(id = 1:10, content=s.desc, start = unlist(solution[3,])-s.nodes,
# end = solution[3,])
# timevis(gnt)
```