BCB 570 HW2

Urminder Singh

January 21, 2018

Sol 4:

```
library("igraph")
fivestargraph <- make_full_graph(n = 5)
plot.igraph(fivestargraph)
```

Sol 5:

```
# read GraphA
graphA <- read_graph("graphA.txt", format = "edgelist", directed = "false")
vcA <- vcount(graphA)
ecA <- ecount(graphA)
densityA <- graph.density(graphA)
diaA <- diameter(graphA, directed = "false")
radA <- radius(graphA)
# read GraphB
graphB <- read_graph("graphB.txt", format = "edgelist", directed = "false")
vcB <- vcount(graphB)
ecB <- ecount(graphB)
densityB <- graph.density(graphB)
diaB <- diameter(graphB, directed = "false")
radB <- radius(graphB)
# read GraphC
graphC <- read_graph("graphC.txt", format = "edgelist", directed = "false")
vcC <- vcount(graphC)
ecC <- ecount(graphC)
densityC <- graph.density(graphC)
diaC <- diameter(graphC, directed = "false")
radC <- radius(graphC)
header <- c("#Vertices", "#Edges", "Density")
tab1 <- data.frame(Feature = header, GraphA = c(vcA, ecA, densityA), GraphB = c(vcB,
    ecB, densityB), GraphC = c(vcC, ecC, densityC))
tab1 %>% knitr::kable(caption = "Features")
```

Table 1: Features

Feature	GraphA	GraphB	GraphC
#Vertices	100.0000000	100.0000000	200.0000000
#Edges	140.0000000	850.0000000	222.0000000
Density	0.0282828	0.1717172	0.0111558

5i) From the table we can see that GraphA has 100 vertices and 140 edges thus the number of edges are of

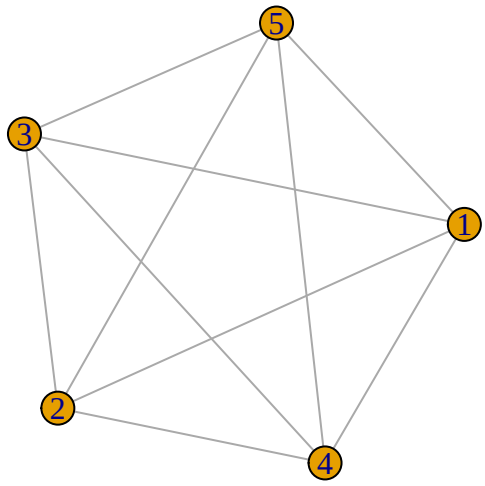

Figure 1: Fully connected 5-point star

order of number of vertices this means graphA is sparse. For similar reason GraphC is also sparse. GraphB has 100 vertices and 850 edges. The number of edges here are closer to the order of $vertices^2$. Thus we can say GraphB is somewhere in-between sparse and dense. Also looking at the edge density we see that egde density of graphB is closer to 1 where as for other graphs are less that

```
header2 <- c("Radius", "Diameter")
tab2 <- data.frame(Feature = header2, GraphA = c(radA, diaA), GraphB = c(radB,
    diaB), GraphC = c(radC, diaC))
tab2 %>% knitr::kable(caption = "Radius and Diameter")
```

Table 2: Radius and Diameter

Feature	GraphA	GraphB	GraphC
Radius	6	2	1
Diameter	10	3	12

5ii) Radius and Diameter of the three graphs are in table 2.

```
ccA <- transitivity(graphA, type = "average")
ccA_g <- transitivity(graphA, type = "global")
ccB <- transitivity(graphB, type = "average")
ccB_g <- transitivity(graphB, type = "global")
ccC <- transitivity(graphC, type = "average")
ccC_g <- transitivity(graphC, type = "global")
header3 <- c(" clustering coefficient(avg)", " clustering coefficient(global)")
tab3 <- data.frame(Feature = header3, GraphA = c(ccA, ccA_g), GraphB = c(ccB,
    ccB_g), GraphC = c(ccC, ccC_g))
tab3 %>% knitr::kable(caption = "Clustering Coeff.")
```

Table 3: Clustering Coeff.

Feature	GraphA	GraphB	GraphC
clustering coefficient(avg)	0.0060000	0.3923619	0.0141667
clustering coefficient(global)	0.0105263	0.2403273	0.0080214

5iii) The global clustering coefficient measure gives an indication of the clustering in the whole network (global). The global clustering coefficient is based on triplets of nodes where a triplet consists of three connected nodes. Thus, global clustering coefficient is the number of closed triplets over the total number of triplets, both closed and open.
The average clustering coefficient is measured by averaging the local clustering coefficients for each vertex in the graph. Where the local clustering coefficient for a vertex is calculated as the ratio of edges between the vertices within its neighbourhood divided by the number of total possible edges between them. The avg. clustering coeff places more weight on the low degree nodes whereas global clustering coefficient places more weight on high degree nodes. Clustering coefficients for the given graphs are shown in table 3.

```
# compute avg shortest dist
mdA <- mean_distance(graphA)
mdB <- mean_distance(graphB)
mdC <- mean_distance(graphC)
header4 <- c("Average shortest path")
tab4 <- data.frame(Feature = header4, GraphA = c(mdA), GraphB = c(mdB), GraphC = c(mdC))
tab4 %>% knitr::kable(caption = "Avg shortest path length")
```

Table 4: Avg shortest path length

Feature	GraphA	GraphB	GraphC
Average shortest path	5.061616	1.860606	4.890743

5iv) Avg shortest path length in a graph is another way to measure the network size. It is defined as average of all the shortest distances between each pair of nodes. The Avg shortest path length for given graphs is shown in table 4.

```
# get central nodes
topDA <- head(order(degree(graphA), decreasing = TRUE), 5)
topDB <- head(order(degree(graphB), decreasing = TRUE), 5)
topDC <- head(order(degree(graphC), decreasing = TRUE), 5)
header5 <- c("Vertices with highest degree", "Vertices with highest betweenness",
    "Common vertices")
topBA <- (head(order(betweenness(graphA), decreasing = TRUE), 5))
topBB <- (head(order(betweenness(graphB), decreasing = TRUE), 5))
topBC <- (head(order(betweenness(graphC), decreasing = TRUE), 5))
commA <- intersect(topDA, topBA)
commB <- intersect(topDB, topBB)
commC <- intersect(topDC, topBC)
tab5 <- data.frame(Feature = header5, GraphA = c(toString(topDA), toString(topBA),
    toString(commA)), GraphB = c(toString(topDB), toString(topBB), toString(commB)),
    GraphC = c(toString(topDC), toString(topBC), toString(commC)))
tab5 %>% knitr::kable(caption = "Centrality")
```

Table 5: Centrality

Feature	GraphA	GraphB	GraphC
Vertices with highest degree	72, 59, 64, 75, 79	10, 44, 48, 73, 39	33, 34, 62, 165, 72
Vertices with highest betweenness	72, 97, 55, 64, 10	10, 48, 44, 73, 81	83, 33, 62, 165, 72
Common vertices	72, 64	10, 44, 48, 73	33, 62, 165, 72

5v) The centrality measure in graph identifies the most important or influential vertices. Two common methods to define centrality are degree centrality and betweenness centrality. Degree centrality is propotional to the number of edges incident upon a vertex. Thus the degree is propotional to the probability that a vertex will catch whatever is flowing through the network. Betweeness centrality tries to quantify the number of times a vertex acts as a bridge along the shortest path between two other nodes. These measures for the given graphs are shown in Table 5. In graphA there are two, in top 5, vertices 64 and 72 which were reported to be central by both methods. In graphB there were four common central vertices i.e. 10,44,48 and 73. Finally in graphC vertices 33,62,165,72 were found to be central by both methods. These nodes are found to be central as they have higher degrees which in turn implies that any shortest between two nodes may involve these high degree nodes thus making the betweenness measure high. Thus we expect that top high degree nodes will also be present as bridges in shortest paths between other pairs.

Sol 6:

To check whether these graphs follow the power law distribution we have to check the degree distribution of the graphs and then try to fit them to a power law distribution. We can do this in following steps:

i) First we should get the degree distribution of the graph and plot it on log scale i.e. logk vs log(p(k)) and compare it to a standard power law distribution, as shown in Fig 2d. If our network follows is power law distribution we should see a straight line with negative slope. In figure 2 we see that our graphA and C's distribution is closer to powerlaw. However this method is just for visual inspection. Thus, We use the function *power.law.fit* from igraph package to fit the degree distribution to power law and get statistical p-values to evaluate the fit.

ii) If the degree distribution follows power law we will have

$$P(k) \sim k^{-\alpha}$$

. *power.law.fit* function reports the parameter of the fitted power law distribution and also reports a p-value. P-value below a confidence level means network is not from a power law distribution, whereas higher p-values indicate in favor of the null i.e. network's degree distribution follows a power law. Table 6 shows the results obtained from applying *power.law.fit* function to the graphs. The xmin parameter was evaluated by power.law.fit function such that the p-value for Kolmogorov-Smirnov test is largest. From the table we can see that the p-values for all three graphs are greater than 0.05 therefore we fail to reject the null hypothesis at 95% significance. Thus, the given networks follow power law distribution.

Sol 7:

A scale free network is charecterized by a highly heterogeneous degree distribution which follows power law. We can check for these properties to assess whether a given network is scale-free or not:

i) First check if network's degree distribution follows power law or not.We can try to fit the power law distribution to the degree distribution using function *power.law.fit* from igraph package and then reject or accept the null hypothesis, that degree distribution is sampled from a scle free distribution, at a confidence level using the p-value reported.

ii) Next we can split the network randomly into smaller sub-networks and check degree distribution of these smaller networs. Ideally, in a scale free network the degree distribution should be the same in the whole network and in the smaller sub-networks.

iii) Since Scale free networks' degree distribution follows power law, we expect smaller number of hubs. Thus, we can make a test statistic based on number of hubs in a network and see if it is significantly less than that under a random network.

```
# plot a graph with powerlaw
g_powlaw <- barabasi.game(1e+05)
par(mfrow = c(2, 2))
plot(degree.distribution(graphA), xlab = "log(k)", ylab = "log(P(k))", log = "xy",
    type = "o", main = "a: Degree Distribution GraphA")
plot(degree.distribution(graphB), xlab = "log(k)", ylab = "log(P(k))", log = "xy",
    type = "o", main = "b: Degree Distribution GraphB")
plot(degree.distribution(graphC), xlab = "log(k)", ylab = "log(P(k))", log = "xy",
    type = "o", main = "c: Degree Distribution GraphC")
plot(degree.distribution(g_powlaw), xlab = "log(k)", ylab = "log(P(k))", log = "xy",
    type = "o", main = "d: Powerlaw distribution")
```

```
# fit powerlaw
fitA <- power.law.fit(degree(graphA, mode = "all"), implementation = "plfit")
fitB <- power.law.fit(degree(graphB, mode = "all"), implementation = "plfit")
fitC <- power.law.fit(degree(graphC, mode = "all"), implementation = "plfit")
head6 <- c("GraphA", "GraphB", "GraphC")
tab6 <- data.frame(Graph = head6, Alpha = c(fitA$alpha, fitB$alpha, fitC$alpha),
    X_min = c(fitA$xmin, fitB$xmin, fitC$xmin), KS_stat = c(fitA$KS.stat, fitB$KS.stat,
```

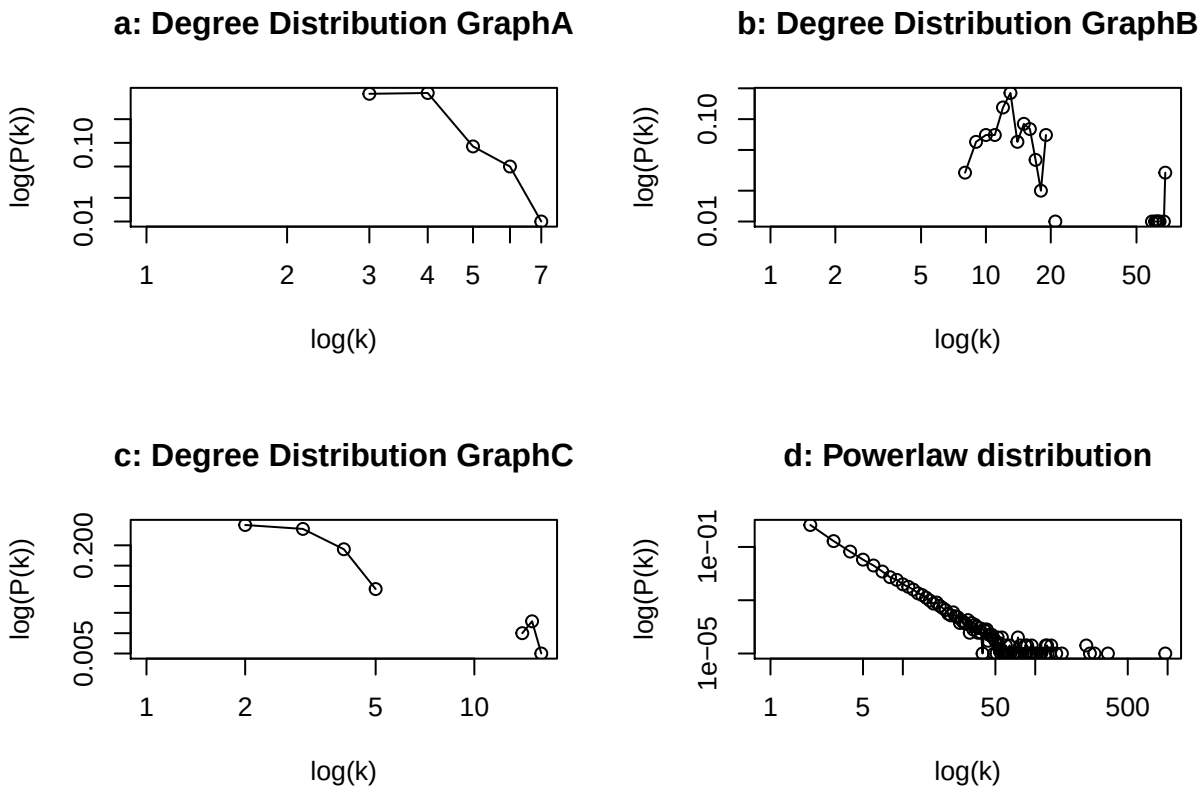

Figure 2: Degree distributions of graphs

```
        fitC$KS.stat), P_Value = c(fitA$KS.p, fitB$KS.p, fitC$KS.p), logLik = c(fitA$logLik,
        fitB$logLik, fitC$logLik))
tab6 %>% knitr::kable(caption = "Power law fits")
```

Table 6: Power law fits

Graph	Alpha	X_min	KS_stat	P_Value	logLik
GraphA	5.384347	3	0.0200026	1.0000000	-47.65304
GraphB	3.346749	11	0.1045514	0.3691128	-225.50559
GraphC	3.095753	2	0.0586762	0.8031255	-153.58968