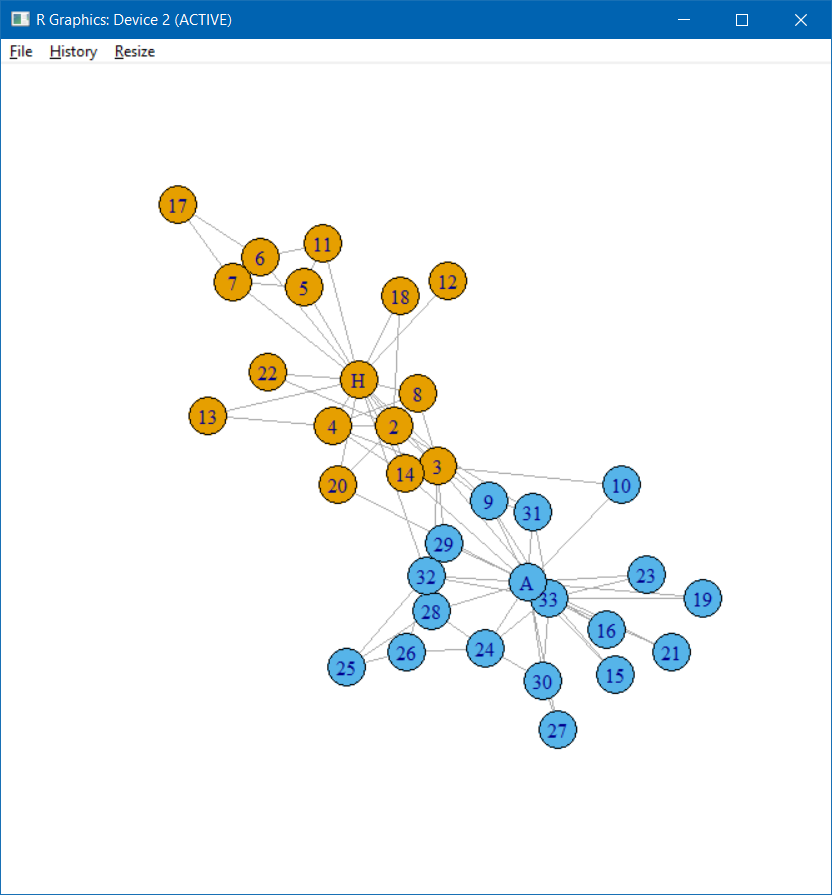
**Homework 4**

Problem 1

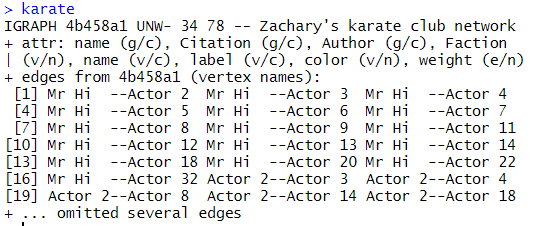
Key:

* 1,2,3 – Karate 5%, 15%, 40%
* 4,5,6 – Kite 5%, 15%, 40%
* 7 – My observations on both
* Sorry, there’s wayyy too many images in here, but I don’t want to lose points for not documenting something. I have explained my work as concisely, here as well in my code annotations, as I could in 1. 2-6 is essentially the same work done for different values of percentages or a different network.

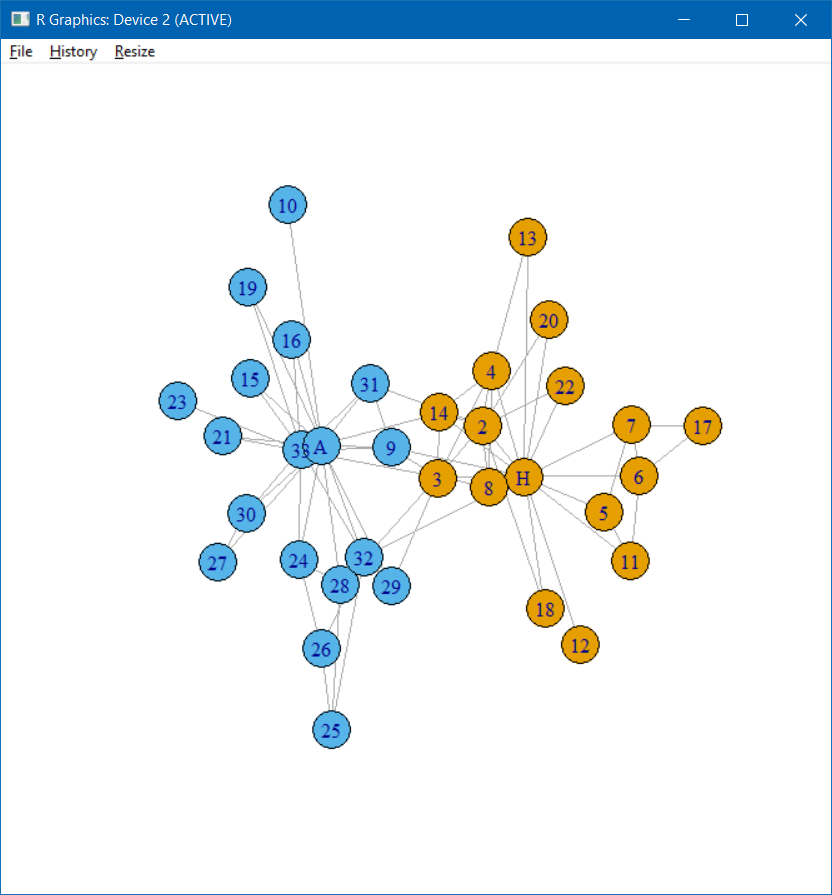
1. First step was to install the igraph and igraphdata packages, which was pretty straightforward. The next step was to access the karate and kite datasets. The following steps detail my process for the noisy dataset creating, MCMC and edge prediction.
   1. For the first step, I scrutinized the data frame and plotted the karate network.



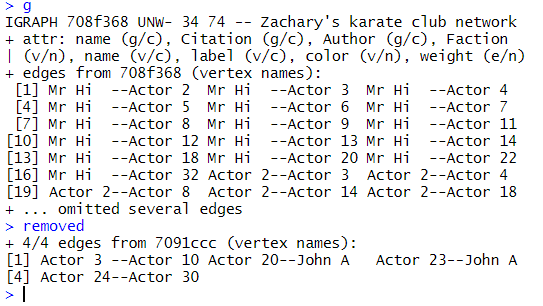
The above image shows the original network. The details below show that there are a total of 34 vertices and 78 edges, and that the dataset is that of Zachary’s karate network



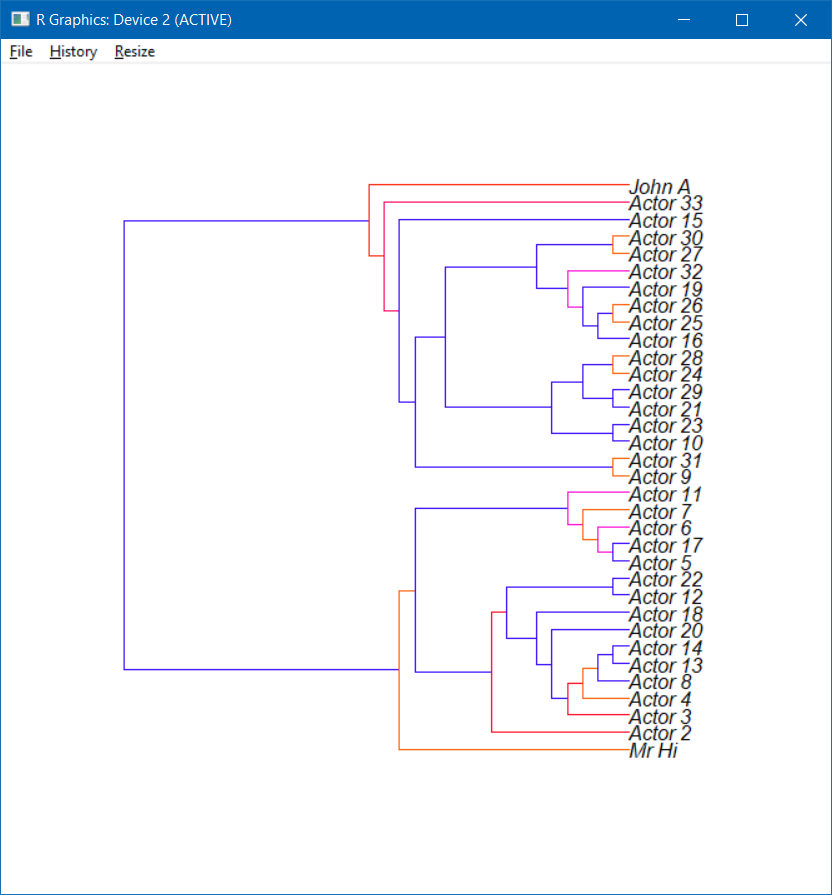
* 1. The question requires me to perform the same activities when 5%, 15% and 40% of the edges are removed. Therefore the next step was to simply introduce a variable called perc that would control the sampling.
  2. Next, to know how many edges are 5% of the total, we first need the number of edges, which I stored in variable nedge using ecount(), also storing the edges of the original karate network in orig.
  3. Next step was to calculate how many edges were to be deleted (perc\*nedge). I used the round function on this, since perc\*edge might well not give an integer as a result.
  4. Next step was to use the sample function to sample the required no of edges from the network, and then removing these sampled edges. This I did by simply subtracting and using the edge() function to convert the samples into edge form. To record the deleted edges, I did original karate minus the new network I created. This I stored in edge form in the variable removed, essentially tracking which of the edges I deleted, as instructed in the question.
  5. The new network looks as shown below



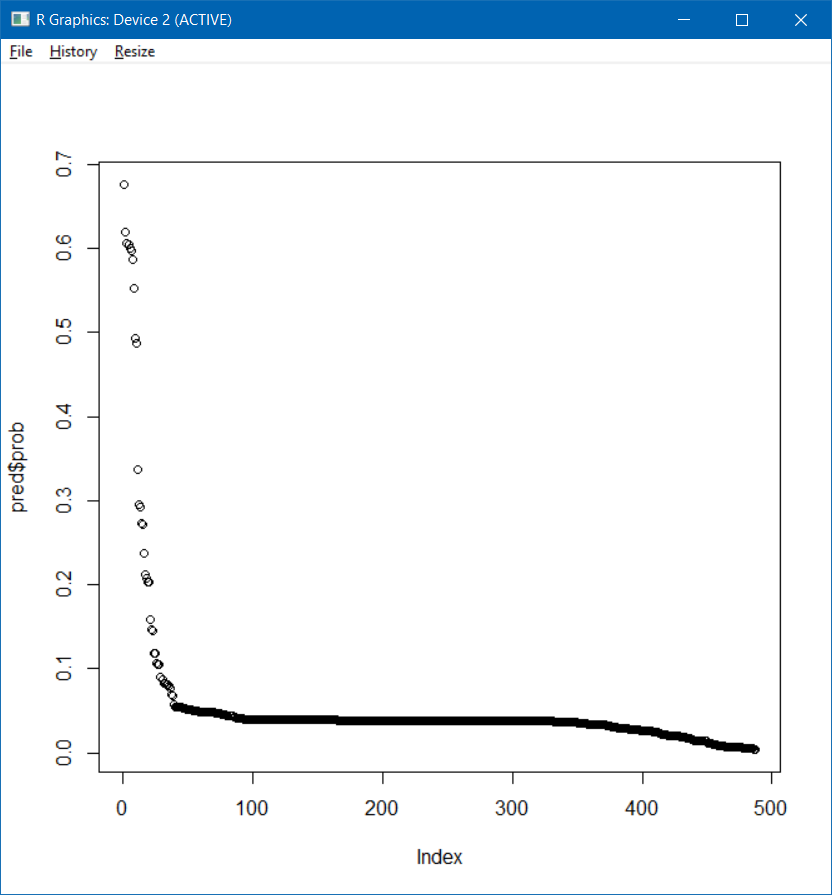
It may not seem apparent from the images due to the high number of edges just which ones were deleted. Details of those are shown below, as well as the removed variable, detailing which edges were removed.



* 1. Next, I fit a hierarchical random graph to the network and plotted a dendrogram corresponding to the same.

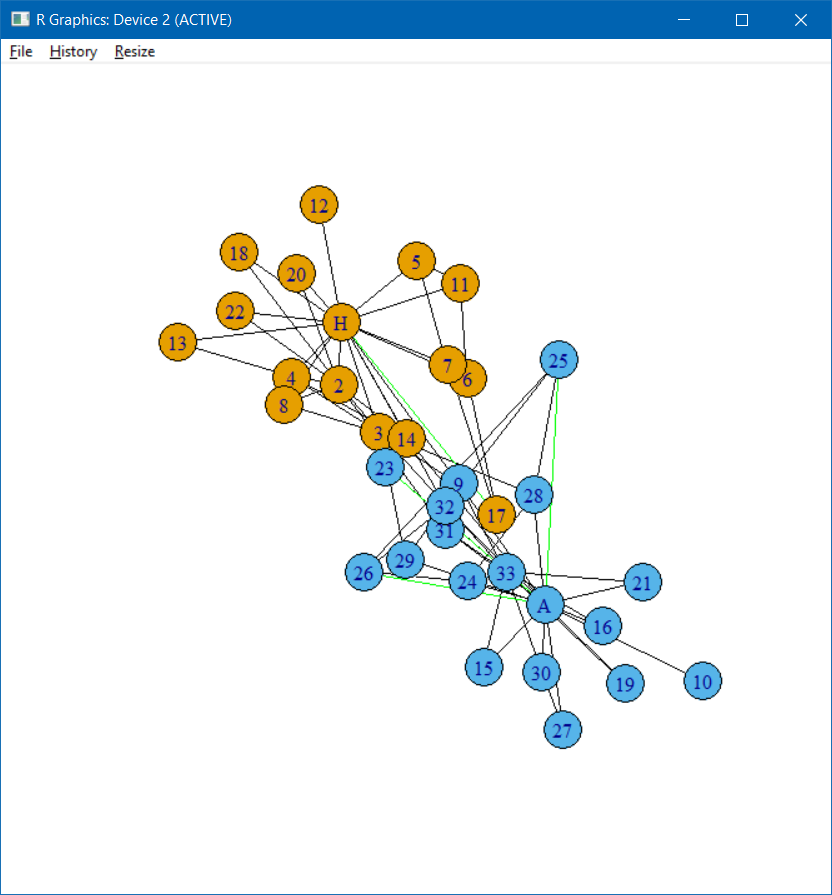


* 1. The next step was the most important, the prediction of the missing edges. The predicted edges are as shown below, with a plot of their probabilities.



From this graph it is clear that apart from the first few predicted edges, the other predicted edges have pretty low probability. The one with the highest leads the pack by a significant gap, followed by a handful of edges with probabilities significantly high.

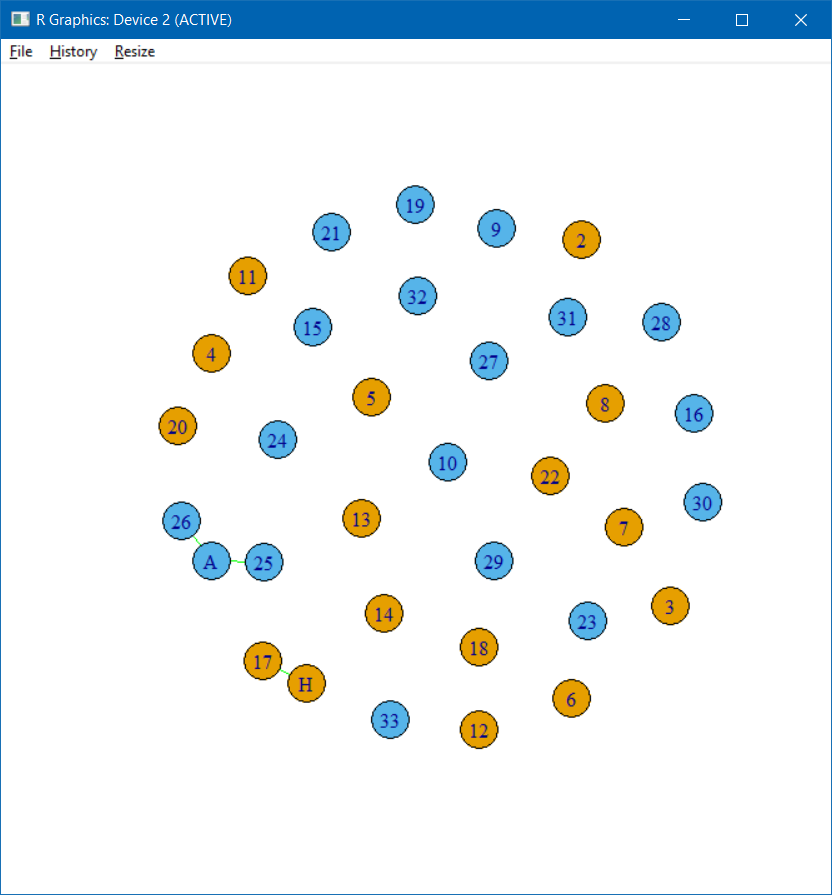
* 1. The next step is to add the same number of edges to the new network as were deleted from the original, and colouring them differently so as to make them conspicuous. I also stored the edges that were added to a variable called added.



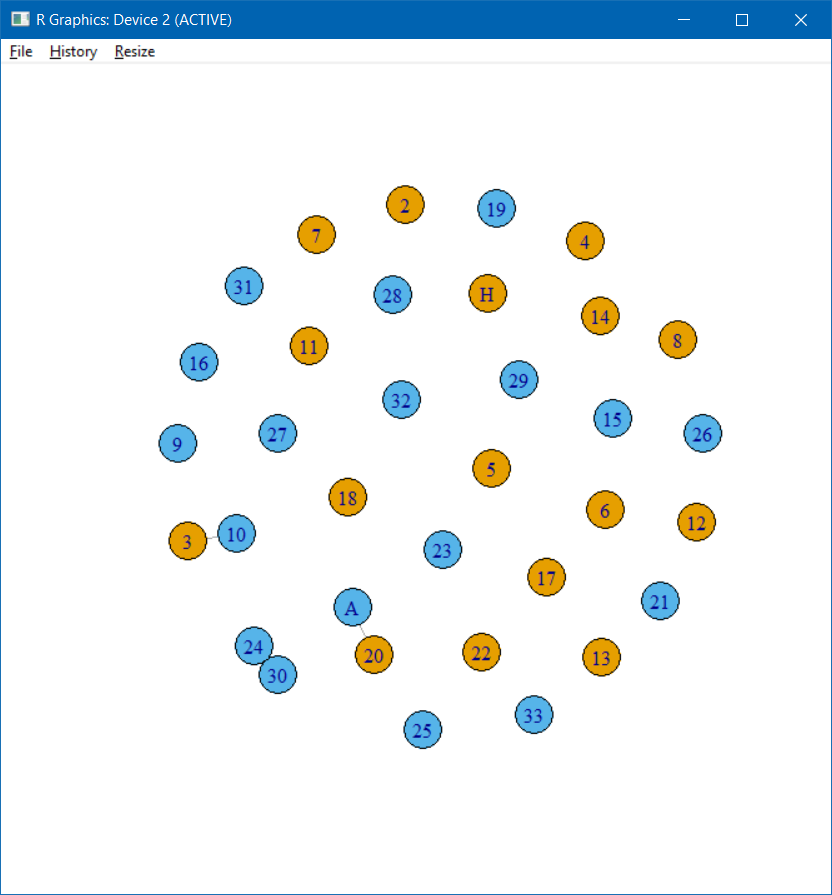
As can be seen in the image, 4 green edges have been added.

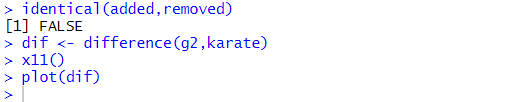
* 1. The last step was to check if the new network with the added edges is different from the original. I could check if the variables added and removed were the same, but I used difference() to plot which of the edges added to the network from the predicted list was not present in the list of deleted ones. The result for both the approaches is as shown below.

Difference between new network and the original



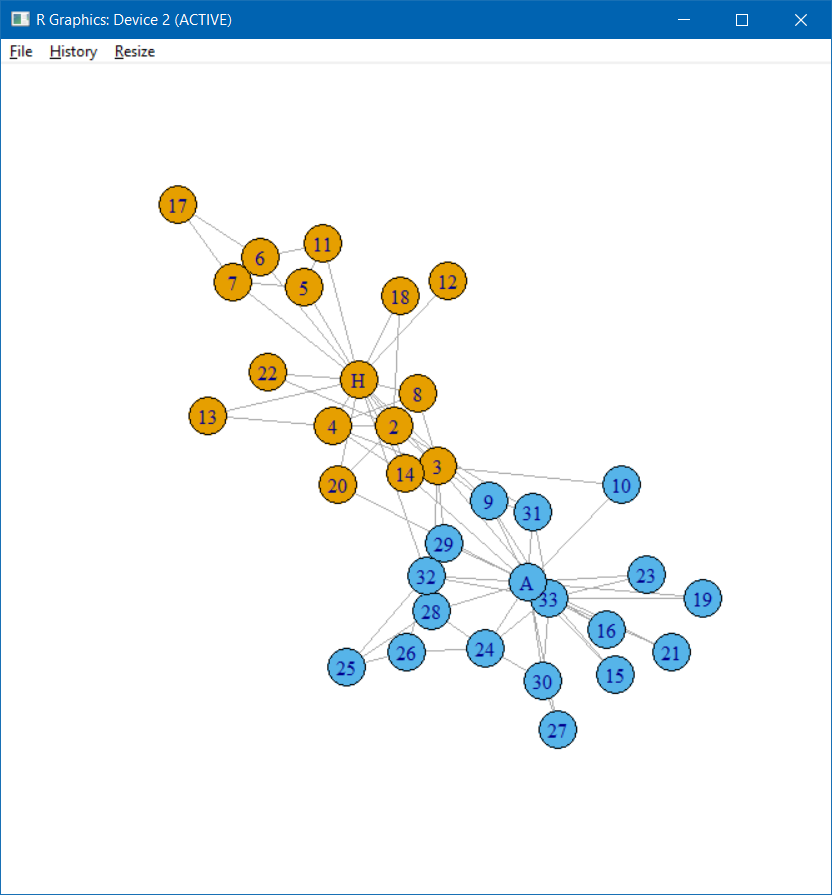
Difference between original network and the new



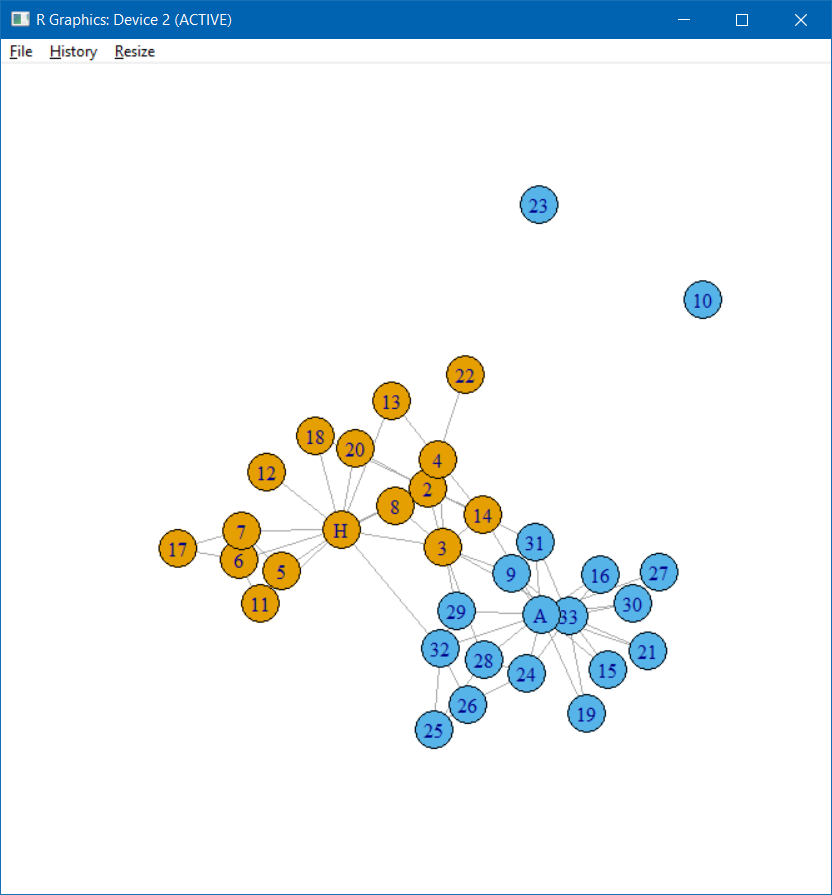


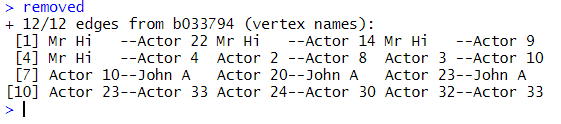
As is evident, the best predicted edges aren’t exactly the ones that were deleted from the original. From the difference between the new and the original, we can see that 34(A) – 25 and 34(A) – 26 are the ones that were added which were not originally deleted. From the difference between the original and new, we can see that 3 – 10 and A – 20 are not part of the top predicted ones to be added to the new network.

* 1. Since this entire process is essentially the same for 15% edges deleted and 40% edges deleted, I am simply displaying the various outputs for the process below. I simply changed the value of perc from 0.05 to 0.15 for the 15% process.
  2. The original network

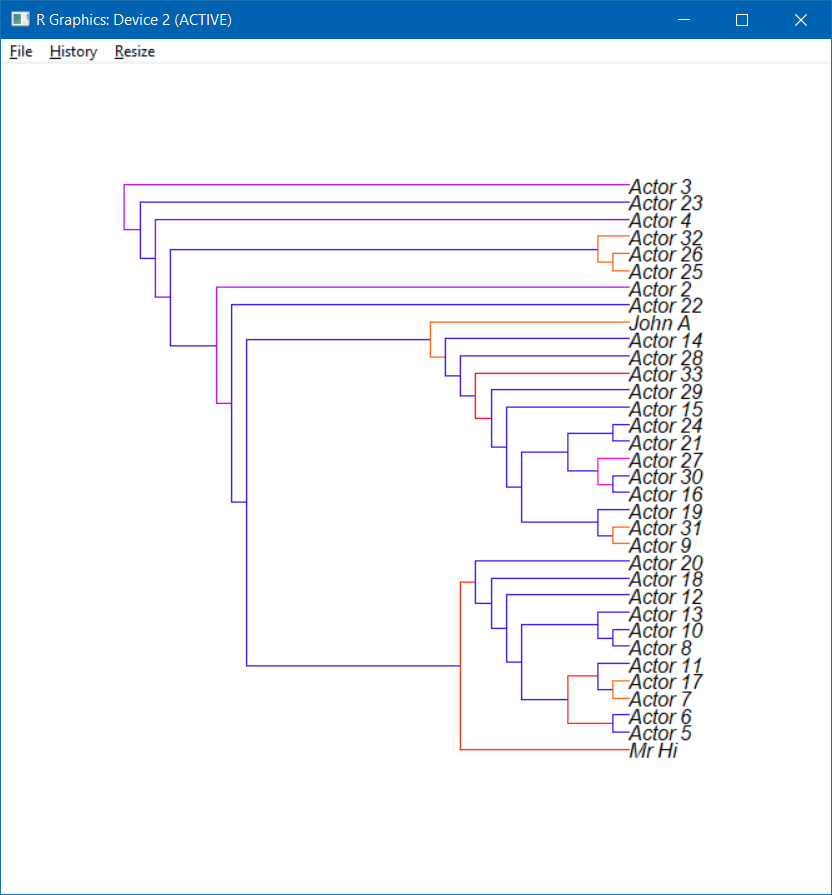


* 1. The new network after 15% edge deletion

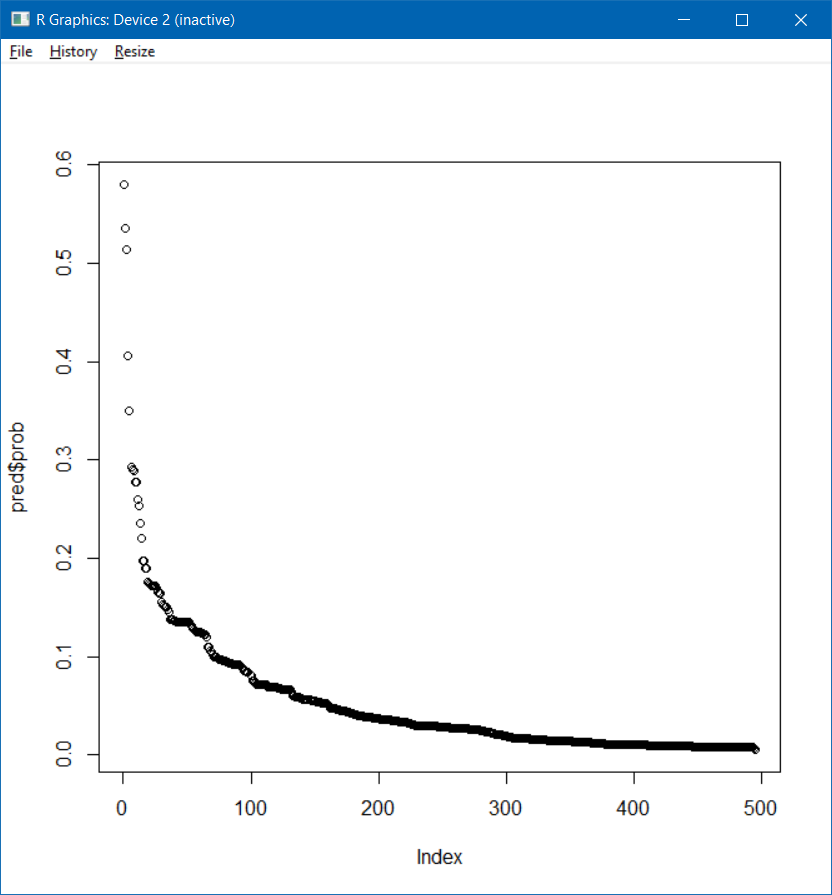




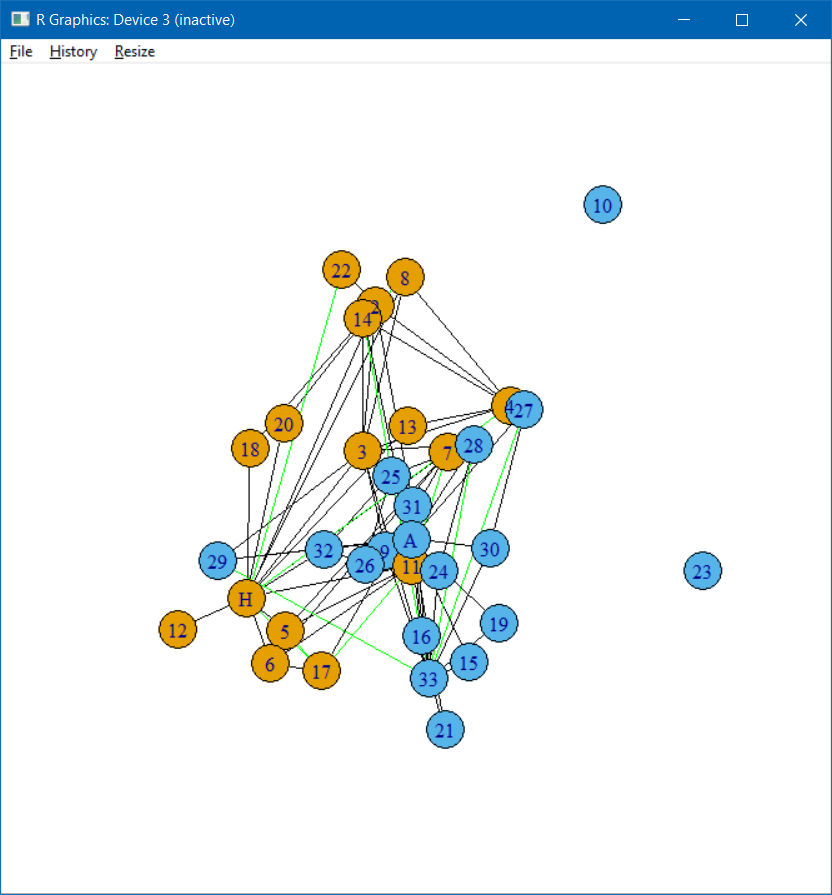
* 1. Dendrogram of hierarchical random graph



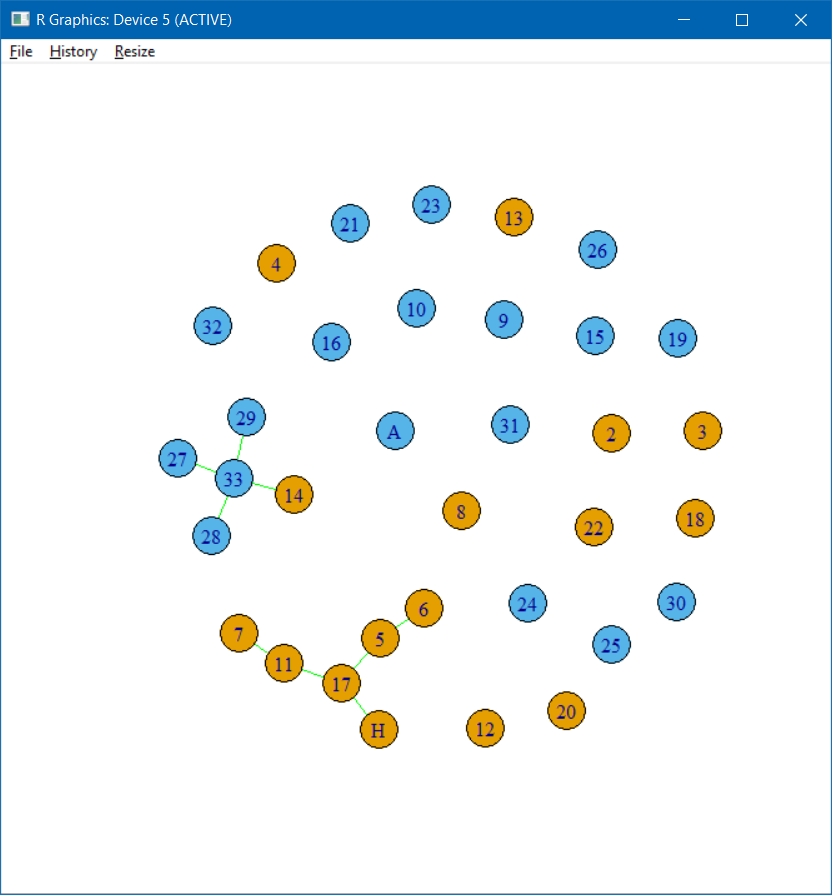
* 1. Predicted edges probabilities



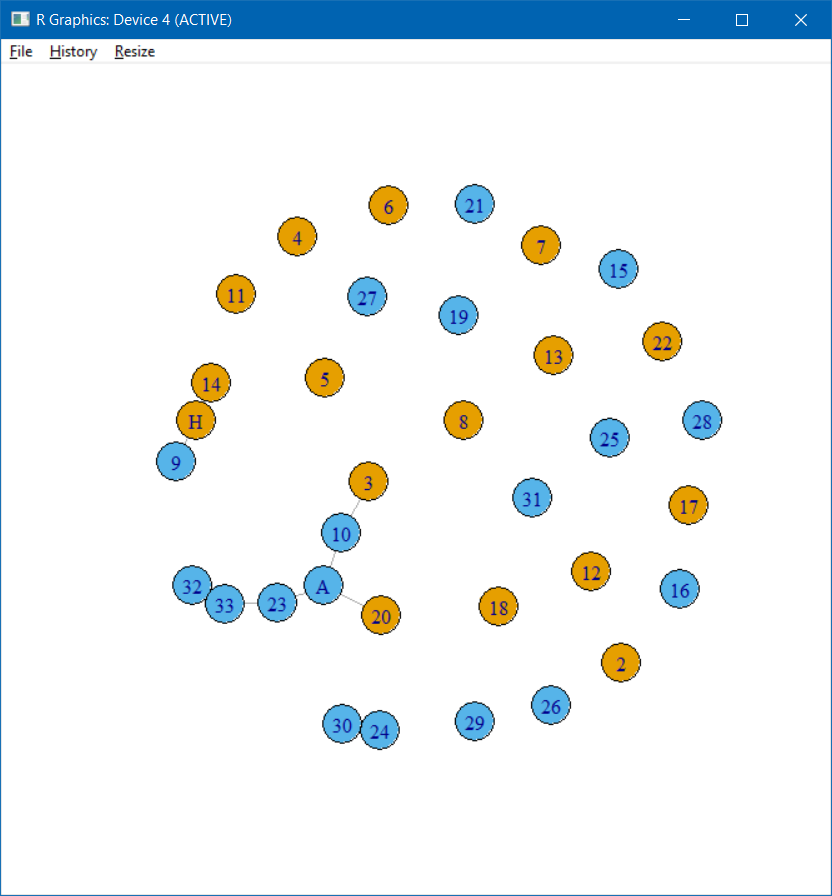
* 1. Network with added edges



* 1. Difference plots

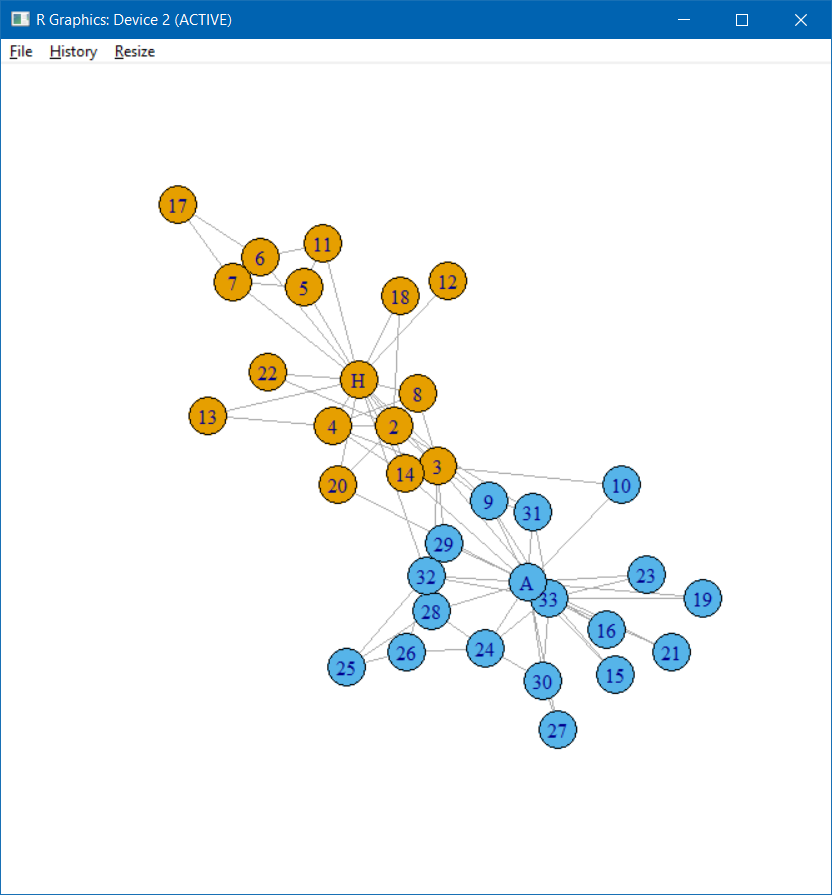


Edges that were added to new network which were not previously deleted from original

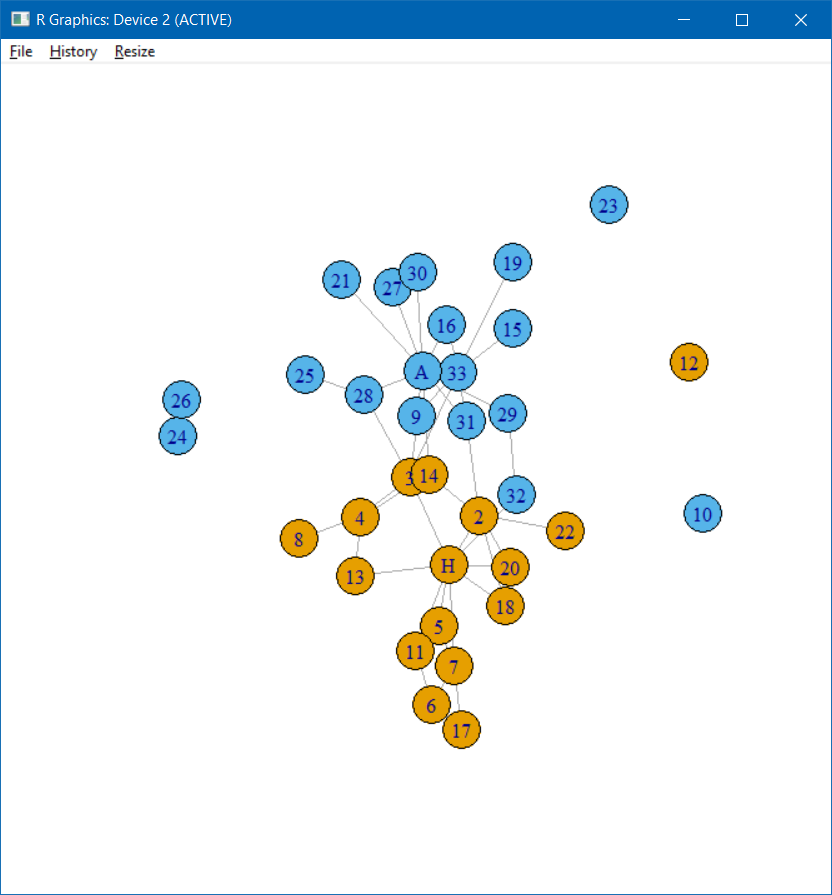


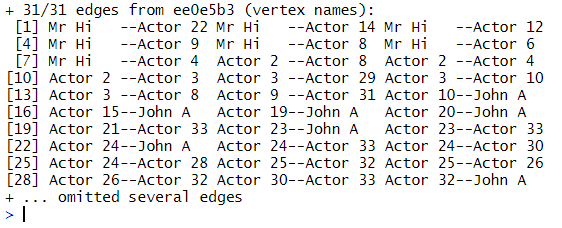
Edges that were deleted from original but not added to the new network due to lower probability.

* 1. I simply changed the value of perc from 0.15 to 0.40 for the 40% process.
  2. The original network

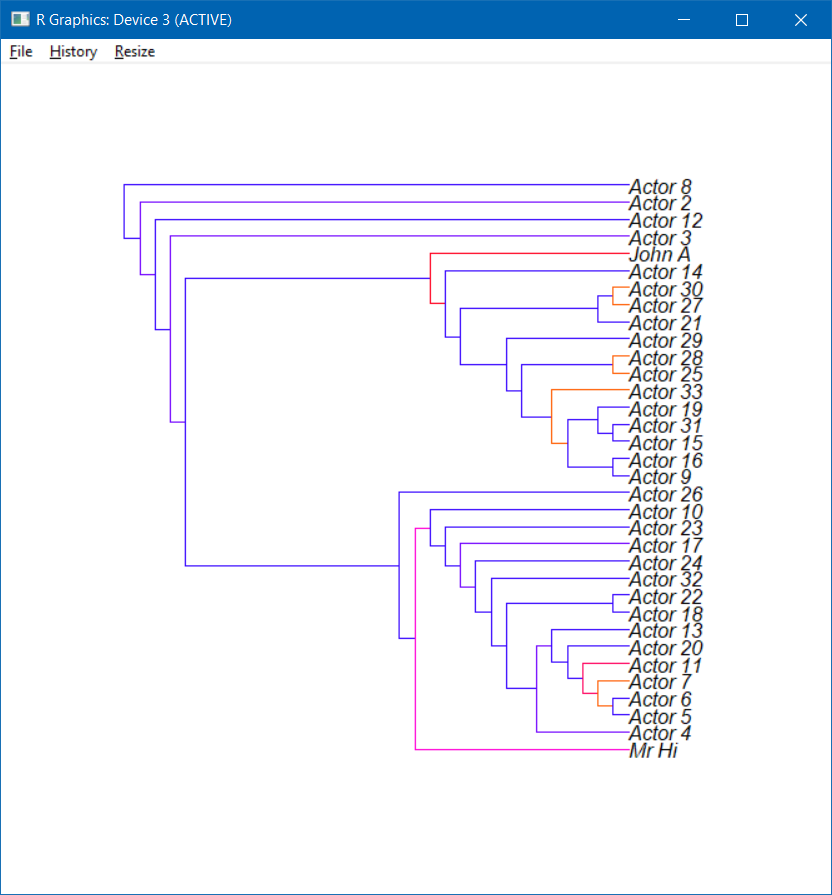


* 1. The new network after 40% edge deletion

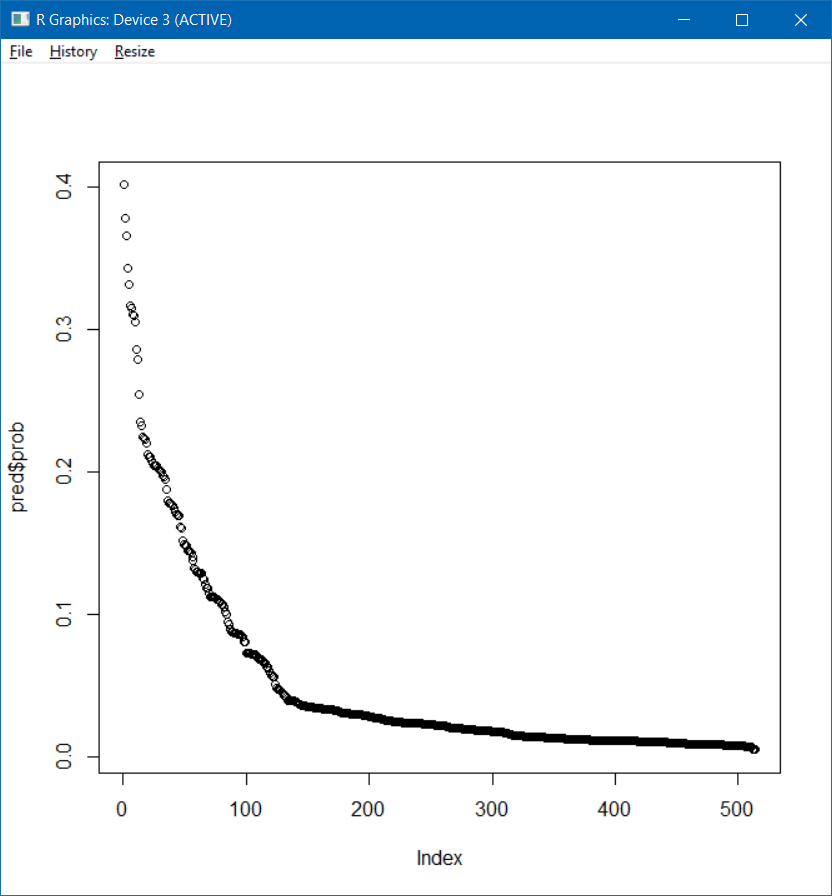




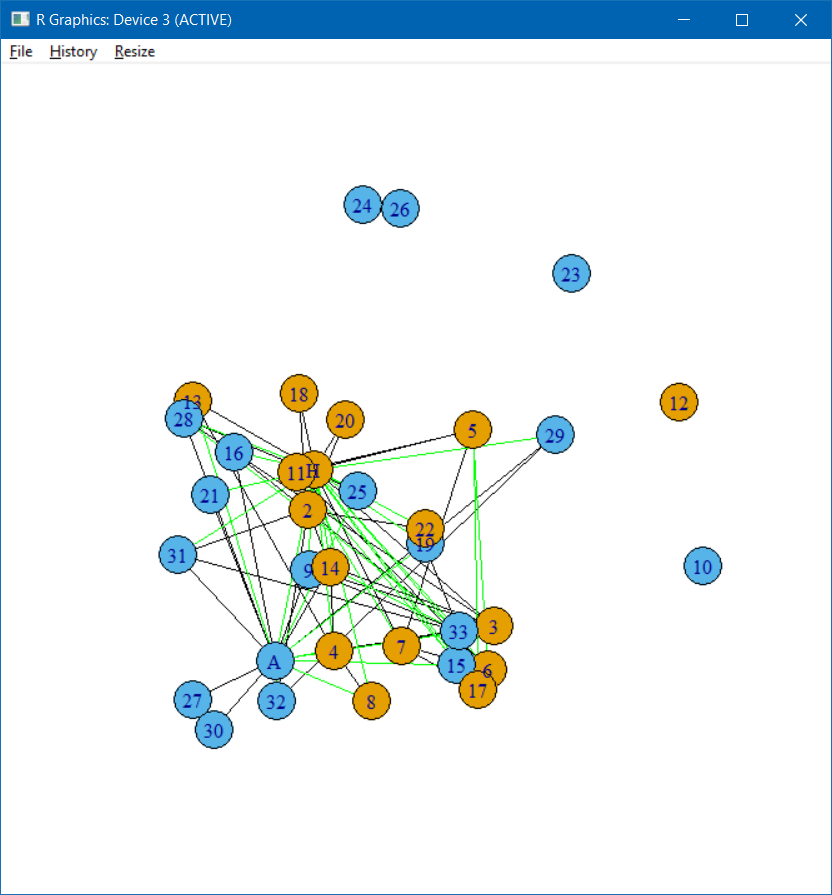
* 1. Dendrogram of hierarchical random graph



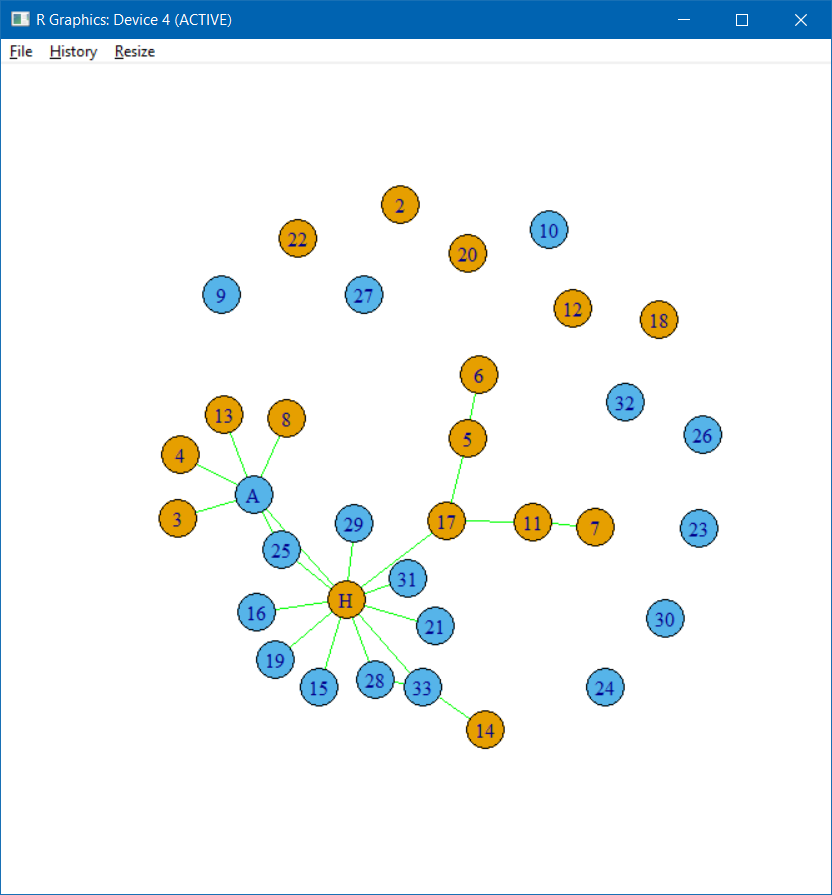
* 1. Predicted edges probabilities



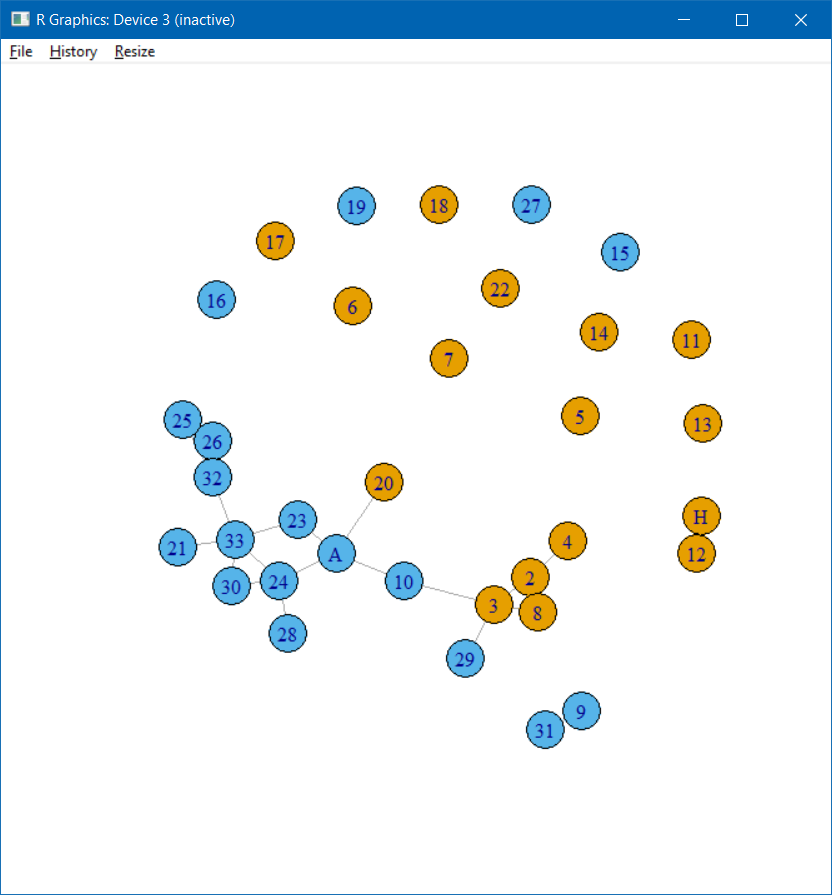
* 1. Network with added edges



* 1. Difference plots

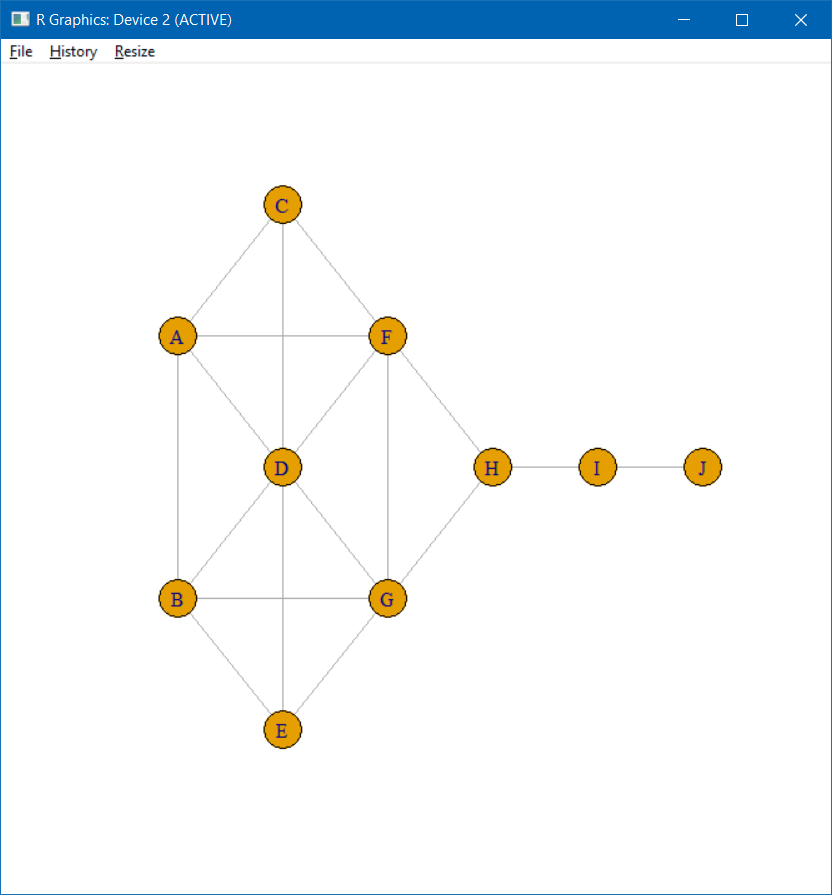


Edges that were added to new network which were not previously deleted from original

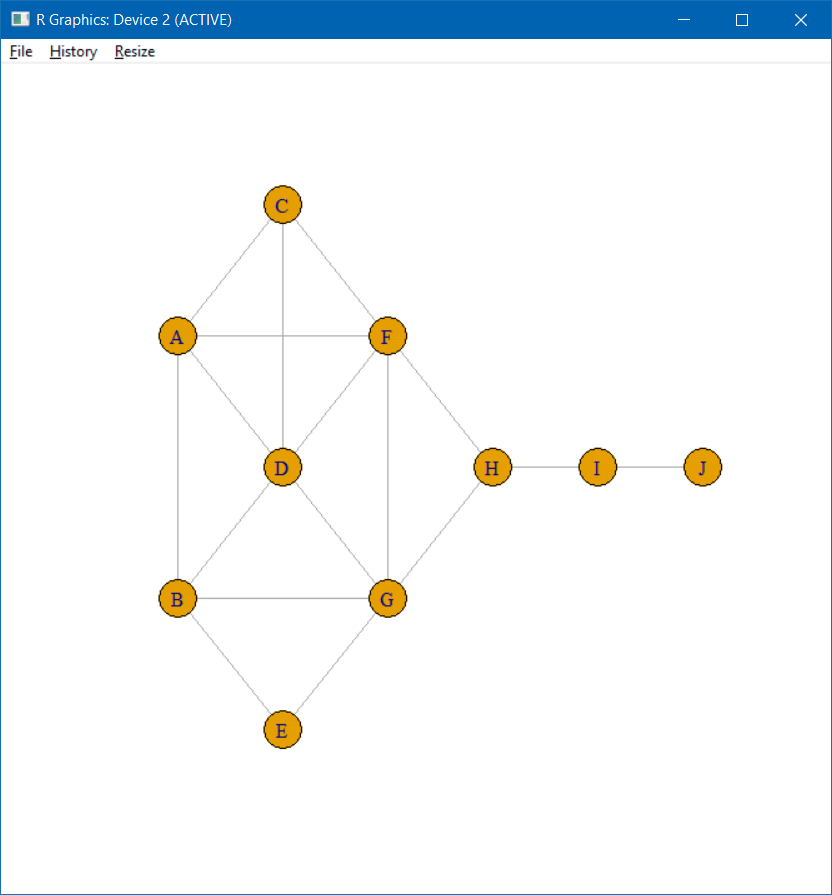


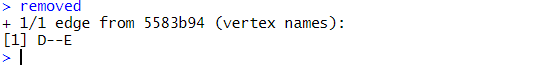
Edges that were deleted from original but not added to the new network due to lower probability.

1. When dealing with the kite network, which is a considerably smaller network than the karate network, I followed entirely the same steps, since the process is the same, just on a different network
   1. The original network

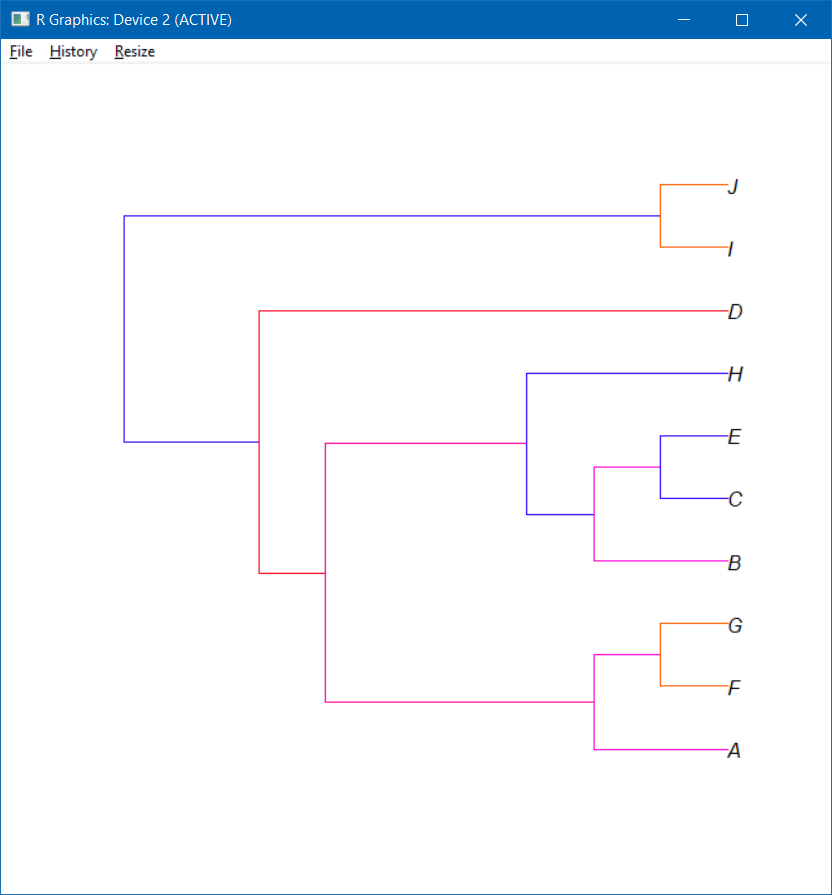


* 1. Network after 5% edge deletion

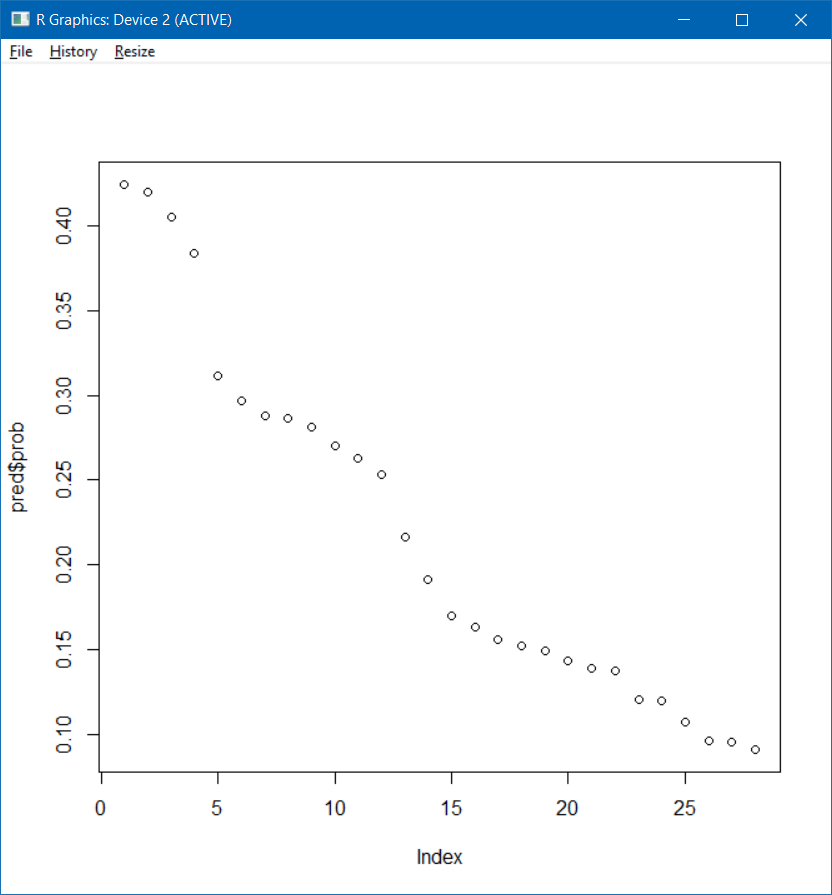




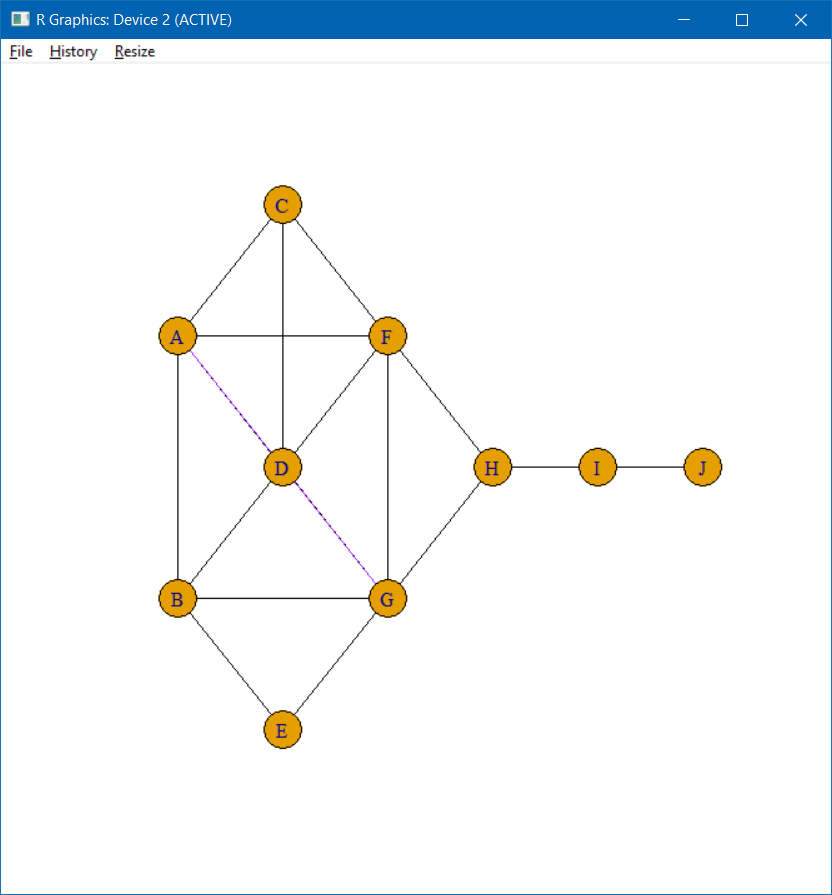
* 1. Dendrogram after fitting hierarchical random graph



* 1. Probability graph of predicted edges

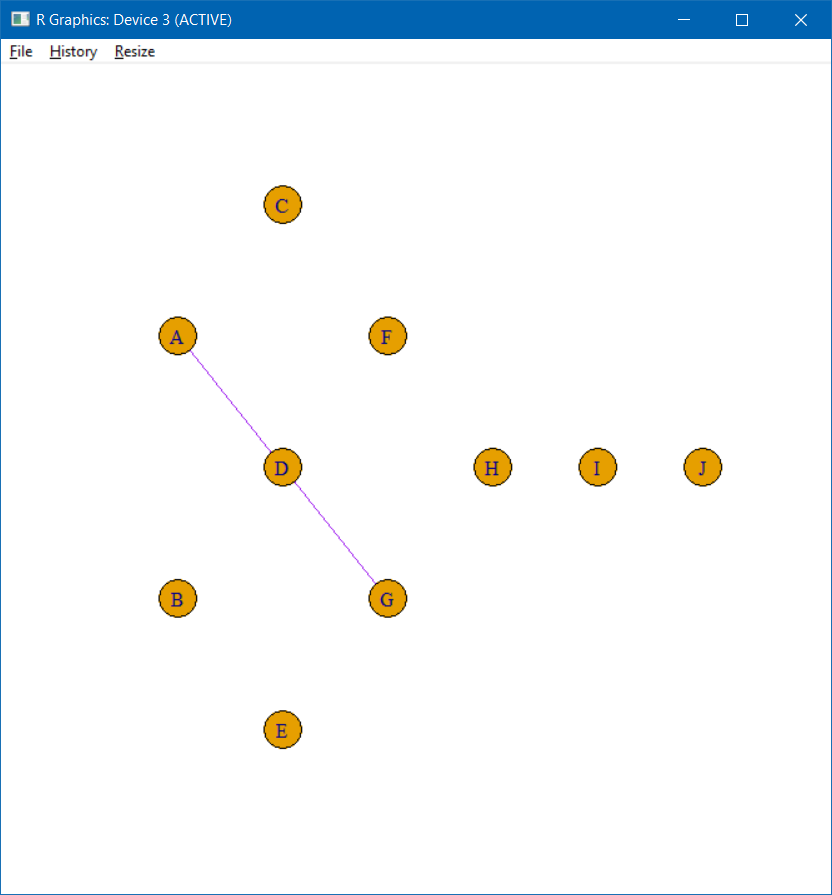


* 1. New network after adding predicted edges

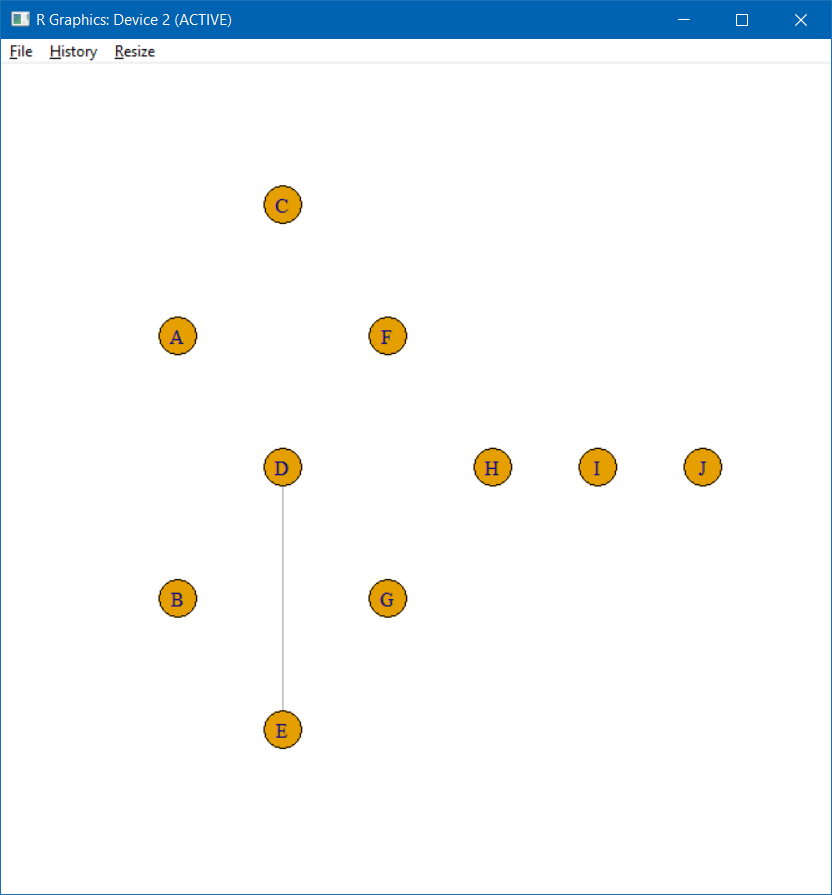


(New edge connects A and G)

* 1. Difference plots

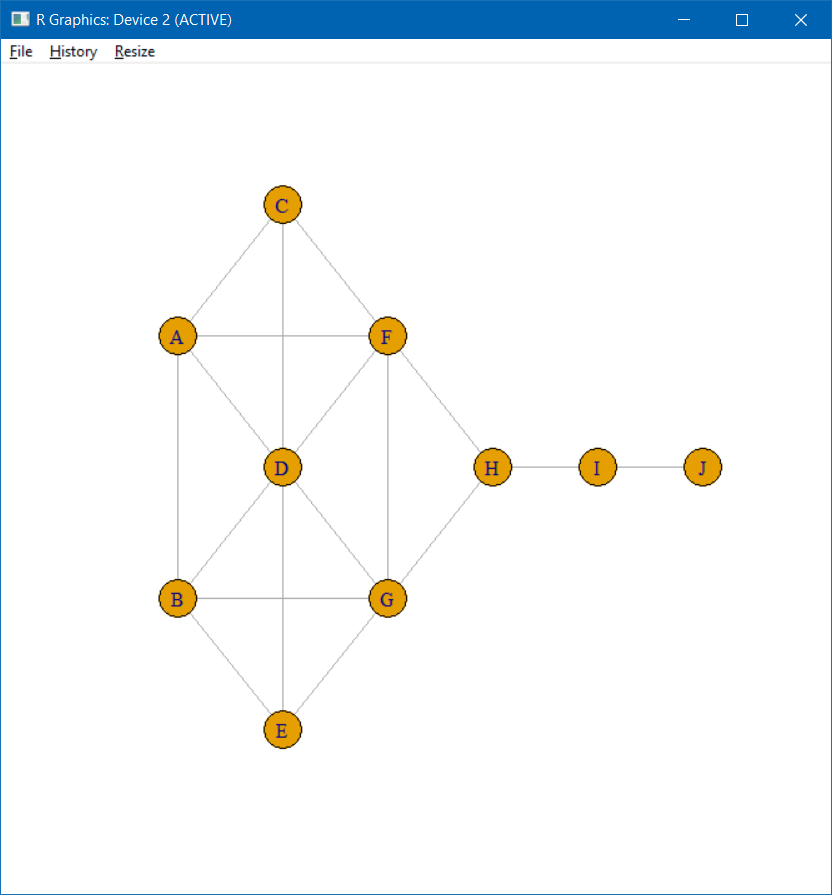


Edges that were added to new network which were not previously deleted from original

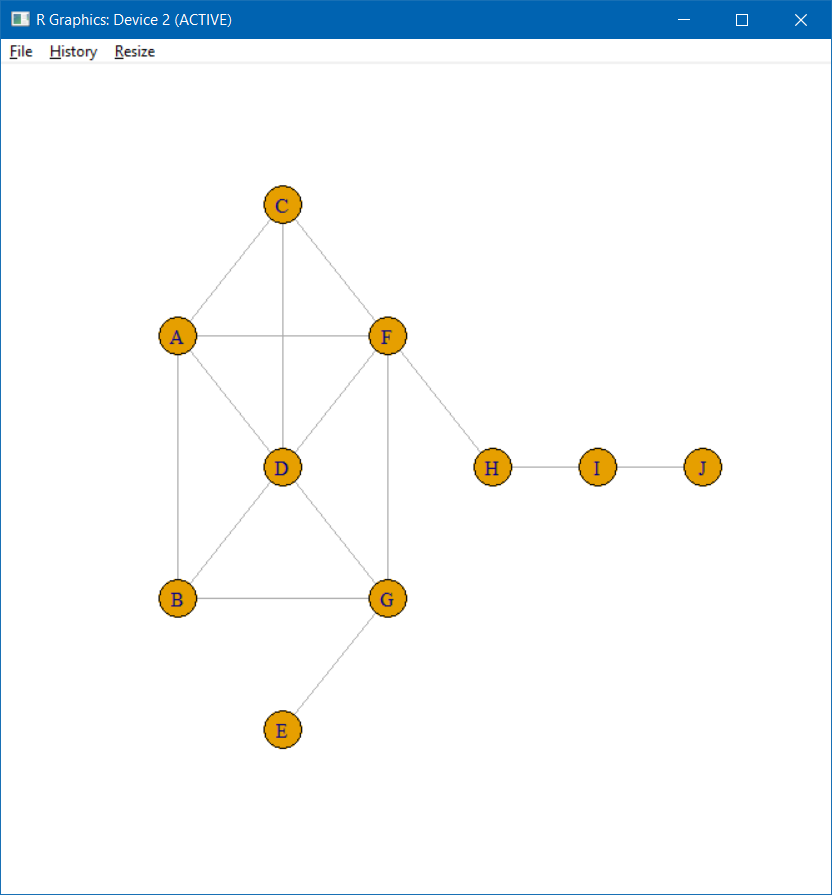


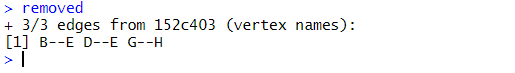
Edges that were deleted from original but not added to the new network due to lower probability.

1. For the next step I simply changed the perc = 0.15
   1. The original network

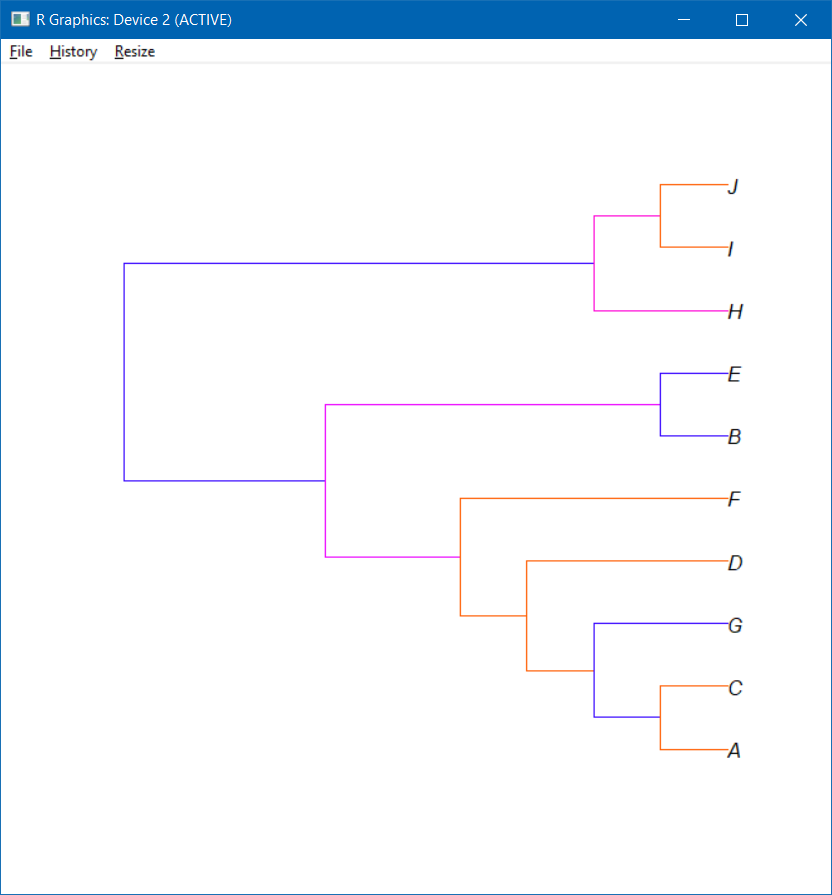


* 1. Network after 15% edge deletion

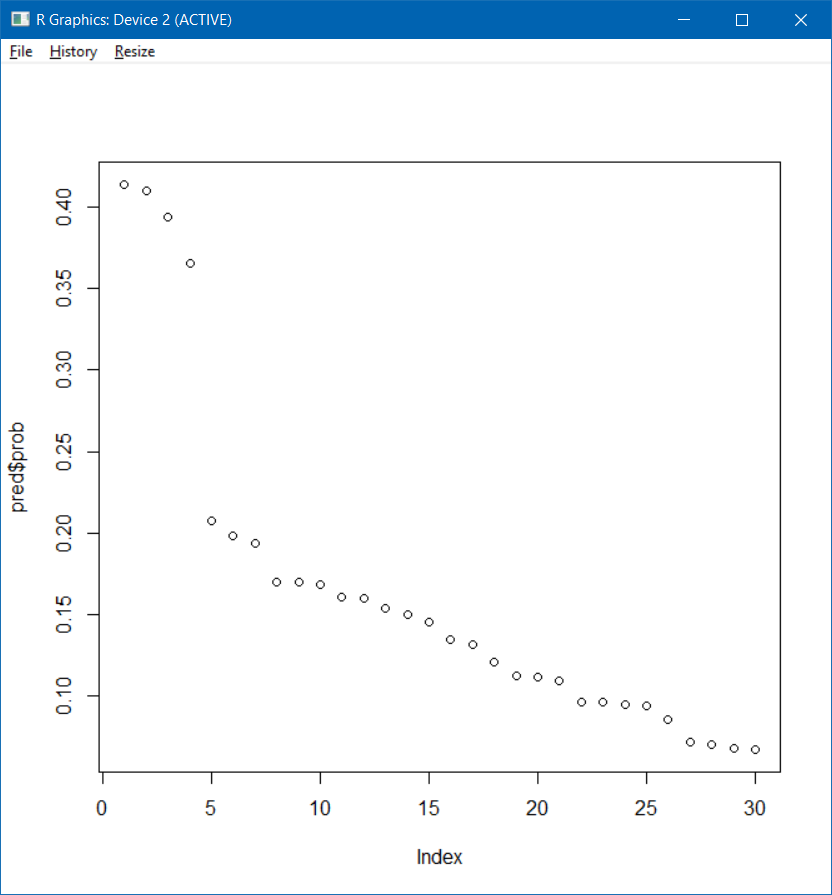




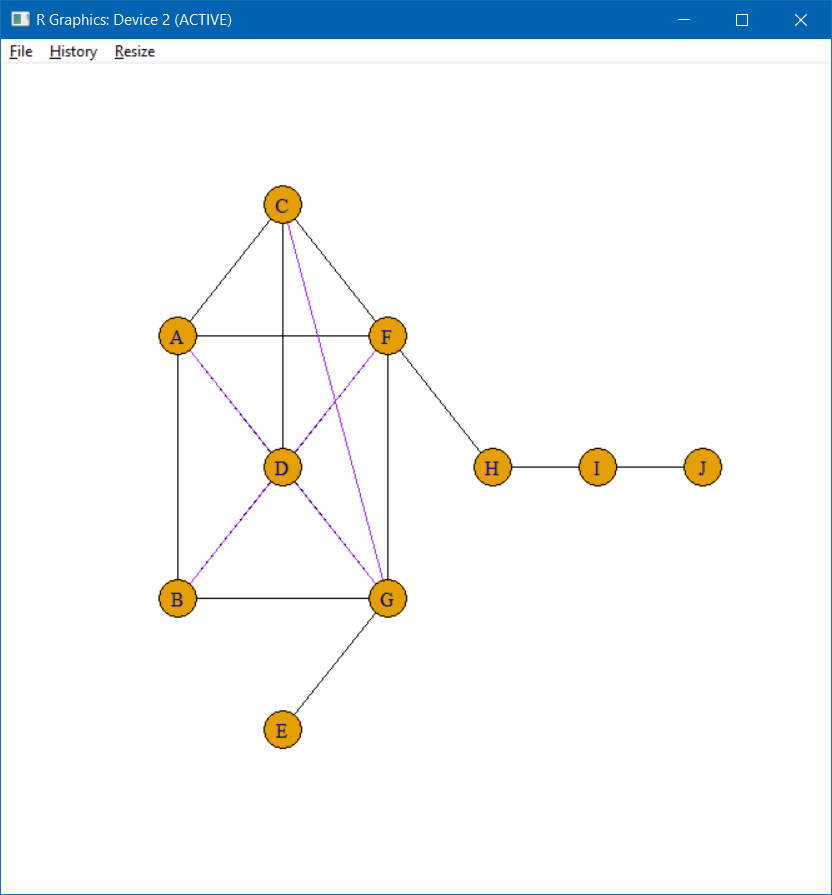
* 1. Dendrogram after fitting hierarchical random graph



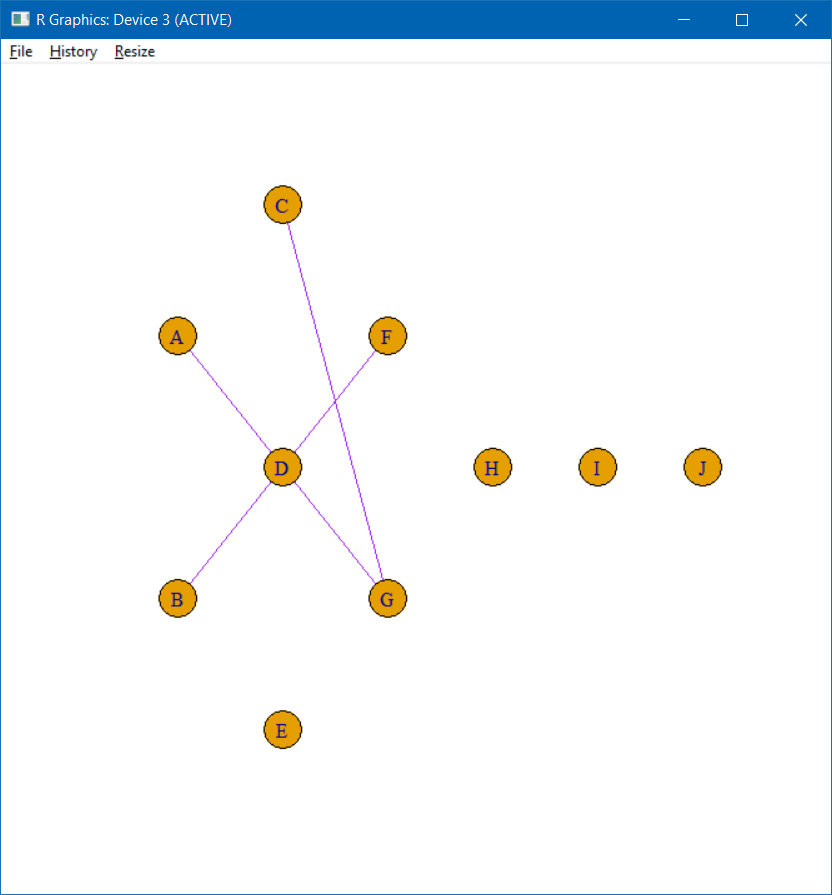
* 1. Probability graph of predicted edges



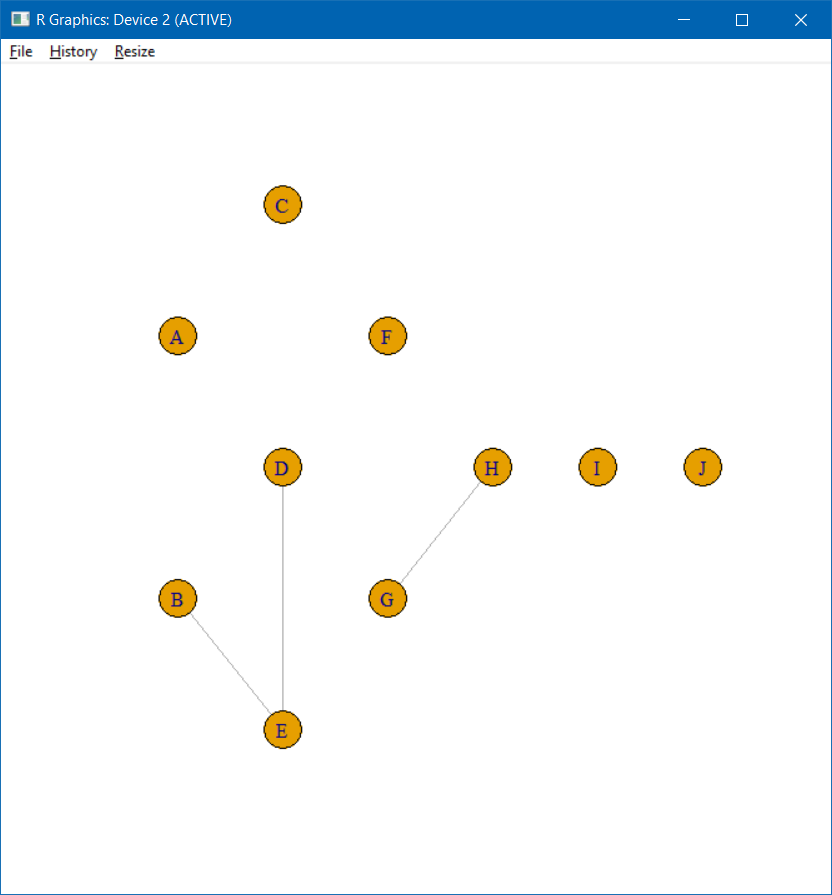
* 1. New network after adding predicted edges



* 1. Difference plots

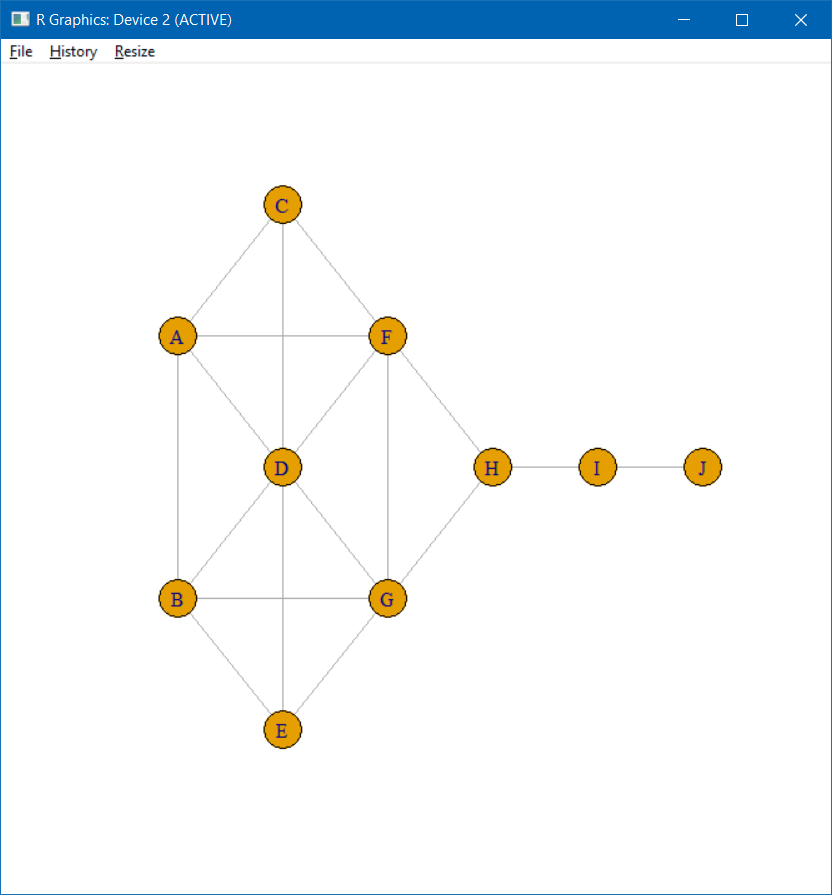


Edges that were added to new network which were not previously deleted from original

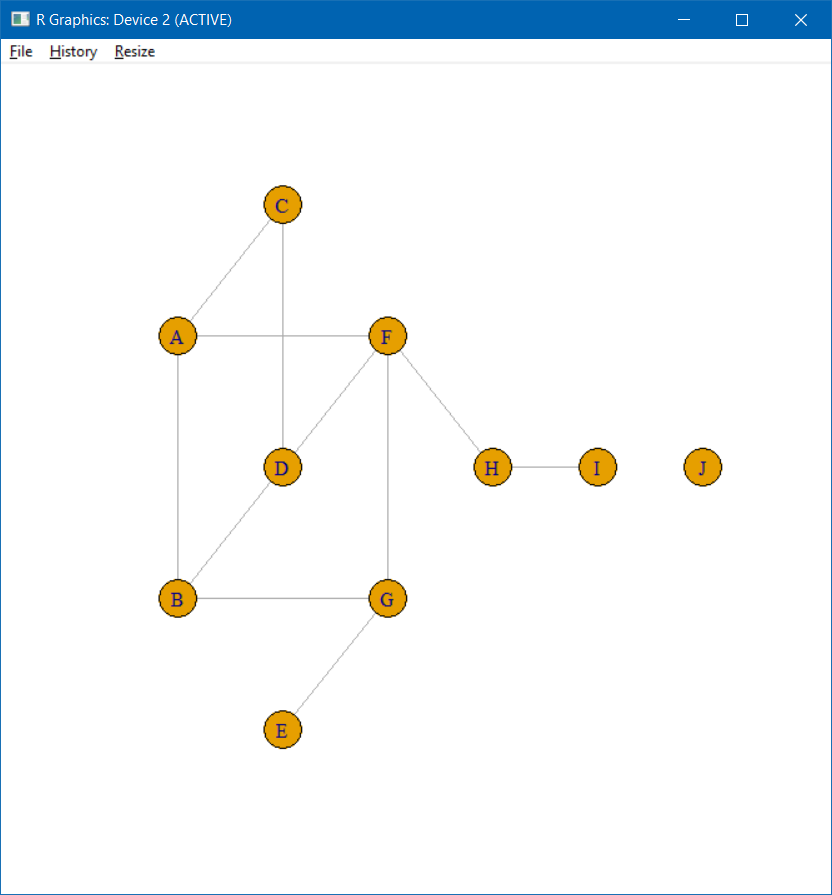


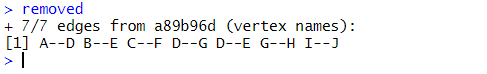
Edges that were deleted from original but not added to the new network due to lower probability.

1. For the next step I simply changed the perc = 0.40
   1. The original network

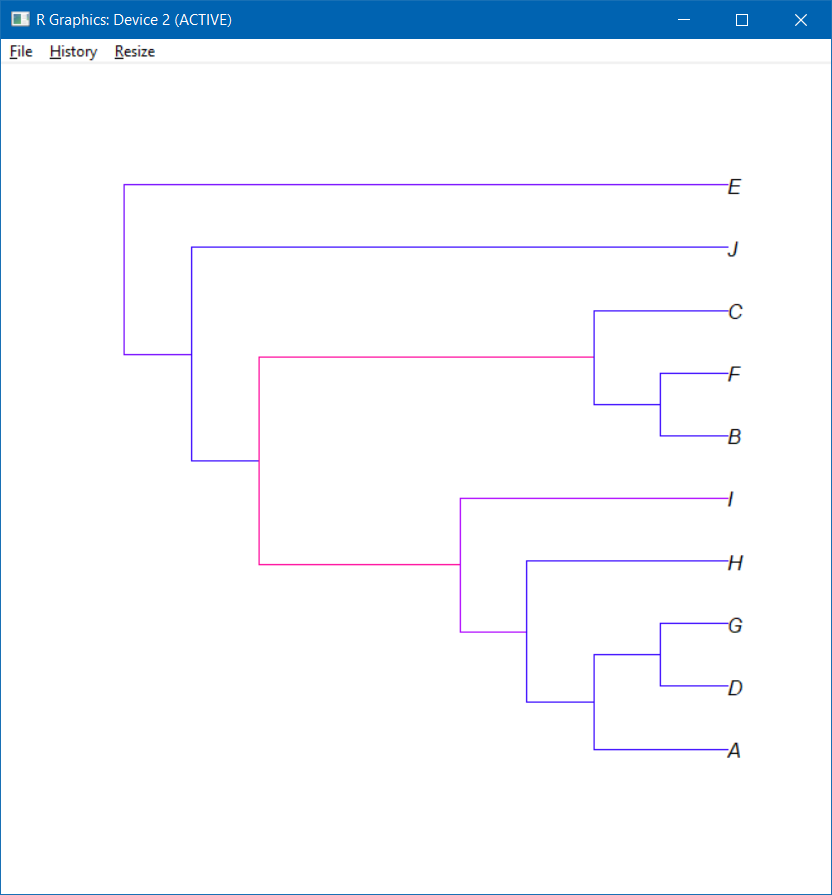


* 1. Network after 40% edge deletion

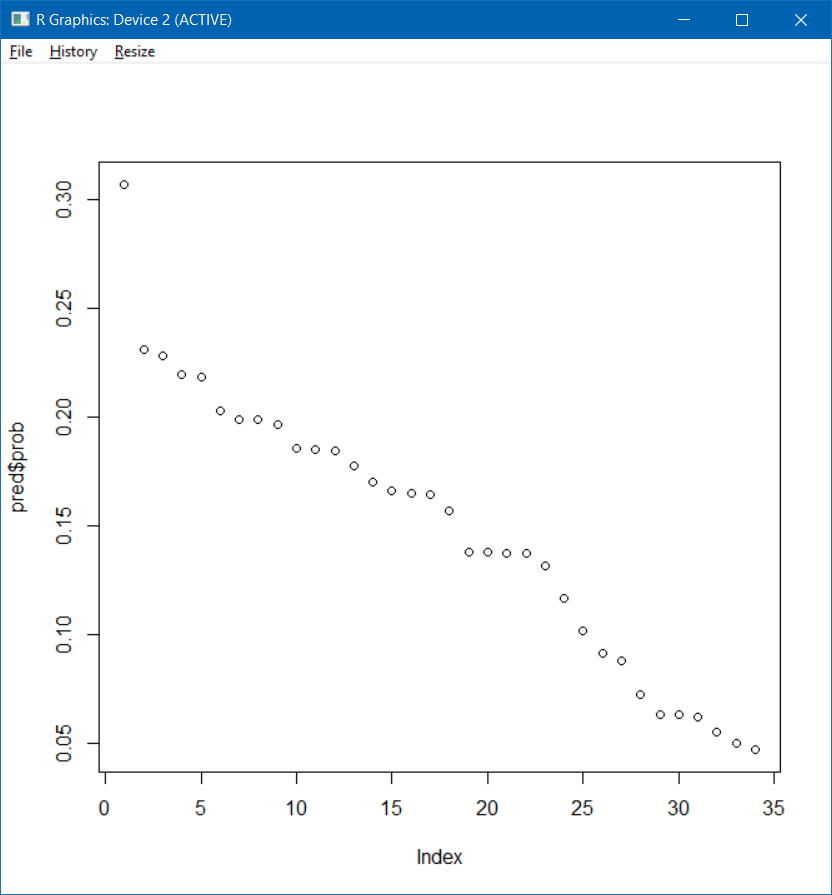




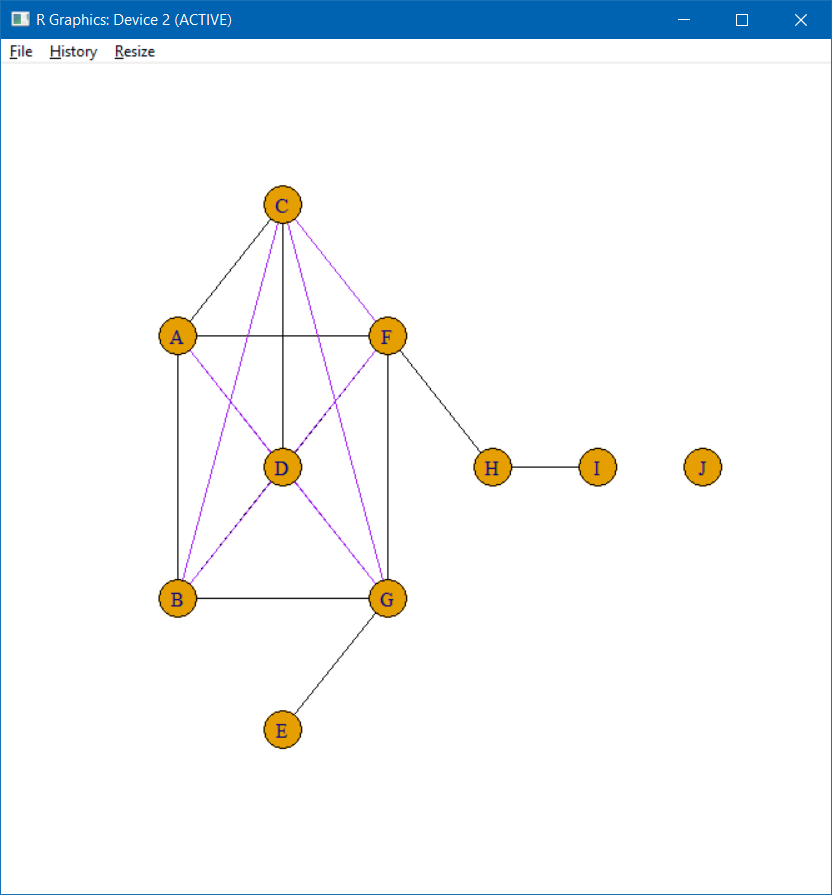
* 1. Dendrogram after fitting hierarchical random graph



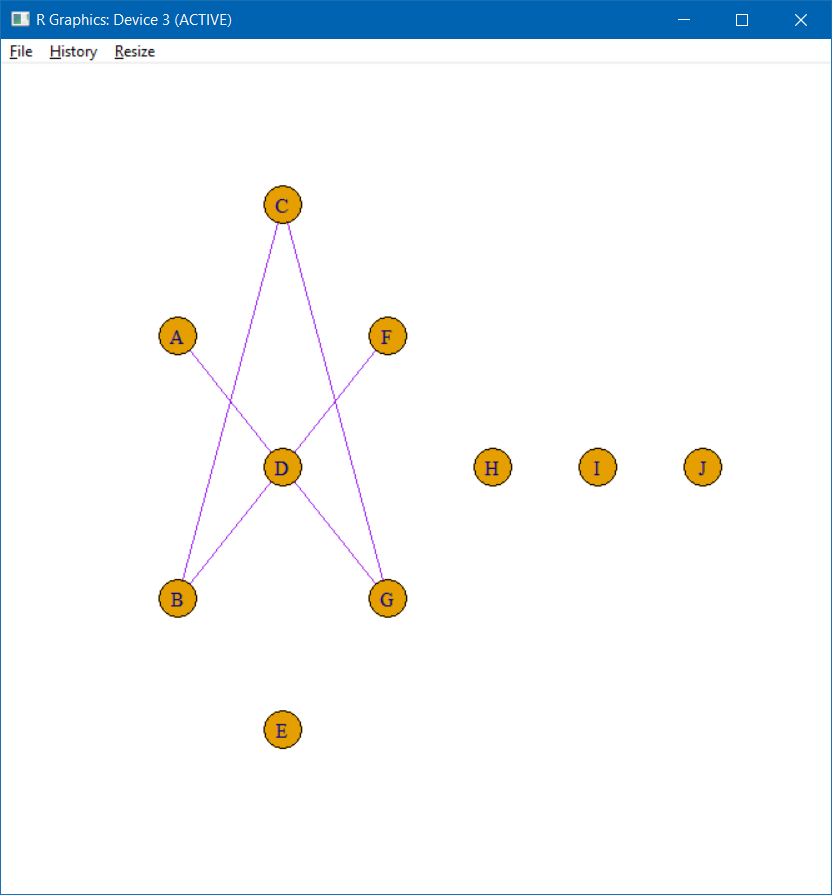
* 1. Probability graph of predicted edges



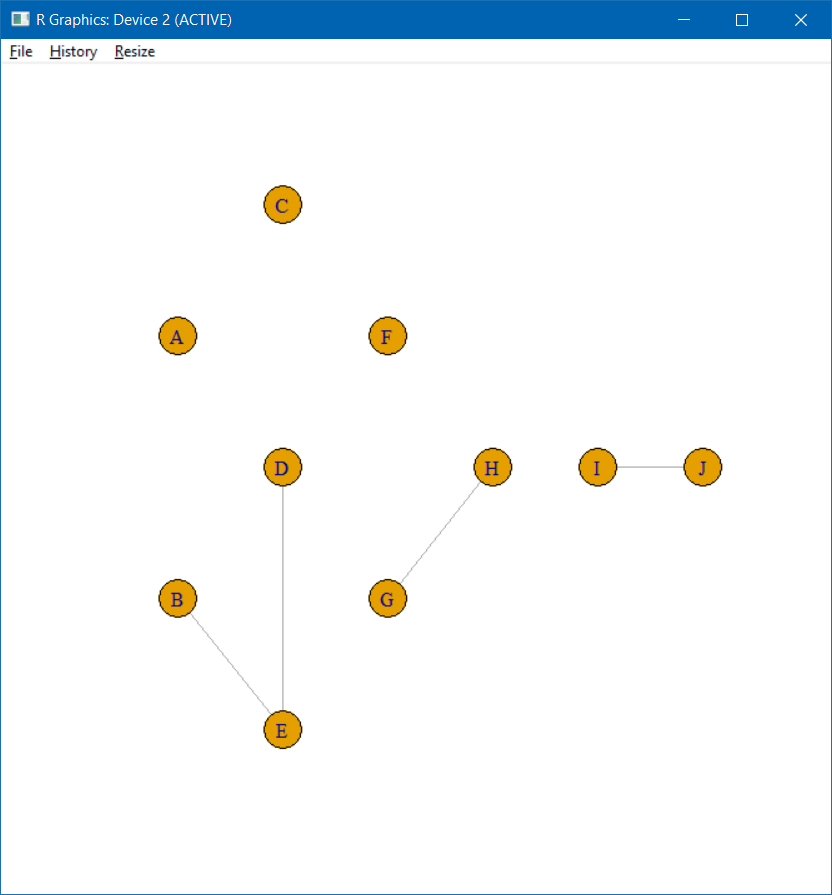
* 1. New network after adding predicted edges



* 1. Difference plots



Edges that were added to new network which were not previously deleted from original



Edges that were deleted from original but not added to the new network due to lower probability.

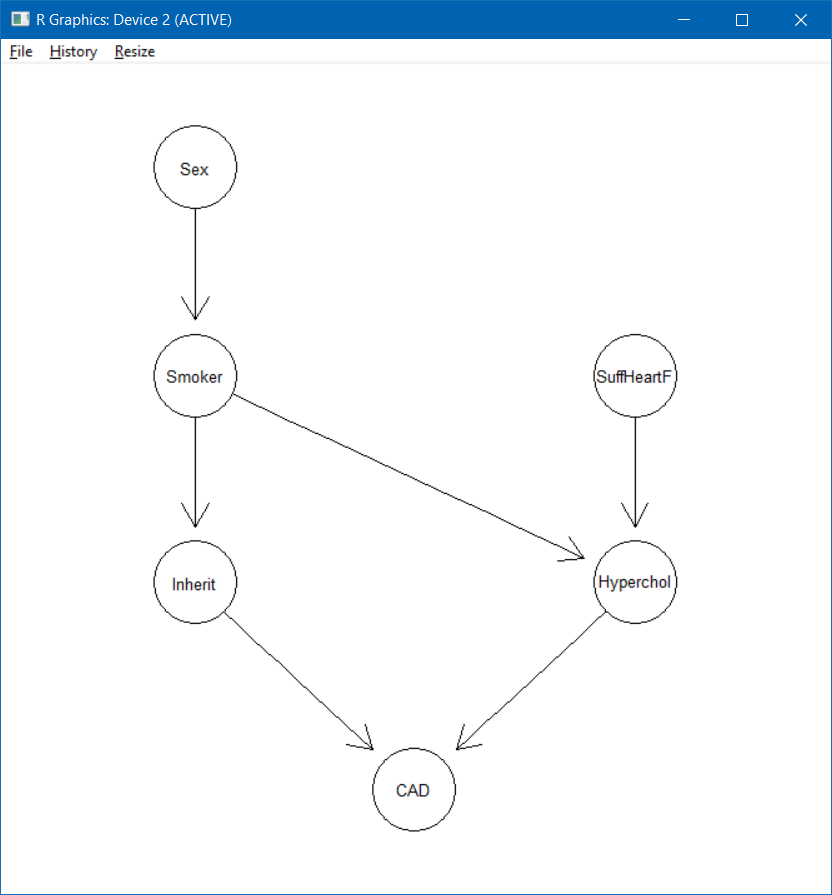
* Even for a network as small as kite, and a deletion of just 5% of the edges (1 edge), the predicted edge was not the one that was deleted. Not one of the cases had a complete success in terms of predicted values being the ones deleted from the original. Technically, the edges deleted from the original are always predicted, but they did not happen to be at the top of the list in terms of probability.
* This only goes to indicate that the predicted edges are based completely on the relation between the vertices of the network from which they are being predicted, and therefore it is incredibly difficult to recreate the original network by simply predicting the missing edges of a partial network.
* Another key observation, chiefly from the cases where a large percentage of edges are removed, is that vertices which get isolated out due to all edges connecting them being removed find it very hard to rejoin the network. This is based on observing the list of predicted edges, edges involving vertices that had been isolated were typically at the bottom of the list in terms of their probability. This is pretty intuitive.

Problem 2

1. False
2. True
3. False
4. True
5. False

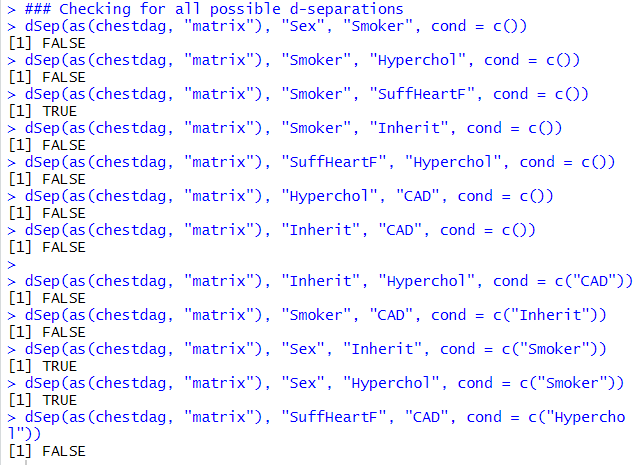
Problem 3

1. For the third problem, the first step was to install all the packages required, which required me to update my version of R and install them via Bioconductor. Once that was completed, I set my working directory as my HW4 folder so I could access the data I saved at the end of the code.
2. The first major step in the code was to specify the directed acyclic graph. I did that by first specifying a list object with the different elements of the network as per the diagram in the question and then make them a DAG using the dagList() command. After doing so, I plotted the graph using the optional type graphNEL to verify if it was indeed the graph specified in the question.



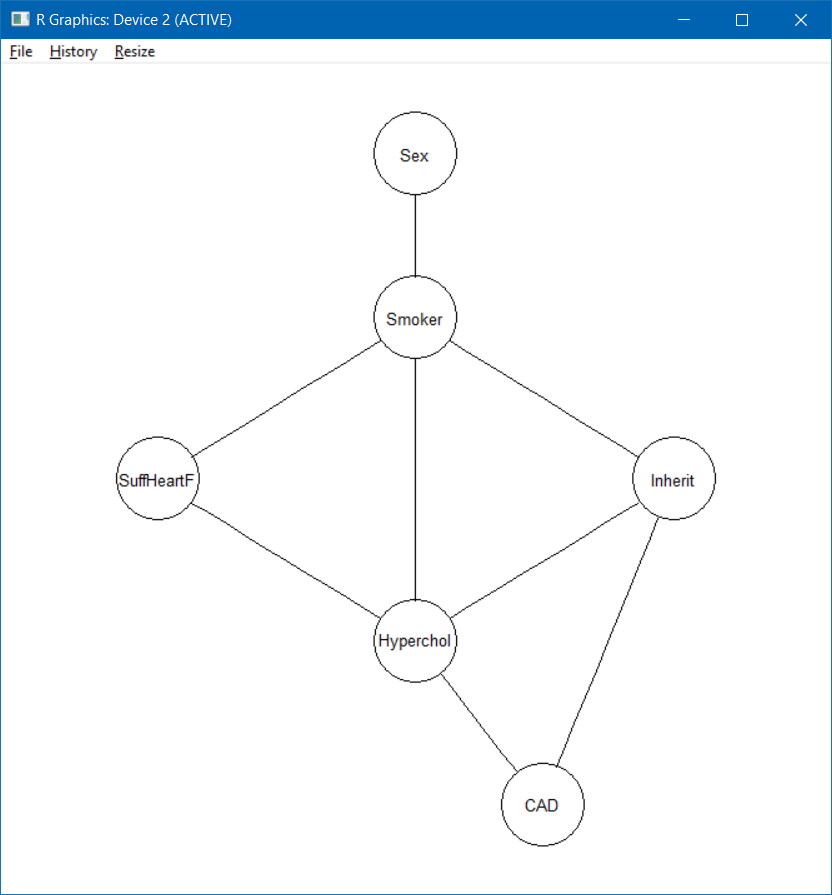
As is evident, the chestdag is the same as the graph specified in the question.

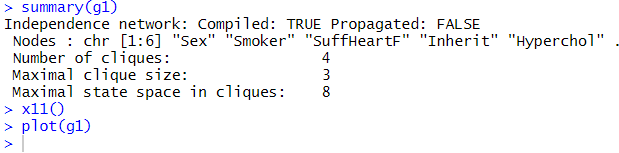
1. The next step was to check for all d-separations, which I did using the dSep() function



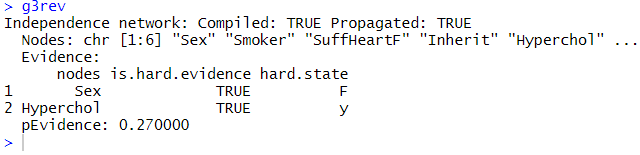
As can be seen, Smoker and SuffheartF are d-separated. Sex and Inherit are d-separated given Smoker, and Sex and Hyperchol are d-separated given Smoker

1. Next up was the task of specifying the conditional probability tables for each of the variables. As shown in the lab, I specified the two possible values for sex as M and F and then a simple list yesno for the two possible yes and no cases.
2. The rest of the cp table specification was fairly straightforward. I used arbitrary values for the specifications. I compiled the cptables using compileCPT() function and built the network using grain().

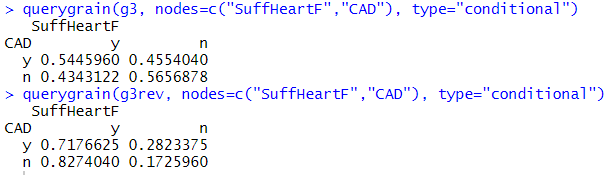




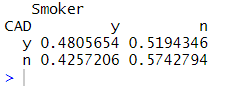
1. Finally, to compile the network, I used the compile() function and to propagate the belief values of the network, I used the propagate() function.
2. The next part of the question gives us information that must be absorbed by the network. I used setFinding to add the data of the patient (Sex = F, Hyperchol = y)



1. To record how the probability of heart failure and coronary artery disease changes after this new information, I recorded the initial value of this probability and then calculated it for the one with the added information.



1. As can be seen, the probabilities of suffering heart failure have increased significantly. The probability of suffering from heart failure but not from CAD has almost doubled, while the probability of suffering from neither of the diseases has plummeted significantly, as has the probability of suffering from CAD but not from heart failure.
2. To create the new dataset based on the conditional info from part B, I used the simulate.grain() function based on g3, and then added the information to be absorbed using the rbind() function. I presented this data in a table format as directed and saved it.
3. To check the probability of Smoker and CAD, I used the querygrain function again.



1. For the last step, I again used simulate.grain() to simulate a dataset of 500 observations based on the initial graph, and saved it in bigdata.RData