CTD Lab Exercise

Lillian Thistlethwaite

3/6/2019

This document was rendered at 2020-08-24 12:46:57

I. Generate background knowledge graph.

I.I: Manually build graphs from adjacency matrices.

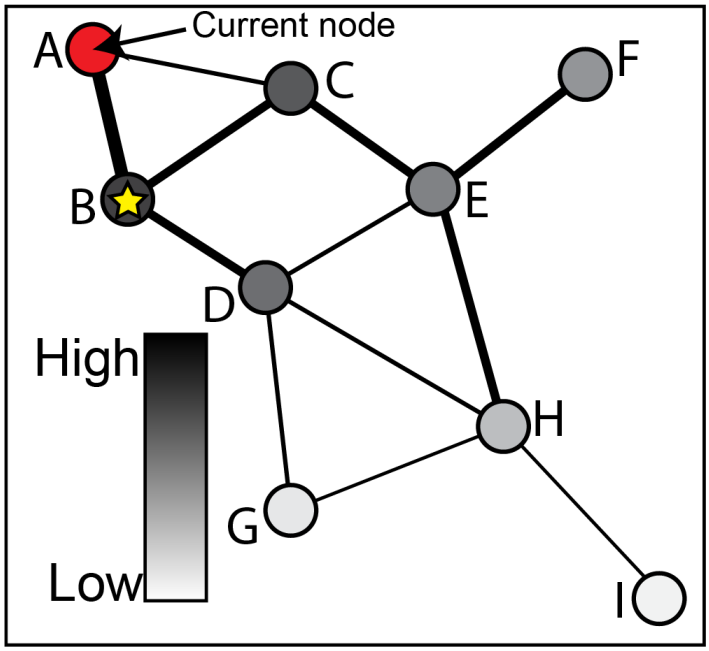


Figure 1: Figure 1. Cartoon of graph we will build manually via its adjacency matrix.

```
adj_mat = rbind(c(0,3,1,0,0,0,0,0,0), #A's neighbors
                c(3,0,2,2,0,0,0,0,0), #B's neighbors
                c(1,2,0,0,2,0,0,0,0), #C's neighbors
                c(0,2,0,0,1,0,1,1,0), #D's neighbors
                c(0,0,2,1,0,2,0,2,0), #E's neighbors
                c(0,0,0,0,2,0,0,0,0), #F's neighbors
                c(0,0,0,1,0,0,0,1,0), #G's neighbors
                c(0,0,0,1,2,0,1,0,1), #H's neighbors
                c(0,0,0,0,0,0,0,1,0) #I's neighbors
                )
rownames(adj_mat) = c("A", "B", "C", "D", "E", "F", "G", "H", "I")
```

```r
colnames(adj_mat) = c("A", "B", "C", "D", "E", "F", "G", "H", "I")
# Convert adjacency matrices to igrpah objects for all three graphs.
ig = graph.adjacency(adj_mat, mode="undirected", weighted=TRUE, add.colnames = "name")
print(ig)
```

```
## IGRAPH d2d27dc UNW- 9 12 --
## + attr: name (v/c), weight (e/n)
## + edges from d2d27dc (vertex names):
##  [1] A--B A--C B--C B--D C--E D--E D--G D--H E--F E--H G--H H--I
```

I.II: Learn a graph from data.

Note all code chunks in sections I - IV may rely on lines in previous code chunks, so do not empty your environment between code chunks.

```r
# Load the Miller2015_Heparin dataset
data(Miller2015)
# Only include metabolites that are present in >90% reference samples.
fil.rate = as.numeric(Miller2015$`Times identifed in all 200 samples`)/200
names(fil.rate) = rownames(Miller2015)
data_mx = Miller2015[,grep("IEM_", colnames(Miller2015))]
data_mx = data_mx[which(fil.rate>0.90), ]
dim(data_mx)
```

```
## [1] 620 186
```

```r
# Remove any metabolites where any profile has a z-score > 1000. These are likely imputed raw values th
rmMets = names(which(apply(data_mx, 1, function(i) any(i>20))))
if (length(rmMets)>0) {
  data_mx = data_mx[-which(rownames(data_mx) %in% rmMets),]
}
dim(data_mx)
```

```
## [1] 208 186
```

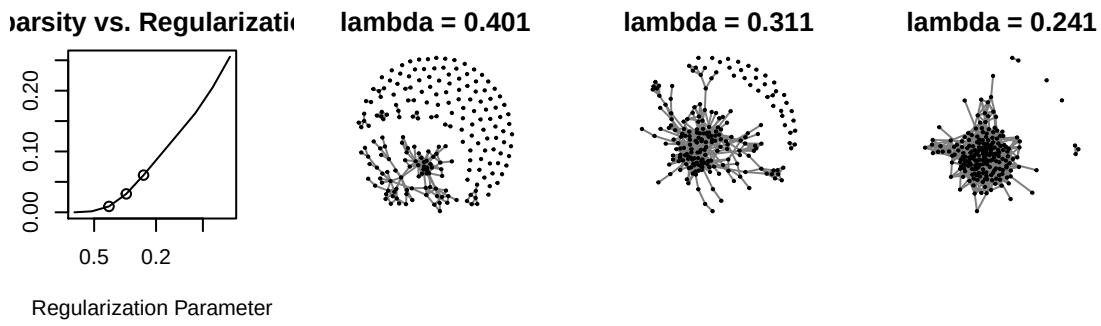
```r
# Get data from all patients with Argininemia
diags = Miller2015["diagnosis", grep("IEM", colnames(Miller2015))]
arg_data = data_mx[,which(diags=="Argininemia")]
# Add surrogate disease and surrogate reference profiles based on 1 standard deviation
# around profiles from real patients to improve rank of matrix when learning Gaussian
# Markov Random Field network on data. While surrogate profiles are not required, they tend
# to learn less complex networks (i.e., networks with less edges) and in faster time.
arg_data = data.surrogateProfiles(arg_data, 1, FALSE, TRUE, ref_data = data_mx[,which(diags=="No bioche
dim(arg_data)
```

```
## [1] 208 312
```

```r
# Learn a Gaussian Markov Random Field model using the Graphical LASSO in the R package "huge".
# Select the regularization parameter based on the "STARS" stability estimate.
require(huge)
#This will take 30 seconds - 1 minute.
arg = huge(t(arg_data), method="glasso")
```

```
## Conducting the graphical lasso (glasso) wtih lossless screening....in progress: 9%Conducting the gra
## Conducting the graphical lasso (glasso)....done.
```

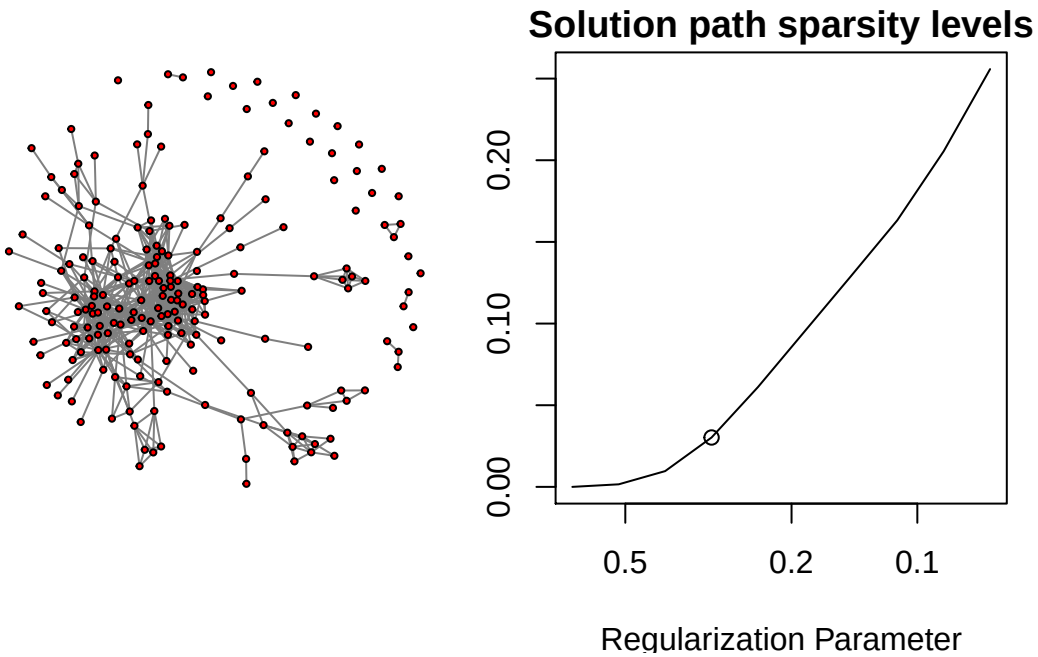
```
plot(arg)
```




```
# This will take several minutes. For a faster option, you can use the "ebic" criterion instead of "sta
arg.select = huge.select(arg, criterion="stars")
```

```
## Conducting Subsampling....in progress:5% Conducting Subsampling....in progress:10% Conducting Subsamp
```

```
plot(arg.select)
```




```
# This is the regularization parameter the STARS method selected.
print(arg.select$opt.lambda)
```

```
## [1] 0.310807
```

```
# This is the corresponding inverse of the covariance matrix that corresponds to the selected regulariz
arg_icov = as.matrix(arg.select$opt.icov)
# Remove all "self" edges, as we are not interested in self-relationships.
diag(arg_icov) = 0
rownames(arg_icov) = rownames(arg_data)
colnames(arg_icov) = rownames(arg_data)
# Convert adjacency matrices to an igraph object.
ig_arg = graph.adjacency(arg_icov, mode="undirected", weighted=TRUE, add.colnames = "name")
print(ig_arg)
```

```
## IGRAPH 041299e UNW- 208 651 --
## + attr: name (v/c), weight (e/n)
## + edges from 041299e (vertex names):
## [1] x - 11442              --1-palmitoyl-gpe (16:0)
## [2] x - 11442              --allantoin
## [3] x - 11442              --x - 17653
## [4] x - 11442              --x - 12792
## [5] x - 11442              --2-arachidonoyl-gpc (20:4)
## [6] x - 11442              --2-aminobutyrate
## [7] 1-palmitoyl-gpe (16:0)--2-oleoyl-gpe (18:1)
## [8] 1-palmitoyl-gpe (16:0)--1-oleoyl-gpe (18:1)
## + ... omitted several edges
```

II. The Probability Diffusion Algorithm

II.I From a starting node.

Run the following code, then go to the directory, and open all diffusionEventMovie*.png files all at once. Starting from the first image, view how the probability diffusion algorithm works to diffuse 100% probability to the rest of the graph. Be sure to pay attention to the recursion level listed in the title of each image, to imagine where in the call stack the algorithm is at the captured time the image was generated.

```
G=vector(mode="list", length=length(V(ig)$name))
G[1:length(G)] = 0
names(G) = c("A", "B", "C", "D", "E", "F", "G", "H", "I")
startNode = "A"
visitedNodes = startNode
coords = layout.fruchterman.reingold(ig)
.GlobalEnv$imgNum = 1
G_new = graph.diffuseP1(p1=1.0, startNode=startNode, G=G, visitedNodes=visitedNodes, thresholdDiff=0.01
                        adj_mat=adj_mat, verbose=TRUE, output_dir = sprintf("%s/images", getwd()),
                        recursion_level = 1, coords = coords)
```

```
## [1] "prob. to diffuse:1.000000 startNode: A, visitedNodes: A"
## [1] "added diffused  probability 0.750000 to child #1: B"
## [1] "subtracted 0.375000 from startNode's neighbor #1: B and sent to its own neighbors."
## [1] "   prob. to diffuse:0.375000 startNode: B, visitedNodes: A, B"
## [1] "   added diffused  probability 0.187500 to child #1: C"
## [1] "   subtracted 0.093750 from startNode's neighbor #1: C and sent to its own neighbors."
## [1] "      prob. to diffuse:0.093750 startNode: C, visitedNodes: A, B, C"
## [1] "      added diffused  probability 0.093750 to child #1: E"
## [1] "      subtracted 0.046875 from startNode's neighbor #1: E and sent to its own neighbors."
## [1] "         prob. to diffuse:0.046875 startNode: E, visitedNodes: A, B, C, E"
## [1] "         added diffused  probability 0.009375 to child #1: D"
## [1] "         added diffused  probability 0.018750 to child #2: F"
## [1] "         added diffused  probability 0.018750 to child #3: H"
## [1] "   added diffused  probability 0.187500 to child #2: D"
## [1] "   subtracted 0.093750 from startNode's neighbor #2: D and sent to its own neighbors."
## [1] "      prob. to diffuse:0.093750 startNode: D, visitedNodes: A, B, D"
## [1] "      added diffused  probability 0.031250 to child #1: E"
## [1] "      subtracted 0.015625 from startNode's neighbor #1: E and sent to its own neighbors."
## [1] "         prob. to diffuse:0.015625 startNode: E, visitedNodes: A, B, D, E"
## [1] "         added diffused  probability 0.005208 to child #1: C"
## [1] "         added diffused  probability 0.005208 to child #2: F"
## [1] "         added diffused  probability 0.005208 to child #3: H"
## [1] "      added diffused  probability 0.031250 to child #2: G"
```

```
## [1] "      subtracted 0.015625 from startNode's neighbor #2: G and sent to its own neighbors."
## [1] "         prob. to diffuse:0.015625 startNode: G, visitedNodes: A, B, D, G"
## [1] "         added diffused  probability 0.015625 to child #1: H"
## [1] "      added diffused  probability 0.031250 to child #3: H"
## [1] "      subtracted 0.015625 from startNode's neighbor #3: H and sent to its own neighbors."
## [1] "         prob. to diffuse:0.015625 startNode: H, visitedNodes: A, B, D, H"
## [1] "         added diffused  probability 0.007812 to child #1: E"
## [1] "         added diffused  probability 0.003906 to child #2: G"
## [1] "         added diffused  probability 0.003906 to child #3: I"
## [1] "added diffused  probability 0.250000 to child #2: C"
## [1] "subtracted 0.125000 from startNode's neighbor #2: C and sent to its own neighbors."
## [1] "   prob. to diffuse:0.125000 startNode: C, visitedNodes: A, C"
## [1] "   added diffused  probability 0.062500 to child #1: B"
## [1] "   subtracted 0.031250 from startNode's neighbor #1: B and sent to its own neighbors."
## [1] "      prob. to diffuse:0.031250 startNode: B, visitedNodes: A, C, B"
## [1] "      added diffused  probability 0.031250 to child #1: D"
## [1] "      subtracted 0.015625 from startNode's neighbor #1: D and sent to its own neighbors."
## [1] "         prob. to diffuse:0.015625 startNode: D, visitedNodes: A, C, B, D"
## [1] "         added diffused  probability 0.005208 to child #1: E"
## [1] "         added diffused  probability 0.005208 to child #2: G"
## [1] "         added diffused  probability 0.005208 to child #3: H"
## [1] "   added diffused  probability 0.062500 to child #2: E"
## [1] "   subtracted 0.031250 from startNode's neighbor #2: E and sent to its own neighbors."
## [1] "      prob. to diffuse:0.031250 startNode: E, visitedNodes: A, C, E"
## [1] "      added diffused  probability 0.006250 to child #1: D"
## [1] "      added diffused  probability 0.012500 to child #2: F"
## [1] "      added diffused  probability 0.012500 to child #3: H"
```

```
# Inherited probabilities across all nodes should add to 1.
sum(unlist(G_new))
```

```
## [1] 1
```

```
# Which node inherited the highest probability from startNode?
G_new[which.max(G_new)]
```

```
## $B
## [1] 0.40625
```

II.II From a starting node, after visiting previous nodes.

Now, delete all diffusionEventMovie*.png files from your current directory, and run the following code. View the new image stack in the same way we did previously.

```
# Now let's see how the probability diffusion algorithm diffuses probability after B has been "stepped"
visitedNodes = c("A", "B")
startNode = "B"
.GlobalEnv$imgNum=1

G_new = graph.diffuseP1(p1=1.0, startNode, G, visitedNodes, thresholdDiff=0.01, adj_mat, TRUE, output_di
```

```
## [1] "prob. to diffuse:1.000000 startNode: B, visitedNodes: A, B"
## [1] "added diffused  probability 0.500000 to child #1: C"
## [1] "subtracted 0.250000 from startNode's neighbor #1: C and sent to its own neighbors."
## [1] "   prob. to diffuse:0.250000 startNode: C, visitedNodes: A, B, C"
## [1] "   added diffused  probability 0.250000 to child #1: E"
## [1] "   subtracted 0.125000 from startNode's neighbor #1: E and sent to its own neighbors."
## [1] "      prob. to diffuse:0.125000 startNode: E, visitedNodes: A, B, C, E"
## [1] "      added diffused  probability 0.025000 to child #1: D"
## [1] "      subtracted 0.012500 from startNode's neighbor #1: D and sent to its own neighbors."
## [1] "         prob. to diffuse:0.012500 startNode: D, visitedNodes: A, B, C, E, D"
## [1] "         added diffused  probability 0.006250 to child #1: G"
## [1] "         added diffused  probability 0.006250 to child #2: H"
## [1] "      added diffused  probability 0.050000 to child #2: F"
## [1] "      subtracted 0.025000 from startNode's neighbor #2: F and sent to its own neighbors."
## [1] "         prob. to diffuse:0.025000 startNode: F, visitedNodes: A, B, C, E, F"
## [1] "         added diffused  probability 0.008333 to child #1: D"
## [1] "         added diffused  probability 0.016667 to child #2: H"
## [1] "      added diffused  probability 0.050000 to child #3: H"
## [1] "      subtracted 0.025000 from startNode's neighbor #3: H and sent to its own neighbors."
## [1] "         prob. to diffuse:0.025000 startNode: H, visitedNodes: A, B, C, E, H"
## [1] "         added diffused  probability 0.008333 to child #1: D"
## [1] "         added diffused  probability 0.008333 to child #2: G"
## [1] "         added diffused  probability 0.008333 to child #3: I"
## [1] "added diffused  probability 0.500000 to child #2: D"
## [1] "subtracted 0.250000 from startNode's neighbor #2: D and sent to its own neighbors."
## [1] "   prob. to diffuse:0.250000 startNode: D, visitedNodes: A, B, D"
## [1] "   added diffused  probability 0.083333 to child #1: E"
## [1] "   subtracted 0.041667 from startNode's neighbor #1: E and sent to its own neighbors."
## [1] "      prob. to diffuse:0.041667 startNode: E, visitedNodes: A, B, D, E"
```

```
## [1] "      added diffused  probability 0.013889 to child #1: C"
## [1] "      added diffused  probability 0.013889 to child #2: F"
## [1] "      added diffused  probability 0.013889 to child #3: H"
## [1] "   added diffused  probability 0.083333 to child #2: G"
## [1] "   subtracted 0.041667 from startNode's neighbor #2: G and sent to its own neighbors."
## [1] "      prob. to diffuse:0.041667 startNode: G, visitedNodes: A, B, D, G"
## [1] "      added diffused  probability 0.041667 to child #1: H"
## [1] "      subtracted 0.020833 from startNode's neighbor #1: H and sent to its own neighbors."
## [1] "         prob. to diffuse:0.020833 startNode: H, visitedNodes: A, B, D, G, H"
## [1] "         added diffused  probability 0.013889 to child #1: E"
## [1] "         added diffused  probability 0.006944 to child #2: I"
## [1] "   added diffused  probability 0.083333 to child #3: H"
## [1] "   subtracted 0.041667 from startNode's neighbor #3: H and sent to its own neighbors."
## [1] "      prob. to diffuse:0.041667 startNode: H, visitedNodes: A, B, D, H"
## [1] "      added diffused  probability 0.020833 to child #1: E"
## [1] "      subtracted 0.010417 from startNode's neighbor #1: E and sent to its own neighbors."
## [1] "         prob. to diffuse:0.010417 startNode: E, visitedNodes: A, B, D, H, E"
## [1] "         added diffused  probability 0.005208 to child #1: C"
## [1] "         added diffused  probability 0.005208 to child #2: F"
## [1] "      added diffused  probability 0.010417 to child #2: G"
## [1] "      added diffused  probability 0.010417 to child #3: I"
```

```
# Inherited probabilities across all nodes should add to 1.
sum(unlist(G_new))
```

```
## [1] 1
```

```
# Which node inherited the highest probability from startNode?
G_new[which.max(G_new)]
```

```
## $D
## [1] 0.2791667
```

II.III Diffusing through visited nodes, based on connectivity.

Sometimes the startNode is "stranded" by a bunch of visited nodes. The diffusion algorithm diffuses "through" visited nodes, so that nodes in the same connected component can be prioritized over nodes in a different connected component, or "island nodes" (e.g. in the below code snippet, "I" is an island node). This only works currently for nodes 2 hops away from the current startNode, however.

```
adj_mat = rbind(c(0,1,2,0,0,0,0,0,0), #A's neighbors
                c(1,0,3,0,0,0,0,0,0), #B's neighbors
                c(2,3,0,0,1,0,0,0,0), #C's neighbors
                c(0,0,0,0,0,0,1,1,0), #D's neighbors
                c(0,0,1,0,0,1,0,0,0), #E's neighbors
```

```
                c(0,0,0,0,1,0,0,0,0), #F's neighbors
                c(0,0,0,1,0,0,0,1,0), #G's neighbors
                c(0,0,0,1,0,0,1,0,0), #H's neighbors
                c(0,0,0,0,0,0,0,0,0) #I's neighbors
                )
rownames(adj_mat) = c("A", "B", "C", "D", "E", "F", "G", "H", "I")
colnames(adj_mat) = c("A", "B", "C", "D", "E", "F", "G", "H", "I")
# Convert adjacency matrices to igrpah objects for all three graphs.
ig = graph.adjacency(adj_mat, mode="undirected", weighted=TRUE, add.colnames = "name")
coords = layout.fruchterman.reingold(ig)
print(ig)
```

```
## IGRAPH f19c39c UNW- 9 8 --
## + attr: name (v/c), weight (e/n)
## + edges from f19c39c (vertex names):
## [1] A--B A--C B--C C--E D--G D--H E--F G--H
```

```
# Now let's see how the probability diffusion algorithm diffuses probability after B has been "stepped"
visitedNodes = c("B", "C", "A")
startNode = "A"
.GlobalEnv$imgNum = 1
G_new = graph.diffuseP1(p1=1.0, startNode, G, visitedNodes, thresholdDiff=0.01, adj_mat, TRUE, output_di
```

```
## [1] "prob. to diffuse:1.000000 startNode: A, visitedNodes: B, C, A"

## [1] "added diffused  probability 1.000000 to child #1: E"

## [1] "subtracted 0.500000 from startNode's neighbor #1: E and sent to its own neighbors."

## [1] "   prob. to diffuse:0.500000 startNode: E, visitedNodes: B, C, A, E"

## [1] "   added diffused  probability 0.500000 to child #1: F"

## [1] "   subtracted 0.250000 from startNode's neighbor #1: F and sent to its own neighbors."

## [1] "      prob. to diffuse:0.250000 startNode: F, visitedNodes: B, C, A, E, F"

## [1] "startNode F is stranded with its visited neighbors, or is a singleton. Diffuse p1 uniformly amor
```

```
# Inherited probabilities across all nodes should add to 1.
sum(unlist(G_new))
```

```
## [1] 1
```

```
# Which node inherited the highest probability from startNode?
G_new[which.max(G_new)]
```

```
## $E
## [1] 0.5
```

III. The Network Encoding Algorithms

III.I Multi-Node Diffusion Encoding

```
# The multi-node network walker tends to overfit on the network and is more computationally
# intensive/slow compared to the single-node network walker. It is therefore not recommended that
# you use this network walker over the single-node network walker. However, it is unclear if this
```

```
# network walker can be beneficial in some circumstances or application areas.
adj_mat = rbind(c(0,3,1,0,0,0,0,0,0), #A's neighbors
                c(3,0,2,2,0,0,0,0,0), #B's neighbors
                c(1,2,0,0,2,0,0,0,0), #C's neighbors
                c(0,2,0,0,1,0,1,1,0), #D's neighbors
                c(0,0,2,1,0,2,0,2,0), #E's neighbors
                c(0,0,0,0,2,0,0,0,0), #F's neighbors
                c(0,0,0,1,0,0,0,1,0), #G's neighbors
                c(0,0,0,1,2,0,1,0,1), #H's neighbors
                c(0,0,0,0,0,0,0,1,0) #I's neighbors
                )
rownames(adj_mat) = c("A", "B", "C", "D", "E", "F", "G", "H", "I")
colnames(adj_mat) = c("A", "B", "C", "D", "E", "F", "G", "H", "I")
# Convert adjacency matrices to igrpah objects for all three graphs.
ig = graph.adjacency(adj_mat, mode="undirected", weighted=TRUE, add.colnames = "name")
coords = layout.fruchterman.reingold(ig)
print(ig)
```

```
## IGRAPH b397e78 UNW- 9 12 --
## + attr: name (v/c), weight (e/n)
## + edges from b397e78 (vertex names):
##  [1] A--B A--C B--C B--D C--E D--E D--G D--H E--F E--H G--H H--I
```

```
# Generate PNGs to animate the multi-node encoding node ranks.
ranks = multiNode.getNodeRanks(S = c("A", "B"), G, p1=1.0, thresholdDiff=0.01, adj_mat, log2(length(G))

# Get node ranks as list object, with no images generated
ranks = multiNode.getNodeRanks(S = c("A", "B"), G, p1=1.0, thresholdDiff=0.01, adj_mat, log2(length(G))
```

III.II Single-Node Diffusion Encoding

```
# This network walker tends to avoid overfitting. Of further note, since single-node network
# walker is not parameterized by the subset being encoded, you can pre-compute node rankings
# using dynamic programming. Pre-computing node ranks enables quick encoding of thousands of subsets
# at a time (see The Encoding Process).
S = c("A", "B")
# Generate PNGs to animate the single-node encoding node ranks.
ranks = list()
for (n in seq_len(length(S))) {
  ind = which(names(G)==S[n])
  ranks[[n]] = singleNode.getNodeRanksN(ind, G, p1=1.0, thresholdDiff=0.01, adj_mat, S = S,
                                        num.misses = log2(length(G)), FALSE,
                                        output_dir=sprintf("%s/images", getwd()), TRUE, coords)
}
names(ranks) = S

# Get node ranks as list object, with no images generated
S = c("A", "B")
ranks = list()
for (n in 1:length(S)) {
  ind = which(names(G)==S[n])
  ranks[[n]] = singleNode.getNodeRanksN(ind, G, p1=1.0, thresholdDiff=0.01, adj_mat, S = S,
                                        num.misses = log2(length(G)), FALSE)
```

```
}
names(ranks) = S
```

IV. The Encoding Process

We're going to go back to our data using the Arginase deficiency network model, and the Miller et al (2015) dataset. ## IV.0 Re-define the Arginase deficiency network model

```
print(ig_arg)
```

```
## IGRAPH 041299e UNW- 208 651 --
## + attr: name (v/c), weight (e/n)
## + edges from 041299e (vertex names):
## [1] x - 11442             --1-palmitoyl-gpe (16:0)
## [2] x - 11442             --allantoin
## [3] x - 11442             --x - 17653
## [4] x - 11442             --x - 12792
## [5] x - 11442             --2-arachidonoyl-gpc (20:4)
## [6] x - 11442             --2-aminobutyrate
## [7] 1-palmitoyl-gpe (16:0)--2-oleoyl-gpe (18:1)
## [8] 1-palmitoyl-gpe (16:0)--1-oleoyl-gpe (18:1)
## + ... omitted several edges
```

```
adj_mat = as.matrix(get.adjacency(ig_arg, attr="weight"))
G=vector(mode="list", length=length(V(ig_arg)$name))
G[1:length(G)] = 0
names(G) = V(ig_arg)$name
```

IV.I Choose your node subset.

```
# Maximum subset size to inspect
kmx=15
# Get our node subset associated with the $KMX highest perturbed (up or down) in our first Arginase def
S_arg = sort(abs(arg_data[,1]), decreasing=TRUE)[1:kmx]
print(S_arg)
```

```
##                           x - 11381                       2-aminobutyrate
##                            4.197347                             3.122232
##                       dihydroorotate             2-arachidonoyl-gpc (20:4)
##                            3.117244                             2.942611
## 2-linoleoylglycerol (2-monolinolein)          1-docosahexaenoyl-gpc (22:6)
##                            2.862771                             2.754610
##                           x - 12792        gamma-glutamyl-2-aminobutyrate
##                            2.707036                             2.661916
##             1-arachidonoyl-gpc (20:4)                           x - 17001
##                            2.653048                             2.586215
##                  prolylhydroxyproline                           x - 11442
##                            2.537495                             2.314537
##                 1-stearoyl-gpc (18:0)                           x - 12798
##                            2.246071                             2.239582
##                             glucose
##                            2.175738
```

IV.II Get k node permutations.

```
# Get the single-node encoding node ranks starting from each node in the subset S_arg.
ranks = list()
for (n in 1:length(S_arg)) {
  ind = which(names(G)==names(S_arg)[n])
  ranks[[n]] = singleNode.getNodeRanksN(ind, G, p1=1.0, thresholdDiff=0.01, adj_mat, S = names(S_arg),
                                        num.misses = log2(length(G)), TRUE)
}
```

```
## [1] "Calculating node rankings 208 of 208."
## [1] "Calculating node rankings 207 of 208."
## [1] "Calculating node rankings 206 of 208."
## [1] "Calculating node rankings 205 of 208."
## [1] "Calculating node rankings 204 of 208."
## [1] "Calculating node rankings 203 of 208."
## [1] "Calculating node rankings 202 of 208."
## [1] "Calculating node rankings 201 of 208."
## [1] "Calculating node rankings 200 of 208."
## [1] "Calculating node rankings 199 of 208."
## [1] "Calculating node rankings 198 of 208."
## [1] "Calculating node rankings 1 of 208."
## [1] "Calculating node rankings 197 of 208."
## [1] "Calculating node rankings 196 of 208."
## [1] "Calculating node rankings 195 of 208."
```

```
names(ranks) = names(S_arg)
```

IV.III Convert to bitstrings.

```
# Get the bitstrings associated with the patient's perturbed metabolites in "S_arg" based
# on the node ranks calculated in the previous step, "ranks".
ptBSbyK = singleNode.getPtBSbyK(names(S_arg), ranks)
```

IV.IV Get encoding length of minimum length codeword.

```
ptID = colnames(arg_data)[1]
data_mx.pvals=apply(arg_data[,which(colnames(arg_data) %in% names(diags))], c(1,2), function(i) 2*pnorm
res = mle.getEncodingLength(ptBSbyK, t(data_mx.pvals), ptID, G)
ind.mx = which.max(res$d.score)
res[ind.mx,]
```

```
##   patientID    optimalBS subsetSize opt.T
## 9  IEM_1006 TTT000TTTTTT          9     9
##                                                                 varPvalue
## 9 4.2e-03/1.8e-03/3.0e-02/2.1e-02/3.3e-03/8.0e-03/5.9e-03/2.7e-05/2.5e-02
##   fishers.Info  IS.null   IS.alt d.score
## 9     40.91931 50.58165 18.70044  31.881
```

IV.V Get probability of node subset.

```
# This is the lower bound of the probability associated with the metabolites in S_arg.
# The higher the probability relative to a random set of the same size, the more
# tightly connected the metabolite set is.
2^-res[ind.mx,"IS.alt"]
```

```
## [1] 2.347506e-06
```

```
# Note the probability printed above may seem low, but there are log2(length(G), kmx) outcomes that pro
# We should expect a probability for a node set of size kmx in a length(G) network to have probability:
2^-(log2(choose(length(G), kmx)))
```

```
## [1] 3.715903e-23
```

```
# You'll notice the probability associated with the metabolite set we encoded, S_arg, is orders of magn
# probability model. This implies the metabolites in S_arg are  connected in the network ig_arg more th
```

IV.V Get p-value of variable length encoding vs. fixed length encoding.

```
# You can interpret the probability assigned to this metabolite set by comparing it to a null encoding
# uses fixed-length codes for all metabolites in the set. The "d.score" is the difference in bitlength
# encoding models. Using the Algorithmic Significance theorem, we can estimate the upper bounds on a p-
2^-res[ind.mx,"d.score"]
```

```
## [1] 2.528499e-10
```

```
# All metabolites in S_arg
names(S_arg)
```

```
##  [1] "x - 11381"
##  [2] "2-aminobutyrate"
##  [3] "dihydroorotate"
##  [4] "2-arachidonoyl-gpc (20:4)"
##  [5] "2-linoleoylglycerol (2-monolinolein)"
##  [6] "1-docosahexaenoyl-gpc (22:6)"
##  [7] "x - 12792"
##  [8] "gamma-glutamyl-2-aminobutyrate"
##  [9] "1-arachidonoyl-gpc (20:4)"
## [10] "x - 17001"
## [11] "prolylhydroxyproline"
## [12] "x - 11442"
## [13] "1-stearoyl-gpc (18:0)"
## [14] "x - 12798"
## [15] "glucose"
```

```
# Which metabolites were a part of the 8 metabolite subset of patient IEM_1006's top 15 perturbed metab
ptBSbyK[[ind.mx]] # all metabolites in the bitstring
```

```
## 2-linoleoylglycerol (2-monolinolein)                 dihydroorotate
##                                    1                              1
##                              glucose                       x - 16036
##                                    1                              0
##                           x - 11793                  bilirubin (e,e)
##                                    0                              0
```

```
##                            x - 11442           2-arachidonoyl-gpc (20:4)
##                                    1                                   1
##           1-arachidonoyl-gpc (20:4)        1-docosahexaenoyl-gpc (22:6)
##                                    1                                   1
##                            x - 11381                           x - 12798
##                                    1                                   1
```

```
names(which(ptBSbyK[[ind.mx]]==1)) # just the F metabolites that are in S_arg that were were "found"
```

```
## [1] "2-linoleoylglycerol (2-monolinolein)"
## [2] "dihydroorotate"
## [3] "glucose"
## [4] "x - 11442"
## [5] "2-arachidonoyl-gpc (20:4)"
## [6] "1-arachidonoyl-gpc (20:4)"
## [7] "1-docosahexaenoyl-gpc (22:6)"
## [8] "x - 11381"
## [9] "x - 12798"
```

V. Patient Distances, Single-Node Encoding Recommended

```
data_mx = arg_data[,which(colnames(arg_data) %in% names(diags))]
data_mx = data_mx[,1:8]
S_arg = c()
for (pt in 1:ncol(data_mx)) {
  ptID=colnames(data_mx)[pt]
  S_arg = c(S_arg, names(sort(abs(data_mx[,pt]), decreasing=TRUE)[1:kmx]))
}
S_arg = unique(S_arg)
# Pre-computing node ranks from all perturbed metabolites across all patients is the overhead we have t
# It feels like a lot of overhead when run serially, but when run in parallel (recommended) (e.g., a co
ranks = list()
for (n in 1:length(S_arg)) {
  print(sprintf("Calculating node ranks for perturbed metabolite %d/%d across %d patients.", n, length(S
  ind = which(names(G)==S_arg[n])
  ranks[[n]] = singleNode.getNodeRanksN(ind, G, p1=1.0, thresholdDiff=0.01, adj_mat, S = S_arg,
                                        num.misses = log2(length(G)), TRUE)
}
```

```
## [1] "Calculating node ranks for perturbed metabolite 1/88 across 8 patients."
## [1] "Calculating node rankings 208 of 208."
## [1] "Calculating node ranks for perturbed metabolite 2/88 across 8 patients."
## [1] "Calculating node rankings 207 of 208."
## [1] "Calculating node ranks for perturbed metabolite 3/88 across 8 patients."
## [1] "Calculating node rankings 206 of 208."
## [1] "Calculating node ranks for perturbed metabolite 4/88 across 8 patients."
## [1] "Calculating node rankings 205 of 208."
## [1] "Calculating node ranks for perturbed metabolite 5/88 across 8 patients."
## [1] "Calculating node rankings 204 of 208."
## [1] "Calculating node ranks for perturbed metabolite 6/88 across 8 patients."
## [1] "Calculating node rankings 203 of 208."
## [1] "Calculating node ranks for perturbed metabolite 7/88 across 8 patients."
## [1] "Calculating node rankings 202 of 208."
```

```
## [1] "Calculating node ranks for perturbed metabolite 8/88 across 8 patients."
## [1] "Calculating node rankings 201 of 208."
## [1] "Calculating node ranks for perturbed metabolite 9/88 across 8 patients."
## [1] "Calculating node rankings 200 of 208."
## [1] "Calculating node ranks for perturbed metabolite 10/88 across 8 patients."
## [1] "Calculating node rankings 199 of 208."
## [1] "Calculating node ranks for perturbed metabolite 11/88 across 8 patients."
## [1] "Calculating node rankings 198 of 208."
## [1] "Calculating node ranks for perturbed metabolite 12/88 across 8 patients."
## [1] "Calculating node rankings 1 of 208."
## [1] "Calculating node ranks for perturbed metabolite 13/88 across 8 patients."
## [1] "Calculating node rankings 197 of 208."
## [1] "Calculating node ranks for perturbed metabolite 14/88 across 8 patients."
## [1] "Calculating node rankings 196 of 208."
## [1] "Calculating node ranks for perturbed metabolite 15/88 across 8 patients."
## [1] "Calculating node rankings 195 of 208."
## [1] "Calculating node ranks for perturbed metabolite 16/88 across 8 patients."
## [1] "Calculating node rankings 150 of 208."
## [1] "Calculating node ranks for perturbed metabolite 17/88 across 8 patients."
## [1] "Calculating node rankings 110 of 208."
## [1] "Calculating node ranks for perturbed metabolite 18/88 across 8 patients."
## [1] "Calculating node rankings 180 of 208."
## [1] "Calculating node ranks for perturbed metabolite 19/88 across 8 patients."
## [1] "Calculating node rankings 113 of 208."
## [1] "Calculating node ranks for perturbed metabolite 20/88 across 8 patients."
## [1] "Calculating node rankings 53 of 208."
## [1] "Calculating node ranks for perturbed metabolite 21/88 across 8 patients."
## [1] "Calculating node rankings 40 of 208."
## [1] "Calculating node ranks for perturbed metabolite 22/88 across 8 patients."
## [1] "Calculating node rankings 128 of 208."
## [1] "Calculating node ranks for perturbed metabolite 23/88 across 8 patients."
## [1] "Calculating node rankings 166 of 208."
## [1] "Calculating node ranks for perturbed metabolite 24/88 across 8 patients."
## [1] "Calculating node rankings 151 of 208."
## [1] "Calculating node ranks for perturbed metabolite 25/88 across 8 patients."
## [1] "Calculating node rankings 9 of 208."
## [1] "Calculating node ranks for perturbed metabolite 26/88 across 8 patients."
## [1] "Calculating node rankings 179 of 208."
## [1] "Calculating node ranks for perturbed metabolite 27/88 across 8 patients."
## [1] "Calculating node rankings 158 of 208."
## [1] "Calculating node ranks for perturbed metabolite 28/88 across 8 patients."
## [1] "Calculating node rankings 14 of 208."
## [1] "Calculating node ranks for perturbed metabolite 29/88 across 8 patients."
## [1] "Calculating node rankings 186 of 208."
## [1] "Calculating node ranks for perturbed metabolite 30/88 across 8 patients."
## [1] "Calculating node rankings 65 of 208."
## [1] "Calculating node ranks for perturbed metabolite 31/88 across 8 patients."
## [1] "Calculating node rankings 165 of 208."
## [1] "Calculating node ranks for perturbed metabolite 32/88 across 8 patients."
## [1] "Calculating node rankings 188 of 208."
## [1] "Calculating node ranks for perturbed metabolite 33/88 across 8 patients."
## [1] "Calculating node rankings 125 of 208."
## [1] "Calculating node ranks for perturbed metabolite 34/88 across 8 patients."
## [1] "Calculating node rankings 143 of 208."
```

```
## [1] "Calculating node ranks for perturbed metabolite 35/88 across 8 patients."
## [1] "Calculating node rankings 94 of 208."
## [1] "Calculating node ranks for perturbed metabolite 36/88 across 8 patients."
## [1] "Calculating node rankings 33 of 208."
## [1] "Calculating node ranks for perturbed metabolite 37/88 across 8 patients."
## [1] "Calculating node rankings 193 of 208."
## [1] "Calculating node ranks for perturbed metabolite 38/88 across 8 patients."
## [1] "Calculating node rankings 59 of 208."
## [1] "Calculating node ranks for perturbed metabolite 39/88 across 8 patients."
## [1] "Calculating node rankings 37 of 208."
## [1] "Calculating node ranks for perturbed metabolite 40/88 across 8 patients."
## [1] "Calculating node rankings 170 of 208."
## [1] "Calculating node ranks for perturbed metabolite 41/88 across 8 patients."
## [1] "Calculating node rankings 172 of 208."
## [1] "Calculating node ranks for perturbed metabolite 42/88 across 8 patients."
## [1] "Calculating node rankings 107 of 208."
## [1] "Calculating node ranks for perturbed metabolite 43/88 across 8 patients."
## [1] "Calculating node rankings 191 of 208."
## [1] "Calculating node ranks for perturbed metabolite 44/88 across 8 patients."
## [1] "Calculating node rankings 185 of 208."
## [1] "Calculating node ranks for perturbed metabolite 45/88 across 8 patients."
## [1] "Calculating node rankings 176 of 208."
## [1] "Calculating node ranks for perturbed metabolite 46/88 across 8 patients."
## [1] "Calculating node rankings 182 of 208."
## [1] "Calculating node ranks for perturbed metabolite 47/88 across 8 patients."
## [1] "Calculating node rankings 187 of 208."
## [1] "Calculating node ranks for perturbed metabolite 48/88 across 8 patients."
## [1] "Calculating node rankings 174 of 208."
## [1] "Calculating node ranks for perturbed metabolite 49/88 across 8 patients."
## [1] "Calculating node rankings 10 of 208."
## [1] "Calculating node ranks for perturbed metabolite 50/88 across 8 patients."
## [1] "Calculating node rankings 177 of 208."
## [1] "Calculating node ranks for perturbed metabolite 51/88 across 8 patients."
## [1] "Calculating node rankings 76 of 208."
## [1] "Calculating node ranks for perturbed metabolite 52/88 across 8 patients."
## [1] "Calculating node rankings 111 of 208."
## [1] "Calculating node ranks for perturbed metabolite 53/88 across 8 patients."
## [1] "Calculating node rankings 194 of 208."
## [1] "Calculating node ranks for perturbed metabolite 54/88 across 8 patients."
## [1] "Calculating node rankings 85 of 208."
## [1] "Calculating node ranks for perturbed metabolite 55/88 across 8 patients."
## [1] "Calculating node rankings 144 of 208."
## [1] "Calculating node ranks for perturbed metabolite 56/88 across 8 patients."
## [1] "Calculating node rankings 8 of 208."
## [1] "Calculating node ranks for perturbed metabolite 57/88 across 8 patients."
## [1] "Calculating node rankings 192 of 208."
## [1] "Calculating node ranks for perturbed metabolite 58/88 across 8 patients."
## [1] "Calculating node rankings 41 of 208."
## [1] "Calculating node ranks for perturbed metabolite 59/88 across 8 patients."
## [1] "Calculating node rankings 131 of 208."
## [1] "Calculating node ranks for perturbed metabolite 60/88 across 8 patients."
## [1] "Calculating node rankings 68 of 208."
## [1] "Calculating node ranks for perturbed metabolite 61/88 across 8 patients."
## [1] "Calculating node rankings 190 of 208."
```

```
## [1] "Calculating node ranks for perturbed metabolite 62/88 across 8 patients."
## [1] "Calculating node rankings 81 of 208."
## [1] "Calculating node ranks for perturbed metabolite 63/88 across 8 patients."
## [1] "Calculating node rankings 17 of 208."
## [1] "Calculating node ranks for perturbed metabolite 64/88 across 8 patients."
## [1] "Calculating node rankings 168 of 208."
## [1] "Calculating node ranks for perturbed metabolite 65/88 across 8 patients."
## [1] "Calculating node rankings 100 of 208."
## [1] "Calculating node ranks for perturbed metabolite 66/88 across 8 patients."
## [1] "Calculating node rankings 34 of 208."
## [1] "Calculating node ranks for perturbed metabolite 67/88 across 8 patients."
## [1] "Calculating node rankings 54 of 208."
## [1] "Calculating node ranks for perturbed metabolite 68/88 across 8 patients."
## [1] "Calculating node rankings 24 of 208."
## [1] "Calculating node ranks for perturbed metabolite 69/88 across 8 patients."
## [1] "Calculating node rankings 106 of 208."
## [1] "Calculating node ranks for perturbed metabolite 70/88 across 8 patients."
## [1] "Calculating node rankings 6 of 208."
## [1] "Calculating node ranks for perturbed metabolite 71/88 across 8 patients."
## [1] "Calculating node rankings 51 of 208."
## [1] "Calculating node ranks for perturbed metabolite 72/88 across 8 patients."
## [1] "Calculating node rankings 20 of 208."
## [1] "Calculating node ranks for perturbed metabolite 73/88 across 8 patients."
## [1] "Calculating node rankings 45 of 208."
## [1] "Calculating node ranks for perturbed metabolite 74/88 across 8 patients."
## [1] "Calculating node rankings 184 of 208."
## [1] "Calculating node ranks for perturbed metabolite 75/88 across 8 patients."
## [1] "Calculating node rankings 136 of 208."
## [1] "Calculating node ranks for perturbed metabolite 76/88 across 8 patients."
## [1] "Calculating node rankings 12 of 208."
## [1] "Calculating node ranks for perturbed metabolite 77/88 across 8 patients."
## [1] "Calculating node rankings 4 of 208."
## [1] "Calculating node ranks for perturbed metabolite 78/88 across 8 patients."
## [1] "Calculating node rankings 83 of 208."
## [1] "Calculating node ranks for perturbed metabolite 79/88 across 8 patients."
## [1] "Calculating node rankings 157 of 208."
## [1] "Calculating node ranks for perturbed metabolite 80/88 across 8 patients."
## [1] "Calculating node rankings 104 of 208."
## [1] "Calculating node ranks for perturbed metabolite 81/88 across 8 patients."
## [1] "Calculating node rankings 71 of 208."
## [1] "Calculating node ranks for perturbed metabolite 82/88 across 8 patients."
## [1] "Calculating node rankings 114 of 208."
## [1] "Calculating node ranks for perturbed metabolite 83/88 across 8 patients."
## [1] "Calculating node rankings 69 of 208."
## [1] "Calculating node ranks for perturbed metabolite 84/88 across 8 patients."
## [1] "Calculating node rankings 181 of 208."
## [1] "Calculating node ranks for perturbed metabolite 85/88 across 8 patients."
## [1] "Calculating node rankings 178 of 208."
## [1] "Calculating node ranks for perturbed metabolite 86/88 across 8 patients."
## [1] "Calculating node rankings 61 of 208."
## [1] "Calculating node ranks for perturbed metabolite 87/88 across 8 patients."
## [1] "Calculating node rankings 160 of 208."
## [1] "Calculating node ranks for perturbed metabolite 88/88 across 8 patients."
## [1] "Calculating node rankings 97 of 208."
```

```r
names(ranks) = S_arg
# Calculate patient bitstrings
ptBSbyK = list()
for (pt in 1:ncol(data_mx)) {
  ptID=colnames(data_mx)[pt]
  S_pt = names(sort(abs(data_mx[,pt]), decreasing=TRUE)[1:kmx])
  ptBSbyK[[ptID]] = singleNode.getPtBSbyK(S_pt, ranks)
}
# Now perform mutual information-based patient similarity scoring
res = list()
t = list(ncd=matrix(NA, nrow=ncol(data_mx), ncol=ncol(data_mx)))
rownames(t$ncd) = colnames(data_mx)
colnames(t$ncd) = colnames(data_mx)
for (i in 1:kmx) { res[[i]] = t }
for (pt in 1:ncol(data_mx)) {
  print(sprintf("Patient %d vs...", pt))
  ptID=colnames(data_mx)[pt]
  for (pt2 in pt:ncol(data_mx)) {
    print(sprintf("Patient %d.", pt2))
    ptID2=colnames(data_mx)[pt2]
    # Because we pre-computed node ranks for all perturbed metabolites across our 8 patients, this will
    tmp = mle.getPtDist(ptBSbyK[[ptID]], ptID, ptBSbyK[[ptID2]], ptID2, data_mx, ranks, p1=1.0, thresho
    for (k in 1:kmx) {
      res[[k]]$ncd[ptID, ptID2] = tmp$NCD[k]
      res[[k]]$ncd[ptID2, ptID] = tmp$NCD[k]
    }
  }
}
```

```
## [1] "Patient 1 vs..."
## [1] "Patient 1."
## [1] "Patient 2."
## [1] "Patient 3."
## [1] "Patient 4."
## [1] "Patient 5."
## [1] "Patient 6."
## [1] "Patient 7."
## [1] "Patient 8."
## [1] "Patient 2 vs..."
## [1] "Patient 2."
## [1] "Patient 3."
## [1] "Patient 4."
## [1] "Patient 5."
## [1] "Patient 6."
## [1] "Patient 7."
## [1] "Patient 8."
## [1] "Patient 3 vs..."
## [1] "Patient 3."
## [1] "Patient 4."
## [1] "Patient 5."
## [1] "Patient 6."
## [1] "Patient 7."
## [1] "Patient 8."
## [1] "Patient 4 vs..."
```

```
## [1] "Patient 4."
## [1] "Patient 5."
## [1] "Patient 6."
## [1] "Patient 7."
## [1] "Patient 8."
## [1] "Patient 5 vs..."
## [1] "Patient 5."
## [1] "Patient 6."
## [1] "Patient 7."
## [1] "Patient 8."
## [1] "Patient 6 vs..."
## [1] "Patient 6."
## [1] "Patient 7."
## [1] "Patient 8."
## [1] "Patient 7 vs..."
## [1] "Patient 7."
## [1] "Patient 8."
## [1] "Patient 8 vs..."
## [1] "Patient 8."
```

VI. Visualizations

```
# Multi-dimensional scaling
plot.mdsDist = function(patientDist, diagnoses, k, diag) {
  if (!(k %in% c(2,3))) {
    print("K must be either 2-dimensions or 3-dimensions.")
    return(0)
  }
  if (is.null(diagnoses)) {
    print("To view patient clusters, please provide clinical labels, corresponding to each patient in t
    return(0)
  }
  fitDist = cmdscale(patientDist, eig=FALSE, k=k)
  x = round(fitDist[,1], 2)
  y = round(fitDist[,2], 2)
  if (k==3) {
    z = round(fitDist[,3], 2)
    df = data.frame(x=x, y=y, z=z, color=diagnoses, label=colnames(patientDist))
    p = plot_ly(df, x=~x, y=~y, z=~z, color=~color, text=~label, marker = list(size = 20))
  } else {
    df = data.frame(x=x, y=y, color=diagnoses, label=colnames(patientDist))
    p = plot_ly(df, x=~x, y=~y, color=~color, text=~label, marker = list(size = 20))
  }
  return(p)
}
# Hierarchical clustering
plot.hmDist = function(patientDist, path, diagnoses=NULL) {
  if (is.null(diagnoses)) {
    png(sprintf("%s/ptDistances_hc.png", path), 3000, 1000)
    heatmap.2(x=patientDist,
              dendrogram="both",
              Rowv=TRUE,
```

```
              Colv=TRUE,
              cexRow=0.75,cexCol=1, margins=c(12,12),
              trace="none", key=TRUE,
              col=bluered,
              notecol="black")
    dev.off()
  } else {
    d = diagnoses
    png(sprintf("%s/ptDistances_hc.png", path), 3000, 1000)
    heatmap.2(x=patientDist,
              dendrogram="both",
              Rowv=TRUE,
              Colv=TRUE,
              ColSideColors = c(brewer.pal(12, "Set3"), brewer.pal(9, "BrBG"))[as.numeric(as.factor(as.
              RowSideColors = c(brewer.pal(12, "Set3"), brewer.pal(9, "BrBG"))[as.numeric(as.factor(as.
              cexRow=0.75,cexCol=1, margins=c(12,12),
              trace="none", key=TRUE,
              col=bluered,
              notecol="black")
    legend("left", legend=unique(sort(as.character(d))), fill=c(brewer.pal(12, "Set3"), brewer.pal(9, "
    dev.off()
  }
  return(0)
}
# K-nearest neighbors
plot.knnDist = function(patientDist, diagnoses, k, diag, path) {
  diagnoses = diagnoses[colnames(patientDist)]
  # Add a GREEN edge between patient nodes if k nearest neighbor is correct diagnosis (either TP or TN)
  # Add a RED edge between patient nodes if k nearest neighbor is incorrect diagnosis (either FP or FN)
  tp = 0
  fp = 0
  tn = 0
  fn = 0
  ig = make_empty_graph(n=ncol(patientDist), directed=TRUE)
  V(ig)$name = colnames(patientDist)
  for (pt1 in 1:length(diagnoses)) {
    diag_pt1 = diagnoses[pt1]
    ind = sort(patientDist[pt1,-pt1], decreasing = FALSE)
    ind = ind[which(ind==min(ind))]
    diag_pt_ind = diagnoses[which(names(diagnoses) %in% names(ind))]
    if (any(diag_pt_ind==diag) && diag_pt1==diag) {
      # True positive
      tp = tp + 1
      ind = ind[which(diag_pt_ind==diag)]
      ig = add.edges(ig, edges=c(colnames(patientDist)[pt1], names(ind[1])), attr=list(color="green", l
    } else if (diag_pt_ind!=diag && diag_pt1!=diag) {
      # True negative
      tn = tn + 1
      ig = add.edges(ig, edges=c(colnames(patientDist)[pt1], names(ind[1])), attr=list(color="green", l
    } else if (diag_pt_ind==diag && diag_pt1!=diag) {
      # Mistake!
      fp = fp + 1
      ig = add.edges(ig, edges=c(colnames(patientDist)[pt1], names(ind[1])), attr=list(color="red", lty
```

```
    } else {
      fn = fn + 1
      ig = add.edges(ig, edges=c(colnames(patientDist)[pt1], names(ind[1])), attr=list(color="red", lty
    }
  }
  print(sprintf("Tp = %d, Tn= %d, Fp = %d, Fn=%d", tp, tn, fp, fn))
  sens = tp / (tp+fn)
  spec = tn / (tn+fp)
  print(sprintf("Sens = %.2f, Spec= %.2f", sens, spec))

  V(ig)$label = rep("", length(V(ig)$name))
  grps = list()
  grps[[1]] = names(diagnoses)[which(diagnoses==diag)]
  grps[[2]] = names(diagnoses)[which(diagnoses!=diag)]
  names(grps) = names(table(diagnoses))
  png(sprintf("%s/ptDistances_knn.png", path), 3000, 1000)
  plot(ig, mark.groups = grps, mark.col = c("white", "black"), layout=layout.circle, edge.width=3, edge
  dev.off()

  return(0)
}

# if you have diagnostic labels associated with the colnames(data_mx), send them using diagnoses parame
res_ncd = lapply(res, function(i) i$ncd)
ncd = mle.getMinPtDistance(res_ncd)
dd = colnames(data_mx)
dd[which(dd %in% names(diags)[which(diags=="Argininemia")])] = "ARG"
dd[which(dd %in% names(diags)[which(diags!="Argininemia")])] = "negCntl"
names(dd) = colnames(res[[1]]$ncd)

require(plotly)
```

```
## Loading required package: plotly
## Loading required package: ggplot2
##
## Attaching package: 'plotly'
## The following object is masked from 'package:ggplot2':
##
##     last_plot
## The following object is masked from 'package:igraph':
##
##     groups
## The following object is masked from 'package:stats':
##
##     filter
## The following object is masked from 'package:graphics':
##
##     layout
```

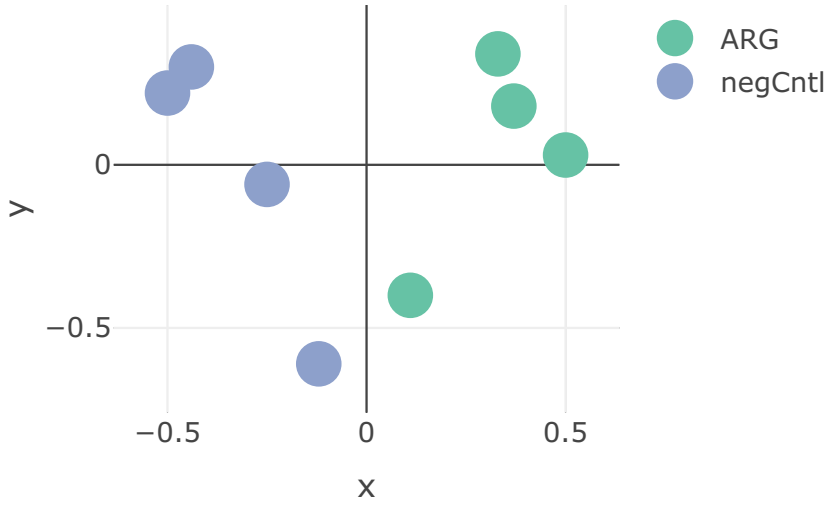
```
require(gplots)
```

```
## Loading required package: gplots
##
## Attaching package: 'gplots'
## The following object is masked from 'package:stats':
##
##     lowess
require(RColorBrewer)
```

```
## Loading required package: RColorBrewer
```

```
colnames(ncd) = colnames(res[[1]]$ncd)
rownames(ncd) = colnames(res[[1]]$ncd)
p = plot.mdsDist(ncd, dd, k=2, NULL)
p
```

```
## No trace type specified:
##   Based on info supplied, a 'scatter' trace seems appropriate.
##   Read more about this trace type -> https://plot.ly/r/reference/#scatter
## No scatter mode specifed:
##   Setting the mode to markers
##   Read more about this attribute -> https://plot.ly/r/reference/#scatter-mode
## Warning: `arrange_()` is deprecated as of dplyr 0.7.0.
## Please use `arrange()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
## Warning in RColorBrewer::brewer.pal(N, "Set2"): minimal value for n is 3, returning requested palett
## Warning in RColorBrewer::brewer.pal(N, "Set2"): minimal value for n is 3, returning requested palett
```




```
# Hierarchical clustering
# A PNG image called "ptDistances.png" will save in the path set as parameter "path" in plot.hmDist fun
plot.hmDist(ncd, path=getwd(), dd)
```

```
## [1] 0
```

```
# K-NN
plot.knnDist(ncd, dd, diag="ARG", path=getwd())
```

```
## [1] "Tp = 4, Tn= 2, Fp = 2, Fn=0"
## [1] "Sens = 1.00, Spec= 0.50"
```

```
## [1] 0
```