Ratio Cut Examples

Ratio cut and spectral clustering

Given a graph of similarities $G(V, E)$ represented by weight matrix W and $L = D - W$ where D is the degree matrix of W, ratio cut partitions the graph into k clusters by minimizing

$$\mathrm{Tr}(H^T L H)$$

... where $H = [h_{ij}]$ is defined as (per Luxburg[1]):

$$h_{ij} = \begin{cases} 1/\sqrt{|C_i|} & \text{if } i \in C_j \\ 0 & \text{otherwise} \end{cases}$$

Then $H \in \mathbb{R}^{n \times k}$ where n is the number of vertices in G.

This problem is NP-hard, so a relaxation of this problem embeds L using its k smallest eigenvectors, in order to match the dimensionality of H, and performs k-means clustering on the embedding. Here we exclude the eigenvector corresponding to the zero eigenvalue (if G is fully connected, L has one zero eigenvalue and its corresponding eigenvector has entries $1/\sqrt{n}$).

However, there is no theoretical basis for this method.

Some observations:

- Performing k-means clustering on the k largest eigenvectors of $L^\dagger$ is equivalent to k-means clustering on the k smallest eigenvectors of L ("smallest" or "largest" corresponding to the sizes of the eigenvalues), since they share the same eigenvectors. However, maximizing $\mathrm{Tr}(H^T L^\dagger H)$ and minimizing $\mathrm{Tr}(H^T L H)$ are not equivalent in the general case.[2]
- If we construct H^* from the first k eigenvalues of L, including the one that corresponds to the zero eigenvalue, then by the Rayleigh-Ritz theorem, this H^* minimizes $\mathrm{Tr}(H^T L H)$. On the other hand, if we exclude the eigenvector that corresponds to the zero eigenvalue to construct H^*, then there are cases where H of our restricted form results in a smaller value of $\mathrm{Tr}(H^T L H)$ than H^*.
- Constructing H^* using the first k eigenvectors of L and finding H that minimizes $||H - H^*||_F^2$ is not equivalent to minimizing $\mathrm{Tr}(H^T L H)$, whether or not we use the eigenvector corresponding to the zero eigenvalue.

Next, we propose a new method of approximating ratio cut. Instead of approximating the optimal H with the first k eigenvectors of L, we fully embed L and perform k-means clustering.

Examples

In this section, we will look at several example graphs and apply both the spectral clustering method outlined by Luxburg and our proposed method. In these examples, the "natural clusters" are known *a priori*, and so when performing k-means, which is usually initialized randomly and can converge at a local optimum, we will initialize the algorithm with the known "natural clusters".

[1] U. Luxburg
[2] M. W. Trosset, pg 110

```r
# packages, etc.
import::from(magrittr, `%>%`, `%<>%`)
library(ggplot2)
import::from(psych, tr)
import::from(qgraph, qgraph)
source('http://pages.iu.edu/~mtrosset/Courses/675/manifold.r')
theme_set(theme_bw())
import::from(GGally, ggpairs)

# --- functions --- #

ratio.cut.obj <- function(L, clustering, cluster.max = max(clustering)) {
  # compute the objective for ratio cut
  H <- construct.H(clustering, cluster.max)
  psych::tr(t(H) %*% L %*% H)
}

#' @title construct H function
#' @description H is constructed based on a vector that designates the clusters
#' @param clustering (numeric) A vector of cluster assignments
#' @return (matrix) H based on the cluster assignments
construct.H <- function(clustering, cluster.max = max(clustering)) {
  clusters <- seq(max(cluster.max))

  if (min(clustering) < 1) {
    stop(simpleError('cluster indexing starts at 1'))
  }

  # find |A_k|
  cluster.sizes <- sapply(clusters, function(i) {
    length(clustering[clustering == i])
  })

  # construct H
  H <- sapply(clustering, function(i) {
    h <- rep(0, length(clusters))
    h[i] <- 1 / sqrt(cluster.sizes[i])
    return(h)
  }) %>%
    t()

  return(H)
}

#' @title k-means clustering with an initial clustering
#' @param X (numeric) A data matrix
#' @param clust.init (numeric) A vector of cluster assignments
#' @return The object returned by stats::kmeans
kmeans.initialized <- function(X, clust.init) {
  # dimensionality checks
  if (is.vector(X)) {
    assertthat::assert_that(length(X) == length(clust.init))
  } else {
```

```
    assertthat::assert_that(nrow(X) == length(clust.init))
  }
  assertthat::assert_that(min(clust.init) == 1)
  assertthat::assert_that(length(unique(clust.init)) == max(clust.init))

  # find initial centers
  if (!is.vector(X)) {
    centers.init <- sapply(unique(clust.init), function(i) {
      clust.ind <- which(clust.init == i)
      apply(X[clust.ind, ], 2, mean)
    }) %>%
      t()
  } else {
    centers.init <- sapply(unique(clust.init), function(i) {
      clust.ind <- which(clust.init == i)
      mean(X[clust.ind])
    })
  }

  # apply stats::kmeans
  kmeans(X, centers.init)
}
```

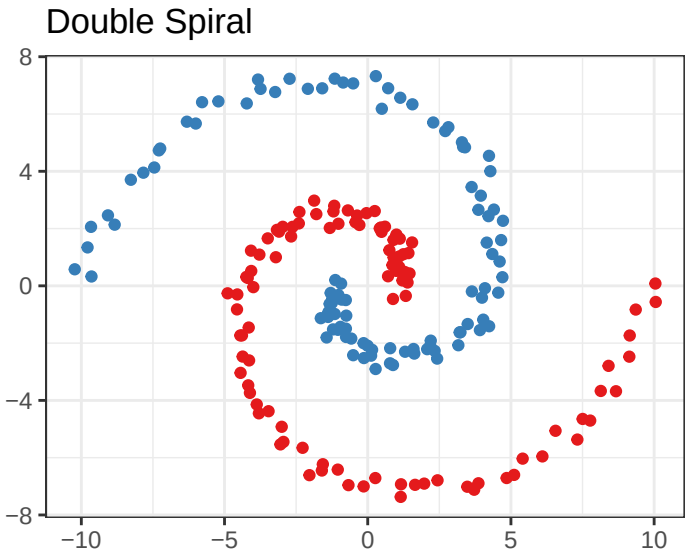
Double spiral

Although this example comes from constructing a double spiral in $\mathbb{R}^2$ and then constructing a k-nearest neighbors graph based on Euclidean distance, the main object of interest is the graph, not the spiral. We are primarily concerned with partitioning a graph rather than finding alternative methods of clustering data in Euclidean space.[3]

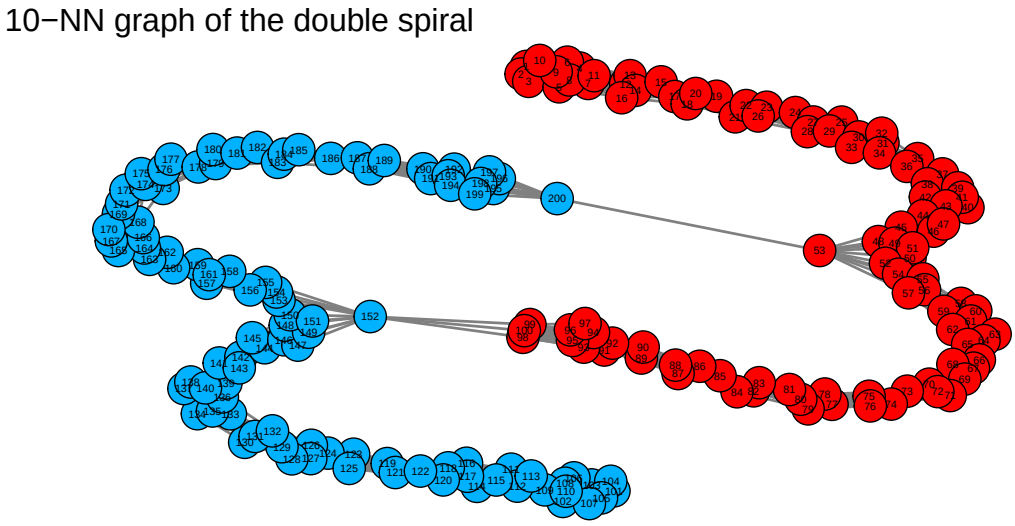
```
# parameters
set.seed(112358)
eps <- 2 ** -2
K <- 10  # for constructing the knn graph
rad.max <- 10
ang.max <- 2 * pi
angles <- seq(0, ang.max, length.out = 100)
radii <- seq(1, sqrt(rad.max), length.out = 100) ** 2

# data
spiral.df <- dplyr::data_frame(X = radii * cos(angles),
                               Y = radii * sin(angles))
spiral.df <- dplyr::data_frame(X = radii * cos(angles),
                               Y = radii * sin(angles))
neg.spiral.df <- dplyr::mutate(spiral.df,
                               X = -X, Y = -Y,
                               id = '2')
spiral.df %<>%
```

[3] If we perform the Spectral Clustering method as outlined by Ng, Jordan, and Weiss on the Euclidean data, we get two clusters that corresponds to each spiral. However, this method is rather particular about how the graph is constructed. If instead, we apply this method on the 10-nearest neighbors graph, we end up with an "unnatural" clustering that neither optimizes the ratio cut metric nor produces clusters that correspond to each spiral. This method is less about producing "good" partitions of a graph and more about clustering data in Euclidean space.

```
  dplyr::mutate(id = '1') %>%
  dplyr::bind_rows(neg.spiral.df) %>%
  dplyr::mutate(X = X + rnorm(n = n(), sd = eps),
                Y = Y + rnorm(n = n(), sd = eps))

# plot the double spiral
ggplot(spiral.df) +
  coord_fixed() +
  scale_colour_brewer(palette = 'Set1') +
  geom_point(aes(x = X, y = Y, colour = id)) +
  labs(title = 'Double Spiral',
       x = NULL, y = NULL) +
  guides(colour = FALSE)
```




```
# construct similarity graph
W <- spiral.df %>%
  dplyr::select(X, Y) %>%
  as.matrix() %>%
  mds.edm1() %>%
  graph.knn(K) %>%
  graph.adj()

# plot the graph (this is what we're interested in, not the spiral)
qgraph(W, groups = spiral.df$id,
       title = '10-NN graph of the double spiral',
       node.width = 2,
       legend = FALSE)
```



Ignoring the double spiral and looking only at the 10-nearest neighbors graph, we can see that the original group memberships do indeed seem appropriate when partitioning this graph. In order to verify that the ratio cut metric agrees with this partitioning, we would have to try all possible partitions, which is not feasible. But as a reference, we can compute the ratio cut metric for this partitioning and then compare it against the results of our various methods.

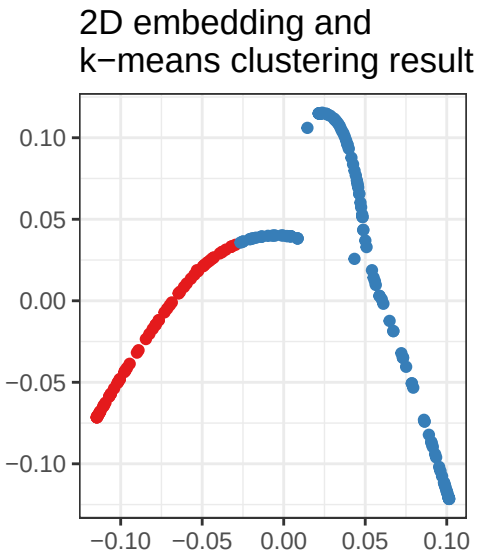
```
# graph laplacian of W
L <- graph.laplacian(W)

# start with clusters that correspond to each spiral
init.clust <- c(rep(1, 100), rep(2, 100))

# compute the ratio cut metric for this clustering
ratio.cut.obj(L, init.clust)
```

```
[1] 0.08
```

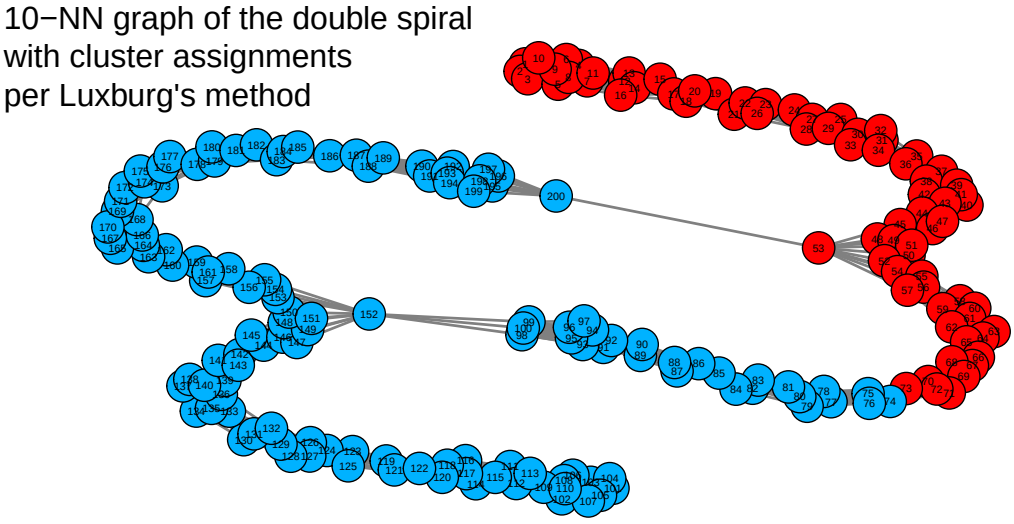
When we use the method outlined by Luxburg (excluding the "zeroth" eigenvector):

```
# decompose L
L.eigen <- eigen(L)

# compute "H-approx" from the first k eigenvectors of L
k <- 2
n <- nrow(spiral.df)
H.approx <- L.eigen$vectors[, seq(n - 1, n - k)] %>%
  as.data.frame()

# k-means clustering
kmeans.embed <- kmeans.initialized(H.approx, init.clust)

# visualize the embedding and clustering
ggplot(H.approx) +
  geom_point(aes(x = V1, y = V2, colour = factor(kmeans.embed$cluster))) +
  coord_fixed() +
  labs(x = NULL, y = NULL,
       title = '2D embedding and\nk-means clustering result') +
```

```
  scale_colour_brewer(palette = 'Set1') +
  guides(colour = FALSE)
```




```
qgraph(W, groups = factor(kmeans.embed$cluster),
       title = paste('10-NN graph of the double spiral',
                     'with cluster assignments',
                     'per Luxburg\'s method',
                     sep = '\n'),
       node.width = 2,
       legend = FALSE)
```




```
# ratio cut metric
ratio.cut.obj(L, kmeans.embed$cluster)
```

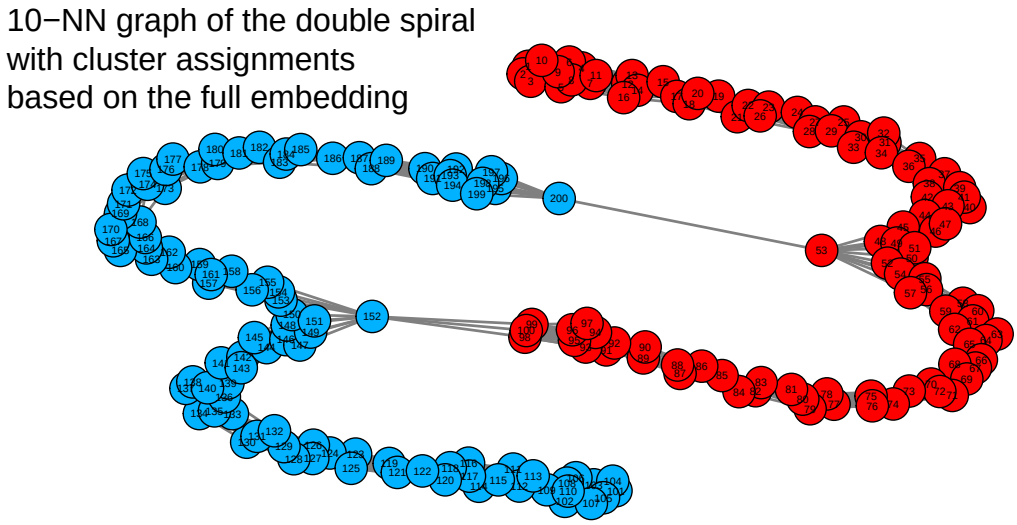
```
[1] 0.3451623
```

We can see that while the embedding does produce something that appear to be two clusters, k-means is not

adequate for this type of data (as seen both visually and by computing the ratio cut metric).

If we fully embed the graph, scale the embedding, and perform k-means clustering on that:

```
# full embeding
full.embed.df <- L.eigen$vectors[, seq(n - 1, 1)] %>%
  apply(1, function(x) x / sqrt(L.eigen$values[seq(n - 1, 1)])) %>%
  t() %>%
  as.data.frame()

# k-means clustering
kmeans.embed <- kmeans.initialized(full.embed.df, init.clust)

qgraph(W, groups = factor(kmeans.embed$cluster),
       title = paste('10-NN graph of the double spiral',
                     'with cluster assignments',
                     'based on the full embedding',
                     sep = '\n'),
       node.width = 2,
       legend = FALSE)
```




```
# ratio cut metric
ratio.cut.obj(L, kmeans.embed$cluster)
```

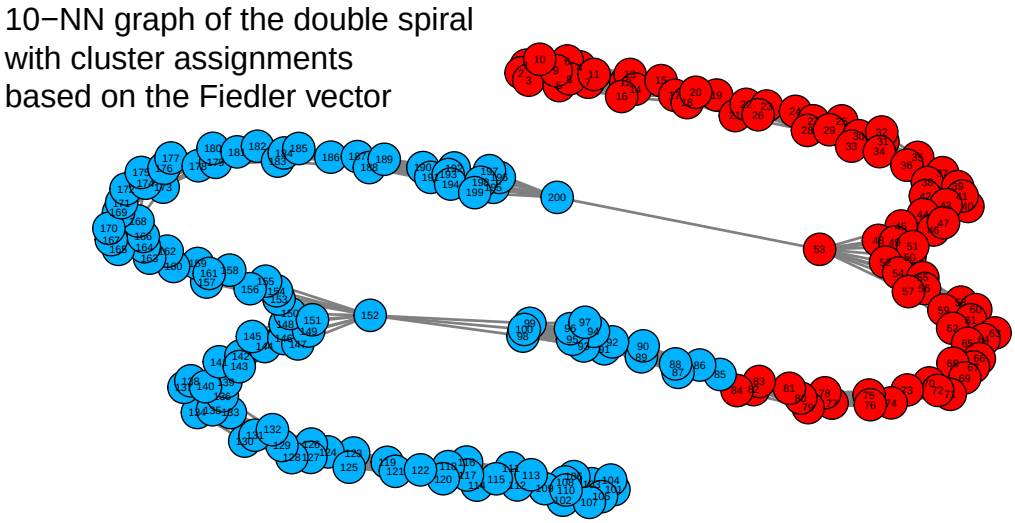
```
[1] 0.08
```

We arrive at something that agrees completely with the *a priori* cluster assignments.

One complication that we observed with the full embedding method is that as we increase the number of dimensions, the less likely it is for k-means to arrive at its optimal solution with random initial cluster assignments (which is why we were careful about initialization).

One thing that was not clear in Luxburg's tutorial is whether we use the "zeroth" eigenvector or if we start at the Fiedler eigenvector when constructing H^* from the eigenvectors of L. If we start at the zeroth eigenvector, then that can be ignored since all of the entries are $1/\sqrt{n}$, and we arrive at an effectively $k-1$ dimensional embedding. In this case, if we perform k-means clustering, we still arrive at a worse clustering according to the ratio cut metric.

```r
# extract the fiedler vector and perform k-means
fiedler.vec <- L.eigen$vectors[, n - 1]
fiedler.clust <- kmeans.initialized(fiedler.vec, init.clust)

qgraph(W, groups = factor(fiedler.clust$cluster),
       title = paste('10-NN graph of the double spiral',
                     'with cluster assignments',
                     'based on the Fiedler vector',
                     sep = '\n'),
       node.width = 2,
       legend = FALSE)
```



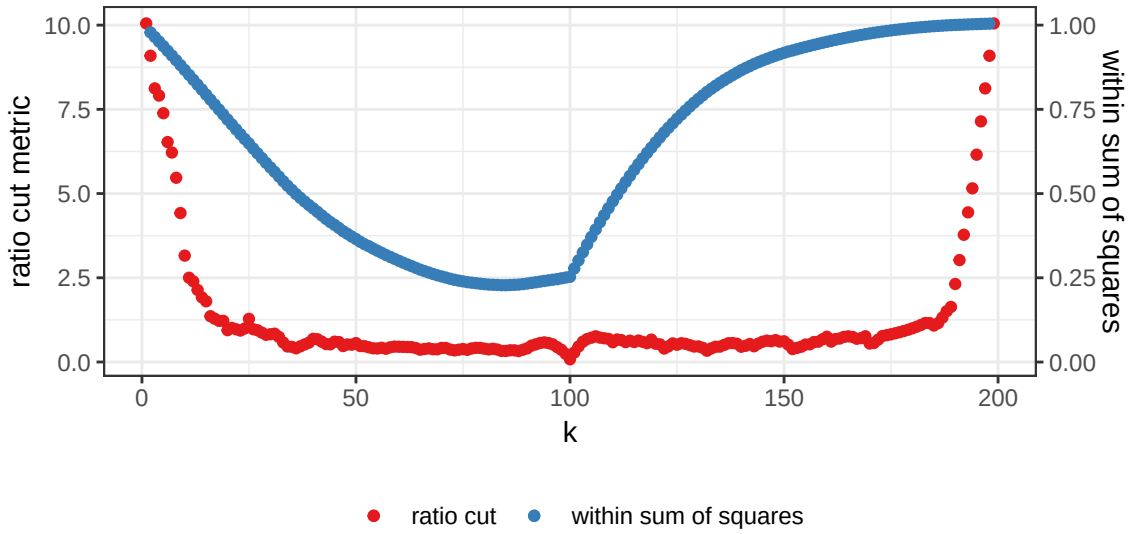

```r
ratio.cut.obj(L, fiedler.clust$cluster)
```

```
[1] 0.3284072
```

On the other hand, if we have a one-dimensional embedding, that makes checking all partitionings very easy, since now we only have to check $n-1$ of them. If we compute the ratio cut metric for each of the $n-1$ partitionings, we find that the one that minimizes the ratio cut metric corresponds to the one that splits the objects based on which spiral it is in.

```r
# compute ratio cut metric and kmeans metric for all n - 1 possible partitions
one.dim.embed.df <- lapply(seq(n - 1), function(i) {
  clustering <- c(rep(1, i), rep(2, n - i))
  ratio.cut <- ratio.cut.obj(L, clustering)
  k.means <- i * var(fiedler.vec[clustering == 1]) +
    (n - i) * var(fiedler.vec[clustering == 2])
  dplyr::data_frame(k = i,
                    ratio.cut = ratio.cut,
                    kmeans.wss = k.means)
}) %>%
  dplyr::bind_rows()

ggplot(one.dim.embed.df) +
  geom_point(aes(x = k, y = ratio.cut, colour = 'ratio cut')) +
  geom_point(aes(x = k, y = kmeans.wss * 10, colour = 'within sum of squares')) +
```

```
  labs(colour = NULL, y = 'ratio cut metric') +
  scale_y_continuous(sec.axis = sec_axis(~ . / 10,
                                         name = 'within sum of squares')) +
  scale_colour_brewer(palette = 'Set1') +
  theme(legend.position = 'bottom')
```



Big 5 dataset

The data here is a collection of 240 personality quiz questions answered by 500 students, found in the `qgraph`[4] package. Each question corresponds to one of five personality types, and the students respond to each question on a scale from 1 to 5 depending on how much they agree with the statment. The similarity graph G with weight matrix $W = [w_{ij}]$ is then constructed as follows:

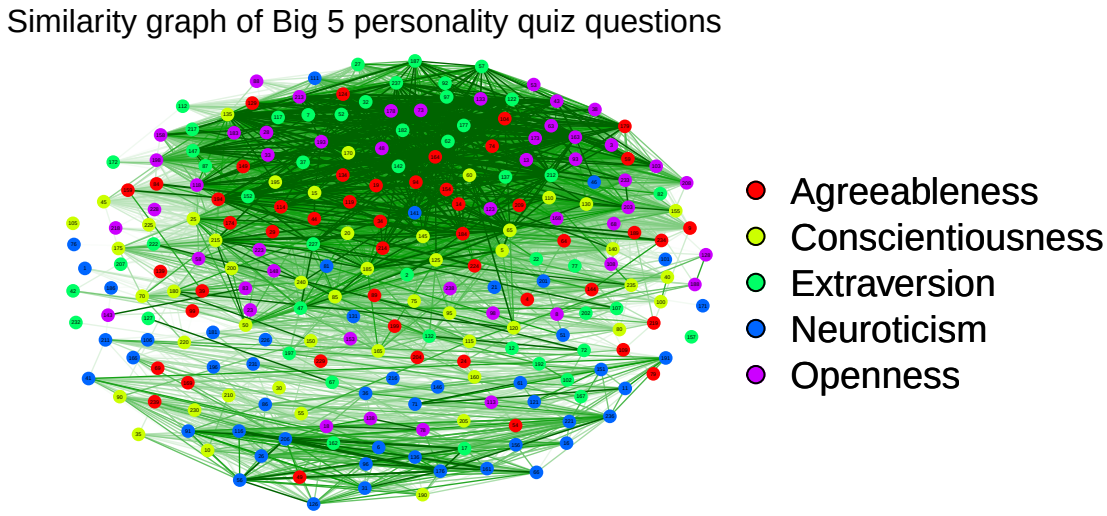
$$w_{ij} = \frac{1}{4 \times 500} \sum_{i=1}^{500} \left(4 - (x_{ik} - x_{jk})\right)$$

```
data(big5, package = 'qgraph')
data(big5groups, package = 'qgraph')

n <- ncol(big5)  # number of vertices
k <- length(big5groups)  # number of groups/clusters
group.names <- names(big5groups)

# rearrange the groups into a vector
group.assign <- rep(NA, n)
for (i in seq(n)) {
  for (j in group.names) {
    if (i %in% big5groups[[j]]) {
      group.assign[i] <- j
    }
  }
```

[4] CRAN - Package qgraph

```
}

# convert cluster assignments to 1, 2, 3, ...
init.clust <- as.numeric(as.factor(group.assign))

# construct weight matrix
W <- matrix(rep(NA, n ** 2), nrow = n)
for (i in seq(n)) {
  for (j in seq(n)) {
    W[i, j] <- 1 - sum(abs(big5[, i] - big5[, j])) / 4 / nrow(big5)
  }
}

# construct graph laplacian
L <- graph.laplacian(W)
```

```
qgraph(W, groups = group.assign,
       borders = FALSE,
       layout = 'spring',
       minimum = quantile(W, .75),
       title = 'Similarity graph of Big 5 personality quiz questions')
```



We can start by computing the ratio cut metric for the *a priori* grouping:

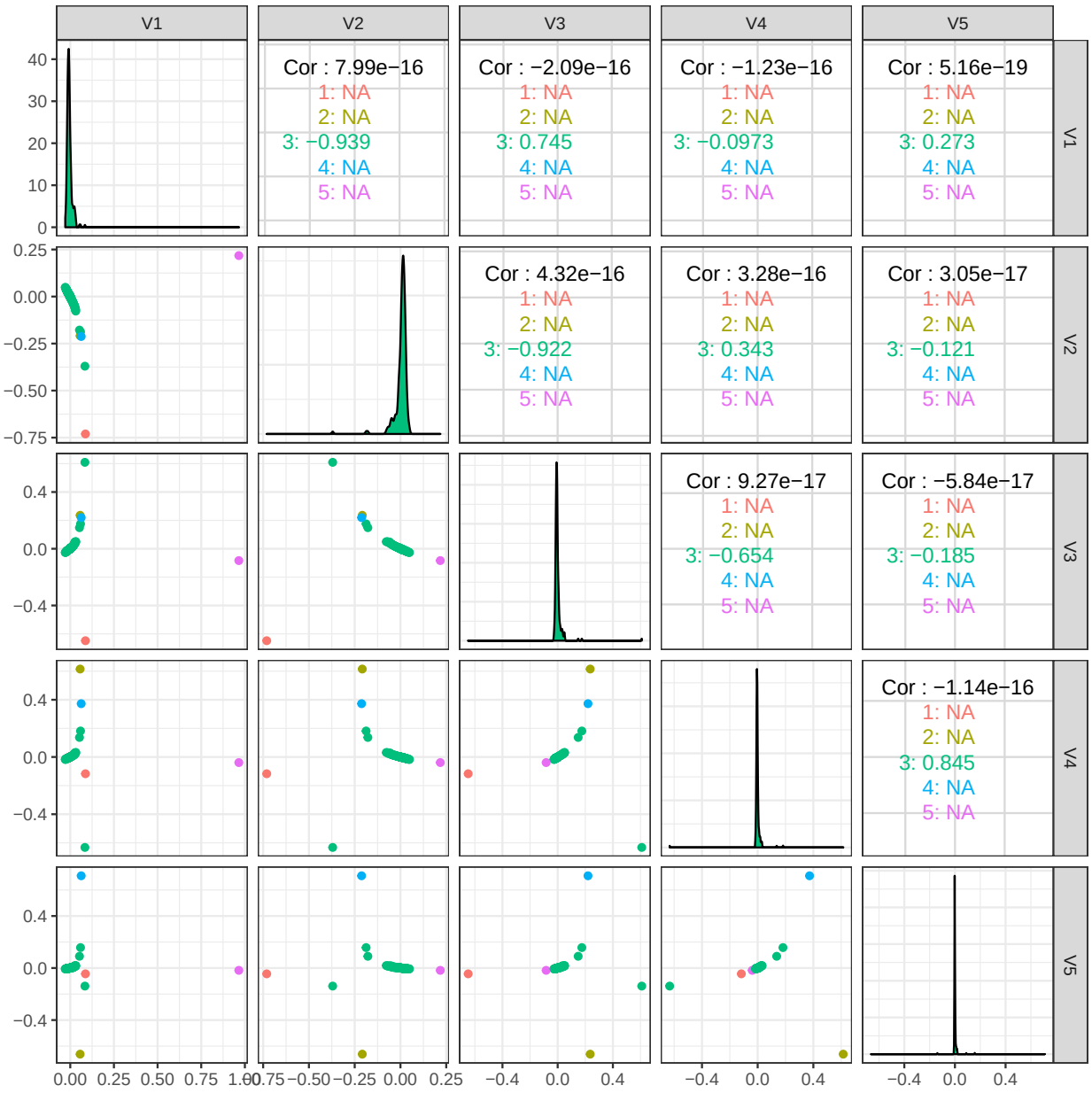
```
ratio.cut.obj(L, init.clust)
```

```
[1] 673.6646
```

Then performing spectral clustering per Luxburg's tutorial:

```
# decompose L
L.eigen <- eigen(L)

# first k eigenvectors
H.approx <- L.eigen$vectors[, seq(n - 1, n - k)] %>%
  as.data.frame()
```

```
# k-means clustering
kmeans.embed <- kmeans.initialized(H.approx, init.clust)

# matrix plot of the k-dimensional embedding
ggpairs(H.approx, mapping = aes(colour = factor(kmeans.embed$cluster)))
```



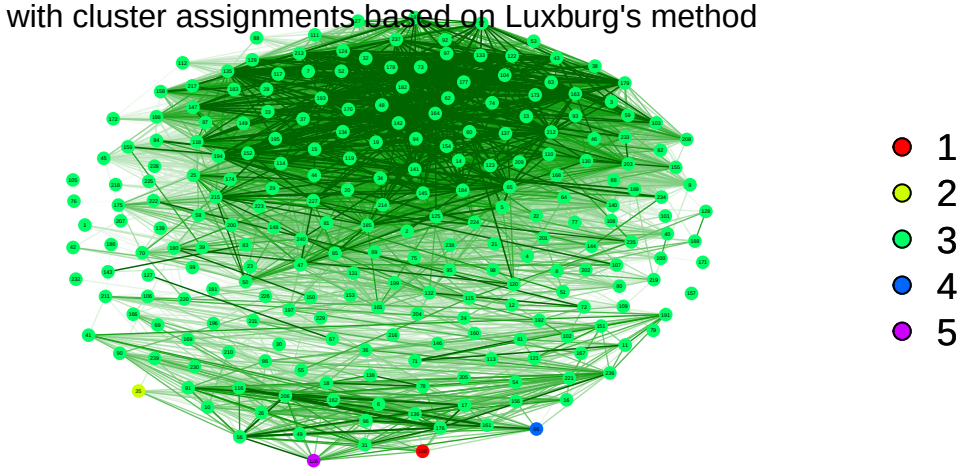

```
# how well did it do according to ratio cut
ratio.cut.obj(L, kmeans.embed$cluster)
```

```
[1] 587.3375
```

```
# does it agree with the a priori groups
table(init.clust, kmeans.embed$cluster)
```

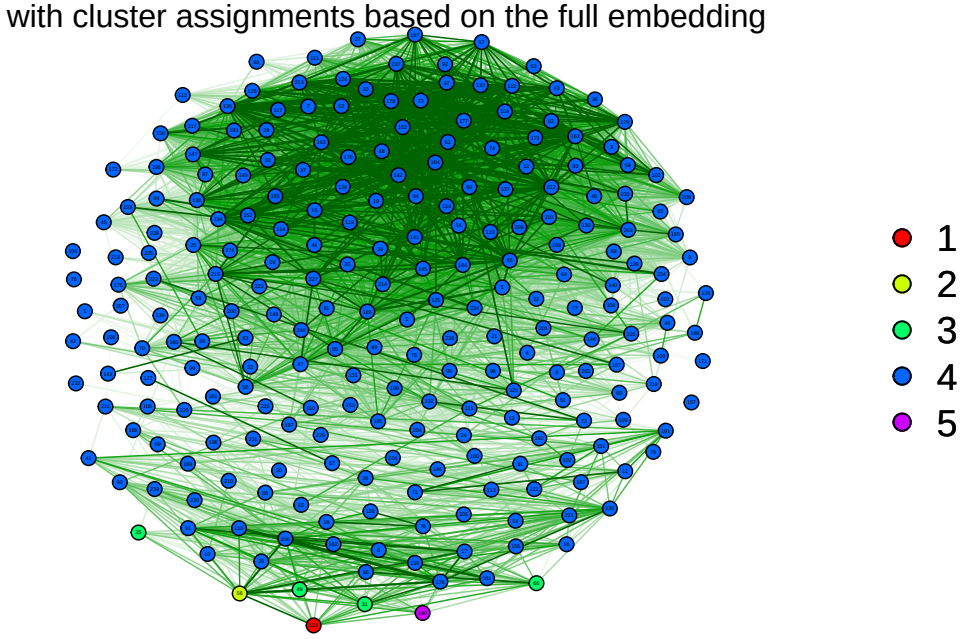
```
init.clust  1  2  3  4  5
         1  0  0 48  0  0
         2  1  1 46  0  0
         3  0  0 48  0  0
         4  0  0 46  1  1
         5  0  0 48  0  0
```

```
qgraph(W, groups = factor(kmeans.embed$cluster),
       borders = FALSE,
       layout = 'spring',
       minimum = quantile(W, .75),
       title = paste('Similarity graph of Big 5 personality quiz questions',
                     'with cluster assignments based on Luxburg\'s method',
                     sep = '\n'))
```



And performing k-means on the full scaled embedding, although to a lesser degree (relatively).

```
# full embeding
full.embed.df <- L.eigen$vectors[, seq(n - 1, 1)] %>%
  apply(1, function(x) x / sqrt(L.eigen$values[seq(n - 1, 1)])) %>%
  t() %>%
  as.data.frame()

# k-means clustering
kmeans.embed <- kmeans.initialized(full.embed.df, init.clust)

qgraph(W, groups = factor(kmeans.embed$cluster),
       title = paste('Similarity graph of Big 5 personality quiz questions',
                     'with cluster assignments based on the full embedding',
                     sep = '\n'),
       layout = 'spring',
       minimum = quantile(W, .75))
```




```
# ratio cut metric
ratio.cut.obj(L, kmeans.embed$cluster)
```

```
[1] 586.3422
```

```
# does it agree with the a priori groups
table(init.clust, kmeans.embed$cluster)
```

```
init.clust  1  2  3  4  5
         1  0  0  1 47  0
         2  0  0  1 46  1
         3  0  0  0 48  0
         4  1  1  2 44  0
         5  0  0  0 48  0
```

And again, k-means on the scaled full embedding performs better than k-means on the embedding in $\mathbb{R}^k$. Unfortunately in this case it is difficult to determine what a "good" clustering is visually.

Simple graph with 4 vertices

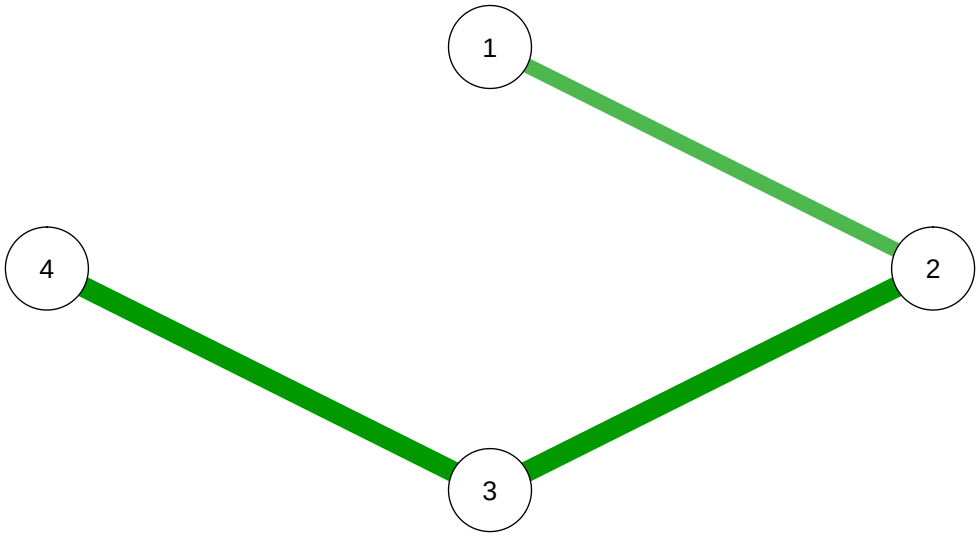
This example was adapted from Proximity in Statistical Machine Learning.[5]

Here, we construct a similarity graph with 4 vertices and 3 edges. There is an edge between vertices 1 and 2, an edge between vertices 2 and 3, and an edge between vertices 3 and 4. All edges have weight 1 except for the one between vertices 1 and 2, which we will set to 0.7.

```
# construct weight matrix
eps <- .7
n <- 4
```

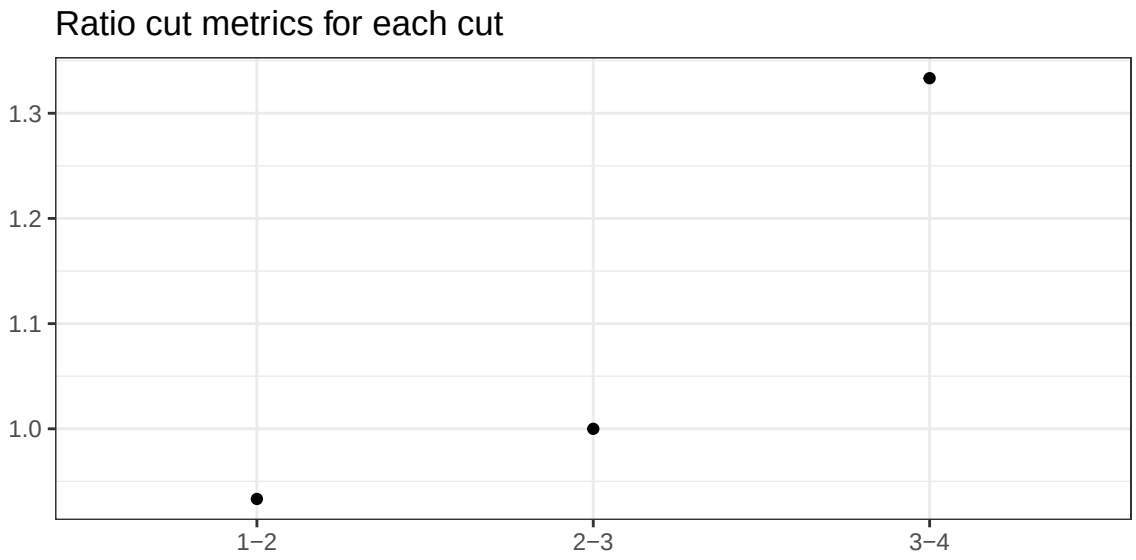
[5] M. W. Trosset, pg 110

```
W <- matrix(rep(0, n ** 2), nrow = n)
for (i in seq(2, n - 1)) W[i, i + 1] <- W[i + 1, i] <- 1
W[1, 2] <- W[2, 1] <- eps

# construct graph laplacian
L <- graph.laplacian(W)

qgraph(W)
```



We want to separate this graph into two clusters. If we view this as a graph cut problem, we can see that there are only 3 possible cuts, and we can compute the ratio cut metric for each cut.

```
ratio.cut.metrics <- sapply(seq(n - 1), function(i) {
  clustering <- c(rep(1, i), rep(2, n - i))
  ratio.cut.obj(L, clustering)
})

ggplot() +
  geom_point(aes(x = c('1-2', '2-3', '3-4'),
                 y = ratio.cut.metrics)) +
  labs(x = NULL, y = NULL, title = 'Ratio cut metrics for each cut')
```



And we can see that the solution to ratio cut is the $(1)(2,3,4)$ clustering.

Then we can see what spectral clustering produces. First, Luxburg's method:

```
L.eigen <- eigen(L)
k <- 2
H.approx <- L.eigen$vectors[, seq(n - 1, n - k)] %>%
  as.data.frame()
kmeans.initialized(H.approx, c(1, 2, 2, 2))
```

```
K-means clustering with 2 clusters of sizes 2, 2

Cluster means:
           V1        V2
1  0.05450564  0.488992
2 -0.05450564 -0.488992

Clustering vector:
[1] 1 2 2 1

Within cluster sum of squares by cluster:
[1] 0.8793452 0.1523187
 (between_SS / total_SS =  48.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"
[5] "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
```

Then the full embedding method:

```
full.embed <- L.eigen$vectors[, seq(n - 1, 1)] %>%
  apply(1, function(x) x / sqrt(L.eigen$values[seq(n - 1, 1)])) %>%
  t() %>%
  as.data.frame()
kmeans.initialized(full.embed, c(1, 2, 2, 2))
```

```
K-means clustering with 2 clusters of sizes 2, 2

Cluster means:
          V1          V2         V3
1  0.6200444 -0.06735501 -0.1131104
2 -0.6200444  0.06735501  0.1131104

Clustering vector:
[1] 1 1 2 2

Within cluster sum of squares by cluster:
[1] 0.7142857 0.5000000
 (between_SS / total_SS =  57.0 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"
[5] "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
```

Here, neither embedding provides the ratio cut solution. Even more strangely, Luxburg's method results in a clustering of $(1, 4)(2, 3)$. If we try computing the ratio cut metric for this clustering, we get a value of 1.7. This happens to be the worst possible clustering for ratio cut.