Social Network Analysis

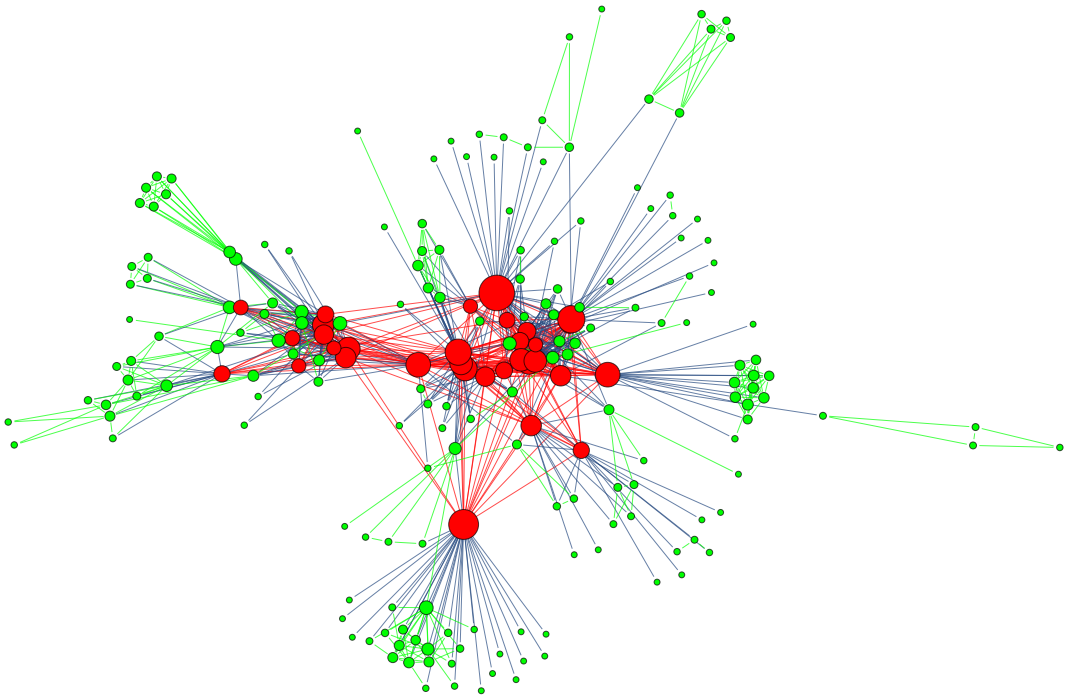
Chen Avin
School of Electrical and Computer Engineering

Unit 7
Community Detection

(Based on Networks: An Introduction. By M.E.J Newman)

Last Time

Core-Periphery



Group of Vertices

- Cliques
- k-plex - A k-plex is a maximal set of vertices such that each vertex is adjacent to all except k others.
- k-clique - A k-clique is a maximal set of vertices that are at a distance no greater than k from each other (using outside).
- k-club - A k-club is a maximal set of vertices where the diameter of the corresponding subgraph is at most k (path is inside the group)

Finally... We get to use our Facebook class networks

In[940]:=

```
SetDirectory[NotebookDirectory[]];
file = Rest[Import["data/stormofswords.csv"]];
tribes = Import["data/tribes.csv"];
nodes = Flatten[tribes[[All, 1]]];
edges = #[[1]] ⟷ #[[2]] & /@ file [[All, {1, 2}]];
ThronesG = Graph[nodes, edges, VertexLabels → Placed["Name", Tooltip]]
```

Out[945]=

k - plex

- k-plex - A k-plex is a maximal set of vertices such that each vertex is adjacent to all except k others.

In[958]:=

```
FindKPlex[ThronesG, 1]
```

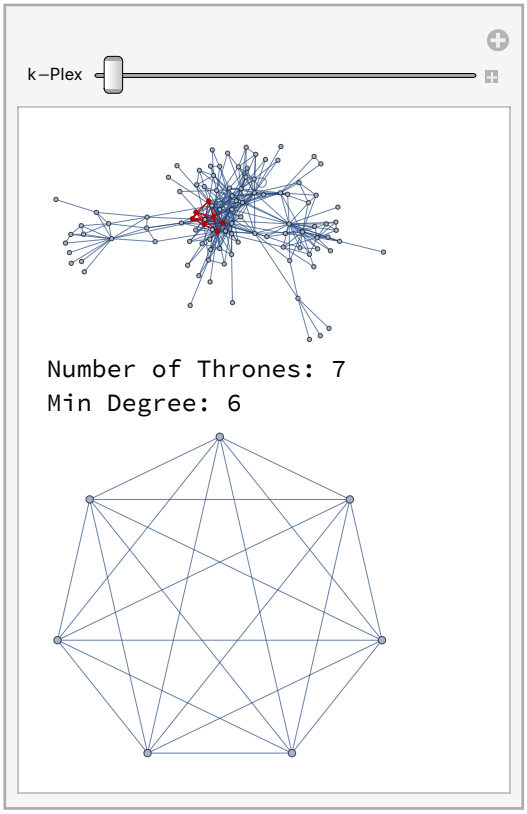
Out[958]=

```
{{Cersei, Gregor, Ilyn, Joffrey, Meryn, Sandor, Tyrion}}
```

In[959]:=

```
Manipulate[
 Column[{HighlightGraph[ThronesG, SG = Subgraph[ThronesG, FindKPlex[ThronesG, i],
      VertexLabels → Placed["Name", Tooltip]]],
   "Number of Thrones: " <> ToString[ VertexCount[SG]],
   "Min Degree: " <> ToString[ Min[VertexDegree[SG]]], SG}],
 {{i, 1, "k-Plex"}, 1, 20, 1}]
```

Out[959]=

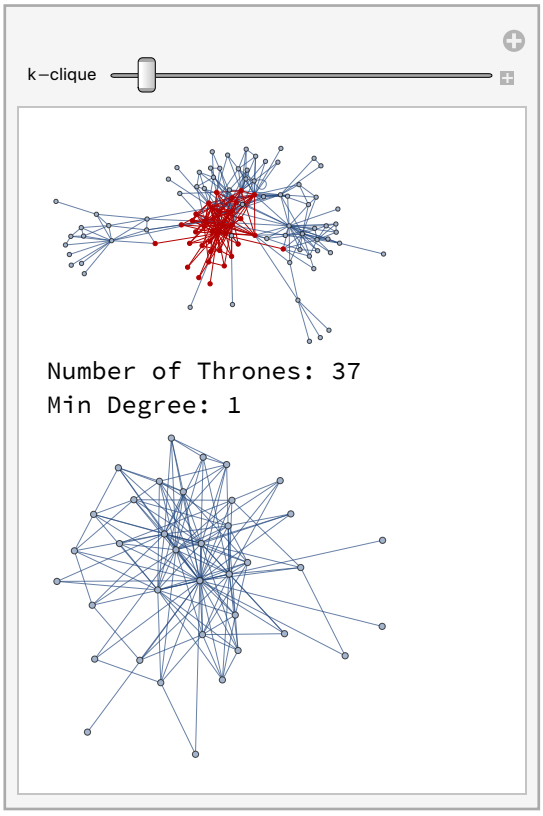


k - clique

- k-clique - A k-clique is a maximal set of vertices that are at a distance no greater than k from each other (using out side).

In[960]:=
```
Manipulate[Column[
  {HighlightGraph[ThronesG, SG = Subgraph[ThronesG, FindKClique[ThronesG, i],
      VertexLabels → Placed["Name", Tooltip]]],
   "Number of Thrones: " <> ToString[ VertexCount[SG]],
   "Min Degree: " <> ToString[ Min[VertexDegree[SG]]], SG}],
 {{i, 1, "k-clique"}, 1, 20, 1}]
```

Out[960]=



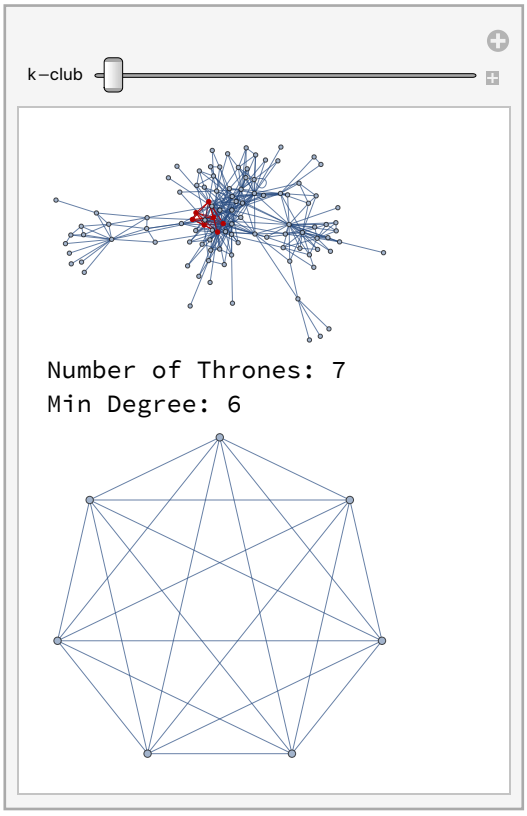
k - club

- k-club - A k-club is a maximal set of vertices where the diameter of the corresponding subgraph is at most k (path is inside the group)

In[961]:=

```
Manipulate[
 Column[{HighlightGraph[ThronesG, SG = Subgraph[ThronesG, FindKClub[ThronesG, i],
       VertexLabels → Placed["Name", Tooltip]]],
   "Number of Thrones: " <> ToString[ VertexCount[SG]],
   "Min Degree: " <> ToString[ Min[VertexDegree[SG]]], SG}],
 {{i, 1, "k-club"}, 1, 20, 1}]
```

Out[961]=



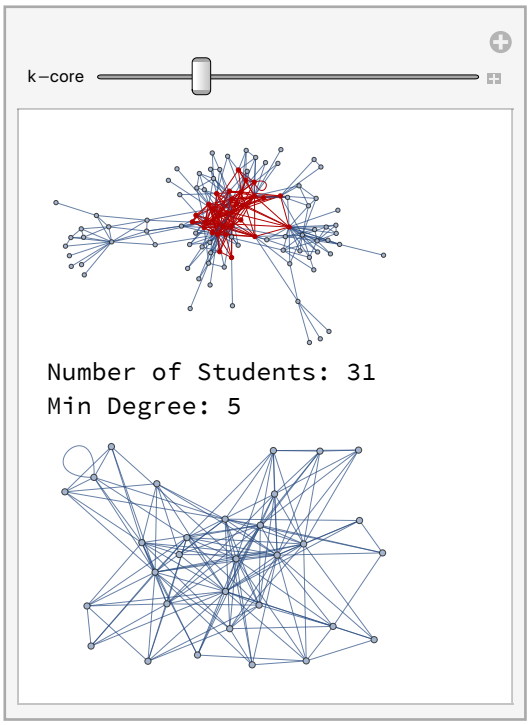
k-core

- k-core - A k-core component is a maximal weakly connected subgraph in which all vertices have degree at least k
- Easy to find, just remove node iteratively

In[962]:=

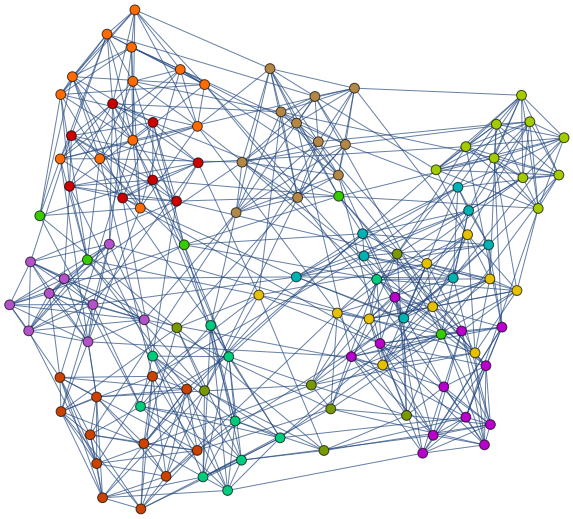
```
Manipulate[Column[{HighlightGraph[ThronesG,
    SG = Subgraph[ThronesG, KCoreComponents[ThronesG, i]]],
   "Number of Students: " <> ToString[ VertexCount[SG]],
   "Min Degree: " <> ToString[ Min[VertexDegree[SG]]], SG}],
 {{i, 0, "k-core"}, 0, 20, 1}]
```

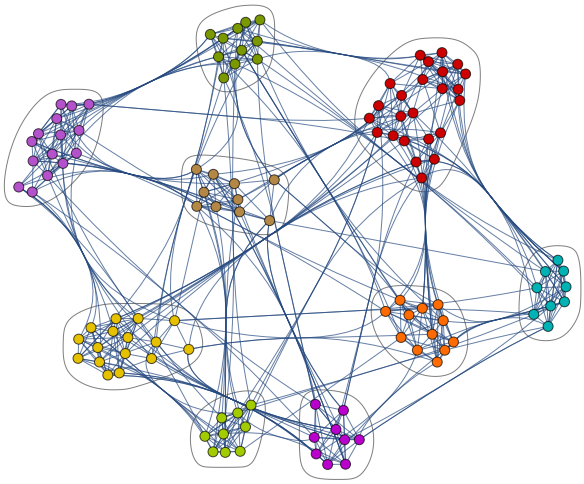
Out[962]=



Graph Partitioning and Community Detection

- Divide the network into groups, clusters or communities according to edges (graph topology)
- Graph Partitioning is a classic (hard) problem
- Both are important to understand the graph structure, processes on the graph (like diffusion) and for efficient parallel computation
- Basic guideline: minimize the edges between groups
- The difference between Graph Partitioning and Community Detection:
  - are the number and/or the size of the group is given?



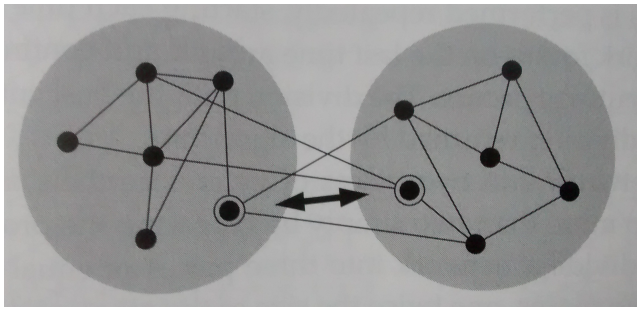


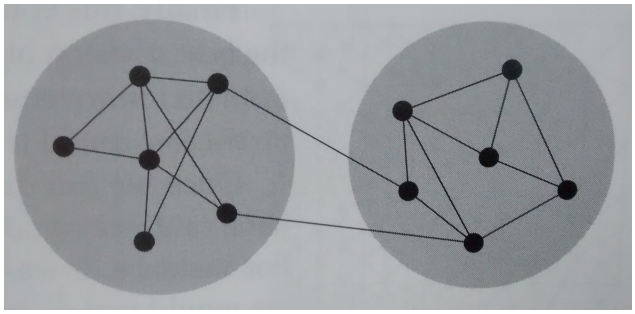
Graph Partitioning

- Consider the “simple” case - partition the graph into two equal groups with minimum cut
- Why is it hard?
- Known to be NP-hard, Also hard to approximate

Kernighan-Lin Algorithm

- Simple and one of the best known heuristic algorithm
- “Vertex Moving” approach
- Start from any partition (say, equal size partition)
- Find the best pair to “switch” (in hope to reduce the cut)
- Repeat of remaining nodes until no more nodes to “switch” (in this round)
- Pick the best partition you have found in the round and start a new round.
- Go until no improvements between rounds





- Problem: it is slow, $O(n^3)$ for a round
- Pseudocode

$D_a = E_a - I_a$ =

External cost of edge from a − Internal cost of edge from a

```
function Kernighan-Lin(G(V,E)):
    determine a balanced initial partition of the nodes into sets A and B

    do
       compute D values for all a in A and b in B
       let gv, av, and bv be empty lists
       for (n := 1 to |V|/2)
          find a from A and b from B, such that g = D[a] + D[b] - 2*c(a, b) is maximal
          remove a and b from further consideration in this pass
          add g to gv, a to av, and b to bv
          update D values for the elements of A = A \ a and B = B \ b
       end for
       find k which maximizes g_max, the sum of gv[1],...,gv[k]
       if (g_max > 0) then
          Exchange av[1],av[2],...,av[k] with bv[1],bv[2],...,bv[k]
    until (g_max <= 0)
return G(V,E)
```

Spectral Partitioning

- Consider Bisection (two equal parts)
- G(V,E), n=|V|, m=|E|, two groups: 1, 2
- The Number of edges between the two groups

$$(1)\quad R = \frac{1}{2} \sum_{\substack{i,j \text{ in} \\ \text{different} \\ \text{groups}}} A_{ij}$$

- Let s_i present the division of the network

$$(2)\quad s_i = \begin{cases} +1 & \text{if vertex } i \text{ belongs to group } 1 \\ -1 & \text{if vertex } i \text{ belongs to group } 2 \end{cases}$$

- Then

$$(3)\quad \frac{1}{2}\,(1 - s_i\, s_j) = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are in different groups} \\ 0 & \text{if } i \text{ and } j \text{ are in the same groups} \end{cases}$$

- So we can rewrite

$$(4)\quad R = \frac{1}{4} \sum_{i,j} A_{ij}\,(1 - s_i\, s_j)$$

- Notice that (where k_i is the degree of i, and δ is the Kronecker delta function)

$$(5)\quad \sum_{i,j} A_{ij} = \sum_{i} k_i = \sum_{i} k_i\, s_i^{\,2} = \sum_{i,j} k_i\, \delta_{ij}\, s_i\, s_j$$

- Putting back to (4) we get

$$(6)\quad R = \frac{1}{4} \sum_{i,j} (k_i\, \delta_{ij} - A_{ij})\, s_i\, s_j = \frac{1}{4} \sum_{i,j} L_{ij}\, s_i\, s_j$$

- **L** is the Laplacian matrix
- This is a fundamental matrix with many uses to study graphs

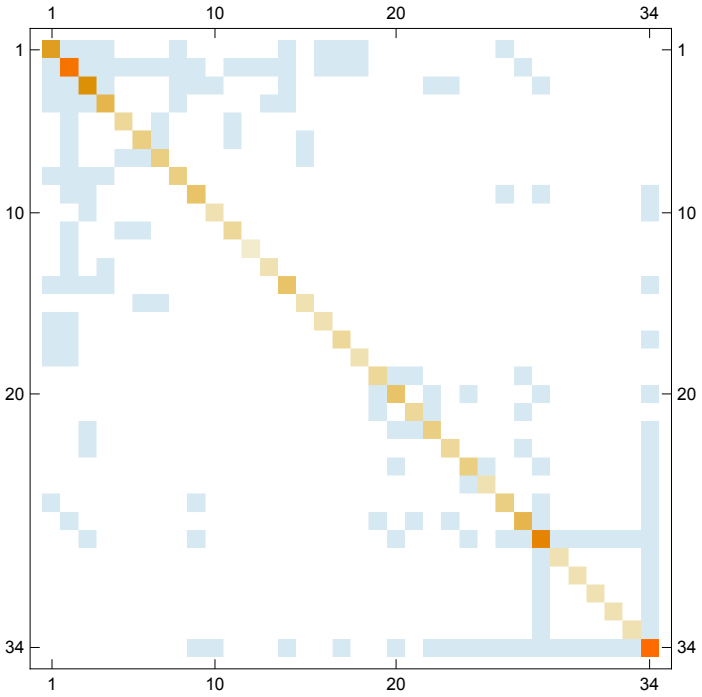
- Has the degree k_i on the diagonal and -1 for each edge (where A_{ij}=1). The sum of each row is then 0.

$$L_{ij} = k_i \, \delta_{ij} - A_{ij}$$

$$L_{ij} = \begin{cases} k_i & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and there is an edge } (i, j) \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{L} =$$

$$\mathbf{D} - \mathbf{A} \quad (\text{where D is diagonal matrix with degrees along diagonal})$$



- Note that

$$\mathbf{L} \cdot \mathbf{1} = 0$$

- So (1,1,1,1,.....) is an eigenvector of **L** and λ_1 =0 is an eigen value of **L** (and all $\lambda_i \geq 0$)
- So in a matrix form we can write (6)

$$(7) \quad R = \frac{1}{4} \mathbf{s}^T \mathbf{L} \mathbf{s}$$

Our goal is to Minimize R (by setting **s**)

$$(7)\quad R = \frac{1}{4}\,\mathbf{s}^T \mathbf{L}\mathbf{s}$$

- To minimize R we need to take a derivative, but the constrains on s (±1) make it hard to find
- This lead to the relaxation method: relax some of the constrains
- We can relax the ±1 constrain to be

$$(8)\quad \sum_i s_i^2 = n$$

- But we keep the constrain that we want two groups of sizes n_1, n_2

$$(9)\quad \sum_i s_i = n_1 - n_2$$

- In Vector notation

$$(10)\quad \mathbf{1}^T \mathbf{s} = n_1 - n_2$$

- Solving (7) using the two constrains (8), (9) is now a standard algebra using Lagrange multiplier (which we denote λ and 2μ)

$$(11)\quad \frac{\partial}{\partial s_i}\left(\sum_{i,j} L_{ij}\, s_i\, s_j + \lambda\left(n - \sum_i s_i^2\right) + 2\,\mu\left((n_1 - n_2) - \sum_i s_i\right)\right) = 0$$

- and we get

$$(12)\quad \sum_j L_{ij}\, s_j = \lambda s_i + \mu$$

- and in matrix form

$$(13)\quad \mathbf{L}\mathbf{s} = \lambda\mathbf{s} + \mu\mathbf{1}$$

- Using $\mathbf{L}\cdot\mathbf{1}=0$ and (10) and multiply (13) and the left with $\mathbf{1}^T$ we get

$$(14)\quad \mathbf{1}^T \mathbf{L}\mathbf{s} = \lambda\mathbf{1}^T \mathbf{s} + \mu\mathbf{1}^T \mathbf{1} \Longrightarrow$$

$$0 = \lambda\,(n_1 - n_2) + \mu n$$

$$(15)\quad \mu = -\lambda\,\frac{n_1 - n_2}{n}$$

- If we define a new vector

$$(16)\quad \mathbf{x} = \mathbf{s} + \frac{\mu}{\lambda}\,\mathbf{1} = \mathbf{s} - \frac{n_1 - n_2}{n}\,\mathbf{1}$$

- And finally (13) tell us

$$(17)\quad \mathbf{L}\mathbf{x} = \mathbf{L}\left(\mathbf{s} + \frac{\mu}{\lambda}\,\mathbf{1}\right) = \mathbf{L}\mathbf{s} = \lambda\mathbf{s} + \mu\mathbf{1} = \lambda\mathbf{x}$$

- So $\mathbf{x}$ is an eigenvector of $\mathbf{L}$ with eigenvalue λ!!!
- Which one we should take? The one that gives the smallest cut size (R)
- Notice that $\mathbf{1}^T\mathbf{x}$ =0 (check at home) so $\mathbf{x}$ is ortogonal to $\mathbf{1}$ and therefore the eigenvector vector (1,1,1,1,1...) is out of question (which goes with eigenvalue λ_1 =0).
- To minimize R we note (check $\mathbf{s}^T\,\mathbf{L}\mathbf{x}$)

$$(18)\quad R = \frac{1}{4}\,\mathbf{s}^T\,\mathbf{L}\mathbf{s} = \frac{1}{4}\,\mathbf{x}^T\,\mathbf{L}\mathbf{x} = \frac{1}{4}\,\lambda\mathbf{x}^T\,\mathbf{x}$$

- Since $\mathbf{x}^T\,\mathbf{x} = \frac{n_1\,n_2}{n}$ (check home)

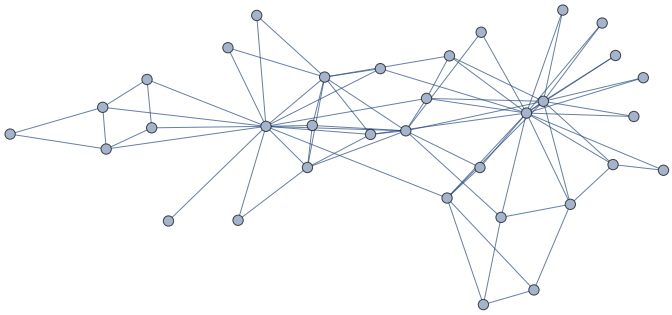
$$(19)\quad R = \frac{n_1\,n_2}{n}\,\lambda$$

- So clearly we need to select the smallest eigen value (that is not 0) usually, λ_2 and let $\boldsymbol{v}_2$ be the corresponding eigenvector
- To finalize the selection we need to take the n_1 (or n_2) nodes correspond to the n_1 largest values of $\boldsymbol{v}_2$
- At the end, very simple solution and algorithm
  - find the eigenvector $\boldsymbol{v}_2$ correspond to the second smallest eigenvalue of the graph Laplacian
  - Sort the elements in $\boldsymbol{v}_2$, compare two possible cuts (first n_1 or first n_2 are in group 1).
  - Take the best cut

Example :-)

In[963]:=
```
Karate = Graph[ExampleData[{"NetworkGraph", "ZacharyKarateClub"}],
  VertexLabels → Placed["Name", Tooltip]]
```

Out[963]=



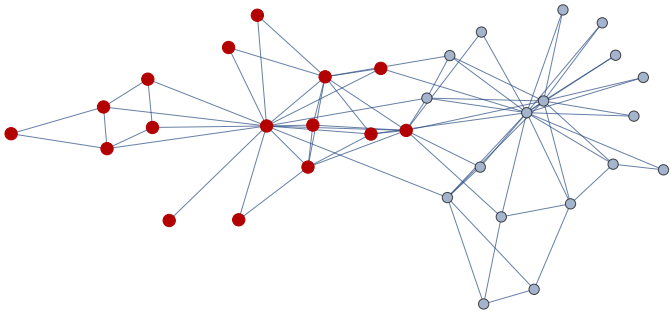
■ The “True” Split

In[964]:=
```
KarateClub1 = {3, 4, 14, 2, 1, 8, 22, 20, 18, 13, 12, 7, 17, 6, 5, 11};
```

In[965]:=
```
KarateClub2 = Complement[VertexList[Karate], KarateClub1];
```

In[966]:=
```
HighlightGraph[Karate, KarateClub1, VertexSize → 0.6]
```

Out[966]=



■ Split Sizes

In[967]:=
```
{Length[KarateClub1], Length[KarateClub2]}
```

Out[967]=
```
{16, 18}
```

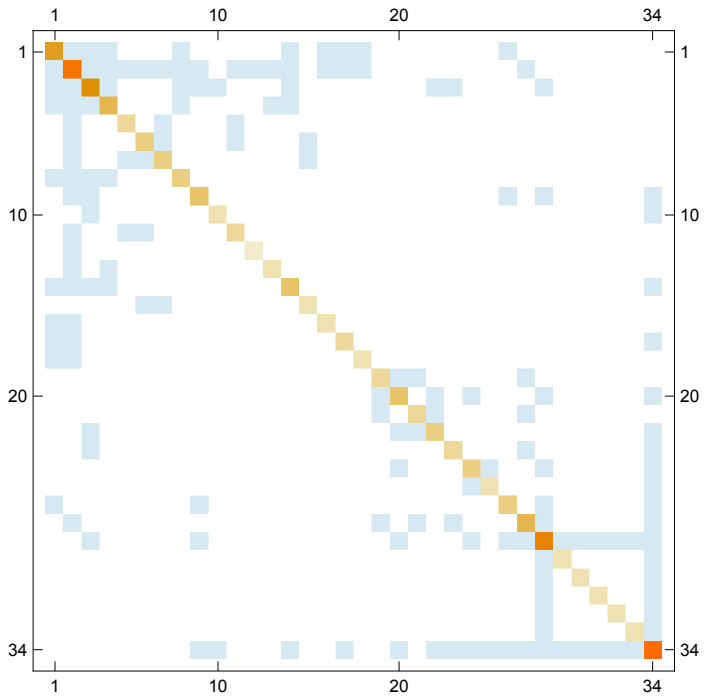
■ Laplacian

In[968]:=
```
LapM = KirchhoffMatrix[Karate];
```

In[969]:=

```
MatrixPlot[LapM]
```

Out[969]=



In[970]:=

```
KarateEigenVlaues = Eigenvalues[LapM] // N
```

Out[970]=

```
{18.1367, 17.0552, 13.3061, 10.9211, 9.77724, 6.9962, 6.51554,
 6.33159, 5.61803, 5.3786, 4.58079, 4.48001, 4.27588, 3.47219, 3.38197,
 3.37615, 3.24207, 3.01396, 2.74916, 2.48709, 2., 2., 2., 2., 2., 1.95505,
 1.82606, 1.7619, 1.59928, 1.2594, 1.12501, 0.909248, 0.468525, 0.}
```

In[971]:=

```
KarateEigenVectors =  Eigenvectors[N[LapM]];
```

In[972]:=

```
V2 = N[KarateEigenVectors[[33]]]
```

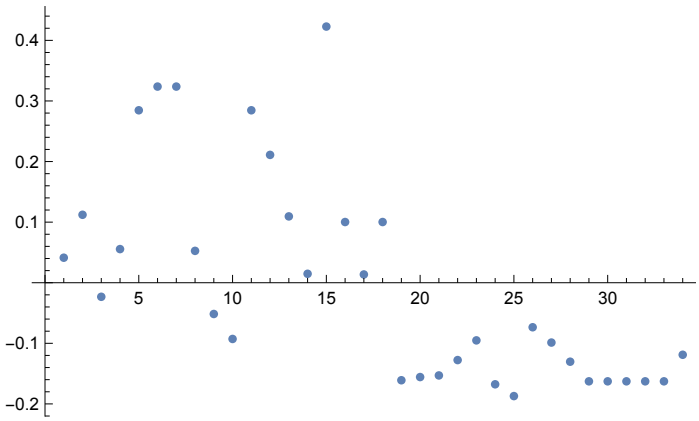
Out[972]=

```
{0.0412879, 0.112137, -0.023219, 0.0554998, 0.284605, 0.323727, 0.323727,
 0.052586, -0.0516013, -0.0928009, 0.284605, 0.210993, 0.109461, 0.014742,
 0.422765, 0.100181, 0.0136371, 0.100181, -0.160963, -0.155695, -0.153026,
 -0.127664, -0.0951523, -0.16765, -0.18711, -0.0734996, -0.0987534,
 -0.130345, -0.162751, -0.162751, -0.162751, -0.162751, -0.162751, -0.118903}
```

In[973]:=

```
ListPlot[V2]
```

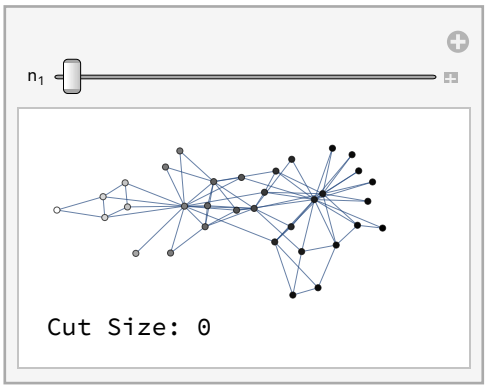
Out[973]=

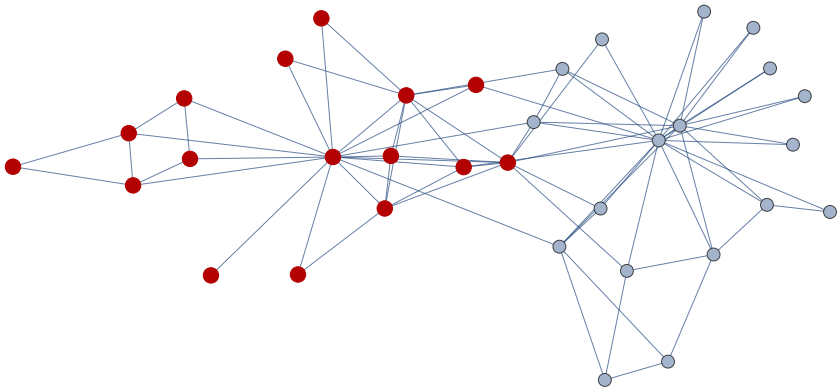


In[974]:=

```
Manipulate[Column[
  {HighlightGraph[Graph[Karate, VertexStyle → Table[VertexList[Karate][[i]] →
        GrayLevel[Rescale[V2][[i]]], {i, 1, VertexCount[Karate]}],
     VertexShapeFunction → Table[VertexList[Karate][[Ordering[V2][[-i]]]] →
        "Square", {i, 1, j}], VertexSize → 0.6], ed = Complement[
      Complement[EdgeList[Karate], EdgeList[Subgraph[Karate, Complement[
          VertexList[Karate], VertexList[Karate][[Ordering[V2, -j]]]]]],
      EdgeList[Subgraph[Karate, VertexList[Karate][[Ordering[V2, -j]]]]]],
   "Cut Size: " <> ToString[Length[ed]]}],
 {{j, 0, "n1"}, 0, VertexCount[Karate],
  1}]
```

Out[974]=





Community Detection

- No input on the community sizes is given
- When to stop? minimum cut? ratio cut?
- The quality of the cut: Do we see less edges than expected
- Minimizing the cut is like maximize the edges with-in groups
- A natural candidate is to look for the division with the highest modularity score (Recall from Homophily)
- Modularity Maximization
- First simple candidate:
  - Modularity maximization by vertex moving algorithm
  - Very similar to Kernighan-Lin Algorithm (initially for 2 groups)
  - Moving one a vertex (not a swap) that increase the modularity (in a round)
  - repeat rounds

Spectral Modularity Maximization

- Recall the modularity of the network (the diffrence between number of "homophily" edges to the expected one in random graph).

$$(20)\ Q = \frac{1}{2m}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta(c_i, c_j)$$

$$(21)\ Q = \frac{1}{2m}\sum_{ij} B_{ij}\,\delta(c_i, c_j)$$

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$$

- Note that

$$(22)\ \sum_j B_{ij} = \sum_j A_{ij} - \frac{k_i}{2m}\sum_j k_j = k_i - \frac{k_i}{2m}\,2m = 0$$

- Lets define s as before (for two groups)

$$(2)\ s_i = \begin{cases} +1 & \text{if vertex } i \text{ belongs to group } 1 \\ -1 & \text{if vertex } i \text{ belongs to group } 2 \end{cases}$$

- And again

$$(3)\ \delta(c_i, c_j) = \frac{1}{2}(s_i s_j + 1) = \begin{cases} 0 & i, j \text{ are in different groups} \\ 1 & i, j \text{ are in the same groups} \end{cases}$$

- We can write (21)

$$(23)\ Q = \frac{1}{4m}\sum_{ij} B_{ij}(s_i s_j + 1) = \frac{1}{4m}\sum_{ij} B_{ij} s_i s_j$$

- And we have the matrix form

$$(24)\ Q = \frac{1}{4m}\,\mathbf{s}^T \mathbf{B}\mathbf{s}$$

- Where B is the modularity matrix and now we have a maximization problem

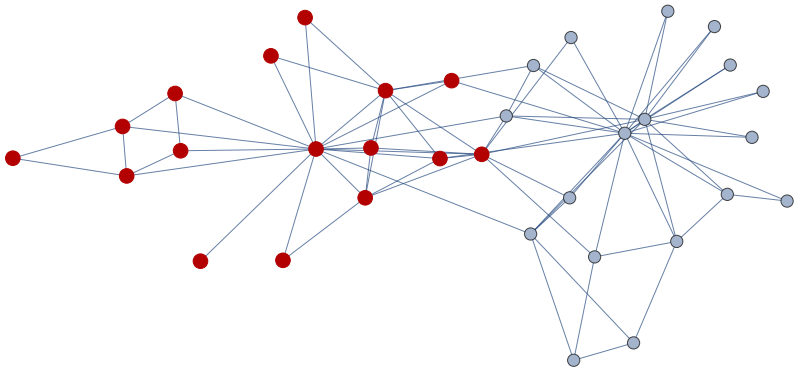
- Few more steps, taking the derivative... like before we find that using the relaxtion method

$$(8)\ \sum_i s_i^2 = n$$

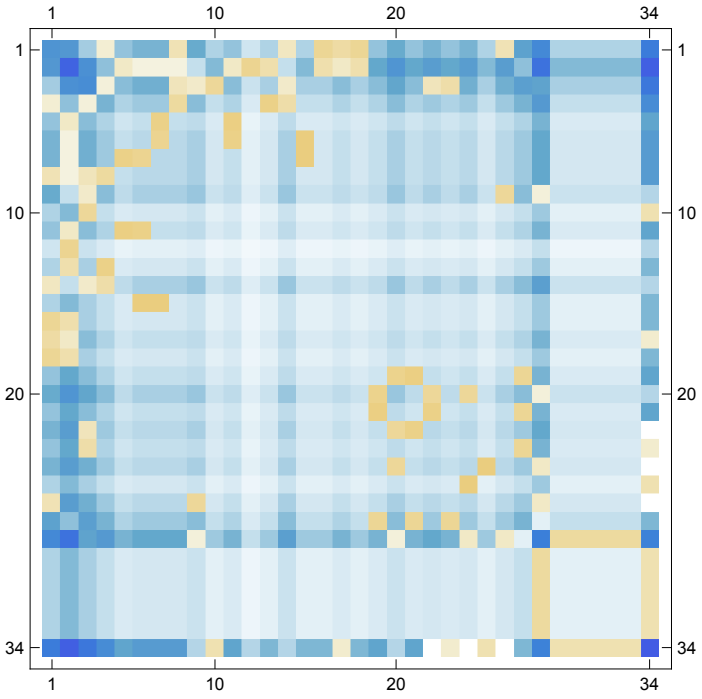
- We note that the solution is the largest eigenvector and the split is according to the sign of the vector elements
  - All positive elements in the eigenvector are in group 1 and all negative are in group 2

Another Example

In[975]:=
```
HighlightGraph[Karate, KarateClub1, VertexSize → 0.6]
```
Out[975]=



In[976]:=
```
ModularityMatrix[G_] :=
 Table[Table[AdjacencyMatrix[G][[i, j]] -
    VertexDegree[G][[i]] × VertexDegree[G][[j]] / (2 VertexCount[G]),
   {i, 1, VertexCount[G]}], {j, 1, VertexCount[G]}]
```
In[977]:=
```
MM = ModularityMatrix[Karate];
```
In[978]:=
```
MatrixPlot[MM]
```
Out[978]=



In[979]:=
```
MMEV = N[Eigenvalues[MM]];
```

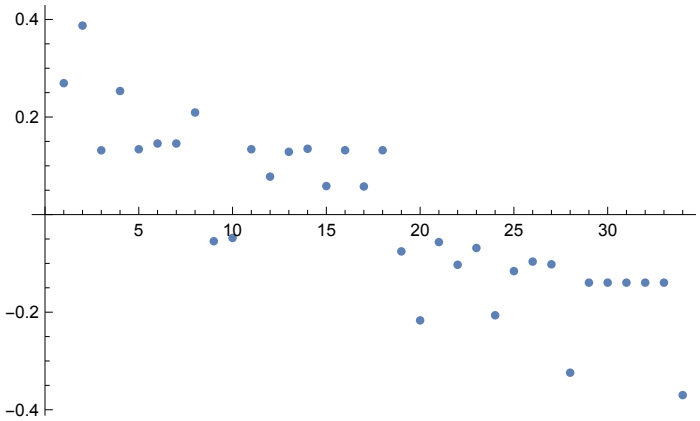
In[980]:=
```
MMEV
```
Out[980]=
```
{-12.6002, 4.97708, -3.46764, -3.26714, 2.9947, -2.93863,
 -2.43275, 2.31542, -2.09044, -2., -1.6874, 1.48711, 1.4557, -1.42928,
 -1.19235, 1.08329, 1.03146, -1.02363, 0.835289, -0.792199, 0.616025,
 0.41973, -0.417146, 0.299473, 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
```

In[981]:=
```
Ordering[MMEV, 1, Greater]
```
Out[981]=
```
{2}
```

In[982]:=
```
MMEVec = Eigenvectors[N[MM]];
```

In[983]:=
```
V1 = MMEVec[[First@Ordering[MMEV, 1, Greater]]]
```
Out[983]=
```
{0.26946, 0.387435, 0.131782, 0.253372, 0.133986, 0.145725, 0.145725, 0.209293,
 -0.054561, -0.0478926, 0.133986, 0.0778248, 0.128713, 0.134942, 0.0585203,
 0.131946, 0.0575945, 0.131946, -0.0754138, -0.216795, -0.0563435,
 -0.10281, -0.0683889, -0.20634, -0.115828, -0.0963309, -0.101917,
 -0.324008, -0.13947, -0.13947, -0.13947, -0.13947, -0.13947, -0.369957}
```

In[984]:=
```
ListPlot[V1]
```
Out[984]=



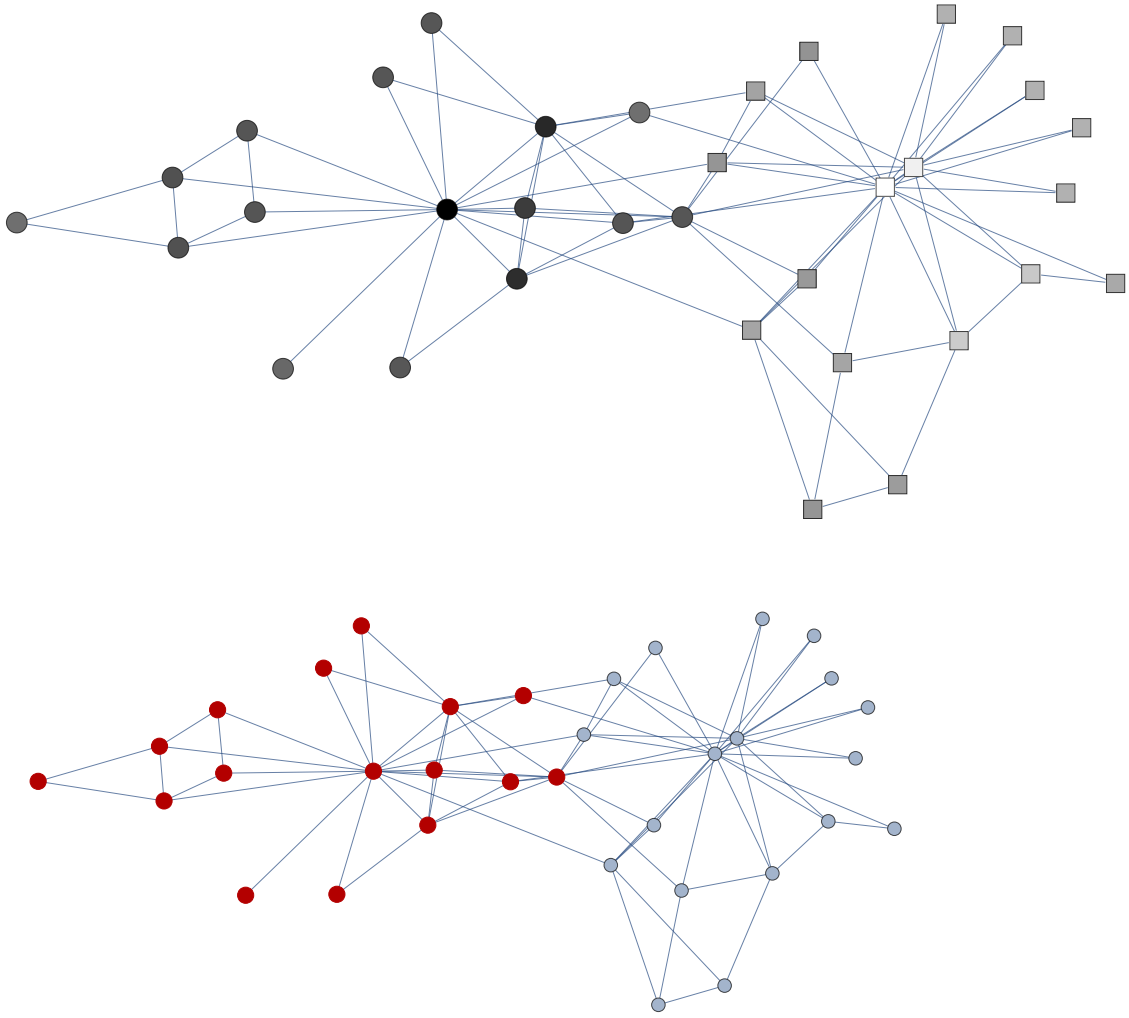
In[985]:=
```
group1 = Flatten[Position[V1, n_ /; n > 0]]
```
Out[985]=
```
{1, 2, 3, 4, 5, 6, 7, 8, 11, 12, 13, 14, 15, 16, 17, 18}
```

In[986]:=
```
group2 = Complement[VertexList[Karate], group1]
```
Out[986]=
```
{9, 10, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34}
```

In[987]:=

```
Graph[Karate,
 VertexStyle → Table[VertexList[Karate][[i]] → GrayLevel[1 - Rescale[V1][[i]]],
   {i, 1, VertexCount[Karate]}],
 VertexShapeFunction → Table[VertexList[Karate][[group2[[i]]]] → "Square",
   {i, 1, Length[group2]}], VertexSize → 0.6]
```

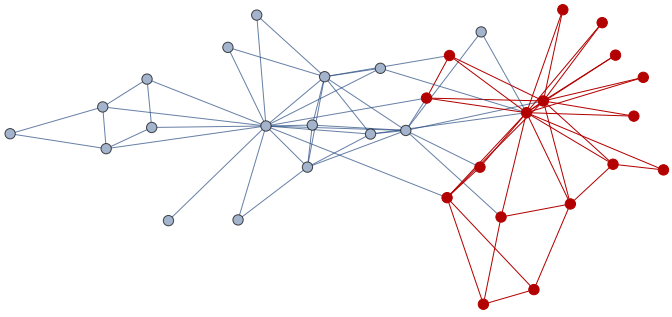
Out[987]=



In[988]:=

```
HighlightGraph[Karate,
 Subgraph[Karate, FindGraphCommunities[Karate, Method → "Modularity"][[1]]]]
```

Out[988]=

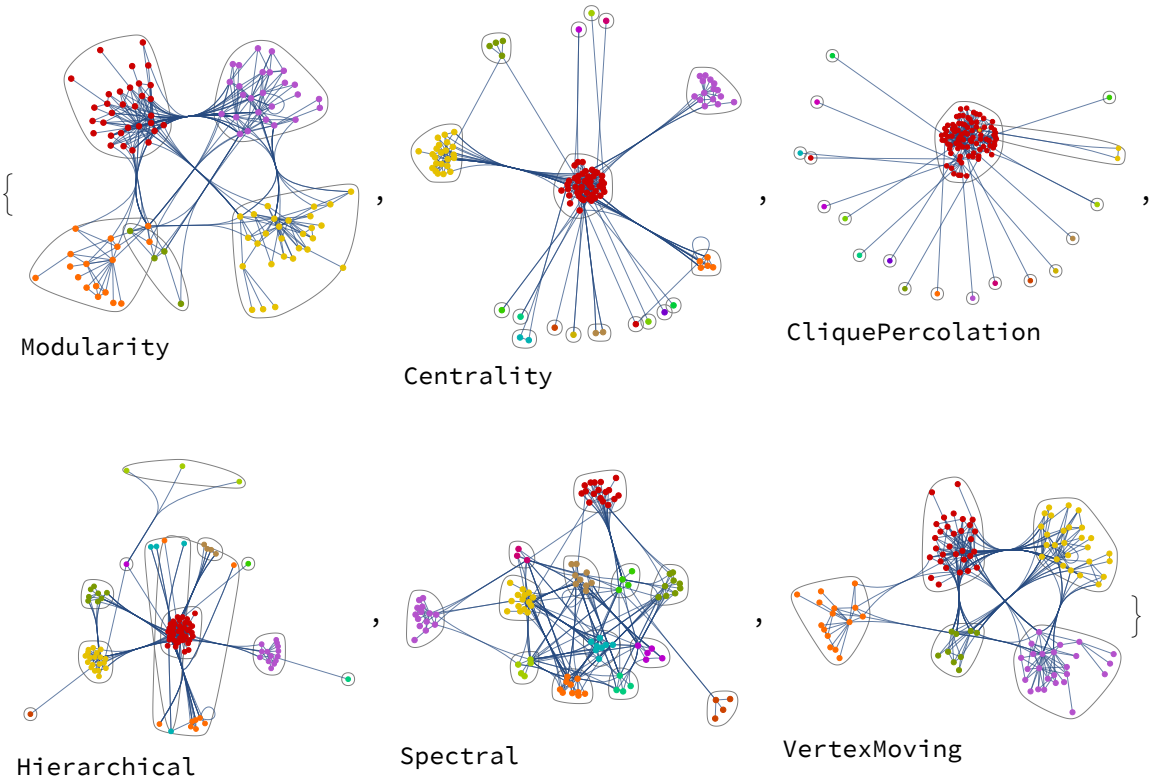


Network of Thrones

In[989]:=
```
Length[FindGraphCommunities[ThronesG, Method → "Modularity"]]
```
Out[989]=
```
5
```

In[990]:=
```
options = {"Modularity", "Centrality",
   "CliquePercolation", "Hierarchical", "Spectral", "VertexMoving"};
```

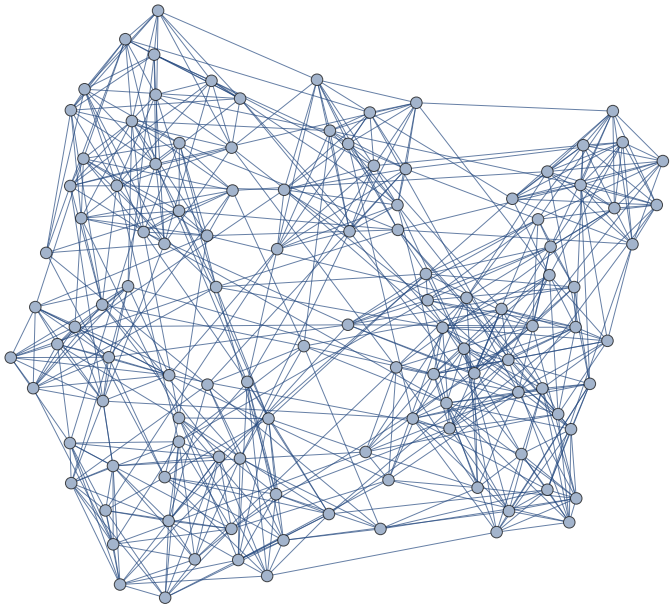
In[991]:=
```
Column[{CommunityGraphPlot[ThronesG,
     FindGraphCommunities[ThronesG, Method → #]], #}] & /@ options
```
Out[991]=



Football Network

In[992]:=
```
FootBall = Graph[ExampleData[{"NetworkGraph", "AmericanCollegeFootball"}],
  VertexLabels → Placed["Name", Tooltip]]
```

Out[992]=



In[993]:=
```
VertexConference =
  PropertyValue[{FootBall, #}, "Conference"] & /@ VertexList[FootBall] ;
```

In[994]:=
```
Tally[VertexConference]
```

Out[994]=
```
{{Big East, 8}, {Western Athletic, 10}, {Atlantic Coast, 9}, {Mid-American, 13},
 {Sun Belt, 7}, {Big Ten, 11}, {Mountain West, 8}, {Pacific Ten, 10},
 {Big Twelve, 12}, {Conference USA, 10}, {Southeastern, 12}, {Independents, 5}}
```

In[995]:=
```
Conference = DeleteDuplicates[VertexConference]
```

Out[995]=
```
{Big East, Western Athletic, Atlantic Coast,
 Mid-American, Sun Belt, Big Ten, Mountain West, Pacific Ten,
 Big Twelve, Conference USA, Southeastern, Independents}
```

In[996]:=
```
Length[Conference]
```

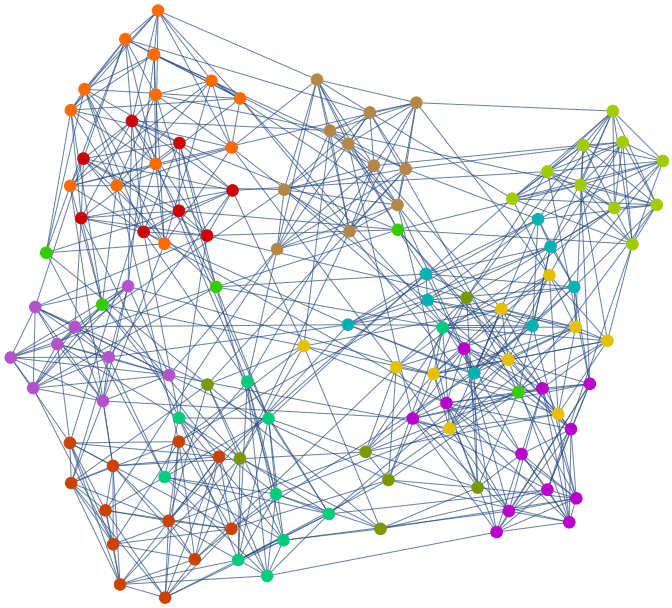
Out[996]=
```
12
```

In[997]:=
```
BallCommunities =
  VertexList[FootBall][[Flatten[Position[VertexConference, #]]]] & /@ Conference;
```

In[998]:=

```
HighlightGraph[FootBall, BallCommunities]
```

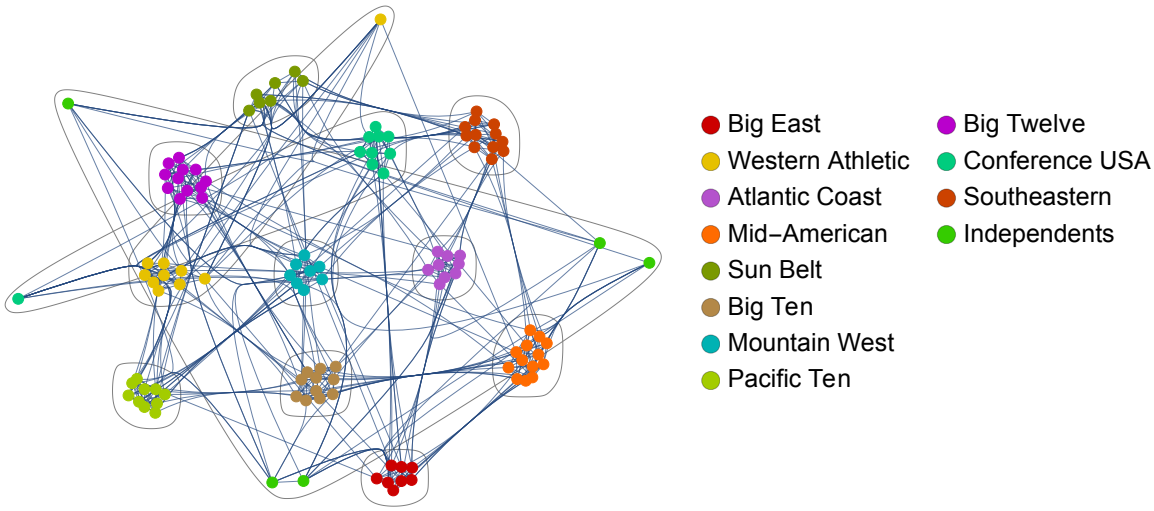
Out[998]=



In[999]:=

```
CommunityGraphPlot[FootBall, BallCommunities, PlotLegends → Conference]
```

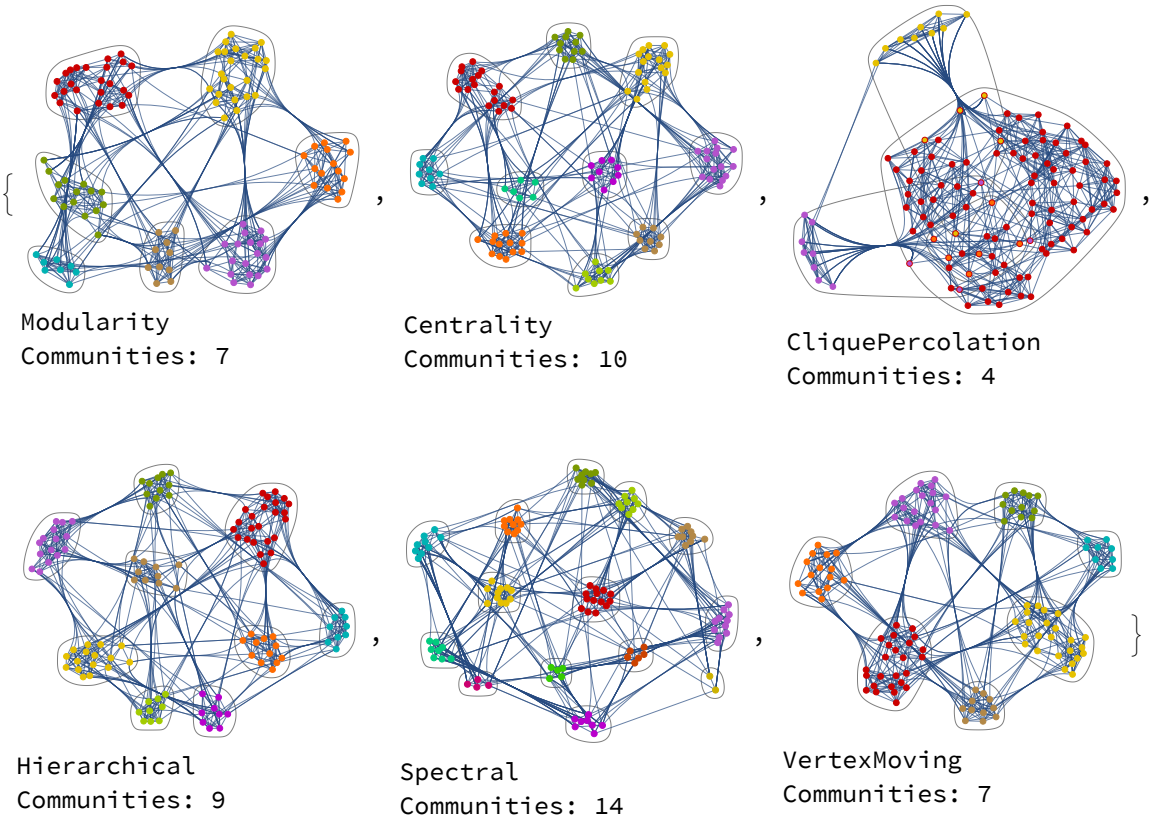
Out[999]=



In[1000]:=

```
Column[{CommunityGraphPlot[FootBall,
    cg = FindGraphCommunities[FootBall, Method → #]],
   #, "Communities: " <> ToString[Length[cg]]}] & /@ options
```

Out[1000]=

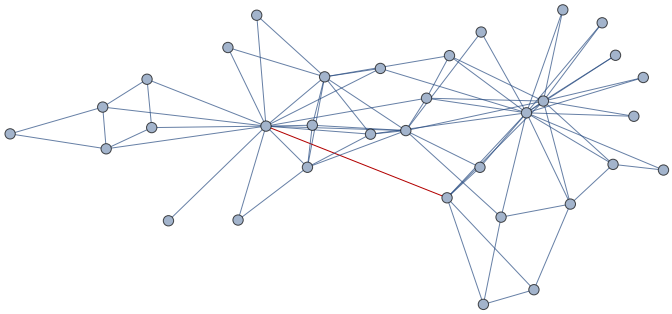


Edge Betweenness Algorithm

Use edge Betweenness to split the graph

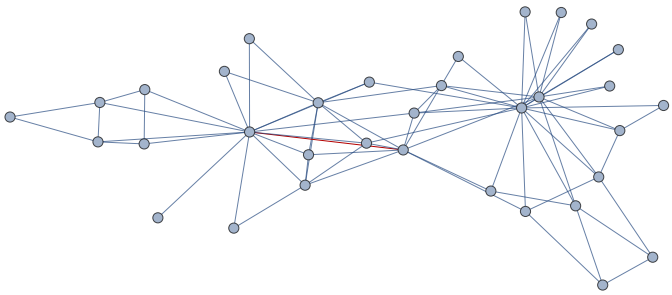
In[1001]:=
```
TopEdgeBetweennessCentrality[g_] :=
 EdgeList[g][[Ordering[EdgeBetweennessCentrality[g], -1]]]
```

In[1002]:=
```
HighlightGraph[Karate, TopEdgeBetweennessCentrality[Karate]]
```
Out[1002]=



In[1003]:=
```
K1 = EdgeDelete[Karate, TopEdgeBetweennessCentrality[Karate]];
```

In[1004]:=
```
HighlightGraph[K1, TopEdgeBetweennessCentrality[K1]]
```
Out[1004]=

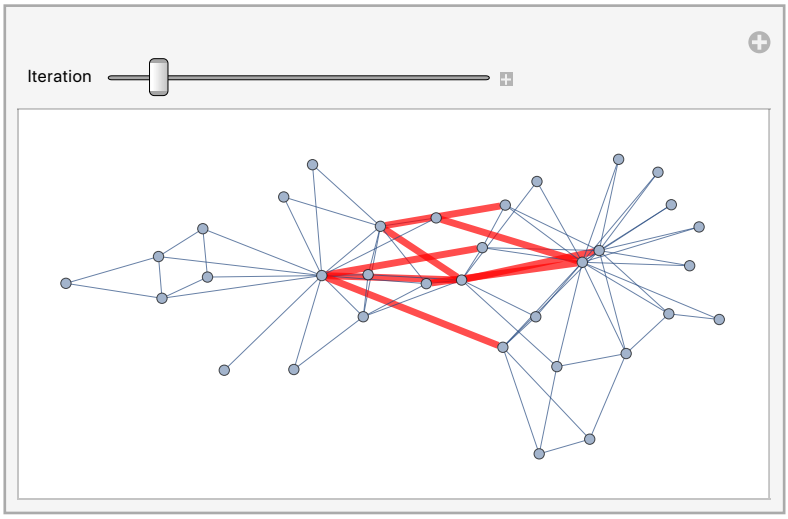


In[1005]:=
```
NextBetweennessEdge[g_, i_] := Module[{graph = g, list = {}},
  For[j = 1, j ≤ i, j++,
   list = Union[list, TopEdgeBetweennessCentrality[graph]];
   graph = EdgeDelete[graph, TopEdgeBetweennessCentrality[graph]];];
  list]
```

In[1006]:=

```
Manipulate[HighlightGraph[Karate,
  Style[NextBetweennessEdge[Karate, i], Red, Thickness[0.01]]],
 {{i, 1, "Iteration"}, 1, EdgeCount[Karate], 1}]
```

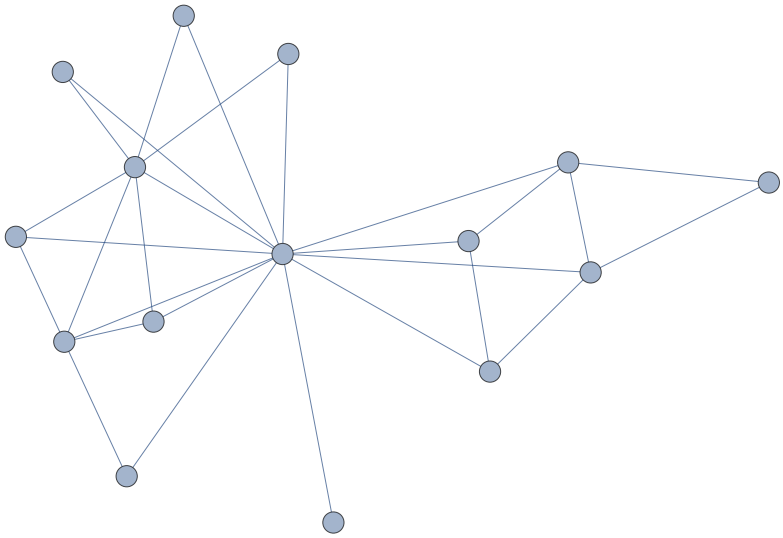
Out[1006]=

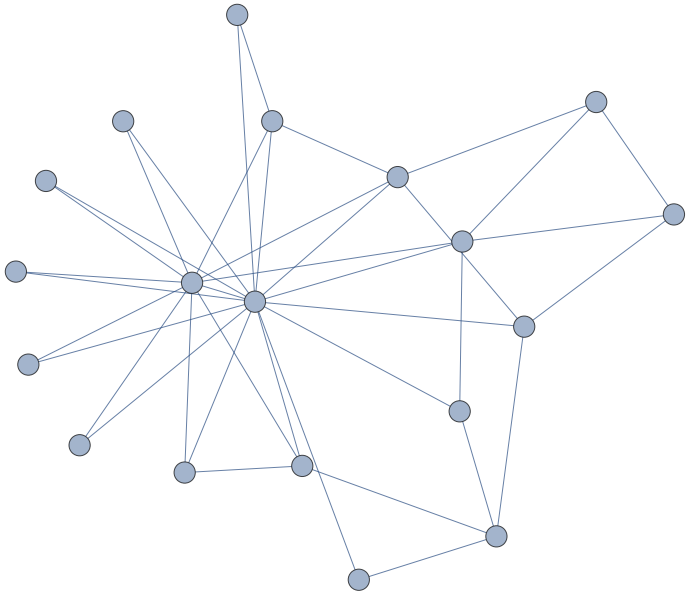


In[1007]:=

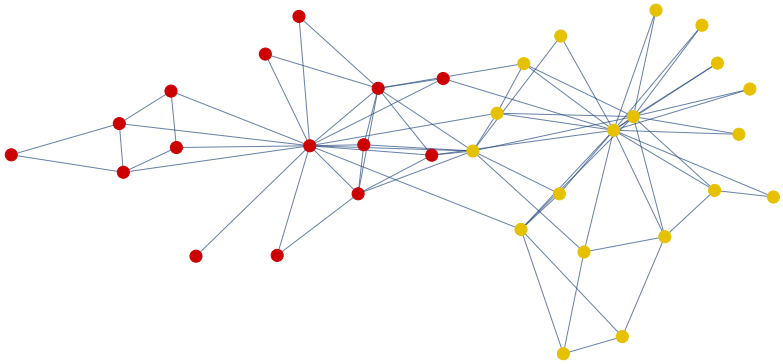
```
Comm = EdgeDelete[Karate, NextBetweennessEdge[Karate, 11]]
HighlightGraph[Karate, Reverse[ConnectedComponents[Comm]]]
```

Out[1007]=

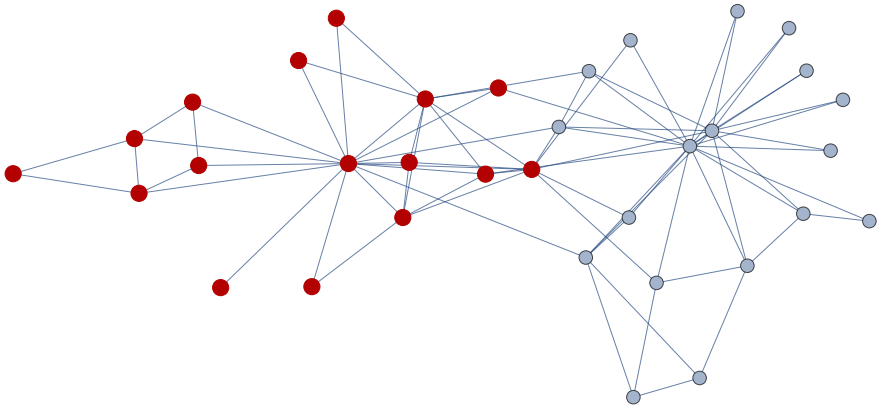




Out[1008]=



■ Ground Truth



A dendrogram tree

An hierarchical decomposition.

In[1009]:=
```
Needs["HierarchicalClustering`"]
```

In[1010]:=
```
Off[DirectAgglomerate::ties]
```

```
Manipulate[DendrogramPlot[{1, 2, 10, 4, 8, 15, 13}, LeafLabels → (# &),
  HighlightLevel → i, Orientation → Left], {{i, None, "Level"}, {None, 2, 3, 6}}]
```

- Find a betweenness cut

In[1019]:=
```
BetweennessCut[g_] := Module[{graph = g},
  While[Length[cc = ConnectedComponents[graph]] < 2,
   graph = EdgeDelete[graph, TopEdgeBetweennessCentrality[graph]];];
  cc]
```

- Find a dendrogram with betweenness

In[1020]:=
```
BetweennessClusters[g_] := Module[{graph = g, list = {}, c, d, e},
  If[VertexCount[g] == 1, First@VertexList[g], c = BetweennessCut[g];
   Cluster[BetweennessClusters[d = Subgraph[g, First@c]], BetweennessClusters[
     e = Subgraph[g, Last@c]], EdgeCount[g], Length[First@c], Length[Last@c]]]]
```

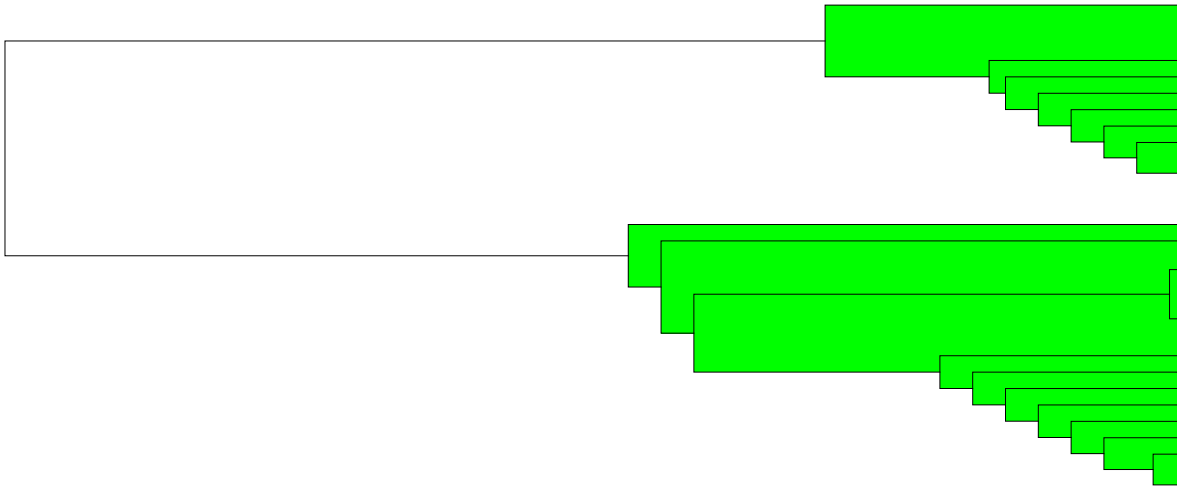
```
Cl = BetweennessClusters[Karate];
```

■ A dendrogram based on edge betweenness

In[1018]:=

```
Show[Graphics[Disk[{3, 15}, .5]],
 DendrogramPlot[Cl, Orientation → Left, LeafLabels → (# &), HighlightLevel → 2]]
```

Out[1018]=



In[1021]:=

```
KarateClub1
```

Out[1021]=

```
{3, 4, 14, 2, 1, 8, 22, 20, 18, 13, 12, 7, 17, 6, 5, 11}
```

Hierarchical Clustering

- Hierarchical decomposition - Bottom up
- Combine nodes based on similarity - start with pairs and and increase “cluster” size
  - Join to groups based on single....
- Algorithm:

1. Choose a similarity measure and evaluate it for all vertex pairs.
2. Assign each vertex to a group of its own, consisting of just that one vertex. The initial similarities of the groups are simply the similarities of the vertices.
3. Find the pair of groups with the highest similarity and join them together into a single group.
4. Calculate the similarity between the new composite group and all others using one of the three methods (single-, complete-, or average- linkage clustering).
5. Repeat from step 3 until all vertices have been joined into a single group.

- Two similarity example (structural similarity):
  - Pearson correlation coefficient

$$r_{ij} = \frac{\text{cov}(A_i, A_j)}{\sigma_i \sigma_j} = \frac{\sum_k (A_{ik} - \mu_i)(A_{jk} - \mu_j)}{\sigma_i \sigma_j}$$

$$\mu_i = \frac{\sum_k A_{ik}}{n}$$

$$\sigma_i = \sqrt{\sum_k (A_{ik} - \mu_i)^2}$$

  - Cosine Similarity

$$\cos\Theta = \frac{\mathbf{x} \cdot \mathbf{y}}{|x| \cdot |y|}$$

$$\sigma_{ij} = \cos\Theta = \frac{\sum_k A_{ik} A_{jk}}{\sqrt{\sum_k A_{ik}^2}\ \sqrt{\sum_k A_{jk}^2}}$$

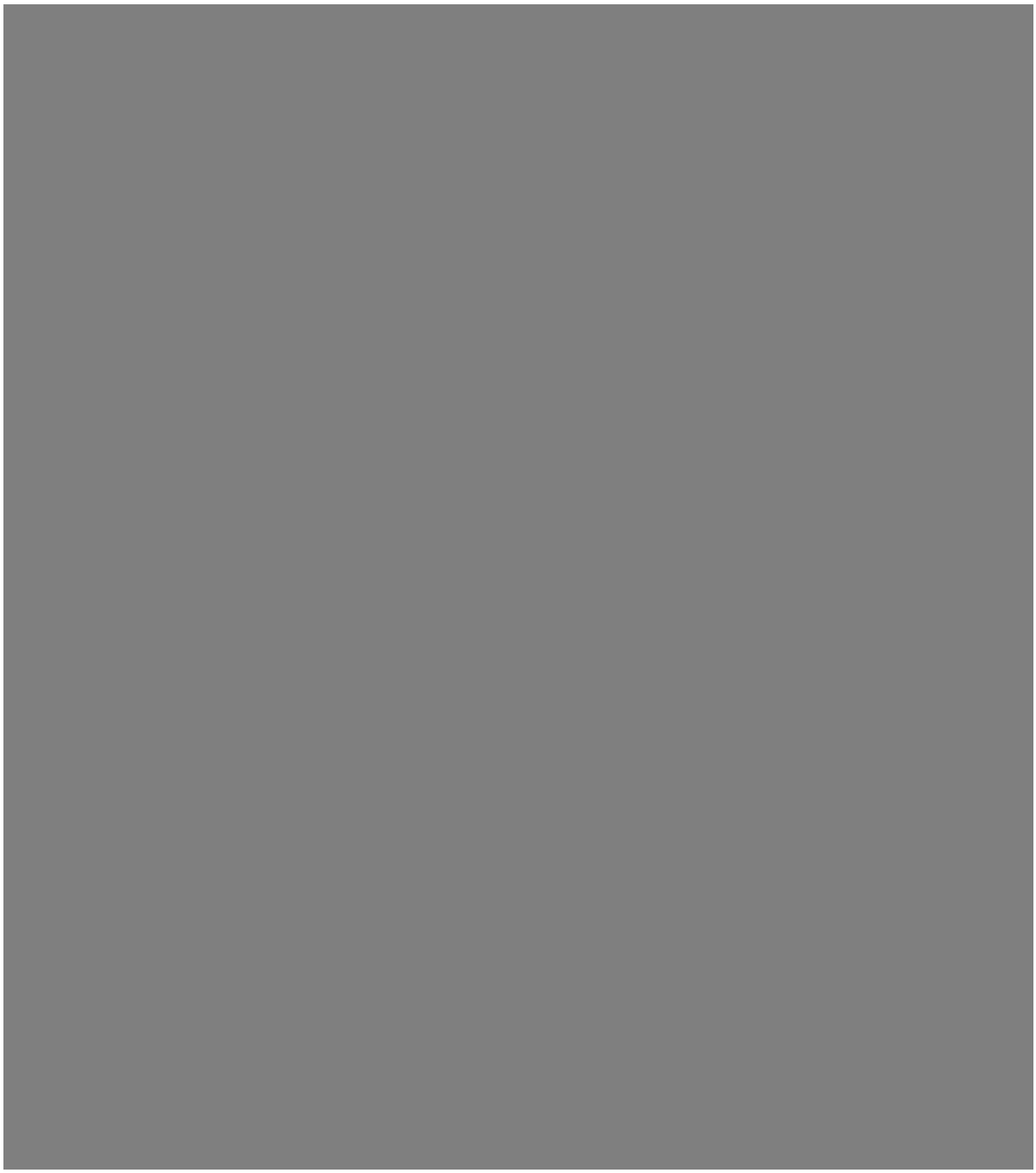
In[1022]:=

```
A = AdjacencyMatrix[Karate];
```

In[1023]:=

```
Manipulate[Column[{HighlightGraph[Karate,
    {VertexList[Karate][[i = First@First@Position[VertexList[Karate], x]]],
     VertexList[Karate][[j = First@First@Position[VertexList[Karate], y]]]}],
   Normal[A[[i]]], Normal[A[[j]]], "Perason Correlation: " <>
    ToString[N[-(Correlation[A[[i]], A[[j]]] - 1), 4]], "Cosine Similarity: " <>
    ToString[N[CosineDistance[Normal[A[[i]]], Normal[A[[j]]]], 4]]}],
 {x, VertexList[Karate]}, {y, VertexList[Karate]}]
```

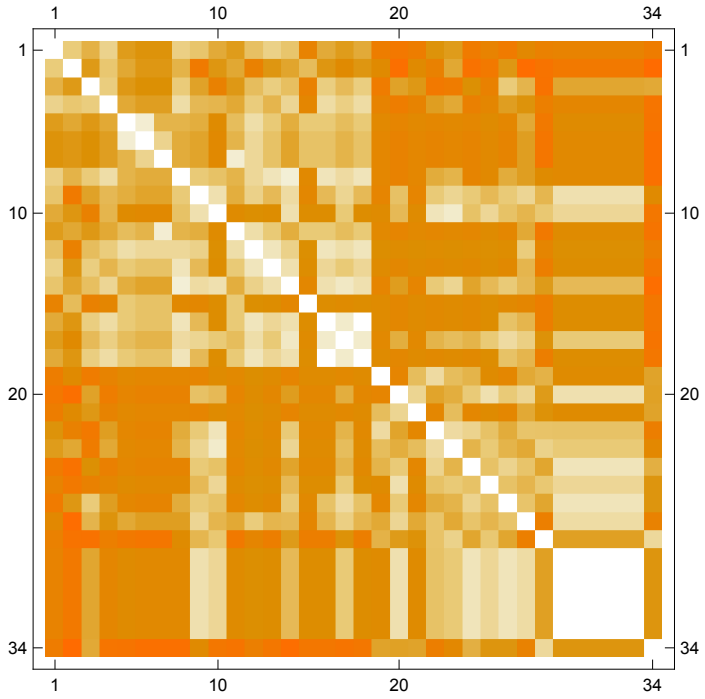
Out[1023]=



Let's Check

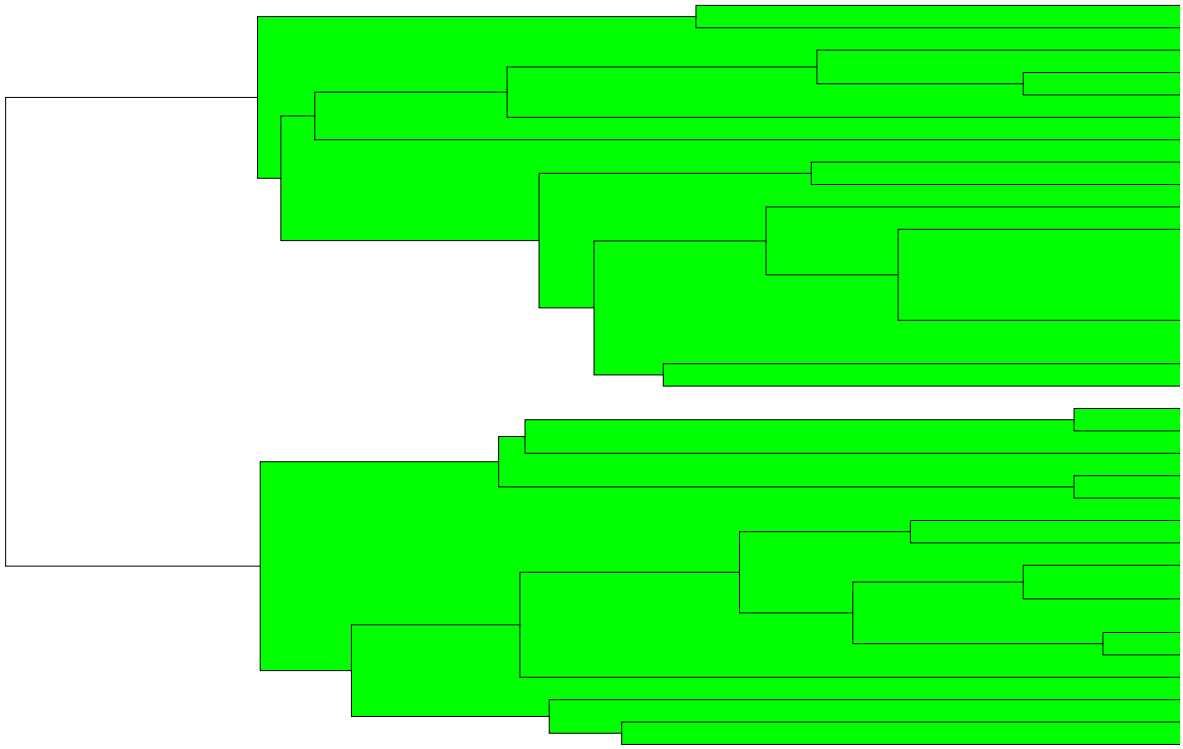
```
A = AdjacencyMatrix[Karate];

MatrixPlot[-(Correlation[A] - 1)]
```



In[1024]:=
```
DendrogramPlot[DirectAgglomerate[-(Correlation[A] - 1), VertexList[Karate],
  Linkage → "Average"], LeafLabels → (# &), Orientation → Left, HighlightLevel → 3]
```

Out[1024]=



In[1026]:=
```
KarateClub1
```

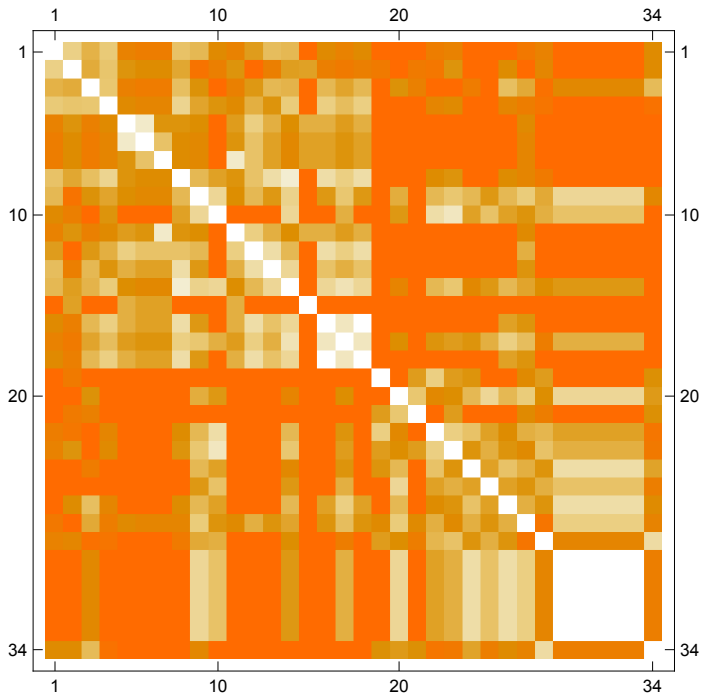
Out[1026]=
```
{3, 4, 14, 2, 1, 8, 22, 20, 18, 13, 12, 7, 17, 6, 5, 11}
```

In[1027]:=
```
CosineDistanceMatrix[A_] :=
 Table[Table[CosineDistance[Normal[A[[i]]], Normal[A[[j]]]], {i, 1, Length[A]}],
  {j, 1, Length[A]}]
```

In[1028]:=

```
MatrixPlot[CosineDistanceMatrix[A]]
```

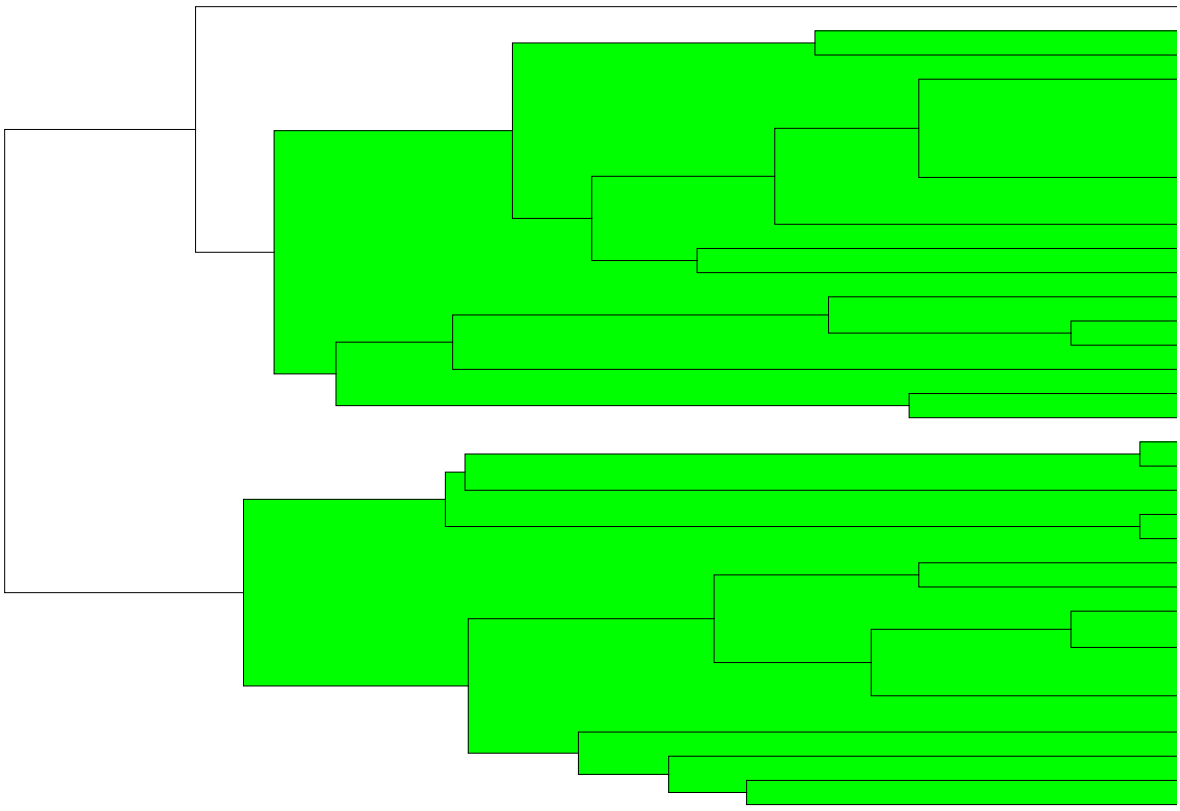
Out[1028]=



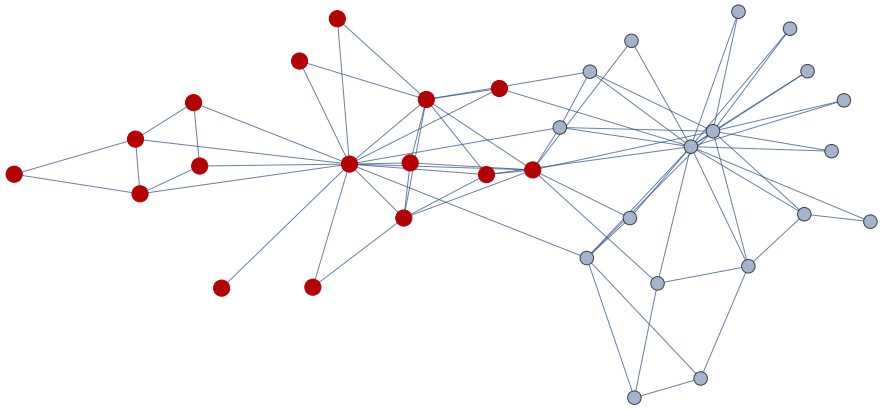
In[1029]:=

```
DendrogramPlot[DirectAgglomerate[CosineDistanceMatrix[A], VertexList[Karate],
  Linkage → "Average"], LeafLabels → (# &), Orientation → Left, HighlightLevel → 3]
```

Out[1029]=

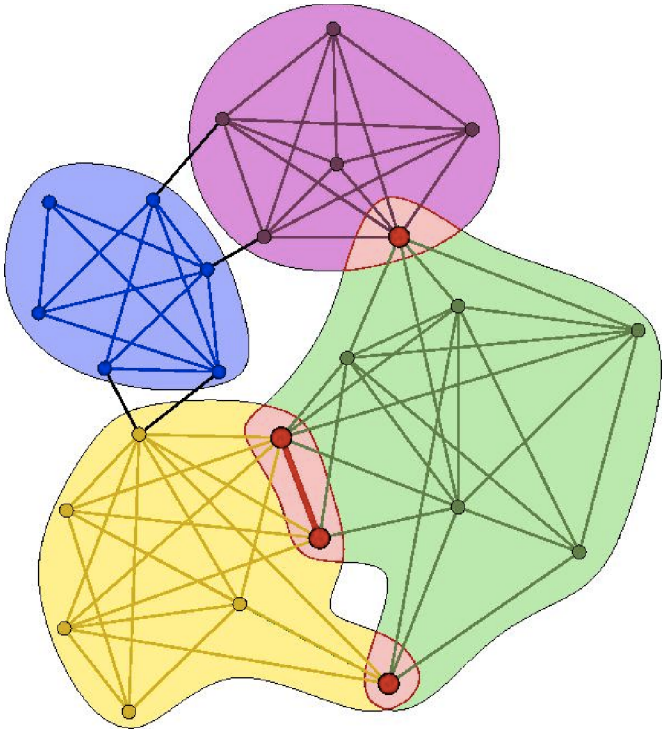


Recall the correct split



Clique Percolation Method

The clique percolation method builds up the communities from k-cliques, which correspond to complete (fully connected) sub-graphs of k nodes. (E.g., a k-clique at k = 3 is equivalent to a triangle). Two k-cliques are considered adjacent if they share k – 1 nodes. A community is defined as the maximal union of k-cliques that can be reached from each other through a series of adjacent k-cliques. Such communities can be best interpreted with the help of a k-clique template (an object isomorphic to a complete graph of k nodes). Such a template can be placed onto any k-clique in the graph, and rolled to an adjacent k-clique by relocating one of its nodes and keeping its other k – 1 nodes fixed. Thus, the k-clique communities of a network are all those sub-graphs that can be fully explored by rolling a k-clique template in them, but cannot be left by this template.



Homework 7

Play around with "Network of Thrones". Compare the tribe to community detecting algorithms. What is working best.

1. Using *Mathematica* FindGraphCommunities function. What is working the best?
2. Using "Edge Betweenness Algorithm" (and Hierarchical Clustering). Does it finds a "good" dendrogram tree ? Explain/show.
3. Using Hierarchical Clustering and a similarity measure Does it finds a "good" dendrogram tree ? Explain/show.
4. Using the "Louvain Modularity" method (need to implement it).
5. Bounus: Playaround with the Facebook network: Find something intresting (you can use all previous units).