L11 Spectral Graph K-partitioning

Ivan Slapničar

May 8, 2018

1 Spectral Graph K-partitioning

Instead of using recursive spectral bipartitioning, the graph k-partitioning problem can be solved using k eigenvectors which correspond to k smallest eigenvalues of Laplaciain matrix or normalized Laplacian matrix, respectively.

Suggested reading is U. von Luxburg, A Tutorial on Spectral Clustering, which includes the quote *"spectral clustering cannot serve as a "black box algorithm" which automatically detects the correct clusters in any given data set. But it can be considered as a powerful tool which can produce good results if applied with care."*

1.1 Prerequisites

The reader should be familiar with k-means algorithm, spectral graph bipartitioning and recursive bipartitioning.

1.2 Competences

The reader should be able to apply graph spectral k-partitioning to data clustering problems.

Credits: The notebook is based on [Mir05].

References

[Mir05] I. Mirošević, 'Spectral Graph Partitioning and Application to Knowledge Extraction', M.Sc. Thesis, University of Zagreb, 2005 (in Croatian).

1.3 The relaxed problem

Let $G = (V, E)$ be a weighted graph with weights ω, with weights matrix W, Laplacian matrix $L = D - W$, and normalized Laplacian matrix $L_n = D^{-1/2}(D - W)D^{-1/2}$.

Let the k-partition $\pi_k = \{V_1, V_2, ..., V_k\}$, the cut $cut(\pi_k)$, the proportional cut $pcut(\pi_k)$ and the normalized cut $ncut(\pi_k)$ be defined as in the Spectral Graph Bipartitioning notebook.

1.3.1 Definition

Partition vectors of a k-partition π_k are

$$\begin{aligned}
h_1 &= [\overbrace{1,\cdots,1}^{|V_1|},0,\cdots,0,\cdots,0,\cdots,0]^T \\
h_2 &= [0,\cdots,0,\overbrace{1,\cdots,1}^{|V_2|},\cdots,0,\cdots,0]^T \\
&\vdots \\
h_k &= [0,\cdots,0,0,\cdots,0,\cdots,\overbrace{1,\cdots,1}^{|V_k|}]^T.
\end{aligned}$$

1.3.2 Facts

1. Set

$$\begin{aligned}
X &= \begin{bmatrix} x_1 & x_2 & \cdots & x_k \end{bmatrix}, \quad x_i = \frac{h_i}{\|h_i\|_2}, \\
Y &= \begin{bmatrix} y_1 & y_2 & \cdots & y_k \end{bmatrix}, \quad y_i = \frac{D^{1/2}h_i}{\|D^{1/2}h_i\|_2}.
\end{aligned}$$

It holds

$$\begin{aligned}
cut(V_i, V\backslash V_i) &= h_i^T(D-W)h_i = h_i^T L h_i, \quad \omega(C_i) = h_i^T D h_i, \quad |C_i| = h_i^T h_i, \\
pcut(\pi_k) &= \frac{h_1^T L h_1}{h_1^T h_1} + \cdots + \frac{h_k^T L h_k}{h_k^T h_k} = x_1^T L x_1 + \cdots + x_k^T L x_k = \text{trace}(X^T L X), \\
ncut(\pi_k) &= \frac{h_1^T L h_1}{h_1^T D h_1} + \cdots + \frac{h_k^T L h_k}{h_k^T D h_k} = \text{trace}(Y^T L_n Y).
\end{aligned}$$

2. The **relaxed** k-partitioning problems are trace-minimization problems,

$$\begin{aligned}
\min_{\pi_k} pcut(\pi_k) &\geq \min_{\substack{X^T X = I \\ X \in \mathbb{R}^{n\times k}}} \text{trace}(X^T L X), \\
\min_{\pi_k} ncut(\pi_k) &\geq \min_{\substack{Y^T Y = I \\ Y \in \mathbb{R}^{n\times k}}} \text{trace}(Y^T L_n Y).
\end{aligned}$$

3. **Ky-Fan Theorem**: Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix with eigenvalues $\lambda_1 \leq \cdots \leq \lambda_n$. Then

$$\min_{\substack{Z \in \mathbb{R}^{n\times k} \\ Z^T Z = I}} \text{trace}\left(Z^T A Z\right) = \sum_{i=1}^{k} \lambda_i.$$

4. Let $\lambda_1 \leq \cdots \leq \lambda_n$ be the eigenvalues of L with eigenvectors $v^{[1]}, \cdots, v^{[k]}$. The solution of the relaxed proportional cut problem is the matrix $X = \begin{bmatrix} v^{[1]} & \cdots & v^{[k]} \end{bmatrix}$, and it holds $\min_{\pi_k} pcut(\pi_k) \geq \sum_{i=1}^{k} \lambda_i$.
5. Let $\mu_1 \leq \cdots \leq \mu_n$ be the eigenvalues of L_n with eigenvectors $w^{[1]}, \cdots, w^{[k]}$. The solution of the relaxed normalized cut problem is the matrix $Y = \begin{bmatrix} w^{[1]} & \cdots & w^{[k]} \end{bmatrix}$, and it holds $\min_{\pi_k} ncut(\pi_k) \geq \sum_{i=1}^{k} \mu_i$.
6. It remains to recover the k-partition. The k-means algorithm applied to rows of the matrices X or $D^{-1/2}Y$, will compute the k centers and the assignment vector whose i-th component denotes the subset V_j to which the vertex i belongs.

1.3.3 Example - Graph with three clusters

In [1]:
```
# Some packages
using LightGraphs
using GraphPlot
using Clustering
```

In [2]:
```
# Some functions
function my_weight_matrix(src::Array,dst::Array,weights::Array)
    n=nv(G)
    sparse([src;dst],[dst;src],[weights;weights],n,n)
end

my_laplacian(W::AbstractMatrix)=sparse(diagm(vec(sum(W,2))))-W

function my_normalized_laplacian(L::AbstractMatrix)
    D=1.0./sqrt.(diag(L))
    n=length(D)
    [L[i,j]*(D[i]*D[j]) for i=1:n, j=1:n]
end
```

Out[2]:
```
my_normalized_laplacian (generic function with 1 method)
```

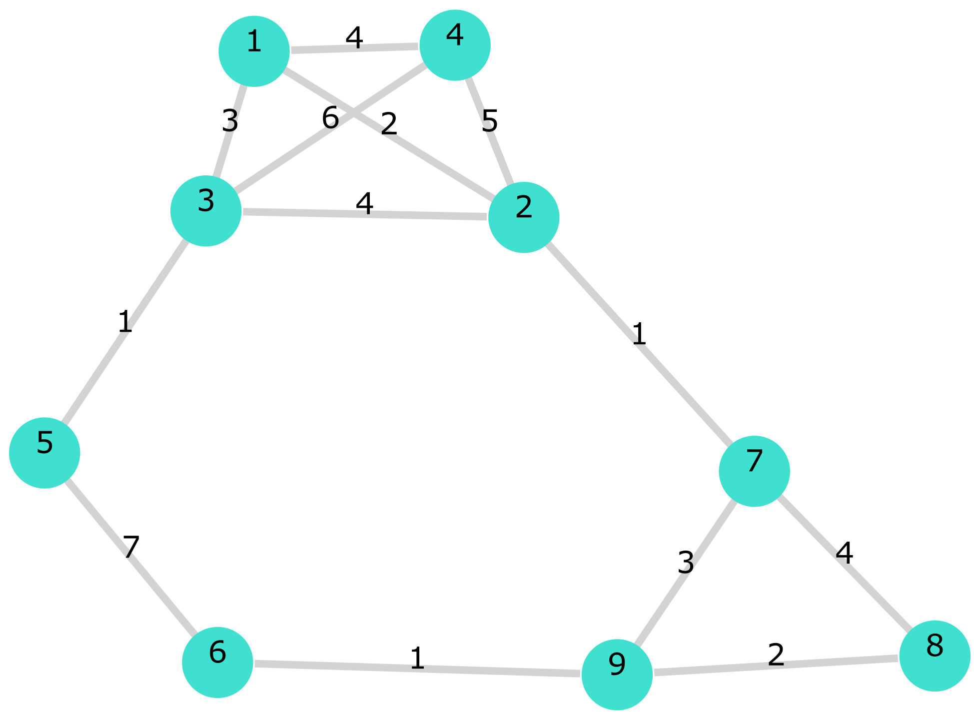
In [3]:
```
# Sources, targets, and weight
n=9
sn=[1,1,1,2,2,2,3,3,5,6,7,7,8]
tn=[2,3,4,3,4,7,4,5,6,9,8,9,9]
wn=[2,3,4,4,5,1,6,1,7,1,4,3,2]
[sn tn wn]
```

Out[3]:
```
13×3 Array{Int64,2}:
 1  2  2
 1  3  3
```

```
 1  4  4
 2  3  4
 2  4  5
 2  7  1
 3  4  6
 3  5  1
 5  6  7
 6  9  1
 7  8  4
 7  9  3
 8  9  2
```

```
In [4]: # What is the optimal tripartition?
        G=Graph(n)
        for i=1:length(sn)
            add_edge!(G,sn[i],tn[i])
        end
        gplot(G, nodelabel=1:n, edgelabel=wn)
```

```
Out[4]:
```




```
In [5]: W=my_weight_matrix(sn,tn,wn)
        L=my_laplacian(W)
        Ln=my_normalized_laplacian(L)
```

```
Out[5]: 9×9 Array{Float64,2}:
          1.0       -0.19245   -0.267261   ...   0.0        0.0        0.0
         -0.19245    1.0       -0.308607      -0.102062   0.0        0.0
         -0.267261  -0.308607   1.0            0.0        0.0        0.0
         -0.344265  -0.372678  -0.414039       0.0        0.0        0.0
          0.0        0.0       -0.0944911      0.0        0.0        0.0
          0.0        0.0        0.0        ...   0.0        0.0       -0.144338
          0.0       -0.102062   0.0            1.0       -0.57735   -0.433013
          0.0        0.0        0.0           -0.57735    1.0       -0.333333
          0.0        0.0        0.0           -0.433013  -0.333333   1.0
```

```
In [6]: full(L)
```

```
Out[6]: 9×9 Array{Int64,2}:
          9  -2  -3  -4   0   0   0   0   0
         -2  12  -4  -5   0   0  -1   0   0
         -3  -4  14  -6  -1   0   0   0   0
         -4  -5  -6  15   0   0   0   0   0
          0   0  -1   0   8  -7   0   0   0
          0   0   0   0  -7   8   0   0  -1
          0  -1   0   0   0   0   8  -4  -3
          0   0   0   0   0   0  -4   6  -2
          0   0   0   0   0  -1  -3  -2   6
```

```
In [7]: # Proportional cut. The clustering is visible in
        # the components of v_2 and v_3
        # λ,Y=eigs(L,nev=3,which=:SM)
        λ,Y=eig(full(L))
```

```
Out[7]: ([1.3076e-15, 0.788523, 1.21049, 7.92138, 11.145, 12.073, 14.9478, 17.0819, 20.8319], [-
```

```
In [8]: out=kmeans(Y[:,1:3]',3)
```

```
Out[8]: Clustering.KmeansResult{Float64}([-0.333333 -0.333333 -0.333333; -0.354766 0.392111 0.12
```

```
In [9]: # Normalized cut
        # Lanczos cannot be used for the "smallest in magnitude"
        # eienvalues of a singular matrix
        # λ,Y=eigs(Ln,nev=3,which=:SM)
        μ,Y=eig(Ln)
        Y=Y[:,1:3]
        D=sqrt.(diag(L))
        Y=diagm(1.0./D)*Y
        out=kmeans(Y',3)
```

```
Out[9]: Clustering.KmeansResult{Float64}([-0.107833 -0.107833 -0.107833; 0.0909339 -0.144522 -0.
```

1.3.4 Example - Concentric rings

```
In [10]: using Winston
         using Colors
         using Distances
```

```
In [11]: function plotKpartresult(C::Vector,X::Array,k::Int)
             p=FramedPlot()
             for j=1:k
                 # Random color
                 col=RGB(rand(),rand(),rand())
                 p1=Points(X[1,find(C.==j)],
                 X[2,find(C.==j)],"color",col,symbolkind="dot")
                 add(p,p1)
             end
             p
         end
```

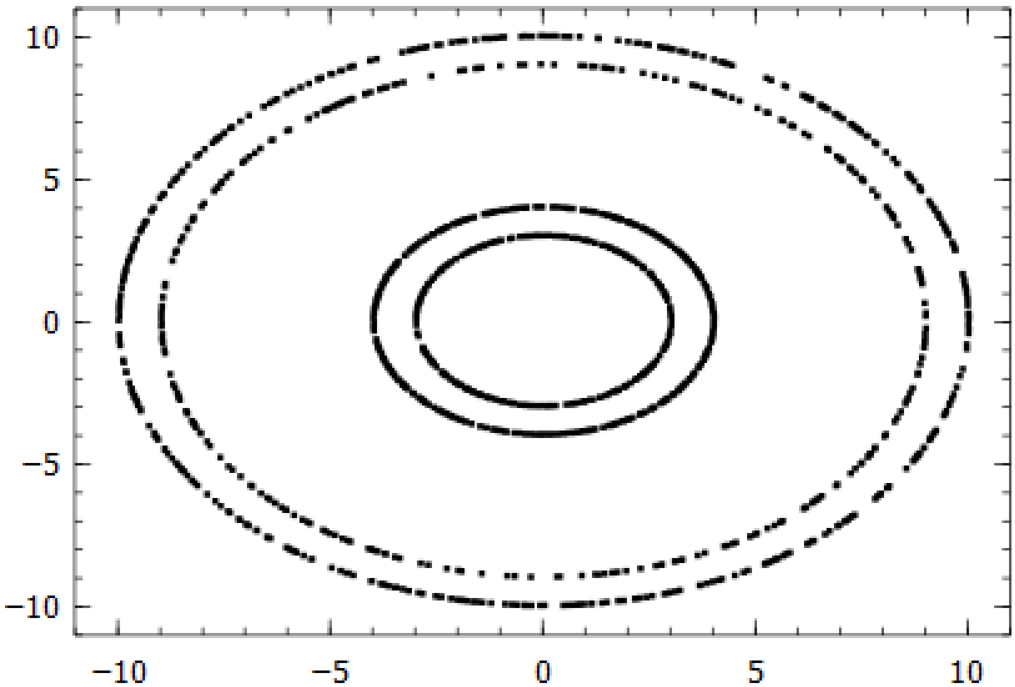
```
Out[11]: plotKpartresult (generic function with 1 method)
```

```
In [12]: # Generate concentric rings
         k=4
         # Center
         center=[0;0]
         # Radii
         radii=randperm(10)[1:k]
         # Number of points in circles
         sizes=rand(300:500,k)
         center,radii,sizes
```

```
Out[12]: ([0, 0], [10, 4, 3, 9], [459, 491, 329, 313])
```

```
In [13]: # Points
         m=sum(sizes)
         X=Array{Float64}(2,m)
         first=0
         last=0
         for j=1:k
             first=last+1
             last=last+sizes[j]
             # Random angles
             ϕ=2*π*rand(sizes[j])
             for i=first:last
                 l=i-first+1
                 X[:,i]=center+radii[j]*[cos(ϕ[l]);sin(ϕ[l])]+(rand(2)-0.5)/50
             end
         end
         Winston.plot(X[1,:],X[2,:],".")
```

Out[13]:



In [14]:
```
S=pairwise(SqEuclidean(),X)
# S=pairwise(Cityblock(),X)
β=1.0
```

Out[14]: 1.0

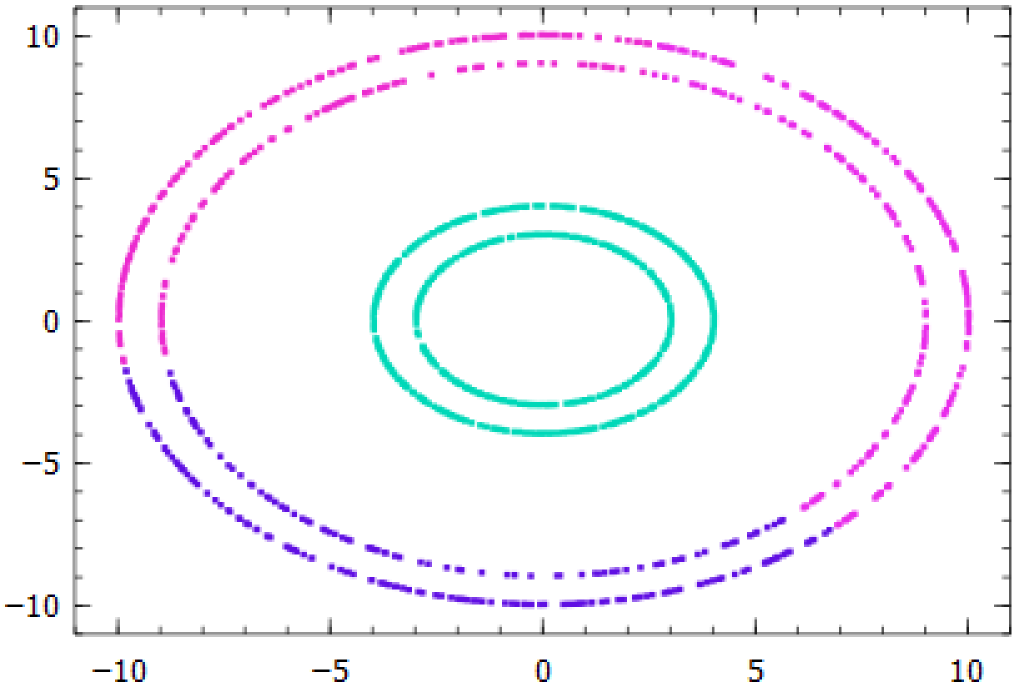
In [15]:
```
W=exp.(-β*S);
```

In [16]:
```
L=my_laplacian(W)
Ln=my_normalized_laplacian(L);
```

In [17]:
```
# Normalized Laplacian
λ,Y=eig(Ln)
sp=sortperm(abs.(λ))[1:k]
λ=λ[sp]
Y=Y[:,sp]
@show λ
Y=diagm(1.0./sqrt.(diag(L)))*Y
out=kmeans(Y',k)
plotKpartresult(out.assignments,X,k)
```

λ = [-3.82911e-16, 1.11076e-11, 0.00238225, 0.00251952]

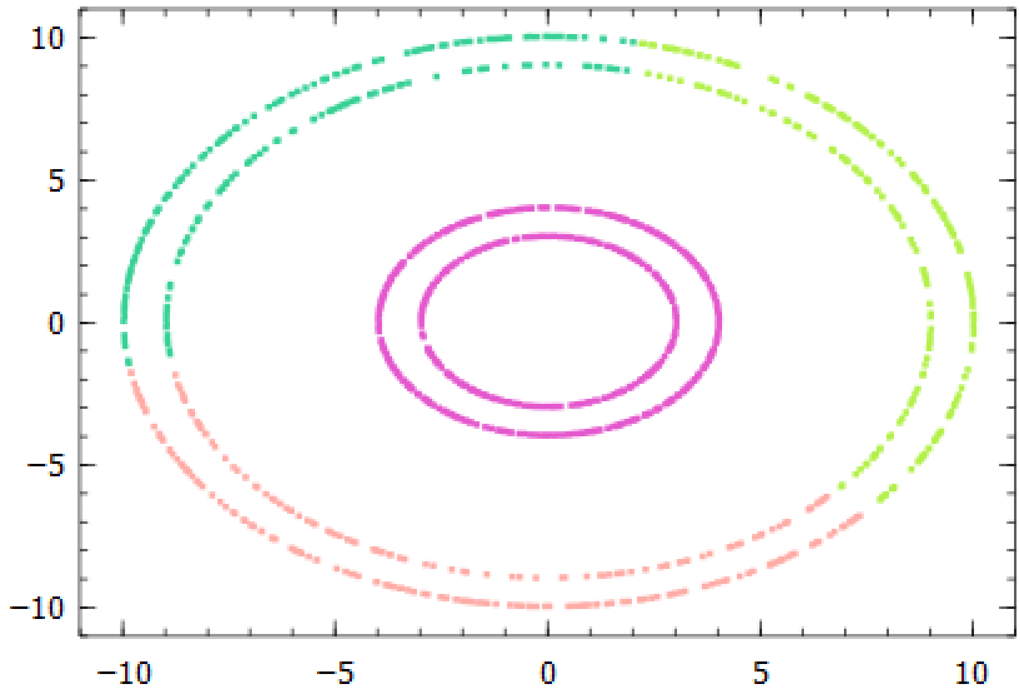
Out[17]:



In [18]:
```
# Laplacian
λ,Y=eig(L)
sp=sortperm(abs.(λ))[1:k]
λ=λ[sp]
Y=Y[:,sp]
@show λ
out=kmeans(Y',k)
plotKpartresult(out.assignments,X,k)
```

λ = [-4.92012e-15, 2.5294e-10, 0.0372006, 0.0390917]

Out[18]:



In [ ]: