# Lecture 6: Community Detection: Spectral Methods and SDP Relaxations

**Radu Balan**

Department of Mathematics, AMSC, CSCAMM and NWC
University of Maryland, College Park, MD

April 13, 2023

# Graph Partitions: Objective Functions

Assume a weighted graph given by the weight matrix $W$ (could be the adjacency matrix). The goal is to perform a disjoint partition into two clusters of the vertex set $\mathcal{V} = S \cup \bar{S}$ that had the largest total weight inside each cluster while maintaining a low cross-weight between clusters.

Two types of objective functions:

1. Min-edge type criterion (Rayleigh type criterion), or Cheeger constant:

$$h_G = \min_{S \subset \mathcal{V}} \frac{|E(S,\bar{S})|}{min(vol(S), vol(\bar{S}))}.$$

where $vol(S) = 1^T W 1_S = \sum_{i \in S} d_i$, $vol(\bar{S}) = 1^T W 1_{\bar{S}} = \sum_{i \in \bar{S}} d_i$, $|E(S,\bar{S})| = 1_S^T W 1_{\bar{S}}$.

2. Modularity function, fraction of the edges that fall within the communities minus the expected fraction if edges were distributed at random (unweighted case):

$$\max_{S \subset \mathcal{V}} \frac{1}{2m} \sum_{(i,j) \in (S \times S) \cup (\bar{S} \times \bar{S})} \left( A_{i,j} - \frac{d_i d_j}{2m} \right) \quad , \quad d_i = \sum_k A_{i,k}.$$

# Optimization Problems

## Second Smallest for $\tilde{\Delta}$

The Algorithm is supposed to provide an approximate solution for the min-edge cut problem of the Cheeger constant

$$h_G = \min_{S \subset \mathcal{V}} \frac{|E(S, \bar{S})|}{min(vol(S), vol(\bar{S}))}.$$

The algorithm has been derived while proving the bound $2h_G \geq \lambda_1$. Implicitely, the second smallest eigenpair solves the optimization problem:

$$\begin{array}{cc} \min & e^T \tilde{\Delta} e \\ e \in \mathbb{R}^n & \\ \|e\|_2 = 1 & \\ e^T D^{1/2} 1 = 0 & \end{array}$$

# Spectral Algorithm using the Symmetric Normalized Graph Laplacian

## Algorithm (Spectral Algorithm with $\tilde{\Delta}$)

**Input**: *Adjacency matrix* $A \in \mathbb{R}^{n \times n}$.
*If the graph is not connected then produce a disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *into connected components.*
*Else:*

1. *Compute the symmetric normalized graph Laplacian* $\tilde{\Delta} = I - D^{-1/2} A D^{-1/2}$, *with* $D = Diag(A \cdot 1)$ *the degree matrix.*
2. *Compute the second smallest eigenpair:* $(e_1, \lambda_1)$, *with* $\tilde{\Delta} e_1 = \lambda_1 e_1$ *and* $\lambda_1 > 0 = \lambda_0$.
3. *Define the partition* $\Omega_1 = \{k : e_1(k) > 0\}$, $\Omega_2 = \{k : e_1(k) \leq 0\}$. *Set* $d = 2$.

**Output**: *The disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *of the set of nodes* $[n] = \{1, 2, \cdots, n\}$.

# Optimization Problems

## MAP and MLE for Balanced Communities

Consider now a slightly different optimization problem. Assume we know we have a symmetric stochastic block model $SSBM(n, 2, a, b)$ with two communities of equal size: $|\Omega_1| = |\Omega_2|$. Then the Maximum A Posteriori (MAP) partition function $Z \in \{1, 2\}^n$ coincides with the Maximum Likelihood Estimator (MLE) and maximizes:

$$\max_{Z:|\Omega_1|=|\Omega_2|} a^{m_{11}+m_{22}}(1-a)^{m_{11}^c+m_{22}^c} b^{m_{12}}(1-b)^{m_{12}^c}$$

But for equal size communities (== balanced communities),

$m_{12} + m_{12}^c = \frac{n^2}{4}$ and $m_{11} + m_{22} + m_{11}^c + m_{22}^c = 2\begin{pmatrix} n/2 \\ 2 \end{pmatrix} \approx \frac{n^2}{4}$.

Furthermore, $m_{11} + m_{12} + m_{22} = m$. Thus, the optimal estimator maximizes:

$$\max_{Z:|\Omega_1|=|\Omega_2|} \left(\frac{a(1-b)}{b(1-a)}\right)^{m_{11}+m_{22}}$$

# Optimization Problems

## MAP and MLE for Balanced Communities

Assume $a > b$. Then $\frac{a(1-b)}{b(1-a)} > 1$ and maximization of $\left(\frac{a(1-b)}{b(1-a)}\right)^{m_{11}+m_{22}}$ is equivalent to maximization of the number of intra-edges while have balanced communities.

$$\max_{Z:|\Omega_1|=|\Omega_2|} \quad m_{11} + m_{22}$$

Equivalently, since $m_{11} + m_{22} + m_{12} = m$ and is invariant to any partition, the solution minimizes the number of cross-edges $m_{12}$ subject to balanced communities:

$$\min_{Z:|\Omega_1|=|\Omega_2|} \quad m_{12}$$

# Optimization Problems

## MAP and MLE for Balanced Communities (2)

Replace the partition vector $Z \in \{1,2\}^n$ with a sign vector $z \in \{-1,1\}^n$ so that $Z_k = 1$ iff $z_k = -1$ and $Z_k = 2$ iff $z_k = +1$. Then

$$z^T A z = \sum_{i,j=1}^{n} A_{i,j} z_i z_j = 2(m_{11}+m_{22})-2m_{12} = 4(m_{11}+m_{22})-2m = 2m-4m_{12}$$

Thus

$$m_{11} + m_{22} = \frac{1}{4} z^T A z + \frac{m}{2}$$

and the number of cross-edges can be computed using:

$$m_{12} = \frac{1}{4}(2m - z^T A z) = \frac{1}{4}(z^T D z - z^T A z) = \frac{1}{4} z^T \Delta z$$

because $z^T D z = 1^T D 1 = \sum_{i,j=1}^{n} A_{i,j} = 2m$.

# The Quadratic Integer Programs

Balanced communities: $|\Omega_1| = |\Omega_2|$ is equivalent to requiring $z^T \cdot 1 = 0$. Thus we obtain the following optimization problems:

1. Graph Laplacian based Minimization:

$$\min_{\substack{z \in \{-1,+1\}^n \\ z^T \cdot 1 = 0}} \quad z^T \Delta z$$

2. Adjacency Matrix based Maximization:

$$\max_{\substack{z \in \{-1,+1\}^n \\ z^T \cdot 1 = 0}} \quad z^T A z$$

These are NP-hard problems, known as Quadratic Integer Programming. We study two relaxations: Euclidean relaxation, and SDP relaxation.

# Euclidean Relaxations

The Euclidean relaxation of the QIP

$$\min / \max_{\substack{z\in\{-1,+1\}^n \\ z^T\cdot 1=0}} z^T S z$$

is obtained by replacing $z \in \{-1,+1\}^n$ with $\|z\|_2 = \sqrt{n}$. Here $S = S^T$ stands for $\Delta$ or $A$. Since different norm values produce same solution up to scaling, we use instead the unit Euclidean norm relaxation:

$$\min / \max_{\substack{\|z\|_2=1 \\ z^T\cdot 1=0}} z^T S z$$

# Spectral Algorithms

Using the Courant-Fisher criterion (related also to the Rayleigh quotient), the Euclidean relaxation is solved using the second eigenvector of the corresponding symmetric matrix.

Why the second eigenvector:

1. In the case of $\tilde{\Delta}$, $1$ is the eigenvector corresponding to the smallest eigenvalue ($\lambda_0 = 0$), hence $z^T 1 = 0$ is satisfied automatically by the second eigenvector.
2. In the case of $A$, $1$ is approximately the leading eigenvector asuming each node has the same valence. This happens when the adjacency matrix approximates well its Expected value matrix $\mathbb{E}[A]$. Note: One can solve exactly (no approximation needed) the optimization problem $max\ z^T Az$ subject to $\|z\|_2 = 1$ and $z^T 1 = 0$. The solution is the normalized eigenvector associated to the largest eigenvalue of $(I - \frac{1}{n}11^T)A(I - \frac{1}{n}11^T)$.

# Spectral Algorithm using the Graph Laplacian

## Algorithm (Spectral Algorithm with $\Delta$)

**Input**: *Adjacency matrix* $A \in \mathbb{R}^{n\times n}$.

*If the graph is not connected then produce a disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *into connected components.*

*Else:*

1. *Compute the graph Laplacian* $\Delta = D - A$, *with* $D = Diag(A \cdot 1)$, *the degree matrix.*
2. *Compute the second smallest eigenpair:* $(e_1, \lambda_1)$, *with* $\Delta e_1 = \lambda_1 e_1$ *and* $\lambda_1 > 0 = \lambda_0$.
3. *Define the partition* $\Omega_1 = \{k : e_1(k) > 0\}$, $\Omega_2 = \{k : e_1(k) \leq 0\}$. *Set* $d = 2$.

**Output**: *The disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *of the set of nodes* $[n] = \{1, 2, \cdots, n\}$.

# Spectral Algorithm using the Adjacency Matrix

## Algorithm (Spectral Algorithm with $A$)

**Input**: *Adjacency matrix* $A \in \mathbb{R}^{n \times n}$.
*If the graph is not connected then produce a disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *into connected components.*
*Else:*

1. *Compute the second largest eigenpair of* $A$*:* $(f_2, \mu_2)$*, with* $Af_2 = \mu_2 f_2$*.*
2. *Define the partition* $\Omega_1 = \{k : f_2(k) > 0\}$, $\Omega_2 = \{k : f_2(k) \leq 0\}$*. Set* $d = 2$*.*

**Output**: *The disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *of the set of nodes* $[n] = \{1, 2, \cdots, n\}$*.*

# The SDP Relaxation

The Semi-Definite Program (SDP) relaxation of the QIP

$$\min / \max_{\substack{z \in \{-1,+1\}^n \\ z^T \cdot 1 = 0}} z^T S z$$

is obtained in the following way: First one replaces the variable vector $z$ by the matrix $Y \in \mathbb{R}^{n \times n}$, $Y = zz^T$. Note:

$$z^T S z = trace(z^T S z) = trace(Szz^T) = trace(SY)$$

The constraints $z \in \{-1,+1\}^n$ is equivalent to $Y_{ii} = 1$. The constraint $z^T \cdot 1 = 0$ is equivalent to $Y \cdot 1 = 0$. Additionally, the matrix $Y$ satisfies also: $Y \geq 0$ and $rank(Y) = 1$.

# The SDP Relaxation - 2

Putting together all conditions, we obtain the (equivalent!) problem:

$$\min / \max_{\substack{Y = Y^T \geq 0 \\ rank(Y) = 1 \\ Y_{ii} = 1 \ , \ 1 \leq i \leq n \\ Y \cdot 1 = 0}} trace(SY)$$

# The SDP Relaxation - 2

Putting together all conditions, we obtain the (equivalent!) problem:

$$\min / \max_{\substack{Y = Y^T \geq 0 \\ rank(Y) = 1 \\ Y_{ii} = 1 \ , \ 1 \leq i \leq n \\ Y \cdot 1 = 0}} trace(SY)$$

However this problem is not convex, due to the rank constraint. The convex relaxation, known as the *SDP relaxation*, simply removes the rank constraint:

$$\min / \max_{\substack{Y = Y^T \geq 0 \\ Y_{ii} = 1 \ , \ 1 \leq i \leq n \\ Y \cdot 1 = 0}} trace(SY)$$

In general the result $Y$ is not rank 1, so one approximates it by the leading eigenvector of solution $\hat{Y}$. Note, for $Y = Y^T \geq 0$, $Y \cdot 1 = 0$ is equivalent to $1^T Y 1 = 0$.

# The Graph Laplacian SDP

## Algorithm (SDP with $\Delta$)

**Input**: *Adjacency matrix* $A \in \mathbb{R}^{n \times n}$.
*If the graph is not connected then produce a disjoint partition* $(\Omega_1, \Omega_2, \ldots, \Omega_d)$ *into connected components.*
*Else:*

1. *Compute the graph Laplacian* $\Delta = D - A$, *with* $D = Diag(A \cdot 1)$, *the degree matrix.*
2. *Solve the Semi-Definite Program:*

$$\begin{array}{cc} \min_{Y} & trace(\Delta Y) \\ \text{subject to} & \\ Y = Y^T \geq 0 & \\ Y_{ii} = 1 \ , \ 1 \leq i \leq n & \\ 1^T \cdot Y \cdot 1 = 0 & \end{array}$$

# The Graph Laplacian SDP

## Algorithm (SDP with $\Delta$ - continued)

3. *Find the leading eigenvector of* $Y$*,* $(e_{max}, \sigma_{max})$*, i.e.,* $Ye_{max} = \sigma_{max} e_{max}$.
4. *Define the partition* $\Omega_1 = \{k : e_{max}(k) > 0\}$*,* $\Omega_2 = \{k : e_{max}(k) \leq 0\}$*. Set* $d = 2$*.*

**Output**: *The disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *of the set of nodes* $[n] = \{1, 2, \cdots, n\}$.

# The Adjacency Matrix SDP

## Algorithm (SDP with $A$)

**Input**: *Adjacency matrix* $A \in \mathbb{R}^{n \times n}$.
*If the graph is not connected then produce a disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *into connected components.*
*Else:*

1. *Solve the Semi-Definite Program:*

$$\begin{array}{cc} \max\limits_{Y \text{ subject to}} & trace(AY) \\ Y = Y^T \geq 0 & \\ Y_{ii} = 1 \ , \ 1 \leq i \leq n & \\ 1^T \cdot Y \cdot 1 = 0 & \end{array}$$

2. *Find the leading eigenvector of* $Y$, $(e_{max}, \sigma_{max})$, *i.e.,* $Ye_{max} = \sigma_{max} e_{max}$.

# The Adjacency Matrix SDP

## Algorithm (SDP with $A$ - continued)

3. *Define the partition* $\Omega_1 = \{k : e_{max}(k) > 0\}$, $\Omega_2 = \{k : e_{max}(k) \leq 0\}$. *Set* $d = 2$.

**Output**: *The disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *of the set of nodes* $[n] = \{1, 2, \cdots, n\}$.

# The Normalized Graph Laplacian SDP

## Algorithm (SDP with $\tilde{\Delta}$)

**Input**: *Adjacency matrix* $A \in \mathbb{R}^{n \times n}$.

*If the graph is not connected then produce a disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *into connected components.*

*Else:*

1. *Compute the symmetric normalized graph Laplacian* $\tilde{\Delta} = I - D^{-1/2} A D^{-1/2}$, *with* $D = Diag(A \cdot 1)$, *the degree matrix.*
2. *Solve the Semi-Definite Program:*

$$\min_{Y \text{ subject to}} \quad trace(\tilde{\Delta} Y)$$

$$Y = Y^T \geq 0$$

$$Y_{ii} = 1 \ , \ 1 \leq i \leq n$$

$$1^T \cdot Y \cdot 1 = 0$$

# The Normalized Graph Laplacian SDP

## Algorithm (SDP with $\tilde{\Delta}$ - continued)

3. *Find the leading eigenvector of* $Y$, $(e_{max}, \sigma_{max})$, *i.e.,* $Ye_{max} = \sigma_{max} e_{max}$.
4. *Define the partition* $\Omega_1 = \{k : e_{max}(k) > 0\}$, $\Omega_2 = \{k : e_{max}(k) \leq 0\}$. *Set* $d = 2$.

**Output**: *The disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *of the set of nodes* $[n] = \{1, 2, \cdots, n\}$.

This is the SDP counterpart of the spectral algorithm we studied last time.

# Partitions of Weighted Graphs

In this section we rewrite all the previous algorithms in the case of weighted graphs.
The idea: The Cheeger constant is simply replaced by total cross-weight between partitions:

$$h_G = \min_S \frac{\sum_{x\in S, y\in\bar{S}} W_{x,y}}{min(\sum_{x\in S} D_{x,x}, \sum_{y\in\bar{S}} D_{y,y})} \quad , \quad D_{i,i} = \sum_j W_{i,j}$$

Solution: replace the adjacency matrix $A$ by the weight matrix $W$.
Thus we obtain a total of six algorithms: 3 spectral algorithms, and 3 SDP relaxations; each class using either$I - D^{-1/2}WD^{-1/2}$ , $D - W$, or $W$.

# Spectral Algorithm using the symmetric normalized Weighted Graph Laplacian

## Algorithm (Spectral Algorithm with symmetric normalized weighted graph Laplacian $\tilde{\Delta}$)

**Input**: *Weight matrix* $W \in \mathbb{R}^{n \times n}$.

*If the graph is not connected then produce a disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *into connected components.*

*Else:*

1. *Compute the symmetric normalized weighted graph Laplacian* $\tilde{\Delta} = I - D^{-1/2} W D^{-1/2}$, *with* $D = Diag(W \cdot 1)$.
2. *Compute the second smallest eigenpair:* $(e_1, \lambda_1)$, *with* $\Delta e_1 = \lambda_1 e_1$ *and* $\lambda_1 > 0 = \lambda_0$.
3. *Define the partition* $\Omega_1 = \{k : e_1(k) > 0\}$, $\Omega_2 = \{k : e_1(k) \leq 0\}$. *Set* $d = 2$.

**Output**: *Disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *of nodes* $[n] = \{1, 2, \cdots, n\}$.

# Spectral Algorithm using the Weighted Graph Laplacian

## Algorithm (Spectral Algorithm with weighted $\Delta$)

**Input**: *Weight matrix* $W \in \mathbb{R}^{n \times n}$.

*If the graph is not connected then produce a disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *into connected components.*

*Else:*

1. *Compute the weighted graph Laplacian* $\Delta = D - W$, *with* $D = Diag(W \cdot 1)$.
2. *Compute the second smallest eigenpair:* $(e_1, \lambda_1)$, *with* $\Delta e_1 = \lambda_1 e_1$ *and* $\lambda_1 > 0 = \lambda_0$.
3. *Define the partition* $\Omega_1 = \{k : e_1(k) > 0\}$, $\Omega_2 = \{k : e_1(k) \leq 0\}$. *Set* $d = 2$.

**Output**: *The disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *of the set of nodes* $[n] = \{1, 2, \cdots, n\}$.

# Spectral Algorithm using the Weight Matrix

## Algorithm (Spectral Algorithm with $W$)

**Input**: *Weight matrix* $W \in \mathbb{R}^{n \times n}$.

*If the graph is not connected then produce a disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *into connected components.*

*Else:*

1. *Compute the second largest eigenpair of* $W$: $(f_2, \mu_2)$, *with* $Wf_2 = \mu_2 f_2$.
2. *Define the partition* $\Omega_1 = \{k : f_2(k) > 0\}$, $\Omega_2 = \{k : f_2(k) \leq 0\}$. *Set* $d = 2$.

**Output**: *The disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *of the set of nodes* $[n] = \{1, 2, \cdots, n\}$.

# The Normalized weighted Graph Laplacian SDP

## Algorithm (SDP with weighted $\tilde{\Delta}$)

**Input**: *Weight matrix* $W \in \mathbb{R}^{n \times n}$.
*If the graph is not connected then produce a disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *into connected components.*
*Else:*

1. *Compute the symmetric normalized weighted graph Laplacian* $\tilde{\Delta} = I - D^{-1/2} W D^{-1/2}$, *with* $D = Diag(W \cdot 1)$.
2. *Solve the Semi-Definite Program:*

$$\begin{array}{cc} \min\limits_{Y} & trace(\tilde{\Delta} Y) \\ \text{subject to} & \\ Y = Y^T \geq 0 & \\ Y_{ii} = 1 \ , \ 1 \leq i \leq n & \\ 1^T \cdot Y \cdot 1 = 0 & \end{array}$$

# The Normalized weighted Graph Laplacian SDP

## Algorithm (SDP with weighted $\tilde{\Delta}$ - continued)

3. *Find the leading eigenvector of $Y$, $(e_{max}, \sigma_{max})$, i.e., $Ye_{max} = \sigma_{max} e_{max}$.*
4. *Define the partition $\Omega_1 = \{k : e_{max}(k) > 0\}$, $\Omega_2 = \{k : e_{max}(k) \leq 0\}$. Set $d = 2$.*

**Output**: *The disjoint partition $(\Omega_1, \Omega_2, ..., \Omega_d)$ of the set of nodes $[n] = \{1, 2, \cdots, n\}$.*

# The weighted Graph Laplacian SDP

## Algorithm (SDP with weighted $\Delta$)

**Input**: *Weight matrix* $W \in \mathbb{R}^{n \times n}$.
*If the graph is not connected then produce a disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *into connected components.*
*Else:*

1. *Compute the weighted graph Laplacian* $\Delta = D - W$, *with* $D = Diag(W \cdot 1)$.
2. *Solve the Semi-Definite Program:*

$$\begin{array}{cc} \min & trace(\Delta Y) \\ Y \text{ subject to} & \\ Y = Y^T \geq 0 & \\ Y_{ii} = 1 \ , \ 1 \leq i \leq n & \\ 1^T \cdot Y \cdot 1 = 0 & \end{array}$$

# The weighted Graph Laplacian SDP

## Algorithm (SDP with weighted $\Delta$ - continued)

3. *Find the leading eigenvector of $Y$, $(e_{max}, \sigma_{max})$, i.e., $Ye_{max} = \sigma_{max} e_{max}$.*
4. *Define the partition $\Omega_1 = \{k : e_{max}(k) > 0\}$, $\Omega_2 = \{k : e_{max}(k) \leq 0\}$. Set $d = 2$.*

**Output**: *The disjoint partition $(\Omega_1, \Omega_2, ..., \Omega_d)$ of the set of nodes $[n] = \{1, 2, \cdots, n\}$.*

# The Weight Matrix SDP

## Algorithm (SDP with $W$)

**Input**: *Weight matrix* $W \in \mathbb{R}^{n\times n}$.
*If the graph is not connected then produce a disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *into connected components.*
*Else:*

1. *Solve the Semi-Definite Program:*

$$\begin{array}{cc} \max\limits_{Y} & trace(WY) \\ \text{subject to} & \\ Y = Y^T \geq 0 & \\ Y_{ii} = 1 \; , \; 1 \leq i \leq n & \\ 1^T \cdot Y \cdot 1 = 0 & \end{array}$$

2. *Find the leading eigenvector of* $Y$, $(e_{max}, \sigma_{max})$, *i.e.,* $Ye_{max} = \sigma_{max} e_{max}$.

# The Weight Matrix SDP

**Algorithm (SDP with $W$ - continued)**

3. *Define the partition* $\Omega_1 = \{k : e_{max}(k) > 0\}$, $\Omega_2 = \{k : e_{max}(k) \leq 0\}$. *Set* $d = 2$.

**Output**: *The disjoint partition* $(\Omega_1, \Omega_2, ..., \Omega_d)$ *of the set of nodes* $[n] = \{1, 2, \cdots, n\}$.

# Measures of Partition Accuracy

Problem: How to measure the quality of a given partition?
We previously studied:

## Definition

*The agreement between two community vectors $x, y \in [k]^n$ is obtained by maximizing the number of common components of these two vectors over all possible relabelling (i.e., permutations):*

$$Agr(x, y) = \frac{1}{n} \max_{\pi \in S_k} \sum_{i=1}^{n} \mathbf{1}(x_i = \pi(y_i))$$

*where $S_k$ denotes the group of permutations.*

# Measures of Partition Accuracy (2)

In the case of 2-community detection, the above formula reduces to:

$$Agr(x,y) = \frac{1}{n}\max\left(\sum_{i=1}^{n} \mathbf{1}(x_i = y_i), \sum_{i=1}^{n} \mathbf{1}(x_i \neq y_i)\right) = \frac{1}{n} max(\alpha, n-\alpha)$$

where

$$\alpha = \sum_{i=1}^{n} \mathbf{1}(x_i = y_i).$$

measures the overlap. Typically it is more appropriate to report the percentage agreement:

$$Agr[\%] = 100\, max(\frac{\alpha}{n}, 1-\frac{\alpha}{n}).$$

Note the agreement is always larger than or equal to 50%. In the case of $k$ communities, the previous formula involves taking maximum over $k!$ possible label assignments.

# Convex Sets. Convex Functions

A set $S \subset \mathbb{R}^n$ is called a *convex set* if for any points $x, y \in S$ the line segment $[x, y] := \{tx + (1-t)y \ , \ 0 \leq t \leq 1\}$ is included in $S$, $[x, y] \subset S$.

A function $f : S \to \mathbb{R}$ is called *convex* if for any $x, y \in S$ and $0 \leq t \leq 1$, $f(t\,x + (1-t)y) \leq t\,f(x) + (1-t)f(y)$.
Here $S$ is supposed to be a convex set in $\mathbb{R}^n$.
Equivalently, $f$ is convex if its epigraph is a convex set in $\mathbb{R}^{n+1}$. Epigraph: $\{(x, u) \ ; \ x \in S, u \geq f(x)\}$.

A function $f : S \to \mathbb{R}$ is called *strictly convex* if for any $x \neq y \in S$ and $0 < t < 1$, $f(t\,x + (1-t)y) < t\,f(x) + (1-t)f(y)$.

# Convex Optimization Problems

The general form of a convex optimization problem:

$$\min_{x\in S} f(x)$$

where $S$ is a closed convex set, and $f$ is a convex function on $S$.
Properties:

1. Any local minimum is a global minimum. The set of minimizers is a convex subset of $S$.
2. If $f$ is strictly convex, then the minimizer is unique: there is only one local minimizer.

In general $S$ is defined by equality and inequality constraints:
$S = \{g_i(x) \leq 0 \ , \ 1 \leq i \leq p\} \cap \{h_j(x) = 0 \ , \ 1 \leq j \leq m\}$. Typically $h_j$ are required to be affine: $h_j(x) = a^T x + b$.

# Convex Programs

The hiarchy of convex optimization problems:

1. Linear Programs: Linear criterion with linear constraints
2. Quadratic Programs: Quadratic Criterion with Linear Constraints; Quadratically Constrained Quadratic Problems (QCQP); Second-Order Cone Program (SOCP)
3. Semi-Definite Programs(SDP)

Typical SDP:

$$\begin{array}{cc} \min\limits_{X = X^T \geq 0} & trace(XA) \\ trace(XB_k) = y_k \ , \ 1 \leq k \leq p & \\ trace(XC_j) \leq z_j \ , \ 1 \leq j \leq m & \end{array}$$

# CVX

## Matlab package

Downloadable from: http://cvxr.com/cvx/ . Follows "Disciplined" Convex Programming – à la Boyd [2].

```
m = 20; n = 10; p = 4;
A = randn(m,n); b = randn(m,1);
C = randn(p,n); d = randn(p,1); e = rand;
cvx_begin
    variable x(n)
    minimize( norm( A * x - b, 2 ) )
    subject to
        C * x == d
        norm( x, Inf ) <= e
cvx_end
```

$$\begin{array}{cc} \min & \|Ax-b\| \\ Cx = d & \\ \|x\|_\infty \le e & \end{array}$$

# CVX
## SDP Example

```
cvx_begin sdp

    variable X(n,n) semidefinite;
    minimize trace(X);
    subject to
    X*ones(n,1) == zeros(n,1);
    abs(trace(E1*X)-d1)<=epsx;
    abs(trace(E2*X)-d2)<=epsx;

cvx_end
```

$$
\begin{array}{ll}
\text{minimize} & trace(X) \\
\text{subject to} & X = X^T \geq 0 \\
 & X \cdot 1^T = 0 \\
 & |trace(E_1 X) - d_1| \leq \varepsilon \\
 & |trace(E_2 X) - d_2| \leq \varepsilon
\end{array}
$$

## References

E. Abbe, Community detection and stochastic block models: recent developments, arXiv:1703.10146 [math.PR] 29 Mar. 2017.

S. Boyd, L. Vandenberghe, **Convex Optimization**, available online at: http://stanford.edu/ boyd/cvxbook/