# C&O 355
# Mathematical Programming
# Fall 2010
# Lecture 16

[N. Harvey]

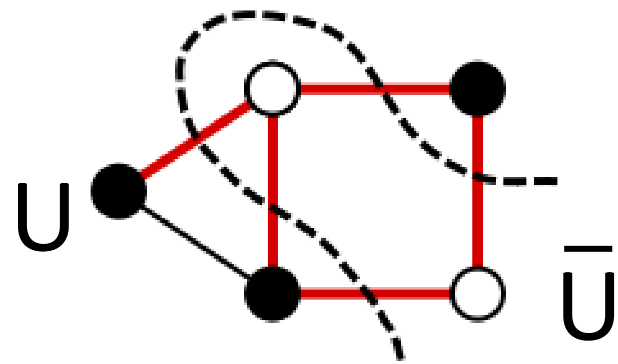
# Topics

- Semidefinite Programs (SDP)
- Vector Programs (VP)
- Quadratic Integer Programs (QIP)
- QIP & SDP for Max Cut
- Finding a cut from the SDP solution
- Analyzing the cut

# The Max Cut Problem

Our first foray into combinatorial optimization

- Let G=(V,E) be a graph with n vertices.
  For $U \subseteq V$, let $\delta(U) = \{ \{u,v\} : u \in U, v \notin U \}$
  Find a set $U \subseteq V$ such that $|\delta(U)|$ is maximized.




- This is a **computationally hard** problem:
  it cannot be solved exactly. (Unless P = NP)
- Our only hope: find a **nearly-optimal** solution,
  i.e., a **big cut** that might not be maximum.
- **Philosophy:** How can our powerful continuous optimization tools help to solve combinatorial problems?

# My Example Data

- Here is (a portion of) the adjacency matrix of a graph with 750 vertices, 3604 edges
- Probably cannot find the max cut before the end of the universe
- Can we find a big cut in this example?

# Semidefinite Programs

$$\begin{aligned} \max \quad & c^{\mathsf{T}} x \\ \text{s.t.} \quad & Ax = b \\ & y^{\mathsf{T}} X y \geq 0 \;\; \forall y \in \mathbb{R}^d \end{aligned}$$

- Where
  - $x \in \mathbb{R}^n$ is a vector and $n = d(d+1)/2$
  - A is a $m \times n$ matrix, $c \in \mathbb{R}^n$ and $b \in R^m$
  - X is a $d \times d$ symmetric matrix, and x is the vector corresponding to X.
- There are **infinitely many** constraints!

# Vectorizing a Matrix

- A dxd matrix can be viewed as a vector in $\mathbb{R}^{d^2}$. (Just write down the entries in some order.)

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

- A dxd **symmetric** matrix can be viewed as a vector in $\mathbb{R}^{d(d+1)/2}$.

$$X = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \Rightarrow x = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}$$

- **Our notation:** X is a dxd symmetric matrix, and x is the corresponding vector.

# PSD matrices $\equiv$ Vectors in $\mathbb{R}^d$

- **Key Observation:** PSD matrices correspond directly to vectors and their dot-products.

- $\rightarrow$: Given vectors $v_1,\ldots,v_d$ in $\mathbb{R}^d$,
  let V be the d×d matrix whose $i^{th}$ column is $v_i$.
  Let $X = V^T V$. Then X is PSD and $X_{i,j} = v_i^T v_j \ \forall i,j$.

- $\leftarrow$: Given a d×d PSD matrix X, find spectral decomposition $X = U D U^T$, and let $V = D^{1/2} U$.
  To get vectors in $\mathbb{R}^d$, let $v_i$ = $i^{th}$ column of V.
  Then $X = V^T V \ \Rightarrow \ X_{i,j} = v_i^T v_j \ \forall i,j$.

# Vector Programs

- A Semidefinite Program:

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & y^\top X y \geq 0 \;\; \forall y \in \mathbb{R}^d \end{aligned}$$

- Equivalent definition as "vector program"

$$\begin{aligned} \max \quad & \sum_{i=1}^{d} \sum_{j=1}^{d} c_{i,j} v_i^\top v_j & & \\ \text{s.t.} \quad & \sum_{i=1}^{d} \sum_{j=1}^{d} a_{k,i,j} v_i^\top v_j & = b_k & \qquad \forall k = 1, \ldots, m \\ & v_1, \ldots, v_d & \in \mathbb{R}^d & \end{aligned}$$

# Integer Programs

- Our usual Integer Program

$$\begin{aligned} \max \quad & \sum_{i=1}^{d} c_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{d} a_{k,i} x_i = b_k \qquad \forall k = 1, \ldots, m \\ & x_1, \ldots, x_d \in \{0, 1\} \end{aligned}$$

There are no efficient, general-purpose algorithms for solving IPs, assuming P≠NP.

- Quadratic Integer Program

$$\begin{aligned} \max \quad & \sum_{i=1}^{d} \sum_{j=1}^{d} c_{i,j} x_i x_j \\ \text{s.t.} \quad & \sum_{i=1}^{d} \sum_{j=1}^{d} a_{k,i,j} x_i x_j = b_k \qquad \forall k = 1, \ldots, m \\ & x_1, \ldots, x_d \in \{-1, 1\} \end{aligned}$$

Let's make things even harder: Quadratic Objective Function & Quadratic Constraints!

Could also use {0,1} here. {-1,1} is more convenient.

# QIPs & Vector Programs

- Quadratic Integer Program

(QIP)
$$\begin{array}{llll} \max & \sum_{i=1}^{d}\sum_{j=1}^{d} c_{i,j}x_i x_j & & \\ \text{s.t.} & \sum_{i=1}^{d}\sum_{j=1}^{d} a_{k,i,j}x_i x_j & = b_k & \forall k = 1, \ldots, m \\ & x_1, \ldots, x_d & \in \{-1, 1\} & \end{array}$$

- Vector Programs give a natural **relaxation**:

(VP)
$$\begin{array}{llll} \max & \sum_{i=1}^{d}\sum_{j=1}^{d} c_{i,j}v_i^{\mathsf{T}} v_j & & \\ \text{s.t.} & \sum_{i=1}^{d}\sum_{j=1}^{d} a_{k,i,j}v_i^{\mathsf{T}} v_j & = b_k & \forall k = 1, \ldots, m \\ & v_i^{\mathsf{T}} v_i & = 1 & \forall i = 1, \ldots, d \\ & v_1, \ldots, v_d & \in \mathbb{R}^d & \end{array}$$

- **Why is this a relaxation?** If we added constraint $v_i \in \{(-1,0,\ldots,0), (1,0,\ldots,0)\}$ $\forall i$, then VP is equivalent to QIP

# QIP for Max Cut

- Let G=(V,E) be a graph with n vertices.
  For $U \subseteq V$, let $\delta(U)$ = { {u,v} : $u \in U$, $v \notin U$ }
  Find a set $U \subseteq V$ such that $|\delta(U)|$ is maximized.
- Make a variable $x_u$ for each $u \in V$

$$\text{(QIP)} \quad \begin{array}{lll} \max & \displaystyle\sum_{\{u,w\}\in E} \tfrac{1}{2}(1 - x_u x_w) & \\ \text{s.t.} & x_u \in \{-1,1\} & \forall u \in V \end{array}$$

- **Claim:** Given feasible solution x, let U = { u : $x_u$ = -1 }.
  Then $|\delta(U)|$ = objective value at x.
- **Proof:** Note that $\frac{1}{2}(1 - x_u x_w) = \begin{cases} 0 & \text{if } x_u = x_w \\ 1 & \text{if } x_u \neq x_w \end{cases}$
  So objective value = |{ {u,w} : $x_u \neq x_w$ }| = $|\delta(U)|$. ☐

# VP & SDP for Max Cut

- Make a variable $x_u$ for each $u \in V$

(QIP)
$$\begin{aligned} \max \quad & \sum_{\{u,w\} \in E} \tfrac{1}{2}(1 - x_u x_w) \\ \text{s.t.} \quad & x_u \in \{-1, 1\} \quad \forall u \in V \end{aligned}$$

- Vector Program Relaxation

(VP)
$$\begin{aligned} \max \quad & \sum_{\{u,w\} \in E} \tfrac{1}{2}(1 - v_u^{\mathsf{T}} v_w) \\ \text{s.t.} \quad & v_u^{\mathsf{T}} v_u = 1 \quad \forall u \in V \\ & v_u \in \mathbb{R}^n \quad \forall u \in V \end{aligned}$$

This used to be d, but now it's n, because n = |V|.

- Corresponding Semidefinite Program

(SDP)
$$\begin{aligned} \max \quad & \sum_{\{u,w\} \in E} \tfrac{1}{2}(1 - X_{u,w}) \\ \text{s.t.} \quad & X_{u,u} = 1 \quad \forall u \in V \\ & y^{\mathsf{T}} X y \geq 0 \quad \forall y \in \mathbb{R}^n \end{aligned}$$

# QIP vs SDP

**(QIP)**

$$\begin{array}{lll} \max & \sum_{\{u,w\}\in E} \frac{1}{2}(1 - x_u x_w) & \\ \text{s.t.} & x_u \in \{-1, 1\} & \forall u \in V \end{array}$$

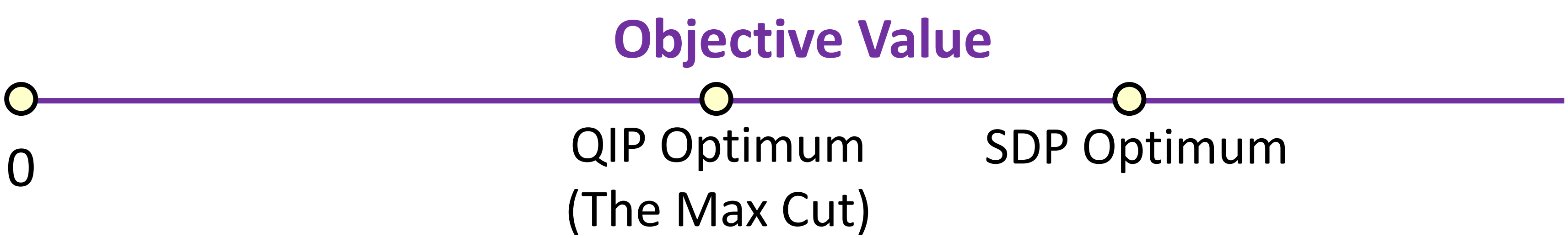
Cannot be solved efficiently,
unless P = NP

**(SDP)**

$$\begin{array}{llll} \max & \sum_{\{u,w\}\in E} \frac{1}{2}(1 - X_{u,w}) & & \\ \text{s.t.} & X_{u,u} & = 1 & \forall u \in V \\ & y^{\mathsf{T}} X y & \geq 0 & \forall y \in \mathbb{R}^n \end{array}$$

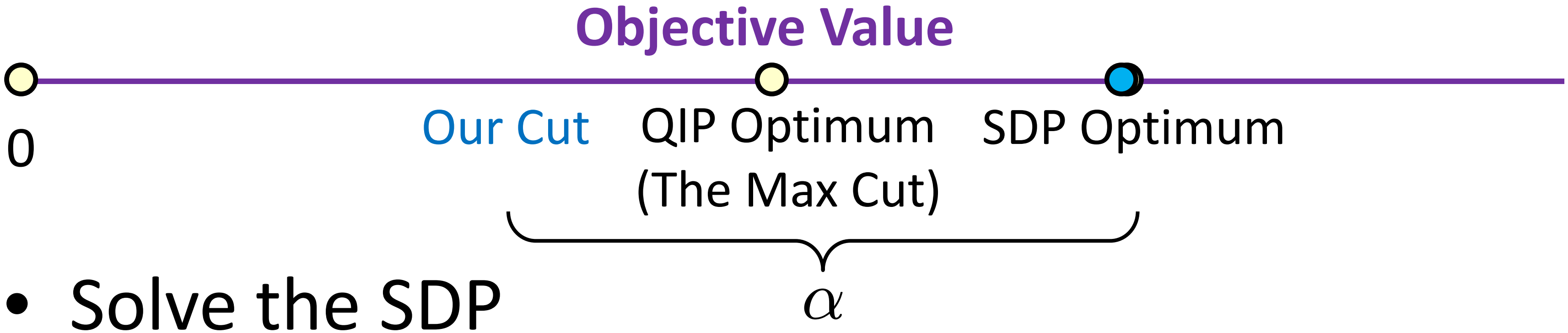
Can be solved by Ellipsoid Method

- How does solving the SDP help us solve the QIP?
- The QIP & SDP can be quite **different**.




- How can the SDP Optimum be better than Max Cut?
  The SDP optimum is not feasible for QIP – it's not a cut!

# Our Game Plan




- Solve the SDP
- **Rounding:** Extract Our Cut from SDP optimum
  (This will be a genuine cut, feasible for QIP)
- Prove that Our Cut is close to SDP Optimum,
  i.e. $\alpha = \dfrac{\text{Value( Our Cut )}}{\text{Value( SDP Opt )}}$ is as **large** as possible.
  $\Rightarrow$ Our Cut is close to QIP Optimum,
  i.e., $\dfrac{\text{Value( Our Cut )}}{\text{Value( QIP Opt )}} \geq \alpha$
- So Our Cut is within a factor $\alpha$ of the optimum

# The Goemans-Williamson Algorithm

- **Theorem:** [Goemans, Williamson 1994]
  There exists an algorithm to extract a cut from the SDP optimum such that

$$\alpha = \frac{\text{Value( Cut )}}{\text{Value( SDP Opt )}} \geq 0.878...$$

Michel Goemans

David Williamson

# The Goemans-Williamson Algorithm

- **Theorem:** [Goemans, Williamson 1994]
  There exists an algorithm to extract a cut from the SDP optimum such that
  $$\alpha = \frac{\text{Value( Cut )}}{\text{Value( SDP Opt )}} \geq 0.878\ldots$$

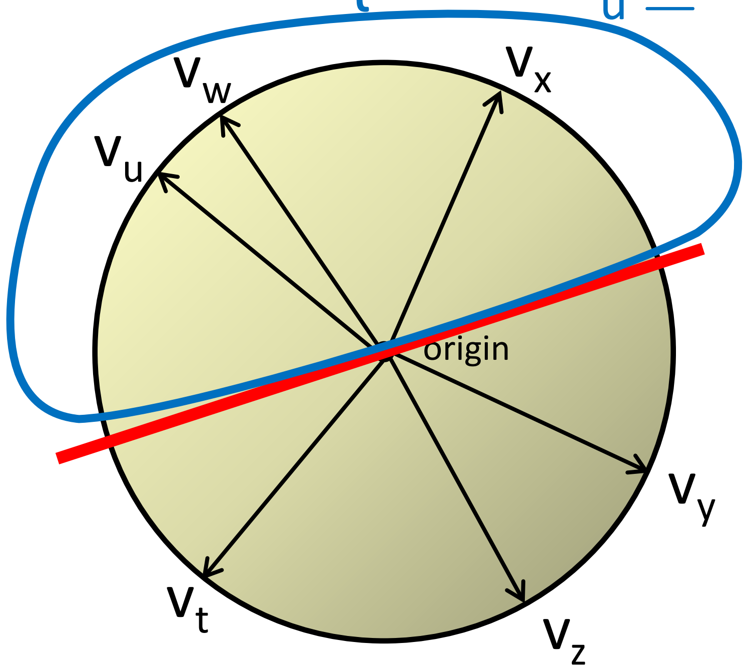
- Astonishingly, this seems to be optimal:
- **Theorem:** [Khot, Kindler, Mossel, O'Donnell 2005]
  No efficient algorithm can approximate Max Cut with factor better than 0.878..., assuming a certain conjecture in complexity theory. (Similar to P$\neq$NP)

# The Goemans-Williamson Algorithm

- Solve the Max Cut Vector Program

$$
\text{(VP)} \qquad \begin{array}{lll} \max & \displaystyle\sum_{\{u,w\}\in E} \tfrac{1}{2}(1 - v_u^{\mathsf{T}} v_w) & \\ \text{s.t.} & v_u^{\mathsf{T}} v_u \quad = 1 & \forall u \in V \\ & v_u \qquad \in \mathbb{R}^n & \forall u \in V \end{array}
$$

- Pick a ***random*** hyperplane through origin
  $H = \{ x : a^{\mathsf{T}}x = 0 \}$ (i.e., a is a **random** vector)
- Return $U = \{ u : a^{\mathsf{T}} v_u \geq 0 \}$

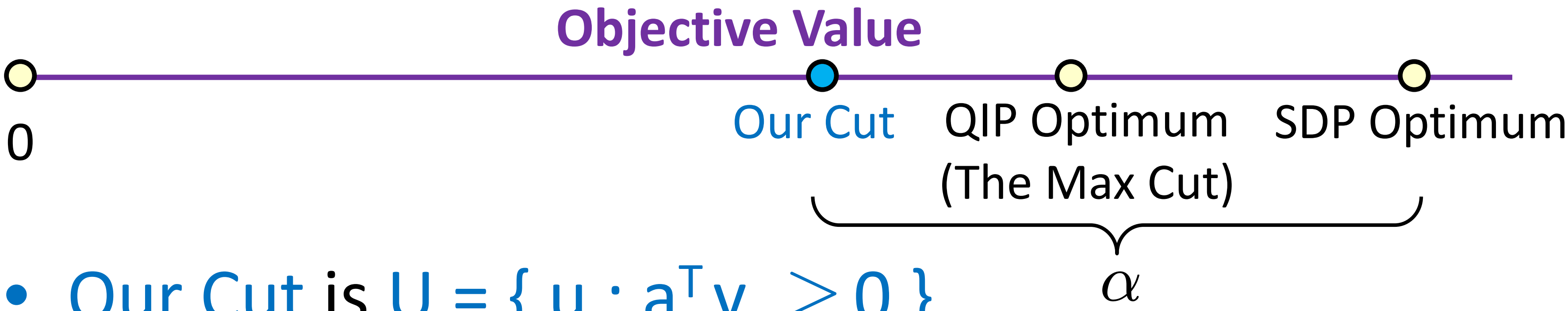



$U = \{ u, w, x \}$

In other words,

$$
x_u = \begin{cases} 1 & \text{if } a^{\mathsf{T}} v_u \geq 0 \\ -1 & \text{if } a^{\mathsf{T}} v_u < 0 \end{cases}
$$

# Analysis of Algorithm




- Our Cut is $U = \{\, u : a^T v_u \geq 0 \,\}$
- Need to prove $\alpha = \dfrac{\text{Value( Our Cut )}}{\text{Value( SDP Opt )}}$ is large
- But a is a random vector, so U is a random set
  $\Rightarrow$ Need to do a probabilistic analysis
- Focus on a particular edge {u,w}:
  What is the probability it is cut by Our Cut?
- **Main Lemma:** $\Pr\left[\, \text{edge } \{u, w\} \text{ cut} \,\right] = \dfrac{\arccos(v_u^T v_w)}{\pi}.$

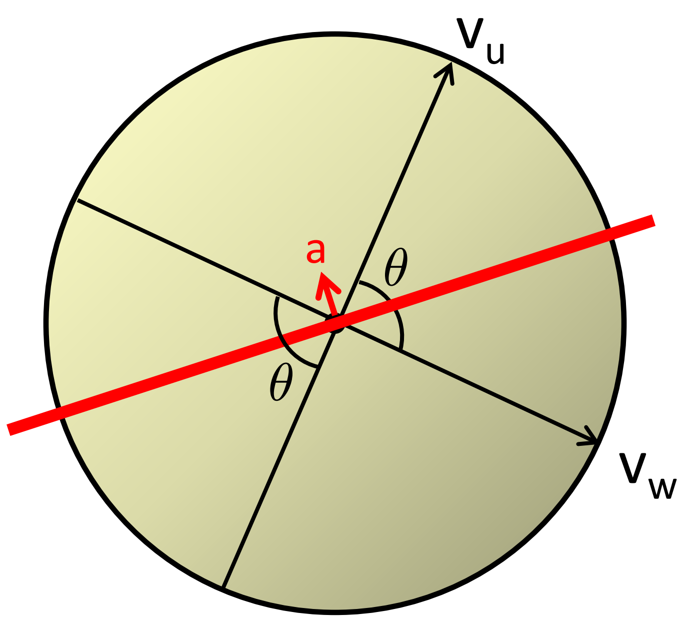
- **Main Lemma:** $\Pr[\,\text{edge } \{u,w\} \text{ cut}\,] = \dfrac{\arccos(v_u^\mathsf{T} v_w)}{\pi}.$
- **Proof:** $\Pr[\,\text{edge } \{u,w\} \text{ cut}\,]$

$$= \Pr[\ \text{exactly one of } u,w \text{ is in } U\ \ ]$$

$$= \Pr\Big[\underbrace{\text{sign}(a^\mathsf{T} v_u) \neq \text{sign}(a^\mathsf{T} v_w)}_{\text{red line lies between } v_u \text{ and } v_w}\Big]$$

- Since direction of red line is uniformly distributed,

$$\Pr[\,\text{red line lies between } v_u \text{ and } v_w\,] = \frac{2\theta}{2\pi}$$



- **Main Lemma:** $\Pr[\,\text{edge } \{u,w\} \text{ cut}\,] = \dfrac{\arccos(v_u^\mathsf{T} v_w)}{\pi}$.
- **Proof:** $\Pr[\,\text{edge } \{u,w\} \text{ cut}\,]$

$$= \Pr[\ \text{exactly one of } u, w \text{ is in } U\ \ ]$$

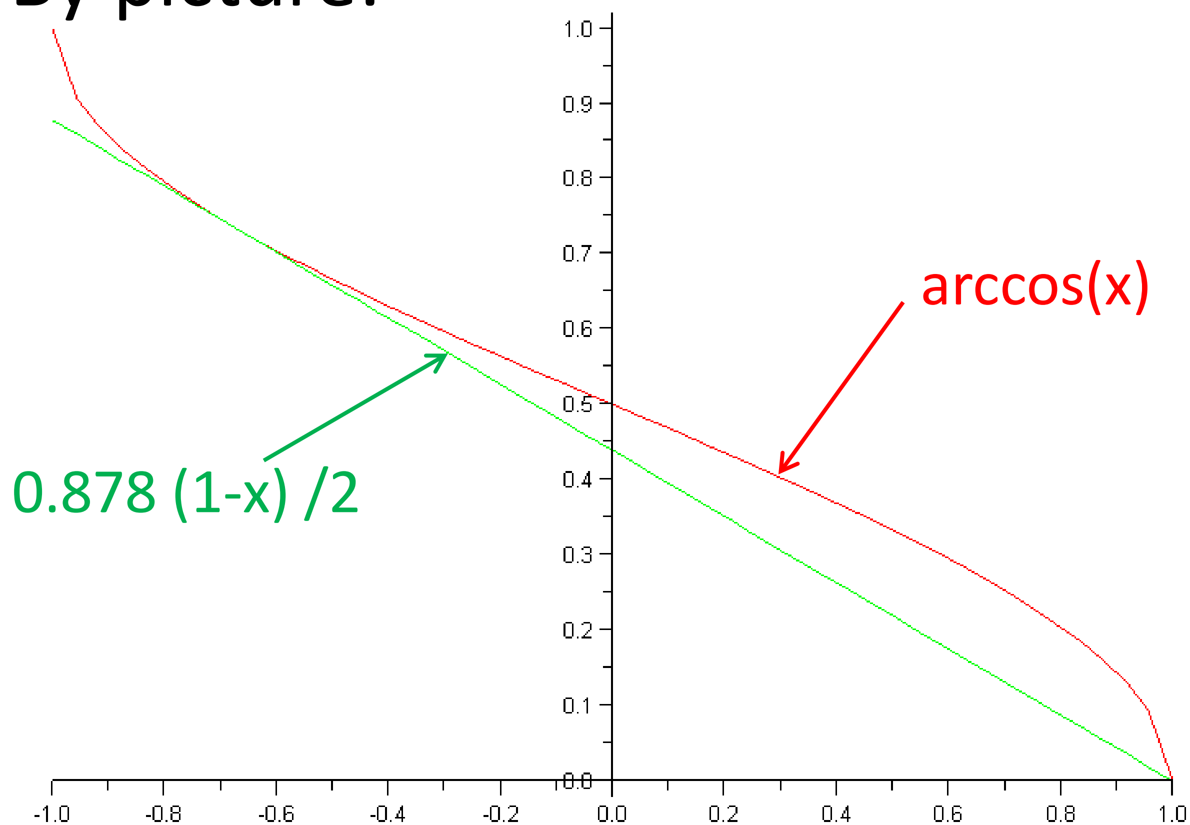
$$= \Pr\Big[\underbrace{\operatorname{sign}(a^\mathsf{T} v_u) \neq \operatorname{sign}(a^\mathsf{T} v_w)}_{\text{red line lies between } v_u \text{ and } v_w}\Big]$$

- Since direction of red line is uniformly distributed,

$$\Pr[\,\text{red line lies between } v_u \text{ and } v_w\,] = \frac{2\theta}{2\pi}$$

- So $\Pr[\,\text{edge } \{u,w\} \text{ cut}\,] = \dfrac{\theta}{\pi}$.
- **Recall:** $v_u^\mathsf{T} v_w = \|v_u\| \cdot \|v_w\| \cdot \cos(\theta)$
- Since $\|v_u\|=\|v_w\|=1$, we have $\theta = \arccos(v_u^\mathsf{T} v_w)$ ■

- **Main Lemma:** $\Pr[\text{edge } \{u,w\} \text{ cut}] = \dfrac{\arccos(v_u^\mathsf{T} v_w)}{\pi}$.
- **Claim:** For all x∈[-1,1], $\dfrac{\arccos(x)}{\pi} \geq 0.878 \cdot \dfrac{1-x}{2}$
- **Proof:** By picture:

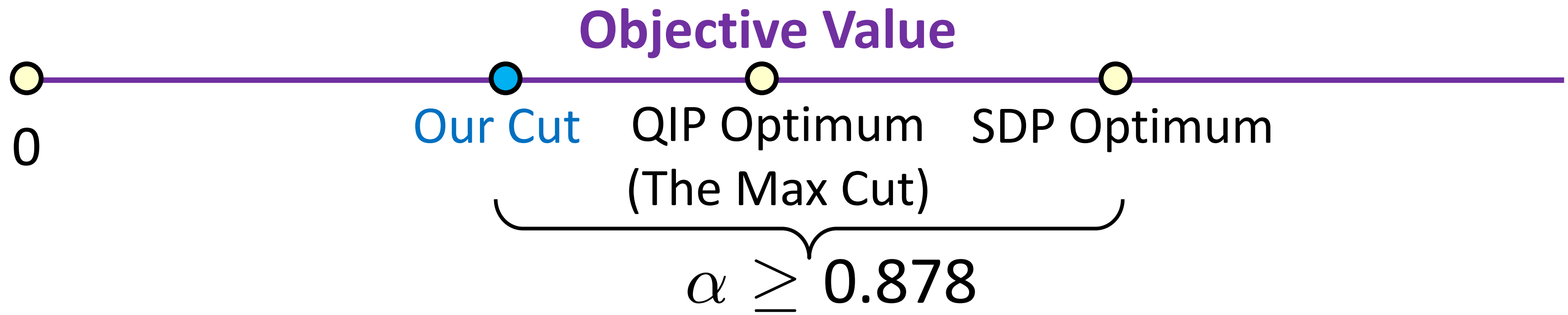



- Can be formalized using calculus. ■

- **Main Lemma:** $\Pr[\,\text{edge } \{u,w\} \text{ cut}\,] = \dfrac{\arccos(v_u^\mathsf{T} v_w)}{\pi}$.
- **Claim:** For all x∈[-1,1], $\dfrac{\arccos(x)}{\pi} \geq 0.878 \cdot \dfrac{1-x}{2}$
- So we can analyze # cut edges:

$$\begin{aligned}
\mathrm{E}[\,\#\text{ cut edges}\,] &= \sum_{\{u,w\}\in E} \Pr[\,\text{edge } \{u,w\} \text{ cut}\,] \\
&= \sum_{\{u,w\}\in E} \frac{\arccos(v_u^\mathsf{T} v_w)}{\pi} \\
&\geq 0.878 \sum_{\{u,w\}\in E} \tfrac{1}{2}(1 - v_u^\mathsf{T} v_w) \\
&= 0.878 \cdot (\text{SDP optimal value})
\end{aligned}$$

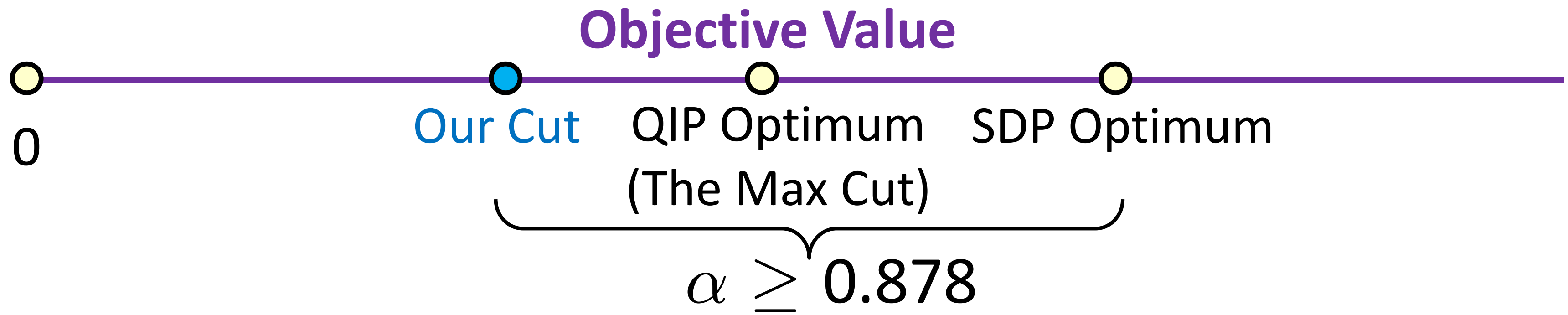
- **Recall:** $\alpha = \dfrac{\text{Value( Our Cut )}}{\text{Value( SDP Opt )}}$. So E[ $\alpha$ ] $\geq$ 0.878.




- So we can analyze # cut edges:

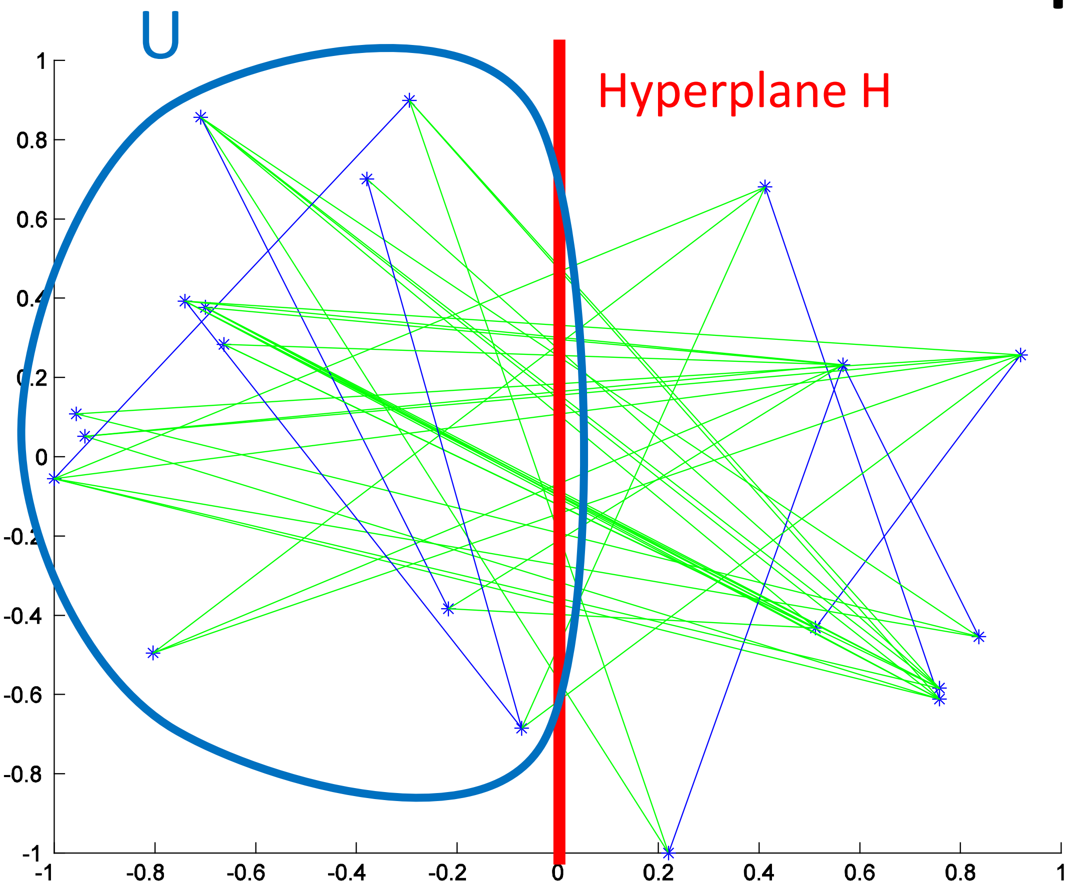
$$
\begin{aligned}
\mathrm{E}[\,\#\text{ cut edges}\,] &= \sum_{\{u,w\}\in E} \Pr[\,\text{edge } \{u,w\} \text{ cut}\,] \\
&= \sum_{\{u,w\}\in E} \frac{\arccos(v_u^{\mathsf{T}} v_w)}{\pi} \\
&\geq 0.878 \sum_{\{u,w\}\in E} \tfrac{1}{2}(1 - v_u^{\mathsf{T}} v_w) \\
&= 0.878 \cdot (\text{SDP optimal value})
\end{aligned}
$$

- **Recall:** $\alpha = \dfrac{\text{Value( Our Cut )}}{\text{Value( SDP Opt )}}$. So E[ $\alpha$ ] $\geq$ 0.878.




- So, in expectation, the algorithm gives a 0.878-approximation to the Max Cut. ■

# Matlab Example




Green edges are cut
38 of them

Blue edges are not cut
8 of them

SDP Opt. Value $\approx$ 39.56
$\Rightarrow$ QIP Opt. Value $\lesssim$ 39
$\alpha \approx$ 38/39.56 = 0.9604

H cuts 38 edges
So Max Cut is either 38 or 39

- **Random graph:** 20 vertices, 46 edges.
- Embedded on unit-sphere in $\mathbb{R}^{20}$, then projected onto 2 random directions.

# Matlab Experiments

- My sample data is a graph with 750 vertices, 3604 edges
- Install SDPT3 (Matlab software for solving SDPs)
  It has example code for solving Max Cut.
- Run this code:

```
load 'Data.txt'; A = Data;      % Load adjacency matrix from file
n = size(A,1);                  % n = number of vertices in the graph
m = sum(sum(A))/2;              % m = number of edges of the graph

[blk,Avec,C,b,X0,y0,Z0,objval,R] = maxcut(A,1,1);      % Run the SDP solver

X = R{1};                       % X is the optimal solution to SDP
V = chol(X);                    % Columns of V are solution to Vector Program

a = randn(1,n);                 % The vector a defines a random hyperplane
x = sign( a * V )';             % x is our integral solution
cut = m/2 - x'*A*x/4            % This counts how many edges are cut by x
sdpOpt = -objval                % This is the SDP optimal value
ratio = cut/sdpOpt              % This compares cut to SDP optimum
```

Here we use the fact that product of Normal Distributions is spherically symmetric.

- **Output:** cut=2880, sdpOpt=3206.5, ratio=0.8982