Lecture 14: Semidefinite Programming Relaxation and MAX-CUT

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

November 11, 2020

Overview

Motivation - NP-hard problems

- Many important problems are NP-hard to solve.

Motivation - NP-hard problems

- Many important problems are NP-hard to solve.
- What do we do when we see one?

Motivation - NP-hard problems

- Many important problems are NP-hard to solve.
- What do we do when we see one?
  1. Find approximate solutions in polynomial time!

Motivation - NP-hard problems

- Many important problems are NP-hard to solve.
- What do we do when we see one?
    1. Find approximate solutions in polynomial time!
    2. Sometimes we even do that for problems in P (but we want much much faster solutions)

Motivation - NP-hard problems

- Many important problems are NP-hard to solve.
- What do we do when we see one?
  1. Find approximate solutions in polynomial time!

- **Integer Linear Program (ILP):**

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x \in \mathbb{N}^n \end{aligned}$$

Motivation - NP-hard problems

- Many important problems are NP-hard to solve.
- What do we do when we see one?
  1. Find approximate solutions in polynomial time!

- **Integer Linear Program (ILP):**

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x \in \mathbb{N}^n \end{aligned}$$

- Advantage of ILPs: very expressive language to formulate optimization problems (capture many combinatorial optimization problems)

Motivation - NP-hard problems

- Many important problems are NP-hard to solve.
- What do we do when we see one?
  1. Find approximate solutions in polynomial time!

- **Integer Linear Program (ILP):**

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x \in \mathbb{N}^n \end{aligned}$$

- Advantage of ILPs: very expressive language to formulate optimization problems (capture many combinatorial optimization problems)
- Disadvantage of ILPs: capture even NP-hard problems (thus NP-hard)

Motivation - NP-hard problems

- Many important problems are NP-hard to solve.
- What do we do when we see one?
  1. Find approximate solutions in polynomial time!

- **Integer Linear Program (ILP):**

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x \in \mathbb{N}^n \end{aligned}$$

- Advantage of ILPs: very expressive language to formulate optimization problems (capture many combinatorial optimization problems)
- Disadvantage of ILPs: capture even NP-hard problems (thus NP-hard)
- But we know how to solve LPs. Can we get partial credit in life?

Motivation - NP-hard problems

- **Quadratic Program (QP):**

$$\begin{aligned} \text{minimize} \quad & g(x) \\ \text{subject to} \quad & q_i(x) \geq 0 \end{aligned}$$

where each $q_i(x)$ and $g(x)$ are quadratic functions on x.

Motivation - NP-hard problems

- **Quadratic Program (QP):**

$$\begin{aligned} \text{minimize} \quad & g(x) \\ \text{subject to} \quad & q_i(x) \geq 0 \end{aligned}$$

  where each $q_i(x)$ and $g(x)$ are quadratic functions on x.
- Advantage of QPs: very expressive language to formulate optimization problems

Motivation - NP-hard problems

- **Quadratic Program (QP):**

$$\begin{aligned} \text{minimize} \quad & g(x) \\ \text{subject to} \quad & q_i(x) \geq 0 \end{aligned}$$

  where each $q_i(x)$ and $g(x)$ are quadratic functions on x.
- Advantage of QPs: very expressive language to formulate optimization problems
- Disadvantage of QPs: capture even NP-hard problems (ILPs for instance)

Motivation - NP-hard problems

- **Quadratic Program (QP):**

$$\begin{aligned} \text{minimize} \quad & g(x) \\ \text{subject to} \quad & q_i(x) \geq 0 \end{aligned}$$

  where each $q_i(x)$ and $g(x)$ are quadratic functions on x.
- Advantage of QPs: very expressive language to formulate optimization problems
- Disadvantage of QPs: capture even NP-hard problems (ILPs for instance)
- Can relax quadratic programs with SDPs

Motivation - NP-hard problems

- **Quadratic Program (QP):**

$$\text{minimize} \quad g(x)$$
$$\text{subject to} \quad q_i(x) \geq 0$$

  where each $q_i(x)$ and $g(x)$ are quadratic functions on x.
- Advantage of QPs: very expressive language to formulate optimization problems
- Disadvantage of QPs: capture even NP-hard problems (ILPs for instance)
- Can relax quadratic programs with SDPs

  - Can we get **better** approximations using SDPs instead of ILPs?

Motivation - NP-hard problems

- **Quadratic Program (QP):**

$$\begin{aligned} \text{minimize} \quad & g(x) \\ \text{subject to} \quad & q_i(x) \geq 0 \end{aligned}$$

  where each $q_i(x)$ and $g(x)$ are quadratic functions on x.
- Advantage of QPs: very expressive language to formulate optimization problems
- Disadvantage of QPs: capture even NP-hard problems (ILPs for instance)
- Can relax quadratic programs with SDPs

  - Can we get **better** approximations using SDPs instead of ILPs?
- Yes. Today we will see Max-Cut (more generally constraint satisfaction relaxations)

Motivation - NP-hard problems

- **Quadratic Program (QP):**

$$\begin{aligned} \text{minimize} \quad & g(x) \\ \text{subject to} \quad & q_i(x) \geq 0 \end{aligned}$$

  where each $q_i(x)$ and $g(x)$ are quadratic functions on x.
- Advantage of QPs: very expressive language to formulate optimization problems
- Disadvantage of QPs: capture even NP-hard problems (ILPs for instance)
- Can relax quadratic programs with SDPs
  - Can we get **better** approximations using SDPs instead of ILPs?
- Yes. Today we will see Max-Cut (more generally constraint satisfaction relaxations)
- Very impressive recent theoretical developments! Unique Games Conjecture, Sum-of-Squares, and more!

Example

Maximum Cut (Max-Cut):

$G(V, E)$ graph.

Cut $S \subseteq V$ and size of cut is

$$|E(S, \overline{S})| = |\{(u, v) \in E \ \mid \ u \in S, v \notin S\}|.$$

Goal: find cut of maximum size.

Example

Maximum Cut (Max-Cut):

$G(V, E)$ graph.

Cut $S \subseteq V$ and size of cut is

$$|E(S, \overline{S})| = |\{(u, v) \in E \mid u \in S, v \notin S\}|.$$

Goal: find cut of maximum size.

Integer Linear Program:

$$\begin{aligned}
\text{maximize} \quad & \sum_{e \in E} z_e \\
\text{subject to} \quad x_u + x_v &\geq z_e \quad \text{for } e = \{u, v\} \in E \\
2 - x_u - x_v &\geq z_e \quad \text{for } e = \{u, v\} \in E \\
x_v &\in \{0, 1\} \quad \text{for } v \in V
\end{aligned}$$

Example - Weighted Variant

Maximum Cut (Max-Cut):

$$G(V, E, w) \text{ weighted graph. } \sum_{e \in E} w_e = 1$$

Cut $S \subseteq V$ and weight of cut is the sum of weights of edges crossing cut.
Goal: find cut of maximum weight.

Integer Linear Program:

$$
\begin{aligned}
\text{maximize} \quad & \sum_{e \in E} z_e \cdot w_e \\
\text{subject to} \quad x_u + x_v \geq z_e & \quad \text{for } e = \{u, v\} \in E \\
2 - x_u - x_v \geq z_e & \quad \text{for } e = \{u, v\} \in E \\
x_v \in \{0, 1\} & \quad \text{for } v \in V
\end{aligned}
$$

Relax... & Round!

In our quest to get efficient (exact or approximate) algorithms for problems of interest, the following strategy is very useful:

[1] Even more general mathematical program, so long as derive SDP from it.

Relax... & Round!

In our quest to get efficient (exact or approximate) algorithms for problems of interest, the following strategy is very useful:

1. Formulate optimization problem as QP[1]

[1] Even more general mathematical program, so long as derive SDP from it.

Relax... & Round!

In our quest to get efficient (exact or approximate) algorithms for problems of interest, the following strategy is very useful:

1. Formulate optimization problem as QP[1]
2. Derive SDP from the QP by going to higher dimensions and imposing PSD constraint

This is called an *SDP relaxation*.

[1] Even more general mathematical program, so long as derive SDP from it.

Relax... & Round!

In our quest to get efficient (exact or approximate) algorithms for problems of interest, the following strategy is very useful:

1. Formulate optimization problem as QP[1]
2. Derive SDP from the QP by going to higher dimensions and imposing PSD constraint

   This is called an *SDP relaxation*.
3. We are still maximizing the same objective function, but over a (potentially) larger set of solutions.

$$OPT(SDP) \geq OPT(ISDP)$$

[1] Even more general mathematical program, so long as derive SDP from it.

Relax... & Round!

In our quest to get efficient (exact or approximate) algorithms for problems of interest, the following strategy is very useful:

1. Formulate optimization problem as QP[1]
2. Derive SDP from the QP by going to higher dimensions and imposing PSD constraint

   This is called an *SDP relaxation*.
3. We are still maximizing the same objective function, but over a (potentially) larger set of solutions.

   $$OPT(SDP) \geq OPT(ISDP)$$
4. Solve SDP (approximately) optimally using efficient algorithm.

[1] Even more general mathematical program, so long as derive SDP from it.

Relax... & Round!

In our quest to get efficient (exact or approximate) algorithms for problems of interest, the following strategy is very useful:

1. Formulate optimization problem as QP[1]
2. Derive SDP from the QP by going to higher dimensions and imposing PSD constraint

   This is called an *SDP relaxation*.
3. We are still maximizing the same objective function, but over a (potentially) larger set of solutions.

   $$OPT(SDP) \geq OPT(ISDP)$$
4. Solve SDP (approximately) optimally using efficient algorithm.
   1. If solution to SDP is *integral* and *one-dimensional*, then it is a solution to QP and we are done

[1] Even more general mathematical program, so long as derive SDP from it.

Relax... & Round!

In our quest to get efficient (exact or approximate) algorithms for problems of interest, the following strategy is very useful:

1. Formulate optimization problem as QP[1]
2. Derive SDP from the QP by going to higher dimensions and imposing PSD constraint

   This is called an *SDP relaxation*.
3. We are still maximizing the same objective function, but over a (potentially) larger set of solutions.

$$OPT(SDP) \geq OPT(ISDP)$$

4. Solve SDP (approximately) optimally using efficient algorithm.
   1. If solution to SDP is *integral* and *one-dimensional*, then it is a solution to QP and we are done
   2. If solution has *higher dimension*, then we have to devise *rounding procedure* that transforms

      high dimensional solutions $\rightarrow$ integral & 1D solutions

$$\text{rounded SDP solution value} \geq c \cdot OPT(QP)$$

[1] Even more general mathematical program, so long as derive SDP from it.

Analyzing ILP for Max-Cut

$G(V, E, w)$ weighted graph. $\sum_{e \in E} w_e = 1$

Integer Linear Program:

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} z_e \cdot w_e \\ \text{subject to} \quad & x_u + x_v \geq z_e \quad \text{for } e = \{u, v\} \in E \\ & 2 - x_u - x_v \geq z_e \quad \text{for } e = \{u, v\} \in E \\ & x_v \in \{0, 1\} \quad \text{for } v \in V \end{aligned}$$

Analyzing ILP for Max-Cut

$G(V, E, w)$ weighted graph. $\sum_{e \in E} w_e = 1$

Integer Linear Program:

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} z_e \cdot w_e \\ \text{subject to} \quad & x_u + x_v \geq z_e \quad \text{for } e = \{u, v\} \in E \\ & 2 - x_u - x_v \geq z_e \quad \text{for } e = \{u, v\} \in E \\ & x_v \in \{0, 1\} \quad \text{for } v \in V \end{aligned}$$

- $OPT(ILP) = 1 \Leftrightarrow G$ is bipartite

Analyzing ILP for Max-Cut

$G(V, E, w)$ weighted graph. $\sum_{e \in E} w_e = 1$

Integer Linear Program:

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} z_e \cdot w_e \\ \text{subject to} \quad & x_u + x_v \geq z_e \quad \text{for } e = \{u, v\} \in E \\ & 2 - x_u - x_v \geq z_e \quad \text{for } e = \{u, v\} \in E \\ & x_v \in \{0, 1\} \quad \text{for } v \in V \end{aligned}$$

- $OPT(ILP) = 1 \Leftrightarrow G$ is bipartite
- $OPT(ILP) \geq 1/2$

Analyzing ILP for Max-Cut

$G(V, E, w)$ weighted graph. $\sum_{e \in E} w_e = 1$

Integer Linear Program:

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} z_e \cdot w_e \\ \text{subject to} \quad & x_u + x_v \geq z_e \quad \text{for } e = \{u, v\} \in E \\ & 2 - x_u - x_v \geq z_e \quad \text{for } e = \{u, v\} \in E \\ & x_v \in \{0, 1\} \quad \text{for } v \in V \end{aligned}$$

- $OPT(ILP) = 1 \Leftrightarrow G$ is bipartite
- $OPT(ILP) \geq 1/2$
- G complete graph $\Rightarrow OPT = \frac{1}{2} + \frac{1}{2(n-1)}$

Analyzing ILP for Max-Cut

$G(V, E, w)$ weighted graph. $\sum_{e \in E} w_e = 1$

Integer Linear Program:

$$\begin{aligned}
\text{maximize} \quad & \sum_{e \in E} z_e \cdot w_e \\
\text{subject to} \quad & x_u + x_v \geq z_e \quad \text{for } e = \{u, v\} \in E \\
& 2 - x_u - x_v \geq z_e \quad \text{for } e = \{u, v\} \in E \\
& x_v \in \{0, 1\} \quad \text{for } v \in V
\end{aligned}$$

- $OPT(ILP) = 1 \Leftrightarrow G$ is bipartite
- $OPT(ILP) \geq 1/2$
- G complete graph $\Rightarrow OPT = \frac{1}{2} + \frac{1}{2(n-1)}$
- Max-Cut NP-hard

Proof that $OPT(ILP) \geq 1/2$

Probabilistic method:

Pick $x_v = \begin{cases} 0 & \text{with probability } 1/2 \\ 1 & \text{with probability } 1/2 \end{cases}$

$\mathbb{E}[z_{uv}] = 1/2$

$\mathbb{E}[\text{value of cut}] = \sum w_{uv} \cdot \mathbb{E}[z_{uv}]$

$= \frac{1}{2} \sum w_{uv} = \frac{1}{2}$

$\therefore$ there is integral solution that is

$\geq$ average (expectation)

Rounding Max-Cut ILP

$G(V, E, w)$ weighted graph. $\sum_{e \in E} w_e = 1$

Linear Program Relaxation:

$$\begin{aligned}
\text{maximize} \quad & \sum_{e \in E} z_e \cdot w_e \\
\text{subject to} \quad & x_u + x_v \geq z_e \quad \text{for } e = \{u, v\} \in E \\
& 2 - x_u - x_v \geq z_e \quad \text{for } e = \{u, v\} \in E \\
& 0 \leq x_v \leq 1 \quad \text{for } v \in V \\
& 0 \leq z_e \leq 1 \quad \text{for } e \in E
\end{aligned}$$

Rounding Max-Cut ILP

$G(V, E, w)$ weighted graph. $\sum_{e \in E} w_e = 1$

Linear Program Relaxation:

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} z_e \cdot w_e \\ \text{subject to} \quad & x_u + x_v \geq z_e \quad \text{for } e = \{u, v\} \in E \\ & 2 - x_u - x_v \geq z_e \quad \text{for } e = \{u, v\} \in E \\ & 0 \leq x_v \leq 1 \quad \text{for } v \in V \\ & 0 \leq z_e \leq 1 \quad \text{for } e \in E \end{aligned}$$

- Setting $x_v = 1/2$, $z_e = 1$ we get $OPT(LP)$ always $= 1$

Rounding Max-Cut ILP

$G(V, E, w)$ weighted graph. $\sum_{e \in E} w_e = 1$

Linear Program Relaxation:

$$
\begin{aligned}
\text{maximize} \quad & \sum_{e \in E} z_e \cdot w_e \\
\text{subject to} \quad & x_u + x_v \geq z_e \quad \text{for } e = \{u, v\} \in E \\
& 2 - x_u - x_v \geq z_e \quad \text{for } e = \{u, v\} \in E \\
& 0 \leq x_v \leq 1 \quad \text{for } v \in V \\
& 0 \leq z_e \leq 1 \quad \text{for } e \in E
\end{aligned}
$$

- Setting $x_v = 1/2$, $z_e = 1$ we get $OPT(LP)$ always $= 1$
- This relaxation is not helpful! :(

- Why Relax & Round?
- Max-Cut SDP Relaxation and Rounding
- Conclusion
- Acknowledgements

Max-Cut

$G(V, E, w)$ weighted graph. $\sum_{e \in E} w_e = 1$

Quadratic Program:

$$\begin{aligned} &\text{maximize} \quad \sum_{\{u,v\} \in E} \frac{1}{2} \cdot w_{u,v} \cdot (1 - x_u x_v) \\ &\text{subject to} \quad x_v^2 = 1 \quad \text{for } v \in V \end{aligned}$$

SDP Relaxation [Delorme, Poljak 1993]

$G(V, E, w)$ weighted graph, $|V| = n$ and $\sum_{e \in E} w_e = 1$

Semidefinite Program:

$$\begin{aligned} \text{maximize} \quad & \sum_{\{u,v\} \in E} \frac{1}{2} \cdot w_{u,v} \cdot \left(1 - y_u^T y_v\right) \\ \text{subject to} \quad & \|y_v\|_2^2 = 1 \quad \text{for } v \in V \\ & y_v \in \mathbb{R}^d \quad \text{for } v \in V \end{aligned}$$

SDP Relaxation [Delorme, Poljak 1993]

$G(V, E, w)$ weighted graph, $|V| = n$ and $\sum_{e \in E} w_e = 1$

Semidefinite Program:

$$\begin{aligned} \text{maximize} \quad & \sum_{\{u,v\} \in E} \frac{1}{2} \cdot w_{u,v} \cdot \left(1 - y_u^T y_v\right) \\ \text{subject to} \quad & \|y_v\|_2^2 = 1 \quad \text{for } v \in V \\ & y_v \in \mathbb{R}^d \quad \text{for } v \in V \end{aligned}$$

$d \leq n$

- How is that an SDP?

$X_{ij} = y_i^T y_j \quad \therefore \quad X = Y^T Y \quad Y = (y_1\ y_2 \cdots y_n)$

$X_{ii} = y_i^T y_i = \|y_i\|^2 = 1$

$\Leftrightarrow X \succeq 0$ and $X_{ii} = 1 \ \forall i \in [n]$

What is this SDP doing?

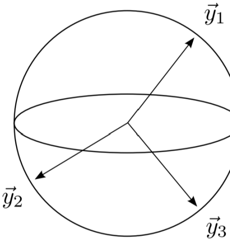


Figure 10.1: Vectors $\vec{y}_v$ embedded onto a unit sphere in $\mathbb{R}^d$.

What is this SDP doing?

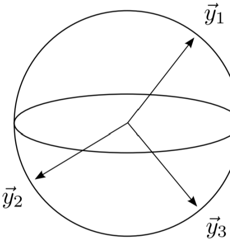


Figure 10.1: Vectors $\vec{y}_v$ embedded onto a unit sphere in $\mathbb{R}^d$.

- Let $\gamma_{u,v} = y_u^T y_v = \cos(y_u, y_v)$

What is this SDP doing?

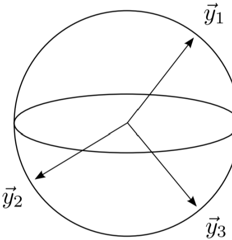


Figure 10.1: Vectors $\vec{y}_v$ embedded onto a unit sphere in $\mathbb{R}^d$.

- Let $\gamma_{u,v} = y_u^T y_v = \cos(y_u, y_v)$
- for any edge, want $\gamma_{uv} \approx -1$, as this maximizes our weight

What is this SDP doing?

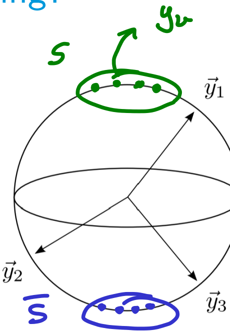


Figure 10.1: Vectors $\vec{y}_v$ embedded onto a unit sphere in $\mathbb{R}^d$.

- Let $\gamma_{u,v} = y_u^T y_v = \cos(y_u, y_v)$
- for any edge, want $\gamma_{uv} \approx -1$, as this maximizes our weight
- Geometrically, want vertices from our max-cut S to be as far away from the complement $\overline{S}$ as possible

What is this SDP doing?

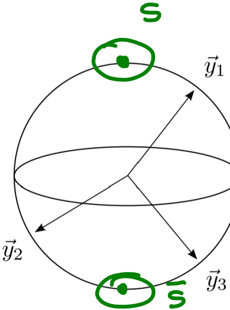


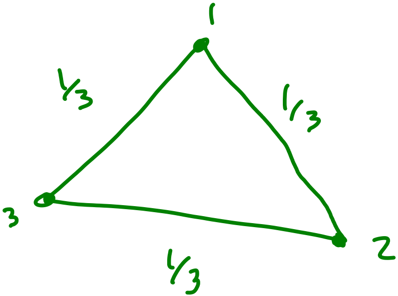
Figure 10.1: Vectors $\vec{y}_v$ embedded onto a unit sphere in $\mathbb{R}^d$.

- Let $\gamma_{u,v} = y_u^T y_v = \cos(y_u, y_v)$
- for any edge, want $\gamma_{uv} \approx -1$, as this maximizes our weight
- Geometrically, want vertices from our max-cut S to be as far away from the complement $\overline{S}$ as possible
- If all y_v's are in a one-dimensional space, then we get original quadratic program

$$OPT(SDP) \geq \text{Weight of Maximum Cut}$$

Example

Let's consider $G = K_3$ with equal weights on edges.



Example

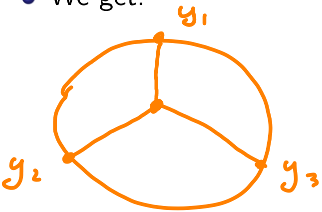
Let's consider $G = K_3$ with equal weights on edges.

- Embed $y_1, y_2, y_3 \in \mathbb{R}^2$ 120 degrees apart in unit circle

Example

Let's consider $G = K_3$ with equal weights on edges. (y_3)

- Embed $y_1, y_2, y_3 \in \mathbb{R}^2$ 120 degrees apart in unit circle
- We get:



$$OPT(SDP) = \sum_{i<j} \frac{1}{3} \cdot \frac{1}{2} \cdot \left(1 - \cos(2\pi/3)\right)$$

$$= 3 \cdot \frac{1}{3} \cdot \frac{1}{2} \cdot \left(1 + \frac{1}{2}\right)$$

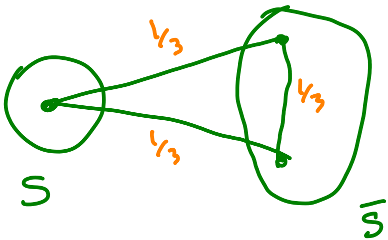
$$= 3/4$$

$$\cos(2\pi/3) = -\cos(\pi - 2\pi/3) = -\cos(\pi/3) = \frac{1}{2}$$

Example

Let's consider $G = K_3$ with equal weights on edges.

- Embed $y_1, y_2, y_3 \in \mathbb{R}^2$ 120 degrees apart in unit circle
- We get:
- $OPT_{SDP}(K_3) = 3/4$
- $OPT_{\text{max-cut}}(K_3) = 2/3$



Example

Let's consider $G = K_3$ with equal weights on edges.

- Embed $y_1, y_2, y_3 \in \mathbb{R}^2$ 120 degrees apart in unit circle
- We get:
- $OPT_{SDP}(K_3) = 3/4$
- $OPT_{\text{max-cut}}(K_3) = 2/3$
- So we get approximation $8/9$ (better than the LP relaxation)

Example

Let's consider $G = K_3$ with equal weights on edges.

- Embed $y_1, y_2, y_3 \in \mathbb{R}^2$ 120 degrees apart in unit circle
- We get:
- $OPT_{SDP}(K_3) = 3/4$
- $OPT_{\text{max-cut}}(K_3) = 2/3$
- So we get approximation 8/9 (better than the LP relaxation)
- **Practice problem:** try this with C_5.

should get ≈ 0.88

Max-Cut - Rounding

1. Let $z_u \in \mathbb{R}^n$ be an optimal solution to our SDP

Max-Cut - Rounding

1. Let $z_u \in \mathbb{R}^n$ be an optimal solution to our SDP
2. How do we convert it into a cut?

Max-Cut - Rounding

1. Let $z_u \in \mathbb{R}^n$ be an optimal solution to our SDP
2. How do we convert it into a cut?
3. Need to "pick sides"

Max-Cut - Rounding

1. Let $z_u \in \mathbb{R}^n$ be an optimal solution to our SDP
2. How do we convert it into a cut?
3. Need to "pick sides"
4. **[Goemans, Williamson 1994]:** Choose a random hyperplane though origin!

Max-Cut - Rounding

1. Let $z_u \in \mathbb{R}^n$ be an optimal solution to our SDP
2. How do we convert it into a cut?
3. Need to “pick sides”
4. **[Goemans, Williamson 1994]:** Choose a random hyperplane though origin!
5. Choose normal vector $g \in \mathbb{R}^n$ from a Gaussian distribution.
6. Set $x_u = \text{sign}(g^T z_u)$ as our solution

Max-Cut - Rounding

1. Let $z_u \in \mathbb{R}^n$ be an optimal solution to our SDP
2. How do we convert it into a cut?
3. Need to "pick sides"
4. **[Goemans, Williamson 1994]:** Choose a random hyperplane though origin!
5. Choose normal vector $g \in \mathbb{R}^n$ from a Gaussian distribution.
6. Set $x_u = \text{sign}(g^T z_u)$ as our solution

$1 \mapsto +1$

$2, 3 \mapsto -1$

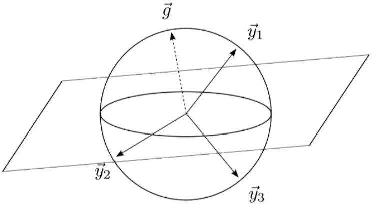


Figure 10.2: Vectors being separated by a hyperplane with normal $\vec{g}$.

Analysis of Rounding - Sketch

- Probability that edge $\{u, v\}$ crosses the cut is same as probability that z_u, z_v fall in different sides of hyperplane

$$\Pr[\{u, v\} \text{ crosses cut }] = \Pr[g \text{ splits } z_u, z_v]$$

Analysis of Rounding - Sketch

- Probability that edge $\{u, v\}$ crosses the cut is same as probability that z_u, z_v fall in different sides of hyperplane

$$\Pr[\{u, v\} \text{ crosses cut }] = \Pr[g \text{ splits } z_u, z_v]$$

- Looking at the problem in the plane:

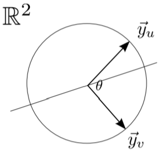


Figure 10.3: The plane of two vectors being cut by the hyperplane

Plane puts u, v different sides of cut.

Analysis of Rounding - Sketch

- Probability that edge $\{u, v\}$ crosses the cut is same as probability that z_u, z_v fall in different sides of hyperplane

$$\Pr[\{u, v\} \text{ crosses cut }] = \Pr[g \text{ splits } z_u, z_v]$$

- Looking at the problem in the plane:

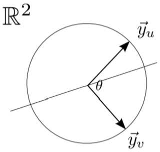


Figure 10.3: The plane of two vectors being cut by the hyperplane

- Probability of splitting z_u, z_v:

$$\Pr[\{u, v\} \text{ crosses cut}] = \frac{\theta}{\pi} = \frac{\cos^{-1}(z_u^T z_v)}{\pi} = \frac{\cos^{-1}(\gamma_{uv})}{\pi}$$

$$\mathbb{E}[\text{value of cut}] = \sum_{\{u,v\} \in E} w_{u,v} \cdot \frac{\cos^{-1}(\gamma_{uv})}{\pi}$$

Analysis of Rounding - Sketch

- Expected value of cut:

$$\mathbb{E}[\text{value of cut}] = \sum_{\{u,v\} \in E} w_{u,v} \cdot \frac{\cos^{-1}(\gamma_{uv})}{\pi}$$

Analysis of Rounding - Sketch

- Expected value of cut:

$$\mathbb{E}[\text{value of cut}] = \sum_{\{u,v\} \in E} w_{u,v} \cdot \frac{\cos^{-1}(\gamma_{uv})}{\pi}$$

- Recall that

$$OPT_{SDP} = \sum_{\{u,v\} \in E} \frac{1}{2} \cdot w_{u,v} \cdot \left(1 - z_u^T z_v\right) = \sum_{\{u,v\} \in E} \frac{1}{2} \cdot w_{u,v} \cdot (1 - \gamma_{uv})$$

Analysis of Rounding - Sketch

- Expected value of cut:

$$\mathbb{E}[\text{value of cut}] = \sum_{\{u,v\}\in E} w_{u,v} \cdot \frac{\cos^{-1}(\gamma_{uv})}{\pi}$$

- Recall that

$$OPT_{SDP} = \sum_{\{u,v\}\in E} \frac{1}{2} \cdot w_{u,v} \cdot \left(1 - z_u^T z_v\right) = \sum_{\{u,v\}\in E} \frac{1}{2} \cdot w_{u,v} \cdot (1 - \gamma_{uv})$$

- If we find α such that

$$\frac{\cos^{-1}(\gamma_{uv})}{\pi} \geq \frac{1}{2}(1 - \gamma_{uv}), \quad \text{for all } \gamma_{uv} \in [-1, 1]$$

Then we have an α-approximation algorithm!

(also need to prove concentration result)

Analysis of Rounding - Sketch

- Expected value of cut:

$$\mathbb{E}[\text{value of cut}] = \sum_{\{u,v\}\in E} w_{u,v} \cdot \frac{\cos^{-1}(\gamma_{uv})}{\pi}$$

- Recall that

$$OPT_{SDP} = \sum_{\{u,v\}\in E} \frac{1}{2} \cdot w_{u,v} \cdot \left(1 - z_u^T z_v\right) = \sum_{\{u,v\}\in E} \frac{1}{2} \cdot w_{u,v} \cdot (1 - \gamma_{uv})$$

- If we find α such that

$$\frac{\cos^{-1}(\gamma_{uv})}{\pi} \geq \frac{1}{2}(1 - \gamma_{uv}), \quad \text{for all } \gamma_{uv} \in [-1, 1]$$

Then we have an α-approximation algorithm!

Theorem ([Goemans, Williamson 1994])

$\alpha = 0.87856...$ *works, and gives us our approximation algorithm.*

Putting Everything Together

1. Formulate Max-Cut problem as Quadratic Programming

Putting Everything Together

1. Formulate Max-Cut problem as Quadratic Programming
2. Derive SDP from the quadratic program *SDP relaxation*

Putting Everything Together

1. Formulate Max-Cut problem as Quadratic Programming
2. Derive SDP from the quadratic program *SDP relaxation*

Putting Everything Together

1. Formulate Max-Cut problem as Quadratic Programming
2. Derive SDP from the quadratic program *SDP relaxation*
3. We are still maximizing the same objective function (weight of cut), but over a (potentially) larger (*higher-dimensional*) set of solutions.

$$OPT(SDP) \geq OPT(\text{Max-Cut})$$

Putting Everything Together

1. Formulate Max-Cut problem as Quadratic Programming
2. Derive SDP from the quadratic program *SDP relaxation*
3. We are still maximizing the same objective function (weight of cut), but over a (potentially) larger (*higher-dimensional*) set of solutions.

$$OPT(SDP) \geq OPT(\text{Max-Cut})$$

4. Solve SDP optimally using efficient algorithm.

Putting Everything Together

1. Formulate Max-Cut problem as Quadratic Programming
2. Derive SDP from the quadratic program *SDP relaxation*
3. We are still maximizing the same objective function (weight of cut), but over a (potentially) larger (*higher-dimensional*) set of solutions.

$$OPT(SDP) \geq OPT(\text{Max-Cut})$$

4. Solve SDP optimally using efficient algorithm.
    1. If solution to SDP is *integral* and *one dimensional*, then it is a solution to Max-Cut and we are done

Putting Everything Together

1. Formulate Max-Cut problem as Quadratic Programming
2. Derive SDP from the quadratic program *SDP relaxation*
3. We are still maximizing the same objective function (weight of cut), but over a (potentially) larger (*higher-dimensional*) set of solutions.

$$OPT(SDP) \geq OPT(\text{Max-Cut})$$

4. Solve SDP optimally using efficient algorithm.
   1. If solution to SDP is *integral* and *one dimensional*, then it is a solution to Max-Cut and we are done
   2. If have *higher dimensional* solutions, *rounding procedure*

      Random Hyperplane Cut algorithm, with high probability we get

$$cost(\text{rounded solution}) \geq 0.878 \cdot OPT(SDP) \geq 0.878 \cdot OPT(\text{Max-Cut})$$

Remarks

1. SDPs are very powerful for solving (approximating) many hard problems

Remarks

1. SDPs are very powerful for solving (approximating) many hard problems
2. Recent and exciting work, driven by *Unique Games Conjecture* (UGC), shows that if UGC is true, then all these approximation algorithms are *tight*!

   `https://www.cs.cmu.edu/~anupamg/adv-approx/lecture24.pdf`

Remarks

1. SDPs are very powerful for solving (approximating) many hard problems
2. Recent and exciting work, driven by *Unique Games Conjecture* (UGC), shows that if UGC is true, then all these approximation algorithms are *tight*!

   https://www.cs.cmu.edu/~anupamg/adv-approx/lecture24.pdf
3. Other applications in robust statistics, via the SDP & Sum-of-Squares connection

   https://arxiv.org/abs/1711.11581

Remarks

1. SDPs are very powerful for solving (approximating) many hard problems
2. Recent and exciting work, driven by *Unique Games Conjecture* (UGC), shows that if UGC is true, then all these approximation algorithms are *tight*!

   `https://www.cs.cmu.edu/~anupamg/adv-approx/lecture24.pdf`
3. Other applications in robust statistics, via the SDP & Sum-of-Squares connection

   `https://arxiv.org/abs/1711.11581`
4. Connections to automated theorem proving

   https://eccc.weizmann.ac.il/report/2019/106/

Remarks

1. SDPs are very powerful for solving (approximating) many hard problems
2. Recent and exciting work, driven by *Unique Games Conjecture* (UGC), shows that if UGC is true, then all these approximation algorithms are *tight*!

   https://www.cs.cmu.edu/~anupamg/adv-approx/lecture24.pdf
3. Other applications in robust statistics, via the SDP & Sum-of-Squares connection

   https://arxiv.org/abs/1711.11581
4. Connections to automated theorem proving

   https://eccc.weizmann.ac.il/report/2019/106/

All of these are amazing final project topics!

Conclusion

- Mathematical programming - very general, and pervasive in (combinatorial) algorithmic life

Conclusion

- Mathematical programming - very general, and pervasive in (combinatorial) algorithmic life
- Mathematical Programming hard in general

Conclusion

- Mathematical programming - very general, and pervasive in (combinatorial) algorithmic life
- Mathematical Programming hard in general
- Sometimes can get SDP rounding!

Conclusion

- Mathematical programming - very general, and pervasive in (combinatorial) algorithmic life
- Mathematical Programming hard in general
- Sometimes can get SDP rounding!
- Solve SDP and round the solution

Conclusion

- Mathematical programming - very general, and pervasive in (combinatorial) algorithmic life
- Mathematical Programming hard in general
- Sometimes can get SDP rounding!
- Solve SDP and round the solution
  - Deterministic rounding when solutions are nice
  - Randomized rounding when things a bit more complicated

our rounding will "decrease dimension" and "make it integral".

Acknowledgement

- Lecture based largely on:
  - Lecture 14 of Anupam Gupta and Ryan O'Donnell's Optimization class

    `https://www.cs.cmu.edu/~anupamg/adv-approx/`
- See their notes at

  `https://www.cs.cmu.edu/~anupamg/adv-approx/lecture14.pdf`

References I

Delorme, Charles, and Svatopluk Poljak (1993)
Laplacian eigenvalues and the maximum cut problem.
Mathematical Programming 62.1-3 (1993): 557-574.

Goemans, Michel and Williamson, David 1994
0.879-approximation algorithms for Max Cut and Max 2SAT.
Proceedings of the twenty-sixth annual ACM symposium on Theory of computing. 1994