Lecture 16: Semidefinite Programming Relaxation and MAX-CUT

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

June 17, 2024

Overview

- Max-Cut SDP Relaxation & Rounding
- Conclusion
- Acknowledgements

Relax... & Round!

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

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

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
2. Derive SDP from the QP by going to higher dimensions and imposing PSD constraint

This is called an *SDP relaxation*.

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
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.

$$OPT(SDP) \geq OPT(QP)$$

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
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.

   $$OPT(SDP) \geq OPT(QP)$$

4. Solve SDP (approximately) optimally using efficient algorithm.

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
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.

$$OPT(SDP) \geq OPT(QP)$$

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

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
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.

$$OPT(SDP) \geq OPT(QP)$$

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

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

Max-Cut

Maximum Cut (Max-Cut):

$G(V, E)$ graph.

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

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

Goal: find cut of maximum size.

Example - Weighted Variant

Maximum Cut (Max-Cut):

$G(V, E, w)$ 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.

Max-Cut

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

Quadratic Program:

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

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:

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

What is this SDP doing?

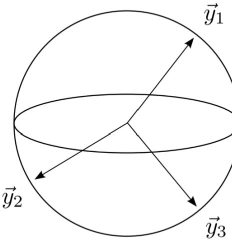

Figure 10.1: Vectors $\vec{y}_v$ embedded onto a unit sphere in $\mathbb{R}^d$.

What is this SDP doing?

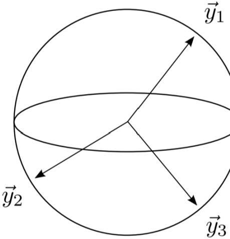

Figure 10.1: Vectors $\vec{y}_v$ embedded onto a unit sphere in $\mathbb{R}^d$.

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

What is this SDP doing?

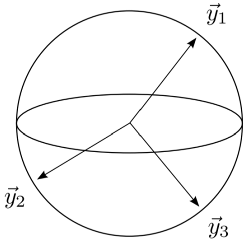

Figure 10.1: Vectors $\vec{y}_v$ embedded onto a unit sphere in $\mathbb{R}^d$.

- 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?

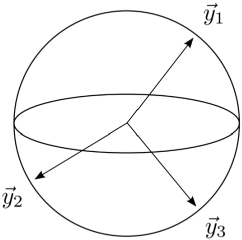

Figure 10.1: Vectors $\vec{y}_v$ embedded onto a unit sphere in $\mathbb{R}^d$.

- 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?

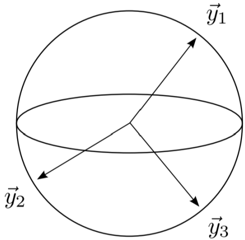

Figure 10.1: Vectors $\vec{y}_v$ embedded onto a unit sphere in $\mathbb{R}^d$.

- 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.

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

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.

Max-Cut - Rounding

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

Max-Cut - Rounding

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

Max-Cut - Rounding

1. Let $y_u \in \mathbb{R}^n$ 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 $y_u \in \mathbb{R}^n$ 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 $y_u \in \mathbb{R}^n$ 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 $x_u = \text{sign}(g^T y_u)$ as our solution

Max-Cut - Rounding

1. Let $y_u \in \mathbb{R}^n$ 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 $x_u = \text{sign}(g^T y_u)$ as our solution

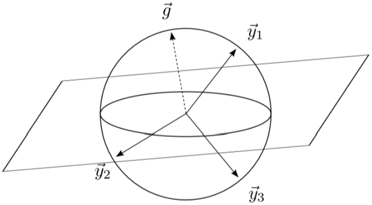

Figure 10.2: Vectors being separated by a hyperplane with normal $\vec{g}$.

Facts we need

- We can pick a random hyperplane through origin in polynomial time.

  sample vector $g = (g_1, \ldots, g_n)$ by drawing $g_i \in \mathcal{N}(0, 1)$

Facts we need

- We can pick a random hyperplane through origin in polynomial time.

  sample vector $g = (g_1, \ldots, g_n)$ by drawing $g_i \in \mathcal{N}(0, 1)$

- If g' is the projection of g onto a two dimensional plane, then $g'/\|g'\|_2$ is *uniformly distributed* over the unit circle in this plane.

Analysis of Rounding

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

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

Analysis of Rounding

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

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

- Looking at the problem in the plane:

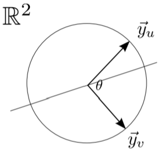

Figure 10.3: The plane of two vectors being cut by the hyperplane

Analysis of Rounding

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

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

- Looking at the problem in the plane:

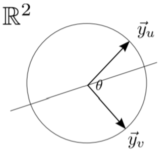

Figure 10.3: The plane of two vectors being cut by the hyperplane

- Probability of splitting y_u, y_v:

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

Analysis of Rounding

- 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

- 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

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

Analysis of Rounding

- 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

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

- If we find α such that

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

  Then we have an α-approximation algorithm!

Analysis of Rounding

- 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

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

- If we find α such that

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

  Then we have an α-approximation algorithm!
- For $x \in [-1, 1]$, we have

$$\frac{\cos^{-1}(x)}{\pi} \geq 0.878 \cdot \frac{1 - x}{2}$$

  proof by elementary calculus.

Conclusion of rounding 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*
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, we get

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

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.
   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, we get

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

   3. With constant probability, our solution will be $\geq 0.878 OPT(\text{Max-Cut})$

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
- 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

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/`
  - Chapter 6 of book [Williamson, Shmoys 2010]
- 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

Williamson, David and Shmoys, David 2010
Design of Approximation Algorithms
Cambridge University Press