CS 466/666: Algorithm Design and Analysis **University of Waterloo: Fall 2025**

Lecture 12

October 21, 2025

Instructor: Sepehr Assadi *Scribe: Helia Yazdanyar*

Topics of this Lecture

We switch from Linear Programming to an even stronger tool: *Semi-definite programming* and see how it can be used to achieve even stronger approximation algorithms for combinatorial optimization problems.

1 Recap on Linear Algebra

We start with a quick refresher on basic definitions in linear algebra.

Eigenvalues and Eigenvectors. Let $A \in \mathbb{R}^{n \times n}$. A scalar $\lambda \in \mathbb{C}$, where $\mathbb{C}$ is the set of all complex numbers, is called an *eigenvalue* of A if there exists a nonzero vector $v \in \mathbb{C}^n$ such that

$$Av = \lambda v.$$

The vector v is called an *eigenvector* corresponding to λ. The eigenvalues of A are precisely the roots of its *characteristic polynomial*

$$p_A(x) = \det(A - xI).$$

That is, λ is an eigenvalue of A if and only if $\det(A - \lambda I) = 0$.

Symmetric matrices. A matrix A is called *symmetric* if $A = A^\top$. Symmetric matrices have particularly nice spectral properties.

- All eigenvalues of a symmetric matrix are *real* numbers.

- There exists an orthogonal matrix Q and a diagonal matrix Λ such that

$$A = Q\Lambda Q^\top.$$

We will only need the first fact above which we prove below.

Fact 1. *All eigenvalues of a symmetric matrix are* real.

Proof. Let $A = A^\top$ and suppose $Av = \lambda v$ for some nonzero $v \in \mathbb{C}^n$. Then

$$v^\top Av = v^\top(\lambda v) = \lambda v^\top v.$$

We know that $A = A^\top$, and for each complex number λ we have $\lambda^\top = \overline{\lambda}$ where $\overline{\lambda}$ is the complex conjugate of λ, therefore

$$v^\top Av = (Av)^\top v = (\lambda v)^\top v = \overline{\lambda} v^\top v.$$

Comparing the two expressions gives

$$\lambda v^\top v = \overline{\lambda} v^\top v.$$

Because $v^\top v > 0$, it follows that $\lambda = \overline{\lambda}$, and thus $\lambda \in \mathbb{R}$. □

1.1 Positive Semi-Definite Matrices

A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is called **positive semi-definite (PSD)** if for all $x \in \mathbb{R}^n$,

$$x^\top Ax \geqslant 0.$$

If the inequality is strict for all nonzero x, the matrix is called *positive definite (PD)*. Let us use notation $A \succeq 0$ to show matrix A is *positive semi-definite*.

For a symmetric matrix A, the following statements are equivalent:

1. A is positive semi-definite, i.e., $x^\top Ax \geqslant 0$ for all $x \in \mathbb{R}^n$.
2. All eigenvalues of A are non-negative.
3. There exists a matrix B such that $A = B^\top B$.

Proof. We show (3) ⇒ (1) ⇒ (2) ⇒ (3).

(3) ⇒ (1): If $A = B^\top B$ then for every $x \in \mathbb{R}^n$,

$$x^\top Ax = x^\top B^\top Bx = \|Bx\|_2^2 \geqslant 0.$$

Hence (1) holds.

(1) ⇒ (2): Let λ be an eigenvalue of A with eigenvector $v \neq 0$, so $Av = \lambda v$. Then

$$v^\top Av = v^\top(\lambda v) = \lambda\, v^\top v.$$

By (1) the left-hand side is $\geqslant 0$, and $v^\top v > 0$, so $\lambda \geqslant 0$. Thus all eigenvalues are nonnegative.

(2) ⇒ (3): Since A is symmetric, it admits an orthogonal diagonalization

$$A = Q\Lambda Q^\top,$$

where Q is orthogonal and $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$ with $\lambda_i \geqslant 0$ by (2). Define $\Lambda^{1/2} = \text{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_n})$ (square roots exist because $\lambda_i \geqslant 0$). Then

$$A = Q\Lambda Q^\top = (\Lambda^{1/2}Q^\top)^\top(\Lambda^{1/2}Q^\top),$$

so with $B := \Lambda^{1/2}Q^\top$ we have $A = B^\top B$. This proves (3).

□

In other words, positive semi-definite matrices can be viewed as matrices that "behave like" squares of other matrices and have non-negative eigenvalues.

2 Semi-Definite Programming

Recall that a linear program (LP) has the following form:

$$\begin{aligned} \max_{x \in \mathbb{R}^n} \quad & \sum_{i=1}^{n} c_i \cdot x_i \\ \text{subject to} \quad & \sum_{k=1}^{n} a_i^{(k)} \cdot x_i \leqslant b^{(k)}, \qquad k \in [m], \\ & x_i \geqslant 0, \qquad\qquad\qquad i \in [n]. \end{aligned}$$

In a **Semi-definite programming (SDP)**, instead of considering $x \in \mathbb{R}^n$, we instead consider variables coming from a *matrix* $X \in \mathbb{R}^{n \times n}$ but instead of the constraints $x_i \geqslant 0$, we have that X is a PSD matrix, i.e., $X \succeq 0$.

$$\begin{aligned} \max_{X \in \mathbb{R}^{n \times n}} \quad & \sum_{i,j \in [n]} c_{ij} \cdot X_{ij} \\ \text{subject to} \quad & \sum_{i,j \in [n]} a_{ij}^{(k)} \cdot X_{ij} \leqslant b^{(k)}, \qquad k \in [m], \\ & X \succeq 0 \end{aligned}$$

In an SDP, instead of requiring $x_i \geqslant 0$, we require that a symmetric matrix X satisfies $X \succeq 0$, meaning that all its eigenvalues are non-negative. Note that this is the only constraint that treats X as a matrix, as opposed to a vector $\in \mathbb{R}^{n^2}$.

This generalization enables SDPs to capture a much broader class of convex optimization problems. At the same time, similar to LPs, one can still solve SDPs in polynomial time still (modulo some technicalities).

Theorem 2. *SDPs can be solved (up to numerical precision) in polynomial time.*

We will now see an application of SDPs to approximate combinatorial optimization problems (similar to using LP relaxations and roundings).

3 The Max-Cut Problem

Given an undirected graph $G = (V, E)$, the **Max-Cut problem** asks for a partition of the vertices S and $V \setminus S$, namely a cut S, such that the total number of edges crossing the cut (the edges between S and $V \setminus S$) is maximized. Let OPT_{MAXCUT} be the maximum cut size in G. Since Max-Cut is an NP-hard problem, we do not expect to solve it in polynomial time; instead, our goal is to find an approximation algorithm in polynomial time, i.e., to find an algorithm that outputs a cut S such that

$$\#\text{ cut edges of } S \geqslant \alpha \cdot OPT_{\text{MAXCUT}},$$

and to maximize the approximation factor α.

3.1 An Easy Algorithm

A simple randomized algorithm achieving $\alpha = 1/2$ is the following:

- For each vertex $v \in V$, independently assign v to S with probability $1/2$, and to $V \setminus S$ otherwise.

We can compute its expected cut size as

$$\begin{aligned}
\mathbb{E}[\text{size of the cut}] &= \sum_{(u,v)\in E} \Pr((u,v) \text{ is a cut edge}) && \text{(by linearity of expectation)} \\
&= \sum_{(u,v)\in E} \Pr(u \text{ and } v \text{ go to different parts}) \\
&= \sum_{(u,v)\in E} \frac{1}{2} \\
&= \frac{|E|}{2} \geqslant \frac{OPT_{\text{MAXCUT}}}{2}. && \text{(since max-cut size is at most the number of edges)}
\end{aligned}$$

But the natural question is: can we do better than $1/2$? To answer this, it is natural to formulate the Max-Cut problem as an integer linear program as follows (to relax it to an LP later):

$$\begin{aligned}
\max_{x\in\mathbb{R}^E, y\in\mathbb{R}^V} \quad & \sum_{e\in E} x_e \\
\text{subject to} \quad & x_e \leqslant y_v + y_u, && \forall (u,v) = e \in E \\
& x_e \leqslant 2 - (y_v + y_u), && \forall (u,v) = e \in E \\
& x_e \in \{0,1\}, && \forall e \in E \\
& y_v \in \{0,1\}. && \forall v \in V
\end{aligned} \tag{1}$$

We claim that this ILP correctly captures the Max-Cut problem. Let $S = \{v \in V : y_v = 1\}$ and $V \setminus S$ be the remaining vertices. We have:

- For any edge $e = (u,v)$ where $y_v = y_u = 0$, we have $x_e \leqslant y_v + y_u = 0$, so $x_e = 0$.
- For any edge $e = (u,v)$ where $y_v = y_u = 1$, we have $x_e \leqslant 2 - (y_v + y_u) = 0$, so $x_e = 0$.
- For any edge $e = (u,v)$ where $y_v \neq y_u$, we have $x_e \leqslant 1$, allowing $x_e = 1$ for a cut edge.

Thus, there is a one-to-one correspondence between optimal solutions of this ILP and the Max-Cut problem.

However, if we relax the integrality constraints to

$$\begin{aligned}
x_e &\in [0,1], \quad \forall e \in E \\
y_v &\in [0,1], \quad \forall v \in V,
\end{aligned}$$

on every graph, an optimal solution is $y_v = 1/2$ for all $v \in V$, which yields $x_e = 1$ for all $e \in E$. Thus, the LP relaxation of the Max-Cut problem basically have no useful information about the original Max-Cut problem, given regardless of the input graph, it may alway return the same optimal solution.

3.2 Quadratic Programming

We now try to formulate a program for Max-cut problem that allows multiplication of variables as well. We can represent the partitioning as below:

$$\begin{aligned} \max_{y \in \mathbb{R}^V} \quad & \frac{1}{2} \cdot \sum_{(u,v) \in E} (1 - y_v \cdot y_u) \\ \text{subject to} \quad & y_v^2 = 1, \quad \forall v \in V. \end{aligned} \tag{2}$$

This result in $y_v \in \{-1, +1\}$ for all $v \in V$, which is a partitioning of the vertices.

Claim 3. *The solution to Eq (2) solves the Max-Cut problem.*

Proof. Let S^* be the maximum cut of graph G, for each $v \in S^*$ set $y_v = 1$ and for all the remaining vertices $u \in V \setminus S^*$ let $y_u = -1$. We have,

$$\frac{1}{2} \cdot \sum_{(u,v) \in E} (1 - y_v \cdot y_u) = \frac{2 \cdot |\text{size of cut } S^*|}{2} = |\text{size of cut } S^*| \,.$$

As for all $v \in V$, we have $y_v^2 = 1$, this is a feasible solution to Eq (2). Therefore the optimal solution (OPT_Q) for this optimization problem is at least as large as the maximum cut size:

$$OPT_Q \geqslant |\text{size of cut } S^*| \,.$$

On the other hand, let $y^* \in \mathbb{R}^n$ be the optimal solution to Eq (2). For all $v \in V$ we have $(y_v^*)^2 = 1$ where (y_v^*) is a real number, therefore $(y_v^*) \in \{-1, +1\}$. Let $S = \{v \in V \mid (y_v^*) = 1\}$ for the $y^* \in \{-1, 1\}^n$ that optimizes Eq (2). As S is a cut where

$$|\text{size of cut } S| = OPT_Q \geqslant |\text{size of cut } S^*| \,,$$

therefore S is the maximum cut, concluding the proof. □

This formulation shows that Max-Cut is a *quadratic optimization problem* over discrete variables, which is NP-hard to solve exactly. Thus, we are still seemingly no closer to our original plan of approximating the Max-Cut problem.

Nevertheless, a major breakthrough in approximation algorithms came from Goemans and Williamson [GW95]: they showed that by relaxing the constraint $y_v \in \{-1, +1\}$ to *vectors* $b_v \in \mathbb{R}^n$ with $\|b_v\| = 1$ through formulating the problem as an SDP, we can round these vectors to achieve a 0.878-approximation ratio — the best known to this day.

3.3 SDP for Max-Cut

Consider any vector $y \in \mathbb{R}^V$ and define the matrix

$$X = y \otimes y^\top,$$

the outer product ($\otimes$) of y and $y^\top$, where

$$y = \begin{bmatrix} y_{u_1} \\ y_{u_2} \\ \vdots \\ y_{u_n} \end{bmatrix}.$$

Expanding the outer product, we get

$$y \otimes y^\top = \begin{bmatrix} y_{u_1} \\ y_{u_2} \\ \vdots \\ y_n \end{bmatrix} \begin{bmatrix} y_{u_1} & y_{u_2} & \cdots & y_{u_n} \end{bmatrix} = \begin{bmatrix} y_{u_1}^2 & y_{u_1} y_{u_2} & \cdots & y_{u_1} y_{u_n} \\ y_{u_2} y_{u_1} & y_{u_2}^2 & \cdots & y_{u_2} y_{u_n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{u_n} y_{u_1} & y_{u_n} y_{u_2} & \cdots & y_{u_n}^2 \end{bmatrix},$$

where y_{u_i} corresponds to vertex u_i for each $i \in [n]$. In particular, for distinct vertices $u \neq v$, we have $X_{uv} = y_u \cdot y_v$, and for any vertex $u \in V$, $X_{uu} = y_u^2$.

Using this notation, we can rewrite the quadratic Max-Cut formulation as

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2} \sum_{(u,v)\in E} \big(1 - X_{uv}\big) \\ \text{subject to} \quad & X_{vv} = 1, \quad \forall v \in V \\ & X = y \otimes y^\top \text{ for some vector } y \in \mathbb{R}^V. \end{aligned} \tag{3}$$

Given this program is identical to the original quadratic program in (2), the following claim is immediate.

Claim 4. *Any solution X^* to Eq (3) is a maximum cut.*

By definition, $X = y \otimes y^\top$ is symmetric and positive semi-definite (PSD), i.e., $X \succeq 0$. Adding the constraint $X = y \otimes y^\top$ is what makes the above program problematic. However, given that the variables X we work with is a PSD matrix, maybe we can *relax* the problem, namely, turn that constraint to $X \succeq 0$? This way, we will have the following SDP:

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2} \sum_{(u,v)\in E} \big(1 - X_{uv}\big) \\ \text{subject to} \quad & X_{vv} = 1, \quad \forall v \in V \\ & X \succeq 0. \end{aligned} \tag{4}$$

We can see that in fact this is an SDP, so we managed to write an SDP for Maximum Cut. As this is a relaxation of constraints for Eq (3): assuming OPT_{SDP} is the optimal answer for Eq (4), we have

$$OPT_{\text{SDP}} \geqslant OPT_{\text{MAXCUT}}.$$

This is simply because by Claim 4 we know maximum cut is a feasible solution X inside our program and since this program is maximizing the objective value, it can even pick "better" PSD matrices X to increase the objective value.

Recall that for any PSD matrix X there exists some matrix B such that $X = B^\top B$. So, If we write $X = B^\top B$ in (4) it will be as follows:

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2} \sum_{(u,v)\in E} \big(1 - \langle b_u, b_v \rangle\big) \\ \text{subject to} \quad & \|b_v\| = 1, \quad \forall v \in V. \end{aligned} \tag{5}$$

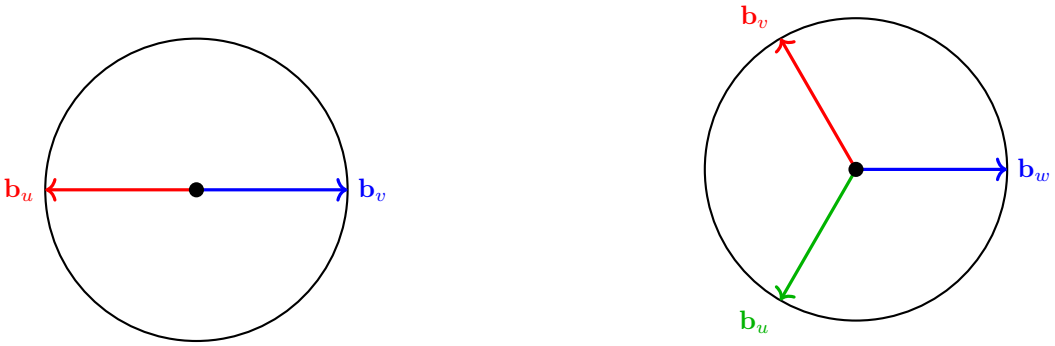
We will be solving the previous one using an SDP solver but for our purpose, we can just look at it as obtaining an optimal solution to the new program instead since they are equivalent.

The program (5) can be seen as a relaxation of the quadratic program (2) since we "relaxed" each number to a *vector*, i.e. we replaced each scalar $y_v \in \{-1, 1\}$ with a vector $b_v \in \mathbb{R}^n$ of unit length, so that $y_u y_v$ is replaced by the inner product $\langle b_u, b_v \rangle$. This is in a way similar to relaxing an ILP where we replaced integers by real numbers.

3.4 Algorithm for Max-Cut

As we can see in (5), for each $v \in V$, the corresponding vector has $\|b_v\| = 1$ which means they are all in a same unit n-dimensional sphere. In order to maximize the objective, every edge (u, v) pushes vectors b_u

and b_v away from each other. For example if the graph only consists of two vertices $\{u, v\}$ and edge (u, v), the resulting vectors would be on a same line in different directions (see Figure 1a). And if the graph is a triangle (a 3-clique), then the vectors would balance in a point were they have the same distance to every other vertex (see Figure 1b).



(a) Graph with one edge

(b) 3-clique

Figure 1: Optimal solutions of SDP on two very basic graphs.

Having these vectors for vertices of the graph balanced as described, the intuition is to cut this n-dimensional ball with a vector passing through the center of this ball so that we cut maximum number of edges (separate their corresponding vertices).

Algorithm 1. Maximum Cut Algorithm

1. Solve SDP to get vectors $b_{u_1}, \ldots, b_{u_n}$.
2. Pick a random vector $r \in \mathbb{R}^n$ such that $\|r\| = 1$.[a]
3. Let $S = \{v \mid \langle r, b_v \rangle \geqslant 0\}$.

[a] This is not a trivial step, we can find r by first sampling each coordinate r_i independently from a standard normal distribution $\mathcal{N}(0, 1)$, and then normalizing the resulting vector. We postpone the details to later lectures.

We pick a random vector r to avoid relying on any specific direction when defining our cut. Intuitively, the solutions to Eq (5) can be located anywhere inside the unit ball, and their relative orientation may vary across different instances. As a result, choosing a fixed direction could lead to unbalanced or uninformative cuts, whereas picking r uniformly at random ensures that, on average, we obtain a direction that fairly reflects the geometric structure of the solution.

Theorem 5. *Algorithm 1 outputs a cut that in expectations is a* 0.8756*-approximation for maximum cut problem.*

Proof. The expected number of edges cut by this algorithm is

$$\begin{aligned}\mathbb{E}\left[\text{size of cut}\right] &= \sum_{(u,v)\in E} \Pr((u, v) \text{ is cut}) \\ &= \sum_{(u,v)\in E} \Pr(u \text{ and } v \text{ are on different sides})\end{aligned}$$

We now fix an edge $(u, v) \in E$ and attempt to bound the probability above. To understand this probability geometrically, recall that in our two-dimensional illustrations we visualize the vectors on a 2D circle. This

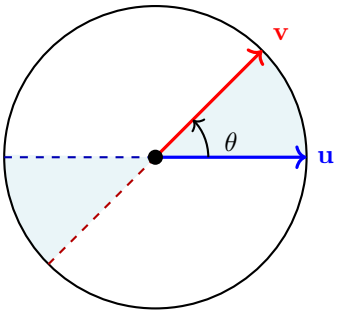


Figure 2: Probability of separating vectors of vertices u and v

circle represents the intersection of the n-dimensional unit sphere with some two-dimensional plane passing through the origin.

When we sample a random vector r uniformly from the n-dimensional unit sphere, the hyperplane orthogonal to r passes through the origin and intersects the sphere, which we denote by C_r. For any edge (u, v), the vectors b_u and b_v lie on a unique two-dimensional plane through the origin; let C_{uv} be intersection of this plane with the n-dimensional sphere containing both b_u and b_v (which is a circle). Let ℓ be the projection of the hyperplane orthogonal to r into C_{uv}: basically, this is the *line* that "cuts" this circle into the two halves based on r.

Because r is chosen uniformly on the n-dimensional sphere, the line ℓ is uniformly distributed in angle, namely, is a diagonal of the circle chosen uniformly[1]. Hence, in our 2D depiction, ℓ cuts the circle uniformly in direction. Let θ_{uv} denote the angle between b_u and b_v on C_{uv}. Then the probability that ℓ lies between these two vectors (so that u and v fall on opposite sides of the hyperplane) is as below (see Figure 2):

$$\Pr(u \text{ and } v \text{ are on different sides}) = \frac{2\theta_{uv}}{2\pi} = \frac{\theta_{uv}}{\pi}.$$

Finally, since

$$\langle b_u, b_v \rangle = \|b_u\| \cdot \|b_v\| \cdot \cos\theta_{uv}.$$

and each vector has unit norm, we have $\cos\theta_{uv} = \langle b_u, b_v \rangle$, which gives

$$\Pr(u \text{ and } v \text{ are in different sides}) = \frac{\arccos\langle b_u, b_v \rangle}{\pi}.$$

As a result,

$$\mathbb{E}[\text{size of cut}] = \sum_{(u,v)\in E} \frac{\arccos\langle b_u, b_v \rangle}{\pi}.$$

On the other hand, in the program (5), we know

$$OPT = \frac{1}{2} \cdot \sum_{(u,v)\in E} (1 - \langle b_u, b_v \rangle),$$

where $OPT = OPT_{\text{SDP}} \geqslant OPT_{\text{MAXCUT}}$. Therefore, it is sufficient to find an α that satisfies

$$\frac{\arccos\langle b_u, b_v \rangle}{\pi} \geqslant \alpha \cdot \frac{1}{2} \cdot (1 - \langle b_u, b_v \rangle);$$

if we find such an α, then we will have

$$\mathbb{E}[\text{size of cut}] = \sum_{(u,v)\in E} \frac{\arccos\langle b_u, b_v \rangle}{\pi} \geqslant \sum_{(u,v)\in E} \alpha \cdot \frac{1}{2} \cdot (1 - \langle b_u, b_v \rangle) = \alpha \cdot OPT \geqslant \alpha \cdot OPT_{\text{MAXCUT}},$$

[1] This statement is intuitive but requires a proof; however, the proof is quite orthogonal to the topics of this lecture/course and we will omit it.

hence, giving us an α-approximation.

To calculate α, we can do as follows. As $\langle b_u \, , \, b_v \rangle \in [-1, 1]$ always (when $\|b_u\| = \|b_v\| = 1$), we can define a new variable z to replace $\langle b_u \, , \, b_v \rangle$ and thus, we will need to have that for all $z \in [-1, 1]$:

$$\frac{\arccos(z)}{\pi} \geqslant \alpha \cdot \frac{1}{2} \cdot (1 - z);$$

hence, we can set α as:

$$\alpha = \min_{z \in [-1,1]} \frac{2 \arccos(z)}{\pi \cdot (1 - z)}.$$

Solving this inequality numerically[2], we get $\alpha = 0.8756$ concluding the proof. □

References

[GW95] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, November 1995. 5

[2] E.g., use WolframAlpha:

https://www.wolframalpha.com/input?i=min+2*arccos%28z%29%2F%28pi+*+%281-z%29%29+for+z+in+%5B-1%2C%2B1%5D