Chapter 11

Semidefinite Programming I: Maximum Cut

We switch from linear programming to an even stronger tool: *Semidefinite programming (SDP)* and see an example of how it can be used to achieve even stronger approximation algorithms for combinatorial optimization problems.

11.1 Recap on Linear Algebra

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

Definition 11.1. 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* defined as:

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

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 properties when it comes to their eigenvalues.

Fact 11.2. *All eigenvalues of a symmetric matrix in $\mathbb{R}^{n \times n}$ are* real.

Fact 11.3. *Every symmetric matrix in $\mathbb{R}^{n \times n}$ admits an orthogonal diagonalization, i.e., there exists an orthogonal matrix Q such that*

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

where Λ is a diagonal matrix whose entries are the eigenvalues of A.

Positive semidefinite matrices. The key definition we need is the following.

Definition 11.4. A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is called **positive semidefinite (PSD)** if for all $x \in \mathbb{R}^n$,

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

We use the notation $A \succeq 0$ to denote that A is PSD.

The following provides a further charactrization of PSD matrices.

Proposition 11.5. *For a symmetric matrix A, the following statements are equivalent:*

1. *A is PSD, 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) $\Rightarrow$ (1) $\Rightarrow$ (2) $\Rightarrow$ (3).

(3) $\Rightarrow$ (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) $\Rightarrow$ (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 LHS is non-negative, and we always have $v^\top v \geqslant 0$, so $\lambda \geqslant 0$. Thus all eigenvalues are nonnegative.

(2) $\Rightarrow$ (3): Since A is symmetric, by Fact 11.3, it admits an orthogonal diagonalization

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

where Q is orthogonal and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ with $\lambda_i \geqslant 0$ by (2). Let $\Lambda^{1/2} = \mathrm{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_n})$ (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), concluding the whole proof. □

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

11.2 Semidefinite Programming

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

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

In **semidefinite programming (SDP)**, instead of considering $x \in \mathbb{R}^n$, we work with variables coming from a *PSD matrix* $X \in \mathbb{R}^{n \times n}$. Formally,

Definition 11.6. In a **semidefinite program (SDP)**, we have an $(n \times n)$-matrix of **variables** $X \in \mathbb{R}^{n \times n}$, an $(n \times n)$-dimensional **objective function** $c \in \mathbb{R}^{n \times n}$, and a set of m **constraints**, each given by a matrix $A^{(k)} \in \mathbb{R}^{n \times n}$ and scalar $b^{(k)} \in \mathbb{R}$. The goal is:

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

Note that the only difference between an SDP and an LP is that, 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 (this is the only constraint that treats X as a matrix, and not 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 modulo numerical precision issues (unlike LPs, the optimal solution to an SDP may not have polynomial bit-length and in fact, it can even contain irrational numbers).

Theorem 11.7 (cf. [GLS81])**.** *Any semidefinite program specified by $c \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^m$, and $\{A^{(k)} \in \mathbb{R}^{n \times n}\}_{k=1}^m$ (Definition 11.6) can be approximated to a $(1-\varepsilon)$ factor, for $\varepsilon \in (0,1)$, in time polynomial in n, m, bit-length of entries in $\{A^{(k)}\}, b, c$, and $\log(1/\varepsilon)$ (or, even with a separation oracle in time independent of the number of constraints m).*

In this and the next chapter, we will review applications of SDPs to approximating combinatorial optimization problems similar to using LP relaxations and rounding.

11.3 The Maximum Cut (Max-Cut) Problem

We now see a canonical application of SDPs to solving the maximum cut problem.

Problem 11.8. Given an undirected graph $G = (V, E)$, the maximum cut problem (Max-Cut) 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_{MC} 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

$$\#\text{ cut edges of } S \geqslant \alpha \cdot \text{opt}_{\text{MC}},$$

and to maximize the approximation factor α.

11.3.1 An Easy Algorithm and a Linear Program

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

Algorithm 11.9.

For each vertex $v \in V$, independently assign v to S or $V \setminus S$ chosen uniformly at random.

Lemma 11.10. *Algorithm 11.9 outputs a cut of expected size at least half* $\mathrm{opt}_{\mathrm{MC}}$.

Proof. We can compute its expected cut size as

$$\begin{aligned}
\mathbb{E}\left[\text{size of the cut}\right] &= \sum_{(u,v)\in E} \Pr((u,v) \text{ is a cut edge}) && \text{(by linearity of expectation)} \\
&= \sum_{(u,v)\in E} \Pr(u \text{ and } v \text{ go to different parts}) \\
&= \sum_{(u,v)\in E} \frac{1}{2} && \text{(as each vertex chooses a side uniformly and independently)} \\
&= \frac{|E|}{2} \geqslant \frac{\mathrm{opt}_{\mathrm{MC}}}{2}. && \text{(since max-cut size is at most the number of edges)}
\end{aligned}$$

□

Beyond half approximation. The natural question at this point is: can we do better than 1/2-approximation? 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):

$$\begin{aligned}
\max_{x\in\mathbb{R}^E, y\in\mathbb{R}^V} \quad & \sum_{e\in E} x_e \\
\text{subject to} \quad & x_e \leqslant y_v + y_u, && \forall (u,v) = e \in E \\
& x_e \leqslant 2 - (y_v + y_u) && \forall (u,v) = e \in E \\
& x_e \in \{0,1\} && \forall e \in E \\
& y_v \in \{0,1\} && \forall v \in V
\end{aligned} \tag{11.1}$$

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 has no useful information about the original Max-Cut problem, given that regardless of the input graph, it can always return the same optimal solution.

11.3.2 Quadratic Programming

We now try to formulate a program for the Max-Cut problem that allows multiplication of variables as well. With this "extra power", we can represent the problem instead as follows:

$$\begin{aligned} \max_{y \in \mathbb{R}^V} \quad & \frac{1}{2} \cdot \sum_{(u,v) \in E} (1 - y_v \cdot y_u) \\ \text{subject to} \quad & y_v^2 = 1, \quad \forall v \in V. \end{aligned} \tag{11.2}$$

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

Claim 11.11. *The solution to Eq (11.2) 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 $y_u = -1$. For any cut S, let $\delta(S)$ denote the cut edges of S. We have,

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

As for all $v \in V$, we have $y_v^2 = 1$, this is a feasible solution to Eq (11.2). Therefore the optimal solution (opt_{QP}) for this optimization problem is at least as large as the maximum cut size:

$$\text{opt}_{\text{QP}} \geqslant |\delta(S^*)|.$$

On the other hand, let $y^* \in \mathbb{R}^n$ be the optimal solution to Eq (11.2). 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 (11.2). As S is a cut with

$$|\delta(S)| = \text{opt}_{\text{QP}} \geqslant |\delta(S^*)|,$$

we have that S is a maximum cut, concluding the proof. □

This formulation shows that Max-Cut can be captured by a **quadratic optimization problem** over discrete variables. But unfortunately, quadratic programming is an **NP**-hard problem. Thus, we are still seemingly no closer to our original plan of approximating the Max-Cut problem.

11.4 A Semidefinite Programming Approach for Max-Cut

A major breakthrough in approximation algorithms came from [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. We review this algorithm in this section.

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

$$X = y \otimes y,$$

the outer product ($\otimes$) of y with itself ($y \otimes y = y \cdot y^\top$), where

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

Expanding the outer product, we get

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

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

$$\begin{aligned} \max_{X \in \mathbb{R}^{n \times n}} \quad & \frac{1}{2} \sum_{(u,v) \in E} (1 - X_{uv}) \\ \text{subject to} \quad & X_{vv} = 1, \quad \forall v \in V \\ & X = y \otimes y \text{ for some vector } y \in \mathbb{R}^V. \end{aligned} \tag{11.3}$$

Given this program is identical to the original quadratic program in (11.2), the following claim is immediate.

Claim 11.12. *Any solution X^* to Eq (11.3) is a maximum cut.*

By definition, $X = y \otimes y$ is symmetric and positive semi-definite (PSD), i.e., $X \succeq 0$. Adding the constraint $X = y \otimes y$ 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:

$$\begin{aligned} \max_{X \in \mathbb{R}^{n \times n}} \quad & \frac{1}{2} \sum_{(u,v) \in E} (1 - X_{uv}) \\ \text{subject to} \quad & X_{vv} = 1, \quad \forall v \in V \\ & X \succeq 0. \end{aligned} \tag{11.4}$$

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 (11.3): assuming opt_{SDP} is the optimal answer for Eq (11.4), we have

$$\text{opt}_{\text{SDP}} \geqslant \text{opt}_{\text{MC}}.$$

This is simply because by Claim 11.12 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.

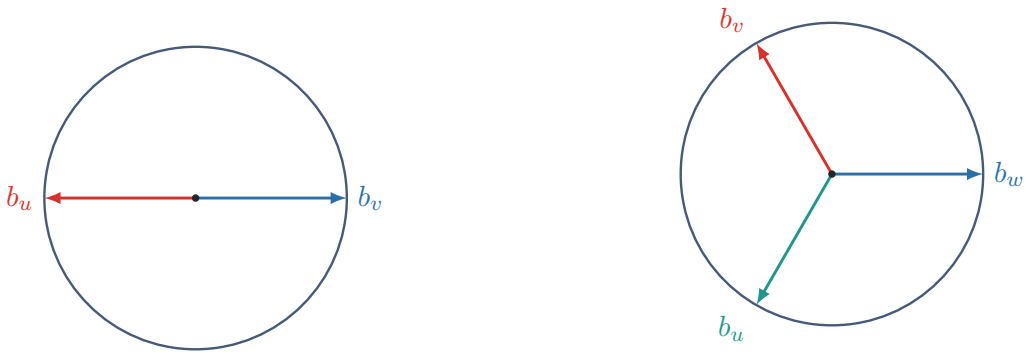
Recall that, by Proposition 11.5, for any PSD matrix X there exists some matrix B such that $X = B^\top B$. So, if we write $X = B^\top B$ in (11.4) it will be as follows:

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2} \sum_{(u,v) \in E} \left(1 - b_u^\top \cdot b_v\right) \\ \text{subject to} \quad & \|b_v\|_2 = 1, \quad \forall v \in V. \end{aligned} \tag{11.5}$$

Notice that Eq (11.4) and Eq (11.5) are equivalent and thus by solving the former using SDP solvers, we obtain a solution for the latter as well.

The program (11.5) can be seen as a relaxation of the quadratic program (11.2) 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 $y_u \cdot y_v$ 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.

Before moving on, it is worth exploring this formulation a bit more. As we can see in (11.5), for each $v \in V$, the corresponding vector has $\|b_v\|_2 = 1$ which means all b_v-vectors lie on the 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 the same line in opposite directions (see Figure 11.1a). And if the graph is a triangle (a 3-clique), then the vectors would balance in a point where they have the same distance to every other vertex (see Figure 12.1).



(a) Graph with one edge **(b)** 3-clique

Figure 11.1: Optimal solutions of the SDP on two basic graphs.

11.4.1 The Algorithm for Max-Cut

We now present an approximation algorithm for Max-Cut by rounding the SDP solution. Recall our intuition from the previous subsection that the SDP aims to "separate" the vectors of endpoints of edges—sitting in an n-dimensional sphere—as much as possible. Thus, the intuition behind the algorithm is to cut this n-dimensional sphere with a vector passing through its center so that we cut the maximum number of edges (separate their corresponding vertices).

Algorithm 11.13.

1. Solve the SDP Eq (11.4) (using Theorem 11.7) and use the equivalent formulation of Eq (11.5) 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 $S = \{v \mid r^\top \cdot b_v \geqslant 0\}$.

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

Notice that the solutions to the SDP Eq (11.5) are rotation invariant and thus choosing the random vector r in Algorithm 11.13 also allows us to avoid relying on any specific direction when defining our cut.

Theorem 11.14 ([GW95])**.** *Algorithm 11.13 outputs a cut that in expectation is a* 0.878*-approximation for the maximum cut problem.*

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

$$\mathbb{E}\left[\text{size of cut}\right] = \sum_{(u,v)\in E} \Pr((u,v) \text{ is cut}) = \sum_{(u,v)\in E} \Pr(u \text{ and } v \text{ are on different sides}). \quad (11.6)$$

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 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 H_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 the 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 H_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 diameter 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}. See Figure 11.2 for an illustration.

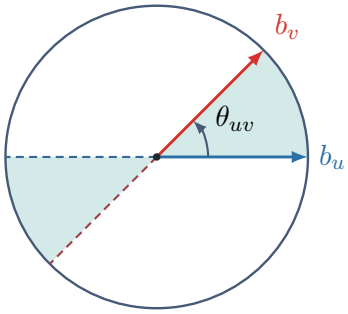


Figure 11.2: Probability of separating the vectors of vertices u and v.

Using the above, the probability that ℓ lies between these two vectors (so that u and v fall on opposite sides of the hyperplane) is:

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

Finally, since

$$b_u^\top \cdot b_v = \|b_u\| \cdot \|b_v\| \cdot \cos\theta_{uv}$$

and each vector has unit norm, we have $\cos\theta_{uv} = b_u^\top \cdot b_v$. This in turn gives us

$$\Pr(u \text{ and } v \text{ are on different sides}) = \frac{\arccos\left(b_u^\top \cdot b_v\right)}{\pi}.$$

Putting this in Eq (11.6) gives us

$$\mathbb{E}\left[\text{size of cut}\right] = \sum_{(u,v)\in E} \frac{\arccos\left(b_u^\top \cdot b_v\right)}{\pi}.$$

[1]This statement is intuitive but needs a proof; however, the proof is orthogonal to our topics and is omitted.

On the other hand, for the optimal solution value opt_{SDP} of the program (11.5), we know[2].

$$\text{opt}_{\text{SDP}} = \frac{1}{2} \cdot \sum_{(u,v)\in E} (1 - b_u^\top \cdot b_v).$$

Since $\text{opt}_{\text{SDP}} \geqslant \text{opt}_{\text{MC}}$, it is sufficient to find an α that satisfies

$$\frac{\arccos\left(b_u^\top \cdot b_v\right)}{\pi} \geqslant \alpha \cdot \frac{1}{2} \cdot (1 - b_u^\top \cdot b_v),$$

for all choices of $\|b_u\| = \|b_v\| = 1$. Assuming we find this α, we will have,

$$\mathbb{E}\left[\text{size of cut}\right] = \sum_{(u,v)\in E} \frac{\arccos\left(b_u^\top \cdot b_v\right)}{\pi} \geqslant \sum_{(u,v)\in E} \alpha \cdot \frac{1}{2} \cdot (1 - b_u^\top \cdot b_v) = \alpha \cdot \text{opt}_{\text{SDP}} \geqslant \alpha \cdot \text{opt}_{\text{MC}},$$

hence, giving us an α-approximation.

To calculate α, we can do as follows. Since $\|b_u\| = \|b_v\| = 1$, we have $b_u^\top \cdot b_v \in [-1, 1]$ always and thus we can define a new variable z to replace $b_u^\top \cdot b_v$; our task is now to find an α where 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[3], we get $\alpha \geqslant 0.878$ concluding the proof. □

Remark. The approximation ratio obtained by [GW95] (Theorem 11.14) remains the state of the art for this problem. Subsequent to [GW95], it was shown by [KKMO04] that this approximation ratio—despite looking "unorthodox" numerically, at least at first glance—is not at all arbitrary: under the **Unique Games Conjecture (UGC)**[a], this approximation ratio is in fact optimal for any polynomial-time algorithm.

[a] This is a conjecture, put forward by [Kho02], which is weaker than $\mathbf{P} \neq \mathbf{NP}$, but is still quite believable.

[2] Technically speaking, using Theorem 11.7, we will not be able to find the optimum solution for the SDP but rather a very close approximation to it, say, by setting $\varepsilon = 10^{-10}$ or even $\varepsilon = n^{-1}$; the resulting error will then be negligible in the numbers we calculate below and we will ignore it for simplicity.

[3] E.g., by using WolframAlpha.