Approximating the Maximum Cut: Theory with Computer Experiments

Yiyang Su

April 24, 2021

Abstract

In mathematics and theoretical computer science, we frequently encounter optimization problems some or all of whose variables are restricted to integers. They are frequently referred to as *integer programs*. Even though there is no known polynomial-time algorithm for integer programs, we are able to approximate solutions to integer programs efficiently. In this paper, we focus on one particular integer program, finding the maximum cut in an arbitrary graph and explore various approximation strategies and visualize them using computer programs.

1 Introduction

1.1 Maximum Cut

In mathematics and computer science, there are many computationally hard problems, for instance, finding the maximum cut of an arbitrary graph, which is formulated below.

Definition 1.1 (Undirected graph)**.** An undirected graph $G = (V, E)$ is defined as a set V of vertices together with a set E of edges where each edge is an unordered pair of vertices.

An example of an undirected graph is provided in Figure 1. In this example, there are 6 vertices and 6 edges. According to our definition above, for this graph we have

$$V = \{1, 2, 3, 4, 5, 6\}$$
$$G = \{(1, 2), (1, 3), (2, 4), (3, 4), (4, 5), (3, 6)\}$$

For simplicity, we restrict our focus to finite simple graphs, graphs with a finite number of edges and without loop or multiple edges. Throughout this paper, we use n to denote $|V|$, the number of vertices of a graph, unless otherwise specified. The graph being referred to should be clear from the context.

When studying graphs, a popular problem to contemplate is the max-cut problem that is formulated below.

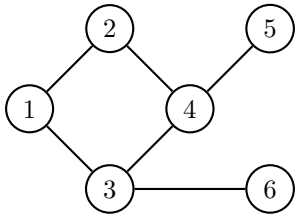

Figure 1: A graph with 6 vertices and 6 edges.

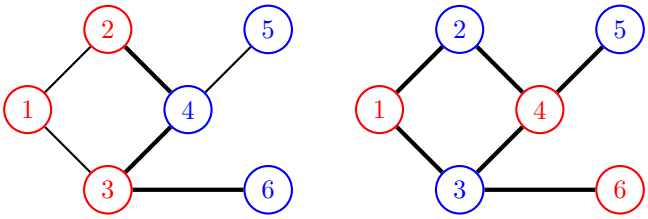

Figure 2: Two cuts on the same graph.

Definition 1.2 (Maximum cut)**.** Suppose that we partition the set of vertices of a graph G into two disjoint sets. The cut is the number of edges crossing between these two sets. The maximum cut of G, denoted MAX-CUT(G), is obtained by maximizing the cut over all partitions of vertices.

Two cuts on the same graph are illustrated in Figure 2. The partitions are illustrated in red and blue, respectively. The graph on the left represents the partition

$$\{1, 2, 3\}, \{4, 5, 6\},$$

and there are 3 edges in the graph crossing between the two partitions that are bold. In the graph on the right, the maximum cut for this graph is illustrated. For these two partitions of this graph, all edges are crossing between partitions. The partition is

$$\{1, 4, 6\}, \{2, 3, 5\}.$$

It is natural to encode a partition as $p \in \{-1, 1\}^n$ so that the two partitions are given by

$$P_0 = \{v \in V | p_v = -1\}, \tag{1}$$
$$P_1 = \{v \in V | p_v = 1\}. \tag{2}$$

In this way, if we uniformly sample a partition, then a partition is essentially a n-dimensional random vector whose coordinates are independent Rademacher random variables with parameter $\frac{1}{2}$. Recall that a random variable is said to

have the Rademacher distribution if it takes value 1 and -1 each with probability $\frac{1}{2}$.

Without loss of generality, we can let $V = \{1, 2, \ldots, n-1\}$ and rewrite E as an adjacency matrix M of size $n \times n$ such that for any $i, j \in V$,

$$M_{i,j} = \begin{cases} 1 & (i,j) \in E, \\ 0 & (i,j) \notin E. \end{cases} \tag{3}$$

Notice that for undirected graphs, since the edges are unordered pairs, we expect M to be symmetric with respect to its main diagonal. Moreover, since we assume that G is a simple graph, the main diagonal of M should be uniformly 0.

Finally, the cut is given by the number of edges that cross the cut. By our representation of partitions, two vertices u, v are in the same partition if and only if $p_u p_v = 1$ and not in the same partition if and only if $p_u p_v = -1$. Therefore, the is the number of edges (u, v) such that $p_u p_v = -1$. Hence, the cut is given by

$$\begin{aligned} \mathrm{Cut}(G, p) &= \sum_{u \in P_0} \sum_{v \in P_1} M_{uv} && (4) \\ &= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} M_{ij} 1_{p_i p_j = -1} \\ &= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} M_{ij} \frac{1 - p_i p_j}{2}, && (5) \end{aligned}$$

where the factor in front accounts for the fact that we count each undirected edge twice and we divide $1 - p_i p_j$ by 2 to scale it such that it is 1 when i, j are in different partitions and 0 otherwise.

Note that we have formulated the problem of finding the maximum cut as an integer program, where want to maximize a function of $\{p_i\}_{i=1}^n$ and each p_i is restricted to ± 1.

1.2 Approximation

Intuitively, finding the maximum cut is computationally hard because an exponentially large number of possible cuts to find the maximum and not many possibilities can be pruned. In complexity theory, we say that finding the maximum cut is NP-hard.

NP-hard problems presents significant computational challenges. Not only do we not yet know whether there exists a polynomial-time algorithm for finding the maximum cut, but many computer scientists also believe that no such algorithm exists. Therefore, the necessity for an approximation algorithm arises when we want solve computationally hard problems efficiently while tolerating some error.

One distinctive feature of approximation algorithms is that they have provable error bounds in the worst case. It sets them apart from many other

approaches like local search and greedy algorithms, which does not have any guarantee on the magnitude of the error in the worst case. In general, such algorithms may be stuck in any arbitrary local optimum.

However, not all approximation algorithms are created equal. Some produce more accurate result than others. As judging approximation using observation and intuition is subjective and error-prone, we introduce a formal definition of the worst case "error".

Definition 1.3 (δ-approximation)**.** Given an optimization problem, we say an approximation algorithm is a δ-approximation if it always runs in polynomial time and the resulting approximate value is always at least δ times the optimal value.

Generally speaking, the value of δ can serve as a measure of the quality of the approximation algorithm - the closer the value of δ is to 1, the more accurate the approximation algorithm.

In the rest of this paper, we introduce to approximation algorithms for the maximum cut problem, a simple 0.5-approximation algorithm and a 0.878-approximation algorithm based on semidefinite relaxation.

2 0.5-Approximation Algorithm

In this section, we present a simple 0.5-approximation algorithm for finding the maximum cut of a graph.

Theorem 2.1 (The 0.5-approximation algorithm for maximum cut)**.** Given an undirected graph G with n vertices and partition the vertices of G into two sets at random, uniformly over all 2^n partitions. Then, the expectation of the resulting cut equals $0.5|E|$.

Proof. We start from Equation 5. When $i \neq j$, by simple calculation, we have $\mathrm{E}p_i p_j = 0$ as they are independent Rademacher random variables. Recall that a random variable is said to have the Rademacher distribution if it takes values 1 and -1, each with probability $\frac{1}{2}$. Also, when $i = j$, we know $M_{ij} = 0$ as we assumed in advance that there is no loop in our graph. Hence,

$$\mathrm{E}\,\mathrm{Cut}(G, p) = \frac{1}{4}\mathrm{E}\left[\sum_{i=1}^{n}\sum_{j=1}^{n} M_{ij}\right] = \frac{1}{2}|E|.$$

QED

We perform experiments to illustrate Theorem 2.1. In our experiments, we use exactly the same representation that we developed above except that we let $p \in \{0, 1\}^n$ instead. In other words, to generate partitions uniformly over all 2^n partitions, we let $p_v = 1$ with probability $\frac{1}{2}$ and $p_v = 0$ with probability $\frac{1}{2}$ for any $v \in V$. And to reduce computational overhead, we directly use Equation 4 to calculate the cuts.

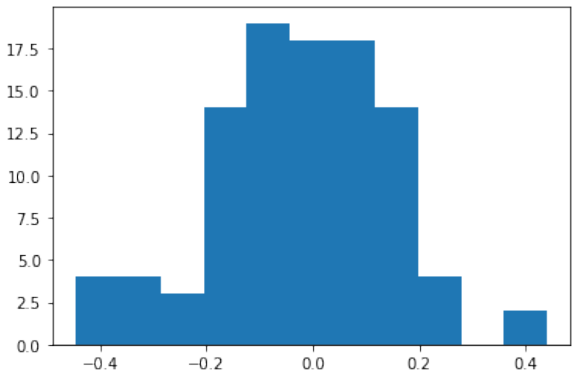

Figure 3: Plot of the values of c' of 100 randomly generated graphs. From the graph, it is easy to see that the values are clustered around 0.

In our experiments, we generate 100 random graphs with 20 vertices. For each graph, we use the average of 1000 randomly sampled cuts $\tilde{c}$ to approximate the expectation and calculate

$$c' = \tilde{c} - \frac{m}{2}. \tag{6}$$

The values of c' of the graphs are plotted in Figure 3 and the code to generate the figure is provided in Appendix A.1. Theorem 2.1 predicts that c' has expectation 0. From the graphs, we observe that the values are clustered around 0, which agrees with the theoretical results.

Theorem 2.1 proves that the maximum cut is at least half of the number the edges. If not, when we calculate the expected cut from the finite number of partitions, the expected cut would be strictly less than $0.5|E|$, contradicting Theorem 2.1. This theorem offers a 0.5-approximation algorithm because it always runs in polynomial time (the least efficient step is counting the number of edges) and the approximate value is always half of the optimal value.

3 Semidefinite Relaxation

3.1 Semidefinite Program

In this section, we discuss how we can employ the powerful tool of semidefinite relaxation to obtain a much better approximation algorithm for finding the maximum cut for an arbitrary graph.

First, we introduce the notion of semidefinite matrices and semidefinite program.

Definition 3.1. A matrix X is said to be positive semidefinite if it satisfies the following properties.

i) for any $1 \leq i, j \leq n$, $X_{i,j}$ equals $\overline{X_{j,i}}$, the complex conjugate of $X_{i,j}$;

ii) there exists a $m \times n$ matrix V such that $X = V^{\mathsf{T}}V$.

We use the notation $X \succeq 0$ to denote that X is a semidefinite matrix.

Positive semidefinite matrices have a wide range of theoretical and empirical applications, including underlying the following definition. In this paper, we will only encounter real-valued matrices, for which the first property is equivalent to that X is symmetric with respect to its main diagonal.

Definition 3.2 (Semidefinite program)**.** A semidefinite program is an optimization problem of the following type

$$\text{maximize } \sum_{i=1}^{n}\sum_{j=1}^{n} A_{i,j}X_{i,j} : X \succeq 0, \sum_{i=1}^{n}\sum_{j=1}^{n} B_{k,i,j}X_{i,j} = b_i \text{ for } k = 1, \ldots, m,$$

where A and B_k are given $n \times n$ matrices, $b_i \in \mathbb{R}$ are given.

One advantage of semidefinite program is that it is a subset of convex optimization problems (i.e., the constraints and the objective are convex functions in X), for which efficient solvers exist whereas solving integer programs is much computationally harder. Therefore, it is tempting to relax some restrictions in an integer program to make it a semidefinite program. After we have solved the semidefinite program, we can exert the restrictions again to obtain an approximate solution for the original integer program.

The following theorem can help ensure that semidefinite program approximations of solutions for integer programs are reasonably accurate, up to a constant factor.

Theorem 3.3 (Grothendieck's inequality)**.** Consider an $m \times n$ matrix A of real numbers. Assume that, for any numbers $x_i, y_j \in \{-1, 1\}$, we have

$$\left| \sum_{i=1}^{m}\sum_{j=1}^{n} A_{ij}x_i y_j \right| \leq 1.$$

Then, for any Hilbert space H and any vectors $u_i, v_j \in H$ satisfying $\|u_i\| = \|v_j\| = 1$, we have

$$\left| \sum_{i=1}^{m}\sum_{j=1}^{n} A_{ij}\langle u_i, v_j \rangle \right| \leq K,$$

where $K \leq 1.783$ is a constant.

Various proofs of this theorem results in various bounds on K, with 1.783 being the smallest explicit bound that is known [3]. The idea is to use kernel method to work with a higher-dimensional space that has more linearities. However, it is also known that the best possible bound must be strictly less than 1.783 [1]. Here, we proceed assuming this theorem without a rigorous proof as giving a bound on K.

3.2 0.878-Approximation

Now, we are ready to present the 0.878-approximation algorithm except that we have not yet proven the following useful lemma.

Lemma 3.4 (Grothendieck's identity)**.** Let g be a uniform random unit vector. Then, for any fixed vectors u, v such that $\|u\| = \|v\| = 1$, we have

$$\mathrm{E}(\mathrm{sign}\langle g, u\rangle \cdot \mathrm{sign}\langle g, v\rangle) = \frac{2}{\pi} \arcsin\langle u, v\rangle \tag{7}$$

Proof. Let $\alpha \in [0, \pi]$ be the angle between the vectors u and v. We claim that $\mathrm{P}(\mathrm{sign}\langle g, u\rangle \cdot \mathrm{sign}\langle g, v\rangle = -1) = \frac{\alpha}{\pi}$.

Recall the inner product is rotation invariant. So we can always rotate the vectors g, u, v such that $u, v \in \mathbb{R}^2$. Moreover, we can also assume $g \in \mathbb{R}^2$ without loss of generality as the the other dimensions do not contribute to the inner product.

Again, by the rotation invariance of inner product, we can let $u = (1, 0)$ and $v = (\cos\theta, \sin\theta)$. Once more, without loss of generality, we can assume that $0 \le \theta \le \pi$ as we can rename u, v if necessary. If we let $g = (\cos\gamma, \sin\gamma)$ for some $0 \le \gamma \le \pi$, it is easy to see that

$$\mathrm{sign}\langle g, u\rangle \cdot \mathrm{sign}\langle g, v\rangle = -1 \iff \frac{2}{\pi} < \gamma < \theta + \frac{2}{\pi}.$$

This is because $\mathrm{sign}\langle g, u\rangle \cdot \mathrm{sign}\langle g, v\rangle = -1$ if and only if a line through the origin whose normal vector is g divides u and v into different half planes. And it happens if and only if $\frac{\pi}{2} < \gamma + \theta + \frac{2}{\pi}$. Thus, we have proven that

$$\mathrm{P}(\mathrm{sign}\langle g, u\rangle \cdot \mathrm{sign}\langle g, v\rangle = -1) = \frac{\alpha}{\pi}.$$

Then, since u, v are unit vectors, $\alpha = \arccos\langle u, v\rangle$. Also, since g takes continuous values, $\mathrm{P}(\mathrm{sign}\langle g, u\rangle \cdot \mathrm{sign}\langle g, v\rangle = 0) = 0$. Hence,

$$\begin{aligned}
\mathrm{E}(\mathrm{sign}\langle g, u\rangle \cdot \mathrm{sign}\langle g, v\rangle) &= (-1) \times \frac{\alpha}{\pi} + 1 \times \left(1 - \frac{\alpha}{\pi}\right) \\
&= 1 - \frac{2\alpha}{\pi} \\
&= 1 - \frac{2}{\pi} \arccos\langle u, v\rangle \\
&= \frac{2}{\pi} \left(\frac{\pi}{2} - \arccos\langle u, v\rangle\right) \\
&= \frac{2}{\pi} \arcsin\langle u, v\rangle.
\end{aligned}$$

QED

We want to provide a visualization Lemma 3.4. The idea is simply to fix arbitrary u and v, sample g, and observe the distribution of $\mathrm{E}(\mathrm{sign}\langle g, u\rangle \cdot \mathrm{sign}\langle g, v\rangle)$.

However, there is one caveat - uniformly sampling a unit vector is more challenging to implement than it seems. Naïve sampling methods usually results in biased distributions.

Fortunately, as [4] suggested, we can take advantage of the properties of the normal distribution.

Theorem 3.5. For some positive integer n, let $X_1, X_2, \ldots, X_n$ be n Gaussian random variables with mean 0 and variance 1, then the random vector

$$\frac{(X_1, X_2, \ldots, X_n)}{\|(X_1, X_2, \ldots, X_n)\|_2} = \frac{1}{\sqrt{\sum_{i=1}^{n} X_i^2}}(X_1, X_2, \ldots, X_n)$$

is uniformly distributed over all unit vectors.

Proof. Let

$$X = (X_1, X_2, \ldots, X_n) \sim N(0, I_n),$$

where $X_1, X_2, \ldots, X_n$ are as in the theorem. And let Q be an $n \times n$ orthogonal matrix. Then, notice that $QX \sim N(0, I_n)$ so X is rotational invariant.

Moreover, the random variable $\frac{X}{\|X\|_2}$ is also rotation invariant as

$$\frac{QX}{\|QX\|_2} = \frac{QX}{\|X\|_2} = \frac{X}{\|X\|_2}.$$

Then, $\frac{X}{\|X\|_2}$ is also rotation invariant. Furthermore, we also have that

$$\left\| \frac{X}{\|X\|_2} \right\|_2 = 1$$

with probability 1. Hence, X is distributed uniformly on the unit sphere. QED

An illustration of this theorem is provided in Figure 4. In this illustration, we specifically choose to work in a 2-dimensional space for the sake of simplicity of the visualization and we use method described in Theorem 3.5 to generate 100 uniform random vectors. From the figure, we can observe that the points are reasonably uniform around the entire unit circle.

With this handy theorem at our disposal, we can perform our experiments in a straightforward manner. In our experiment, we arbitrarily choose 50 pairs of 200-dimensional unit vectors u and v and for each pair we uniformly sample 1000 random vectors g. Then, we calculate the average of $\text{sign}\langle g, u\rangle \cdot \text{sign}\langle g, v\rangle)$ for each pair of u, v and plot $\text{sign}\langle g, u\rangle \cdot \text{sign}\langle g, v\rangle) - \frac{2}{\pi} \arcsin\langle u, v\rangle$. The results are outlined in Figure 5. From the figure, we can see that the values are very small and clustered around 0.

Now, we are ready to use a semidefinite program and prior results to obtain the famous 0.878-approximation algorithm for maximum cut due to Goemans and Williamson [2].

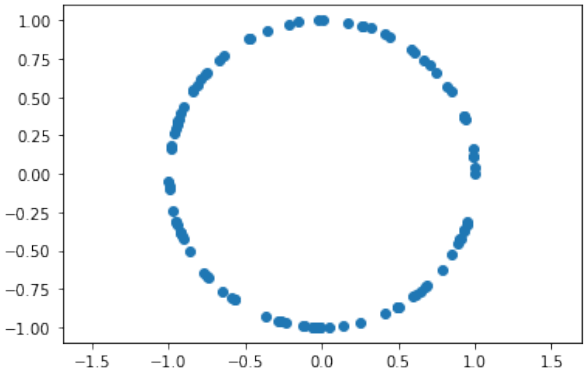

Figure 4: 100 uniform random vectors illustrating Theorem 3.5 in 2-dimension.

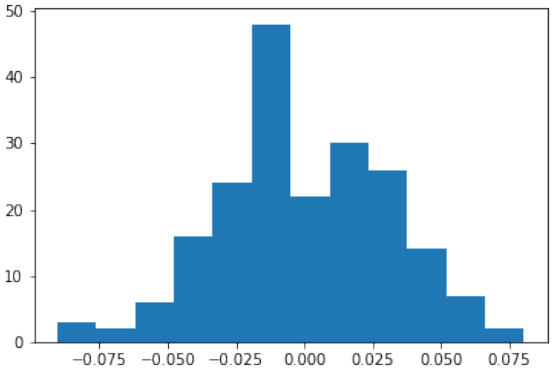

Figure 5: Illustration of Lemma 3.4. The histogram of $(\text{sign}\langle g, u\rangle \cdot \text{sign}\langle g, v\rangle) - \frac{2}{\pi}\arcsin\langle u, v\rangle$ for different g, u, v is plotted. We can see from the plot that the expectation is approximately $\frac{2}{\pi}\arcsin\langle u, v\rangle$.

Theorem 3.6 (0.878-approximation algorithm)**.** Let G be a graph with adjacency matrix A. Define the following semidefinite program

$$\mathrm{SDP}(G) = \frac{1}{4} \max \left\{ \sum_{i=1}^{n} \sum_{j=1}^{n} A_{i,j} \left(1 - \langle X_i, X_j \rangle\right) : \|X_i\|_2 = 1 \, \forall 1 \leq i \leq n \right\} \quad (8)$$

where $\langle \cdot, \cdot \rangle$ denotes the canonical inner product of two vectors and

$$\|X_i\|_2 = \sqrt{\sum_{j=1}^{n} X_{i,j}^2}.$$

Then, $\mathrm{SDP}(G)$ is a semidefinite program. Moreover, if we let X be its solution and define

$$x_i = \mathrm{sign}\left(\langle X_i, g \rangle\right)$$

where $g \sim N(0, I_n)$ is a standard normal random vector, then

$$\mathrm{E}\,\mathrm{Cut}(G, x) \geq 0.878\,\mathrm{SDP}(G) \geq 0.878\,\text{MAX-CUT}(G).$$

Proof. First, we verify that $\mathrm{SDP}(G)$ is indeed a semidefinite program. Even though $\mathrm{SDP}(G)$ is not in the original format of a semidefinite program, we can transform it into one as follows.

Define Y to be a $n \times n$ matrix such that

$$Y_{i,j} = \langle X_i, X_j \rangle.$$

Since inner product is symmetric, Y must be symmetric as well. And in this way, our objective functions becomes

$$\sum_{i=1}^{n} \sum_{j=1}^{n} A_{i,j} \left(1 - \langle X_i, X_j \rangle\right) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{i,j} \left(1 - Y_{i,j}\right),$$

which agrees with the original formulation of semidefinite programs.

Moreover, the restriction on X such that $\|X\|_2 = 1$ means that Y is uniformly 1 on the main diagonal. In other words,

$$\|X\|_2 = 1 \iff Y_{i,i} = 1 \, \forall i = 1, 2, \ldots, n.$$

Also, by the definition of Y, we notice that $Y = X^\mathsf{T} X$ so Y is a positive semidefinite matrix. Thus, $\mathrm{SDP}(G)$ is a semidefinite program as it contains all components and satisfies all conditions of a semidefinite program.

If X is a solution for $\mathrm{SDP}(G)$, we can simply obtain a solution Y for our transformed semidefinite program by taking $Y = X^\mathsf{T} X$. Conversely, although we cannot compute an explicit formula for arbitrary X in an efficient manner given Y, we still have approximation algorithms for the decomposition.

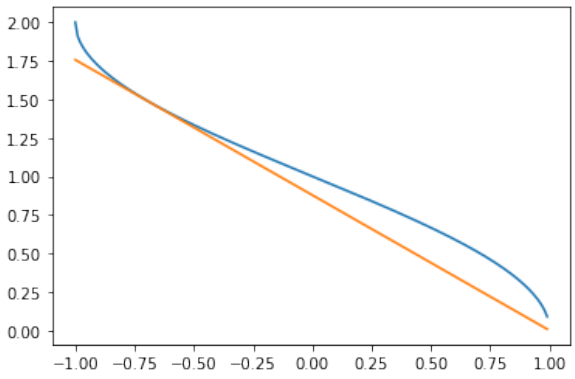

Figure 6: Illustration of Equation 9. The orange straight line represents $y = 0.878(1-x)$ and the blue curve represents $y = \frac{2}{\pi}\arccos x$.

Then, we remove the nonlinear arcsin function in Lemma 3.4 using that

$$\frac{2}{\pi}\arccos\langle u, v\rangle \geq 0.878(1 - \langle u, v\rangle),$$

which can be verified using a simple script. This inequality, together with the trigonometry identity $\arcsin t + \arccos t = \frac{\pi}{2}$ implies that

$$1 - \frac{2}{\pi}\arcsin\langle u, v\rangle = \frac{2}{\pi}\arccos t \geq 0.878(1-t). \tag{9}$$

Then, we can rewrite Equation 5 as

$$\begin{aligned}\mathrm{E}\,\mathrm{Cut}(G, x) &= \frac{1}{4}\mathrm{E}\left[\sum_{i=1}^{n}\sum_{j=1}^{n} M_{ij}\left(1 - x_i x_j\right)\right] \\ &= \frac{1}{4}\sum_{i=1}^{n}\sum_{j=1}^{n} M_{ij}\left(1 - \mathrm{E}\left[x_i x_j\right]\right).\end{aligned}$$

However,

$$\begin{aligned}1 - \mathrm{E}\left[x_i x_j\right] &= \mathrm{E}[\mathrm{sign}\langle g, X_i\rangle \cdot \mathrm{sign}\langle g, X_j\rangle] && \text{by the construction of } x \\ &= 1 - \frac{2}{\pi}\arcsin\langle X_i, X_j\rangle && \text{by Lemma 3.4} \\ &\geq 0.878(1 - \langle X_i, X_j\rangle) && \text{by Equation 9.}\end{aligned}$$

Hence,

$$\begin{aligned}\mathrm{E}\,\mathrm{Cut}(G, x) &\geq 0.878 \times \frac{1}{4}\sum_{i=1}^{n}\sum_{j=1}^{n} M_{ij}\left(1 - \langle X_i, X_j\rangle\right) \\ &= 0.878\,\mathrm{SDP}(G),\end{aligned}$$

which proves the first inequality. So, it only remains to prove the second inequality.

To see the second inequality, let $p^* \in \{-1, 1\}^n$ denote the partition generating the maximum cut. If we let $X_i = (p_i^*, 0, 0, \ldots, 0)$ for any $i = 1, 2, \ldots, n$. Then,

$$\begin{aligned}
\text{SDP}(G) &\geq \sum_{i=1}^{n} \sum_{j=1}^{n} A_{i,j} \left(1 - \langle X_i, X_j \rangle\right) \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} A_{i,j} \left(1 - p_i^* p_j^*\right) \\
&= \text{Cut}(G, p^*) \\
&= \text{MAX-CUT}(G).
\end{aligned}$$

QED

Besides its superior accuracy over the 0.5-approximation algorithm, Theorem 3.6 also allows us to easily recover a partition that attains at least 0.878 times the maximum cut given the solution of $\text{SDP}(G)$ using the randomized rounding technique in Theorem 3.6.

References

[1] Mark Braverman et al. "The Grothendieck constant is strictly smaller than Krivine's bound". In: *Forum of Mathematics, Pi*. Vol. 1. Cambridge University Press. 2013.

[2] Michel X Goemans and David P Williamson. "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming". In: *Journal of the ACM (JACM)* 42.6 (1995), pp. 1115–1145.

[3] JL Krivine. "Constantes de Grothendieck et fonctions de type positif sur les sphères". In: *Advances in Mathematics* 31.1 (1979), pp. 16–30.

[4] Mervin E Muller. "A note on a method for generating points uniformly on n-dimensional spheres". In: *Communications of the ACM* 2.4 (1959), pp. 19–20.

[5] Roman Vershynin. *High-dimensional probability: An introduction with applications in data science*. Vol. 47. Cambridge university press, 2018.

Appendices

A Code

A.1 Expectation of Uniform Random Cut

```
# ----- Graph ------
import numpy as np

class Graph:
    """
    A simple implementation of graph with the help of numpy.
    """

    def __init__(self, n):
        """Initializes an empty graph"""
        self.n = n
        self.V = np.arange(n)
        self.E = np.zeros((n, n), dtype='int8')

    def generate_random_edges(self):
        """As name suggests, but we want E to be reflective w.r.t its main diagonal"""
        self.E = (np.random.random((self.n, self.n)) >= 0.5).astype('int8')
        self.E = np.triu(self.E) # upper triangle of E
        self.E = (self.E + self.E.T) - np.diag(np.diag(self.E))

    def cut(self, C):
        """C: an 1D numpy array consisting of 0's and 1's of length self.n"""
        partition_0 = self.V[C[self.V] == 0]
        partition_1 = self.V[C[self.V] == 1]
        return np.sum(np.array([self.E[u, v] for u in partition_0 for v in partition_1]))

    def num_edges(self):
        # we subtract the orthogonal so that we have a simple graph
        # this works because the loops are never calculated in any cut
        return (np.sum(self.E) - np.sum(np.diag(self.E))) / 2
# ----- End of Graph ------

# ----- Experiments ------
# Hyper-parameters
# the number of graphs
NUM_GRAPHS = 100
# number of cuts for each graph
NUM_CUTS = 1000
# the number of vertices in each graph
N = 20

diffs = []
for _ in range(NUM_GRAPHS):
    graph = Graph(N)
    graph.generate_random_edges()
```

```
    cuts = []
    for _ in range(NUM_CUTS):
        # generate a random cut
        cut = (np.random.random((N)) >= 0.5).astype('int8')
        cuts.append(graph.cut(cut))
    diffs.append(np.mean(np.array(cuts)) - graph.num_edges() / 2)
# ----- End of Experiments -----

# ----- Illustration -----
import matplotlib.pyplot as plt

_, _, _ = plt.hist(x=diffs, bins='auto')
plt.show()
# ----- End of Illustration -----
```

A.2 Uniform Random 2D Unit Vector

```
import numpy as np
from matplotlib import pyplot as plt

xs = []
ys = []
for _ in range(100):
    g = np.random.default_rng().normal(size=2)
    g /= np.linalg.norm(g)
    xs.append(g[0])
    ys.append(g[1])

plt.scatter(xs, ys)
plt.axis('equal')
plt.show()
```

A.3 Visualization of Grothendieck's Identity

```
import numpy as np
from matplotlib import pyplot as plt
from math import pi

# hyperparameters
NUM_DIM = 50
NUM_UV = 200
NUM_G = 1000

diffs = []
for _ in range(NUM_UV):
    u = np.random.default_rng().normal(size=NUM_DIM)
    u /= np.linalg.norm(u)
    v = np.random.default_rng().normal(size=NUM_DIM)
    v /= np.linalg.norm(v)
    sign_list = []
    for _ in range(NUM_G):
        g = np.random.default_rng().normal(size=NUM_DIM)
        g /= np.linalg.norm(g)
        sign_list.append(np.sign(np.dot(g, u) * np.sign(np.dot(g, v))))
```

```
21      diffs.append(np.array(sign_list).mean() - 2/pi * np.arcsin(np.dot(u
        , v)))
22
23  _ = plt.hist(x=diffs, bins='auto')
24  plt.show()
```