Math 170 Term Paper: An Introduction to Semidefinite Programming

Michael Zheng, ID: 25550648

May 2017

1 Notation

1.1 Symmetric Positive Semidefinite Matrices

The name semidefinite programming refers to the fact that we are optimizing some cost function across the set of semidefinite symmetric matrices. An n x n matrix, X, is called **positive semidefinite** if the following property holds

$$v^T X v \geq 0, \quad \forall v \in \mathbb{R}^n.$$

Similarly, An n x n matrix, X, is called **positive definite** if the following property holds

$$v^T X v > 0, \quad \forall v \in \mathbb{R}^n.$$

Some notation for these matrices

- S^n is the set of symmetric n x n matrices
- S^n_+ is the set of positive semidefinite symmetric n x n matrices
- S^n_{++} is the set of positive definite symmetric n x n matrices
- $X \succeq 0$ denotes that matrix X is symmetric and positive semidefinite
- $X \succ 0$ denotes that matrix X is symmetric and positive definite
- $X \succeq Y$ denotes that $X - Y \succeq 0$

Symmetric matrices have a special matrix factorization. If $X \in S^n$, then $X = PDP^T$, where P is an orthonormal matrix ($P^T = P^{-1}$), and D is a diagonal matrix. Facts about symmetric matrices and positive definite and semidefinite matrices

- The columns of P are the eigenvectors of X and the diagonal entries in D are the corresponding eigenvalues of X
- $X \succeq 0$ if and only if X is symmetric and all of its eigenvalues are nonnegative
- $X \succ 0$ if and only if X is symmetric and all of its eigenvalues are positive

2 Formulation

There are two common formulations of semidefinite programs. Let $A_1, A_2, ..., A_m, C \in S^n$ and $b_1, ..., b_m \in \mathbb{R}$. The first common formulation is as follows.

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & C \bullet X \\ \text{subject to} \quad & A_i \bullet X = b_i, \quad i = 1, \ldots, m, \\ & X \succeq 0 \end{aligned} \tag{1}$$

We can think of $C \bullet X$ as the dot product of two matrices. It is equivalent to flattening out the matrices into vector form and taking the dot product.

$$C \bullet X = \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} X_{ij}$$

2.1 Linear Program as a SDP

It turns out that linear programs are a special case of semidefinite programs. Consider A_i, C, and X as the following diagonal matrices

$$A_i = \begin{bmatrix} a_{i1} & & \\ & \ddots & \\ & & a_{in} \end{bmatrix}, \quad C = \begin{bmatrix} c_1 & & \\ & \ddots & \\ & & c_n \end{bmatrix}, \quad X = \begin{bmatrix} x_1 & & \\ & \ddots & \\ & & x_n \end{bmatrix}$$

We see that $C \bullet X$ is equivalent to $c^T x$ where c, $x \in \mathbb{R}^n$ are vectors with values of the diagonal of the matrices. The SDP given by (1) is the equivalent linear program

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & a_i^T x = b_i, \quad i = 1, \ldots, m, \\ & x \geq 0 \end{aligned} \tag{2}$$

3 Schur Complement

For a block matrix M defined as

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

where A, B, C, D are pxp, pxq, qxp, qxq matrices and D is invertible, the Schur Complement of the block D matrix of M is the pxp matrix

$$M/D = A - BD^{-1}C$$

If M is symmetric then the Schur Complement of M has special properties that can allow us to model certain constraints as positive semidefinite constraints on matrices.

Theorem 1. *For a symmetric matrix M defined as*

$$M = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}$$

If C is invertible then the following holds

1. *$M \succ 0$ iff $C \succ 0$ and $A - BC^{-1}B^T \succ 0$*
2. *If $C \succ 0$, then $M \succeq 0$ iff $A - BC^{-1}B^T \succeq 0$*

Proof. (1) Given a symmetric matrix A and an invertible matrix U, $A \succ 0$ iff $UAU^T \succ 0$. If we let the matrix U be defined as

$$U = \begin{bmatrix} I & BC^{-1} \\ 0 & I \end{bmatrix}$$

U here is an invertible matrix. We can decompose M as

$$M = \begin{bmatrix} I & BC^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} A - BC^{-1}B^T & 0 \\ 0 & C \end{bmatrix} \begin{bmatrix} I & BC^{-1} \\ 0 & I \end{bmatrix}^T$$

If $M \succ 0$ then the block matrix must also be positive definite. Then the matrices on the diagonal of the block matrix must also be positive definite. If the block matrix is positive definite, $C \succ 0$ and $A - BC^{-1}B^T \succ 0$, and therefore $M \succeq 0$

(2) Similar to (1), Given a symmetric matrix A and an invertible matrix U, $A \succeq 0$ iff $UAU^T \succeq 0$. The proof then follows from the same reasoning as (1). $\square$

4 Duality

The dual problem for the given semidefinite program is defined in a very similar fashion as the dual problem of a linear program.

$$\begin{aligned} \underset{y}{\text{maximize}} \quad & b^T y \\ \text{subject to} \quad & \sum_{i=1}^{m} y_i A_i + S = C, \quad i = 1, \ldots, m, \\ & S \succeq 0 \end{aligned} \tag{3}$$

Continuing with to paint the similarity between SDP and LP, if A_i and C are diagonal matrices. In order for this to make sense, we see that S is a 1x1 matrix, or just a real value. This means that $S \succeq 0$ implies that $S \geq 0$. Thus the dual for (2) is the same dual problem for a standard LP.

4.1 Weak Duality

Weak duality holds for SDP. Before we prove this fact, let's introduce the following fact about the dot product of symmetric positive semidefinite matrices.

Lemma 1. *$X, Y \in S_+^n$, $X \bullet Y \geq 0$*

Proof. First we prove that for $X, Y \in S_+^n$, $X \bullet Y \geq 0$. Since X and Y are symmetric matrices, by the spectral theorem they are diagonalizable. Thus $X = UDU^T = \sum_{i=1}^n d_i u_i u_i^T$ and Y where U are unitary matrices and D is a diagonal matrix. We see that $X \bullet Y = \left(\sum_{i=1}^n d_i u_i u_i^T\right) \bullet Y$. Since we know that $d_i > 0$ (property of positive semidefinite matrix), we require that $u_i u_i^T \bullet Y \geq 0$.

$$(u_i u_i^T) \bullet Y = Trace((u_i u_i^T)^T Y) = Trace(u_i^T Y u_i) = u_i^T Y u_i \geq 0$$

$\square$

Theorem 2. *Given a feasible solution X and y to the primal and dual respectively $b^T y \leq C \bullet X$.*

Proof. We have that $S = C - \sum_{i=0}^m y_i A_i \succeq 0$. By Lemma 1 we have that for any $X \in S_+^n$, $S \bullet X = (C - \sum_{i=0}^m y_i A_i) \bullet X = C \bullet X - \sum_{i=0}^m (A_i \bullet X) y_i = C \bullet X - \sum_{i=0}^m b_i y_i = C \bullet X - b^T y \geq 0$. This implies that $C \bullet X \geq b^T y$ for feasible solutions to the primal and dual. $\square$

5 Interior Point Method for SDP

5.1 Barrier Function

We have studied Interior Point Methods for solving linear programs, but they can also be used to solve nonlinear optimization problems. First we establish a barrier function for SDP. The barrier function is meant to penalize heavier the closer you move to the boundary of the feasible region. The feasible region for semidefinite programs is the positive semidefinite cone. One way we can characterize the interior of this cone is through the eigenvalues. We know that positive definite matrices have nonnegative eigenvalues. We define the interior of this cone $\text{Int}(X) \in S_+^n$ as

$$\text{Int}(X) = \{X | \lambda_1(X) > 0, ..., \lambda_n(X) > 0\}$$

where $\lambda_i(X)$ is the i-th eigenvalue of X.

With this we can use a similar barrier that we used for linear programs.

$$-\sum_{i=1}^{n} \ln(\lambda_i(X)) = -\ln\left(\prod_{i=1}^{n} \lambda_i(X)\right) = -\ln(\det(X))$$

We can rewrite the standard SDP problem (1) as a new problem that includes the barrier function.

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & C \bullet X - \theta \ln(\det(X)) \\ \text{subject to} \quad & A_i \bullet X = b_i, \quad i = 1, ..., m, \\ & X \succ 0 \end{aligned} \tag{4}$$

We define the function we are trying to minimize as $f_\theta(X)$. The next step is to solve for the minimum of this function. We know that the optimal point $x*$ to $f_\theta(X)$ has to satisfy the Karush-Kuhn-Tucker (KKT) Conditions. We use this along with Newton's method to get the updates.

$$\nabla f_\theta(X) = C \bullet \nabla X - \theta \nabla \ln(\det(X)) = C - \theta \frac{1}{\det(X)} \det(X)(X^{-1})^T = C - \theta X^{-1}$$

The KKT conditions are:

$$\begin{cases} A_i \bullet X = b_i, \quad i = 1, ..., m, \\ X \succ 0, \\ C - \theta X^{-1} = \sum_{i=1}^{m} y_i A_i \end{cases}$$

By letting $S = \theta X^{-1}$, we see that $XS = \theta I$. Thus we can rewrite the conditions as:

$$\begin{cases} A_i \bullet X = b_i, \quad i = 1, ..., m, \\ X \succ 0, \\ \sum_{i=1}^{m} y_i A_i + S = C \\ XS = \theta I \end{cases}$$

$S \bullet X = n\theta$ gives us our duality gap. We can solve the SDP as $\theta \to 0$ using Newton's method.

6 Applications of SDP

There is a wide variety of problems that can be cast as a semidefinite program. This section will go over a couple examples with also some code that demonstrates how to solve semidefinite programming problems using CVXPY a python library used to solve convex optimization problems.

6.1 Largest Eigenvalue

Given $X \in S_+^n$, we can cast the problem of finding the largest eigenvalue of this matrix into a SDP problem.

$$\begin{aligned} &\text{minimize} \quad t \\ &\text{subject to} \quad tI - X \succeq 0 \end{aligned} \tag{5}$$

The derivation of this problems stems from the fact that a PSD matrix, X, has nonnegative eigenvalues. If the eigenvalues of X are $\lambda_1, \ldots, \lambda_n$, the eigenvalues of $tI - X$ are $t - \lambda_1, \ldots, t - \lambda_n$. Requiring that $tI - X \succeq 0$ is equivalent to requiring $t - \lambda_1 \geq 0, \ldots, t - \lambda_n \geq 0$. This implies that $t \geq \lambda_i$ for $i = 1, \ldots, n$. Therefore, t is an upper bound for the eigenvalues of X. Finally, we use the trick of minimizing the upper bound of the eigenvalues in order to find the largest eigenvalue of X.

Listing 1: Solving Max Eigenvalue with CVXPY in Python

```python
from cvxpy import *
import numpy as np

t = Variable()

A_1 = np.random.rand(2,2)
A = (A_1 + A_1.transpose())/2

objective = Minimize(t)
constraints = [t*np.identity(2) - A >> 0]
prob = Problem(objective, constraints)
prob.solve()
print "status:", prob.status
print "optimal value", prob.value
print "A matrix:", A
print np.linalg.eig(A)
```

```
status: optimal
optimal value 1.13720128304
A matrix: [[ 0.06016364 0.80742434] [ 0.80742434 0.53277347]]
Eigenvalue Decomp: (array([-0.54482454, 1.13776166]), array([[-0.80027588, -0.59963198], [ 0.59963198, -0.80027588]]))
```

6.2 Minimum Volume Ellipsoid Covering A Finite Set

Given a collection of points $x_1, \dots, x_n \in \mathbb{R}^2$, we can cast the problem of finding the minimum volume ellipsoid covering this set of points into a semidefinite programming problem. An ellipsoid in $\mathbb{R}^2$ is defined to be solutions $(x, y) \in \mathbb{R}^2$ to the equation

$$\frac{x^2}{c^2} + \frac{y^2}{d^2} = 1$$

where $c > 0$, $d > 0$. The volume bounded by an ellipsoid is given by $V = \frac{4}{3}\pi cd$. Ellipsoids can also be defined using quadratic forms. An ellipsoid centered at b can be defined by solutions to the quadratic form where $A \in S^2_{++}$.

$$E = \{x | (x-b)^T A(x-b) \leq 1\}$$

It can be shown that the eigenvalues of A are $\frac{1}{c^2}$ and $\frac{1}{d^2}$. Therefore the volume of this ellipsoid is directly proportional to $\det(A)^{-1}$. Minimize the natural log of this function is equivalent to maximizing $\ln(\det(A))$, which is how we reach the objective function. This gives us the optimization problem

$$\begin{array}{ll} \underset{A,\, b \in \mathbb{R}^2}{\text{maximize}} & \ln\det(A) \\ \text{subject to} & (x_i - b)^T A(x_i - b) \leq 1, \quad i = 1, \dots, n, \\ & \qquad\qquad A \succ 0 \end{array} \tag{6}$$

Since A is positive definite, we can find its Cholesky Decomposition and write $A = U^T U$, where U is an upper triangular matrix with all positive entries on the diagonal, which implies that $U \succ 0$. Now we can rewrite problem (6) using U.

$$\begin{array}{ll} \underset{U,\, b \in \mathbb{R}^2}{\text{maximize}} & \ln\det(U^T U) \\ \text{subject to} & (x_i - b)^T U^T U(x_i - b) \leq 1, \quad i = 1, \dots, n, \\ & \qquad\qquad U \succ 0 \end{array} \tag{7}$$

Using Schur Complement we can rewrite the constraint $(x_i - b)^T U^T U(x_i - b) \leq 1$ for $i = 1, \dots, n$ using positive semidefinite matrices.

$$\begin{bmatrix} I & U(x_i - b) \\ (U(x_i - b))^T & 1 \end{bmatrix} \succeq 0$$

With all of this, we can write this optimization problem as a SDP with a nonlinear objective function.

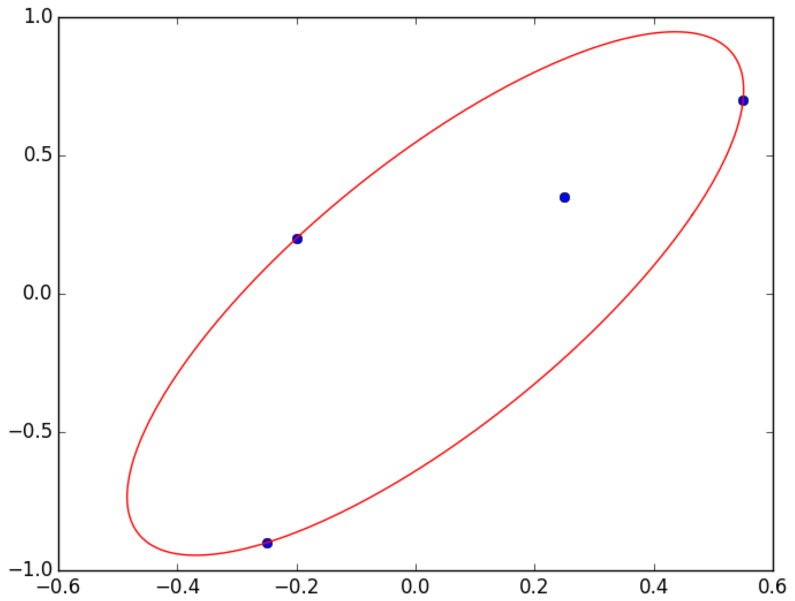
$$\begin{array}{ll} \underset{U,\, b \in \mathbb{R}^2}{\text{maximize}} & 2\ln\det(U) \\ \text{subject to} & \begin{bmatrix} I & U(x_i - b) \\ (U(x_i - b))^T & 1 \end{bmatrix} \succ 0 \quad i = 1, \dots, n \end{array} \tag{8}$$

Listing 2: Solving Minimum Volume Ellipsoid Covering A Finite Set with CVXPY in Python

```
from cvxpy import *
import numpy as np
import matplotlib.pyplot as plt
import math

A = Semidef(2)
b = Variable(2)
x_1 = np.array([0.55, 0.7])
x_2 = np.array([0.25, 0.35])
x_3 = np.array([-0.25, -0.9])
x_4 = np.array([-0.2, 0.2])
X = np.vstack([x_1, x_2, x_3, x_4])
objective = Maximize(log_det(A))
constraints = [norm(A*x_1 + b, 2) <= 1, norm(A * x_2 + b, 2) <= 1,
 norm(A * x_3 + b, 2) <= 1, norm(A * x_4 + b, 2) <= 1]
prob = Problem(objective, constraints)
prob.solve()
print "status:", prob.status
print "optimal value", prob.value
print "A", A.value
print "b", b.value
numangles = 200
angles = np.linspace(0, 2*np.pi, numangles)
t = np.array([np.cos(angles) - b.value[0], np.sin(angles)-b.value[1]]).reshape(2,200)
ellipse = np.linalg.solve(A.value, t)
plt.plot(X[:,0], X[:,1], 'o')
plt.ylim(-1, 1)
plt.plot(ellipse[0,:].transpose(), ellipse[1,:].transpose(), 'r')
plt.show()
```

Figure 1: Code Output



7 MAXCUT Relaxation with SDP

One of the applications of Semidefinite Programming to combinatorial optimization is the relaxation of the max cut problem. This section will present the MAXCUT problem and various techniques to solve for the relaxation of this problem.

7.1 MAXCUT Problem

A cut in an undirected weighted graph $G = (V, E)$ is a partition of the vertex set V into two sets V_1 and V_2. The problem of finding the maximum cut is finding the cut that maximizes the sum of the edge weights connecting the sets V_1 and V_2. In this problem we assume all the edge weights are positive.

7.2 Formulation

We use w_{ij} the edge weight of the edge connecting vertex i and vertex j, and use e_{ij} as the edge connecting vertex i and vertex j. We create a decision variable x_i for every vertex in $G = (V, E)$, where x_i is defined as

$$x_i = \begin{cases} 1 & \text{if } x_i \in V_1 \\ -1 & \text{if } x_i \in V_2 \end{cases}$$

We formulate our optimization problem as follows

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2} \sum_{(i,j)\in E} w_{ij}(1 - x_i x_j) \\ \text{subject to} \quad & x_i \in \{1, -1\}, \quad i = 1, \ldots, n \end{aligned} \tag{9}$$

The constraint clearly gives us a cut of the graph G. The objective function comes from the fact that $(1 - x_i x_j)$ is 0 if vertex i and j are part of the same partition set, so the weight of the edge connecting the two should not be included in the cut weight, and $(1 - x_i x_j)$ is 2 if those two vertices are in the same partition set. We only need to examine the edges of $i < j$ because an edge from i to j is also an edge from j to i. We can also modify the constraint to look over all pairs i and j.

$$\begin{aligned} \text{maximize} \quad & \frac{1}{4} \sum_{(i,j):i,j\in V} w_{ij}(1 - x_i x_j) \\ \text{subject to} \quad & x_i \in \{1, -1\}, \quad i = 1, \ldots, n \end{aligned} \tag{10}$$

We can actually modify this optimization problem and model it with the laplacian matrix of the graph. The laplacian matrix of a weighted graph $G = (V, E)$ is given by

$$L_{ij} = \begin{cases} -w_{ij} & \text{if } (i,j) \in E \\ w_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Where $w_i = \sum_{j:(i,j)\in E} w_{ij}$. Thus we see that the quadratic form of the laplacian matrix gives us:

$$x^T L x = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j L_{ij} = \sum_{i=1}^{n} x_i^2 w_i - \sum_{(i,j)\in E} x_i x_j w_{ij} = 2\sum_{(i,j)\in E} w_{ij} - \sum_{(i,j)\in E} x_i x_j w_{ij}$$

Notice that we can break up $\sum_{(i,j)\in E} x_i x_j w_{ij}$ into weights of edges between two vertices in the set V_1, weights of edges between two vertices in the set V_2, and the cut weights.

$$\sum_{(i,j)\in E} x_i x_j w_{ij} = \sum_{(i,j):i,j\in V_1} w_{ij} + \sum_{(i,j):i,j\in V_2} w_{ij} - \sum_{(i,j):i\in V_1, j\in V_2} w_{ij} = 2\sum_{(i,j)\in E} w_{ij} - 2\sum_{(i,j):i\in V_1, j\in V_2} w_{ij}$$

This gives us that:

$$x^T L x = 2\sum_{(i,j)\in E} w_{ij} - \sum_{(i,j)\in E} x_i x_j w_{ij} = 2\sum_{(i,j):i\in V_1, j\in V_2} w_{ij} = 4\left(\frac{1}{2}\sum_{(i,j)\in E} w_{ij}(1 - x_i x_j)\right)$$

This tells us that our objective function given in (9) can equivalently be expressed as $\frac{1}{4}x^T Lx$. The maximum cut is therefore also equal to

$$\begin{aligned} &\text{maximize} && \frac{1}{4}x^T Lx \\ &\text{subject to} && x \in \{-1, 1\}^n \end{aligned} \tag{11}$$

From here we get an upper bound for the value of the maximum cut, by relaxing the constraint on the vector $x \in \{-1, 1\}^n$ to $x \in \mathbb{R}^n$ with $\|x\|_2^2 = n$. This gives us the optimization problem:

$$\begin{aligned} &\underset{x \in \mathbb{R}^n}{\text{maximize}} && \frac{1}{4}x^T Lx \\ &\text{subject to} && \|x\|_2^2 = n \end{aligned} \tag{12}$$

This is solved by taking x to be the normalized eigenvector corresponding to the maximum eigenvalue of L, which returns an optimal value of $\frac{n}{4}\lambda_{max}(L)$, which will be an upper bound on the actual optimal maxcut value. A better bound can be achieved through semidefinite programming. We can rewrite the quadratic $x^T Lx$ as the Frobenius inner product of two matrices:

$$x^T Lx = Trace(x^T Lx) = Trace(Lxx^T) = L \bullet (xx^T)$$

Given $x \in \{-1, 1\}^n$, $X = xx^T$ has a couple special properties.

1. $X \succeq 0$.
2. X has all 1's on the diagonal.
3. $\text{rank}(X) = 1$

Proof.

1. For any vector $v \in \mathbb{R}^n$ we see that $v^T Xv = v^T xx^T v = (x^T v)^2 \geq 0$
2. The elements on the main diagonal are just x_i^2 which will always be 1.
3. X is a rank 1 matrix because for any $v \in \mathbb{R}^n$ we see that $Xv = xx^T v = (x^T v)x$, which is just a multiple of x. This means that X maps all elements to the 1-dimensional subspace spanned by x.

Thus we can write our optimization problem as

$$\begin{aligned} &\text{maximize} && \frac{1}{4}L \bullet X \\ &\text{subject to} && X \succeq 0, \\ & && X_{ii} = 1, \quad i = 1 \ldots n, \\ & && \text{rank}(X) = 1 \end{aligned} \tag{13}$$

Since we can not write the constraint $\text{rank}(X) = 1$ as a semidefinite program, we eliminate that constraint to get the SDP relaxation of maxcut.

$$\begin{aligned} &\text{maximize} && \frac{1}{4}L \bullet X \\ &\text{subject to} && X \succeq 0, \\ & && X_{ii} = 1, \quad i = 1 \ldots n \end{aligned} \tag{14}$$

This is a tighter constraint than the optimization problem in (12) because we have the additional constraint that all the diagonal entries are 1's. Therefore we know that the optimal cost to (14) $\leq \frac{n}{4}\lambda_{max}(L)$.

7.3 Goemans-Williamson Algorithm

Goemans-Williamson Algorithm is a rounding procedure that is used to find an approximate solution to the maxcut problem. The algorithm is as follows:

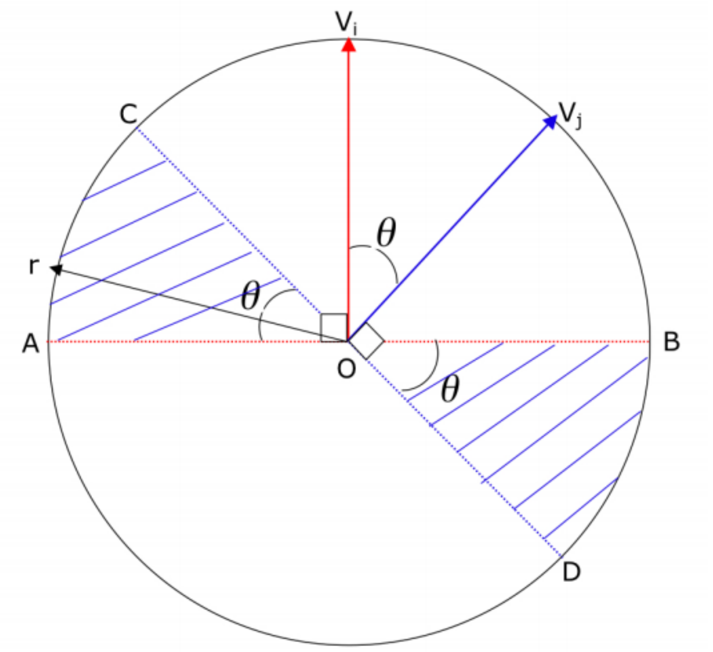
1. Solve the SDP relaxation given in (14). Label this solution as X^*.
2. Compute the Cholesky decomposition of $X^* = V^T V$ and let v_i be the normalized columns of U.
3. Set $V_1 = \emptyset$ and generate a random vector r in the unit n-sphere. If $v_i^T r > 0$ then we place vertex i in the set V_1. Otherwise i goes into the set V_2.
4. Calculate the weight of the new cut $\frac{1}{4} L \bullet X$.
5. Repeat steps 3 and 4 until you converge to some value which will be the approximate solution.

The intuition behind this algorithm is finding an better bound through find the expectation of the cut weight. The expected cut weight is given by

$$\mathbb{E}[\text{cut}] = \mathbb{E}\left[\sum_{(i,j)\in E} w_{ij} \mathbb{1}[(i,j) \text{ is cut})]\right] = \sum_{(i,j)\in E} w_{ij} \mathbf{Pr}((i,j) \text{ is cut}))$$

The probability that (i,j) represented by the vectors v_i and v_j will be in the cut is given by if $v_i^T r > 0$ and $v_j^T r < 0$ or if $v_i^T r < 0$ and $v_j^T r > 0$. The picture below depicts the plane spanned by v_i and v_j. The randomly chosen r on the n-sphere projects to a random vector on the circle in the plane formed by v_i and v_j. The blue regions correspond to the areas where r cuts the edge (i,j).

Figure 2: Regions of interest on the plane spanned by v_i and v_j



We see that the total area that is blue is given by 2θ where θ is the angle between v_i and v_j. We can find θ through the equation for the dot product.

$$v_i \cdot v_j = v_i^T v_j = \|v_i\|\|v_j\| \cos(\theta)$$

Since v_i and v_j are normalized, we know that their lengths are just 1. Therefore the angle θ is given by

$$\theta = \arccos(v_i^T v_j)$$

This gives us that the probability of cutting edge (i,j) through the approximation technique is

$$\mathbf{Pr}((i,j) \text{ is cut})) = \frac{2\theta}{2\pi} = \frac{1}{\pi} \arccos(v_i^T v_j)$$

Theorem 3. *This algorithm has a guaranteed performance of 0.878.*

Proof. Since the vectors v_i and v_j are normalized, the range of values of $v_i^T v_j$ are values between -1 and 1. Using mathematical software we get a bound for the following minimization problem.

$$\min_{x \in [-1,1]} \frac{\frac{1}{\pi}\arccos(x)}{\frac{1}{2}(1-x)} \geq 0.878$$

Using this we can get a value for the expected maxcut weight.

$$\mathbb{E}[\text{cut}] = \sum_{(i,j)\in E} w_{ij}\mathbf{Pr}((i,j) \text{ is cut})) = \sum_{(i,j)\in E} w_{ij}\frac{1}{\pi}\arccos(v_i^T v_j) \geq 0.878\frac{1}{2}\sum_{(i,j)\in E} w_{ij}(1 - v_i^T v_j)$$

$$= 0.878 OPT_{SDP} \geq 0.878 OPT_{maxcut}$$

Where OPT_{SDP} is the optimal value of the SDP relaxation of maxcut and OPT_{maxcut} is the optimal value to the maxcut problem. □

References

[1] Michel X Goemans. 18.415/6.854 advanced algorithms.

[2] Jean Gallier. The schur complement and symmetric positive semidefinite (and definite) matrices, Dec 2010.

[3] Robert M Freund. Introduction to semidefinite programming (sdp).

[4] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.