In this lecture we will study an extension of linear programming, namely semi-definite programming. Semi-definite programs are much more general than linear program and can encode certain kinds of quadratic programs (that is, those involving quadratic constraints or a quadratic objective). They cannot be solved optimally in polynomial time (because the optimal solution may be irrational), but can be approximated up to arbitrarily precision.

We will illustrate the use of semi-definite programming in algorithm design through the max-cut problem.

13.1 Semi-Definite Programming

Semi-definite programming, as a generalization of linear programming, enables us to specify in addition to a set of linear constraints a "semi-definite" constraint, a special form of nonlinear constraints. In this section, we introduce the basic concept of semi-definite programming.

13.1.1 Definitions

First Let us define a positive semi-definite (*p.s.d.*) matrix A, which we denote as $A \succeq 0$.

Definition 13.1.1 *A matrix $A \in \mathbb{R}^{n \times n}$ is positive semi-definite if and only if (1) A is symmetric, and (2) for all $x \in \mathbb{R}^n$, $x^T Ax = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} x_i x_j \geq 0$.*

Then we define a semi-definite program (SDP) as follows.

Definition 13.1.2 *An SDP is a mathematical program with four components:*

- *a set of variables x_{ij};*
- *a linear objective function to minimize/maximize;*
- *a set of linear constraints over x_{ij};*
- *a semi-definite constraint $X = [x_{ij}] \succeq 0$.*

The semi-definite constraint is what differentiates SDPs from LPs. Interestingly, the constraint can be interpreted as an infinite class of linear constraints. This is because by Definition 13.1.1, if $X \succeq 0$, then for all $v \in \mathbb{R}^n$, $v^T Xv \geq 0$. Each possible real vector v gives us one linear constraint. Altogether, we have an infinite number of linear constraints. We will use this interpretation later in this section.

13.1.2 A Different View of Semi-Definite Programming

In fact, semi-definite programming is equivalent to vector programming. To show this, we first present the following theorem.

Theorem 13.1.3 *For a symmetric matrix A, the following statements are equivalent:*

1. $A \succeq 0$.

2. *All eigenvalues of A are non-negative.*

3. $A = C^T C$, *where* $C \in \mathbb{R}^{m \times n}$.

Proof: We prove the theorem by showing statement 1 implies statement 2, statement 2 implies statement 3, and statement 3 implies statement 1.

$1 \Rightarrow 2$**:** If $A \succeq 0$, then all eigenvalues of A are non-negative.

Recall the concept of eigenvectors and eigenvalues. The set of eigenvectors ω for A is defined as those vectors that satisfy $A\omega = \lambda\omega$, for some scalar λ. The corresponding λ values are called eigenvalues. Moreover, when A is a symmetric matrix, all the eigenvalues are real-valued. Now let us look at each eigenvalue λ. First of all, we have $A\omega = \lambda\omega$, where ω is the corresponding eigenvector. Multiplying both sides by ω^T, then we have:

$$\omega^T A\omega = \lambda\omega^T\omega \tag{13.1.1}$$

The LHS of Equation 13.1.1 is non-negative by the definition of positive semi-definite matrix A. On the RHS of the equation, $\omega^T\omega \geq 0$. Thus $\lambda \geq 0$.

$2 \Rightarrow 3$**:** If all eigenvalues of A are non-negative, then $A = C^T C$ for some real matrix C.

Let Λ denote the diagonal matrix with all of A's eigenvalues:

$$\begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix}.$$

Let W denote the matrix of the corresponding eigenvectors $[\omega_1\ \omega_2\ \cdots\ \omega_n]$. Since eigenvectors are orthogonal to one another (*i.e.*, $\omega_i^T\omega_j = 1$ if $i = j$; 0, otherwise), W is an orthogonal matrix. Given Λ and W, we obtain the matrix representation of eigenvalues: $AW = W\Lambda$. Multiplying both side by W^T, we have:

$$AWW^T = W\Lambda W^T. \tag{13.1.2}$$

Since W is an orthogonal matrix, $WW^T = I$ and thus the LHS of Equation 13.1.2 is equal to A. Since all the eigenvalues are non-negative, we can decompose Λ into $\Lambda^{\frac{1}{2}}(\Lambda^{\frac{1}{2}})^T$, where $\Lambda^{\frac{1}{2}}$ is defined as:

$$\begin{bmatrix} (\lambda_1)^{\frac{1}{2}} & & 0 \\ & \ddots & \\ 0 & & (\lambda_n)^{\frac{1}{2}} \end{bmatrix}.$$

Thus $W\Lambda W^T = W\Lambda^{\frac{1}{2}}(\Lambda^{\frac{1}{2}})^T W^T = W\Lambda^{\frac{1}{2}}(W\Lambda^{\frac{1}{2}})^T$. Let $C^T = W\Lambda^{\frac{1}{2}}$. Equation 13.1.2 is equivalent to $A = C^T C$.

$3 \Rightarrow 1$**:** If $A = C^T C$ for some real matrix C, then $A \succeq 0$.

We prove it from the definition of A being *p.s.d.*:

$$\begin{aligned} A &= C^T C \\ \Leftrightarrow x^T A x &= x^T C^T C x, \quad \text{where } x \in \mathbb{R}^n \\ \Leftrightarrow x^T A x &= y^T y, \qquad \text{where } y = Cx \\ \Rightarrow x^T A x &\geq 0, \qquad \text{since } y^T y \geq 0 \end{aligned}$$

■

Note that by statement 3, a positive semi-definite matrix can be decomposed into $C^T C$. Let $C = [C_1, C_2 \cdots C_n]$, where $C_i \in \mathbb{R}^m$. Thus A_{ij} is the dot product of C_i and C_j. This gives us the following corollary.

Corollary 13.1.4 *SDP is equivalent to vector programming.*

13.1.3 Feasible Region of an SDP

Similar to an LP, a linear constraint in an SDP produces a hyper-plane (or a flat face) that restricts the feasible region of the SDP. The semi-definite constraint, which is nonlinear, produces a non-flat face. In fact, as we discussed at the end of Section 13.1.1, this nonlinear constraint can be interpreted as an infinite number of linear constraints. As an example, the feasible region of an SDP can be visualized as in Figure 13.1.3. In the figure, C_1, C_2 and C_3 are three linear constraints, and C_4 is the nonlinear constraint. Constraints C_a and C_b are two instances among the infinite number of linear constraints corresponding to C_4. These infinite linear constraints produces the non-flat face of C_4.

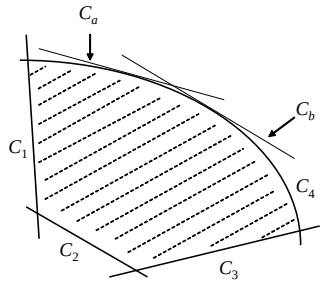

Figure 13.1.1: The feasible region of an SDP

The optimal solution to an SDP can lie on the non-flat face of the feasible region and thus can be irrational. Below we give a simple example of an irrational optimum to illustrate this point.

Example: Minimize x subject to the constraint:

$$\begin{bmatrix} x & \sqrt{2} \\ \sqrt{2} & x \end{bmatrix} \succeq 0.$$

For positive semi-definite matrices, all the leading principal minors are non-negative. The leading principal minors of an $n \times n$ matrix are the determinants of the submatrices obtained by deleting the last k rows and the last k columns, where $k = n-1, n-2, \ldots, 0$. In the example, the matrix has two leading principal minors:

$$\begin{aligned} m_1 &= \big|[x]\big| = x, \quad \text{and} \\ m_2 &= \left|\begin{bmatrix} x & \sqrt{2} \\ \sqrt{2} & x \end{bmatrix}\right| = x^2 - 2. \end{aligned}$$

Thus we have $x \geq 0$ and $x^2 - 2 \geq 0$. It immediately follows that the minimum value of x is $\sqrt{2}$. ■

Finally we state without proving the following theorem on SDP approximation.

Theorem 13.1.5 *We can achieve a (1+ϵ)-approximation to an SDP in time polynomial to n and $1/\epsilon$, where n is the size of the program.*

13.2 Max-Cut

Recall that for the Max-Cut problem we want to find a non-trivial cut of a given graph such that the edges crossing the cut are maximized, i.e.

Given: $G = (V, E)$ with c_e =cost on edge e.
Goal: Find a partition (V_1, V_2), $V_1, V_2 \neq \phi$, $\max \sum_{e \in (V_1 \times V_2) \cap E} c_e$.

13.2.1 Representations

How can we convert Max-Cut into a Vector Program?

13.2.1.1 Quadratic Programs

First write Max-Cut as a Quadratic Program.

Let $x_u = 0$ if vertex u is on the left side of the cut and $x_u = 1$ if vertex u is on the right side of the cut. Then we have

Program 1:

$$\begin{array}{lll} \text{maximize} & \sum_{(u,v) \in E} (x_u(1 - x_v) + x_v(1 - x_u))c_{uv} & \text{s.t.} \\ & x_u \in \{0, 1\} & \forall u \in V \end{array}$$

Alternatively, let $x_u = -1$ if vertex u is on the left side of the cut and $x_u = 1$ if vertex u is on the right side of the cut. Then we have

Program 2:

$$\begin{array}{lll} \text{maximize} & \sum_{(u,v) \in E} \frac{(1 - x_u x_v)}{2} c_{uv} & \text{s.t.} \\ & x_u \in \{-1, 1\} & \forall u \in V \end{array}$$

Note that $x_u x_v = -1$ exactly when the edge (u, v) crosses the cut.

We can express the integrality constraint as a quadratic constraint yielding

Program 3:

$$\begin{array}{lll} \text{maximize} & \sum_{(u,v)\in E} \frac{(1-x_u x_v)}{2} c_{uv} & \text{s.t.} \\ & x_u^2 = 1 & \forall u \in V \end{array}$$

In these programs an edge contributes c_{uv} exactly when the edge crosses the cut. So in any solution to these quadratic programs the value of the objective function exactly equals the total cost of the cut. We now have an exact representation of the Max-Cut Problem. If we can solve Program 3 exactly we can solve the Max-Cut Problem exactly.

13.2.1.2 Vector Program

Now we want to relax Program 3 into a Vector Program.

Recall:
A Vector Program is a Linear Program over dot products.

So relax Program 3 by thinking of every product as a dot product of two n-dimensional vectors, x_u.

Program 4:

$$\begin{array}{lll} \text{maximize} & \sum_{uv\in E} \frac{(1-x_u \cdot x_v)}{2} c_{uv} & \text{s.t.} \\ & x_u \cdot x_u = 1 & \forall u \in V \end{array}$$

This is something that we know how to solve.

13.2.1.3 Semi-Definite Program

How would we write this as a Semi-Definite Program?

Recall:

Definition 13.2.1 *A Semi-Definite Program has a linear objective function subject to linear constraints over x_{ij} together with the semi-definite constraint $[x_{ij}] \succeq 0$.*

Theorem 13.2.2 *For any matrix A, A is a Positive Semi-Definite Matrix if the following holds.*

$$\begin{aligned} A \succeq 0 &\Rightarrow v^T A v \geq 0 \quad \forall v \in V \\ &\Leftrightarrow A = C^T C, \text{ where } C \in \mathbb{R}^{m \times n} \\ &\Leftrightarrow A_{ij} = C_i \cdot C_j \end{aligned}$$

To convert a Vector Program into a Semi-Definite Program replace dot products with variables and add the constraint that the matrix of dot products is Positive Semi-Definite.

So our Vector Program becomes

Program 5:

$$\begin{array}{lll} \text{maximize} & \sum_{(u,v)\in E} \frac{(1-y_{uv})}{2} c_{uv} & \text{s.t.} \\ & y_{uu} = 1 & \forall u \in V \\ & [y_{ij}] \succeq 0 & \end{array}$$

If we have any feasible solution to this Semi-Definite Program (Program 5), then by the above theorem there are vectors $\{x_u\}$ such that $y_{uv} = x_u \cdot x_v \forall u, v \in V$. The vectors $\{x_u\}$ are a feasible solution to the Vector Program (Program 4). Thus the Semi-Definite Program (Program 5) is exactly equivalent to the Vector Program (Program 4).

13.2.2 Solving Vector Programs

We know how to get arbitrarily close to the exact solution to a Vector Program. So we get some set of vectors for which each vertex is mapped to some n-dimensional vector around the origin. These vectors are all unit vectors by the constraint that $x_u \cdot x_u = 1$, so the vectors all lie in some unit ball around the origin.

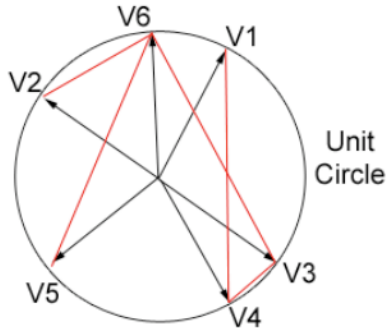

Figure 13.2.2: The unit vectors in the Vector Program solution.

Now we have some optimal solution to the Vector Program. Just as in the case when we had an exact representation as an Integer Program and relaxed it to a Linear Program to get an optimal solution that was even better than the optimal integral solution to the Integer Program, here we have some exact representation of the problem and we are relaxing it to some program that solves the program over a larger space. The set of integral solutions to Program 3 form a subset of feasible solutions to Program 4, i.e., Program 4 has a larger set of feasible solutions. Thus the optimal solution for the Vector Program is going to be no worse than the optimal solution to the original problem.

Fact: $OPT_{VP} \geq OPT_{Max-Cut}$.

Goal: Round the vector solution obtained by solving the VP to an integral solution (V_1, V_2) such that the total value of our integral solution $\geq \alpha OPT_{VP} \geq \alpha OPT_{Max-Cut}$.

Our solution will put some of the vectors on one side of the cut and the rest of the vectors on the other side of the cut. Our solution is benefitting from the edges going across the cut that is produced. Long edges crossing the cut will contribute more to the solution than short edges

crossing the cut because the dot product is smaller for the longer edges than the shorter edges. We want to come up with some way of partitioning these vertices so that we are more likely to cut long edges.

13.2.3 Algorithm

In two dimensions good cut would be some plane through the origin such that vectors on one side of the plane go on one side of the cut and vectors that go on the other side of the plane go on the other side of the cut. In this way we divide contiguous portions of space rather than separating the vectors piecemeal.

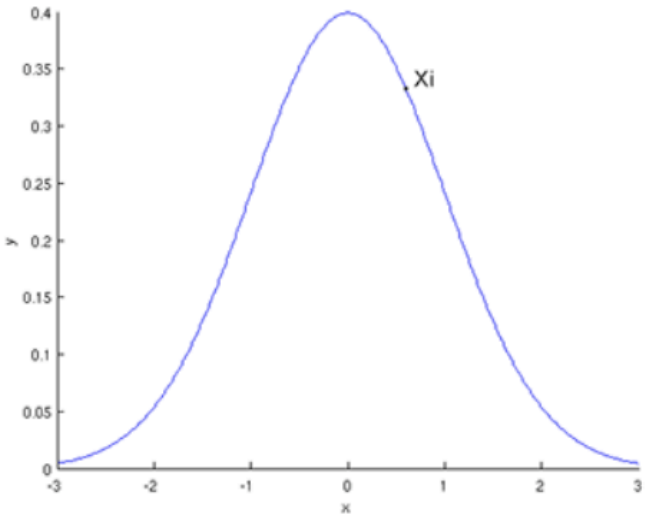

Figure 13.2.3: A sample Gaussian variable, x.

1. Pick a "random" direction - unit vector $\hat{n}$.
2. Define $V_1 = \{v | x_v \cdot \hat{n} \geq 0\}$, $V_2 = \{v | x_v \cdot \hat{n} < 0\}$.
3. Output (V_1, V_2).

To pick a "random" direction we want to pick a vector uniformly at random from the set of all unit vectors. Suppose we know how to pick a normal, or gaussian, variable in one dimension. Then pick from this normal distribution for every dimension.

This gives us a point that is spherically symmetric in the distribution. The density of any point x in the distribution is proportional to $exp^{-\frac{1}{2}x^2}$. So if each of the components, x_i is picked using this density, the density of the vector is proportional to $\prod_i exp^{-\frac{1}{2}x_i^2} = exp^{(-\frac{1}{2}\sum_i x_i^2)}$, which depends only on the length of the vector and not the direction. Now normalize the vector to make it a unit vector.

We now want to show that the probability that an edge is cut is proportional to it's length. In this way we are more likely to cut the longer edges that contribute more to the value of the cut. So what is the probability that we will cut any particular edge?

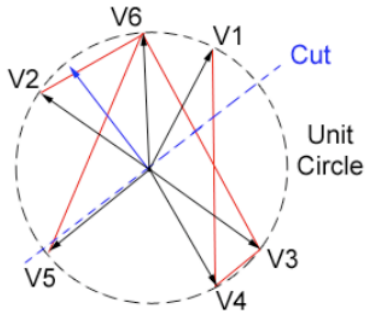

Figure 13.2.4: An example cut across the unit circle.

From Figure 13.2.4 we can see that

$Pr[$ our algorithm cuts edge $(u, v)] = \frac{\theta_{uv}}{\pi}$.

This is for two dimensions, what about n dimensions? If the cutting plane is defined by some random direction in n dimensions, then we can project down to two dimensions and it is still uniformly at random over the n dimensions. So we have the same probability for n dimensions. So the expected value of our solution is

$E[$ our solution $] = \sum_{(u,v)\in E} \frac{\theta_{uv}}{\pi} c_{uv}$

In terms of θ_{uv} the value of our Vector Program is

$Val_{VP} = \sum_{(u,v)\in E} (\frac{1-cos(\theta_{uv})}{2}) c_{uv}$

Now we want to say that $E[$ our solution $]$ is not much smaller than Val_{VP}. So we look at the ratio of $E[$ our solution $]$ to Val_{VP} is not small.

Claim 13.2.3 *For all* θ, $\frac{2\theta}{\pi(1-cos\theta)} \geq 0.878$

As a corollary we have the following theorem. (Note: $1.12 \approx 1/0.878$.)

Theorem 13.2.4 *We get a* 1.12*-approximation.*

13.2.4 Wrap-up

This algorithm is certainly better than the 2-approximation that we saw before. Semi-Definate programming is a powerful technique, and for a number of problems gives us stronger approximations than just Linear Programming relaxations. As it turns out, the Semi-Definate solution for Max-cut is the currently best-known approximation for the problem.

There is reason to believe that unless $P = NP$, you cannot do better than this approximation. This result is highly surprising, given that we derived this number from a geometric argument,

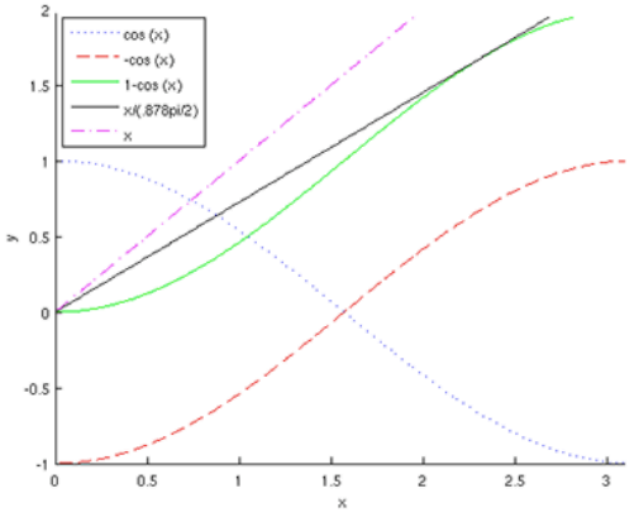

Figure 13.2.5: A visual representation of the .878-approximation to the solution.

which seems like it should have nothing to do with the $P = NP$ problem.