St. Francis Xavier University
Department of Computer Science

CSCI 550: Approximation Algorithms
Lecture 7: Randomized Rounding of Semidefinite Programs
Fall 2021

1 Enter the Matrix

Thus far, we have focused on modeling problems using both integer programs and their relaxed forms, linear programs. We have seen that there are methods to solve linear programs and to round solutions to linear programs in order to obtain solutions to the corresponding integer programs. However, there were some issues surrounding these programs and methods: first, integer programs cannot be solved in a reasonable amount of time, so we need to perform the relaxation step each time we want a solution to an integer program; and second, some methods to solve linear programs aren't guaranteed to run in polynomial time.

In this lecture, we will introduce a notion closely related to integer programs and linear programs called *semidefinite programming*. A semidefinite program is like a generalization of a linear program that makes use of a special kind of matrix in its objective function and constraints. Semidefinite programming is a newer development in the field of optimization theory, but it is an exciting one, as we can typically find solutions to semidefinite programs efficiently in both theory and practice.

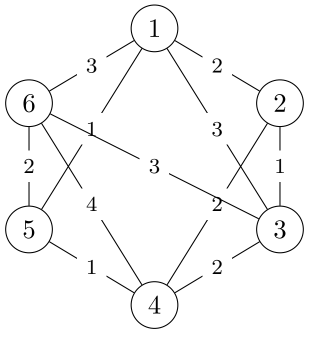
The problem we will focus on in this lecture is known as the *maximum cut problem*, which is a problem specific to graphs. A *cut* of a graph is a partitioning of the vertices of the graph into two disjoint subsets, determined by the subset of edges in the graph that must be removed in order to perform this partitioning. Finding a maximum cut in a graph, then, involves finding a single cut in a given graph that removes as many edges as possible. We can equivalently think of a maximum cut as giving the largest number of edges between two disjoint subsets of vertices.

As we did in the last lecture with the maximum satisfiability problem, here we will consider the *weighted* variant of the maximum cut problem, where we wish to maximize the sum of weights of the cut edges.

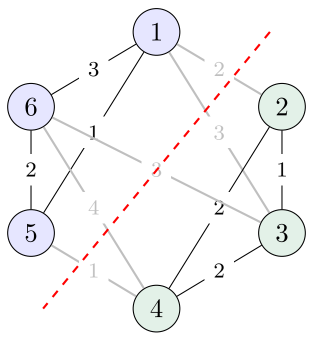
Weighted-Max-Cut
Given: an undirected graph $G = (V, E)$ and an associated weight $w_{ij} \geq 0$ for each edge $(i, j) \in E$
Determine: a subset of vertices $S \subseteq V$ that maximizes $w_S = \sum_{\substack{i \in S, j \notin S \\ \text{or } i \notin S, j \in S}} w_{ij}$

Note that, in our definition, each edge in the set E has a weight associated with it. If there does not exist an edge between two vertices i and j, then we can assume that $w_{ij} = 0$.

Example 1. Consider the following graph G, where the numbers on each edge denote the weight of that edge:



We can specify a cut by partitioning the set of vertices of G into two disjoint subsets. We can construct a subset S of vertices to act as one of these disjoint subsets, and all vertices not in S will constitute the other disjoint subset. Suppose, in this example, we take $S = \{1, 5, 6\}$. Then $V \setminus S = \{2, 3, 4\}$.



The weight of this cut is equal to the sum of the edges (i, j) where vertex i is in S and vertex j is not in S. Therefore, in this case, the weight of the cut is $2 + 3 + 3 + 4 + 1 = 13$.

1.1 A Naïve Approach

Just like with the weighted maximum satisfiability problem, we can use randomness to obtain a reasonably good solution to the weighted maximum cut problem. Given a graph, we will go through the entire vertex set $V = \{1, \ldots, n\}$ and, for each vertex, we will randomly choose whether or not to add that vertex to our subset S. This idea forms the basis for our naïve algorithm.

Algorithm 1: Weighted maximum cut problem—naïve

```
$S \leftarrow \emptyset$
for $1 \le i \le n$ do
    if random $\left(\frac{1}{2}\right) = 1$ then
        $S \leftarrow S \cup \{i\}$
```

It should perhaps come as no surprise that, if we choose to add a vertex to our subset S with probability $\frac{1}{2}$, then our naïve algorithm will have a performance guarantee of $\frac{1}{2}$.

Theorem 2. *Algorithm 1 gives a randomized $\frac{1}{2}$-approximation algorithm for the weighted maximum cut problem.*

Proof. For all vertices $i, j \in V$, define a random variable X_{ij} as follows:

$$X_{ij} = \begin{cases} 1 & \text{if } i \in S \text{ and } j \notin S \text{ or if } i \notin S \text{ and } j \in S; \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Denote the random variable corresponding to the total weight of the cut by $W = \sum_{i<j} w_{ij} X_{ij}$. The expected value of W, then, is the value of the cut obtained by Algorithm 1. That is,

$$\mathbb{E}[W] = \mathbb{E}\left[\sum_{i<j} w_{ij} X_{ij}\right].$$

By linearity of expectation, we know that

$$\begin{aligned}\mathbb{E}[W] &= \sum_{i<j} w_{ij} \cdot \mathbb{E}[X_{ij}] \\ &= \sum_{i<j} w_{ij} \cdot \mathbb{P}[(i \in S, j \notin S) \text{ or } (i \notin S, j \in S)] \\ &= \frac{1}{2} \cdot \sum_{i<j} w_{ij} \\ &\geq \frac{1}{2} \cdot \text{OPT},\end{aligned}$$

since each weight w_{ij} being nonnegative implies the sum of all weights is an upper bound for the optimal solution. □

Interestingly, Algorithm 1 is one of the first examples of a randomized approximation algorithm, having been studied by Erdős in the late 1960s. We can also derandomize Algorithm 1 to obtain a deterministic algorithm that retains the performance guarantee of $\frac{1}{2}$.

What may come as a surprise, however, is the fact that—unlike the incremental improvements we obtained with our algorithms for the weighted maximum satisfiability problem—we can't do any better than a performance guarantee of $\frac{1}{2}$ for the weighted maximum cut problem using the tools we've learned so far in the course. In fact, the maximum cut problem is APX-hard, meaning that no polynomial-time approximation scheme exists for this problem unless $\mathsf{P} = \mathsf{NP}$. This hardness result implies that every approximation algorithm for the maximum cut problem will have a performance guarantee that is strictly less than 1.

We don't need to be satisfied with our naïve algorithm, though. Let's now introduce the main technique of this lecture, which we will use to obtain an approximation algorithm for our problem with an improved performance guarantee.

1.2 Semidefinite Programming

The technique we will focus on in this lecture involves the use of *semidefinite programs*, which generalize the notion of linear programs in the sense that they are not just a linear relaxation, but a *convex* relaxation of an integer program for a given problem. By "convex" here, we are referring to *convex functions*, or functions where the line segment between any two points of the function does not lie below the function between those two points.

Semidefinite programs rely on the use of certain special matrices. We will not review all of linear algebra here, but we will fix some definitions and conventions: if $M \in \mathbb{R}^{m \times n}$ is a matrix, then we write $M^\mathsf{T} \in \mathbb{R}^{n \times m}$ to denote the *transpose* of M. A matrix M is *symmetric* if $M = M^\mathsf{T}$. We also assume that a vector with n elements, $\vec{v} \in \mathbb{R}^n$, is a column vector. Therefore, $\vec{v}^\mathsf{T}\vec{v}$ denotes the dot product of $\vec{v}$ with itself, while $\vec{v}\vec{v}^\mathsf{T}$ produces a matrix of size $n \times n$.

The "special matrices" we referred to earlier are *positive semidefinite matrices.* Such matrices are symmetric, and they have the added property that their quadratic form (i.e., $\vec{v}^\mathsf{T}M\vec{v}$ for all vectors $\vec{v} \in \mathbb{R}^n$) is always zero or positive.

Definition 3 (Positive semidefinite matrix)**.** A symmetric matrix $M \in \mathbb{R}^{n \times n}$ is positive semidefinite if, for all vectors $\vec{v} \in \mathbb{R}^n$, we have that $\vec{v}^\mathsf{T}M\vec{v} \geq 0$.

As a shorthand, we sometimes indicate that a matrix M is positive semidefinite by writing $M \succeq 0$. We can characterize positive semidefinite matrices in a handful of ways.

Proposition 4. *Given a symmetric matrix $M \in \mathbb{R}^{n \times n}$, each of the following statements are equivalent:*

1. *M is positive semidefinite;*
2. *all eigenvalues of M are nonnegative; and*
3. *$M = N^{\mathsf{T}} N$ for some matrix $N \in \mathbb{R}^{m \times n}$, where $m \leq n$.*

Example 5. Each of the following matrices are positive semidefinite:

- the $n \times n$ identity matrix, I_n;
- the $n \times n$ matrix J_n where all entries are 1s; and
- the matrix $M = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$.

Given a positive semidefinite matrix M, we can define a semidefinite program that uses M. A semidefinite program is similar to a linear program, in that we still have an objective function and a set of constraints that must be satisfied. However, semidefinite programs also incorporate entries of the matrix M and constrain each entry in such a way that, overall, M must remain positive semidefinite. Specifically, a semidefinite program is of the following form, where m_{ij} is the entry at row i and column j of M:

$$\begin{aligned} \text{maximize/minimize} \quad & \sum_{i,j} c_{ij} m_{ij} \\ \text{subject to} \quad & \sum_{i,j} a_{ijk} m_{ij} = b_k, \quad \text{for all } k \\ & m_{ij} = m_{ji}, \quad \text{for all } i \text{ and } j \\ & M \succeq 0 \end{aligned} \tag{SDP}$$

The primary benefit we get from using semidefinite programs comes from the fact that such programs can *always* be solved in polynomial time using modifications of methods we use to solve linear programs—especially the ellipsoid method—and the solution to the semidefinite program comes within an additive factor of $\epsilon > 0$. Moreover, the time needed to solve the semidefinite program is polynomial in both the size of the input and in $\log(1/\epsilon)$.

If we don't wish to work with matrices, then we may reformulate semidefinite programs as *vector programs*: programs whose variables are vectors $\vec{v}_i \in \mathbb{R}^n$, where the dimension n is equal to the number of vectors in the vector program. The objective function and constraints of a vector program are also linear in the dot product of each vector. Overall, vector programs take the following form:

$$\begin{aligned} \text{maximize/minimize} \quad & \sum_{i,j} c_{ij} \left(\vec{v}_i^{\mathsf{T}} \vec{v}_j\right) \\ \text{subject to} \quad & \sum_{i,j} a_{ijk} \left(\vec{v}_i^{\mathsf{T}} \vec{v}_j\right) = b_k, \quad \text{for all } k \\ & \vec{v}_i \in \mathbb{R}^n, \quad \text{for all } 1 \leq i \leq n \end{aligned} \tag{VP}$$

Indeed, the two approaches (semidefinite programs and vector programs) are equivalent, since a symmetric matrix M is positive semidefinite if and only if $M = N^{\mathsf{T}} N$ for some other matrix N. Thus, given a solution X to a semidefinite program, we can compute in polynomial time some matrix Y such that $X = Y^{\mathsf{T}} Y$, and then take $\vec{y}_i$ to be the ith column of Y; that is,

$$Y = \begin{bmatrix} \vdots & \vdots & & \vdots \\ \vec{y}_1 & \vec{y}_2 & \dots & \vec{y}_n \\ \vdots & \vdots & & \vdots \end{bmatrix}.$$

In this way, we have that $x_{ij} = \vec{y}_i^{\mathsf{T}}\vec{y}_j$, making each $\vec{y}_i$ a feasible solution to the same value of the corresponding vector program. We can also perform this conversion in the other direction by constructing a matrix from the set of vectors, thus giving us a solution to a semidefinite program from a solution to the corresponding vector program.

1.3 A Refined Approach

Now that we've established the fundamentals of semilinear programs, we can formulate our weighted maximum cut problem as a semilinear program. To begin, let's consider a general formulation of the weighted maximum cut problem:

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2} \cdot \sum_{i<j} w_{ij}\,(1 - y_i y_j) \\ \text{subject to} \quad & y_i \in \{-1, +1\}, \quad \text{for all } 1 \le i \le n \end{aligned} \tag{WMC-P}$$

How does this program model the weighted maximum cut problem? Given some graph $G = (V, E)$, consider the cut given by the subset $S = \{i \in V \mid y_i = -1\}$. Naturally, then, we would have that $V \setminus S = \{i \in V \mid y_i = +1\}$. If an edge (i, j) is in this cut, then the product $y_i y_j$ will be equal to -1, while it will be equal to $+1$ if the edge is not in the cut. Then, the expression

$$\frac{1}{2} \cdot \sum_{i<j} w_{ij}\,(1 - y_i y_j)$$

gives the total weight of all edges in the cut: if $y_i y_j = +1$, then $w_{ij}(1 - y_i y_j) = 0$ and the edge doesn't contribute to the sum, while if $y_i y_j = -1$, then $w_{ij}(1 - y_i y_j) = 2w_{ij}$ and multiplying by $1/2$ contributes w_{ij} to the sum.

Let's now take our program WMC-P and reformulate it as a vector program:

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2} \cdot \sum_{i<j} w_{ij}\left(1 - \vec{v}_i^{\mathsf{T}}\vec{v}_j\right) \\ \text{subject to} \quad & \vec{v}_i^{\mathsf{T}}\vec{v}_i = 1, \quad \text{for all } 1 \le i \le n \\ & \vec{v}_i \in \mathbb{R}^n, \quad \text{for all } 1 \le i \le n \end{aligned} \tag{WMC-VP}$$

Indeed, our program WMC-VP is a relaxation of our program WMC-P: we can take each variable y_i in the program WMC-P to be a vector $\vec{v}_i = (y_i, 0, \ldots, 0)$, and then we have both that $\vec{v}_i^{\mathsf{T}}\vec{v}_i = 1$ for all i and that $\vec{v}_i^{\mathsf{T}}\vec{v}_j = y_i y_j$ for all i and j. By this observation, we also know that $Z_{\text{VP}} \ge \text{OPT}$, where Z_{VP} denotes the value of an optimal solution to the program WMC-VP.

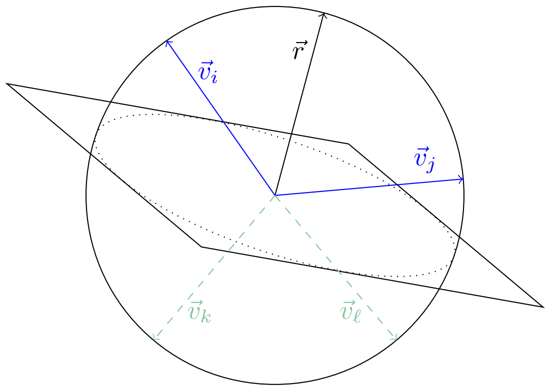
Now, we know from the previous section that we can solve the program WMC-VP in polynomial time. All that remains is to obtain a method of converting from the program WMC-VP to the program WMC-P, and for that, we can use our familiar technique of randomized rounding.

Algorithm 2: Weighted maximum cut problem—randomized rounding

```
solve the program WMC-VP and get a set of vectors $\vec{v}^*$
choose a random vector $\vec{r}$ uniformly from the unit $n$-sphere
$S \leftarrow \emptyset$
for $1 \le i \le n$ do
    if ${\vec{v}_i^*}^{\mathsf{T}}\vec{r} \ge 0$ then
        $S \leftarrow S \cup \{i\}$
```

What exactly is going on in this algorithm? We're applying randomized rounding, but adapted to work especially for vector programs. In the first step, we solve the vector program to obtain a solution consisting of a set of vectors. We then pick some random vector $\vec{r} = (r_1, \ldots, r_n)$ from the "unit n-sphere"; this is another way of saying that we draw each element of the vector uniformly at random from the normal distribution with a mean of 0 and a variance of 1. Then, the remainder of the algorithm follows through in much the same way as Algorithm 1: we iterate through each vertex and add it to our cut if $\vec{v}_i^{*\mathsf{T}}\vec{r} \geq 0$.

To get a bit more technical with the second step of Algorithm 2, the random vector $\vec{r}$ is the normal to some hyperplane. We know that all vectors $\vec{v}_i$ lie on the unit sphere, since each $\vec{v}_i$ is a unit vector and $\vec{v}_i^{\mathsf{T}}\vec{v}_i = 1$. Therefore, the hyperplane corresponding to the random vector $\vec{r}$ splits the unit sphere in half. Moreover, all vectors $\vec{v}_i$ where $\vec{v}_i^{*\mathsf{T}}\vec{r} \geq 0$ will fall into one half of the sphere (and, therefore, will be added to our cut), while all other vectors will fall into the other half of the sphere.



We can state a couple of facts about the random vector $\vec{r}$, which we will need in the coming proofs. We will not prove these facts here, though.

Proposition 6. *The normalization of the vector $\vec{r}$, $\widehat{r} = \vec{r}/\|\vec{r}\|$, is uniformly distributed over the unit n-sphere.*

Proposition 7. *Let $\vec{r}\,'$ denote the projection of the vector $\vec{r}$ onto a two-dimensional plane. Then the normalization of the vector $\vec{r}\,'$, $\widehat{r}\,' = \vec{r}\,'/\|\vec{r}\,'\|$, is uniformly distributed over a unit circle in the plane.*

We now prove one required lemma before considering the main theorem of this section.

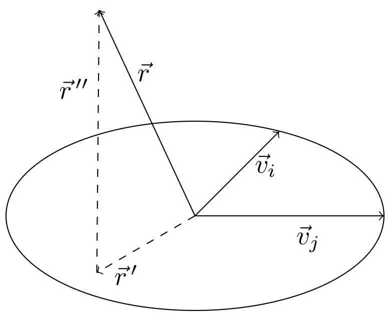
Lemma 8.

$$\mathbb{P}[(i \in S, j \notin S) \text{ or } (i \notin S, j \in S)] = \frac{1}{\pi} \cdot \arccos\left(\vec{v}_i^{\mathsf{T}}\vec{v}_j\right).$$

Proof. Let $\vec{r}\,'$ denote the projection of $\vec{r}$ onto the two-dimensional plane defined by $\vec{v}_i$ and $\vec{v}_j$. If $\vec{r} = \vec{r}\,' + \vec{r}\,''$, then $\vec{r}\,''$ is orthogonal to both $\vec{v}_i$ and $\vec{v}_j$, and

$$\begin{aligned}\vec{v}_i^{*\mathsf{T}}\vec{r} &= \vec{v}_i^{*\mathsf{T}}\left(\vec{r}\,' + \vec{r}\,''\right)\\ &= \vec{v}_i^{*\mathsf{T}}\vec{r}\,'.\end{aligned}$$

Similarly, we have that $\vec{v}_j^{*\mathsf{T}}\vec{r} = \vec{v}_j^{*\mathsf{T}}\vec{r}\,'$.

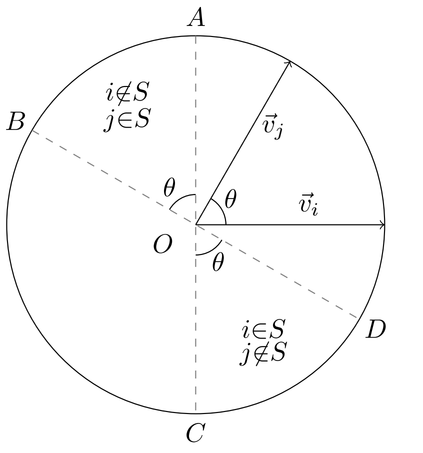


Now, on the two-dimensional plane, consider some line AC drawn perpendicular to $\vec{v}_i$ and some line BD drawn perpendicular to $\vec{v}_j$. Both of these lines pass through the origin O. Moreover, if $\vec{v}_i$ and $\vec{v}_j$ are at an angle of θ radians to one another, then $\angle AOB = \angle COD = \theta$ as well.

By Proposition 7, $\vec{r}'$ with its tail at O is oriented with respect to $\vec{v}_i$ at an angle α chosen uniformly at random from $[0, 2\pi)$. Then,

- if $\vec{r}'$ is to the right of the line AC, then $\vec{v}_i^{\mathsf{T}}\vec{r} \geq 0$ and $i \in S$; and
- if $\vec{r}'$ is to the right of the line BD, then $\vec{v}_j^{\mathsf{T}}\vec{r} \geq 0$ and $j \in S$.

We therefore have four "zones" of the unit circle that uniquely specify whether or not i and j are in S. In particular, the sector AOB corresponds to the case where $i \notin S$ and $j \in S$, while the sector COD corresponds to the case where $i \in S$ and $j \notin S$.



The probability that the angle of $\vec{r}'$, α, corresponds to edge (i, j) being in the cut is $2\theta/2\pi$, since α must fall into one of the two sectors AOB or COD, and both sectors make up a total of 2θ radians out of the unit circle's 2π radians. Thus, α will fall into one of these sectors with probability θ/π.

Since $\vec{v}_i^{\mathsf{T}}\vec{v}_j = \|\vec{v}_i\|\|\vec{v}_j\|\cos(\theta)$, and since $\vec{v}_i$ and $\vec{v}_j$ are both unit vectors, we have that $\theta = \arccos\left(\vec{v}_i^{\mathsf{T}}\vec{v}_j\right)$. Dividing by π gives us the desired result. $\square$

Finally, using our lemma, we can determine the performance guarantee of our more refined algorithm that uses randomized rounding. Indeed, the performance guarantee is *much* better than our previous guarantee of $\frac{1}{2}$.

Theorem 9. *Algorithm 2 gives a randomized* 0.878*-approximation algorithm for the weighted maximum cut problem.*

Proof. For all vertices $i, j \in V$, define a random variable X_{ij} as follows:

$$X_{ij} = \begin{cases} 1 & \text{if } i \in S \text{ and } j \notin S \text{ or if } i \notin S \text{ and } j \in S;\text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Denote the random variable corresponding to the total weight of the cut by $W = \sum_{i<j} w_{ij} X_{ij}$. Then, by Lemma 8, we have that

$$\begin{aligned} \mathbb{E}[W] &= \sum_{i<j} w_{ij} \cdot \mathbb{P}[(i \in S, j \notin S) \text{ or } (i \notin S, j \in S)] \\ &= \sum_{i<j} w_{ij} \cdot \frac{1}{\pi} \cdot \arccos\left(\vec{v}_i^{\mathsf{T}}\vec{v}_j\right). \end{aligned}$$

We can verify computationally that, for all $x \in [-1, 1]$,

$$\frac{1}{\pi} \cdot \arccos(x) \geq 0.878 \cdot \frac{1}{2}(1 - x).$$

Thus, by substitution, we can bound each term $\frac{1}{\pi} \cdot \arccos\left(\vec{v}_i^{\mathsf{T}} \vec{v}_j\right)$ in the sum from below by the value $0.878 \cdot \frac{1}{2}\left(1 - \vec{v}_i^{\mathsf{T}} \vec{v}_j\right)$, and therefore

$$\begin{aligned}
\mathbb{E}[W] &\geq 0.878 \cdot \frac{1}{2} \cdot \sum_{i<j} w_{ij} \left(1 - \vec{v}_i^{\mathsf{T}} \vec{v}_j\right) \\
&\geq 0.878 \cdot Z_{\mathrm{VP}} \\
&\geq 0.878 \cdot \mathrm{OPT},
\end{aligned}$$

by our previous observation that $Z_{\mathrm{VP}} \geq \mathrm{OPT}$. □