Approximate Max Cut

Lecture 24

November 19, 2014

Part I

Normal distribution

Normal distribution – proof

$$\tau^2 = \left(\int_{x=-\infty}^{\infty} \exp\left(-\frac{x^2}{2} \right) \mathrm{d}x \right)^2$$

Normal distribution – proof

$$\tau^2 = \left(\int_{x=-\infty}^{\infty} \exp\left(-\frac{x^2}{2} \right) \mathrm{d}x \right)^2$$
$$= \int_{(x,y)\in\mathbb{R}^2} \exp\left(-\frac{x^2+y^2}{2} \right) \mathrm{d}x\mathrm{d}y$$

Normal distribution – proof

$$\tau^2 = \left(\int_{x=-\infty}^{\infty} \exp\left(-\frac{x^2}{2} \right) \mathrm{d}x \right)^2$$

$$= \int_{(x,y)\in\mathbb{R}^2} \exp\left(-\frac{x^2+y^2}{2} \right) \mathrm{d}x\mathrm{d}y$$

Change of vars: $x = r\cos c$ $y = r\sin c$

Normal distribution – proof

$$\tau^2 = \left(\int_{x=-\infty}^{\infty} \exp\left(-\frac{x^2}{2} \right) \mathrm{d}x \right)^2$$

$$= \int_{(x,y)\in\mathbb{R}^2} \exp\left(-\frac{x^2+y^2}{2} \right) \mathrm{d}x\mathrm{d}y$$

Change of vars: $\begin{aligned} x &= r\cos\alpha \\ y &= r\sin\alpha \end{aligned}$

$$= \int_{\alpha=0}^{2\pi} \int_{r=0}^{\infty} \exp\left(-\frac{r^2}{2} \right) \left| \det \begin{pmatrix} \frac{\partial r\cos\alpha}{\partial r} & \frac{\partial r\cos\alpha}{\partial \alpha} \\ \frac{\partial r\sin\alpha}{\partial r} & \frac{\partial r\sin\alpha}{\partial \alpha} \end{pmatrix} \right| \mathrm{d}r\, \mathrm{d}\alpha$$

Normal distribution – proof

$$\tau^2 = \left(\int_{x=-\infty}^{\infty} \exp\left(-\frac{x^2}{2}\right) \mathrm{d}x \right)^2$$

$$= \int_{(x,y)\in\mathbb{R}^2} \exp\left(-\frac{x^2+y^2}{2}\right) \mathrm{d}x\mathrm{d}y$$

Change of vars: $\begin{array}{l} x = r\cos\alpha \\ y = r\sin\alpha \end{array}$

$$= \int_{\alpha=0}^{2\pi} \int_{r=0}^{\infty} \exp\left(-\frac{r^2}{2}\right) \left| \det \begin{pmatrix} \frac{\partial r\cos\alpha}{\partial r} & \frac{\partial r\cos\alpha}{\partial \alpha} \\ \frac{\partial r\sin\alpha}{\partial r} & \frac{\partial r\sin\alpha}{\partial \alpha} \end{pmatrix} \right| \mathrm{d}r\,\mathrm{d}\alpha$$

$$= \int_{\alpha=0}^{2\pi} \int_{r=0}^{\infty} \exp\left(-\frac{r^2}{2}\right) \left| \det \begin{pmatrix} \cos\alpha & -r\sin\alpha \\ \sin\alpha & r\cos\alpha \end{pmatrix} \right| \mathrm{d}r\,\mathrm{d}\alpha$$

Normal distribution – proof

$$\tau^2 = \left(\int_{x=-\infty}^{\infty} \exp\left(-\frac{x^2}{2}\right) \mathrm{d}x \right)^2$$

$$= \int_{(x,y)\in\mathbb{R}^2} \exp\left(-\frac{x^2+y^2}{2}\right) \mathrm{d}x\mathrm{d}y$$

Change of vars: $\begin{array}{c} x = r\cos\alpha \\ y = r\sin\alpha \end{array}$

$$= \int_{\alpha=0}^{2\pi}\int_{r=0}^{\infty} \exp\left(-\frac{r^2}{2}\right) \left|\det\begin{pmatrix} \frac{\partial r\cos\alpha}{\partial r} & \frac{\partial r\cos\alpha}{\partial\alpha} \\ \frac{\partial r\sin\alpha}{\partial r} & \frac{\partial r\sin\alpha}{\partial\alpha} \end{pmatrix}\right| \mathrm{d}r\,\mathrm{d}\alpha$$

$$= \int_{\alpha=0}^{2\pi}\int_{r=0}^{\infty} \exp\left(-\frac{r^2}{2}\right) \left|\det\begin{pmatrix} \cos\alpha & -r\sin\alpha \\ \sin\alpha & r\cos\alpha \end{pmatrix}\right| \mathrm{d}r\,\mathrm{d}\alpha$$

$$= \int_{\alpha=0}^{2\pi}\int_{r=0}^{\infty} \exp\left(-\frac{r^2}{2}\right) r\,\mathrm{d}r\,\mathrm{d}\alpha$$

Normal distribution – proof

$$\tau^2 = \left(\int_{x=-\infty}^{\infty} \exp\left(-\frac{x^2}{2} \right) \mathrm{d}x \right)^2$$

$$= \int_{(x,y)\in\mathbb{R}^2} \exp\left(-\frac{x^2+y^2}{2} \right) \mathrm{d}x\mathrm{d}y \qquad \text{Change of vars: } \begin{array}{l} x = r\cos \alpha \\ y = r\sin \alpha \end{array}$$

$$= \int_{\alpha=0}^{2\pi} \int_{r=0}^{\infty} \exp\left(-\frac{r^2}{2} \right) \left| \det \begin{pmatrix} \frac{\partial r\cos\alpha}{\partial r} & \frac{\partial r\cos\alpha}{\partial \alpha} \\ \frac{\partial r\sin\alpha}{\partial r} & \frac{\partial r\sin\alpha}{\partial \alpha} \end{pmatrix} \right| \mathrm{d}r\,\mathrm{d}\alpha$$

$$= \int_{\alpha=0}^{2\pi} \int_{r=0}^{\infty} \exp\left(-\frac{r^2}{2} \right) \left| \det \begin{pmatrix} \cos\alpha & -r\sin\alpha \\ \sin\alpha & r\cos\alpha \end{pmatrix} \right| \mathrm{d}r\,\mathrm{d}\alpha$$

$$= \int_{\alpha=0}^{2\pi} \int_{r=0}^{\infty} \exp\left(-\frac{r^2}{2} \right) r\,\mathrm{d}r\,\mathrm{d}\alpha$$

$$= \int_{\alpha=0}^{2\pi} \left[-\exp\left(-\frac{r^2}{2} \right) \right]_{r=0}^{\infty} \mathrm{d}\alpha$$

Normal distribution – proof

$$\tau^2 = \left(\int_{x=-\infty}^{\infty} \exp\left(-\frac{x^2}{2} \right) \mathrm{d}x \right)^2$$

$$= \int_{(x,y)\in\mathbb{R}^2} \exp\left(-\frac{x^2+y^2}{2} \right) \mathrm{d}x\mathrm{d}y$$

Change of vars: $\begin{array}{l} x = r\cos\alpha \\ y = r\sin\alpha \end{array}$

$$= \int_{\alpha=0}^{2\pi} \int_{r=0}^{\infty} \exp\left(-\frac{r^2}{2} \right) \left| \det \begin{pmatrix} \frac{\partial r\cos\alpha}{\partial r} & \frac{\partial r\cos\alpha}{\partial\alpha} \\ \frac{\partial r\sin\alpha}{\partial r} & \frac{\partial r\sin\alpha}{\partial\alpha} \end{pmatrix} \right| \mathrm{d}r\,\mathrm{d}\alpha$$

$$= \int_{\alpha=0}^{2\pi} \int_{r=0}^{\infty} \exp\left(-\frac{r^2}{2} \right) \left| \det \begin{pmatrix} \cos\alpha & -r\sin\alpha \\ \sin\alpha & r\cos\alpha \end{pmatrix} \right| \mathrm{d}r\,\mathrm{d}\alpha$$

$$= \int_{\alpha=0}^{2\pi} \int_{r=0}^{\infty} \exp\left(-\frac{r^2}{2} \right) r\,\mathrm{d}r\,\mathrm{d}\alpha$$

$$= \int_{\alpha=0}^{2\pi} \left[-\exp\left(-\frac{r^2}{2} \right) \right]_{r=0}^{\infty} \mathrm{d}\alpha = \int_{\alpha=0}^{2\pi} 1\,\mathrm{d}\alpha$$

Normal distribution – proof

$$\tau^2 = \left(\int_{x=-\infty}^{\infty} \exp\left(-\frac{x^2}{2} \right) \mathrm{d}x \right)^2$$

$$= \int_{(x,y)\in\mathbb{R}^2} \exp\left(-\frac{x^2+y^2}{2} \right) \mathrm{d}x\mathrm{d}y$$

Change of vars: $\begin{array}{l} x = r\cos \alpha \\ y = r\sin \alpha \end{array}$

$$= \int_{\alpha=0}^{2\pi} \int_{r=0}^{\infty} \exp\left(-\frac{r^2}{2} \right) \left| \det \begin{pmatrix} \frac{\partial r\cos\alpha}{\partial r} & \frac{\partial r\cos\alpha}{\partial \alpha} \\ \frac{\partial r\sin\alpha}{\partial r} & \frac{\partial r\sin\alpha}{\partial \alpha} \end{pmatrix} \right| \mathrm{d}r\,\mathrm{d}\alpha$$

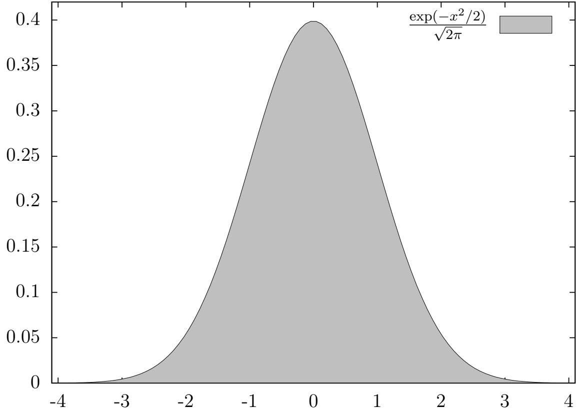
$$= \int_{\alpha=0}^{2\pi} \int_{r=0}^{\infty} \exp\left(-\frac{r^2}{2} \right) \left| \det \begin{pmatrix} \cos\alpha & -r\sin\alpha \\ \sin\alpha & r\cos\alpha \end{pmatrix} \right| \mathrm{d}r\,\mathrm{d}\alpha$$

$$= \int_{\alpha=0}^{2\pi} \int_{r=0}^{\infty} \exp\left(-\frac{r^2}{2} \right) r\,\mathrm{d}r\,\mathrm{d}\alpha$$

$$= \int_{\alpha=0}^{2\pi} \left[-\exp\left(-\frac{r^2}{2} \right) \right]_{r=0}^{\infty} \mathrm{d}\alpha = \int_{\alpha=0}^{2\pi} 1\,\mathrm{d}\alpha = 2\pi$$

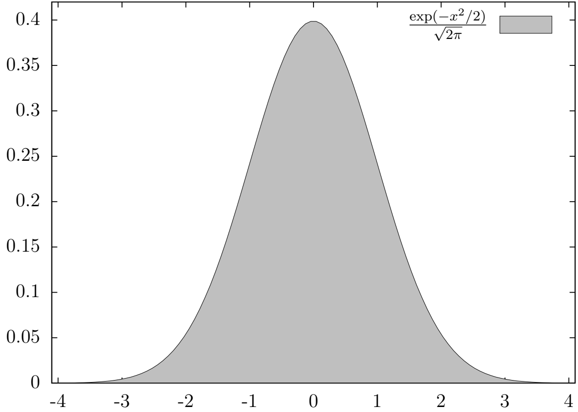
One dimensional normal distribution

1. A random variable X has ***normal distribution*** if $\Pr[X = x] = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)$.
2. $X \sim N(0, 1)$.



One dimensional normal distribution

1. A random variable X has ***normal distribution*** if $\Pr[X = x] = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)$.
2. $X \sim N(0,1)$.



Multidimensional normal distribution

1. A random variable X has ***normal distribution*** if $\Pr[X = x] = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)$.
2. $X \sim N(0,1)$.
3. $\mathrm{x} = (x_1, \ldots, x_n)$ has d-dimensional normal distributed (i.e., $\mathrm{v} \sim N^n(0,1)$
   $\iff v_1, \ldots, v_n \sim N(0,1)$
4. $\mathrm{v} \in \mathbb{R}^n$, such that $\|\mathrm{v}\| = 1$.
5. Let $\mathrm{x} \sim N^n(0,1)$. Then $z = \langle \mathrm{v}, \mathrm{x} \rangle$ has...
6. ...normal distribution!

Multidimensional normal distribution

1. A random variable X has ***normal distribution*** if $\Pr[X = x] = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)$.
2. $X \sim N(0, 1)$.
3. $\mathrm{x} = (x_1, \ldots, x_n)$ has d-dimensional normal distributed (i.e., $\mathrm{v} \sim N^n(0, 1)$
   $\iff \; v_1, \ldots, v_n \sim N(0, 1)$
4. $\mathrm{v} \in \mathbb{R}^n$, such that $\|\mathrm{v}\| = 1$.
5. Let $\mathrm{x} \sim N^n(0, 1)$. Then $z = \langle \mathrm{v}, \mathrm{x} \rangle$ has...
6. ...normal distribution!

Multidimensional normal distribution

1. A random variable X has ***normal distribution*** if $\Pr[X = x] = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)$.
2. $X \sim N(0,1)$.
3. $\mathrm{x} = (x_1, \ldots, x_n)$ has d-dimensional normal distributed (i.e., $\mathrm{v} \sim N^n(0,1)$)
   $\iff v_1, \ldots, v_n \sim N(0,1)$
4. $\mathrm{v} \in \mathbb{R}^n$, such that $\|\mathrm{v}\| = 1$.
5. Let $\mathrm{x} \sim N^n(0,1)$. Then $z = \langle \mathrm{v}, \mathrm{x} \rangle$ has...
6. ...normal distribution!

Multidimensional normal distribution

1. A random variable X has ***normal distribution*** if $\Pr[X = x] = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)$.
2. $X \sim N(0,1)$.
3. $\mathbf{x} = (x_1, \ldots, x_n)$ has d-dimensional normal distributed (i.e., $\mathbf{v} \sim N^n(0,1)$
   $\iff \; v_1, \ldots, v_n \sim N(0,1)$)
4. $\mathbf{v} \in \mathbb{R}^n$, such that $\|\mathbf{v}\| = 1$.
5. Let $\mathbf{x} \sim N^n(0,1)$. Then $z = \langle \mathbf{v}, \mathbf{x} \rangle$ has...
6. ...normal distribution!

Multidimensional normal distribution

1. A random variable X has ***normal distribution*** if $\Pr[X = x] = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)$.
2. $X \sim N(0,1)$.
3. $\mathrm{x} = (x_1, \ldots, x_n)$ has d-dimensional normal distributed (i.e., $\mathrm{v} \sim N^n(0,1)$ $\iff v_1, \ldots, v_n \sim N(0,1)$
4. $\mathrm{v} \in \mathbb{R}^n$, such that $\|\mathrm{v}\| = 1$.
5. Let $\mathrm{x} \sim N^n(0,1)$. Then $z = \langle \mathrm{v}, \mathrm{x} \rangle$ has...
6. ...normal distribution!

Multidimensional normal distribution

1. A random variable X has ***normal distribution*** if $\Pr[X = x] = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)$.
2. $X \sim N(0, 1)$.
3. $\mathbf{x} = (x_1, \ldots, x_n)$ has d-dimensional normal distributed (i.e., $\mathbf{v} \sim N^n(0, 1)$
   $\iff \; v_1, \ldots, v_n \sim N(0, 1)$
4. $\mathbf{v} \in \mathbb{R}^n$, such that $\|\mathbf{v}\| = 1$.
5. Let $\mathbf{x} \sim N^n(0, 1)$. Then $z = \langle \mathbf{v}, \mathbf{x} \rangle$ has...
6. ...normal distribution!

Multidimensional normal distribution

1. A random variable X has ***normal distribution*** if $\Pr[X = x] = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)$.
2. $X \sim N(0, 1)$.
3. $\mathbf{x} = (x_1, \ldots, x_n)$ has d-dimensional normal distributed (i.e., $\mathbf{v} \sim N^n(0, 1)$
   $\iff \; v_1, \ldots, v_n \sim N(0, 1)$
4. $\mathbf{v} \in \mathbb{R}^n$, such that $\|\mathbf{v}\| = 1$.
5. Let $\mathbf{x} \sim N^n(0, 1)$. Then $z = \langle \mathbf{v}, \mathbf{x} \rangle$ has...
6. ...normal distribution!

Part II

Approximate Max Cut

The movie so far...

Summary: It sucks.

1. Seen: Examples of using rounding techniques for approximation.
2. So far: Relaxed optimization problem is LP.
3. But... We know how to solve ***convex programming***.
4. Convex programming $\gg$ LP.
5. Convex programming can be solved in polynomial time.
6. Solving convex programming is outside scope: assume doable in polynomial time.
7. Today's lecture:
    1. Revisit **MAX CUT**.
    2. Show how to relax it into semi-definite programming problem.
    3. Solve relaxation.
    4. Show how to round the relaxed problem.

The movie so far...

Summary: It sucks.

1. Seen: Examples of using rounding techniques for approximation.
2. So far: Relaxed optimization problem is LP.
3. But... We know how to solve ***convex programming***.
4. Convex programming $\gg$ LP.
5. Convex programming can be solved in polynomial time.
6. Solving convex programming is outside scope: assume doable in polynomial time.
7. Today's lecture:
    1. Revisit **MAX CUT**.
    2. Show how to relax it into semi-definite programming problem.
    3. Solve relaxation.
    4. Show how to round the relaxed problem.

The movie so far...

Summary: It sucks.

1. Seen: Examples of using rounding techniques for approximation.
2. So far: Relaxed optimization problem is LP.
3. But... We know how to solve ***convex programming***.
4. Convex programming $\gg$ LP.
5. Convex programming can be solved in polynomial time.
6. Solving convex programming is outside scope: assume doable in polynomial time.
7. Today's lecture:
   1. Revisit **MAX CUT**.
   2. Show how to relax it into semi-definite programming problem.
   3. Solve relaxation.
   4. Show how to round the relaxed problem.

The movie so far...

Summary: It sucks.

1. Seen: Examples of using rounding techniques for approximation.
2. So far: Relaxed optimization problem is LP.
3. But... We know how to solve ***convex programming***.
4. Convex programming $\gg$ LP.
5. Convex programming can be solved in polynomial time.
6. Solving convex programming is outside scope: assume doable in polynomial time.
7. Today's lecture:
    1. Revisit **MAX CUT**.
    2. Show how to relax it into semi-definite programming problem.
    3. Solve relaxation.
    4. Show how to round the relaxed problem.

The movie so far...

Summary: It sucks.

1. Seen: Examples of using rounding techniques for approximation.
2. So far: Relaxed optimization problem is LP.
3. But... We know how to solve ***convex programming***.
4. Convex programming $\gg$ LP.
5. Convex programming can be solved in polynomial time.
6. Solving convex programming is outside scope: assume doable in polynomial time.
7. Today's lecture:
    1. Revisit **MAX CUT**.
    2. Show how to relax it into semi-definite programming problem.
    3. Solve relaxation.
    4. Show how to round the relaxed problem.

The movie so far...

Summary: It sucks.

1. Seen: Examples of using rounding techniques for approximation.
2. So far: Relaxed optimization problem is LP.
3. But... We know how to solve ***convex programming***.
4. Convex programming $\gg$ LP.
5. Convex programming can be solved in polynomial time.
6. Solving convex programming is outside scope: assume doable in polynomial time.
7. Today's lecture:
    1. Revisit **MAX CUT**.
    2. Show how to relax it into semi-definite programming problem.
    3. Solve relaxation.
    4. Show how to round the relaxed problem.

The movie so far...

Summary: It sucks.

1. Seen: Examples of using rounding techniques for approximation.
2. So far: Relaxed optimization problem is LP.
3. But... We know how to solve ***convex programming***.
4. Convex programming $\gg$ LP.
5. Convex programming can be solved in polynomial time.
6. Solving convex programming is outside scope: assume doable in polynomial time.
7. Today's lecture:
    1. Revisit **MAX CUT**.
    2. Show how to relax it into semi-definite programming problem.
    3. Solve relaxation.
    4. Show how to round the relaxed problem.

The movie so far...

Summary: It sucks.

1. Seen: Examples of using rounding techniques for approximation.
2. So far: Relaxed optimization problem is LP.
3. But... We know how to solve ***convex programming***.
4. Convex programming $\gg$ LP.
5. Convex programming can be solved in polynomial time.
6. Solving convex programming is outside scope: assume doable in polynomial time.
7. Today's lecture:
    1. Revisit **MAX CUT**.
    2. Show how to relax it into semi-definite programming problem.
    3. Solve relaxation.
    4. Show how to round the relaxed problem.

The movie so far...

Summary: It sucks.

1. Seen: Examples of using rounding techniques for approximation.
2. So far: Relaxed optimization problem is LP.
3. But... We know how to solve ***convex programming***.
4. Convex programming $\gg$ LP.
5. Convex programming can be solved in polynomial time.
6. Solving convex programming is outside scope: assume doable in polynomial time.
7. Today's lecture:
    1. Revisit **MAX CUT**.
    2. Show how to relax it into semi-definite programming problem.
    3. Solve relaxation.
    4. Show how to round the relaxed problem.

The movie so far...

Summary: It sucks.

1. Seen: Examples of using rounding techniques for approximation.
2. So far: Relaxed optimization problem is LP.
3. But... We know how to solve ***convex programming***.
4. Convex programming $\gg$ LP.
5. Convex programming can be solved in polynomial time.
6. Solving convex programming is outside scope: assume doable in polynomial time.
7. Today's lecture:
    1. Revisit **MAX CUT**.
    2. Show how to relax it into semi-definite programming problem.
    3. Solve relaxation.
    4. Show how to round the relaxed problem.

The movie so far...

Summary: It sucks.

1. Seen: Examples of using rounding techniques for approximation.
2. So far: Relaxed optimization problem is LP.
3. But... We know how to solve ***convex programming***.
4. Convex programming $\gg$ LP.
5. Convex programming can be solved in polynomial time.
6. Solving convex programming is outside scope: assume doable in polynomial time.
7. Today's lecture:
    1. Revisit **MAX CUT**.
    2. Show how to relax it into semi-definite programming problem.
    3. Solve relaxation.
    4. Show how to round the relaxed problem.

Problem Statement: MAX CUT

Since this is a theory class, we will define our problem.

1. $\mathbf{G} = (\mathbf{V}, \mathbf{E})$: undirected graph.
2. $\forall ij \in \mathbf{E}$: nonnegative weights ω_{ij}.
3. **MAX CUT** (***maximum cut problem***): Compute set $S \subseteq \mathbf{V}$ maximizing weight of edges in cut $\left(S, \overline{S}\right)$.
4. $ij \notin \mathbf{E} \implies \omega_{ij} = 0$.
5. ***weight*** of cut: $w\left(S, \overline{S}\right) = \sum_{i \in S,\ j \in \overline{S}} \omega_{ij}$.
6. Known: problem is **NP-Complete**.
   Hard to approximate within a certain constant.

Problem Statement: MAX CUT

Since this is a theory class, we will define our problem.

1. $\mathbf{G} = (\mathbf{V}, \mathbf{E})$: undirected graph.
2. $\forall ij \in \mathbf{E}$: nonnegative weights ω_{ij}.
3. MAX CUT (***maximum cut problem***): Compute set $S \subseteq \mathbf{V}$ maximizing weight of edges in cut $\left(S, \overline{S}\right)$.
4. $ij \notin \mathbf{E} \implies \omega_{ij} = 0$.
5. ***weight*** of cut: $w\left(S, \overline{S}\right) = \sum_{i \in S,\ j \in \overline{S}} \omega_{ij}$.
6. Known: problem is **NP-Complete**.
   Hard to approximate within a certain constant.

Problem Statement: MAX CUT

Since this is a theory class, we will define our problem.

1. $\mathbf{G} = (\mathbf{V}, \mathbf{E})$: undirected graph.
2. $\forall ij \in \mathbf{E}$: nonnegative weights ω_{ij}.
3. **MAX CUT** (***maximum cut problem***): Compute set $S \subseteq \mathbf{V}$ maximizing weight of edges in cut $\left(S, \overline{S}\right)$.
4. $ij \notin \mathbf{E} \implies \omega_{ij} = 0$.
5. ***weight*** of cut: $w\left(S, \overline{S}\right) = \sum_{i \in S,\ j \in \overline{S}} \omega_{ij}$.
6. Known: problem is **NP-Complete**.
   Hard to approximate within a certain constant.

Problem Statement: MAX CUT

Since this is a theory class, we will define our problem.

1. $\mathbf{G} = (\mathbf{V}, \mathbf{E})$: undirected graph.
2. $\forall ij \in \mathbf{E}$: nonnegative weights ω_{ij}.
3. **MAX CUT** (***maximum cut problem***): Compute set $S \subseteq \mathbf{V}$ maximizing weight of edges in cut $\left(S, \overline{S}\right)$.
4. $ij \notin \mathbf{E} \implies \omega_{ij} = 0$.
5. ***weight*** of cut: $w\left(S, \overline{S}\right) = \sum_{i \in S,\ j \in \overline{S}} \omega_{ij}$.
6. Known: problem is **NP-Complete**.
   Hard to approximate within a certain constant.

Problem Statement: MAX CUT

Since this is a theory class, we will define our problem.

1. $\mathsf{G} = (\mathsf{V}, \mathsf{E})$: undirected graph.
2. $\forall ij \in \mathsf{E}$: nonnegative weights ω_{ij}.
3. **MAX CUT** (***maximum cut problem***): Compute set $S \subseteq \mathsf{V}$ maximizing weight of edges in cut $\left(S, \overline{S}\right)$.
4. $ij \notin \mathsf{E} \implies \omega_{ij} = 0$.
5. ***weight*** of cut: $w\left(S, \overline{S}\right) = \displaystyle\sum_{i \in S,\ j \in \overline{S}} \omega_{ij}$.
6. Known: problem is **NP-Complete**.
   Hard to approximate within a certain constant.

Problem Statement: MAX CUT

Since this is a theory class, we will define our problem.

1. $\mathbf{G} = (\mathbf{V}, \mathbf{E})$: undirected graph.
2. $\forall ij \in \mathbf{E}$: nonnegative weights ω_{ij}.
3. **MAX CUT** (***maximum cut problem***): Compute set $S \subseteq \mathbf{V}$ maximizing weight of edges in cut $\left(S, \overline{S}\right)$.
4. $ij \notin \mathbf{E} \implies \omega_{ij} = 0$.
5. ***weight*** of cut: $w\left(S, \overline{S}\right) = \sum_{i \in S,\ j \in \overline{S}} \omega_{ij}$.
6. Known: problem is **NP-Complete**.
   Hard to approximate within a certain constant.

Max cut as integer program

because what can go wrong?

1. Vertices: $\mathbf{V} = \{1, \ldots, n\}$.
2. ω_{ij}: non-negative weights on edges.
3. max cut $w(S, \overline{S})$ is computed by the integer quadratic program:

$$(Q) \qquad \max \quad \frac{1}{2}\sum_{i<j} \omega_{ij}(1 - y_i y_j)$$

$$\text{subject to:} \quad y_i \in \{-1, 1\} \qquad \forall i \in \mathbf{V}.$$

4. Set: $S = \left\{ i \;\middle|\; y_i = 1 \right\}$.
5. $\omega\left(S, \overline{S}\right) = \frac{1}{2}\sum_{i<j} \omega_{ij}(1 - y_i y_j)$.

Max cut as integer program

because what can go wrong?

1. Vertices: $\mathbf{V} = \{1, \ldots, n\}$.
2. ω_{ij}: non-negative weights on edges.
3. max cut $w(S, \overline{S})$ is computed by the integer quadratic program:

$$
\begin{aligned}
\text{(Q)} \qquad & \max \quad \frac{1}{2} \sum_{i<j} \omega_{ij}(1 - y_i y_j) \\
& \text{subject to:} \quad y_i \in \{-1, 1\} \qquad \forall i \in \mathbf{V}.
\end{aligned}
$$

4. Set: $S = \left\{ i \;\middle|\; y_i = 1 \right\}$.
5. $\omega\left(S, \overline{S}\right) = \frac{1}{2} \sum_{i<j} \omega_{ij}(1 - y_i y_j)$.

Max cut as integer program

because what can go wrong?

1. Vertices: $\mathbf{V} = \{1, \ldots, n\}$.
2. ω_{ij}: non-negative weights on edges.
3. max cut $w(S, \overline{S})$ is computed by the integer quadratic program:

$$
\begin{array}{llll}
(Q) & \max & \dfrac{1}{2}\sum_{i<j} \omega_{ij}(1 - y_i y_j) & \\
 & \text{subject to:} & y_i \in \{-1, 1\} & \forall i \in \mathbf{V}.
\end{array}
$$

4. Set: $S = \left\{ i \,\middle|\, y_i = 1 \right\}$.
5. $\omega\left(S, \overline{S}\right) = \frac{1}{2}\sum_{i<j} \omega_{ij}(1 - y_i y_j)$.

Max cut as integer program

because what can go wrong?

1. Vertices: $\mathbf{V} = \{1, \ldots, n\}$.
2. ω_{ij}: non-negative weights on edges.
3. max cut $w(S, \overline{S})$ is computed by the integer quadratic program:

$$(Q) \qquad \max \quad \frac{1}{2}\sum_{i<j} \omega_{ij}(1 - y_i y_j)$$
$$\text{subject to:} \quad y_i \in \{-1, 1\} \qquad \forall i \in \mathbf{V}.$$

4. Set: $S = \left\{ i \;\middle|\; y_i = 1 \right\}$.
5. $\omega\left(S, \overline{S}\right) = \frac{1}{2}\sum_{i<j} \omega_{ij}(1 - y_i y_j)$.

Max cut as integer program

because what can go wrong?

1. Vertices: $\mathsf{V} = \{1, \ldots, n\}$.
2. ω_{ij}: non-negative weights on edges.
3. max cut $w(S, \overline{S})$ is computed by the integer quadratic program:

$$
\begin{aligned}
\text{(Q)} \qquad & \max \quad \frac{1}{2} \sum_{i<j} \omega_{ij} (1 - y_i y_j) \\
& \text{subject to:} \quad y_i \in \{-1, 1\} \qquad \forall i \in \mathsf{V}.
\end{aligned}
$$

4. Set: $S = \left\{ i \;\middle|\; y_i = 1 \right\}$.
5. $\omega\left(S, \overline{S}\right) = \frac{1}{2} \sum_{i<j} \omega_{ij} (1 - y_i y_j)$.

Relaxing $-1, 1$...

Because 1 and -1 are just vectors.

1. Solving quadratic integer programming is of course **NP-Hard**.
2. Want a relaxation...
3. 1 and -1 are just roots of unity.
4. FFT: All roots of unity are a circle.
5. In higher dimensions: All unit vectors are points on unit sphere.
6. y_i are just unit vectors.
7. $y_i * y_j$ is replaced by dot product $\langle y_i, y_j \rangle$.

Relaxing $-1, 1$...

Because 1 and -1 are just vectors.

1. Solving quadratic integer programming is of course **NP-Hard**.
2. Want a relaxation...
3. 1 and -1 are just roots of unity.
4. FFT: All roots of unity are a circle.
5. In higher dimensions: All unit vectors are points on unit sphere.
6. y_i are just unit vectors.
7. $y_i * y_j$ is replaced by dot product $\langle y_i, y_j \rangle$.

Relaxing $-1, 1$...

Because 1 and -1 are just vectors.

1. Solving quadratic integer programming is of course **NP-Hard**.
2. Want a relaxation...
3. 1 and -1 are just roots of unity.
4. FFT: All roots of unity are a circle.
5. In higher dimensions: All unit vectors are points on unit sphere.
6. y_i are just unit vectors.
7. $y_i * y_j$ is replaced by dot product $\langle y_i, y_j \rangle$.

Relaxing $-1, 1$...

Because 1 and -1 are just vectors.

1. Solving quadratic integer programming is of course **NP-Hard**.
2. Want a relaxation...
3. 1 and -1 are just roots of unity.
4. FFT: All roots of unity are a circle.
5. In higher dimensions: All unit vectors are points on unit sphere.
6. y_i are just unit vectors.
7. $y_i * y_j$ is replaced by dot product $\langle y_i, y_j \rangle$.

Relaxing $-1, 1$...

Because 1 and -1 are just vectors.

1. Solving quadratic integer programming is of course **NP-Hard**.
2. Want a relaxation...
3. 1 and -1 are just roots of unity.
4. FFT: All roots of unity are a circle.
5. In higher dimensions: All unit vectors are points on unit sphere.
6. y_i are just unit vectors.
7. $y_i * y_j$ is replaced by dot product $\langle y_i, y_j \rangle$.

Relaxing $-1, 1$...

Because 1 and -1 are just vectors.

1. Solving quadratic integer programming is of course **NP-Hard**.
2. Want a relaxation...
3. 1 and -1 are just roots of unity.
4. FFT: All roots of unity are a circle.
5. In higher dimensions: All unit vectors are points on unit sphere.
6. y_i are just unit vectors.
7. $y_i * y_j$ is replaced by dot product $\langle y_i, y_j \rangle$.

Quick reminder about dot products

Everybody knows, thats how it goes

1. $\mathbf{x} = (x_1, \ldots, x_d)$, $\mathbf{y} = (y_1, \ldots, y_d)$.
2. $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{d} x_i y_i$.
3. For a vector $\mathbf{v} \in \mathbb{R}^d$: $\|\mathbf{v}\|^2 = \langle \mathbf{v}, \mathbf{v} \rangle$.
4. $\langle \mathbf{x}, \mathbf{y} \rangle = \|\mathbf{x}\| \|\mathbf{y}\| \cos \alpha$.
   α: Angle between $\mathbf{x}$ and $\mathbf{y}$.
5. $\mathbf{x} \perp \mathbf{y}$: $\langle \mathbf{x}, \mathbf{y} \rangle = 0$.
6. $\mathbf{x} = \mathbf{y}$ and $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$: $\langle \mathbf{x}, \mathbf{y} \rangle = 1$.
7. $\mathbf{x} = -\mathbf{y}$ and $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$: $\langle \mathbf{x}, \mathbf{y} \rangle = -1$.

Quick reminder about dot products

Everybody knows, thats how it goes

1. $\mathbf{x} = (x_1, \ldots, x_d)$, $\mathbf{y} = (y_1, \ldots, y_d)$.
2. $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{d} x_i y_i$.
3. For a vector $\mathbf{v} \in \mathbb{R}^d$: $\|\mathbf{v}\|^2 = \langle \mathbf{v}, \mathbf{v} \rangle$.
4. $\langle \mathbf{x}, \mathbf{y} \rangle = \|\mathbf{x}\| \, \|\mathbf{y}\| \cos \alpha$.
   α: Angle between $\mathbf{x}$ and $\mathbf{y}$.
5. $\mathbf{x} \perp \mathbf{y}$: $\langle \mathbf{x}, \mathbf{y} \rangle = 0$.
6. $\mathbf{x} = \mathbf{y}$ and $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$: $\langle \mathbf{x}, \mathbf{y} \rangle = 1$.
7. $\mathbf{x} = -\mathbf{y}$ and $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$: $\langle \mathbf{x}, \mathbf{y} \rangle = -1$.

Quick reminder about dot products

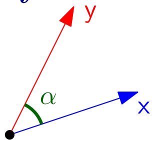
Everybody knows, thats how it goes

1. $\mathbf{x} = (x_1, \ldots, x_d)$, $\mathbf{y} = (y_1, \ldots, y_d)$.
2. $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{d} x_i y_i$.
3. For a vector $\mathbf{v} \in \mathbb{R}^d$: $\|\mathbf{v}\|^2 = \langle \mathbf{v}, \mathbf{v} \rangle$.
4. $\langle \mathbf{x}, \mathbf{y} \rangle = \|\mathbf{x}\| \|\mathbf{y}\| \cos \alpha$.
   α: Angle between $\mathbf{x}$ and $\mathbf{y}$.
5. $\mathbf{x} \perp \mathbf{y}$: $\langle \mathbf{x}, \mathbf{y} \rangle = 0$.
6. $\mathbf{x} = \mathbf{y}$ and $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$: $\langle \mathbf{x}, \mathbf{y} \rangle = 1$.
7. $\mathbf{x} = -\mathbf{y}$ and $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$: $\langle \mathbf{x}, \mathbf{y} \rangle = -1$.

Quick reminder about dot products

Everybody knows, thats how it goes

1. $\mathbf{x} = (x_1, \ldots, x_d)$, $\mathbf{y} = (y_1, \ldots, y_d)$.
2. $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{d} x_i y_i$.
3. For a vector $\mathbf{v} \in \mathbb{R}^d$: $\|\mathbf{v}\|^2 = \langle \mathbf{v}, \mathbf{v} \rangle$.
4. $\langle \mathbf{x}, \mathbf{y} \rangle = \|\mathbf{x}\| \, \|\mathbf{y}\| \cos \alpha$.
   α: Angle between $\mathbf{x}$ and $\mathbf{y}$.

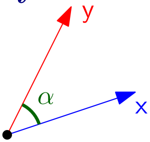


5. $\mathbf{x} \perp \mathbf{y}$: $\langle \mathbf{x}, \mathbf{y} \rangle = 0$.
6. $\mathbf{x} = \mathbf{y}$ and $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$: $\langle \mathbf{x}, \mathbf{y} \rangle = 1$.
7. $\mathbf{x} = -\mathbf{y}$ and $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$: $\langle \mathbf{x}, \mathbf{y} \rangle = -1$.

Quick reminder about dot products

Everybody knows, thats how it goes

1. $\mathbf{x} = (x_1, \ldots, x_d)$, $\mathbf{y} = (y_1, \ldots, y_d)$.
2. $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{d} x_i y_i$.
3. For a vector $\mathbf{v} \in \mathbb{R}^d$: $\|\mathbf{v}\|^2 = \langle \mathbf{v}, \mathbf{v} \rangle$.
4. $\langle \mathbf{x}, \mathbf{y} \rangle = \|\mathbf{x}\| \, \|\mathbf{y}\| \cos \alpha$.
   α: Angle between $\mathbf{x}$ and $\mathbf{y}$.

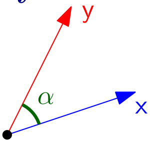


5. $\mathbf{x} \perp \mathbf{y}$: $\langle \mathbf{x}, \mathbf{y} \rangle = 0$.
6. $\mathbf{x} = \mathbf{y}$ and $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$: $\langle \mathbf{x}, \mathbf{y} \rangle = 1$.
7. $\mathbf{x} = -\mathbf{y}$ and $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$: $\langle \mathbf{x}, \mathbf{y} \rangle = -1$.

Quick reminder about dot products

Everybody knows, thats how it goes

1. $\mathbf{x} = (x_1, \ldots, x_d)$, $\mathbf{y} = (y_1, \ldots, y_d)$.
2. $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{d} x_i y_i$.
3. For a vector $\mathbf{v} \in \mathbb{R}^d$: $\|\mathbf{v}\|^2 = \langle \mathbf{v}, \mathbf{v} \rangle$.
4. $\langle \mathbf{x}, \mathbf{y} \rangle = \|\mathbf{x}\| \, \|\mathbf{y}\| \cos \alpha$.
   α: Angle between $\mathbf{x}$ and $\mathbf{y}$.

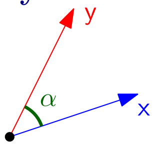


5. $\mathbf{x} \perp \mathbf{y}$: $\langle \mathbf{x}, \mathbf{y} \rangle = 0$.
6. $\mathbf{x} = \mathbf{y}$ and $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$: $\langle \mathbf{x}, \mathbf{y} \rangle = 1$.
7. $\mathbf{x} = -\mathbf{y}$ and $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$: $\langle \mathbf{x}, \mathbf{y} \rangle = -1$.

Quick reminder about dot products

Everybody knows, thats how it goes

1. $\mathbf{x} = (x_1, \ldots, x_d)$, $\mathbf{y} = (y_1, \ldots, y_d)$.
2. $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{d} x_i y_i$.
3. For a vector $\mathbf{v} \in \mathbb{R}^d$: $\|\mathbf{v}\|^2 = \langle \mathbf{v}, \mathbf{v} \rangle$.
4. $\langle \mathbf{x}, \mathbf{y} \rangle = \|\mathbf{x}\| \, \|\mathbf{y}\| \cos \alpha$.
   α: Angle between $\mathbf{x}$ and $\mathbf{y}$.



5. $\mathbf{x} \perp \mathbf{y}$: $\langle \mathbf{x}, \mathbf{y} \rangle = 0$.
6. $\mathbf{x} = \mathbf{y}$ and $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$: $\langle \mathbf{x}, \mathbf{y} \rangle = 1$.
7. $\mathbf{x} = -\mathbf{y}$ and $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$: $\langle \mathbf{x}, \mathbf{y} \rangle = -1$.

Relaxing $-1, 1$...

Because 1 and -1 are just vectors.

1. max cut $w(S, \overline{S})$ as integer quadratic program:

$$
\begin{aligned}
(\mathrm{Q}) \qquad & \max \quad \frac{1}{2} \sum_{i<j} \omega_{ij}(1 - y_i y_j) \\
& \text{subject to:} \quad y_i \in \{-1, 1\} \qquad \forall i \in \mathsf{V}.
\end{aligned}
$$

2. Relaxed semi-definite programming version:

$$
\begin{aligned}
(\mathrm{P}) \qquad & \max \quad \gamma = \frac{1}{2} \sum_{i<j} \omega_{ij} \left(1 - \langle v_i, v_j \rangle\right) \\
& \text{subject to:} \quad v_i \in \mathbb{S}^{(n)} \qquad \forall i \in V,
\end{aligned}
$$

$\mathbb{S}^{(n)}$: n dimensional unit sphere in $\mathbb{R}^{n+1}$.

Relaxing $-1, 1$...

Because 1 and -1 are just vectors.

1. max cut $w(S, \overline{S})$ as integer quadratic program:

$$(Q) \qquad \max \quad \frac{1}{2}\sum_{i<j} \omega_{ij}(1 - y_i y_j)$$

$$\text{subject to:} \quad y_i \in \{-1, 1\} \qquad \forall i \in \mathsf{V}.$$

2. Relaxed semi-definite programming version:

$$(P) \qquad \max \quad \gamma = \frac{1}{2}\sum_{i<j} \omega_{ij}\left(1 - \langle v_i, v_j \rangle\right)$$

$$\text{subject to:} \quad v_i \in \mathbb{S}^{(n)} \qquad \forall i \in V,$$

$\mathbb{S}^{(n)}$: n dimensional unit sphere in $\mathbb{R}^{n+1}$.

Discussion...

1. semi-definite programming: special case of convex programming.
2. Can be solved in polynomial time.
3. Solve within a factor of $(1 + \varepsilon)$ of optimal, for any $\varepsilon > 0$, in polynomial time.
4. Intuition: vectors of one side of the cut, and vertices on the other sides, would have faraway vectors.

Discussion...

1. semi-definite programming: special case of convex programming.
2. Can be solved in polynomial time.
3. Solve within a factor of $(1+\varepsilon)$ of optimal, for any $\varepsilon > 0$, in polynomial time.
4. Intuition: vectors of one side of the cut, and vertices on the other sides, would have faraway vectors.

Discussion...

1. semi-definite programming: special case of convex programming.
2. Can be solved in polynomial time.
3. Solve within a factor of $(1+\varepsilon)$ of optimal, for any $\varepsilon > 0$, in polynomial time.
4. Intuition: vectors of one side of the cut, and vertices on the other sides, would have faraway vectors.

Discussion...

1. semi-definite programming: special case of convex programming.
2. Can be solved in polynomial time.
3. Solve within a factor of $(1+\varepsilon)$ of optimal, for any $\varepsilon > 0$, in polynomial time.
4. Intuition: vectors of one side of the cut, and vertices on the other sides, would have faraway vectors.

1. Given instance, compute Semi-definite program (P).
2. Compute optimal solution for (P).
3. $\vec{r}$: Pick random vector on the unit sphere $\mathbb{S}^{(n)}$.
4. induces hyperplane $h \equiv \langle \vec{r}, \mathrm{x} \rangle = 0$
5. assign all vectors on one side of h to S, and rest to $\overline{S}$.

$$S = \left\{ v_i \;\middle|\; \langle v_i, \vec{r} \rangle \geq 0 \right\}.$$

The approximation algorithm

For max cut

1. Given instance, compute Semi-definite program (P).
2. Compute optimal solution for (P).
3. $\vec{r}$: Pick random vector on the unit sphere $\mathbb{S}^{(n)}$.
4. induces hyperplane $h \quad \equiv \quad \langle \vec{r}, \mathrm{x} \rangle = 0$
5. assign all vectors on one side of h to S, and rest to $\overline{S}$.

$$S = \left\{ v_i \;\middle|\; \langle v_i, \vec{r} \rangle \geq 0 \right\}.$$

The approximation algorithm

For max cut

1. Given instance, compute Semi-definite program (P).
2. Compute optimal solution for (P).
3. $\vec{r}$: Pick random vector on the unit sphere $\mathbb{S}^{(n)}$.
4. induces hyperplane $h \equiv \langle \vec{r}, \mathrm{x} \rangle = 0$
5. assign all vectors on one side of h to S, and rest to $\overline{S}$.

$$S = \left\{ v_i \;\middle|\; \langle v_i, \vec{r} \rangle \geq 0 \right\}.$$

The approximation algorithm

For max cut

1. Given instance, compute Semi-definite program (P).
2. Compute optimal solution for (P).
3. $\vec{r}$: Pick random vector on the unit sphere $\mathbb{S}^{(n)}$.
4. induces hyperplane $h \equiv \langle \vec{r}, \mathrm{x} \rangle = 0$
5. assign all vectors on one side of h to S, and rest to $\overline{S}$.

$$S = \left\{ v_i \;\middle|\; \langle v_i, \vec{r} \rangle \geq 0 \right\}.$$

The approximation algorithm

For max cut

1. Given instance, compute Semi-definite program (P).
2. Compute optimal solution for (P).
3. $\vec{r}$: Pick random vector on the unit sphere $\mathbb{S}^{(n)}$.
4. induces hyperplane $h \equiv \langle \vec{r}, \mathrm{x} \rangle = 0$
5. assign all vectors on one side of h to S, and rest to $\overline{S}$.

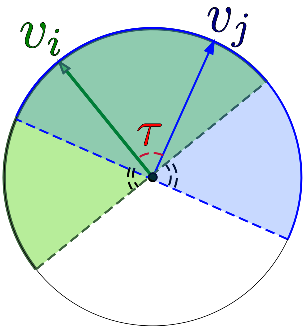
$$S = \left\{ v_i \;\middle|\; \langle v_i, \vec{r} \rangle \geq 0 \right\}.$$

Analysis...

Intuition: with good probability, vectors in the solution of (P) that have large angle between them would be separated by cut.

Lemma

$$\Pr\Big[\mathrm{sign}\Big(\langle v_i, \vec{r}\,\rangle\Big) \neq \mathrm{sign}(\langle v_j, \vec{r}\,\rangle)\Big] = \frac{1}{\pi}\arccos\Big(\langle v_i, v_j\rangle\Big) = \frac{\tau}{\pi}.$$



Proof...

1. Think v_i, v_j and $\vec{r}$ as being in the plane.
2. ... reasonable assumption!
   1. g: plane spanned by v_i and v_j.
   2. Only care about signs of $\langle v_i, \vec{r} \rangle$ and $\langle v_j, \vec{r} \rangle$
   3. can be decided by projecting $\vec{r}$ on g... and normalizing it to have length 1.
   4. Sphere is symmetric $\implies$ sampling $\vec{r}$ from $\mathbb{S}^{(n)}$ projecting it down to g, and then normalizing it
      $\equiv$ choosing uniformly a vector from the unit circle in g

Proof...

1. Think v_i, v_j and $\vec{r}$ as being in the plane.
2. ... reasonable assumption!
    1. g: plane spanned by v_i and v_j.
    2. Only care about signs of $\langle v_i, \vec{r} \rangle$ and $\langle v_j, \vec{r} \rangle$
    3. can be decided by projecting $\vec{r}$ on g... and normalizing it to have length 1.
    4. Sphere is symmetric $\implies$ sampling $\vec{r}$ from $\mathbb{S}^{(n)}$ projecting it down to g, and then normalizing it
    $\equiv$ choosing uniformly a vector from the unit circle in g

Proof...

1. Think v_i, v_j and $\vec{r}$ as being in the plane.
2. ... reasonable assumption!
   1. g: plane spanned by v_i and v_j.
   2. Only care about signs of $\langle v_i, \vec{r} \rangle$ and $\langle v_j, \vec{r} \rangle$
   3. can be decided by projecting $\vec{r}$ on g... and normalizing it to have length 1.
   4. Sphere is symmetric $\implies$ sampling $\vec{r}$ from $\mathbb{S}^{(n)}$ projecting it down to g, and then normalizing it
      $\equiv$ choosing uniformly a vector from the unit circle in g

Proof...

1. Think v_i, v_j and $\vec{r}$ as being in the plane.
2. ... reasonable assumption!
   1. g: plane spanned by v_i and v_j.
   2. Only care about signs of $\langle v_i, \vec{r} \rangle$ and $\langle v_j, \vec{r} \rangle$
   3. can be decided by projecting $\vec{r}$ on g... and normalizing it to have length 1.
   4. Sphere is symmetric $\implies$ sampling $\vec{r}$ from $\mathbb{S}^{(n)}$ projecting it down to g, and then normalizing it
      $\equiv$ choosing uniformly a vector from the unit circle in g

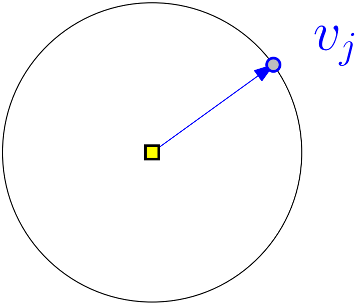
Proof...

1. Think v_i, v_j and $\vec{r}$ as being in the plane.
2. ... reasonable assumption!
    1. g: plane spanned by v_i and v_j.
    2. Only care about signs of $\langle v_i, \vec{r} \rangle$ and $\langle v_j, \vec{r} \rangle$
    3. can be decided by projecting $\vec{r}$ on g... and normalizing it to have length 1.
    4. Sphere is symmetric $\implies$ sampling $\vec{r}$ from $\mathbb{S}^{(n)}$ projecting it down to g, and then normalizing it
    $\equiv$ choosing uniformly a vector from the unit circle in g

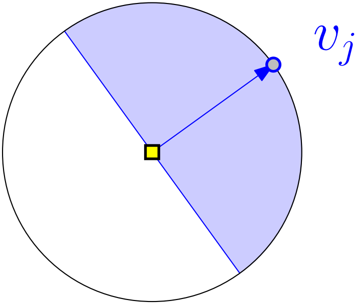
Proof...

1. Think v_i, v_j and $\vec{r}$ as being in the plane.
2. ... reasonable assumption!
   1. g: plane spanned by v_i and v_j.
   2. Only care about signs of $\langle v_i, \vec{r} \rangle$ and $\langle v_j, \vec{r} \rangle$
   3. can be decided by projecting $\vec{r}$ on g... and normalizing it to have length 1.
   4. Sphere is symmetric $\implies$ sampling $\vec{r}$ from $\mathbb{S}^{(n)}$ projecting it down to g, and then normalizing it
      $\equiv$ choosing uniformly a vector from the unit circle in g

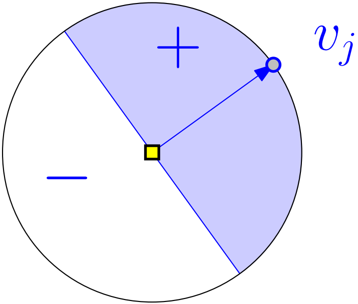
Proof via figure...



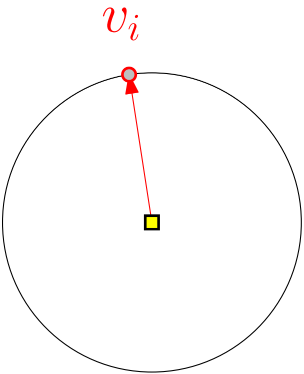
Proof via figure...



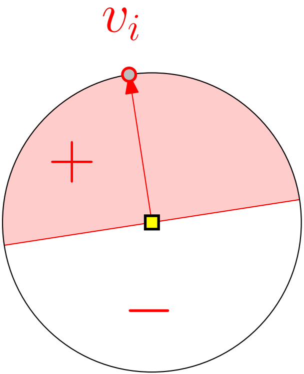
Proof via figure...



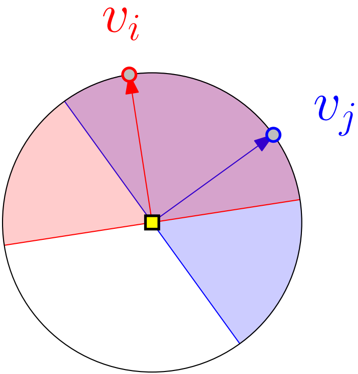
Proof via figure...



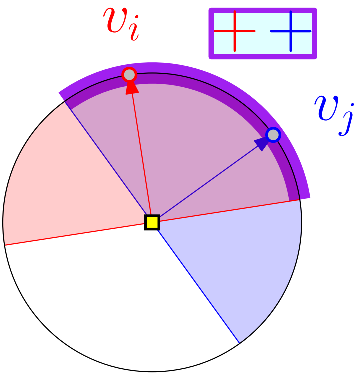
Proof via figure...



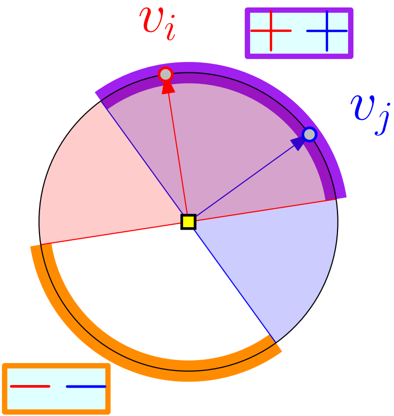
Proof via figure...



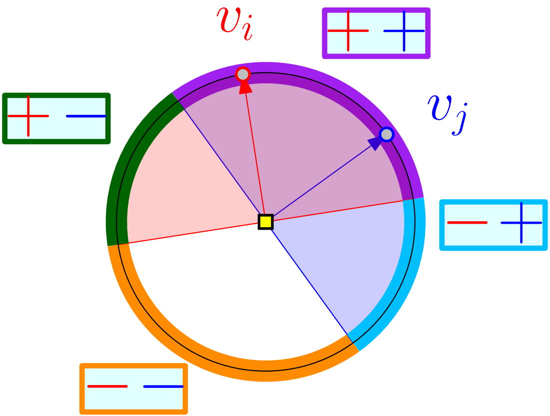
Proof via figure...



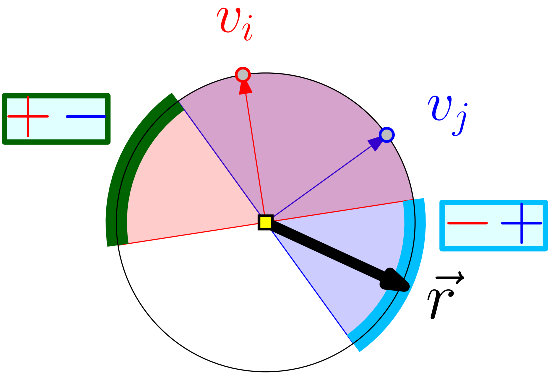
Proof via figure...



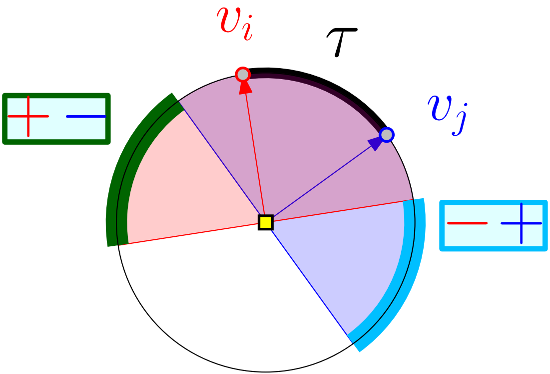
Proof via figure...



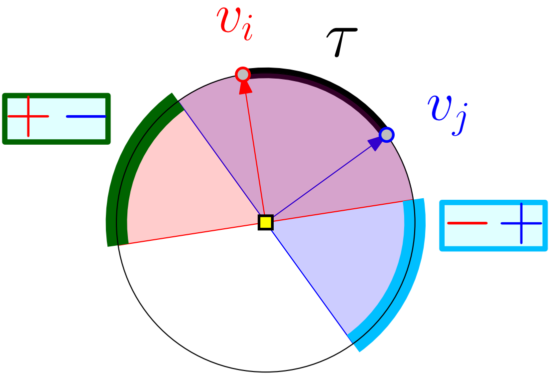
Proof via figure...



Proof via figure...



Proof via figure...



$\boldsymbol{\tau} = \mathbf{arccos}\Big(\langle v_i, v_j\rangle\Big)$

Proof...

1. Think v_i, v_j and $\vec{r}$ as being in the plane.
2. $\mathrm{sign}(\langle v_i, \vec{r}\,\rangle) \neq \mathrm{sign}(\langle v_j, \vec{r}\,\rangle)$ happens only if $\vec{r}$ falls in the double wedge formed by the lines perpendicular to v_i and v_j.
3. angle of double wedge $=$ angle τ between v_i and v_j.
4. v_i and v_j are unit vectors: $\langle v_i, v_j\rangle = \cos(\tau)$.
   $\tau = \angle v_i v_j$.
5. Thus,

$$\Pr\Big[\mathrm{sign}(\langle v_i, \vec{r}\,\rangle) \neq \mathrm{sign}(\langle v_j, \vec{r}\,\rangle)\Big] = \frac{2\tau}{2\pi} = \frac{1}{\pi} \cdot \arccos(\langle v_i, v_j\rangle)\,,$$

as claimed. ∎

Theorem

Theorem

Let W *be the random variable which is the weight of the cut generated by the algorithm. We have*

$$\mathbf{E}\Big[W\Big] = \frac{1}{\pi}\sum_{i<j}\omega_{ij}\arccos\Big(\langle v_i, v_j\rangle\Big).$$

Proof

1. X_{ij}: indicator variable $= 1 \iff$ edge ij is in the cut.
2. $\mathbf{E}[X_{ij}] = \Pr\left[\operatorname{sign}(\langle v_i, \vec{r}\,\rangle) \neq \operatorname{sign}(\langle v_j, \vec{r}\,\rangle)\right]$
   $= \frac{1}{\pi}\arccos\Big(\langle v_i, v_j\rangle\Big)$, by lemma.
3. $W = \sum_{i<j} \omega_{ij} X_{ij}$, and by linearity of expectation...

$$\mathbf{E}[W] = \sum_{i<j} \omega_{ij}\,\mathbf{E}[X_{ij}] = \frac{1}{\pi}\sum_{i<j} \omega_{ij} \arccos\Big(\langle v_i, v_j\rangle\Big).$$

Proof

1. X_{ij}: indicator variable $= 1 \iff$ edge ij is in the cut.
2. $\mathbf{E}[X_{ij}] = \mathbf{Pr}\Big[\mathrm{sign}(\langle v_i, \vec{r}\,\rangle) \neq \mathrm{sign}(\langle v_j, \vec{r}\,\rangle)\Big]$
   $= \frac{1}{\pi}\arccos\Big(\langle v_i, v_j\rangle\Big)$, by lemma.
3. $W = \sum_{i<j} \omega_{ij} X_{ij}$, and by linearity of expectation...

$$\mathbf{E}[W] = \sum_{i<j} \omega_{ij}\,\mathbf{E}[X_{ij}] = \frac{1}{\pi}\sum_{i<j} \omega_{ij} \arccos\Big(\langle v_i, v_j\rangle\Big).$$

$\blacksquare$

Proof

1. X_{ij}: indicator variable $= 1 \iff$ edge ij is in the cut.
2. $\mathbf{E}[X_{ij}] = \mathbf{Pr}\Big[\mathbf{sign}(\langle v_i, \vec{r}\,\rangle) \neq \mathbf{sign}(\langle v_j, \vec{r}\,\rangle)\Big]$
   $= \frac{1}{\pi}\arccos\Big(\langle v_i, v_j\rangle\Big)$, by lemma.
3. $W = \sum_{i<j} \omega_{ij} X_{ij}$, and by linearity of expectation...

$$\mathbf{E}[W] = \sum_{i<j} \omega_{ij}\,\mathbf{E}[X_{ij}] = \frac{1}{\pi}\sum_{i<j} \omega_{ij} \arccos\Big(\langle v_i, v_j\rangle\Big).$$

Proof

1. X_{ij}: indicator variable $= 1 \iff$ edge ij is in the cut.
2. $\mathbf{E}[X_{ij}] = \Pr\Big[\mathrm{sign}(\langle v_i, \vec{r}\,\rangle) \neq \mathrm{sign}(\langle v_j, \vec{r}\,\rangle)\Big]$
   $= \frac{1}{\pi}\arccos\Big(\langle v_i, v_j\rangle\Big)$, by lemma.
3. $W = \sum_{i<j} \omega_{ij} X_{ij}$, and by linearity of expectation...

$$\mathbf{E}[W] = \sum_{i<j} \omega_{ij}\, \mathbf{E}[X_{ij}] = \frac{1}{\pi} \sum_{i<j} \omega_{ij} \arccos\Big(\langle v_i, v_j\rangle\Big).$$

Proof

1. X_{ij}: indicator variable $= 1 \iff$ edge ij is in the cut.
2. $\mathbf{E}[X_{ij}] = \Pr\Big[\mathrm{sign}(\langle v_i, \vec{r}\,\rangle) \neq \mathrm{sign}(\langle v_j, \vec{r}\,\rangle)\Big]$
   $= \frac{1}{\pi}\arccos\Big(\langle v_i, v_j\rangle\Big)$, by lemma.
3. $W = \sum_{i<j} \omega_{ij} X_{ij}$, and by linearity of expectation...

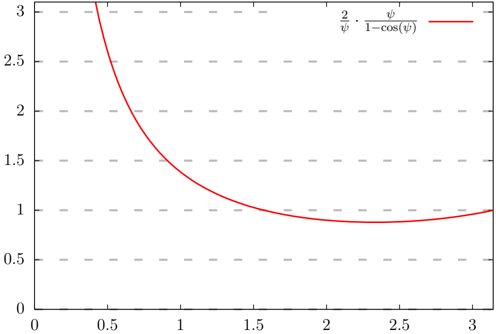
$$\mathbf{E}[W] = \sum_{i<j} \omega_{ij}\,\mathbf{E}[X_{ij}] = \frac{1}{\pi}\sum_{i<j}\omega_{ij}\arccos\Big(\langle v_i, v_j\rangle\Big).$$

■

Lemma

Lemma

For $-1 \leq y \leq 1$, *we have* $\dfrac{\arccos(y)}{\pi} \geq \alpha \cdot \dfrac{1}{2}(1 - y)$, *where*
$$\alpha = \min_{0 \leq \psi \leq \pi} \frac{2}{\pi} \frac{\psi}{1 - \cos(\psi)}.$$



Lemma restated + proof

Lemma

For $-1 \leq y \leq 1$, *we have* $\dfrac{\arccos(y)}{\pi} \geq \alpha \cdot \dfrac{1}{2}(1 - y)$, *where* $\alpha = \min_{0 \leq \psi \leq \pi} \dfrac{2}{\pi} \dfrac{\psi}{1 - \cos(\psi)}$.

Proof.

1. $y = \cos(\psi)$.
2. Inequality becomes: $\frac{\psi}{\pi} \geq \alpha \frac{1}{2}(1 - \cos\psi)$. Reorganizing,
3. $\implies \frac{2}{\pi} \frac{\psi}{1 - \cos\psi} \geq \alpha$, holds by definition of α.

Lemma restated + proof

Lemma

For $-1 \leq y \leq 1$, *we have* $\dfrac{\arccos(y)}{\pi} \geq \alpha \cdot \dfrac{1}{2}(1-y)$, *where* $\alpha = \displaystyle\min_{0 \leq \psi \leq \pi} \frac{2}{\pi}\frac{\psi}{1-\cos(\psi)}$.

Proof.

1. $y = \cos(\psi)$.
2. Inequality becomes: $\frac{\psi}{\pi} \geq \alpha \frac{1}{2}(1 - \cos\psi)$. Reorganizing,
3. $\implies \frac{2}{\pi}\frac{\psi}{1-\cos\psi} \geq \alpha$, holds by definition of α.

Lemma restated + proof

Lemma

For $-1 \leq y \leq 1$, *we have* $\dfrac{\arccos(y)}{\pi} \geq \alpha \cdot \dfrac{1}{2}(1 - y)$, *where*
$\alpha = \min\limits_{0 \leq \psi \leq \pi} \dfrac{2}{\pi} \dfrac{\psi}{1 - \cos(\psi)}$.

Proof.

1. $y = \cos(\psi)$.
2. Inequality becomes: $\frac{\psi}{\pi} \geq \alpha \frac{1}{2}(1 - \cos\psi)$. Reorganizing,
3. $\implies \frac{2}{\pi} \frac{\psi}{1 - \cos\psi} \geq \alpha$, holds by definition of α.

Lemma restated + proof

Lemma

For $-1 \leq y \leq 1$, *we have* $\dfrac{\arccos(y)}{\pi} \geq \alpha \cdot \dfrac{1}{2}(1 - y)$, *where* $\alpha = \displaystyle\min_{0 \leq \psi \leq \pi} \frac{2}{\pi} \frac{\psi}{1 - \cos(\psi)}$.

Proof.

1. $y = \cos(\psi)$.
2. Inequality becomes: $\frac{\psi}{\pi} \geq \alpha \frac{1}{2}(1 - \cos\psi)$. Reorganizing,
3. $\implies \frac{2}{\pi} \frac{\psi}{1 - \cos\psi} \geq \alpha$, holds by definition of α.

□

Lemma restated + proof

Lemma

For $-1 \leq y \leq 1$, *we have* $\dfrac{\arccos(y)}{\pi} \geq \alpha \cdot \dfrac{1}{2}(1-y)$, *where* $\alpha = \displaystyle\min_{0 \leq \psi \leq \pi} \frac{2}{\pi} \frac{\psi}{1-\cos(\psi)}$.

Proof.

1. $y = \cos(\psi)$.
2. Inequality becomes: $\frac{\psi}{\pi} \geq \alpha \frac{1}{2}(1 - \cos\psi)$. Reorganizing,
3. $\implies \frac{2}{\pi} \frac{\psi}{1-\cos\psi} \geq \alpha$, holds by definition of α.

□

Lemma

Lemma

$\alpha > 0.87856$.

Proof.

Using simple calculus, one can see that α achieves its value for $\psi = 2.331122...$, the nonzero root of $\cos\psi + \psi\sin\psi = 1$. □

Result

Theorem

The above algorithm computes in expectation a cut with total weight $\alpha \cdot \mathbf{Opt} \geq \mathbf{0.87856Opt}$, *where* $\mathbf{Opt}$ *is the weight of the maximal cut.*

Proof.

Consider the optimal solution to (P), and lets its value be $\gamma \geq \mathbf{Opt}$. By lemma:

$$\mathbf{E}[W] = \frac{1}{\pi} \sum_{i<j} \omega_{ij} \arccos(\langle v_i, v_j \rangle)$$

$$\geq \sum_{i<j} \omega_{ij} \alpha \frac{1}{2}(1 - \langle v_i, v_j \rangle) = \alpha\gamma \geq \alpha \cdot \mathbf{Opt}. \quad \blacksquare$$

$\square$

SDP: Semi-definite programming

1. $x_{ij} = \langle v_i, v_j \rangle$.
2. M: $n \times n$ matrix with x_{ij} as entries.
3. $x_{ii} = 1$, for $i = 1, \ldots, n$.
4. V: matrix having vectors $v_1, \ldots, v_n$ as its columns.
5. $M = V^T V$.
6. $\forall v \in \mathbb{R}^n$: $v^T M v = v^T A^T A v = (Av)^T (Av) \geq 0$.
7. M is ***positive semidefinite*** (PSD).
8. Fact: Any PSD matrix P can be written as $P = B^T B$.
9. Furthermore, given such a matrix P of size $n \times n$, we can compute B such that $P = B^T B$ in $O(n^3)$ time.
10. Known as ***Cholesky decomposition***.

SDP: Semi-definite programming

1. $x_{ij} = \langle v_i, v_j \rangle$.
2. M: $n \times n$ matrix with x_{ij} as entries.
3. $x_{ii} = 1$, for $i = 1, \ldots, n$.
4. V: matrix having vectors $v_1, \ldots, v_n$ as its columns.
5. $M = V^T V$.
6. $\forall v \in \mathbb{R}^n$: $v^T M v = v^T A^T A v = (Av)^T (Av) \geq 0$.
7. M is ***positive semidefinite*** (PSD).
8. Fact: Any PSD matrix P can be written as $P = B^T B$.
9. Furthermore, given such a matrix P of size $n \times n$, we can compute B such that $P = B^T B$ in $O(n^3)$ time.
10. Known as ***Cholesky decomposition***.

SDP: Semi-definite programming

1. $x_{ij} = \langle v_i, v_j \rangle$.
2. M: $n \times n$ matrix with x_{ij} as entries.
3. $x_{ii} = 1$, for $i = 1, \ldots, n$.
4. V: matrix having vectors $v_1, \ldots, v_n$ as its columns.
5. $M = V^T V$.
6. $\forall v \in \mathbb{R}^n$: $v^T M v = v^T A^T A v = (Av)^T (Av) \geq 0$.
7. M is ***positive semidefinite*** (PSD).
8. Fact: Any PSD matrix P can be written as $P = B^T B$.
9. Furthermore, given such a matrix P of size $n \times n$, we can compute B such that $P = B^T B$ in $O(n^3)$ time.
10. Known as ***Cholesky decomposition***.

SDP: Semi-definite programming

1. $x_{ij} = \langle v_i, v_j \rangle$.
2. M: $n \times n$ matrix with x_{ij} as entries.
3. $x_{ii} = 1$, for $i = 1, \ldots, n$.
4. V: matrix having vectors $v_1, \ldots, v_n$ as its columns.
5. $M = V^T V$.
6. $\forall v \in \mathbb{R}^n$: $v^T M v = v^T A^T A v = (Av)^T (Av) \geq 0$.
7. M is ***positive semidefinite*** (PSD).
8. Fact: Any PSD matrix P can be written as $P = B^T B$.
9. Furthermore, given such a matrix P of size $n \times n$, we can compute B such that $P = B^T B$ in $O(n^3)$ time.
10. Known as ***Cholesky decomposition***.

SDP: Semi-definite programming

1. $x_{ij} = \langle v_i, v_j \rangle$.
2. M: $n \times n$ matrix with x_{ij} as entries.
3. $x_{ii} = 1$, for $i = 1, \ldots, n$.
4. V: matrix having vectors $v_1, \ldots, v_n$ as its columns.
5. $M = V^T V$.
6. $\forall v \in \mathbb{R}^n$: $v^T M v = v^T A^T A v = (Av)^T (Av) \geq 0$.
7. M is ***positive semidefinite*** (PSD).
8. Fact: Any PSD matrix P can be written as $P = B^T B$.
9. Furthermore, given such a matrix P of size $n \times n$, we can compute B such that $P = B^T B$ in $O(n^3)$ time.
10. Known as ***Cholesky decomposition***.

SDP: Semi-definite programming

1. $x_{ij} = \langle v_i, v_j \rangle$.
2. M: $n \times n$ matrix with x_{ij} as entries.
3. $x_{ii} = 1$, for $i = 1, \ldots, n$.
4. V: matrix having vectors $v_1, \ldots, v_n$ as its columns.
5. $M = V^T V$.
6. $\forall v \in \mathbb{R}^n$: $v^T M v = v^T A^T A v = (Av)^T (Av) \geq 0$.
7. M is ***positive semidefinite*** (PSD).
8. Fact: Any PSD matrix P can be written as $P = B^T B$.
9. Furthermore, given such a matrix P of size $n \times n$, we can compute B such that $P = B^T B$ in $O(n^3)$ time.
10. Known as ***Cholesky decomposition***.

SDP: Semi-definite programming

1. $x_{ij} = \langle v_i, v_j \rangle$.
2. M: $n \times n$ matrix with x_{ij} as entries.
3. $x_{ii} = 1$, for $i = 1, \ldots, n$.
4. V: matrix having vectors $v_1, \ldots, v_n$ as its columns.
5. $M = V^T V$.
6. $\forall v \in \mathbb{R}^n$: $v^T M v = v^T A^T A v = (Av)^T (Av) \geq 0$.
7. M is ***positive semidefinite*** (PSD).
8. Fact: Any PSD matrix P can be written as $P = B^T B$.
9. Furthermore, given such a matrix P of size $n \times n$, we can compute B such that $P = B^T B$ in $O(n^3)$ time.
10. Known as ***Cholesky decomposition***.

SDP: Semi-definite programming

1. $x_{ij} = \langle v_i, v_j \rangle$.
2. M: $n \times n$ matrix with x_{ij} as entries.
3. $x_{ii} = 1$, for $i = 1, \ldots, n$.
4. V: matrix having vectors $v_1, \ldots, v_n$ as its columns.
5. $M = V^T V$.
6. $\forall v \in \mathbb{R}^n$: $v^T M v = v^T A^T A v = (Av)^T (Av) \geq 0$.
7. M is ***positive semidefinite*** (PSD).
8. Fact: Any PSD matrix P can be written as $P = B^T B$.
9. Furthermore, given such a matrix P of size $n \times n$, we can compute B such that $P = B^T B$ in $O(n^3)$ time.
10. Known as ***Cholesky decomposition***.

SDP: Semi-definite programming

1. $x_{ij} = \langle v_i, v_j \rangle$.
2. M: $n \times n$ matrix with x_{ij} as entries.
3. $x_{ii} = 1$, for $i = 1, \ldots, n$.
4. V: matrix having vectors $v_1, \ldots, v_n$ as its columns.
5. $M = V^T V$.
6. $\forall v \in \mathbb{R}^n$: $v^T M v = v^T A^T A v = (Av)^T (Av) \geq 0$.
7. M is ***positive semidefinite*** (PSD).
8. Fact: Any PSD matrix P can be written as $P = B^T B$.
9. Furthermore, given such a matrix P of size $n \times n$, we can compute B such that $P = B^T B$ in $O(n^3)$ time.
10. Known as ***Cholesky decomposition***.

SDP: Semi-definite programming

1. $x_{ij} = \langle v_i, v_j \rangle$.
2. M: $n \times n$ matrix with x_{ij} as entries.
3. $x_{ii} = 1$, for $i = 1, \ldots, n$.
4. V: matrix having vectors $v_1, \ldots, v_n$ as its columns.
5. $M = V^T V$.
6. $\forall v \in \mathbb{R}^n$: $v^T M v = v^T A^T A v = (Av)^T (Av) \geq 0$.
7. M is ***positive semidefinite*** (PSD).
8. Fact: Any PSD matrix P can be written as $P = B^T B$.
9. Furthermore, given such a matrix P of size $n \times n$, we can compute B such that $P = B^T B$ in $O(n^3)$ time.
10. Known as ***Cholesky decomposition***.

SDP: Semi-definite programming

1. If PSD $P = B^T B$ has a diagonal of 1
2. $\implies$ B has columns which are unit vectors.
3. If solve SDP (P), get back semi-definite matrix...
4. ... recover the vectors realizing the solution (i.e., compute B)
5. Now, do the rounding.
6. SDP (P) can be restated as

$$
\begin{aligned}
(SD) \qquad & \max && \frac{1}{2}\sum_{i<j}\omega_{ij}(1 - x_{ij}) \\
& \text{subject to:} && x_{ii} = 1 \qquad \text{for } i = 1, \ldots, n \\
& && \Bigl(x_{ij}\Bigr)_{i=1,\ldots,n,j=1,\ldots,n} \qquad \text{is a PSD matrix.}
\end{aligned}
$$

SDP: Semi-definite programming

1. If PSD $P = B^T B$ has a diagonal of 1
2. $\implies$ B has columns which are unit vectors.
3. If solve SDP (P), get back semi-definite matrix...
4. ... recover the vectors realizing the solution (i.e., compute B)
5. Now, do the rounding.
6. SDP (P) can be restated as

$$
\begin{aligned}
(SD) \qquad & \max && \frac{1}{2}\sum_{i<j}\omega_{ij}(1 - x_{ij}) \\
& \text{subject to:} && x_{ii} = 1 \qquad \text{for } i = 1, \ldots, n \\
& && \Big(x_{ij}\Big)_{i=1,\ldots,n, j=1,\ldots,n} \qquad \text{is a PSD matrix.}
\end{aligned}
$$

SDP: Semi-definite programming

1. If PSD $P = B^T B$ has a diagonal of 1
2. $\implies$ B has columns which are unit vectors.
3. If solve SDP (P), get back semi-definite matrix...
4. ... recover the vectors realizing the solution (i.e., compute B)
5. Now, do the rounding.
6. SDP (P) can be restated as

$$(SD) \qquad \max \quad \frac{1}{2}\sum_{i<j}\omega_{ij}(1 - x_{ij})$$

$$\text{subject to:} \quad x_{ii} = 1 \qquad \text{for } i = 1, \ldots, n$$

$$\left(x_{ij}\right)_{i=1,\ldots,n,j=1,\ldots,n} \quad \text{is a PSD matrix.}$$

SDP: Semi-definite programming

1. If PSD $\boldsymbol{P} = \boldsymbol{B}^T\boldsymbol{B}$ has a diagonal of 1
2. $\implies$ $\boldsymbol{B}$ has columns which are unit vectors.
3. If solve SDP $(\boldsymbol{P})$, get back semi-definite matrix...
4. ... recover the vectors realizing the solution (i.e., compute $\boldsymbol{B}$)
5. Now, do the rounding.
6. SDP $(\boldsymbol{P})$ can be restated as

$$
\begin{array}{llll}
(SD) & \max & \frac{1}{2}\sum_{i<j}\omega_{ij}(1 - x_{ij}) & \\
 & \text{subject to:} & x_{ii} = 1 & \text{for } i = 1, \ldots, n \\
 & & \left(x_{ij}\right)_{i=1,\ldots,n,j=1,\ldots,n} & \text{is a PSD matrix.}
\end{array}
$$

SDP: Semi-definite programming

1. If PSD $P = B^T B$ has a diagonal of 1
2. $\implies$ B has columns which are unit vectors.
3. If solve SDP (P), get back semi-definite matrix...
4. ... recover the vectors realizing the solution (i.e., compute B)
5. Now, do the rounding.
6. SDP (P) can be restated as

$$
\begin{array}{llll}
(SD) & \max & \frac{1}{2}\sum_{i<j} \omega_{ij}(1 - x_{ij}) & \\
& \text{subject to:} & x_{ii} = 1 & \text{for } i = 1, \ldots, n \\
& & \left(x_{ij}\right)_{i=1,\ldots,n, j=1,\ldots,n} & \text{is a PSD matrix.}
\end{array}
$$

SDP: Semi-definite programming

1. If PSD $P = B^T B$ has a diagonal of 1
2. $\implies$ B has columns which are unit vectors.
3. If solve SDP (P), get back semi-definite matrix...
4. ... recover the vectors realizing the solution (i.e., compute B)
5. Now, do the rounding.
6. SDP (P) can be restated as

$$
\begin{array}{lll}
(SD) & \max & \displaystyle\frac{1}{2}\sum_{i<j} \omega_{ij}(1 - x_{ij}) \\
 & \text{subject to:} & x_{ii} = 1 \qquad \text{for } i = 1, \ldots, n \\
 & & \left(x_{ij}\right)_{i=1,\ldots,n, j=1,\ldots,n} \quad \text{is a PSD matrix.}
\end{array}
$$

SDP: Semi-definite programming

1. SDP is

$$(SD) \qquad \max \quad \frac{1}{2}\sum_{i<j}\omega_{ij}(1 - x_{ij})$$
$$\text{subject to:} \quad x_{ii} = 1 \qquad \text{for } i = 1, \ldots, n$$
$$\left(x_{ij}\right)_{i=1,\ldots,n,j=1,\ldots,n} \quad \text{is a PSD matrix.}$$

2. find optimal value of a linear function...
3. ... over a set which is the intersection of:
   1. linear constraints, and
   2. set of positive semi-definite matrices.

SDP: Semi-definite programming

1. SDP is

$$
\begin{array}{llll}
(SD) & \max & \frac{1}{2}\sum_{i<j}\omega_{ij}(1 - x_{ij}) & \\
\text{subject to:} & & x_{ii} = 1 & \text{for } i = 1, \ldots, n \\
& & \left(x_{ij}\right)_{i=1,\ldots,n,j=1,\ldots,n} & \text{is a PSD matrix.}
\end{array}
$$

2. find optimal value of a linear function...
3. ... over a set which is the intersection of:
    1. linear constraints, and
    2. set of positive semi-definite matrices.

SDP: Semi-definite programming

1. SDP is

$$
\begin{array}{llll}
(SD) & \max & \dfrac{1}{2}\displaystyle\sum_{i<j}\omega_{ij}(1-x_{ij}) & \\
& \text{subject to:} & x_{ii}=1 & \text{for } i=1,\ldots,n \\
& & \Big(x_{ij}\Big)_{i=1,\ldots,n,j=1,\ldots,n} & \text{is a PSD matrix.}
\end{array}
$$

2. find optimal value of a linear function...
3. ... over a set which is the intersection of:
    1. linear constraints, and
    2. set of positive semi-definite matrices.

SDP: Semi-definite programming

1. SDP is

$$
\begin{aligned}
(SD) \qquad\qquad \max \quad & \frac{1}{2}\sum_{i<j}\omega_{ij}(1 - x_{ij}) \\
\text{subject to:} \quad & x_{ii} = 1 \qquad \text{for } i = 1, \ldots, n \\
& \left(x_{ij}\right)_{i=1,\ldots,n,j=1,\ldots,n} \quad \text{is a PSD matrix.}
\end{aligned}
$$

2. find optimal value of a linear function...
3. ... over a set which is the intersection of:
    1. linear constraints, and
    2. set of positive semi-definite matrices.

SDP: Semi-definite programming

1. SDP is

$$(SD) \qquad \max \quad \frac{1}{2}\sum_{i<j}\omega_{ij}(1 - x_{ij})$$

$$\text{subject to:} \quad x_{ii} = 1 \qquad \text{for } i = 1, \ldots, n$$

$$\left(x_{ij}\right)_{i=1,\ldots,n,j=1,\ldots,n} \quad \text{is a PSD matrix.}$$

2. find optimal value of a linear function...
3. ... over a set which is the intersection of:
   1. linear constraints, and
   2. set of positive semi-definite matrices.

SDP: Semi-definite programming

1. SDP is

$$(SD) \qquad \max \quad \frac{1}{2}\sum_{i<j} \omega_{ij}(1 - x_{ij})$$
$$\text{subject to:} \quad x_{ii} = 1 \qquad \text{for } i = 1, \ldots, n$$
$$\left(x_{ij}\right)_{i=1,\ldots,n, j=1,\ldots,n} \quad \text{is a PSD matrix.}$$

2. find optimal value of a linear function...
3. ... over a set which is the intersection of:
   1. linear constraints, and
   2. set of positive semi-definite matrices.

Lemma

Lemma

Let $\mathcal{U}$ be the set of $n \times n$ positive semidefinite matrices. The set $\mathcal{U}$ is convex.

Proof.

Consider $A, B \in \mathcal{U}$, and observe that for any $t \in [0,1]$, and vector $v \in \mathbb{R}^n$, we have:

$$v^T \Big(tA + (1-t)B \Big)\, v = v^T \Big(tAv + (1-t)Bv \Big)$$
$$= tv^T Av + (1-t)v^T Bv \geq 0 + 0 \geq 0,$$

since A and B are positive semidefinite. □

More on positive semidefinite matrices

1. PSD matrices corresponds to ellipsoids.
2. $x^T A x = 1$: the set of vectors solve this equation is an ellipsoid.
3. Eigenvalues of a PSD are all non-negative real numbers.
4. Given matrix: can in polynomial time decide if it is PSD.
5. ... by computing the eigenvalues of the matrix.
6. $\implies$ SDP: optimize a linear function over a convex domain.
7. SDP can be solved using interior point method, or the ellipsoid method.
8. See **Boyd and Vandenberghe [2004], Grötschel et al. [1993]** for more details.
9. Membership oracle: ability to decide in polynomial time, given a solution, whether its feasible or not.

More on positive semidefinite matrices

1. PSD matrices corresponds to ellipsoids.
2. $x^T A x = 1$: the set of vectors solve this equation is an ellipsoid.
3. Eigenvalues of a PSD are all non-negative real numbers.
4. Given matrix: can in polynomial time decide if it is PSD.
5. ... by computing the eigenvalues of the matrix.
6. $\implies$ SDP: optimize a linear function over a convex domain.
7. SDP can be solved using interior point method, or the ellipsoid method.
8. See **Boyd and Vandenberghe [2004], Grötschel et al. [1993]** for more details.
9. Membership oracle: ability to decide in polynomial time, given a solution, whether its feasible or not.

More on positive semidefinite matrices

1. PSD matrices corresponds to ellipsoids.
2. $x^T A x = 1$: the set of vectors solve this equation is an ellipsoid.
3. Eigenvalues of a PSD are all non-negative real numbers.
4. Given matrix: can in polynomial time decide if it is PSD.
5. ... by computing the eigenvalues of the matrix.
6. $\implies$ SDP: optimize a linear function over a convex domain.
7. SDP can be solved using interior point method, or the ellipsoid method.
8. See **Boyd and Vandenberghe [2004], Grötschel et al. [1993]** for more details.
9. Membership oracle: ability to decide in polynomial time, given a solution, whether its feasible or not.

More on positive semidefinite matrices

1. PSD matrices corresponds to ellipsoids.
2. $x^T A x = 1$: the set of vectors solve this equation is an ellipsoid.
3. Eigenvalues of a PSD are all non-negative real numbers.
4. Given matrix: can in polynomial time decide if it is PSD.
5. ... by computing the eigenvalues of the matrix.
6. $\implies$ SDP: optimize a linear function over a convex domain.
7. SDP can be solved using interior point method, or the ellipsoid method.
8. See **Boyd and Vandenberghe [2004], Grötschel et al. [1993]** for more details.
9. Membership oracle: ability to decide in polynomial time, given a solution, whether its feasible or not.

More on positive semidefinite matrices

1. PSD matrices corresponds to ellipsoids.
2. $x^T A x = 1$: the set of vectors solve this equation is an ellipsoid.
3. Eigenvalues of a PSD are all non-negative real numbers.
4. Given matrix: can in polynomial time decide if it is PSD.
5. ... by computing the eigenvalues of the matrix.
6. $\implies$ SDP: optimize a linear function over a convex domain.
7. SDP can be solved using interior point method, or the ellipsoid method.
8. See **Boyd and Vandenberghe [2004], Grötschel et al. [1993]** for more details.
9. Membership oracle: ability to decide in polynomial time, given a solution, whether its feasible or not.

More on positive semidefinite matrices

1. PSD matrices corresponds to ellipsoids.
2. $x^T A x = 1$: the set of vectors solve this equation is an ellipsoid.
3. Eigenvalues of a PSD are all non-negative real numbers.
4. Given matrix: can in polynomial time decide if it is PSD.
5. ... by computing the eigenvalues of the matrix.
6. $\implies$ SDP: optimize a linear function over a convex domain.
7. SDP can be solved using interior point method, or the ellipsoid method.
8. See **Boyd and Vandenberghe [2004], Grötschel et al. [1993]** for more details.
9. Membership oracle: ability to decide in polynomial time, given a solution, whether its feasible or not.

More on positive semidefinite matrices

1. PSD matrices corresponds to ellipsoids.
2. $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} = 1$: the set of vectors solve this equation is an ellipsoid.
3. Eigenvalues of a PSD are all non-negative real numbers.
4. Given matrix: can in polynomial time decide if it is PSD.
5. ... by computing the eigenvalues of the matrix.
6. $\implies$ SDP: optimize a linear function over a convex domain.
7. SDP can be solved using interior point method, or the ellipsoid method.
8. See **Boyd and Vandenberghe [2004], Grötschel et al. [1993]** for more details.
9. Membership oracle: ability to decide in polynomial time, given a solution, whether its feasible or not.

More on positive semidefinite matrices

1. PSD matrices corresponds to ellipsoids.
2. $x^T A x = 1$: the set of vectors solve this equation is an ellipsoid.
3. Eigenvalues of a PSD are all non-negative real numbers.
4. Given matrix: can in polynomial time decide if it is PSD.
5. ... by computing the eigenvalues of the matrix.
6. $\implies$ SDP: optimize a linear function over a convex domain.
7. SDP can be solved using interior point method, or the ellipsoid method.
8. See **Boyd and Vandenberghe [2004], Grötschel et al. [1993]** for more details.
9. Membership oracle: ability to decide in polynomial time, given a solution, whether its feasible or not.

More on positive semidefinite matrices

1. PSD matrices corresponds to ellipsoids.
2. $x^T A x = 1$: the set of vectors solve this equation is an ellipsoid.
3. Eigenvalues of a PSD are all non-negative real numbers.
4. Given matrix: can in polynomial time decide if it is PSD.
5. ... by computing the eigenvalues of the matrix.
6. $\implies$ SDP: optimize a linear function over a convex domain.
7. SDP can be solved using interior point method, or the ellipsoid method.
8. See **Boyd and Vandenberghe [2004], Grötschel et al. [1993]** for more details.
9. Membership oracle: ability to decide in polynomial time, given a solution, whether its feasible or not.

Bibliographical Notes

1. Approx. algorithm presented by Goemans and Williamson **Goemans and Williamson [1995]**.
2. **Håstad [2001]** showed that MAX CUT can not be approximated within a factor of $\mathbf{16/17 \approx 0.941176}$.
3. **Khot et al. [2004]** showed a hardness result that matches the constant of Goemans and Williamson (i.e., one can not approximate it better than α, unless $\mathsf{P} = \mathsf{NP}$).

Bibliographical Notes

1. Relies on two conjectures: "Unique Games Conjecture" and "Majority is Stablest".
2. "Majority is Stablest" conjecture was proved by **Mossel et al. [2005]**.
3. Not clear if the "Unique Games Conjecture" is true, see the discussion in **Khot et al. [2004]**.
4. Goemans and Williamson work spurred wide research on using SDP for approximation algorithms.

Notes

Notes

Notes

Notes

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge, 2004. URL http://www.stanford.edu/~boyd/cvxbook/.

M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, 42(6): 1115–1145, November 1995.

M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin Heidelberg, 2nd edition, 1993.

J. Håstad. Some optimal inapproximability results. *J. Assoc. Comput. Mach.*, 48(4):798–859, 2001. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/502090.502098.

S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for max cut and other 2-variable csps. In *Proc. 45th Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), pages 146–154, 2004. To appear in SICOMP.

E. Mossel, R. O'Donnell, and K. Oleszkiewicz. Noise stability of functions with low influences invariance and optimality. In *Proc. 46th Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), pages 21–30, 2005.