Design and Analysis of Algorithms

Approximation Algorithms – Part 5

Pan Peng

School of Computer Science and Technology
University of Science and Technology of China

Outline

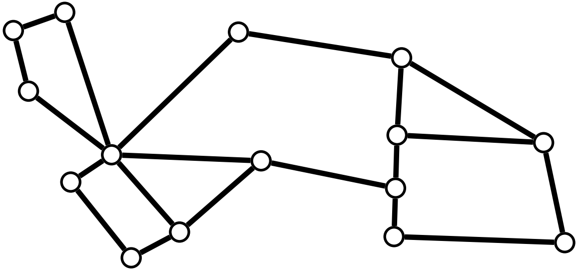
- Approximation algorithms for
  - max cut
  - correlation clustering

Max cut

Max (or maximum) cut problem

Input: an undirected weighted graph $G = (V, E, w)$ such that $w_{ij} \geq 0$ for edges $(i, j) \in E$

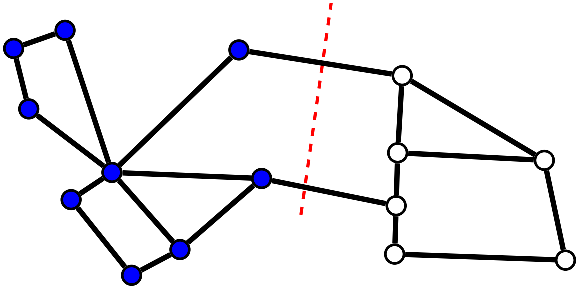
Objective: find a set $S \subseteq V$, such that $\sum_{i \in S, j \in V \setminus S} w_{ij}$, i.e., the weight of the corresponding cut, is maximized.



Max (or maximum) cut problem

Input: an undirected weighted graph $G = (V, E, w)$ such that $w_{ij} \geq 0$ for edges $(i, j) \in E$

Objective: find a set $S \subseteq V$, such that $\sum_{i \in S, j \in V \setminus S} w_{ij}$, i.e., the weight of the corresponding cut, is maximized.

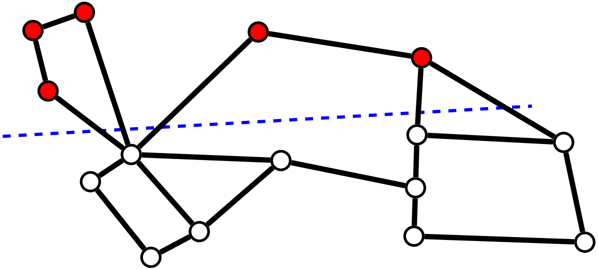


unweighted graph: a cut of size 2

Max (or maximum) cut problem

Input: an undirected weighted graph $G = (V, E, w)$ such that $w_{ij} \geq 0$ for edges $(i, j) \in E$

Objective: find a set $S \subseteq V$, such that $\sum_{i \in S, j \in V \setminus S} w_{ij}$, i.e., the weight of the corresponding cut, is maximized.



unweighted graph: a cut of size 5

Max (or maximum) cut problem

Input: an undirected weighted graph $G = (V, E, w)$ such that $w_{ij} \geq 0$ for edges $(i, j) \in E$

Objective: find a set $S \subseteq V$, such that $\sum_{i \in S, j \in V \setminus S} w_{ij}$, i.e., the weight of the corresponding cut, is maximized.

- The max cut problem is **NP**-hard
- There are several algorithms achieving $\frac{1}{2}$-approximation, e.g., local search, randomized assignment
- In the following, we present an SDP based algorithm achieving 0.878-approximation ratio, by Goemans and Williamson 1996.

A Tool: Semidefinite Programming (SDP)

Positive semidefinite (PSD) matrix

X : $n \times n$ real symmetric matrix

Positive semidefinite (PSD) matrix

X : $n \times n$ real symmetric matrix

X is called a positive semidefinite (PSD) matrix, denoted $X \succeq 0$, if

- $v^T X v \geq 0$ for all $v \in \mathbb{R}^n$.

Positive semidefinite (PSD) matrix

X : $n \times n$ real symmetric matrix

X is called a positive semidefinite (PSD) matrix, denoted $X \succeq 0$, if

- $v^T X v \geq 0$ for all $v \in \mathbb{R}^n$.

Recall: the following are equivalent:

- $X \succeq 0$ (i.e., X is PSD)
- all eigenvalues of X are non-negative ($\lambda \in \mathbb{R}$ is an eigenvalue of X if there is some $v \in \mathbb{R}^n$ with $Xv = \lambda v$)
- $X = U^T U$ for some matrix $U \in \mathbb{R}^{k \times n}$, where $k \leq n$

Semidefinite program (SDP)

SDPs: a class of convex programs; many of them solvable in polynomial time (e.g., by ellipsoid algorithm)

Two equivalent definitions of SDP:

Def 1:

$$\begin{aligned} \max \quad & \sum_{i,j} w_{ij} \cdot x_{ij} \\ \text{s.t.} \quad & \sum_{i,j} a^r_{ij} \cdot x_{ij} \geq b_r \quad \forall\, r \in I \\ & X \succeq 0 \end{aligned}$$

where w_{ij}, a^r_{ij}, b_r are given numbers and $X = (x_{ij})_{1 \leq i,j \leq n}$ denote the variables. Note that the objective and the first set of constraints are linear in the variables

Semidefinite program (SDP)

Since X is PSD, we can write $X = U^T U$ for some $k \times n$ matrix U. If $u_1, \ldots, u_n \in \mathbb{R}^k$ denote the columns of U, then $x_{ij} = \langle u_i, u_j \rangle$. (here $\langle u, v \rangle$ denotes the inner product of vectors.) As $k \leq n$, we can write the previous SDP as

Def 2:

$$\begin{aligned} \max \quad & \sum_{i,j} w_{ij} \cdot \langle u_i, u_j \rangle \\ \text{s.t.} \quad & \sum_{i,j} a_{ij}^r \cdot \langle u_i, u_j \rangle \geq b_r \quad && \forall\, r \in I \\ & u_i \in \mathbb{R}^n && \forall i \in [n] \end{aligned}$$

Remark: note that if $k < n$, then a vector $u \in \mathbb{R}^k$ can be naturally extended to an n-dimensional vector $u' \in \mathbb{R}^n$ by adding some zero entries.

SDP for Max Cut

SDP for Max Cut

Input: an undirected weighted graph $G = (V, E, w)$ such that $w_{ij} \geq 0$ for edges $(i, j) \in E$

Objective: find a set $S \subseteq V$, such that $\sum_{i \in S, j \in V \setminus S} w_{ij}$, i.e., the weight of the corresponding cut, is maximized.

SDP for Max Cut

Input: an undirected weighted graph $G = (V, E, w)$ such that $w_{ij} \geq 0$ for edges $(i, j) \in E$

Objective: find a set $S \subseteq V$, such that $\sum_{i \in S, j \in V \setminus S} w_{ij}$, i.e., the weight of the corresponding cut, is maximized.

- use variable v_i for each $i \in V$
- ideally, want the solution to only have two directions, either $v_i = e$ (corresponding to $i \in S$), or $v_i = -e$ (corresponding to $i \notin S$), where e is a fixed unit vector
- We set up the SDP as

$$
\begin{aligned}
\max \quad & \sum_{(i,j) \in E} \frac{1}{2} w_{ij} \cdot (1 - \langle v_i, v_j \rangle) & \\
\text{s.t.} \quad & \langle v_i, v_i \rangle = 1 & \forall\, i \in V \\
& v_i \in \mathbb{R}^n & \forall i \in V
\end{aligned}
$$

SDP for Max Cut

- note the optimal solution of SDP has objective $\mathrm{OPT}_{\mathrm{SDP}}$ that is at least OPT, the value of the optimal solution of the Max Cut problem.
- we need to round the optimal SDP solution

SDP based algorithm for Max Cut

Algorithm SDP-MaxCut

1. solve the SDP for Max Cut to obtain the optimum solution $v_i, \forall i \in V$
2. choose a uniformly random unit vector $r \in \mathbb{R}^n$
   ("unit" means $\|r\|_2 = 1$)
3. output $S := \{i \in V \mid \langle r, v_i \rangle \geq 0\}$

SDP based algorithm for Max Cut

Algorithm $\textsc{SDP-MaxCut}$

1. solve the SDP for Max Cut to obtain the optimum solution $v_i, \forall i \in V$
2. choose a uniformly random unit vector $r \in \mathbb{R}^n$
   ("unit" means $\|r\|_2 = 1$)
3. output $S := \{i \in V \mid \langle r, v_i \rangle \geq 0\}$

Remark: Implementation of Step 2:

- generate n Gaussian random variables $x_1, x_2, ..., x_n \in \mathcal{N}(0, 1)$ (i.e., Gaussian distribution with expectation 0 and variance 1) independently at random, and let

$$r = \frac{1}{\sqrt{x_1^2 + \cdots + x_n^2}} (x_1, \ldots, x_n)^T$$

Analysis of SDP-MaxCut

Lemma: For each $i, j \in V$,

$$\Pr[i, j \text{ separated by } S] = \Pr[|\{i, j\} \cap S| = 1] \geq 0.878 \cdot \frac{1 - \langle v_i, v_j \rangle}{2}$$

Analysis of SDP-MaxCut

Lemma: For each $i, j \in V$,

$$\Pr[i, j \text{ separated by } S] = \Pr[|\{i, j\} \cap S| = 1] \geq 0.878 \cdot \frac{1 - \langle v_i, v_j \rangle}{2}$$

Proof:

- Since only two vectors v_i, v_j are involved, consider the 2-dimensional plane H where $v_i \in H, v_j \in H$.

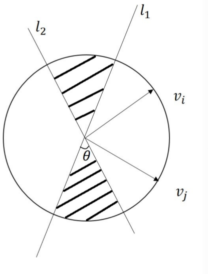


Figure: v_i, v_j in 2-dimensional plane

Proof of Lemma - cont.

- let $\bar{r}$ be the projection of r on H

Proof of Lemma - cont.

- let $\bar{r}$ be the projection of r on H
- note that the normalization $\frac{\bar{r}}{\|\bar{r}\|}$ is a uniform unit vector in $\mathbb{R}^d$

Proof of Lemma - cont.

- let $\bar{r}$ be the projection of r on H
- note that the normalization $\frac{\bar{r}}{\|\bar{r}\|}$ is a uniform unit vector in $\mathbb{R}^d$
- note that $\langle r, v_i \rangle = \langle \bar{r}, v_i \rangle$

Proof of Lemma - cont.

- let $\bar{r}$ be the projection of r on H
- note that the normalization $\frac{\bar{r}}{\|\bar{r}\|}$ is a uniform unit vector in $\mathbb{R}^d$
- note that $\langle r, v_i \rangle = \langle \bar{r}, v_i \rangle$
- Then

$$\Pr[i, j \text{ separated by } S] = \Pr[\langle \bar{r}, v_i \rangle \text{ and } \langle \bar{r}, v_j \rangle \text{ have opposite sign}]$$

Proof of Lemma - cont.

- let $\bar{r}$ be the projection of r on H
- note that the normalization $\frac{\bar{r}}{\|\bar{r}\|}$ is a uniform unit vector in $\mathbb{R}^d$
- note that $\langle r, v_i \rangle = \langle \bar{r}, v_i \rangle$
- Then

$$\Pr[i, j \text{ separated by } S] = \Pr[\langle \bar{r}, v_i \rangle \text{ and } \langle \bar{r}, v_j \rangle \text{ have opposite sign}]$$

- If $\bar{r}$ falls into the shadow area in the above Figure, $\langle \bar{r}, v_i \rangle$ and $\langle \bar{r}, v_j \rangle$ have opposite sign.

Proof of Lemma - cont.

- let $\bar{r}$ be the projection of r on H
- note that the normalization $\frac{\bar{r}}{\|\bar{r}\|}$ is a uniform unit vector in $\mathbb{R}^d$
- note that $\langle r, v_i \rangle = \langle \bar{r}, v_i \rangle$
- Then

$$\Pr[i, j \text{ separated by } S] = \Pr[\langle \bar{r}, v_i \rangle \text{ and } \langle \bar{r}, v_j \rangle \text{ have opposite sign}]$$

- If $\bar{r}$ falls into the shadow area in the above Figure, $\langle \bar{r}, v_i \rangle$ and $\langle \bar{r}, v_j \rangle$ have opposite sign.
- Thus

$$\begin{aligned}
\Pr[i, j \text{ separated by } S] &= \frac{2\theta}{2\pi} = \frac{1}{\pi} \arccos\langle v_i, v_j \rangle \\
&= \frac{1 - \langle v_i, v_j \rangle}{\pi} \cdot \frac{\arccos\langle v_i, v_j \rangle}{1 - \langle v_i, v_j \rangle} \\
&\geq \frac{1 - \langle v_i, v_j \rangle}{2} \cdot \frac{2}{\pi} \left(\min_{x \in [-1,1]} \frac{\arccos x}{1 - x} \right) \\
&\geq 0.878 \cdot \frac{1 - \langle v_i, v_j \rangle}{2}
\end{aligned}$$

Performance guarantee of $\textsc{SDP-MaxCut}$

By taking the weighted sum over all pairs $i, j \in V$ and linearity of expectation, and noting that the algorithm $\textsc{SDP-MaxCut}$ runs in polynomial time, it follows that:

Theorem: The expected value of the solution S of $\textsc{SDP-MaxCut}$ is at least $0.878 \cdot \mathrm{OPT}_{\mathrm{SDP}} \geq 0.878 \cdot \mathrm{OPT}$. Thus, $\textsc{SDP-MaxCut}$ is a 0.878-approximation algorithm for the Max Cut problem.

Randomized Approximation Algorithms

Approximation in expectation: $\mathrm{E}[\frac{\text{result}}{\mathrm{OPT}}] \lesseqgtr \alpha$.

- We have a bound on the expected approximation factor.

Randomized Approximation Algorithms

Approximation in expectation: $\mathrm{E}[\frac{\text{result}}{\mathrm{OPT}}] \lesseqgtr \alpha$.

- We have a bound on the expected approximation factor.
- The algorithm may return worse results *most of the time*.

Randomized Approximation Algorithms

Approximation in expectation: $\mathrm{E}[\frac{\text{result}}{\mathrm{OPT}}] \lesseqgtr \alpha$.

- We have a bound on the expected approximation factor.
- The algorithm may return worse results *most of the time*.
- Example: the above algorithm.

Randomized Approximation Algorithms

Approximation in expectation: $\mathrm{E}[\frac{\text{result}}{\mathrm{OPT}}] \lesseqgtr \alpha$.

- We have a bound on the expected approximation factor.
- The algorithm may return worse results *most of the time*.
- Example: the above algorithm.
- Remark: the above algorithm can be *derandomized*; deterministic algorithm with 0.878-approximation.

Randomized Approximation Algorithms

Approximation in expectation: $\mathrm{E}[\frac{\text{result}}{\mathrm{OPT}}] \lesseqgtr \alpha$.

- We have a bound on the expected approximation factor.
- The algorithm may return worse results *most of the time*.
- Example: the above algorithm.
- Remark: the above algorithm can be *derandomized*; deterministic algorithm with 0.878-approximation.

Randomized Approximation Algorithms

Approximation in expectation: $\mathrm{E}[\frac{\text{result}}{\text{OPT}}] \lesseqgtr \alpha$.

- We have a bound on the expected approximation factor.
- The algorithm may return worse results *most of the time*.
- Example: the above algorithm.
- Remark: the above algorithm can be *derandomized*; deterministic algorithm with 0.878-approximation.

Approximation with high probability: $\Pr[\frac{\text{result}}{\text{OPT}} \lesseqgtr \alpha] > 1 - \varepsilon$

- The approximation factor is bounded with high probability.

Randomized Approximation Algorithms

Approximation in expectation: $\mathrm{E}[\frac{\text{result}}{\mathrm{OPT}}] \lesseqgtr \alpha$.

- We have a bound on the expected approximation factor.
- The algorithm may return worse results *most of the time*.
- Example: the above algorithm.
- Remark: the above algorithm can be *derandomized*; deterministic algorithm with 0.878-approximation.

Approximation with high probability: $\Pr[\frac{\text{result}}{\mathrm{OPT}} \lesseqgtr \alpha] > 1 - \varepsilon$

- The approximation factor is bounded with high probability.
- The algorithm may return worse results *occasionally*.

Randomized Approximation Algorithms

Approximation in expectation: $\mathrm{E}[\frac{\text{result}}{\mathrm{OPT}}] \lesseqgtr \alpha$.

- We have a bound on the expected approximation factor.
- The algorithm may return worse results *most of the time*.
- Example: the above algorithm.
- Remark: the above algorithm can be *derandomized*; deterministic algorithm with 0.878-approximation.

Approximation with high probability: $\Pr[\frac{\text{result}}{\mathrm{OPT}} \lesseqgtr \alpha] > 1 - \varepsilon$

- The approximation factor is bounded with high probability.
- The algorithm may return worse results *occasionally*.

Randomized Approximation Algorithms

Approximation in expectation: $\mathrm{E}[\frac{\text{result}}{\mathrm{OPT}}] \lesseqgtr \alpha$.

- We have a bound on the expected approximation factor.
- The algorithm may return worse results *most of the time*.
- Example: the above algorithm.
- Remark: the above algorithm can be *derandomized*; deterministic algorithm with 0.878-approximation.

Approximation with high probability: $\Pr[\frac{\text{result}}{\mathrm{OPT}} \lesseqgtr \alpha] > 1 - \varepsilon$

- The approximation factor is bounded with high probability.
- The algorithm may return worse results *occasionally*.

Approximation in expected running time: $\frac{\text{result}}{\mathrm{OPT}} \lesseqgtr \alpha$

- The approximation factor satisfies the bound *at all times*.

Randomized Approximation Algorithms

Approximation in expectation: $\mathrm{E}[\frac{\text{result}}{\text{OPT}}] \lesseqgtr \alpha$.

- We have a bound on the expected approximation factor.
- The algorithm may return worse results *most of the time*.
- Example: the above algorithm.
- Remark: the above algorithm can be *derandomized*; deterministic algorithm with 0.878-approximation.

Approximation with high probability: $\Pr[\frac{\text{result}}{\text{OPT}} \lesseqgtr \alpha] > 1 - \varepsilon$

- The approximation factor is bounded with high probability.
- The algorithm may return worse results *occasionally*.

Approximation in expected running time: $\frac{\text{result}}{\text{OPT}} \lesseqgtr \alpha$

- The approximation factor satisfies the bound *at all times*.
- Occasionally the *running time* might be bad.

Correlation clustering - version 1

Correlation clustering

Input: Given a complete graph $G = (V, E)$ on n vertices. Each edge is labeled $+$ or $-$.

Goal: find a partition of vertices so that the agreement is maximized.

- for a partition $\mathcal{P} = \{P_1, \ldots, P_t\}$ of V, the agreement of $\mathcal{P}$ is the number of positive edges within a cluster (i.e. some part P_i) plus the number of negative edges between clusters

Correlation clustering

Input: Given a complete graph $G = (V, E)$ on n vertices. Each edge is labeled $+$ or $-$.

Goal: find a partition of vertices so that the agreement is maximized.

- for a partition $\mathcal{P} = \{P_1, \ldots, P_t\}$ of V, the agreement of $\mathcal{P}$ is the number of positive edges within a cluster (i.e. some part P_i) plus the number of negative edges between clusters

Remark: a clustering problem without any assumption on the number of clusters

Correlation clustering

Input: Given a complete graph $G = (V, E)$ on n vertices. Each edge is labeled $+$ or $-$.

Goal: find a partition of vertices so that the agreement is maximized.

- for a partition $\mathcal{P} = \{P_1, \ldots, P_t\}$ of V, the agreement of $\mathcal{P}$ is the number of positive edges within a cluster (i.e. some part P_i) plus the number of negative edges between clusters

Remark: a clustering problem without any assumption on the number of clusters

Intuition

- Partition a set of objects such that “similar” objects are grouped together and “dissimilar” objects are set apart.

From vector program to SDP

For an edge $(i, j) \in E$, let $\mathrm{sgn}(ij) = 1$ if (i, j) is labeled $+$, and $\mathrm{sgn}(ij) = -1$ otherwise.

From vector program to SDP

For an edge $(i, j) \in E$, let $\mathrm{sgn}(ij) = 1$ if (i, j) is labeled $+$, and $\mathrm{sgn}(ij) = -1$ otherwise. For $1 \le k \le n$, let $e_k \in \{0, 1\}^n$ be the k-th unit vector, i.e., $e_k(t) = 1$ if $t = k$ and 0 otherwise.

From vector program to SDP

For an edge $(i,j) \in E$, let $\mathrm{sgn}(ij) = 1$ if (i,j) is labeled $+$, and $\mathrm{sgn}(ij) = -1$ otherwise.
For $1 \le k \le n$, let $e_k \in \{0,1\}^n$ be the k-th unit vector, i.e., $e_k(t) = 1$ if $t = k$ and 0 otherwise.

$$\begin{aligned} \max \quad & \sum_{i<j:\mathrm{sgn}(ij)=1} x_i^T x_j + \sum_{i<j:\mathrm{sgn}(ij)=-1} (1 - x_i^T x_j) \\ \text{s.t.} \quad & x_i \in \{e_1, \ldots, e_n\} \quad \forall\, i \in V \end{aligned}$$

(Intuition: in the optimal solution, vertices v in the k-th cluster are assigned vector e_k)

From vector program to SDP

For an edge $(i, j) \in E$, let $\operatorname{sgn}(ij) = 1$ if (i, j) is labeled $+$, and $\operatorname{sgn}(ij) = -1$ otherwise. For $1 \leq k \leq n$, let $e_k \in \{0, 1\}^n$ be the k-th unit vector, i.e., $e_k(t) = 1$ if $t = k$ and 0 otherwise.

$$\begin{aligned} \max \quad & \sum_{i<j:\operatorname{sgn}(ij)=1} x_i^T x_j + \sum_{i<j:\operatorname{sgn}(ij)=-1} (1 - x_i^T x_j) \\ \text{s.t.} \quad & x_i \in \{e_1, \ldots, e_n\} \quad \forall\, i \in V \end{aligned}$$

(Intuition: in the optimal solution, vertices v in the k-th cluster are assigned vector e_k)

SDP relaxation of the correlation clustering problem

$$\begin{aligned} \max \quad & \sum_{i<j:\operatorname{sgn}(ij)=1} x_i^T x_j + \sum_{i<j:\operatorname{sgn}(ij)=-1} (1 - x_i^T x_j) \\ \text{s.t.} \quad & x_i^T x_i = 1 \quad \forall\, i \in V \\ & x_i^T x_j \geq 0 \quad \forall\, i, j \in V \\ & x_i \in \mathbb{R}^n \quad \forall\, i \in V \end{aligned}$$

Rounding of the SDP

Main ideas:

- Follow the Max-Cut approach, and choose two hyperplanes
- We always obtain at most four clusters

Rounding of the SDP

Main ideas:

- Follow the Max-Cut approach, and choose two hyperplanes
- We always obtain at most four clusters

Algorithm SDP-CC

1. solve the SDP for the correlation clustering problem to obtain the optimum solution $v_i, \forall i \in V$
2. choose two uniformly random unit vectors $r_1, r_2 \in \mathbb{R}^n$
3. output
   - $R_1 := \{i \in V \mid \langle r_1, v_i \rangle \geq 0$ and $\langle r_2, v_i \rangle \geq 0\}$
   - $R_2 := \{i \in V \mid \langle r_1, v_i \rangle \geq 0$ and $\langle r_2, v_i \rangle < 0\}$
   - $R_3 := \{i \in V \mid \langle r_1, v_i \rangle < 0$ and $\langle r_2, v_i \rangle \geq 0\}$
   - $R_4 := \{i \in V \mid \langle r_1, v_i \rangle < 0$ and $\langle r_2, v_i \rangle < 0\}$

Analysis of $\mathrm{SDP\text{-}CC}$

- for $(i,j) \in E$, let q_{ij} be the probability that v_i, v_j are on the same side of both hyperplanes, and the expected cost of our rounding algorithm is

$$\sum_{i<j:\mathrm{sgn}(ij)=1} q_{ij} + \sum_{i<j:\mathrm{sgn}(ij)=-1} (1 - q_{ij})$$

Analysis of SDP-CC

- for $(i,j) \in E$, let q_{ij} be the probability that v_i, v_j are on the same side of both hyperplanes, and the expected cost of our rounding algorithm is

$$\sum_{i<j:\mathrm{sgn}(ij)=1} q_{ij} + \sum_{i<j:\mathrm{sgn}(ij)=-1} (1-q_{ij})$$

- moreover, $q_{ij} = (1 - \frac{\arccos v_i^T v_j}{\pi})^2$

Analysis of SDP-CC

- for $(i,j) \in E$, let q_{ij} be the probability that v_i, v_j are on the same side of both hyperplanes, and the expected cost of our rounding algorithm is

$$\sum_{i<j:\mathrm{sgn}(ij)=1} q_{ij} + \sum_{i<j:\mathrm{sgn}(ij)=-1} (1 - q_{ij})$$

- moreover, $q_{ij} = (1 - \frac{\arccos v_i^T v_j}{\pi})^2$
- if we let

$$\alpha_1 = \min_{0 \le z \le 1} (1 - \frac{\arccos z}{\pi})^2 / z, \;\; \alpha_2 = \min_{0 \le z \le 1} (1 - (1 - \frac{\arccos z}{\pi})^2)/(1 - z),$$

then $\alpha^* = \min\{\alpha_1, \alpha_2\} \geq 0.75$ (see the book "The Design of Approximation Algorithms Chapter 6")

Analysis of SDP-CC

- for $(i,j) \in E$, let q_{ij} be the probability that v_i, v_j are on the same side of both hyperplanes, and the expected cost of our rounding algorithm is

$$\sum_{i<j:\mathrm{sgn}(ij)=1} q_{ij} + \sum_{i<j:\mathrm{sgn}(ij)=-1} (1-q_{ij})$$

- moreover, $q_{ij} = (1 - \frac{\arccos v_i^T v_j}{\pi})^2$
- if we let

$$\alpha_1 = \min_{0\le z\le 1} (1 - \frac{\arccos z}{\pi})^2/z, \ \ \alpha_2 = \min_{0\le z\le 1} (1-(1-\frac{\arccos z}{\pi})^2)/(1-z),$$

then $\alpha^* = \min\{\alpha_1, \alpha_2\} \ge 0.75$ (see the book "The Design of Approximation Algorithms Chapter 6")
- Then the expected cost is at least

$$\alpha^* \cdot (\sum_{i<j:\mathrm{sgn}(ij)=1} v_i^T v_j + \sum_{i<j:\mathrm{sgn}(ij)=-1} (1 - v_i^T v_j)) \ge \alpha^* \cdot \mathrm{OPT}$$

Analysis of $\mathrm{SDP\text{-}CC}$

Theorem: The algorithm $\mathrm{SDP\text{-}CC}$ is a 0.75-approximation for the agreement maximization version of the correlation clustering problem.

Analysis of $\mathrm{SDP\text{-}CC}$

Theorem: The algorithm $\mathrm{SDP\text{-}CC}$ is a 0.75-approximation for the agreement maximization version of the correlation clustering problem.

Remark: There exists a $(1-\varepsilon)$-approximation algorithm for the above problem, for any constant $\varepsilon > 0$.

Correlation clustering - version 2

Correlation clustering

Input: Given a complete graph $G = (V, E)$ on n vertices. Each edge is labeled $+$ or $-$.

Goal: find a partition of vertices so that the disagreement is minimized.

- for a partition $\mathcal{P} = \{P_1, \ldots, P_t\}$ of V, the disagreement of $\mathcal{P}$ is the number of negative edges within a cluster plus the number of positive edges between clusters

Correlation clustering

Input: Given a complete graph $G = (V, E)$ on n vertices. Each edge is labeled $+$ or $-$.

Goal: find a partition of vertices so that the disagreement is minimized.

- for a partition $\mathcal{P} = \{P_1, \ldots, P_t\}$ of V, the disagreement of $\mathcal{P}$ is the number of negative edges within a cluster plus the number of positive edges between clusters

Remark:

- The best known approximation algorithm for the above problem achieves approximation ratio $1.994 + \varepsilon$ for any constant $\varepsilon > 0$.
- Here we present a simple greedy algorithm.

A greed algorithm

Let $G = (V, E)$ be the input graph such that each edge is labeled $+$ or $-$.

A greed algorithm

Let $G = (V, E)$ be the input graph such that each edge is labeled $+$ or $-$.

Algorithm $\textsc{Pivot-CC}$

1. $E^+ \leftarrow$ the set of all $+$ edges
2. while $V \neq \emptyset$ do
   - 2.1 $i \leftarrow$ uniformly random node from V
   - 2.2 create cluster $C_i = \{i\} \cup E^+(i)$ ($E^+(i)$: set of $+$ neighbors of i)
   - 2.3 $V \leftarrow V \setminus C_i$
   - 2.4 $E^+ \leftarrow E^+ \cap (V \times V)$
3. output all the clusters C_i's

Performance guarantee of PIVOT-CC

Theorem: Algorithm PIVOT-CC is a randomized expected 3-approximation algorithm for the disagreement minimization version of the correlation clustering problem.

Performance guarantee of PIVOT-CC

Theorem: Algorithm PIVOT-CC is a randomized expected 3-approximation algorithm for the disagreement minimization version of the correlation clustering problem.

Note: for the proof, see the paper "Aggregating inconsistent information: ranking and clustering", Ailon, Charikar and Newman, STOC 05