Approximation Algorithms (ADM III)

5- Random Sampling & Randomized Rounding

Guillaume Sagnol

Technische Universität Berlin

Randomized Approximation Algorithm

Definition 5.1

A randomized α-approximation algorithm is a polynomial-time randomized algorithm which always finds a feasible solution whose *expected* value is bounded by $\alpha \cdot \texttt{OPT}$.

Randomized Approximation Algorithm

Definition 5.1

A randomized α-approximation algorithm is a polynomial-time randomized algorithm which always finds a feasible solution whose *expected* value is bounded by $\alpha \cdot \texttt{OPT}$.

Remarks

- Often, a randomized α-approximation algorithm can be *derandomized*, i.e., turned into a deterministic α-approximation algorithm.

Randomized Approximation Algorithm

Definition 5.1

A randomized α-approximation algorithm is a polynomial-time randomized algorithm which always finds a feasible solution whose *expected* value is bounded by $\alpha \cdot \texttt{OPT}$.

Remarks

- Often, a randomized α-approximation algorithm can be *derandomized*, i.e., turned into a deterministic α-approximation algorithm.
- It is usually simpler to state and analyze the randomized algorithm.

Randomized Approximation Algorithm

Definition 5.1

A randomized α-approximation algorithm is a polynomial-time randomized algorithm which always finds a feasible solution whose *expected* value is bounded by $\alpha \cdot \texttt{OPT}$.

Remarks

- Often, a randomized α-approximation algorithm can be *derandomized*, i.e., turned into a deterministic α-approximation algorithm.
- It is usually simpler to state and analyze the randomized algorithm.
- Sometimes, the only known way of analyzing a deterministic approximation algorithm is to analyze a randomized version.

Randomized Approximation Algorithm

Definition 5.1

A randomized α-approximation algorithm is a polynomial-time randomized algorithm which always finds a feasible solution whose *expected* value is bounded by $\alpha \cdot \texttt{OPT}$.

Remarks

- Often, a randomized α-approximation algorithm can be *derandomized*, i.e., turned into a deterministic α-approximation algorithm.
- It is usually simpler to state and analyze the randomized algorithm.
- Sometimes, the only known way of analyzing a deterministic approximation algorithm is to analyze a randomized version.
- Sometimes one can show that the performance guarantee of a randomized algorithm holds with high probability.

Outline

1. Random sampling for MAX SAT and MAX CUT
2. Randomized Rounding for MAX SAT
3. Price-Collecting Steiner Tree Problem
4. Uncapacited Facility Location Problem
5. Minimizing the Weighted Sum of Completion Times
6. Minimum-Capacity Multicommodity Flow Problem
7. Rounding a semidefinite programming relaxation for MAXCUT

Maximum Satisfiability Problem (MAX SAT)

Given: Boolean variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$ with weights $w_1, \ldots, w_m \in \mathbb{R}_{\geq 0}$.

(Clause is disjunction of Boolean variables or negations, e.g., $x_1 \vee \overline{x_2} \vee x_3$)

Task: Find a truth assignment to $x_1, \ldots, x_n$.

Objective: Maximize the total weight of satisfied clauses.

Maximum Satisfiability Problem (MAX SAT)

Given: Boolean variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$ with weights $w_1, \ldots, w_m \in \mathbb{R}_{\geq 0}$.

(Clause is disjunction of Boolean variables or negations, e.g., $x_1 \vee \overline{x_2} \vee x_3$)

Task: Find a truth assignment to $x_1, \ldots, x_n$.

Objective: Maximize the total weight of satisfied clauses.

Example: $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (x_2 \vee x_3) \wedge (\overline{x_3})$

Maximum Satisfiability Problem (MAX SAT)

Given: Boolean variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$ with weights $w_1, \ldots, w_m \in \mathbb{R}_{\geq 0}$.

(Clause is disjunction of Boolean variables or negations, e.g., $x_1 \vee \overline{x_2} \vee x_3$)

Task: Find a truth assignment to $x_1, \ldots, x_n$.

Objective: Maximize the total weight of satisfied clauses.

Example: $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (x_2 \vee x_3) \wedge (\overline{x_3})$

Remarks:

- A variable x_i or its negation $\overline{x_i}$ is a literal.
- The number of literals ℓ_j in clause C_j is its size or length.
- If $\ell_j = 1$, then C_j is a unit clause.
- W.l.o.g. no literal is repeated in a clause and clauses are distinct.
- W.l.o.g. at most one of x_i and $\overline{x_i}$ appears in a clause.

Randomized Truth Assignment

Theorem 5.2

- a Setting each x_i to true independently with probability $1/2$ gives a randomized $1/2$-approximation algorithm for MAX SAT.
- b If $\ell_j \geq k$ for all $j = 1, \ldots, m$, then the above algorithm is a randomized $(1 - 1/2^k)$-approximation algorithm.

Randomized Truth Assignment

Theorem 5.2

- **a** Setting each x_i to true independently with probability $1/2$ gives a randomized $1/2$-approximation algorithm for MAX SAT.
- **b** If $\ell_j \geq k$ for all $j = 1, \ldots, m$, then the above algorithm is a randomized $(1 - 1/2^k)$-approximation algorithm.

Proof:… □

Randomized Truth Assignment

Theorem 5.2

a. Setting each x_i to true independently with probability $1/2$ gives a randomized $1/2$-approximation algorithm for MAX SAT.

b. If $\ell_j \geq k$ for all $j = 1, \ldots, m$, then the above algorithm is a randomized $(1 - 1/2^k)$-approximation algorithm.

Proof:… □

Maximum Exactly 3SAT (MAX E 3SAT): The special case of MAX SAT where $\ell_j = 3$ for all $j = 1, \ldots, m$ is called MAX E 3SAT.

Randomized Truth Assignment

Theorem 5.2

a. Setting each x_i to true independently with probability $1/2$ gives a randomized $1/2$-approximation algorithm for MAX SAT.

b. If $\ell_j \geq k$ for all $j = 1, \ldots, m$, then the above algorithm is a randomized $(1 - 1/2^k)$-approximation algorithm.

Proof:... □

Maximum Exactly 3SAT (MAX E 3SAT): The special case of MAX SAT where $\ell_j = 3$ for all $j = 1, \ldots, m$ is called MAX E 3SAT.

We state the following theorem without proof.

Randomized Truth Assignment

Theorem 5.2

a. Setting each x_i to true independently with probability $1/2$ gives a randomized $1/2$-approximation algorithm for MAX SAT.

b. If $\ell_j \geq k$ for all $j = 1, \ldots, m$, then the above algorithm is a randomized $(1 - 1/2^k)$-approximation algorithm.

Proof:… □

Maximum Exactly 3SAT (MAX E 3SAT): The special case of MAX SAT where $\ell_j = 3$ for all $j = 1, \ldots, m$ is called MAX E 3SAT.

We state the following theorem without proof.

Theorem 5.3

Unless $P = NP$, there is no $(7/8 + \varepsilon)$-approximation algorithm for MAX E 3SAT for any constant $\varepsilon > 0$.

Maximum Cut Problem (MAX CUT)

Given: Undirected Graph $G = (V, E)$ with edge weights $w_e \geq 0$, $e \in E$.

Task: Find $S \subset V$ maximizing $\sum_{e \in \delta(S)} w_e$.

Maximum Cut Problem (MAX CUT)

Given: Undirected Graph $G = (V, E)$ with edge weights $w_e \geq 0$, $e \in E$.

Task: Find $S \subset V$ maximizing $\sum_{e \in \delta(S)} w_e$.

Theorem 5.4

Placing each node $v \in V$ into S independently at random with probability $1/2$ gives a randomized $1/2$-approximation algorithm for MAX CUT.

Maximum Cut Problem (MAX CUT)

Given: Undirected Graph $G = (V, E)$ with edge weights $w_e \geq 0$, $e \in E$.

Task: Find $S \subset V$ maximizing $\sum_{e \in \delta(S)} w_e$.

Theorem 5.4

Placing each node $v \in V$ into S independently at random with probability $1/2$ gives a randomized $1/2$-approximation algorithm for MAX CUT.

Proof:... □

Derandomization: Method of Conditional Expectations

Basic Idea:

- Consider random decisions sequentially one after another.
- Take next decision deterministically optimizing the expected solution value assuming that all remaining decisions are taken randomly.

Derandomization: Method of Conditional Expectations

Basic Idea:

- Consider random decisions sequentially one after another.
- Take next decision deterministically optimizing the expected solution value assuming that all remaining decisions are taken randomly.

Example: Derandomized version of randomized MAX SATalgorithm

Let W denote the total weight of satisfied clauses in final solution.

Derandomization: Method of Conditional Expectations

Basic Idea:

- Consider random decisions sequentially one after another.
- Take next decision deterministically optimizing the expected solution value assuming that all remaining decisions are taken randomly.

Example: Derandomized version of randomized MAX SATalgorithm

Let W denote the total weight of satisfied clauses in final solution.

for $i = 1$ to n
    if $\mathrm{E}[W \mid x_1 = b_1, \ldots, x_{i-1} = b_{i-1}, x_i = \text{true}] \geq \mathrm{E}[W \mid x_1 = b_1, \ldots, x_{i-1} = b_{i-1}, x_i = \text{false}]$
    then set $b_i :=$ true;
    else set $b_i := \mathit{false}$;
return x:=b;

Method of Conditional Expectations: Analysis

Theorem 5.5

The value of the solution computed by the deterministic MAX SAT algorithm is at least the expected value of the randomized solution.

Method of Conditional Expectations: Analysis

Theorem 5.5

The value of the solution computed by the deterministic MAX SAT algorithm is at least the expected value of the randomized solution.

Remarks.

- The crucial step of the derandomized algorithm is to compute the conditional expectations.

Method of Conditional Expectations: Analysis

Theorem 5.5

The value of the solution computed by the deterministic MAX SAT algorithm is at least the expected value of the randomized solution.

Remarks.

- The crucial step of the derandomized algorithm is to compute the conditional expectations.
- Notice that $\mathrm{E}[W \mid x_1 = b_1, \ldots, x_i = b_i]$

$$= \sum_{j=1}^{m} w_j \cdot \Pr[C_j = \text{true} \mid x_1 = b_1, \ldots, x_i = b_i]$$

Method of Conditional Expectations: Analysis

Theorem 5.5

The value of the solution computed by the deterministic MAX SAT algorithm is at least the expected value of the randomized solution.

Remarks.

- The crucial step of the derandomized algorithm is to compute the conditional expectations.
- Notice that $\mathrm{E}[W \mid x_1 = b_1, \ldots, x_i = b_i]$

$$= \sum_{j=1}^{m} w_j \cdot \Pr[C_j = \text{true} \mid x_1 = b_1, \ldots, x_i = b_i]$$

and $\Pr[C_j = \text{true} \mid x_1 = b_1, \ldots, x_i = b_i]$

$$= \begin{cases} 1 & \text{if } x_1 = b_1, \ldots, x_i = b_i \text{ satisfies } C_j, \\ 1 - 1/2^k & \text{else,} \end{cases}$$

where k is the number of remaining literals in clause C_j.

Flipping Biased Coins

We first restrict to MAX SAT instances with no negated unit clause.

Flipping Biased Coins

We first restrict to MAX SAT instances with no negated unit clause.

Lemma 5.6

If each x_i is independently set to true with probability $p > 1/2$, then the probability that a clause is satisfied is at least $\min\{p, 1 - p^2\}$.

Flipping Biased Coins

We first restrict to MAX SAT instances with no negated unit clause.

Lemma 5.6

If each x_i is independently set to true with probability $p > 1/2$, then the probability that a clause is satisfied is at least $\min\{p, 1 - p^2\}$.

Proof:… □

Flipping Biased Coins

We first restrict to MAX SAT instances with no negated unit clause.

Lemma 5.6

If each x_i is independently set to true with probability $p > 1/2$, then the probability that a clause is satisfied is at least $\min\{p, 1 - p^2\}$.

Proof:… □

Theorem 5.7

For $1/2 < p \leq 1$ this gives a randomized $\min\{p, 1 - p^2\}$-approximation algorithm for MAX SAT. □

Flipping Biased Coins

We first restrict to MAX SAT instances with no negated unit clause.

Lemma 5.6

If each x_i is independently set to true with probability $p > 1/2$, then the probability that a clause is satisfied is at least $\min\{p, 1 - p^2\}$.

Proof:... □

Theorem 5.7

For $1/2 < p \leq 1$ this gives a randomized $\min\{p, 1 - p^2\}$-approximation algorithm for MAX SAT. □

Notice: For $p = (\sqrt{5} - 1)/2$ we get $\min\{p, 1 - p^2\} = (\sqrt{5} - 1)/2 \approx 0.618$.

Flipping Biased Coins

We first restrict to MAX SAT instances with no negated unit clause.

Lemma 5.6

If each x_i is independently set to true with probability $p > 1/2$, then the probability that a clause is satisfied is at least $\min\{p, 1-p^2\}$.

Proof:… □

Theorem 5.7

For $1/2 < p \leq 1$ this gives a randomized $\min\{p, 1-p^2\}$-approximation algorithm for MAX SAT. □

Notice: For $p = (\sqrt{5}-1)/2$ we get $\min\{p, 1-p^2\} = (\sqrt{5}-1)/2 \approx 0.618$.

Remark:

The initial assumption on the absence of negated unit clauses holds w.l.o.g. !

Outline

Integer Programming Formulation for MAX SAT

For $j = 1, \ldots, m$ let $P_j := \{i \mid \text{literal } x_i \text{ occurs in } C_j\}$
and $N_j := \{i \mid \text{literal } \overline{x_i} \text{ occurs in } C_j\}$.

That is,

$$C_j = \bigvee_{i \in P_j} x_i \vee \bigvee_{i \in N_j} \overline{x_i}.$$

Integer Programming Formulation for MAX SAT

For $j = 1, \ldots, m$ let $P_j := \{i \mid \text{literal } x_i \text{ occurs in } C_j\}$
and $N_j := \{i \mid \text{literal } \overline{x_i} \text{ occurs in } C_j\}$.

That is,

$$C_j = \bigvee_{i \in P_j} x_i \vee \bigvee_{i \in N_j} \overline{x_i}.$$

IP formulation:

$$\begin{aligned}
\max \quad & \sum_{j=1}^{m} w_j \cdot z_j \\
\text{s.t.} \quad & \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j && \text{for all } j = 1, \ldots, m, \\
& y_i \in \{0, 1\} && \text{for all } i = 1, \ldots, n, \\
& 0 \leq z_j \leq 1 && \text{for all } j = 1, \ldots, m.
\end{aligned}$$

Integer Programming Formulation for MAX SAT

For $j = 1, \ldots, m$ let $P_j := \{i \mid \text{literal } x_i \text{ occurs in } C_j\}$
and $N_j := \{i \mid \text{literal } \overline{x_i} \text{ occurs in } C_j\}$.

That is,
$$C_j = \bigvee_{i \in P_j} x_i \vee \bigvee_{i \in N_j} \overline{x_i}.$$

IP formulation:

$$\begin{array}{lll}
\max & \sum_{j=1}^{m} w_j \cdot z_j & \\
\text{s.t.} & \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j & \text{for all } j = 1, \ldots, m, \\
& y_i \in \{0, 1\} & \text{for all } i = 1, \ldots, n, \\
& 0 \leq z_j \leq 1 & \text{for all } j = 1, \ldots, m.
\end{array}$$

LP relaxation: Replace $y_i \in \{0, 1\}$ with $0 \leq y_i \leq 1$ for all $i = 1, \ldots, n$.

Randomized Rounding

compute an optimal solution (y^*, z^*) to the LP relaxation;
for $i = 1$ to n do
    set x_i to true independently at random with probability y_i^*;

Randomized Rounding

1. compute an optimal solution (y^*, z^*) to the LP relaxation;
2. for $i = 1$ to n do
3. set x_i to true independently at random with probability y_i^*;

Theorem 5.8

Randomized rounding gives a randomized $(1 - 1/e)$-approximation algorithm for MAX SAT.

Randomized Rounding

1. compute an optimal solution (y^*, z^*) to the LP relaxation;
2. for $i = 1$ to n do
3. set x_i to true independently at random with probability y_i^*;

Theorem 5.8

Randomized rounding gives a randomized $(1 - 1/e)$-approximation algorithm for MAX SAT.

Proof:… □

Randomized Rounding

1. compute an optimal solution (y^*, z^*) to the LP relaxation;
2. for $i = 1$ to n do
3. set x_i to true independently at random with probability y_i^*;

Theorem 5.8

Randomized rounding gives a randomized $(1 - 1/e)$-approximation algorithm for MAX SAT.

Proof:… □

Remark.
Algorithm can be derandomized by method of conditional expectations.

Choosing the Better of Two Solutions

Theorem 5.9

Running either the unbiased randomized $1/2$-approximation algorithm or the randomized rounding algorithm, both with probability $1/2$, yields a randomized $3/4$-approximation algorithm.

Choosing the Better of Two Solutions

Theorem 5.9

Running either the unbiased randomized $1/2$-approximation algorithm or the randomized rounding algorithm, both with probability $1/2$, yields a randomized $3/4$-approximation algorithm.

Proof: Consider clause C_j of length ℓ_j:

- 1st algorithm: $\Pr[C_j = \text{true}] = 1 - 1/2^{\ell_j}$.
- 2nd algorithm: $\Pr[C_j = \text{true}] \geq \left(1 - (1 - 1/\ell_j)^{\ell_j}\right) z_j^*$.

… □

Choosing the Better of Two Solutions

Theorem 5.9

Running either the unbiased randomized $1/2$-approximation algorithm or the randomized rounding algorithm, both with probability $1/2$, yields a randomized $3/4$-approximation algorithm.

Proof: Consider clause C_j of length ℓ_j:

- 1st algorithm: $\Pr[C_j = \text{true}] = 1 - 1/2^{\ell_j}$.
- 2nd algorithm: $\Pr[C_j = \text{true}] \geq \left(1 - (1 - 1/\ell_j)^{\ell_j}\right) z_j^*$.

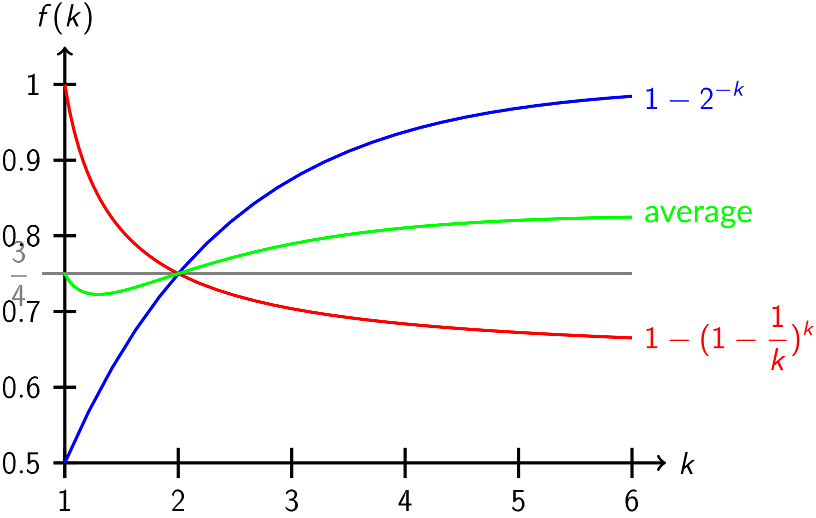
... □

Derandomizing the initial coin flip yields:

Corollary 5.10

Running both algorithms and choosing the better of the two solutions is a randomized $3/4$-approximation algorithm. □

Visualization of Proof of Theorem 5.9



Non-linear Randomized Rounding

Consider a function $f : [0,1] \to [0,1]$.

1. compute an optimal solution (y^*, z^*) to the LP relaxation;
2. for $i = 1$ to n do
3. set x_i to true independently at random with probability $f(y_i^*)$;

Non-linear Randomized Rounding

Consider a function $f : [0, 1] \to [0, 1]$.

1. compute an optimal solution (y^*, z^*) to the LP relaxation;
2. for $i = 1$ to n do
3. set x_i to true independently at random with probability $f(y_i^*)$;

Theorem 5.11

Let $f : [0, 1] \to [0, 1]$ with $1 - 4^{-x} \leq f(x) \leq 4^{x-1}$ for all $x \in [0, 1]$. Then non-linear randomized rounding with function f is a randomized $3/4$-approximation algorithm.

Non-linear Randomized Rounding

Consider a function $f : [0,1] \to [0,1]$.

1. compute an optimal solution (y^*, z^*) to the LP relaxation;
2. for $i = 1$ to n do
3. set x_i to true independently at random with probability $f(y_i^*)$;

Theorem 5.11

Let $f : [0,1] \to [0,1]$ with $1 - 4^{-x} \leq f(x) \leq 4^{x-1}$ for all $x \in [0,1]$. Then non-linear randomized rounding with function f is a randomized $3/4$-approximation algorithm.

Proof:… □

Non-linear Randomized Rounding

Consider a function $f : [0,1] \rightarrow [0,1]$.

1. compute an optimal solution (y^*, z^*) to the LP relaxation;
2. for $i = 1$ to n do
3.   set x_i to true independently at random with probability $f(y_i^*)$;

Theorem 5.11

Let $f : [0,1] \rightarrow [0,1]$ with $1 - 4^{-x} \leq f(x) \leq 4^{x-1}$ for all $x \in [0,1]$. Then non-linear randomized rounding with function f is a randomized $3/4$-approximation algorithm.

Proof:… □

Remark:

- The integrality gap of the LP relaxation for MAX SAT is $3/4$.

Non-linear Randomized Rounding

Consider a function $f : [0, 1] \rightarrow [0, 1]$.

1. compute an optimal solution (y^*, z^*) to the LP relaxation;
2. for $i = 1$ to n do
3. set x_i to true independently at random with probability $f(y_i^*)$;

Theorem 5.11

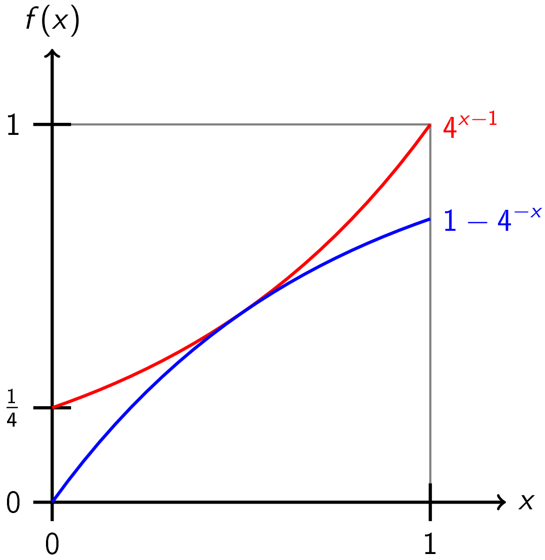
Let $f : [0, 1] \rightarrow [0, 1]$ with $1 - 4^{-x} \leq f(x) \leq 4^{x-1}$ for all $x \in [0, 1]$. Then non-linear randomized rounding with function f is a randomized $3/4$-approximation algorithm.

Proof:… □

Remark:

- The integrality gap of the LP relaxation for MAX SAT is $3/4$.
- Thus, $3/4$ is best performance ratio one can prove based on the LP.

Visualization of Lower and Upper Bound on f



Outline

Randomized Algo for Prize-Collecting Steiner Trees

Idea:

- Obtain randomized variant of deterministic LP rounding algorithm from Chapter 4 by choosing α randomly.

Randomized Algo for Prize-Collecting Steiner Trees

Idea:

- Obtain randomized variant of deterministic LP rounding algorithm from Chapter 4 by choosing α randomly.
- For some fixed $\gamma > 0$ choose α uniformly at random from $[\gamma, 1]$.

Randomized Algo for Prize-Collecting Steiner Trees

Idea:

- Obtain randomized variant of deterministic LP rounding algorithm from Chapter 4 by choosing α randomly.
- For some fixed $\gamma > 0$ choose α uniformly at random from $[\gamma, 1]$.
- That is, choose α from $[\gamma, 1]$ with constant density function $1/(1-\gamma)$.

Randomized Algo for Prize-Collecting Steiner Trees

Idea:

- Obtain randomized variant of deterministic LP rounding algorithm from Chapter 4 by choosing α randomly.
- For some fixed $\gamma > 0$ choose α uniformly at random from $[\gamma, 1]$.
- That is, choose α from $[\gamma, 1]$ with constant density function $1/(1-\gamma)$.

Lemma 5.12

The tree T returned by the randomized algorithm has expected cost

$$\mathrm{E}\left[\sum_{e\in E(T)} c_e\right] \leq \frac{2}{1-\gamma}\ln\frac{1}{\gamma}\sum_{e\in E} c_e \cdot x_e^* \quad .$$

Randomized Algo for Prize-Collecting Steiner Trees

Idea:

- Obtain randomized variant of deterministic LP rounding algorithm from Chapter 4 by choosing α randomly.
- For some fixed $\gamma > 0$ choose α uniformly at random from $[\gamma, 1]$.
- That is, choose α from $[\gamma, 1]$ with constant density function $1/(1-\gamma)$.

Lemma 5.12

The tree T returned by the randomized algorithm has expected cost

$$\mathrm{E}\left[\sum_{e \in E(T)} c_e\right] \leq \frac{2}{1-\gamma} \ln \frac{1}{\gamma} \sum_{e \in E} c_e \cdot x_e^* \quad .$$

Proof:… □

Randomized Algo for Prize-Collecting Steiner Trees

Lemma 5.13

The expected penalty costs are

$$\mathsf{E}\left[\sum_{i\in V\setminus V(T)} \pi_i\right] \leq \frac{1}{1-\gamma}\sum_{i\in V}\pi_i \cdot (1-y_i^*) \enspace .$$

Randomized Algo for Prize-Collecting Steiner Trees

Lemma 5.13

The expected penalty costs are

$$\mathsf{E}\left[\sum_{i \in V \setminus V(T)} \pi_i\right] \leq \frac{1}{1-\gamma} \sum_{i \in V} \pi_i \cdot (1 - y_i^*) \ .$$

Proof:… □

Randomized Algo for Prize-Collecting Steiner Trees

Lemma 5.13

The expected penalty costs are

$$\mathrm{E}\left[\sum_{i\in V\setminus V(T)} \pi_i\right] \le \frac{1}{1-\gamma}\sum_{i\in V}\pi_i\cdot(1-y_i^*)\ .$$

Proof:... □

Theorem 5.14

For $\gamma := e^{-1/2}$ the expected cost of the solution is

$$\mathrm{E}\left[\sum_{e\in E(T)} c_e + \sum_{i\in V\setminus V(T)} \pi_i\right] \le \frac{1}{1-1/\sqrt{e}}\cdot \mathtt{OPT}_{LP}\ .$$

Thus, we have a randomized 2.54-approximation algorithm. □

Derandomization and Integrality Gap

Derandomization.

- There are at most $n := |V|$ distinct values of y_i^*.
- Consider n sets $U_j := \{i \in V \mid y_i^* \geq y_j^*\}$, for $j = 1, \ldots, n$.
- Any possible value of α corresponds to one of these n sets.
- Derandomize by trying each set U_j and choosing the best solution.

Derandomization and Integrality Gap

Derandomization.

- There are at most $n := |V|$ distinct values of y_i^*.
- Consider n sets $U_j := \{i \in V \mid y_i^* \geq y_j^*\}$, for $j = 1, \ldots, n$.
- Any possible value of α corresponds to one of these n sets.
- Derandomize by trying each set U_j and choosing the best solution.

Integrality gap.

- There exist instances with integrality gap $2 - \dfrac{2}{n}$.

Derandomization and Integrality Gap

Derandomization.

- There are at most $n := |V|$ distinct values of y_i^*.
- Consider n sets $U_j := \{i \in V \mid y_i^* \geq y_j^*\}$, for $j = 1, \ldots, n$.
- Any possible value of α corresponds to one of these n sets.
- Derandomize by trying each set U_j and choosing the best solution.

Integrality gap.

- There exist instances with integrality gap $2 - \frac{2}{n}$.
- By Theorem 5.14 the integrality gap is at most $\frac{1}{1 - 1/\sqrt{e}} \approx 2.54$.

Derandomization and Integrality Gap

Derandomization.

- There are at most $n := |V|$ distinct values of y_i^*.
- Consider n sets $U_j := \{i \in V \mid y_i^* \geq y_j^*\}$, for $j = 1, \ldots, n$.
- Any possible value of α corresponds to one of these n sets.
- Derandomize by trying each set U_j and choosing the best solution.

Integrality gap.

- There exist instances with integrality gap $2 - \frac{2}{n}$.
- By Theorem 5.14 the integrality gap is at most $\frac{1}{1 - 1/\sqrt{e}} \approx 2.54$.
- We will prove later that the integrality gap is at most 2.

Outline

Randomized Algo for Uncapacitated Facility Location

In Chapter 4 we obtained an LP-based 4-approximation algorithm which computes a solution of cost at most

$$\sum_{i \in F} f_i \cdot y_i^* + 3 \cdot \sum_{j \in D} v_j^* \quad .$$

Randomized Algo for Uncapacitated Facility Location

In Chapter 4 we obtained an LP-based 4-approximation algorithm which computes a solution of cost at most

$$\sum_{i \in F} f_i \cdot y_i^* + 3 \cdot \sum_{j \in D} v_j^* \quad .$$

Notation.
Let $C_j^* := \sum_{i \in F} c_{ij} \cdot x_{ij}^*$ denote the assignment cost of j paid by the LP, i.e.,

$$\mathtt{OPT}_{LP} = \sum_{i \in F} f_i \cdot y_i^* + \sum_{j \in D} C_j^* \quad .$$

Randomized Algo for Uncapacitated Facility Location

In Chapter 4 we obtained an LP-based 4-approximation algorithm which computes a solution of cost at most

$$\sum_{i \in F} f_i \cdot y_i^* + 3 \cdot \sum_{j \in D} v_j^* \quad .$$

Notation.

Let $C_j^* := \sum_{i \in F} c_{ij} \cdot x_{ij}^*$ denote the assignment cost of j paid by the LP, i.e.,

$$\mathtt{OPT}_{LP} = \sum_{i \in F} f_i \cdot y_i^* + \sum_{j \in D} C_j^* \quad .$$

Idea:

- Include the assignment cost C_j^* in the analysis.
- Instead of bounding only the facility cost by $\mathtt{OPT}_{LP}$, bound both the facility cost and part of the assignment cost by $\mathtt{OPT}_{LP}$.

Randomized Algorithm for Uncapacitated Facility Location

Randomized algorithm for Uncapacitated Facility Location Problem

```
compute optimal LP solutions (x*, y*) and (v*, w*);
while D ≠ ∅
        choose j := argmin_{j'∈D}(v*_{j'} + C*_{j'});
        choose i ∈ N(j) according to probability distribution x*_{ij};
        assign all unassigned clients in N^2(j) to facility i;
        set D := D \ N^2(j);
```

Randomized Algorithm for Uncapacitated Facility Location

Randomized algorithm for Uncapacitated Facility Location Problem

1. compute optimal LP solutions (x^*, y^*) and (v^*, w^*);
2. while $D \neq \emptyset$
3. choose $j := \text{argmin}_{j' \in D}(v_{j'}^* + C_{j'}^*)$;
4. choose $i \in N(j)$ according to probability distribution x_{ij}^*;
5. assign all unassigned clients in $N^2(j)$ to facility i;
6. set $D := D \setminus N^2(j)$;

Theorem 5.15

The algorithm above is a randomized 3-approximation algorithm for the Uncapacitated Facility Location Problem.

Randomized Algorithm for Uncapacitated Facility Location

Randomized algorithm for Uncapacitated Facility Location Problem

1. compute optimal LP solutions (x^*, y^*) and (v^*, w^*);
2. while $D \neq \emptyset$
3. choose $j := \text{argmin}_{j' \in D}(v^*_{j'} + C^*_{j'})$;
4. choose $i \in N(j)$ according to probability distribution x^*_{ij};
5. assign all unassigned clients in $N^2(j)$ to facility i;
6. set $D := D \setminus N^2(j)$;

Theorem 5.15

The algorithm above is a randomized 3-approximation algorithm for the Uncapacitated Facility Location Problem.

Proof:… □

Outline

Min Weighted Sum of Completion Times $1|r_j|\sum w_j C_j$

Given: jobs with processing time $p_j \in \mathbb{Z}_{>0}$, weight $w_j \geq 0$, and release date $r_j \in \mathbb{Z}_{\geq 0}$, $j = 1, \ldots, n$.

Task: Schedule the jobs nonpreemptively on a single machine; minimize the total weighted completion time $\sum_{j=1}^{n} w_j \cdot C_j$.

Min Weighted Sum of Completion Times $1|r_j|\sum w_j C_j$

Given: jobs with processing time $p_j \in \mathbb{Z}_{>0}$, weight $w_j \geq 0$, and release date $r_j \in \mathbb{Z}_{\geq 0}$, $j = 1, \ldots, n$.

Task: Schedule the jobs nonpreemptively on a single machine; minimize the total weighted completion time $\sum_{j=1}^{n} w_j \cdot C_j$.

Let $T := \max_j r_j + \sum_{j=1}^{n} p_j$ (upper bound on all completion times).

Min Weighted Sum of Completion Times $1|r_j|\sum w_j C_j$

Given: jobs with processing time $p_j \in \mathbb{Z}_{>0}$, weight $w_j \geq 0$,
and release date $r_j \in \mathbb{Z}_{\geq 0}$, $j = 1, \ldots, n$.

Task: Schedule the jobs nonpreemptively on a single machine;
minimize the total weighted completion time $\sum_{j=1}^{n} w_j \cdot C_j$.

Let $T := \max_j r_j + \sum_{j=1}^{n} p_j$ (upper bound on all completion times).

Consider an integer programming relaxation with variables

$$y_{jt} = \begin{cases} 1 & \text{if job } j \text{ is processed in time } [t-1, t), \\ 0 & \text{otherwise} \end{cases}$$

for $j = 1, \ldots, n$, $t = 1, \ldots, T$.

Integer Programming Relaxation

$$\begin{aligned}
\min \quad & \sum_{j=1}^{n} w_j \cdot C_j & \\
\text{s.t.} \quad & \sum_{j=1}^{n} y_{jt} \leq 1 & \text{for } t = 1, \ldots, T, \\
& \sum_{t=1}^{T} y_{jt} = p_j & \text{for } j = 1, \ldots, n, \\
& y_{jt} = 0 & \text{for } j = 1, \ldots, n,\ t = 1, \ldots, r_j, \\
& C_j = \frac{1}{p_j} \sum_{t=1}^{T} y_{jt}\big(t - \tfrac{1}{2}\big) + \tfrac{1}{2} p_j & \text{for } j = 1, \ldots, n, \\
& y_{jt} \in \{0, 1\} & \text{for } j = 1, \ldots, n,\ t = 1, \ldots, T.
\end{aligned}$$

Integer Programming Relaxation

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{n} w_j \cdot C_j \\
\text{s.t.} \quad & \sum_{j=1}^{n} y_{jt} \leq 1 && \text{for } t = 1, \ldots, T, \\
& \sum_{t=1}^{T} y_{jt} = p_j && \text{for } j = 1, \ldots, n, \\
& y_{jt} = 0 && \text{for } j = 1, \ldots, n,\ t = 1, \ldots, r_j, \\
& C_j = \frac{1}{p_j} \sum_{t=1}^{T} y_{jt}\left(t - \tfrac{1}{2}\right) + \tfrac{1}{2} p_j && \text{for } j = 1, \ldots, n, \\
& y_{jt} \in \{0, 1\} && \text{for } j = 1, \ldots, n,\ t = 1, \ldots, T.
\end{aligned}
$$

Remarks.

- Notice that in a feasible IP solution jobs might be preempted.

Integer Programming Relaxation

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{n} w_j \cdot C_j & \\
\text{s.t.} \quad & \sum_{j=1}^{n} y_{jt} \leq 1 & \text{for } t = 1, \dots, T, \\
& \sum_{t=1}^{T} y_{jt} = p_j & \text{for } j = 1, \dots, n, \\
& y_{jt} = 0 & \text{for } j = 1, \dots, n,\ t = 1, \dots, r_j, \\
& C_j = \frac{1}{p_j} \sum_{t=1}^{T} y_{jt} \big(t - \tfrac{1}{2}\big) + \tfrac{1}{2} p_j & \text{for } j = 1, \dots, n, \\
& y_{jt} \in \{0, 1\} & \text{for } j = 1, \dots, n,\ t = 1, \dots, T.
\end{aligned}
$$

Remarks.

- Notice that in a feasible IP solution jobs might be preempted.
- In this case, C_j underestimates the actual completion time of job j.

Randomized Rounding

1. compute optimal IP solution (y^*, C^*);
2. for $j = 1$ to n set random variable X_j to $t - \frac{1}{2}$ with probability y^*_{jt}/p_j;
3. sort the jobs such that $X_1 \leq X_2 \leq \cdots \leq X_n$;
4. schedule all jobs nonpreemptively and as early as possible in this order;

Randomized Rounding

1. compute optimal IP solution (y^*, C^*);
2. for $j = 1$ to n set random variable X_j to $t - \frac{1}{2}$ with probability y^*_{jt}/p_j;
3. sort the jobs such that $X_1 \leq X_2 \leq \cdots \leq X_n$;
4. schedule all jobs nonpreemptively and as early as possible in this order;

Lemma 5.16

If the random variables X_j are independent, then $\mathrm{E}\left[C_j \mid X_j = x\right] \leq p_j + 2x$.

Randomized Rounding

1. compute optimal IP solution (y^*, C^*);
2. for $j = 1$ to n set random variable X_j to $t - \frac{1}{2}$ with probability y^*_{jt}/p_j;
3. sort the jobs such that $X_1 \leq X_2 \leq \cdots \leq X_n$;
4. schedule all jobs nonpreemptively and as early as possible in this order;

Lemma 5.16

If the random variables X_j are independent, then
$\mathrm{E}\left[C_j \mid X_j = x\right] \leq p_j + 2x$.

Theorem 5.17

The expected performance ratio of the randomized algorithm is at most 2.

Computing an Optimum IP Solution

1. sort the jobs such that $w_1/p_1 \geq w_2/p_2 \geq \cdots \geq w_n/p_n$;
2. construct a preemptive schedule:
3. $\rightarrow$ always schedule the first available job which is not yet completed;
4. implicitely assign the variables y_{jt} (and C_j) accordingly;

Computing an Optimum IP Solution

1. sort the jobs such that $w_1/p_1 \geq w_2/p_2 \geq \cdots \geq w_n/p_n$;
2. construct a preemptive schedule:
3. $\rightarrow$ always schedule the first available job which is not yet completed;
4. implicitely assign the variables y_{jt} (and C_j) accordingly;

Lemma 5.18

The algorithm finds an optimal IP solution in polynomial time.

Computing an Optimum IP Solution

1. sort the jobs such that $w_1/p_1 \geq w_2/p_2 \geq \cdots \geq w_n/p_n$;
2. construct a preemptive schedule:
3. $\rightarrow$ always schedule the first available job which is not yet completed;
4. implicitely assign the variables y_{jt} (and C_j) accordingly;

Lemma 5.18

The algorithm finds an optimal IP solution in polynomial time.

Proof: Exchange argument… □

Computing an Optimum IP Solution

1. sort the jobs such that $w_1/p_1 \geq w_2/p_2 \geq \cdots \geq w_n/p_n$;
2. construct a preemptive schedule:
3. $\rightarrow$ always schedule the first available job which is not yet completed;
4. implicitely assign the variables y_{jt} (and C_j) accordingly;

Lemma 5.18

The algorithm finds an optimal IP solution in polynomial time.

Proof: Exchange argument… □

Remarks.

- This schedule consists of at most $2n$ intervals of time.

Computing an Optimum IP Solution

1. sort the jobs such that $w_1/p_1 \geq w_2/p_2 \geq \cdots \geq w_n/p_n$;
2. construct a preemptive schedule:
3. $\rightarrow$ always schedule the first available job which is not yet completed;
4. implicitely assign the variables y_{jt} (and C_j) accordingly;

Lemma 5.18

The algorithm finds an optimal IP solution in polynomial time.

Proof: Exchange argument… $\square$

Remarks.

- This schedule consists of at most $2n$ intervals of time.
- Randomized rounding can be implemented to run in polytime.

Computing an Optimum IP Solution

1. sort the jobs such that $w_1/p_1 \geq w_2/p_2 \geq \cdots \geq w_n/p_n$;
2. construct a preemptive schedule:
3. $\rightarrow$ always schedule the first available job which is not yet completed;
4. implicitely assign the variables y_{jt} (and C_j) accordingly;

Lemma 5.18

The algorithm finds an optimal IP solution in polynomial time.

Proof: Exchange argument… □

Remarks.

- This schedule consists of at most $2n$ intervals of time.
- Randomized rounding can be implemented to run in polytime.
- Derandomization (of a variant) of this algo by method of conditional expectations.

Outline

Minimum-Capacity Multicommodity Flow Problem

Given: Undirected graph $G = (V, E)$ and k pairs $s_i, t_i \in V$, $i = 1, \ldots, k$.

Task: Find single s_i-t_i-path in G, for $i = 1, \ldots, k$.

Objective: Minimize maximum number of paths containing same edge.

Minimum-Capacity Multicommodity Flow Problem

Given: Undirected graph $G = (V, E)$ and k pairs $s_i, t_i \in V$, $i = 1, \ldots, k$.

Task: Find single s_i-t_i-path in G, for $i = 1, \ldots, k$.

Objective: Minimize maximum number of paths containing same edge.

Path-based IP formulation: Let $\mathcal{P}_i := \{P \mid P \text{ is } s_i\text{-}t_i\text{-path}\}$.

$$
\begin{aligned}
\min \quad & W \\
\text{s.t.} \quad & \sum_{P \in \mathcal{P}_i} x_P = 1 && \text{for all } i = 1, \ldots, k, \\
& \sum_{P : e \in P} x_P \leq W && \text{for all } e \in E, \\
& x_P \in \{0, 1\} && \text{for all } P \in \mathcal{P}_i,\ i = 1, \ldots, k.
\end{aligned}
$$

Minimum-Capacity Multicommodity Flow Problem

Given: Undirected graph $G = (V, E)$ and k pairs $s_i, t_i \in V$, $i = 1, \ldots, k$.

Task: Find single s_i-t_i-path in G, for $i = 1, \ldots, k$.

Objective: Minimize maximum number of paths containing same edge.

Path-based IP formulation: Let $\mathcal{P}_i := \{P \mid P \text{ is } s_i\text{-}t_i\text{-path}\}$.

$$\begin{aligned}
\min \quad & W \\
\text{s.t.} \quad & \sum_{P \in \mathcal{P}_i} x_P = 1 && \text{for all } i = 1, \ldots, k, \\
& \sum_{P : e \in P} x_P \leq W && \text{for all } e \in E, \\
& x_P \in \{0, 1\} && \text{for all } P \in \mathcal{P}_i,\ i = 1, \ldots, k.
\end{aligned}$$

LP relaxation: Replace $x_P \in \{0, 1\}$ with $x_P \geq 0$.

Minimum-Capacity Multicommodity Flow Problem

Given: Undirected graph $G = (V, E)$ and k pairs $s_i, t_i \in V$, $i = 1, \ldots, k$.

Task: Find single s_i-t_i-path in G, for $i = 1, \ldots, k$.

Objective: Minimize maximum number of paths containing same edge.

Path-based IP formulation: Let $\mathcal{P}_i := \{P \mid P \text{ is } s_i\text{-}t_i\text{-path}\}$.

$$
\begin{aligned}
\min \quad & W \\
\text{s.t.} \quad & \sum_{P \in \mathcal{P}_i} x_P = 1 && \text{for all } i = 1, \ldots, k, \\
& \sum_{P : e \in P} x_P \leq W && \text{for all } e \in E, \\
& x_P \in \{0, 1\} && \text{for all } P \in \mathcal{P}_i,\ i = 1, \ldots, k.
\end{aligned}
$$

LP relaxation: Replace $x_P \in \{0, 1\}$ with $x_P \geq 0$.

- Despite exponential number of variables, LP relaxation can be solved in polynomial time!

Randomized Rounding

1. compute optimal LP solution (x^*, W^*);
2. for $i = 1$ to k
3. independently choose one path $P \in \mathcal{P}_i$ with probability x_P^*;

Randomized Rounding

1. compute optimal LP solution (x^*, W^*);
2. for $i = 1$ to k
3. independently choose one path $P \in \mathcal{P}_i$ with probability x^*_P;

Definition 5.19

A probabilistic event happens with high probability if the probability that it does not occur is at most n^{-c} for some constant $c \geq 1$.

Theorem 5.20

If $W^* \geq c \cdot \ln n$ for a large enough constant c, then with high probability, the total number of paths using any edge is at most $W^* + \sqrt{c \cdot W^* \ln n}$.

Markov's Inequality and Chernoff Bound

Lemma 5.21 (Markov's Inequality)

If $X \geq 0$ is a random variable, then $\Pr[X \geq a] \leq \mathrm{E}[X]/a$ for $a > 0$. $\square$

Markov's Inequality and Chernoff Bound

Lemma 5.21 (Markov's Inequality)

If $X \geq 0$ is a random variable, then $\Pr[X \geq a] \leq \mathrm{E}[X]/a$ for $a > 0$. □

Proof:

… □

Markov's Inequality and Chernoff Bound

Lemma 5.21 (Markov's Inequality)

If $X \geq 0$ is a random variable, then $\Pr[X \geq a] \leq \mathrm{E}[X]/a$ for $a > 0$. □

Proof:

… □

Theorem 5.22 (Chernoff Bound)

Let $X_1, \ldots, X_k$ be independent 0-1 random variables. Then for $X := \sum_{i=1}^{k} X_i$, $\mu \geq \mathrm{E}[X]$, and $0 < \delta \leq 1$

$$\Pr[X \geq (1+\delta)\cdot\mu] < \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu} \leq e^{-\mu\cdot\delta^2/3} \quad .$$

Proof idea: Apply Markov inequality to the event $\Pr\left[e^{tX} \geq e^{t(1+\delta)\mu}\right]$ for a well-chosen value of t.

Performance Guarantees

Corollary 5.23

- a If $W^* \geq c \cdot \ln n$, then randomized rounding with high probability produces a solution of value at most $2W^*$.
- b If $W^* \geq 1$, then with high probability the total number of paths using any edge is $O(\log n) \cdot W^*$.

Performance Guarantees

Corollary 5.23

- a. If $W^* \geq c \cdot \ln n$, then randomized rounding with high probability produces a solution of value at most $2W^*$.
- b. If $W^* \geq 1$, then with high probability the total number of paths using any edge is $O(\log n) \cdot W^*$.

Proof:. . . □

Performance Guarantees

Corollary 5.23

a. If $W^* \geq c \cdot \ln n$, then randomized rounding with high probability produces a solution of value at most $2W^*$.

b. If $W^* \geq 1$, then with high probability the total number of paths using any edge is $O(\log n) \cdot W^*$.

Proof:… □

Remarks.

- The statement of Corollary 5.23 can be sharpened by replacing the term $O(\log n)$ with $O(\log n / \log\log n)$.

Performance Guarantees

Corollary 5.23

a. If $W^* \geq c \cdot \ln n$, then randomized rounding with high probability produces a solution of value at most $2W^*$.

b. If $W^* \geq 1$, then with high probability the total number of paths using any edge is $O(\log n) \cdot W^*$.

Proof:... □

Remarks.

- The statement of Corollary 5.23 can be sharpened by replacing the term $O(\log n)$ with $O(\log n / \log \log n)$.
- On the other hand, the integrality gap of the IP formulation is in $\Omega(\log n / \log \log n)$.

Outline

Semidefinite Matrices

Definition 5.24

A symmetric matrix $X \in \mathbb{R}^{n \times n}$ is positive semidefinite if $y^T \cdot X \cdot y \geq 0$ for all $y \in \mathbb{R}^n$. In this case we write $X \succeq 0$.

Semidefinite Matrices

Definition 5.24

A symmetric matrix $X \in \mathbb{R}^{n \times n}$ is positive semidefinite if $y^T \cdot X \cdot y \geq 0$ for all $y \in \mathbb{R}^n$. In this case we write $X \succeq 0$.

Theorem 5.25

For a symmetric $X \in \mathbb{R}^{n \times n}$ the following statements are equivalent:

i. X is positive semidefinite;
ii. all eigenvalues of X are non-negative;
iii. $X = V^T \cdot V$ for some $V \in \mathbb{R}^{m \times n}$ where $m \leq n$;
iv. $X = \sum_{i=1}^{n} \lambda_i (w_i \cdot w_i^T)$ for some $\lambda_i \geq 0$ and $w_i \in \mathbb{R}^n$ such that $w_i^T \cdot w_i = 1$ and $w_i^T \cdot w_j = 0$ for $i \neq j$. □

Semidefinite Programs (SDPs)

Definition 5.26

A semidefinite program is a linear program with the additional constraint that a square symmetric matrix of variables must be positive semidefinite.

Semidefinite Programs (SDPs)

Definition 5.26

A semidefinite program is a linear program with the additional constraint that a square symmetric matrix of variables must be positive semidefinite.

Example.

$$\begin{aligned} \min \,/\, \max \quad & \sum_{i,j} c_{ij} x_{ij} & \\ \text{s.t.} \quad & \sum_{i,j} a_{ijk} x_{ij} = b_k & \text{for all } k, \\ & x_{ij} = x_{ji} & \text{for all } i,j, \\ & X = (x_{ij}) \succeq 0 & \end{aligned}$$

Semidefinite Programs (SDPs)

Definition 5.26

A semidefinite program is a linear program with the additional constraint that a square symmetric matrix of variables must be positive semidefinite.

Example.

$$
\begin{aligned}
\min / \max \quad & \sum_{i,j} c_{ij} x_{ij} & \\
\text{s.t.} \quad & \sum_{i,j} a_{ijk} x_{ij} = b_k & \text{for all } k, \\
& x_{ij} = x_{ji} & \text{for all } i, j, \\
& X = (x_{ij}) \succeq 0 &
\end{aligned}
$$

Remark. The set of feasible solutions of a semidefinite program is convex.

Vector Programs

A semidefinite program can be stated equivalently as a vector program and vice versa (see Theorem 5.25(iii)):

$$\begin{aligned}
\text{min / max} \quad & \sum_{i,j} c_{ij} \left(v_i^T \cdot v_j \right) & \\
\text{s.t.} \quad & \sum_{i,j} a_{ijk} \left(v_i^T \cdot v_j \right) = b_k & \text{for all } k = 1, \ldots, K, \\
& v_i \in \mathbb{R}^n & \text{for all } i = 1, \ldots, n.
\end{aligned}$$

Vector Programs

A semidefinite program can be stated equivalently as a vector program and vice versa (see Theorem 5.25(iii)):

$$\begin{aligned} \min \;/\; \max \quad & \sum_{i,j} c_{ij} \, (v_i^T \cdot v_j) & \\ \text{s.t.} \quad & \sum_{i,j} a_{ijk} \, (v_i^T \cdot v_j) = b_k & \text{for all } k = 1, \ldots, K, \\ & v_i \in \mathbb{R}^n & \text{for all } i = 1, \ldots, n. \end{aligned}$$

Remark.

- Under mild technical conditions, semidefinite programs can be solved within additive error ε in time polynomial in input size and $\log(1/\varepsilon)$.

Vector Programs

A semidefinite program can be stated equivalently as a vector program and vice versa (see Theorem 5.25(iii)):

$$\begin{aligned} \min \ / \ \max \quad & \sum_{i,j} c_{ij} \, (v_i^T \cdot v_j) & \\ \text{s.t.} \quad & \sum_{i,j} a_{ijk} \, (v_i^T \cdot v_j) = b_k & \text{for all } k = 1, \ldots, K, \\ & v_i \in \mathbb{R}^n & \text{for all } i = 1, \ldots, n. \end{aligned}$$

Remark.

- Under mild technical conditions, semidefinite programs can be solved within additive error ε in time polynomial in input size and $\log(1/\varepsilon)$.
- For simplicity, we assume in the following that we can efficiently obtain an optimal solution.

SDP Relaxation of MAX CUT

Integer quadratic programming formulation of MAX CUT

$$\begin{aligned}
\max \quad & \frac{1}{2}\sum_{ij\in E} w_{ij}\,(1 - y_i y_j) \\
\text{s.t.} \quad & y_i \in \{-1, 1\} \qquad \text{for all } i \in V.
\end{aligned}$$

SDP Relaxation of MAX CUT

Integer quadratic programming formulation of MAX CUT

$$\begin{aligned} \max \quad & \frac{1}{2} \sum_{ij \in E} w_{ij} \left(1 - y_i y_j\right) \\ \text{s.t.} \quad & y_i \in \{-1, 1\} && \text{for all } i \in V. \end{aligned}$$

Semidefinite programming relaxation of MAX CUT

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

Lemma 5.27

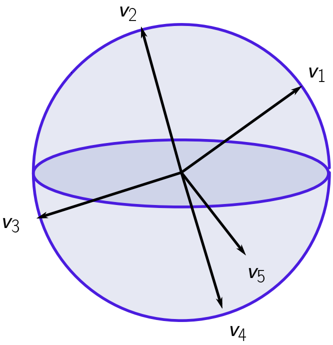
The above SDP is a relaxation of MAXCUT, therefore $\mathtt{OPT} \leq \mathtt{SDP}$.

Randomized Rounding of Vector Program

1. compute (near-)optimal solution (v^*) to SDP relaxation;
2. pick a random vector $r = (r_1, \ldots, r_n)^T$ by drawing each component from $\mathcal{N}(0, 1)$, the normal distribution with mean 0 and variance 1;
3. for $i = 1, \ldots, n$: if $r^T \cdot v_i^* \geq 0$ then put i in S;

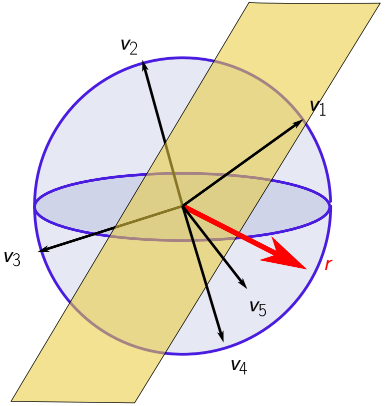
Randomized Rounding of Vector Program

1. compute (near-)optimal solution (v^*) to SDP relaxation;
2. pick a random vector $r = (r_1, \dots, r_n)^T$ by drawing each component from $\mathcal{N}(0, 1)$, the normal distribution with mean 0 and variance 1;
3. for $i = 1, \dots, n$: if $r^T \cdot v_i^* \geq 0$ then put i in S;



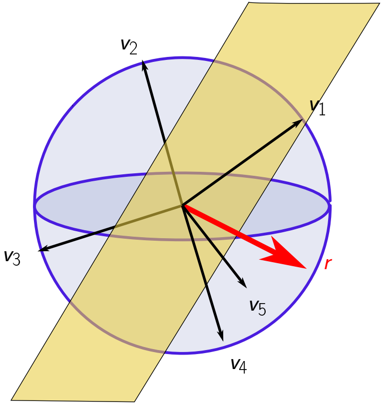
Randomized Rounding of Vector Program

1. compute (near-)optimal solution (v^*) to SDP relaxation;
2. pick a random vector $r = (r_1, \dots, r_n)^T$ by drawing each component from $\mathcal{N}(0, 1)$, the normal distribution with mean 0 and variance 1;
3. for $i = 1, \dots, n$: if $r^T \cdot v_i^* \geq 0$ then put i in S;



Randomized Rounding of Vector Program

1. compute (near-)optimal solution (v^*) to SDP relaxation;
2. pick a random vector $r = (r_1, \ldots, r_n)^T$ by drawing each component from $\mathcal{N}(0,1)$, the normal distribution with mean 0 and variance 1;
3. for $i = 1, \ldots, n$: if $r^T \cdot v_i^* \geq 0$ then put i in S;



The random hyperplane with normal vector ***r*** produces the cut

$$S = \{1, 4, 5\},$$

$$V \setminus S = \{2, 3\}.$$

Randomized Rounding of Vector Program

1. compute (near-)optimal solution (v^*) to SDP relaxation;
2. pick a random vector $r = (r_1, \ldots, r_n)^T$ by drawing each component from $\mathcal{N}(0, 1)$, the normal distribution with mean 0 and variance 1;
3. for $i = 1, \ldots, n$: if $r^T \cdot v_i^* \geq 0$ then put i in S;

Remarks.

- The hyperplane orthogonal to r partitions the n-dimensional unit sphere into two halves, corresponding to S and $V \setminus S$.

Randomized Rounding of Vector Program

1. compute (near-)optimal solution (v^*) to SDP relaxation;
2. pick a random vector $r = (r_1, \ldots, r_n)^T$ by drawing each component from $\mathcal{N}(0, 1)$, the normal distribution with mean 0 and variance 1;
3. for $i = 1, \ldots, n$: if $r^T \cdot v_i^* \geq 0$ then put i in S;

Remarks.

- The hyperplane orthogonal to r partitions the n-dimensional unit sphere into two halves, corresponding to S and $V \setminus S$.
- The normalization $r/\|r\|$ of r is uniformly distributed over the n-dimensional unit sphere.

Randomized Rounding of Vector Program

1. compute (near-)optimal solution (v^*) to SDP relaxation;
2. pick a random vector $r = (r_1, \ldots, r_n)^T$ by drawing each component from $\mathcal{N}(0,1)$, the normal distribution with mean 0 and variance 1;
3. for $i = 1, \ldots, n$: if $r^T \cdot v_i^* \geq 0$ then put i in S;

Remarks.

- The hyperplane orthogonal to r partitions the n-dimensional unit sphere into two halves, corresponding to S and $V \setminus S$.
- The normalization $r/\|r\|$ of r is uniformly distributed over the n-dimensional unit sphere.
- The projections of r onto two unit vectors e_1, e_2 are independent and normally distributed if and only if e_1 and e_2 are orthogonal.

Randomized Rounding of Vector Program

1. compute (near-)optimal solution (v^*) to SDP relaxation;
2. pick a random vector $r = (r_1, \ldots, r_n)^T$ by drawing each component from $\mathcal{N}(0, 1)$, the normal distribution with mean 0 and variance 1;
3. for $i = 1, \ldots, n$: if $r^T \cdot v_i^* \geq 0$ then put i in S;

Remarks.

- The hyperplane orthogonal to r partitions the n-dimensional unit sphere into two halves, corresponding to S and $V \setminus S$.
- The normalization $r/\|r\|$ of r is uniformly distributed over the n-dimensional unit sphere.
- The projections of r onto two unit vectors e_1, e_2 are independent and normally distributed if and only if e_1 and e_2 are orthogonal.

Corollary 5.28

Let r' the projection of r onto a 2-dimensional plane. The normalization $r'/\|r'\|$ of r', is uniformly distributed on a unit circle in the plane. □

Analysis of the SDP-based Algorithm

Lemma 5.29

The probability that edge $ij \in E$ is in the cut is $\frac{1}{\pi} \arccos({v_i}^T \cdot v_j)$.

Analysis of the SDP-based Algorithm

Lemma 5.29

The probability that edge $ij \in E$ is in the cut is $\dfrac{1}{\pi} \arccos({v_i}^T \cdot v_j)$.

Proof:… □

Analysis of the SDP-based Algorithm

Lemma 5.29

The probability that edge $ij \in E$ is in the cut is $\frac{1}{\pi} \arccos({v_i}^T \cdot v_j)$.

Proof:… □

Lemma 5.30

For $x \in [-1, 1]$ it holds that $\frac{1}{\pi} \arccos(x) \geq 0.878 \cdot \frac{1}{2}(1 - x)$. □

Analysis of the SDP-based Algorithm

Lemma 5.29

The probability that edge $ij \in E$ is in the cut is $\frac{1}{\pi} \arccos(v_i^T \cdot v_j)$.

Proof:... □

Lemma 5.30

For $x \in [-1, 1]$ it holds that $\frac{1}{\pi} \arccos(x) \geq 0.878 \cdot \frac{1}{2}(1 - x)$. □

Theorem 5.31 (Goemans & Williamson)

SDP-based randomized rounding is a randomized 0.878-approximation algorithm for MAX CUT.

Analysis of the SDP-based Algorithm

Lemma 5.29

The probability that edge $ij \in E$ is in the cut is $\frac{1}{\pi} \arccos({v_i}^T \cdot v_j)$.

Proof:... □

Lemma 5.30

For $x \in [-1, 1]$ it holds that $\frac{1}{\pi} \arccos(x) \geq 0.878 \cdot \frac{1}{2}(1 - x)$. □

Theorem 5.31 (Goemans & Williamson)

SDP-based randomized rounding is a randomized 0.878-approximation algorithm for MAX CUT.

Proof:... □

Analysis of the SDP-based Algorithm

Lemma 5.29

The probability that edge $ij \in E$ is in the cut is $\frac{1}{\pi} \arccos({v_i}^T \cdot v_j)$.

Proof:... □

Lemma 5.30

For $x \in [-1, 1]$ it holds that $\frac{1}{\pi} \arccos(x) \geq 0.878 \cdot \frac{1}{2}(1 - x)$. □

Theorem 5.31 (Goemans & Williamson)

SDP-based randomized rounding is a randomized 0.878-approximation algorithm for MAX CUT.

Proof:... □

Remark. The algorithm can be derandomized by using a sophisticated application of the method of conditional expectations.

Illustration of Lemma 5.30

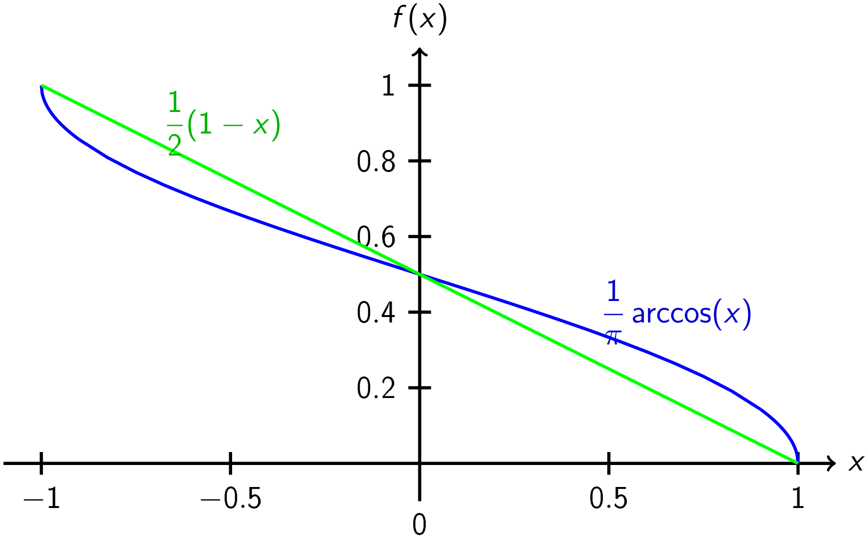
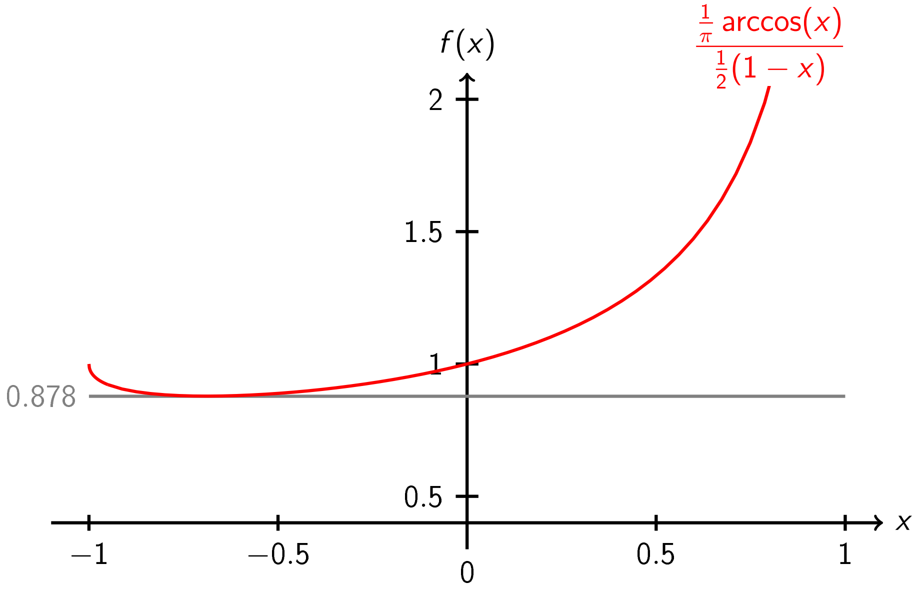


Illustration of Lemma 5.30 (Cont.)



Inapproximability Results for MAX CUT

We state the following results without proof.

Theorem 5.32

If there is an α-approximation algorithm for MAX CUT with $\alpha > 16/17 \approx 0.941$, then $P = NP$. $\square$

Inapproximability Results for MAX CUT

We state the following results without proof.

Theorem 5.32

If there is an α-approximation algorithm for MAX CUT with $\alpha > 16/17 \approx 0.941$, then $P = NP$. □

Theorem 5.33

Given the *Unique Games Conjecture* there is no α-approximation algorithm for MAX CUT with constant

$$\alpha > \min_{-1 \le x \le 1} \frac{\frac{1}{\pi}\arccos(x)}{\frac{1}{2}(1-x)} \approx 0.878$$

unless $P = NP$. □