Approximation Algorithms

Group Members:

1. Geng Xue (A0095628R)
2. Cai Jingli (A0095623B)
3. Xing Zhe (A0095644W)
4. Zhu Xiaolu (A0109657W)
5. Wang Zixiao (A0095670X)
6. Jiao Qing (A0095637R)
7. Zhang Hao (A0095627U)

Introduction

Geng Xue

Optimization problem

Find the minimum/maximum of …

– Vertex Cover : A minimum set of vertices that covers all the edges in a graph.

NP optimization problem

- The hardness of NP optimization is NP hard.

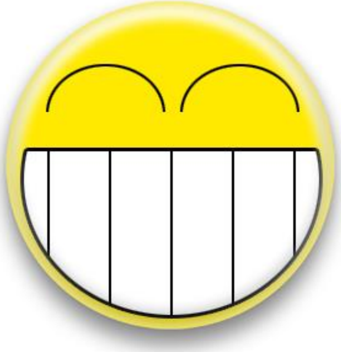
The hardest problem in NP

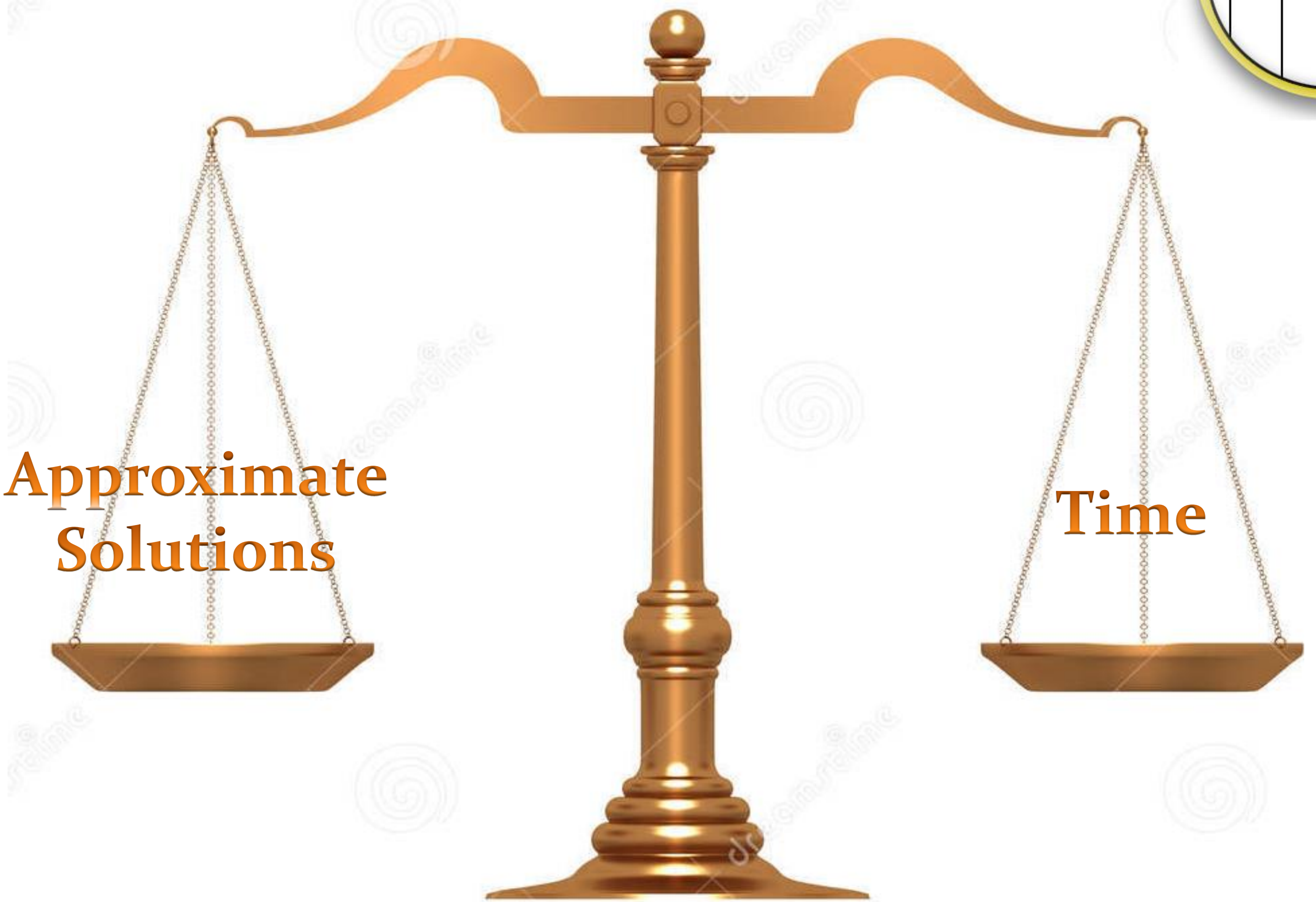
- Can't be solved in polynomial time.

Time consuming !!!

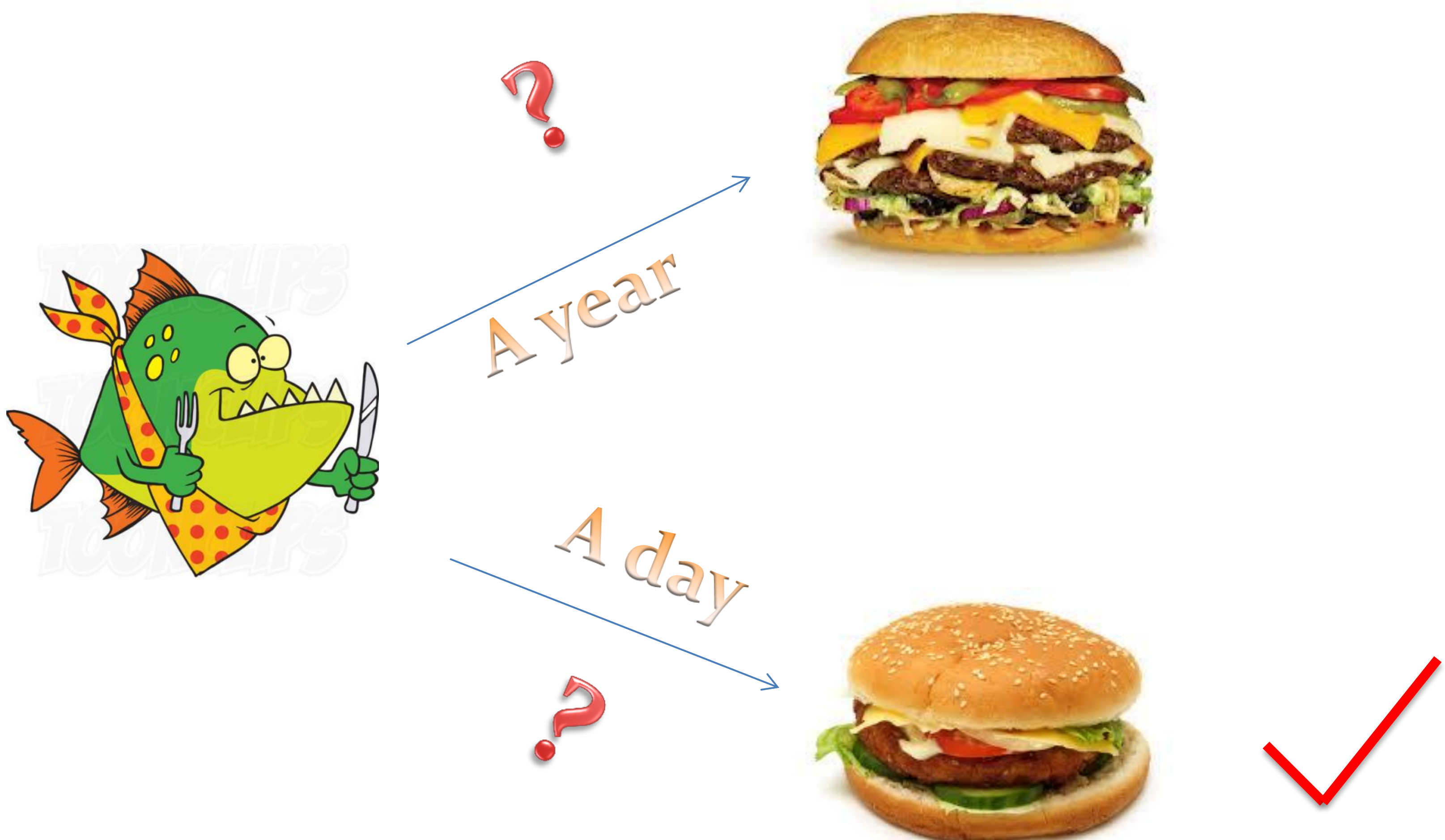
How to solve ?

How to solve ?





How to solve ?



How to solve ?

Approximation Algorithm!

- **sub-optimal solution!**
- **Polynomial Time!**

How to evaluate ?

Mine is better!

Standard!

How to evaluate ?

OPT

Exact optimal value of optimization problem.

Factor

relative performance guarantee.

- Sub-optimal solution : $\rho \times OPT$ (ρ is factor).
- The closer factor ρ is to 1, The better the algorithm is.

Set cover

- It leads to the development of fundamental techniques for the entire approximation algorithms field.
- Due to set cover, many of the basic algorithm design techniques can be explained with great ease.

What is set cover ?

- Definition:
  - Given a universe U of n elements, a collection of subsets of U, $S = \{S_1, S_2, \dots, S_k\}$, and a cost function $c: S \rightarrow Q^+$, find a minimum cost sub-collection of S that covers all elements of U.

What is set cover ?

- Example:

S	Sets	Cost
S_1	{1,2}	1
S_2	{3,4}	1
S_3	{1,3}	2
S_4	{5,6}	1
S_5	{1,5}	3
S_6	{4,6}	1

Universal Set :
$U = \{1,2,3,4,5,6\}$

Find a sub-collection of S that covers all the elements of U with minimum cost.

Solution?

$S_1 \quad S_2 \quad S_4$

Cost = 1 + 1 + 1 = 3

Approximation algorithms to set cover problem

- Combinatorial algorithms
- Linear programming based Algorithms (LP-based)

Combinatorial & LP-based

- Combinatorial algorithms
  - Greedy algorithms
  - Layering algorithms
- LP-based algorithms
  - Dual Fitting
  - Rounding
  - Primal–Dual Schema

Greedy set cover algorithm

Cai Jingli

Greedy set cover algorithm

1. $C \leftarrow \emptyset$
2. While $C \neq U$ do

   Find the most cost-effective set in the current iteration, say S.

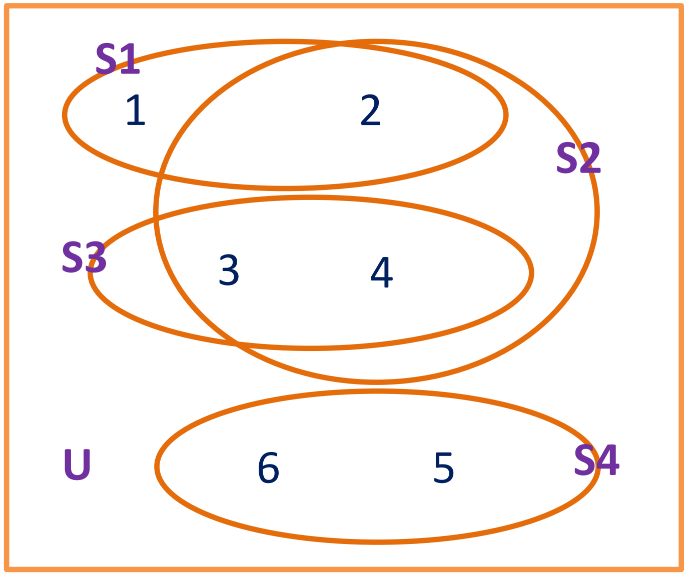
   Let $\alpha = \frac{\mathrm{cost}(S)}{|S-C|}$, i.e., the cost-effectiveness of S.

   Pick S, and for each $e \in S - C$, set $\mathrm{price}(e) = \alpha$.

   $C \leftarrow C \cup S$.
3. Output the picked sets.

Where C is the set of elements already covered at the beginning of an iteration and α is the average cost at which it covers new elements.

Example



subset	cost
S1 = {1, 2}	1
S2 = {2, 3, 4}	2
S3 = {3, 4}	3
S4 = {5, 6}	3

Iteration 1: C = {}

α1 = 1 / 2 = 0.5

α2 = 2 / 3 = 0.67

α3 = 3 / 2 = 1.5

α4 = 3 / 2 = 1.5

C = {1, 2}

Price(1) = 0.5

Price(2) = 0.5

Iteration 2: C = {1, 2}

α2 = 2 / 2 = 1

α3 = 3 / 2 = 1.5

α4 = 3 / 2 = 1.5

C = {1, 2, 3, 4}

Price(3) = 1

Price(4) = 1

Iteration 3: C = {1, 2, 3, 4 }

α3 = 3 / 0 = infinite

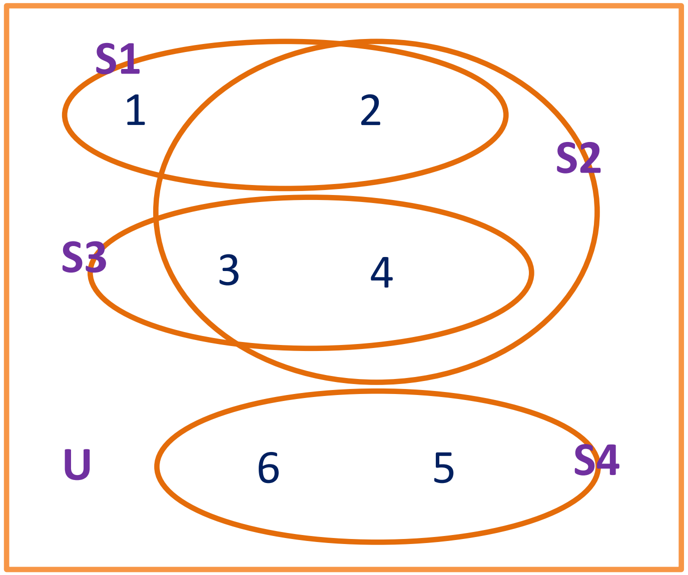
α4 = 3 / 2 = 1.5

C = {1, 2, 3, 4, 5, 6}

Price(5) = 1.5

Price(6) = 1.5

Example



subset	cost
S1 = {1, 2}	1
S2 = {2, 3, 4}	2
S3 = {3, 4}	3
S4 = {5, 6}	3

The picked sets = {S1, S2, S4}

Total cost = Cost(S1)+Cost(S2)+Cost(S4)

= Price(1) + Price(2) + … + Price(6)

= 0.5 + 0.5 + 1 + 1 + 1.5 + 1.5 = 6

Theorem

- The greedy algorithm is an $\boldsymbol{H_n}$ factor approximation algorithm for the minimum set cover problem, where

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

$$\sum_i Cost(Si) \leq H_n \cdot OPT$$

Proof

1) We know $\sum_{e \in U} price(e) =$ cost of the greedy algorithm $= c(S_1) + \text{c}(S_2) + \cdots + c(S_m)$.
2) We will show $price(e_k) \le \frac{OPT}{n-k+1}$, where e_k is the kth element covered.

If 2) is proved then theorem is proved also.

$$\sum_{i} Cost(Si) = \sum_{e \in U} price(e) \le \sum_{k=1}^{n} \frac{OPT}{n-k+1} = OPT * \sum_{k=1}^{n} \frac{1}{n-k+1}$$

$$= OPT * (1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}) = H_n \cdot OPT$$

$$price(e_k) \le \frac{OPT}{n-k+1}$$

- Say the optimal sets are $O_1, O_2, \dots, O_p$, so $\mathrm{OPT} = \mathrm{c}(O_1) + c(O_2) + \cdots + c\left(O_p\right) = \sum_{i=1}^{p} \mathrm{c}(O_i)$
- Now, assume the greedy algorithm has covered the elements in C so far.

$$|U - C| \le |O_1 \cap (U - C)| + \cdots + \left|O_p \cap (U - C)\right|$$

$$= \sum_{i=1}^{p} |O_i \cap (U - C)|$$

- $\mathrm{price}(e_k) = \alpha \le \frac{c(O_i)}{|O_i \cap (U-C)|}$ (2), *i*=1,…,*p*. we know this because of greedy algorithm

$$price(e_k) \leq \frac{OPT}{n-k+1}$$

$$(1)\ \mathrm{OPT} = \sum_{i=1}^{p} \mathrm{c}(O_i)$$

$$(2) |U-C| \leq \sum_{i=1}^{p} |O_i \cap (U-C)|$$

$$(3) c(O_i) \geq \alpha \cdot |O_i \cap (U-C)|$$

- So $OPT = \sum_{i=1}^{p} \mathrm{c}(O_i) \geq \alpha \cdot \sum_{i=1}^{p} |O_i \cap (U-C)| \geq \alpha \cdot |U-C|$

$$|U-C| = n-k+1$$

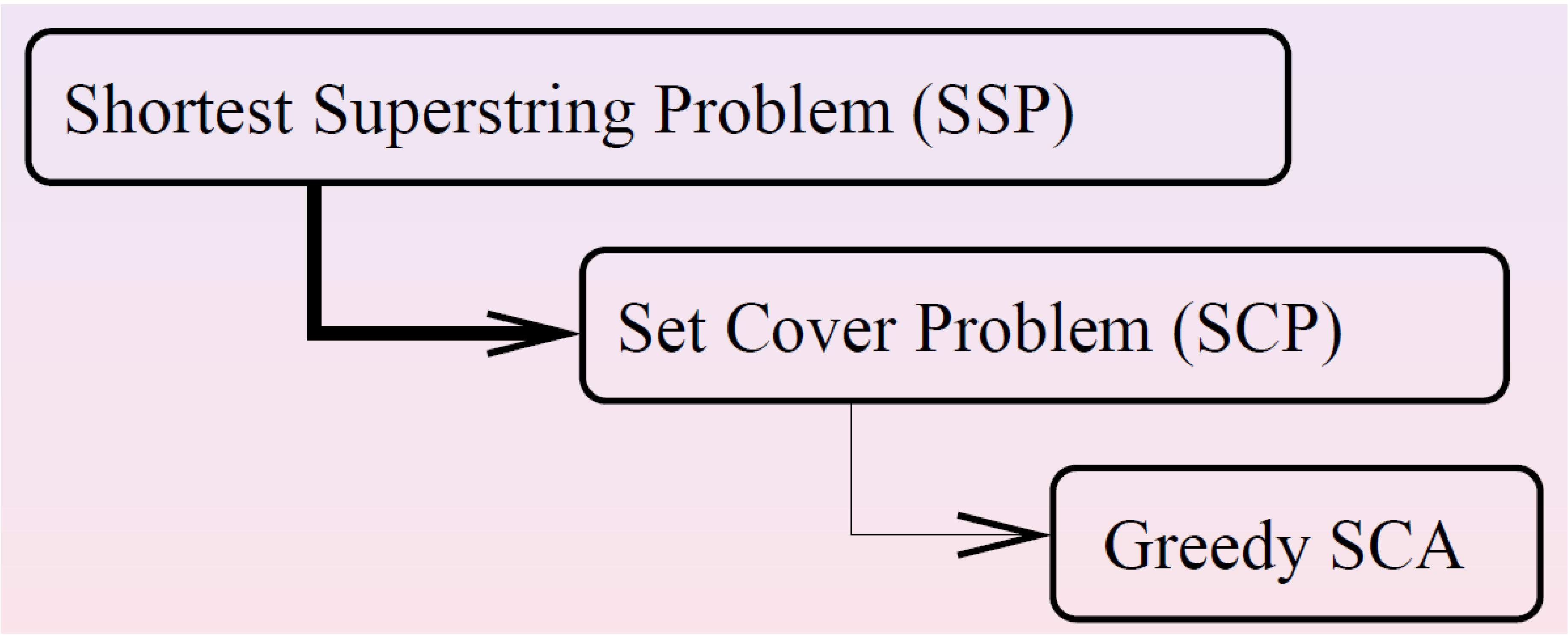
$$price(e_k) = \alpha \leq \frac{OPT}{n-k+1}$$

Shortest Superstring Problem (SSP)

- Given a finite alphabet Σ, and set of n strings, $S = \{s_1, \dots, s_n\} \subseteq \Sigma^+$.
- Find a shortest string s that contains each s_i as a substring.
- Without loss of generality, we may assume that no string s_i is a substring of another string s_j , $j \neq i$.

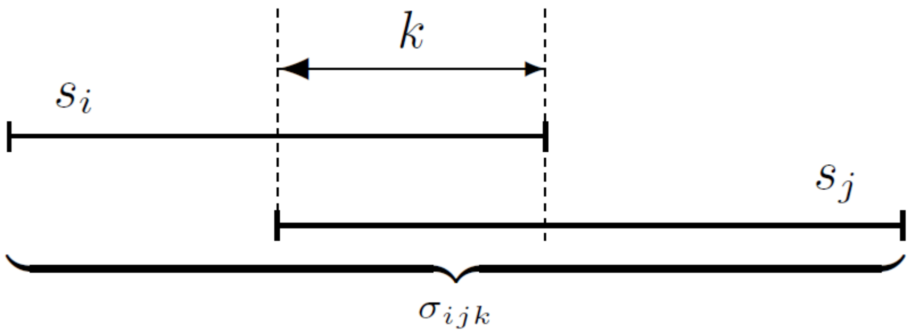
Application: Shotgun sequencing

Approximating SSP Using Set Cover



Set Cover Based Algorithm

- **Set Cover Problem:**
  - Choose some sets that cover all **elements** with least cost
- **Elements**
  - The input strings
- **Subsets**
  - σ_{ijk} = string obtained by overlapping input strings s_i and s_j , with k letters.
  - $\beta = \boldsymbol{S} \cup \sigma_{ijk},\ all\ i,j,k$
  - Let $\pi \in \beta$
  - **set**(π) = $\{s \in \boldsymbol{S} \mid s$ is a substr. of $\pi\}$



Set Cover Based Algorithm

- **Set Cover Problem:**
  - Choose some sets that cover all **elements** with least cost
- **Elements**
  - The input strings
- **Subsets**
  - σ_{ijk} = string obtained by overlapping input strings s_i and s_j , with k letters.
  - $\beta = \boldsymbol{S} \cup \sigma_{ijk},\ all\ i, j, k$
  - Let $\pi \in \beta$
  - **set**(π) = $\{s \in \boldsymbol{S} \mid s$ is a substr. of $\pi\}$
- **Cost of a subset**
  - set(π) is $|\pi|$
- A solution to SSP is the concatenation of π obtained from SCP

Example

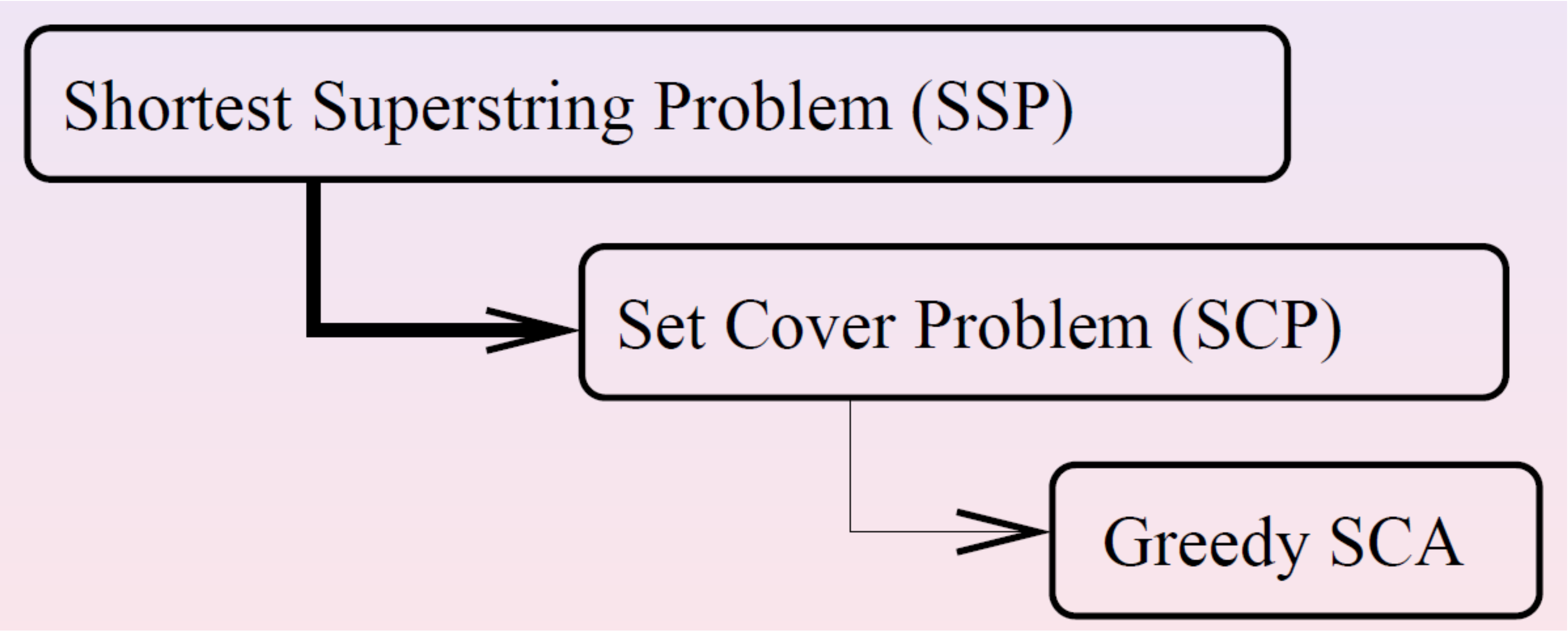
- S = {CATGC, CTAAGT, GCTA, TTCA, ATGCATC}

π	Set	Cost
CATGC . . . . CTAAGT CATGCTAAGT	CATGC, CTAAGT, GCTA	10
CATGC . . . . . GCTA CATGCTA	CATGC, GCTA	7
. . . . CATGC ATGCATC . . . . . ATGCATCATGC	CATGC, ATGCATC	11
CTAAGT . . . . TTCA CTAAGTTCA	CTAAGT, TTCA	9
ATGCATC . . . . CTAAGT ATGCATCTAAGT	CTAAGT, ATGCATC	12

GCTA . . . . ATGCATC GCTATGCATC	GCTA, ATGCATC	10
TTCA . . . . ATGCATC TTCATGCATC	TTCA, ATGCATC, CATGC	10
GCTA . . . . CTAAGT GCTAAGT	GCTA, CTAAGT	7
TTCA . . . . . CATGC TTCATGC	CATGC, TTCA	7
CATGC . . . . ATGCATC CATGCATC	CATGC, ATGCATC	8
CATGC	CATGC	5
CTAAGT	CTAAGT	6
GCTA	GCTA	4
TTCA	TTCA	4
ATGCATC	ATGCATC	7

Lemma

$$OPT \leq OPT_{SCA} \leq 2H_n \cdot OPT$$



Layering Technique

Xing Zhe

Layering Technique

Set Cover Problem

Combinatorial algorithms

Greedy: H_n

Layering ?

Vertex Cover Problem

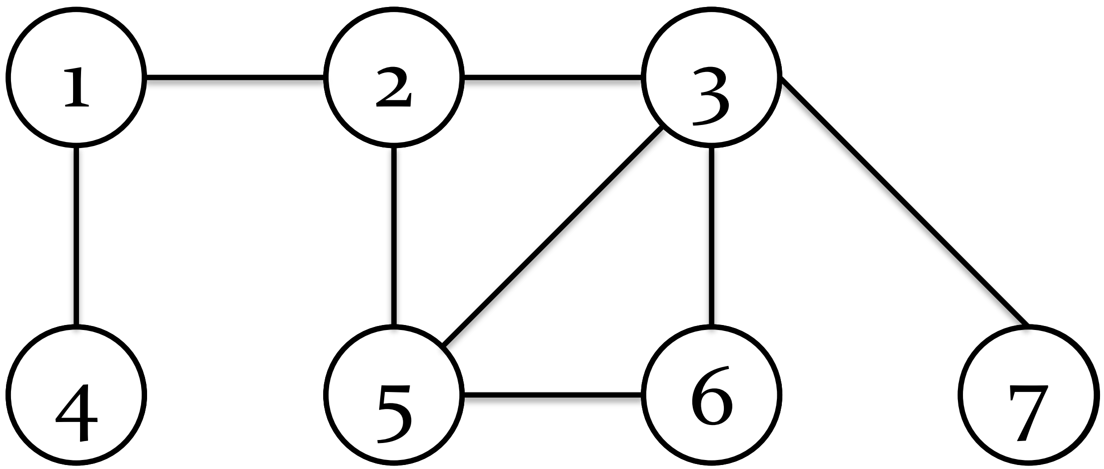
Definition:

Given a graph $G = (V, E)$, and a weight function $w\colon V \to Q^+$ assigning weights to the vertices, find a minimum weighted subset of vertices $C \subseteq V$, such that C "covers" all edges in E, i.e., every edge $e_i \in E$ is incident to at least one vertex in C.

Vertex Cover Problem

Example:

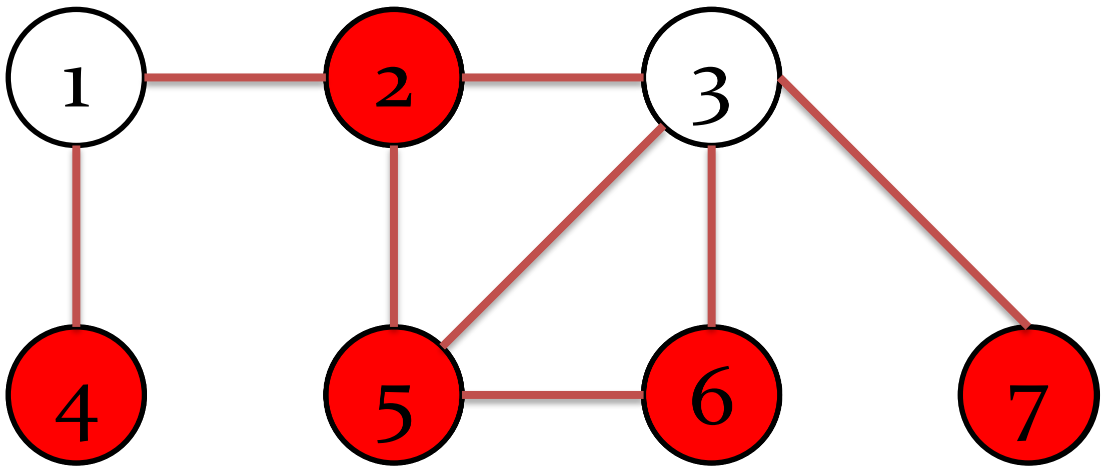
all vertices of unit weight



Vertex Cover Problem

Example:

all vertices of unit weight

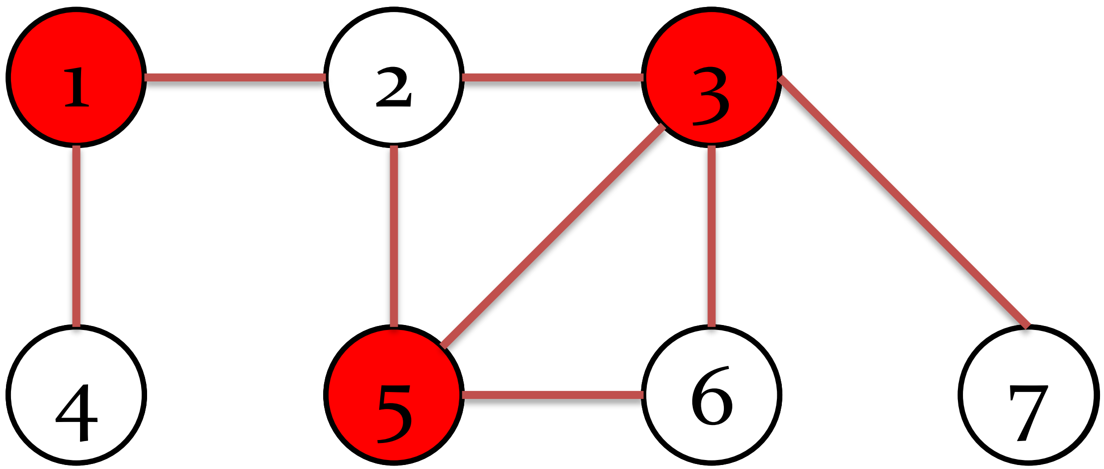


Total cost = 5

Vertex Cover Problem

Example:

all vertices of unit weight

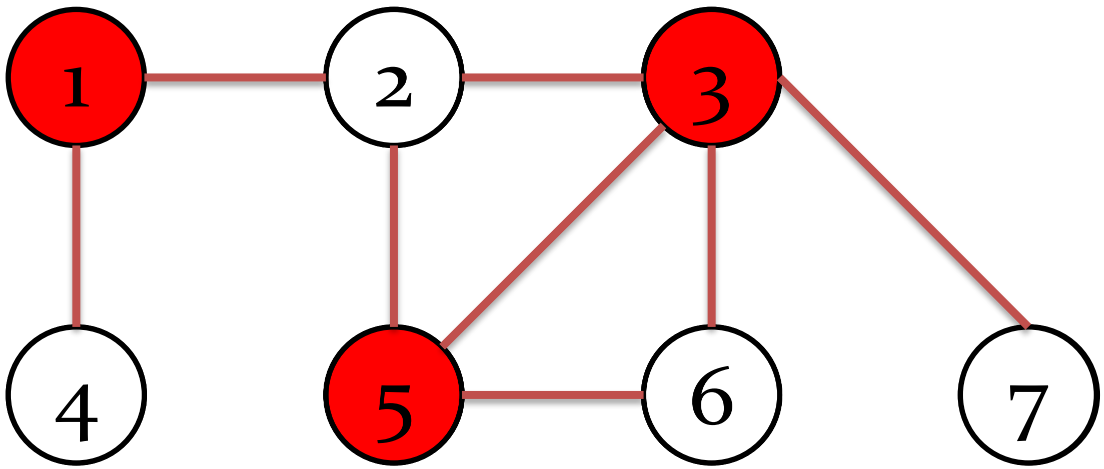


Total cost = 3

Vertex Cover Problem

Example:

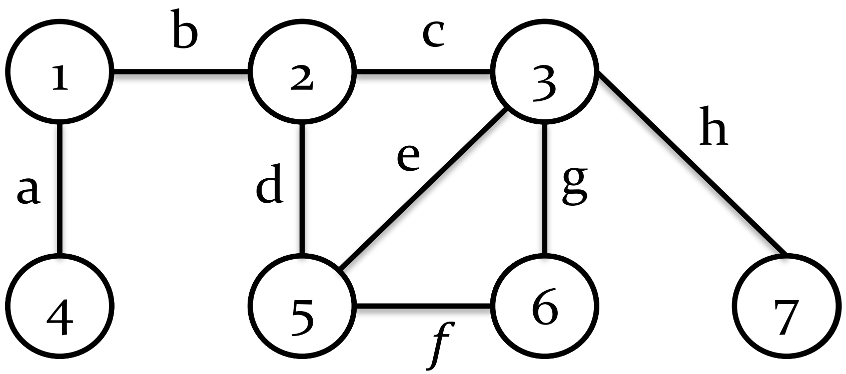
all vertices of unit weight



Total cost = 3 ⟵ **OPT**

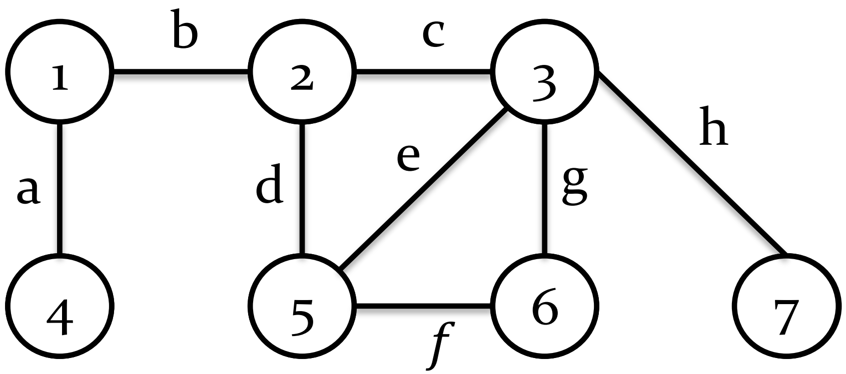
Vertex Cover Problem

Vertex cover is a special case of set cover.



Vertex Cover Problem

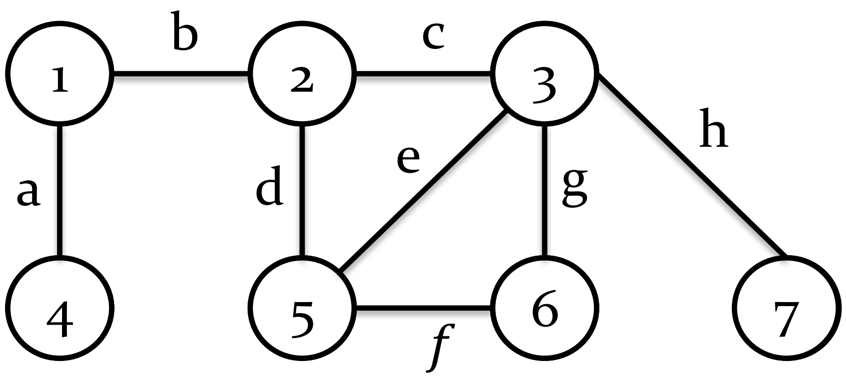
Vertex cover is a special case of set cover.



Edges are **elements**, **vertices** are **subsets**

Vertex Cover Problem

Vertex cover is a special case of set cover.

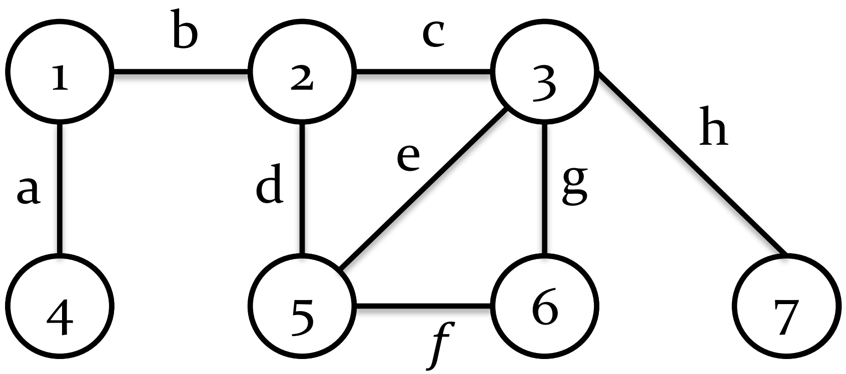


Edges are **elements**, **vertices** are **subsets**

Universe $U = \{a, b, c, d, e, f, g, h\}$

Vertex Cover Problem

Vertex cover is a special case of set cover.



$$S_1 = \{a, b\}$$
$$S_2 = \{b, c, d\}$$
$$S_3 = \{c, e, g, h\}$$
$$S_4 = \{a\}$$
$$S_5 = \{d, e, f\}$$
$$S_6 = \{f, g\}$$
$$S_7 = \{h\}$$

Edges are **elements**, **vertices** are **subsets**

Universe $U = \{a, b, c, d, e, f, g, h\}$

A collection of subsets $S = \{S_1, S_2, S_3, S_4, S_5, S_6, S_7\}$

Notation

1) frequency

2) f : the frequency of the most frequent element.

Notation

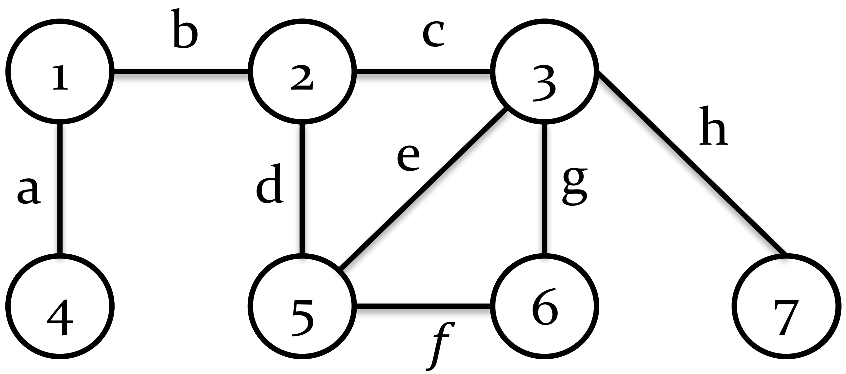
1) frequency

2) f : the frequency of the most frequent element.

In the vertex cover problem, $f = ?$

Vertex Cover Problem

Vertex cover is a special case of set cover.



$$S_1 = \{a, b\}$$
$$S_2 = \{b, c, d\}$$
$$S_3 = \{c, e, g, h\}$$
$$S_4 = \{a\}$$
$$S_5 = \{d, e, f\}$$
$$S_6 = \{f, g\}$$
$$S_7 = \{h\}$$

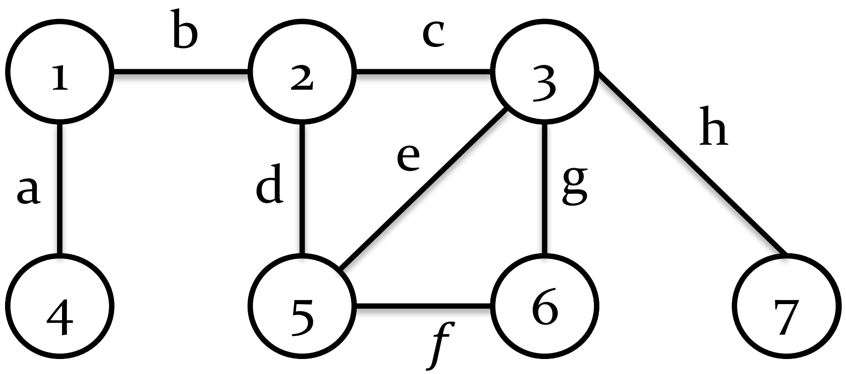
Edges are **elements**, **vertices** are **subsets**

Universe $U = \{a, b, c, d, e, f, g, h\}$

A collection of subsets $S = \{S_1, S_2, S_3, S_4, S_5, S_6, S_7\}$

Vertex Cover Problem

Vertex cover is a special case of set cover.



$S_1 = \{a, b\}$
$S_2 = \{b, c, d\}$
$S_3 = \{c, e, g, h\}$
$S_4 = \{a\}$
$S_5 = \{d, e, f\}$
$S_6 = \{f, g\}$
$S_7 = \{h\}$

Edges are **elements**, **vertices** are **subsets**

Universe $U = \{a, b, c, d, e, f, g, h\}$

A collection of subsets $S = \{S_1, S_2, S_3, S_4, S_5, S_6, S_7\}$

Vertex cover problem is a set cover problem with $\boldsymbol{f = 2}$

Approximation factor

1) frequency

2) f : the frequency of the most frequent element.

In the vertex cover problem, $f = 2$

Approximation factor

1) frequency

2) f : the frequency of the most frequent element.

In the vertex cover problem, $f = 2$

	Set Cover Problem	Vertex Cover Problem
Layering Algorithm	**factor** $= f$	**factor** $= 2$

Vertex Cover Problem

arbitrary weight function:

$$w: V \to Q^+$$

Vertex Cover Problem

arbitrary weight function:

$$w: V \to Q^+$$

degree-weighted function:

$$\exists\, c > 0 \text{ s.t.}$$

$$\forall\, v \in V, \quad w(v) = c \cdot \deg(v)$$

Degree weighted function

Lemma:

If $w\colon V \to Q^{+}$ is a degree-weighted function.

Then $w(V) \leq \mathbf{2} \cdot OPT$

Lemma:

If $w: V \to Q^+$ is a degree-weighted function.

Then $w(V) \leq 2 \cdot OPT$

Lemma:

If $w\colon V \rightarrow Q^+$ is a degree-weighted function.

Then $w(V) \leq 2 \cdot OPT$

Proof:

degree-weighted function, $w(v) = c \cdot \deg(v)$

Lemma:

If $w\colon V \to Q^+$ is a degree-weighted function.

Then $w(V) \leq 2 \cdot OPT$

Proof:

degree-weighted function, $w(v) = c \cdot \deg(v)$

C^* is the optimal vertex cover in G, $\sum_{v \in C^*} \deg(v) \geq |E|$

Lemma:

If $w \colon V \to Q^{+}$ is a degree-weighted function.

Then $w(V) \leq 2 \cdot OPT$

Proof:

degree-weighted function, $w(v) = c \cdot \deg(v)$

C^* is the optimal vertex cover in G, $\sum_{v \in C^*} \deg(v) \geq |E|$

$OPT = w(C^*) = \mathrm{c} \cdot \sum_{v \in C^*} \deg(v) \geq c \cdot |E|$ **(1)**

Lemma:

If $w: V \to Q^+$ is a degree-weighted function.

Then $w(V) \leq 2 \cdot OPT$

Proof:

degree-weighted function, $w(v) = c \cdot \deg(v)$

C^* is the optimal vertex cover in G, $\sum_{v \in C^*} \deg(v) \geq |E|$

$OPT = w(C^*) = \mathrm{c} \cdot \sum_{v \in C^*} \deg(v) \geq c \cdot |E|$ **(1)**

in worst case, we pick all vertices.

handshaking lemma, $\sum_{v \in V} \deg(v) = 2|E|$

Lemma:

If $w: V \rightarrow Q^{+}$ is a degree-weighted function.

Then $w(V) \leq 2 \cdot OPT$

Proof:

degree-weighted function, $w(v) = c \cdot \deg(v)$

C^* is the optimal vertex cover in G, $\sum_{v \in C^*} \deg(v) \geq |E|$

$OPT = w(C^*) = \mathrm{c} \cdot \sum_{v \in C^*} \deg(v) \geq c \cdot |E|$ **(1)**

in worst case, we pick all vertices.

handshaking lemma, $\sum_{v \in V} \deg(v) = 2|E|$

$w(V) = \sum_{v \in V} w(v) = \sum_{v \in V} c \cdot \deg(v) = c \cdot \sum_{v \in V} \deg(v) = c \cdot 2|E|$ **(2)**

Lemma:

If $w\colon V \to Q^{+}$ is a degree-weighted function.

Then $w(V) \leq 2 \cdot OPT$

Proof:

degree-weighted function, $w(v) = c \cdot \deg(v)$

C^* is the optimal vertex cover in G, $\sum_{v \in C^*} \deg(v) \geq |E|$

$OPT = w(C^*) = \mathrm{c} \cdot \sum_{v \in C^*} \deg(v) \geq c \cdot |E|$ **(1)**

in worst case, we pick all vertices.

handshaking lemma, $\sum_{v \in V} \deg(v) = 2|E|$

$w(V) = \sum_{v \in V} w(v) = \sum_{v \in V} c \cdot \deg(v) = c \cdot \sum_{v \in V} \deg(v) = c \cdot 2|E|$ **(2)**

from (1) and (2), $w(V) \leq 2 \cdot OPT$

Layering algorithm

Basic idea:

arbitrary weight function

↓

several degree-weighted functions

↓

nice property (*factor* = 2) holds in each layer

Layering algorithm

1) remove all degree zero vertices, say this set is $\boldsymbol{D_0}$

2) compute $\boldsymbol{c} = \boldsymbol{min}\{\boldsymbol{w}(\boldsymbol{v})/\boldsymbol{deg}(\boldsymbol{v})\}$

3) compute degree-weighted function $\mathbf{t}(\boldsymbol{v}) = \boldsymbol{c} \cdot \mathbf{deg}(\boldsymbol{v})$

4) compute residual weight function $\boldsymbol{w}'(\boldsymbol{v}) = \boldsymbol{w}(\boldsymbol{v}) - \boldsymbol{t}(\boldsymbol{v})$

5) let $\boldsymbol{W_0} = \{\boldsymbol{v} \mid \boldsymbol{w}'(\boldsymbol{v}) = \mathbf{0}\}$, pick zero residual vertices into the cover set

6) let $\boldsymbol{G_1}$ be the graph induced on $\boldsymbol{V} - (\boldsymbol{D_0} \cup \boldsymbol{W_0})$

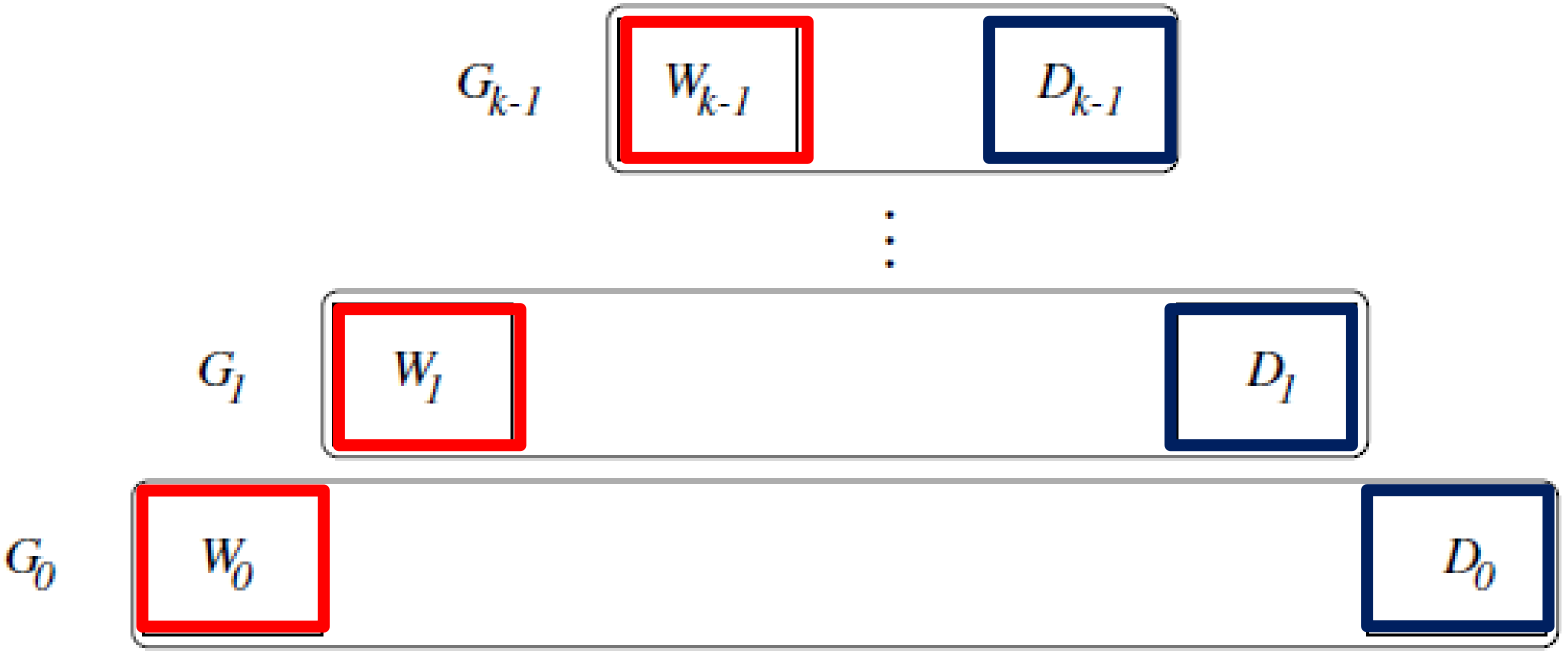
7) repeat the entire process on $\boldsymbol{G_1}$ w.r.t. the **residual** weight function,

until all vertices are of degree zero

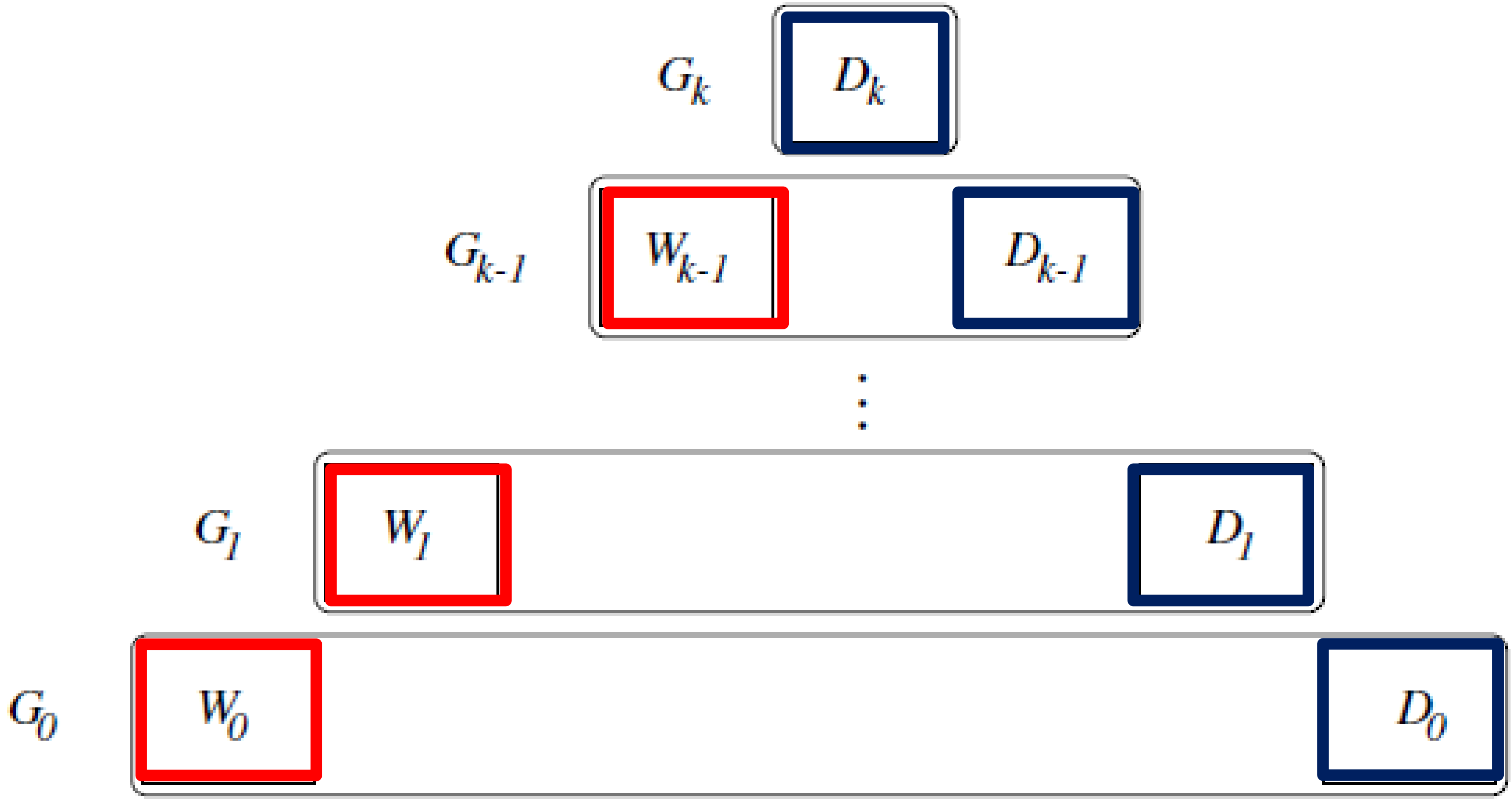
Layering algorithm

Layering algorithm

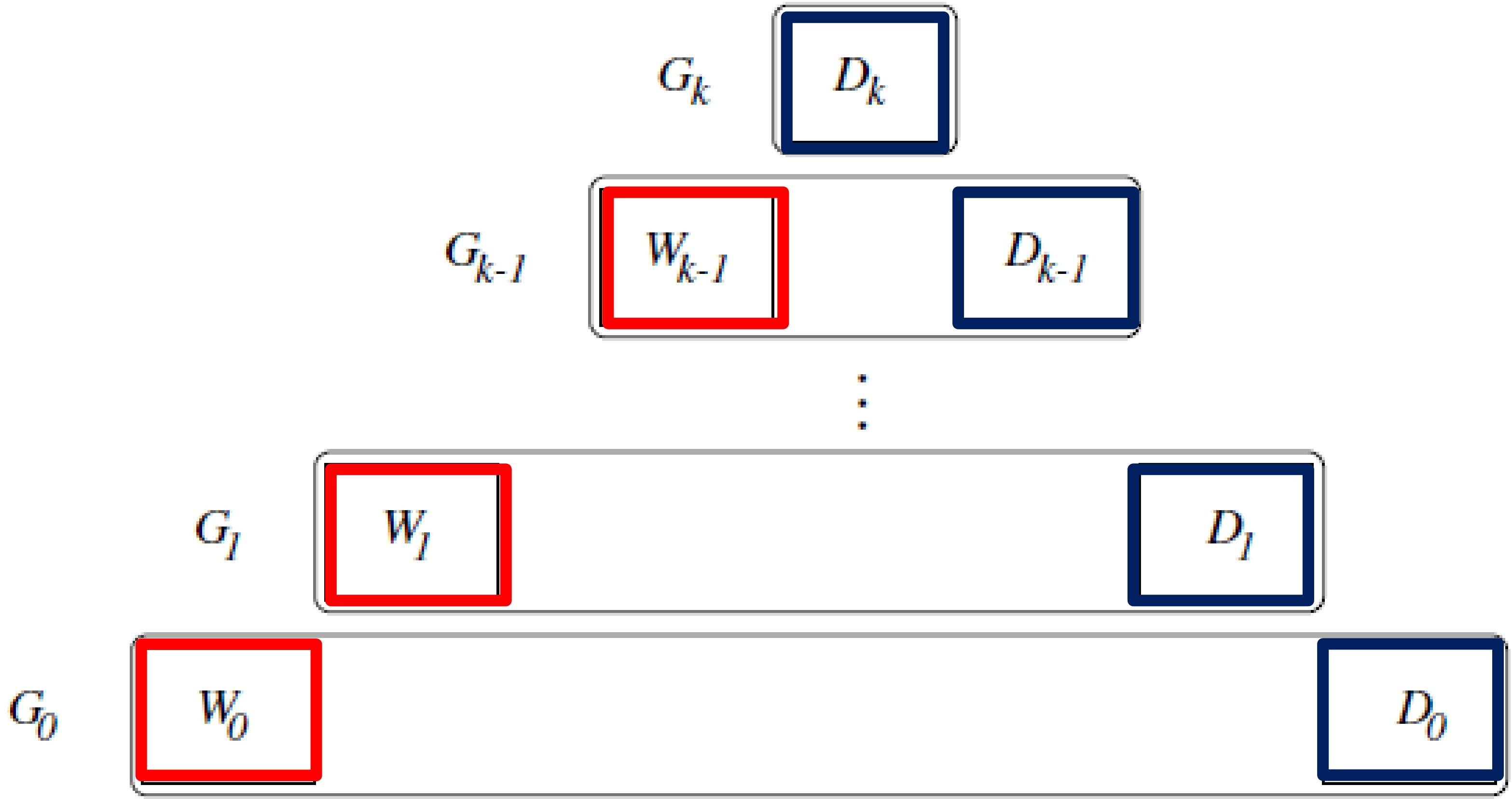
Layering algorithm



Layering algorithm



Layering algorithm



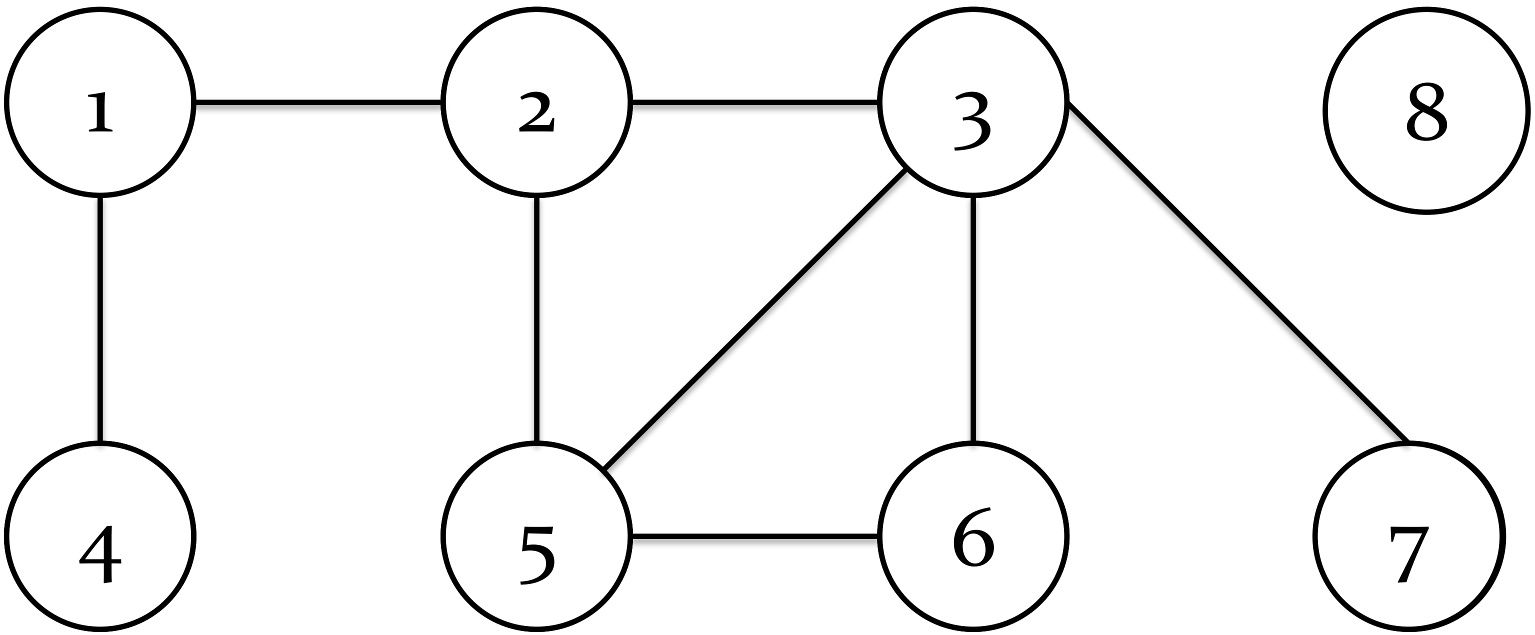
The vertex cover chosen is $\boldsymbol{C} = \boldsymbol{W_0} \cup \boldsymbol{W_1} \cup \ldots \cup \boldsymbol{W_{k-1}}$

Clearly, $\boldsymbol{V} - \boldsymbol{C} = \boldsymbol{D_0} \cup \boldsymbol{D_1} \cup \ldots \cup \boldsymbol{D_k}$

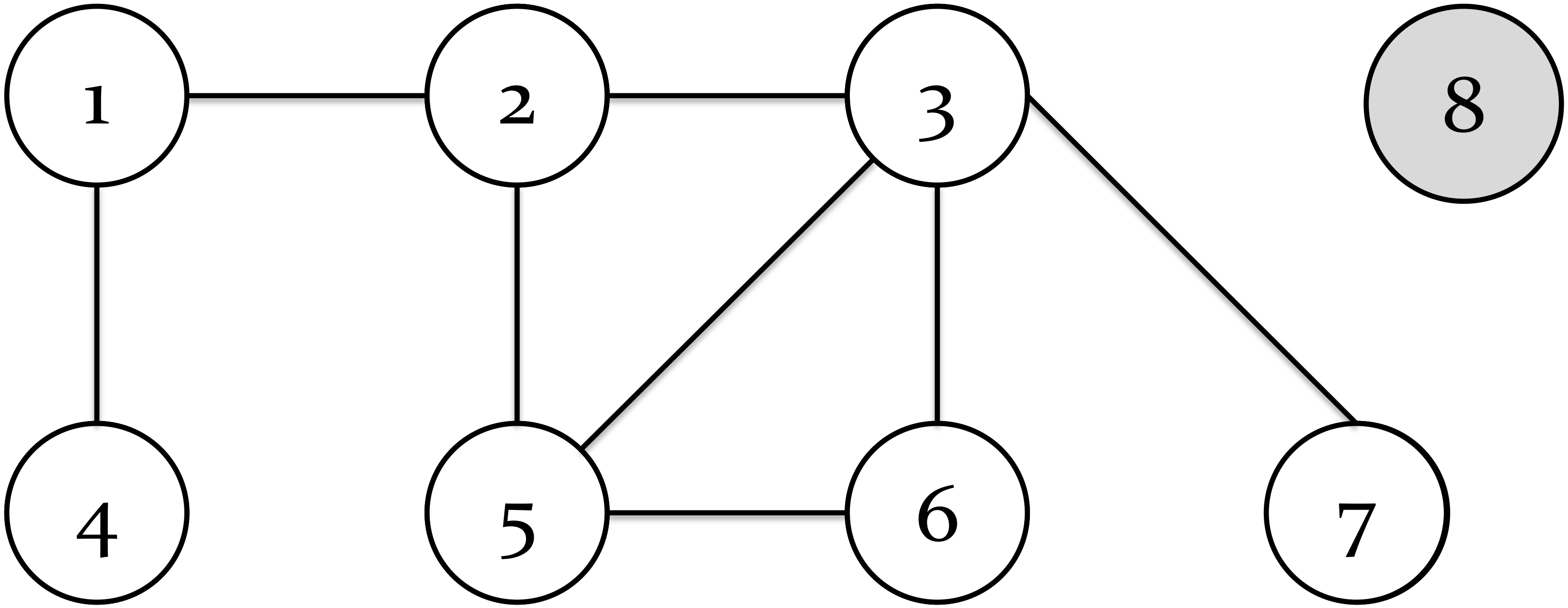
Layering algorithm

Example:

all vertices of unit weight: $w(v) = 1$

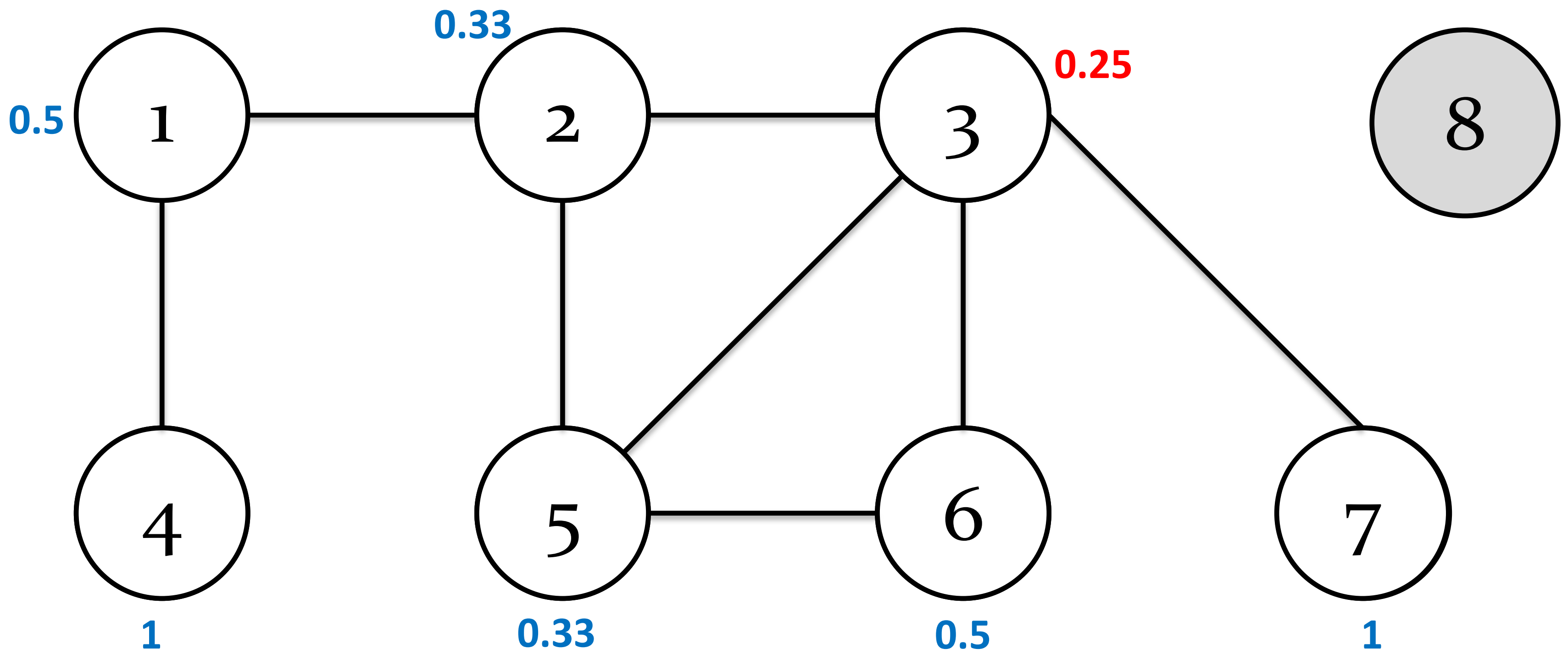


Iteration 0



$D_0 = \{\, 8 \,\}$

Iteration 0

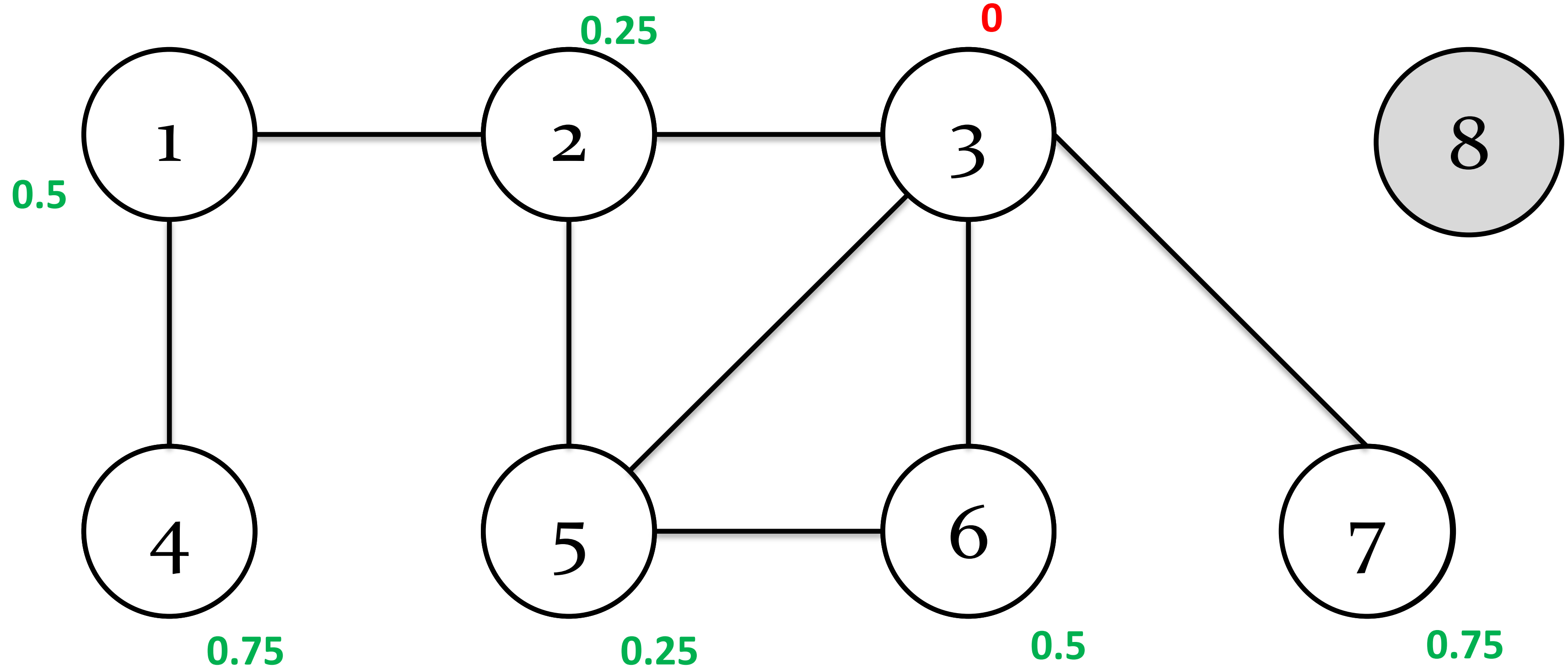


$D_0 = \{ 8 \}$

compute c = $min\{w(v)/deg(v)\}$ **= 0.25**

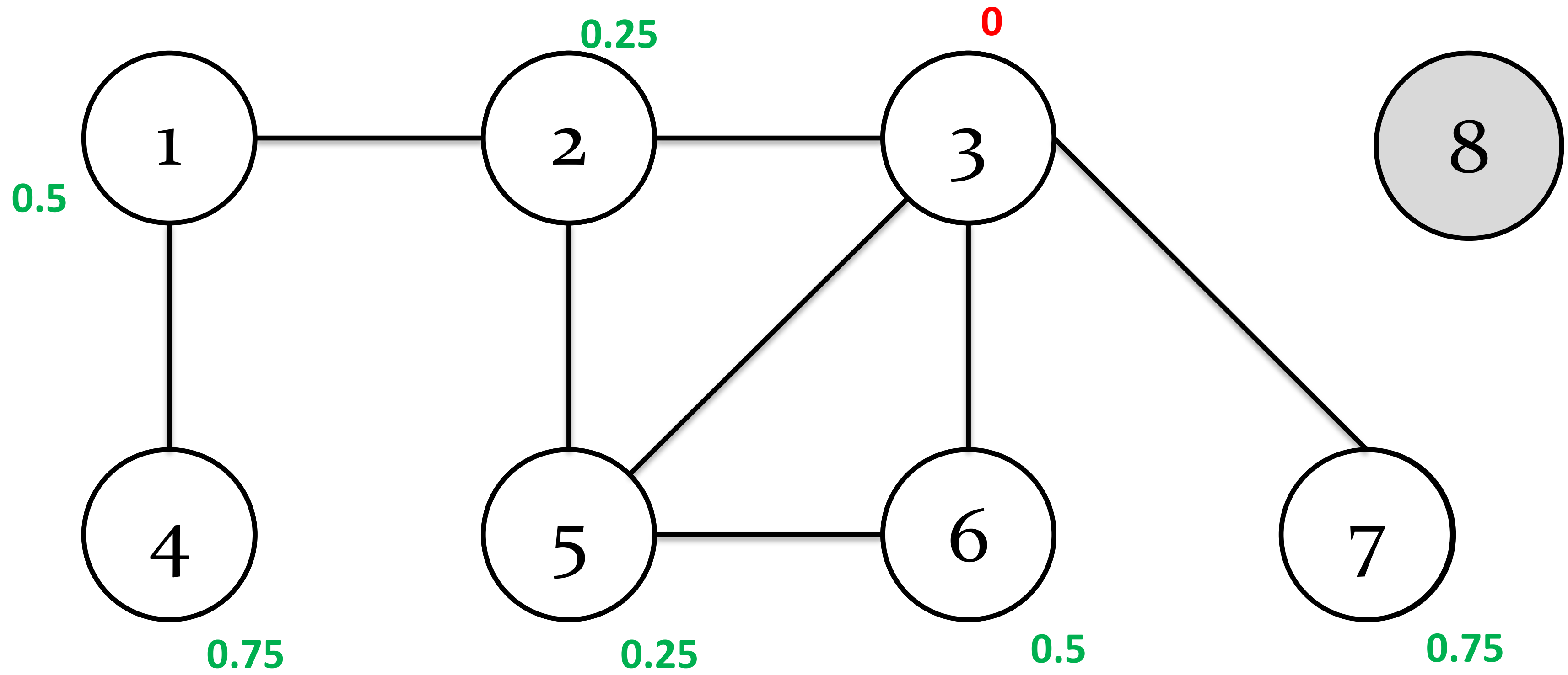
degree-weighted function: $t(v) = c \cdot \deg(v) = \mathbf{0.25} \cdot \boldsymbol{deg(v)}$

Iteration 0



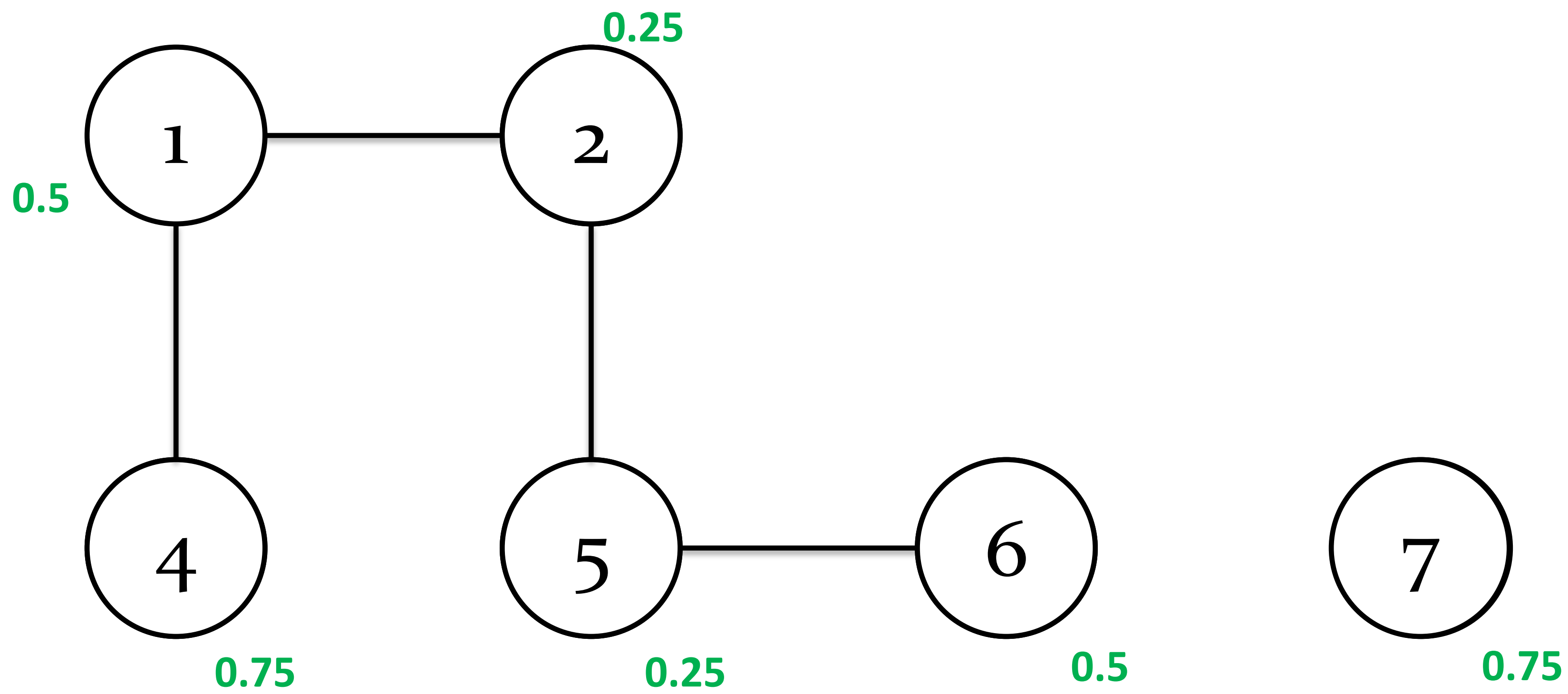
$D_0 = \{ 8 \}$
compute c = $min\{w(v)/deg(v)\}$ **= 0.25**
degree-weighted function: $t(v) = c \cdot \text{deg(v)} = 0.25 \cdot \boldsymbol{deg(v)}$
compute residual weight: w'(v) = w(v) – t(v)

Iteration 0



$D_0 = \{ 8 \}$
compute c = $min\{w(v)/deg(v)\}$ **= 0.25**
degree-weighted function: $t(v) = c \cdot \deg(v) = \mathbf{0.25} \cdot \boldsymbol{deg(v)}$
compute residual weight: w'(v) = w(v) – t(v)
$W_0 = \{ 3 \}$

Iteration 0



$D_0 = \{ 8 \}$

compute c = $min\{w(v)/deg(v)\}$ **= 0.25**

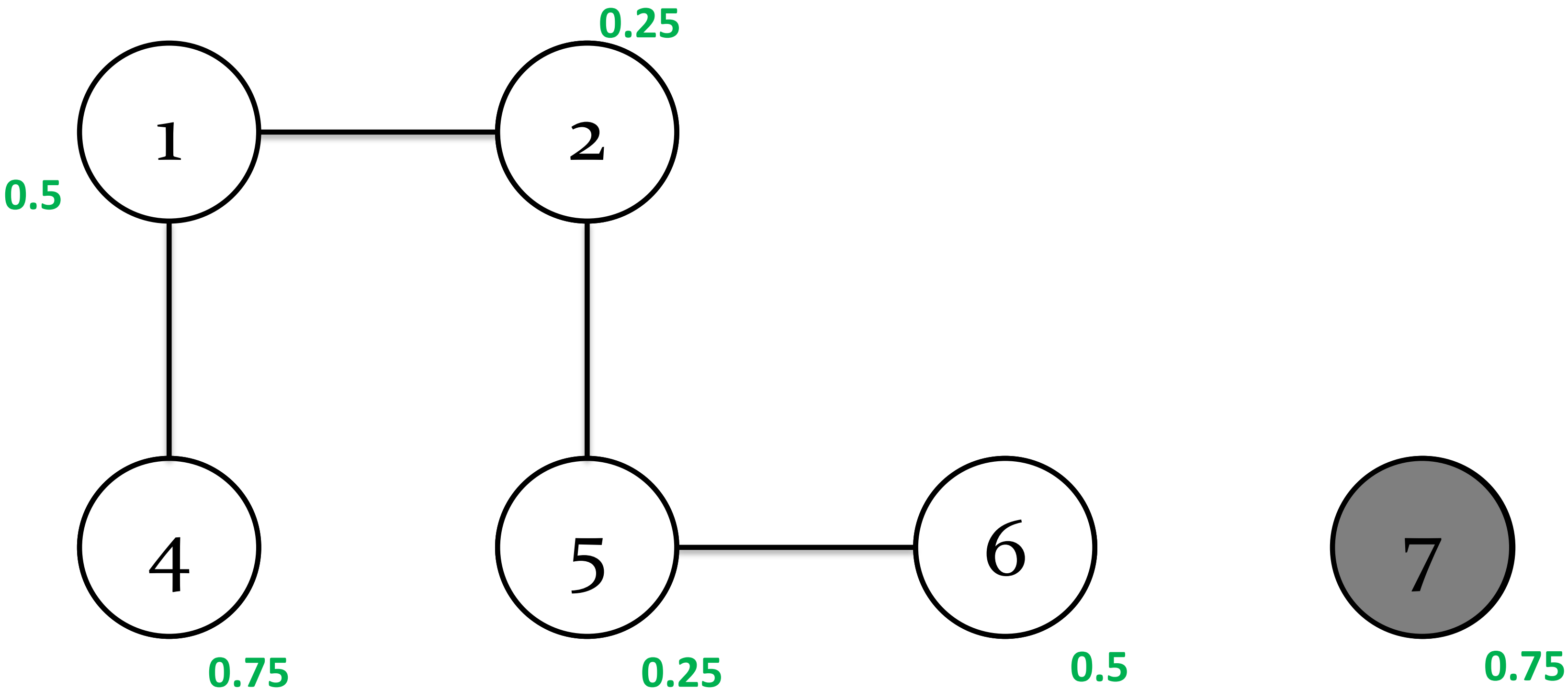
degree-weighted function: $t(v) = c \cdot \mathrm{deg(v)} = \mathbf{0.25} \cdot \boldsymbol{deg(v)}$

compute residual weight: w'(v) = w(v) – t(v)

$W_0 = \{ 3 \}$

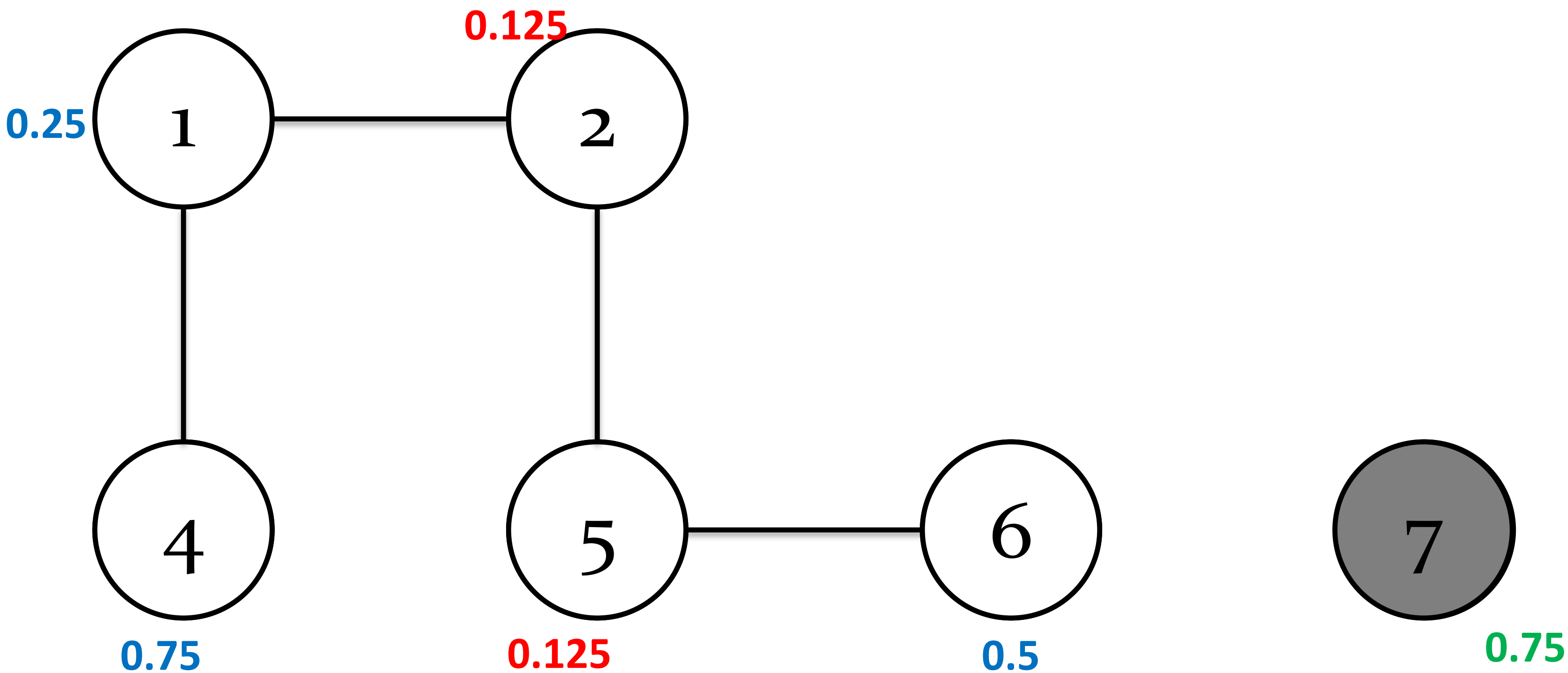
remove D_0 and W_0

Iteration 1



$D_1 = \{ 7 \}$

Iteration 1

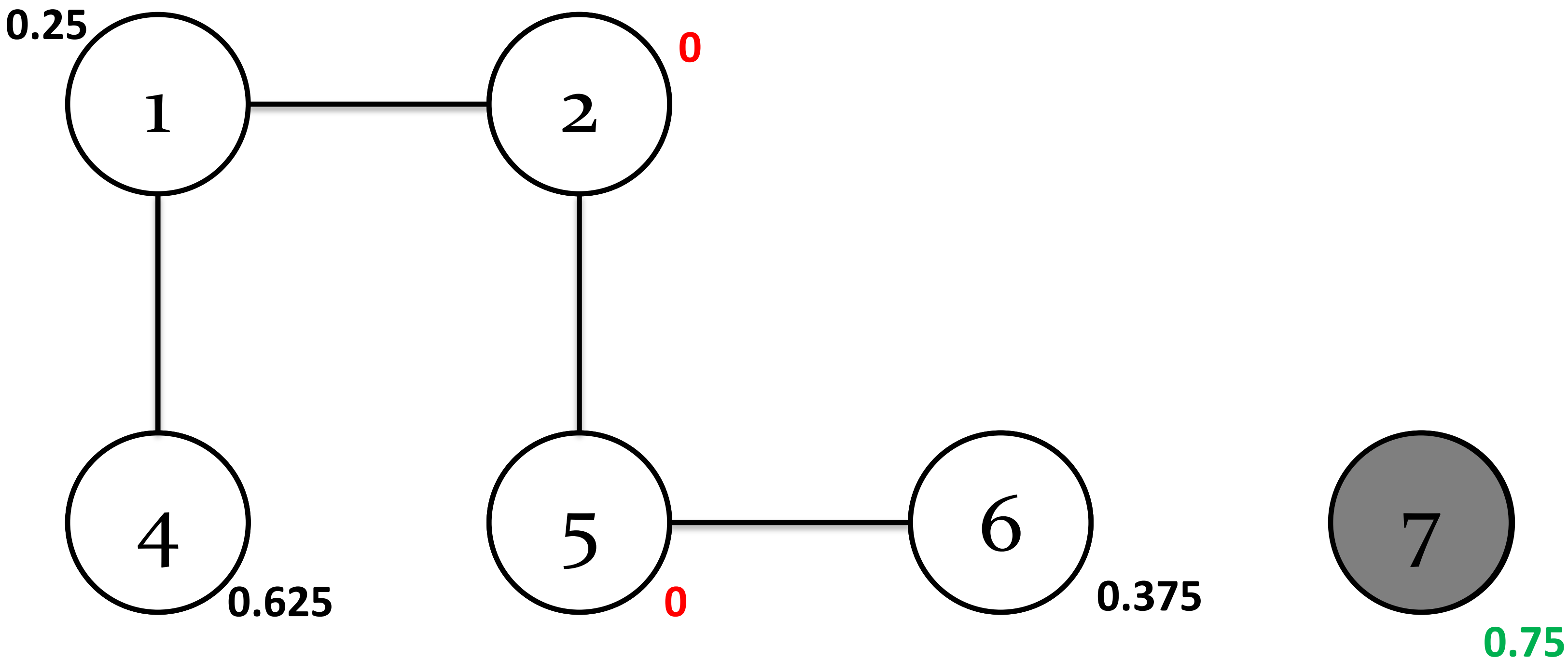


$D_1 = \{\ 7\ \}$

compute c = $min\{w(v)/deg(v)\}$ **= 0.125**

degree-weighted function: $t(v) = c \cdot \deg(v) = \mathbf{0.125} \cdot \boldsymbol{deg(v)}$

Iteration 1



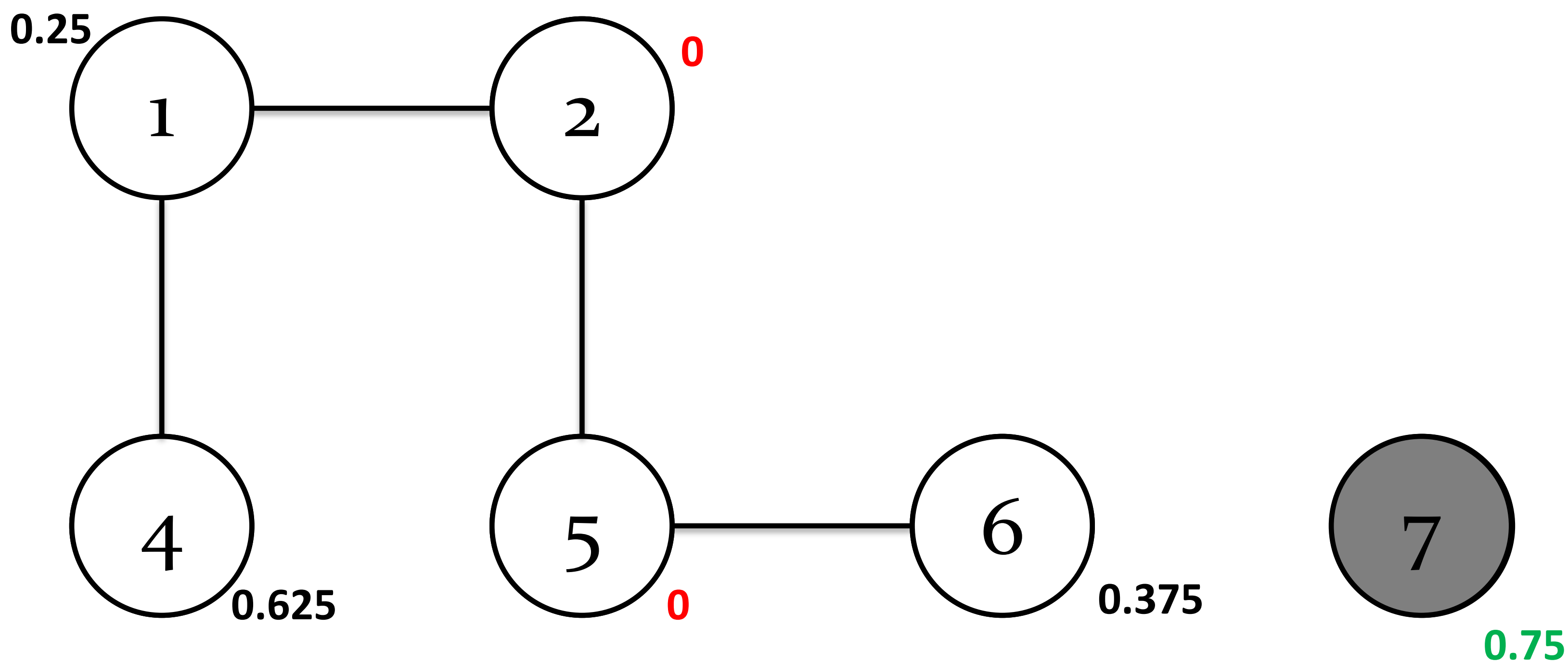
$D_1 = \{\ 7\ \}$

compute c = $min\{w(v)/deg(v)\}$ **= 0.125**

degree-weighted function: $t(v) = c \cdot \deg(v) = \mathbf{0.125} \cdot \boldsymbol{deg}(\boldsymbol{v})$

compute residual weight: w'(v) = w(v) – t(v)

Iteration 1



$D_1 = \{ 7 \}$
compute c = $min\{w(v)/deg(v)\}$ **= 0.125**
degree-weighted function: $t(v) = c \cdot \deg(v) = \mathbf{0.125} \cdot \boldsymbol{deg}(\boldsymbol{v})$
compute residual weight: w'(v) = w(v) – t(v)
$W_1 = \{ 2, 5 \}$

Iteration 1

$D_1 = \{ 7 \}$

compute c = $min\{w(v)/deg(v)\}$ **= 0.125**

degree-weighted function: $t(v) = c \cdot \text{deg}(v) = \mathbf{0.125} \cdot \boldsymbol{deg}(\boldsymbol{v})$

compute residual weight: w'(v) = w(v) – t(v)

$W_1 = \{ 2, 5 \}$

remove D_1 and W_1

Iteration 2

$D_2 = \{\ \color{red}{6}\ \}$

Iteration 2

$D_2 = \{\ 6\ \}$

compute c = $min\{w(v)/deg(v)\}$ **= 0.25**

degree-weighted function: $t(v) = c \cdot \text{deg(v)} = 0.25 \cdot \boldsymbol{deg(v)}$

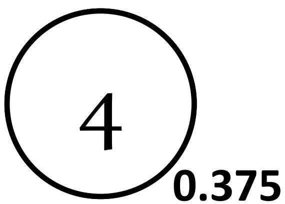
Iteration 2

$D_2 = \{\ 6\ \}$
compute c = $min\{w(v)/deg(v)\}$ **= 0.25**
degree-weighted function: $t(v) = c \cdot \deg(v) = \mathbf{0.25} \cdot \boldsymbol{deg(v)}$
compute residual weight: w'(v) = w(v) – t(v)

Iteration 2

$D_2 = \{ 6 \}$
compute c = $min\{w(v)/deg(v)\}$ **= 0.25**
degree-weighted function: $t(v) = c \cdot \deg(v) = \mathbf{0.25} \cdot \boldsymbol{deg}(\boldsymbol{v})$
compute residual weight: w'(v) = w(v) – t(v)
$W_2 = \{ 1 \}$

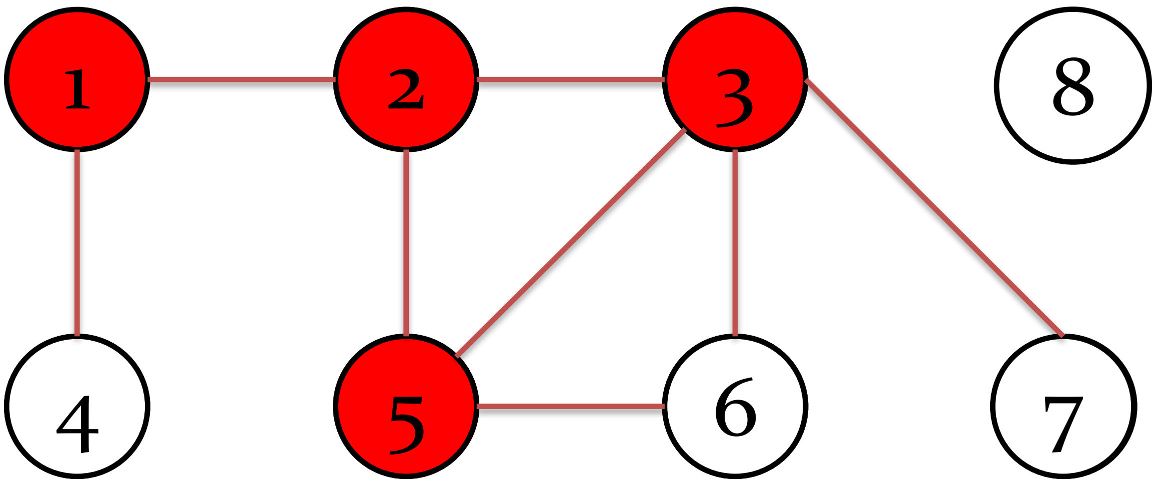
Iteration 2



$D_2 = \{ 6 \}$
compute c = $min\{w(v)/deg(v)\}$ **= 0.25**
degree-weighted function: $t(v) = c \cdot \mathrm{deg(v)} = \mathbf{0.25} \cdot \boldsymbol{deg(v)}$
compute residual weight: w'(v) = w(v) – t(v)
$W_2 = \{ 1 \}$
remove D_2 and W_2

Layering algorithm

all vertices of unit weight: w(v) = 1

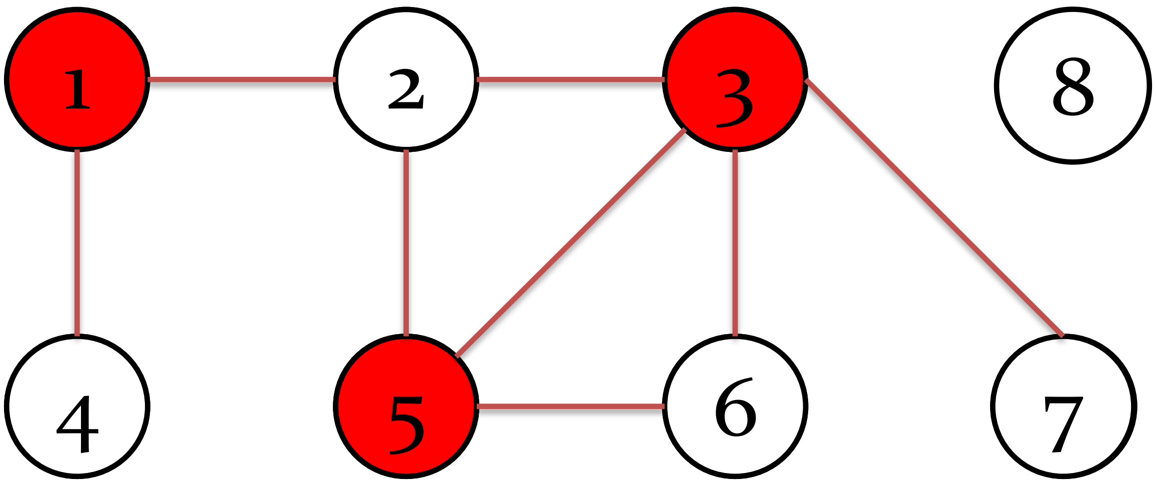


Vertex cover $\boldsymbol{C} = \boldsymbol{W_0} \cup \boldsymbol{W_1} \cup \boldsymbol{W_2} = \{\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{5}\}$

Total cost: $\mathbf{w}(\boldsymbol{C}) = \mathbf{4}$

Layering algorithm

all vertices of unit weight: w(v) = 1

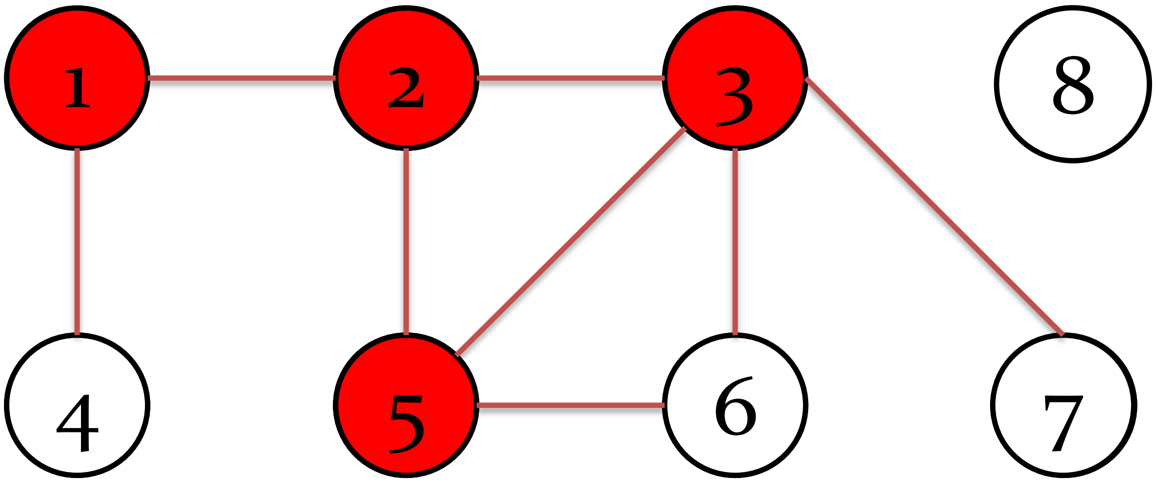


Optimal vertex cover $\boldsymbol{C}^{*} = \{\mathbf{1}, \mathbf{3}, \mathbf{5}\}$

Optimal cost: $\boldsymbol{OPT} = \mathbf{w}(\boldsymbol{C}^{*}) = \mathbf{3}$

Layering algorithm

all vertices of unit weight: w(v) = 1



Vertex cover $\boldsymbol{C} = \boldsymbol{W_0} \cup \boldsymbol{W_1} \cup \boldsymbol{W_2} = \{\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{5}\}$
Total cost: $\mathbf{w}(\boldsymbol{C}) = \mathbf{4}$

Optimal cost: $\boldsymbol{OPT} = \mathbf{w}(\boldsymbol{C}^*) = \mathbf{3}$

$\mathbf{w}(\boldsymbol{C}) < \mathbf{2 \cdot OPT}$

Approximation factor

The layering algorithm for vertex cover problem (assuming arbitrary vertex weights) achieves an approximation guarantee of factor **2**.

Approximation factor

The layering algorithm for vertex cover problem (assuming arbitrary vertex weights) achieves an approximation guarantee of factor **2**.

We need to prove:

1) set C is a vertex cover for G
2) $w(C) \leq 2 \cdot OPT$

Proof

1) set C is a vertex cover for G

layering algorithm terminates until all nodes are zero degree.

all edges have been already covered.

Proof

2) $w(C) \leq 2 \cdot OPT$

a vertex $v \in C$, if $v \in W_j$,

its weight can be decomposed as $\mathbf{w}(\boldsymbol{v}) = \sum_{i \leq j} \boldsymbol{t_i}(\boldsymbol{v})$

a vertex $v \in V - C$, if $v \in D_j$,

a lower bound on its weight is given by $\mathbf{w}(\boldsymbol{v}) \geq \sum_{i \leq j} \boldsymbol{t_i}(\boldsymbol{v})$

Let C^* be the optimal vertex cover,
in each layer i, $\boldsymbol{C}^* \cap \boldsymbol{G_i}$ is a vertex cover for G_i

recall the lemma :
if $w: V \rightarrow Q^+$ is a degree-weighted function, then $w(V) \leq 2 \cdot OPT$
by lemma, $\mathbf{t_i}(\mathbf{C} \cap \mathbf{G_i}) \leq \mathbf{2} \cdot \mathbf{t_i}(\mathbf{C}^* \cap \mathbf{G_i})$

$$\mathbf{w}(\mathbf{C}) = \sum_{\mathbf{i=0}}^{\mathbf{k-1}} \mathbf{t_i}(\mathbf{C} \cap \mathbf{G_i}) \leq \mathbf{2} \cdot \sum_{\mathbf{i=0}}^{\mathbf{k-1}} \mathbf{t_i}(\mathbf{C}^* \cap \mathbf{G_i}) \leq \mathbf{2} \cdot \mathbf{w}(\mathbf{C}^*)$$

Summary

	Vertex Cover Problem	Set Cover Problem
Layering Algorithm	**factor = 2**	**factor = f**

Introduction to LP-Duality

Zhu Xiaolu

Introduction to LP-Duality

- Linear Programming (LP)
- LP-Duality
- Theorem

a) Weak duality theorem

b) LP-duality theorem

c) Complementary slackness conditions

Why use LP ?

obtaining approximation algorithms using LP

- Rounding Techniques

to solve the linear program and convert the fractional solution into an integral solution.

- Primal-dual Schema

to use the dual of the LP-relaxation in the design of the algorithm.

analyzing combinatorially obtained approximation algorithms

- LP-duality theory is useful
- using the method of dual fitting

What is LP ?

Linear programming: the problem of optimizing (i.e., minimizing or maximizing) a linear function subject to linear inequality constraints.

$$\begin{aligned} \text{Minimize} \quad & 7X_1 + X_2 + 5X_3 \\ \text{Subject to} \quad & X_1 - X_2 + 3X_3 \geq 10 \\ & 5X_1 + 2X_2 - X_3 \geq 6 \\ & X_1, X_2, X_3 \geq 0 \end{aligned}$$

Objective function: The linear function that we want to optimize.

Feasible solution: An assignment of values to the variables that satisfies the inequalities. E.g. X=(2,1,3)

Cost: The value that the objective function gives to an assignment. E.g. $7 \cdot 2 + 1 + 5 \cdot 3$=30

What is LP-Duality ?

Minimize $7X_1 + X_2 + 5X_3$

Subject to $X_1 - X_2 + 3X_3 \geq 10 \longrightarrow a \geq b$

$5X_1 + 2X_2 - X_3 \geq 6 \longrightarrow c \geq d$

$X_1, X_2, X_3 \geq 0$

Abstract the constraints:

$a \geq b$

$c \geq d$

Find multipliers: $y_1, y_2 \geq 0$

$y_1 a \geq y_1 b$

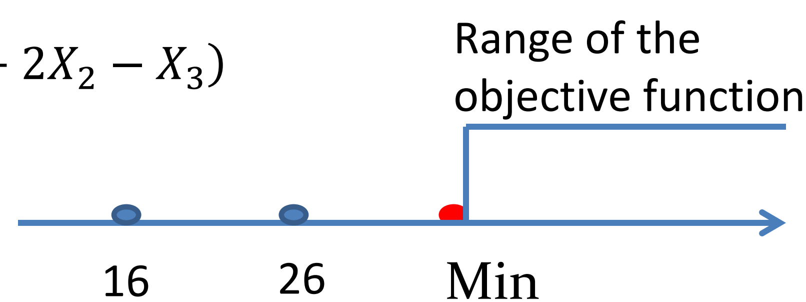
$y_2 c \geq y_2 d$

Objective function $\geq y_1 a + y_2 c \geq \boxed{y_1 b + y_2 d}$ (1)

As large as possible

$$7X_1 + X_2 + 5X_3 \geq \mathbf{1} \times (X_1 - X_2 + 3X_3) + \mathbf{1} \times (5X_1 + 2X_2 - X_3) = 6X_1 + X_2 + 2X_3 = 1 \times 10 + 1 \times 6 \geq 16$$

$$7X_1 + X_2 + 5X_3 \geq \mathbf{2} \times (X_1 - X_2 + 3X_3) + \mathbf{1} \times (5X_1 + 2X_2 - X_3) = 7X_1 + 5X_3 = 2 \times 10 + 1 \times 6 \geq 26$$



What is LP-Duality ?

The primal program

Minimize $7X_1 + X_2 + 5X_3$
Subject to $X_1 - X_2 + 3X_3 \geq 10$
$5X_1 + 2X_2 - X_3 \geq 6$
$X_1, X_2, X_3 \geq 0$

The dual program

Maximize $10Y_1 + 6Y_2$
Subject to $Y_1 + 5Y_2 \leq 7$
$-Y_1 + 2Y_2 \leq 1$
$3Y_1 - Y_2 \leq 5$
$Y_1, Y_2 \geq 0$

Minimize $\sum_{j=1}^{n} c_j x_j$
Subject to $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$,
$i = 1, \cdots, m$
$x_j \geq 0$, $j = 1, \cdots, n$
a_{ij}, b_i , c_j are given rational numbers

Maximize $\sum_{i=1}^{m} b_i y_i$
Subject to $\sum_{i=1}^{m} a_{ij} y_i \leq c_j$,
$j = 1, \cdots, n$
$y_i \geq 0$, $i = 1, \cdots, m$
a_{ij}, b_i , c_j are given rational numbers

Minimize $c_1X_1 + \cdots + c_nX_n$
Subject to $a_{1,1}X_1 + \cdots + a_{1,n}X_n \geq b_1$
$\cdots$
$a_{m,1}X_1 + \cdots + a_{m,n}X_n \geq b_m$
$X_1, \cdots, X_n \geq 0$

Minimize $7X_1 + X_2 + 5X_3$
Subject to $X_1 - X_2 + 3X_3 \geq 10$
$5X_1 + 2X_2 - X_3 \geq 6$
$X_1, X_2, X_3 \geq 0$

$$Y_1(a_{1,1}X_1 + \cdots + a_{1,n}X_n) + \cdots + Y_m(a_{m,1}X_1 + \cdots + a_{m,n}X_n) \geq Y_1b_1 + \cdots + Y_mb_m \quad (1)$$

$$(a_{1,1}Y_1 + \cdots + a_{m,1}Y_m)X_1 + \cdots + (a_{1,n}Y_1 + \cdots + a_{m,n}Y_m)\,X_n \geq Y_1b_1 + \cdots + Y_mb_m \quad (2)$$

Assume:

$$a_{1,1}Y_1 + \cdots + a_{m,1}Y_m \leq c_1$$
$$\cdots$$
$$a_{1,n}Y_1 + \cdots + a_{m,n}Y_m \leq c_n$$

$$c_1X_1 + \cdots + c_nX_n \geq (a_{1,1}Y_1 + \cdots + a_{m,1}Y_m)X_1 + \cdots + (a_{1,n}Y_1 + \cdots + a_{m,n}Y_m)\,X_n \quad (3)$$
$$\geq b_1Y_1 + \cdots + b_mY_m$$

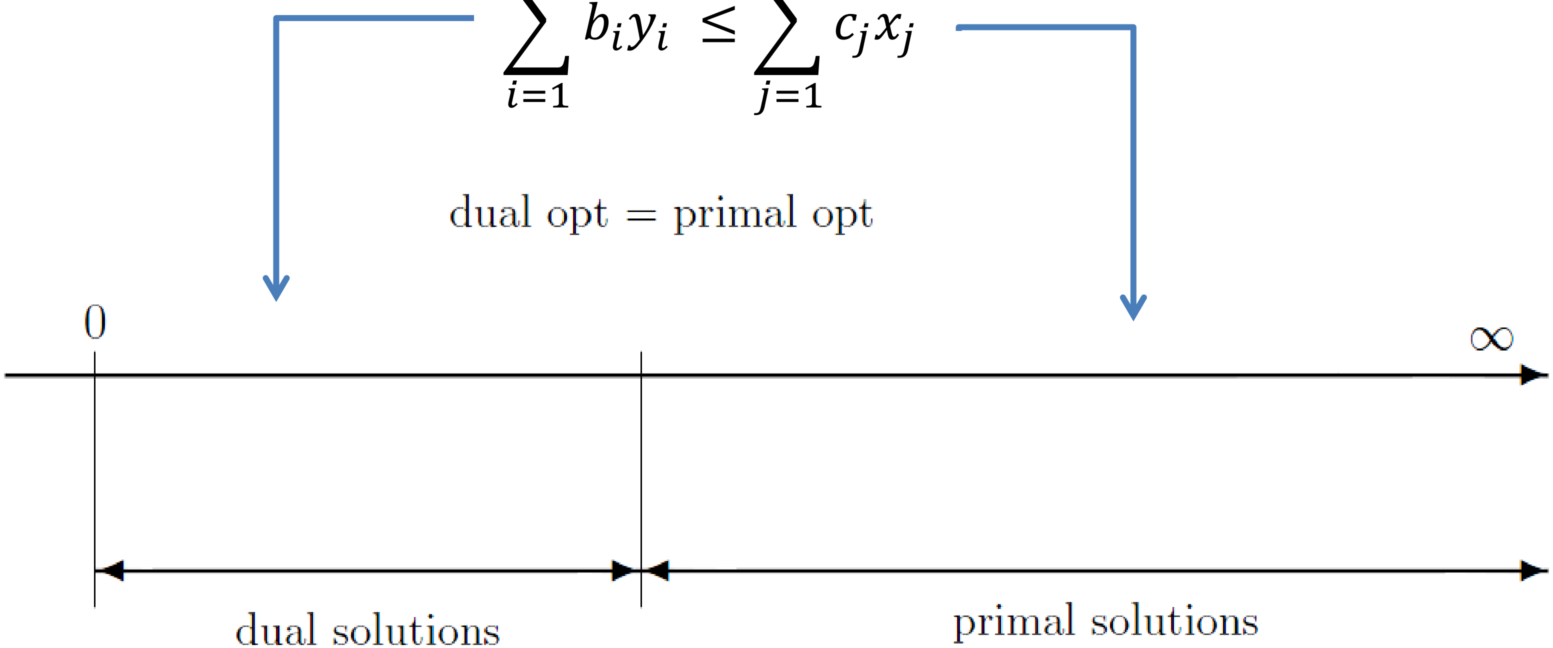
Maximize $b_1Y_1 + \cdots + b_mY_m$
Subject to $a_{1,1}Y_1 + \cdots + a_{m,1}Y_m \leq c_1$
$\cdots$
$a_{1,n}Y_1 + \cdots + a_{m,n}Y_m \leq c_n$
$Y_1, \cdots, Y_m \geq 0$

Maximize $10Y_1 + 6Y_2$
Subject to $Y_1 + 5Y_2 \leq 7$
$-Y_1 + 2Y_2 \leq 1$
$3Y_1 - Y_2 \leq 5$
$Y_1, Y_2 \geq 0$

Weak duality theorem

If $X=(X_1,\cdots,X_n)$ and $Y=(Y_1,\cdots,Y_m)$ are feasible solutions for the primal and dual program, respectively, then

$$\sum_{i=1}^{m} b_i y_i \leq \sum_{j=1}^{n} c_j x_j$$



Weak duality theorem - proof

Minimize $\sum_{j=1}^{n} c_j x_j$ Subject to $\sum_{j=1}^{n} a_{ij} x_j \ge b_i$, $i = 1, \cdots, m$ $x_j \ge 0$, $j = 1, \cdots, n$ a_{ij}, b_i , c_j are given rational numbers	Maximize $\sum_{i=1}^{m} b_i y_i$ Subject to $\sum_{i=1}^{m} a_{ij} y_i \le c_j$, $j = 1, \cdots, n$ $y_i \ge 0$, $i = 1, \cdots, m$ a_{ij}, b_i , c_j are given rational numbers

Since y is dual feasible and x_j's are nonnegative,

$$\sum_{j=1}^{n} c_j x_j \ge \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} y_i \right) x_j$$

Since x is primal feasible and y_j's are nonnegative,

$$\sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j \right) y_i \ge \sum_{i=1}^{m} b_i y_i$$

Obviously,

$$\sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} y_i \right) x_j = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j \right) y_i$$

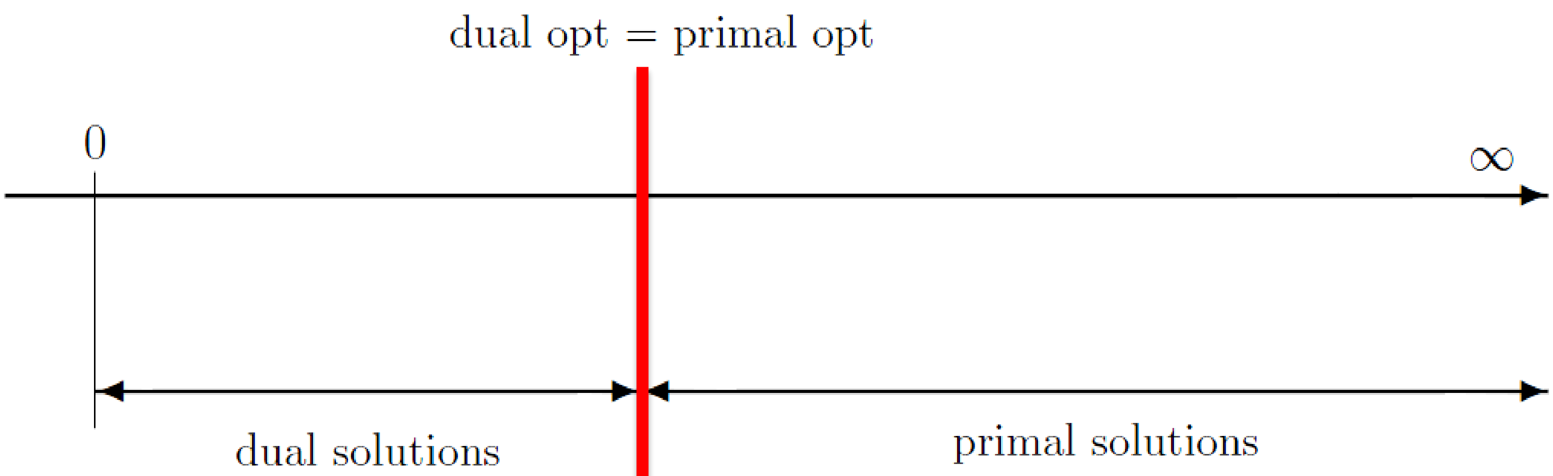
So, we proof that

$$\sum_{j=1}^{n} c_j x_j \ge \sum_{i=1}^{m} b_i y_i$$

LP-duality theorem

If $X^* = (X_1^*, \cdots, X_n^*)$ and $Y^* = (Y_1^*, \cdots, Y_m^*)$ are optimal solutions for the primal and dual programs, respectively, then

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*$$



Complementary slackness conditions

Let X and Y be primal and dual feasible solutions, respectively. Then, X and Y are both optimal iff all of the following conditions are satisfied:

- **Primal complementary slackness conditions**

For each $1 \le j \le n$: either $x_j = 0$ or $\sum_{i=1}^{m} a_{ij} y_i = c_j$;

And

- **Dual complementary slackness conditions**

For each $1 \le i \le m$: either $y_i = 0$ or $\sum_{j=1}^{n} a_{ij} x_j = b_i$.

Complementary slackness conditions -proof

Proof how these two conditions comes up
From the proof of weak duality theorem:

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} y_i \right) x_j \qquad (1)$$

$$\sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j \right) y_i \geq \sum_{i=1}^{m} b_i y_i \qquad (2)$$

By the LP-duality theorem, x and y are both optimal solutions iff:

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*$$

These happens iff (1) and (2) hold with equality:

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} y_i \right) x_j^* = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j \right) y_i^* = \sum_{i=1}^{m} b_i y_i^*$$

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} y_i\right) x_j^* = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j\right) y_i^* = \sum_{i=1}^{m} b_i y_i^*$$

$$\sum_{j=1}^{n} \left(c_j - \left(\sum_{i=1}^{m} a_{ij} y_i\right)\right) x_j^* = 0$$

Same Process

Dual complementary slackness conditions

For each $1 \leq j \leq n$

If $x_j^* > 0$ then $c_j - \left(\sum_{i=1}^{m} a_{ij} y_i\right) = 0$

If $c_j - \left(\sum_{i=1}^{m} a_{ij} y_i\right) > 0$ then $x_j^* = 0$

Primal complementary slackness conditions

For each $1 \leq j \leq n$: either $x_j = 0$ or $\sum_{i=1}^{m} a_{ij} y_i = c_j$

Set Cover via Dual Fitting

Wang Zixiao

Set Cover via Dual Fitting

- Dual Fitting: help analyze combinatorial algorithms using LP-duality theory

- Analysis of the greedy algorithm for the set cover problem

- Give a lower bound

Dual Fitting

- Minimization problem: combinatorial algorithm
- Linear programming relaxation, Dual
- Primal is ***fully paid for*** by the dual
- Infeasible → Feasible (shrunk with factor)
- Factor → Approximation guarantee

Formulation

U: universe

S: $\{S_1, S_2, \ldots, S_k\}$

C: ***S*** —> Q^+

Goal:sub-collection with minimum cost

$$\downarrow$$

$$\text{minimize} \quad \sum_{S \in \mathcal{S}} c(S) x_S, \quad x_S \in \{0, \ 1\}$$

$$\text{subject to} \quad \forall e \in U, \sum_{S : e \in S} x_S \geq 1$$

LP-Relaxation

- Motivation: NP-hard → Polynomial
- Integer program → Fractional program
- Letting x_S: $0 \leq x_S$

$$\textit{minimize} \quad \sum_{S \in \mathcal{S}} c(S) x_S$$

$$\textit{subject to} \quad \forall e \in U, \sum_{S : e \in S} x_S \geq 1$$

$$x_S \geq 0, \qquad S \in \mathcal{S}$$

LP Dual Problem

$$\textit{minimize} \quad \sum_{S \in \mathbf{S}} c(S) x_S$$

$$\textit{subject to} \quad \forall e \in U, \sum_{S: e \in S} x_S \geq 1$$

$$x_S \geq 0, \qquad S \in \mathbf{S}$$

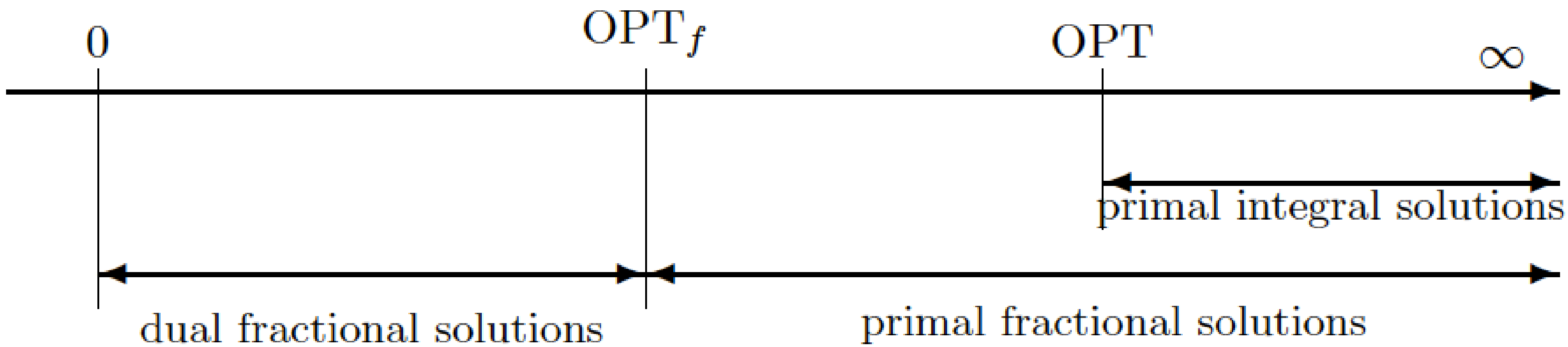
$$\downarrow$$

$$\textit{maximize} \quad \sum_{e \in U} y_e$$

$$\textit{subject to} \quad \forall S \in \mathbf{S}, \sum_{e:\ e \in S} y_e \leq c(S)$$

$$y_e \geq 0, \qquad e \in U$$

LP Dual Problem



OPT: cost of optimal integral set cover
OPT$_f$: cost of optimal fractional set cover

Solution

y_e = price(e)?

Not feasible!

U = {1, 2, 3}
S = {{1, 2, 3}, {1, 3}}
C({1, 2, 3}) = 3
C({1, 3}) = 1

- Iteration 1: {1, 3} chosen
  price(1) = price(3) = 0.5
- Iteration 2: {1, 2, 3} chosen
  price(2) = 3

price(1) + price(2) + price(3) = 4 > C({1, 2, 3})

violation

$$\text{maximize} \quad \sum_{e \in U} y_e$$

$$\text{subject to} \quad \forall S \in \mathbf{S}, \sum_{e:\ e \in S} y_e \leq c(S)$$

$$y_e \geq 0, \qquad e \in U$$

Solution

$$y_e = \frac{price(e)}{H_n}$$

*The vector **y** defined above is a feasible solution for the dual program*

$\Updownarrow$

There is no set S in **S** overpacked

Proof

Consider a set S∈**S**
| S | = k

Number the elements
in the order of being covered:
$e_1, e_2, \ldots, e_k$

Consider the iteration for e_i:
$e_1, e_2, \ldots, \underline{e_i}, \ldots, e_k$

At least k-i+1 elements
uncovered in S

Select S:

$$price(e_i) \le \frac{C(S)}{k-i+1}$$

Select S' :

$$price(e_i) < \frac{C(S)}{k-i+1}$$

$$price(e_i) \le \frac{C(S)}{k-i+1}$$

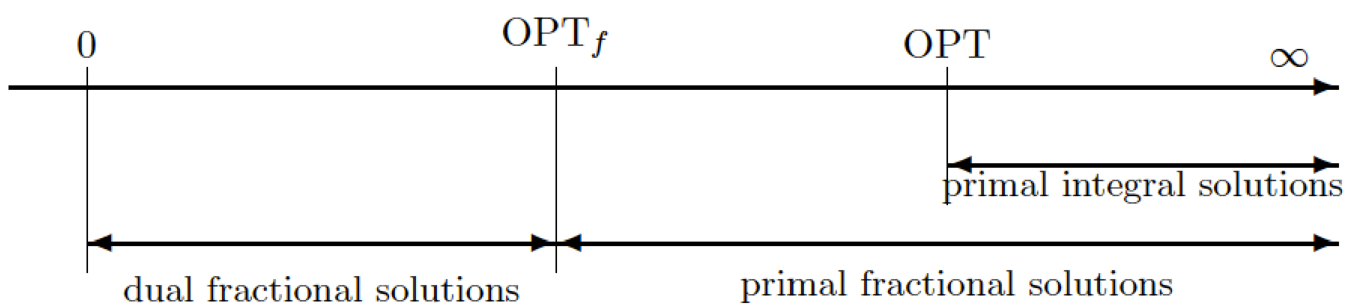
$$y_{e_i} = \frac{price(e_i)}{H_n} \le \frac{C(S)}{H_n} * \frac{1}{k-i+1}$$

$$\sum_{i=1}^{k} y_{e_i} \le \frac{C(S)}{H_n} * \left(\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{1}\right) = \frac{H_k}{H_n} * C(S) \le C(S)$$

The approximation of guarantee of the greedy set cover algorithm is H_n

Proof: The cost of the set cover picked is:

$$\sum_{e \in U} price(e) = H_n(\sum_{e \in U} y_e) \leq H_n * OPT_f \leq H_n OPT$$



Constrained Set Multicover

- Constrained Set Multicover: Each element e has to be covered r_e times

- Greedy algorithm: Pick the most cost-effective set until all elements have been covered by required times

Analysis of the greedy algorithm

IP formulation:

- minimize $\sum_{S \in \mathcal{S}} c(S) \cdot x_S$
- subject to:
  - $\sum_{S: e \in S} x_S \geq r_e, \; e \in \mathcal{U}$
  - $x_S \in \{0, 1\}, \; S \in \mathcal{S}$

LP relaxation:

- minimize $\sum_{S \in \mathcal{S}} c(S) \cdot x_S$
- subject to:
  - $\sum_{S: e \in S} x_S \geq r_e, \; e \in \mathcal{U}$
  - $-x_S \geq -1, \; S \in \mathcal{S}$
  - $x_S \geq 0, \; S \in \mathcal{S}$

Dual program:

- maximize $\sum_{e \in \mathcal{U}} r_e \cdot y_e - \sum_{S \in \mathcal{S}} z_S$
- subject to:
  - $\sum_{e \in S} y_e - z_S \leq c(S), \; S \in \mathcal{S}$
  - $y_e \geq 0, \; e \in \mathcal{U}$
  - $z_S \geq 0, \; S \in \mathcal{S}$

Analysis of the greedy algorithm

Consider the objective function value D of the dual solution (α, β), where $\alpha_e = \text{price}(e, r_e)$ for each $e \in \mathcal{U}$ and

$$\beta_S = \sum_{e \text{ covered by } S} \left(\text{price}(e, r_e) - \text{price}(e, j_e)\right),$$

for each $S \in \mathcal{S}$ that was picked by the algorithm and $\beta_S = 0$ otherwise.

$\mathrm{D} = \sum_{e \in \mathcal{U}} r_e \cdot \text{price}(e, r_e) -$
$\sum_{e \in \mathcal{U}} \left(r_e \cdot \text{price}(e, r_e) - \sum_{j=1}^{r_e} \text{price}(e, j) \right) = \mathrm{SOL}$

Analysis of the greedy alrogithm

Lemma 2 *The pair* $(\mathbf{y}, \mathbf{z})$ *where* $y_e = \frac{\alpha_e}{H_n}$ *and* $z_S = \frac{\beta_S}{H_n}$ *is a feasible solution for the dual program.*

Therefore, $\frac{\mathrm{D}}{H_n} \leq \mathrm{OPT_f} \Rightarrow \mathrm{SOL} \leq H_n \cdot \mathrm{OPT}$ which implies an approximation factor of H_n for the previous algorithm.

Analysis of the greedy algorithm

Proof

Consider $S \in \mathcal{S}$ and its elements $e_1, e_2, \ldots, e_i, \ldots, e_k$ in the order in which their requirements are covered by the algorithm.

Case 1: S is not picked

Just before the last copy of e_i is covered, S contains at least $k-i+1$ alive elements. So, price$(e_i, r_{e_i}) \leq \frac{c(S)}{k-i+1}$. Moreover, since $z_S = 0$ we get:

$$\sum_{e \in S} y_e - z_S \leq \frac{1}{H_n} \cdot \sum_{i=1}^{k} \frac{c(S)}{k-i+1} \leq c(S)$$

Analysis of the greedy algorithm

Case 2: S **is picked**

Assume that just before S is picked, k' of its elements are completely covered.

$$\sum_{e \in S} y_e - z_S = \frac{1}{H_n} \cdot \left(\sum_{i=1}^{k} \text{price}\left(e_i, r_{e_i}\right) - \sum_{i=k'+1}^{k} \left(\text{price}\left(e_i, r_{e_i}\right) - \text{price}\left(e_i, j_i\right)\right) \right)$$

$$= \frac{1}{H_n} \cdot \left(\sum_{i=1}^{k'} \text{price}\left(e_i, r_{e_i}\right) + \sum_{i=k'+1}^{k} \text{price}\left(e_i, j_i\right) \right) \leq c\left(S\right)$$

Rounding Applied to LP to solve Set Cover

Jiao Qing

Why rounding?

- Previous slides show LP-relaxation for the set cover problem, but for set cover real applications, they usually need integral solution.
- Next section will introduce two rounding algorithms and their approximation factor to OPT_f and OPT.

Rounding Algorithms

- A simple rounding algorithm
- Randomized rounding

A simple rounding algorithm

- Algorithm Description
  1. Find an optimal solution to the LP-relaxation.
  2. Pick all sets S for which $x_s \geq \frac{1}{f}$ in this solution.
- Apparently, its algorithm complexity equals to LP problem.

A simple rounding algorithm

- Proving that it solves set cover problem.
  1. Remember the LP-relaxation:

     minimize $\sum_{s \in \boldsymbol{S}} c(s) x_s$

     subject to $\sum_{s: e \in s} x_s \geq 1, \quad e \in U$

     $x_s \geq 0, S \in \mathbf{S}$
  2. Let C be the collection of picked sets. For $\forall e \in U$, e is in at most f sets, one of these sets must has $x_s \geq \frac{1}{f}$, therefore, this simple algorithm at least solve the set cover problem.

A simple rounding algorithm

- Firstly we prove that it solves set cover problem.
- Then we estimate its approximation factor to OPT_f and OPT is f.

Approximation Factor

- Estimating its approximation factor
  1. The rounding process increases x_s by a factor of at most f, and further it reduces the number of sets.
  2. Thus it gives a desired approximation guarantee of f.

Randomized Rounding

- This rounding views the LP-relaxation coefficient x_s of sets as probabilities.
- Using probabilities theory it proves this rounding method has approximation factor $O(\log n)$.

Randomized Rounding

- Algorithm Description:
  1. Find an optimal solution to the LP-relaxation.
  2. Independently picks $v(\log n)$ subsets of full set S.
  3. Get these subsets' union C', check whether C' is a valid set cover and has cost $\leq OPT_f \times 4v \log n$. If not, repeat the step 2 and 3 again.
  4. The expected number of repetitions needed at most 2.

- Apparently, its algorithm complexity equals to LP problem.

Proof Approximation Factor

- Next,we prove its approximate factor is $O(\log n)$.
  1. For each set S, its probability of being picked is p_s(equals to x_s coefficient).
  2. Let C be one collection of sets picked. The expected cost of C is:

$$E[cost(C)] = \sum_{s \in \boldsymbol{s}} p_r[s\ is\ picked] \times c_s = \sum_{s \in \boldsymbol{s}} p_s \times c_s = OPT_f$$

Proof Approximation Factor

3. Let us compute the probability that an element a is covered by C.
   - Suppose that a occurs in k sets of S, with the probabilities associated with these sets be $p_1, \ldots p_k$. Because their sum greater than 1. Using elementary calculus, we get:

     $$p_r[a \text{ is covered by } C] \geq 1 - \left(1 - \frac{1}{k}\right)^k \geq 1 - \frac{1}{e}$$

     and thus,

     $$p_r[a \text{ is not covered by } C] \leq \frac{1}{e}$$

     where e is the base of natural logarithms.
   - Hence each element is covered with constant probability by C.

Proof Approximation Factor

4. And then considering their union C', we get:

$$p_r[a\ is\ not\ covered\ by\ c'] \leq \left(\frac{1}{e}\right)^{v(\log n)} \leq \frac{1}{4n}$$ With $\left(\frac{1}{e}\right)^{v \log n} \leq \frac{1}{4n}$

5. Summing over all elements $a \in U$, we get

$$p_r[c'\ is\ not\ a\ valid\ set\ cover\ ] = 1 - \left(1 - \frac{1}{4n}\right)^n \leq \frac{1}{4}$$

6. With:

$E[cost(C')] \leq \sum_i E[cost(c_i)]$=$OPT_f \times v \log n = OPT_f \times v \log n$

Applying Markov's Inequality with t=$OPT_f \times 4v \log n$, we get:

$$p_r\left[cost(C') \geq OPT_f \times 4v \log n\right] \leq \frac{1}{4}$$

Proof Approximation Factor

7. With

$$p_r[c' \, is \; not \; a \; valid \; set \; cover\,] \leq \frac{1}{4}$$

$$p_r\left[cost(C') \geq OPT_f \times 4v \log n\right] \leq \frac{1}{4}$$

Hence,

$$p_r\left[C' \, is \; a \; valid \; set \; cover \; and \; has \; cost \leq OPT_f \times 4v \log n\right] \geq \frac{3}{4} \times \frac{3}{4} \geq \frac{1}{2}$$

Approximation Factor

- The chance one could find a set cover and has a cost smaller than $O(\log n)OPT_f$ is bigger than 50%, and the expected number of iteration is two.
- Thus this randomized rounding algorithm provides a factor $O(\log n)$ approximation, with a high probability guarantee.

Summary

- Those two algorithm provide us with factor f and $O(\log n)$ approximation, which remind us the factor of greedy and layering algorithms'.
- Even through LP-relaxation, right now we could only find algorithm with factor of f and $O(\log n)$.

Set Cover via the Primal-dual Schema

Zhang Hao

Primal-dual Schema?

- A broad outline for the algorithm

  The details have to be designed individually to specific problems

- Good approximation factors and good running times

Primal-dual Program(Recall)

The primal program

Minimize $\sum_{j=1}^{n} c_j x_j$

Subject to $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$,

$i = 1, \cdots, m$

$x_j \geq 0$, $j = 1, \cdots, n$

a_{ij}, b_i , c_j are given rational numbers

The dual program

Maximize $\sum_{i=1}^{m} b_i y_i$

Subject to $\sum_{i=1}^{m} a_{ij} y_i \leq c_j$,

$j = 1, \cdots, n$

$y_i \geq 0$, $i = 1, \cdots, m$

a_{ij}, b_i , c_j are given rational numbers

Standard Complementary slackness conditions (Recall)

Let **x** and **y** be primal and dual feasible solutions, respectively. Then, **x** and **y** are both optimal iff all of the following conditions are satisfied:

- **Primal complementary slackness conditions**

For each $1 \le j \le n$: either $x_j = 0$ or $\sum_{i=1}^{m} a_{ij} y_i = c_j$;
And

- **Dual complementary slackness conditions**

For each $1 \le i \le m$: either $y_i = 0$ or $\sum_{j=1}^{n} a_{ij} x_j = b_i$.

Relaxed Complementary slackness conditions

Let **x** and **y** be primal and dual feasible solutions, respectively.

- **Primal complementary slackness conditions**

For each $1 \leq j \leq n$: either $x_j = 0$ or $c_i/\alpha \leq \sum_{i=1}^{m} a_{ij}\, y_i \leq c_i$;

And

- **Dual complementary slackness conditions**

For each $1 \leq i \leq m$: either $y_i = 0$ or $b_i \leq \sum_{j=1}^{n} a_{ij} x_j \leq \beta b_i$.

*α, β => The optimality of **x** and **y** solutions*

If $\alpha = \beta = 1$ => standard complementary slackness conditions

Overview of the Schema

- Pick a primal *infeasible* solution **x**, and a dual *feasible* solution **y**, such that the slackness conditions are satisfied for chosen α and β (usually **x** = **0**, **y** = **0**).
- Iteratively improve the feasibility of **x** (integrally) and the optimality of **y**, such that the conditions remain satisfied, until **x** becomes feasible.

Proposition

An approximation guarantee of $\alpha\beta$ is achieved using this schema.

Proof:

$$\underbrace{\sum_{j=1}^{n} c_j x_j \le \alpha \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} y_i \right) x_j}_{\left({}^{c_i}/_{\alpha} \le \sum_{i=1}^{m} a_{ij}\, y_i\right)} = \underbrace{\alpha \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j \right) y_i \le \alpha\beta \sum_{i=1}^{m} b_i y_i}_{\left(\sum_{j=1}^{n} a_{ij} x_j \;\le\; \beta b_i\right)}$$

$$\underbrace{\sum_{j=1}^{n} c_j x_j \le \alpha\beta \sum_{i=1}^{m} b_i y_i}_{(\textit{primal and dual condition})} \le \alpha\beta \cdot OPT_f \le \alpha\beta \cdot OPT$$

Set Cover(Recall)

Standard Primal Program

minimize $\sum_{\mathrm{s} \in \boldsymbol{S}} \boldsymbol{c}(\mathrm{S})\, \boldsymbol{x}_{\mathrm{s}}$

subject to $\sum_{\boldsymbol{e}:\mathrm{e}\, \in \mathrm{S}} \boldsymbol{x}_{\mathrm{s}} \geq 1, e \in U$

$\boldsymbol{x}_{\mathrm{s}} \in \{0,1\}, S \in \boldsymbol{S}$

The Dual Program

maximize $\sum_{\mathrm{e} \in \boldsymbol{U}} \boldsymbol{y}_{\mathrm{e}}$

subject to $\sum_{\mathrm{e}:\mathrm{e}\, \in \mathrm{S}} \boldsymbol{y}_{\mathrm{e}} \leq c(S), S \in \boldsymbol{S}$

$\boldsymbol{y}_{\mathrm{e}} \geq 0, e \in \boldsymbol{U}$

Relaxed Complementary Slackness Conditions

Definition: Set S is called tight if $\sum_{e:e\in S} y_e = c(S)$

Set $\alpha = 1, \beta = f$ (to obtain approximation factor f)
Primal Conditions – "Pick only tight sets in the cover"

$$\forall \mathrm{S} \in \boldsymbol{S}: \boldsymbol{x}_{\mathrm{S}} \neq \mathbf{0} \Rightarrow \sum_{\boldsymbol{e}:\boldsymbol{e}\in \mathrm{S}} \boldsymbol{y}_{\boldsymbol{e}} = \boldsymbol{c}(\mathrm{S})$$

Dual Conditions – "Each e, $\boldsymbol{y}_{\boldsymbol{e}} \neq \mathbf{0}$, can be covered at most $\boldsymbol{f}$ times"
- **trivially satisfied**

$$\forall \boldsymbol{e}: \boldsymbol{y}_{\boldsymbol{e}} \neq \mathbf{0} \Rightarrow \sum_{\boldsymbol{S}:\boldsymbol{e}\in \boldsymbol{S}} \boldsymbol{x}_{\boldsymbol{S}} \leq \boldsymbol{f}$$

Algorithm (Set Cover – factor f)

- Initialization: $\mathbf{x} = \mathbf{0}$, $\mathbf{y} = \mathbf{0}$.
- Until all elements are covered, do

  Pick an uncovered element e, and raise y_e until some set goes tight.

  Pick all tight sets in the cover and update **x**.

  Declare all the elements occuring in these sets as "covered".
- Output the set cover **x**.

Theorem

The algorithm can achieve an approximation factor of f.

Proof

- Clearly, there will be no uncovered and no overpacked sets in the end. Thus, primal and dual solutions will be feasible.
- Since they satisfy the relaxed complementary slackness conditions with $\alpha = 1, \beta = f$, the approximation factor is f.

Conclusion

- Combinatorial algorithms are greedy and local.
- LP-based algorithms are global.

Conclusion

- Combinatorial algorithms
  - Greedy algorithms Factor: Hn
  - Layering algorithms Factor: f
- LP-based algorithms
  - Rounding Factor: f, Hn
  - Primal–Dual Schema Factor: f