# Introduction to Approximation Algorithms, part II

2-1 2023, Mikkel Abrahamsen,
Department of Computer Science

APPROX-SUBSET-SUM$(S, t, \varepsilon)$
  $L'_0 = [0]$
  for $k = 1, \ldots, n$
    $L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$
    $L'_k$=TRIM$(L'_k, \varepsilon/2n)$
    remove duplicates and elm.s $> t$
  return last$(L'_n)$

# Definition

**Def.:** An algorithm for an optimization problem has *approximation ratio* $\rho(n)$ if for every input of size $n$,

$$\max\left\{\frac{C}{C^*}, \frac{C^*}{C}\right\} \leq \rho(n).$$

How much can the solution be wrong that this algorithm produces?

# Definition

**Def.:** An algorithm for an optimization problem has *approximation ratio* $\rho(n)$ if for every input of size $n$,

$$\max\left\{\frac{C}{C^*}, \frac{C^*}{C}\right\} \le \rho(n).$$

$C^*$ :=cost(opt. sol.)

$C$ :=cost(produced sol.)

# Definition

**Def.:** An algorithm for an optimization problem has *approximation ratio* $\rho(n)$ if for every input of size $n$,

$$\max\left\{\boxed{\frac{C}{C^*}}, \boxed{\frac{C^*}{C}}\right\} \leq \rho(n).$$

$C^*$ :=cost(opt. sol.)

$C$ :=cost(produced sol.)

minimization problem

maximization problem

# Definition

**Def.:** An algorithm for an optimization problem has *approximation ratio* $\rho(n)$ if for every input of size $n$,

$$\max\left\{\boxed{\frac{C}{C^*}}, \boxed{\frac{C^*}{C}}\right\} \le \rho(n).$$

$C^*$ :=cost(opt. sol.)

$C$ :=cost(produced sol.)

minimization problem

maximization problem

**Today:** Examples of use of randomization, linear programming, and a fully polynomial time approximation scheme (FPTAS).

# Definition

**Def.:** An algorithm for an optimization problem has *approximation ratio* $\rho(n)$ if for every input of size $n$,

$$\max\left\{\boxed{\frac{C}{C^*}}, \boxed{\frac{C^*}{C}}\right\} \leq \rho(n).$$

$C^*$ :=cost(opt. sol.)

$C$ :=cost(produced sol.)

minimization problem

maximization problem

**Today:** Examples of use of randomization, linear programming, and a fully polynomial time approximation scheme (FPTAS).

$C := \mathbf{E}\,[\text{cost(produced sol.)}]$

The expected value of the cost of the produced solution

# 3-SAT

$$
\begin{aligned}
&(x_1 \vee x_7 \vee \neg x_9) \\
\wedge&(\neg x_7 \vee x_8 \vee x_9) \\
\wedge&(\neg x_2 \vee x_3 \vee \neg x_4) \\
&\vdots
\end{aligned}
$$

# 3-SAT

$$\left.\begin{aligned} &(x_1 \vee x_7 \vee \neg x_9) \\ \wedge&(\neg x_7 \vee x_8 \vee x_9) \\ \wedge&(\neg x_2 \vee x_3 \vee \neg x_4) \\ &\vdots \end{aligned}\right] n \text{ clauses}$$

# 3-SAT

A literal is either a variable or an negation of a variable.

So, in each clause, we don't have the same variable appearing twice.

Each clause has three *literals* involving three distinct variables.

$$\boxed{(x_1 \vee x_7 \vee \neg x_9)}$$
$$\wedge(\neg x_7 \vee x_8 \vee x_9)$$
$$\wedge(\neg x_2 \vee x_3 \vee \neg x_4)$$
$$\vdots$$

$n$ clauses

# 3-SAT

Each clause has three *literals*
involving three distinct variables.

$$
\begin{aligned}
&\boxed{(x_1 \vee x_7 \vee \neg x_9)} \\
\wedge&(\neg x_7 \vee x_8 \vee x_9) \\
\wedge&(\neg x_2 \vee x_3 \vee \neg x_4) \\
&\vdots
\end{aligned}
\quad \Bigg] \; n \text{ clauses}
$$

Decision version:
NP-complete!

# 3-SAT

Each clause has three *literals* involving three distinct variables.

Decision version: NP-complete!

$$\left.\begin{array}{l} \boxed{(x_1 \vee x_7 \vee \neg x_9)} \\ \wedge(\neg x_7 \vee x_8 \vee x_9) \\ \wedge(\neg x_2 \vee x_3 \vee \neg x_4) \\ \vdots \end{array}\right] n \text{ clauses}$$

MAX-3-SAT: Find assignment that maximizes the number of true clauses.

尽可能多的从句是true.

# Randomly assigning values

$\Phi$ is a MAX-3-SAT instance

RANDOM-ASSIGNMENT($\Phi$)
  for each variable $x_i$ of $\Phi$
    choose $x_i \in \{0, 1\}$ by flipping fair coin
  return assignment

# Randomly assigning values

$\Phi$ is a MAX-3-SAT instance

RANDOM-ASSIGNMENT($\Phi$)
  for each variable $x_i$ of $\Phi$
    choose $x_i \in \{0, 1\}$ by flipping fair coin
  return assignment

**Thm.:** RANDOM-ASSIGNMENT is a $8/7$-approximation algorithm.

# Randomly assigning values

$\Phi$ is a MAX-3-SAT instance

RANDOM-ASSIGNMENT($\Phi$)
  for each variable $x_i$ of $\Phi$
    choose $x_i \in \{0, 1\}$ by flipping fair coin
  return assignment

**Thm.:** RANDOM-ASSIGNMENT is a $8/7$-approximation algorithm.

*Proof:* Let $\Phi = C_1 \wedge \ldots \wedge C_n$. Consider $C_i = \ell_1 \vee \ell_2 \vee \ell_3$.

# Randomly assigning values

$\Phi$ is a MAX-3-SAT instance

RANDOM-ASSIGNMENT($\Phi$)
  for each variable $x_i$ of $\Phi$
    choose $x_i \in \{0, 1\}$ by flipping fair coin
  return assignment

**Thm.:** RANDOM-ASSIGNMENT is a $8/7$-approximation algorithm.

*Proof:* Let $\Phi = C_1 \wedge \ldots \wedge C_n$. Consider $C_i = \ell_1 \vee \ell_2 \vee \ell_3$.

Clause $C_i$ not satisfied $\iff$

# Randomly assigning values

$\Phi$ is a MAX-3-SAT instance

RANDOM-ASSIGNMENT($\Phi$)
  for each variable $x_i$ of $\Phi$
    choose $x_i \in \{0, 1\}$ by flipping fair coin
  return assignment

**Thm.:** RANDOM-ASSIGNMENT is a $8/7$-approximation algorithm.

*Proof:* Let $\Phi = C_1 \wedge \ldots \wedge C_n$. Consider $C_i = \ell_1 \vee \ell_2 \vee \ell_3$.

Clause $C_i$ not satisfied $\Longleftrightarrow \neg\ell_1 \wedge \neg\ell_2 \wedge \neg\ell_3$.

All of them should be false

# Randomly assigning values

$\Phi$ is a MAX-3-SAT instance

RANDOM-ASSIGNMENT($\Phi$)
  for each variable $x_i$ of $\Phi$
    choose $x_i \in \{0, 1\}$ by flipping fair coin
  return assignment

**Thm.:** RANDOM-ASSIGNMENT is a $8/7$-approximation algorithm.

*Proof:* Let $\Phi = C_1 \wedge \ldots \wedge C_n$. Consider $C_i = \ell_1 \vee \ell_2 \vee \ell_3$.

Clause $C_i$ not satisfied $\iff \neg\ell_1 \wedge \neg\ell_2 \wedge \neg\ell_3$.

$\mathbf{Pr}\left[\neg C_i\right] = \mathbf{Pr}\left[\neg\ell_1\right] \cdot \mathbf{Pr}\left[\neg\ell_2\right] \cdot \mathbf{Pr}\left[\neg\ell_2\right]$

# Randomly assigning values

$\Phi$ is a MAX-3-SAT instance

RANDOM-ASSIGNMENT($\Phi$)
  for each variable $x_i$ of $\Phi$
    choose $x_i \in \{0, 1\}$ by flipping fair coin
  return assignment

**Thm.:** RANDOM-ASSIGNMENT is a $8/7$-approximation algorithm.

*Proof:* Let $\Phi = C_1 \wedge \ldots \wedge C_n$. Consider $C_i = \ell_1 \vee \ell_2 \vee \ell_3$.

Clause $C_i$ not satisfied $\iff \neg\ell_1 \wedge \neg\ell_2 \wedge \neg\ell_3$.

$$\mathbf{Pr}\left[\neg C_i\right] = \mathbf{Pr}\left[\neg\ell_1\right] \cdot \mathbf{Pr}\left[\neg\ell_2\right] \cdot \mathbf{Pr}\left[\neg\ell_2\right]$$

variables in $C_i$ chosen independently

# Randomly assigning values

$\Phi$ is a MAX-3-SAT instance

RANDOM-ASSIGNMENT($\Phi$)

for each variable $x_i$ of $\Phi$

choose $x_i \in \{0, 1\}$ by flipping fair coin

return assignment

**Thm.:** RANDOM-ASSIGNMENT is a $8/7$-approximation algorithm.

*Proof:* Let $\Phi = C_1 \wedge \ldots \wedge C_n$. Consider $C_i = \ell_1 \vee \ell_2 \vee \ell_3$.

Clause $C_i$ not satisfied $\iff \neg\ell_1 \wedge \neg\ell_2 \wedge \neg\ell_3$.

$$\mathbf{Pr}\left[\neg C_i\right] = \mathbf{Pr}\left[\neg\ell_1\right] \cdot \mathbf{Pr}\left[\neg\ell_2\right] \cdot \mathbf{Pr}\left[\neg\ell_2\right] = (\tfrac{1}{2})^3 = 1/8$$

variables in $C_i$ chosen independently

# Randomly assigning values

$\Phi$ is a MAX-3-SAT instance

RANDOM-ASSIGNMENT($\Phi$)
  for each variable $x_i$ of $\Phi$
    choose $x_i \in \{0, 1\}$ by flipping fair coin
  return assignment

**Thm.:** RANDOM-ASSIGNMENT is a $8/7$-approximation algorithm.

*Proof:* Let $\Phi = C_1 \wedge \ldots \wedge C_n$. Consider $C_i = \ell_1 \vee \ell_2 \vee \ell_3$.

Clause $C_i$ not satisfied $\iff \neg\ell_1 \wedge \neg\ell_2 \wedge \neg\ell_3$.

$$\mathbf{Pr}\left[\neg C_i\right] = \mathbf{Pr}\left[\neg\ell_1\right] \cdot \mathbf{Pr}\left[\neg\ell_2\right] \cdot \mathbf{Pr}\left[\neg\ell_2\right] = (\tfrac{1}{2})^3 = 1/8$$

variables in $C_i$ chosen independently

$$\mathbf{Pr}\left[C_i\right] = 1 - \mathbf{Pr}\left[\neg C_i\right] = 1 - 1/8 = 7/8$$

# Randomly assigning values

$\Phi$ is a MAX-3-SAT instance

RANDOM-ASSIGNMENT($\Phi$)
  for each variable $x_i$ of $\Phi$
    choose $x_i \in \{0,1\}$ by flipping fair coin
  return assignment

**Thm.:** RANDOM-ASSIGNMENT is a $8/7$-approximation algorithm.

*Proof:* Let $\Phi = C_1 \wedge \ldots \wedge C_n$. Consider $C_i = \ell_1 \vee \ell_2 \vee \ell_3$.

Clause $C_i$ not satisfied $\iff \neg\ell_1 \wedge \neg\ell_2 \wedge \neg\ell_3$.

$$\mathbf{Pr}\left[\neg C_i\right] = \mathbf{Pr}\left[\neg \ell_1\right] \cdot \mathbf{Pr}\left[\neg \ell_2\right] \cdot \mathbf{Pr}\left[\neg \ell_2\right] = \left(\tfrac{1}{2}\right)^3 = 1/8$$

variables in $C_i$ chosen independently

$$\mathbf{Pr}\left[C_i\right] = 1 - \mathbf{Pr}\left[\neg C_i\right] = 1 - 1/8 = 7/8$$

$X := \sum_{i=1}^{n} \left[C_i\right] = $ #satisfied clauses

# Randomly assigning values

$\Phi$ is a MAX-3-SAT instance

RANDOM-ASSIGNMENT($\Phi$)
  for each variable $x_i$ of $\Phi$
    choose $x_i \in \{0, 1\}$ by flipping fair coin
  return assignment

**Thm.:** RANDOM-ASSIGNMENT is a $8/7$-approximation algorithm.

*Proof:* Let $\Phi = C_1 \wedge \ldots \wedge C_n$. Consider $C_i = \ell_1 \vee \ell_2 \vee \ell_3$.

Clause $C_i$ not satisfied $\iff \neg\ell_1 \wedge \neg\ell_2 \wedge \neg\ell_3$.

$$\mathbf{Pr}\left[\neg C_i\right] = \mathbf{Pr}\left[\neg \ell_1\right] \cdot \mathbf{Pr}\left[\neg \ell_2\right] \cdot \mathbf{Pr}\left[\neg \ell_2\right] = (\tfrac{1}{2})^3 = 1/8$$

variables in $C_i$ chosen independently

Linearity of expectation

$$\mathbf{Pr}\left[C_i\right] = 1 - \mathbf{Pr}\left[\neg C_i\right] = 1 - 1/8 = 7/8$$

$X := \sum_{i=1}^{n} [C_i] = \#$satisfied clauses

probability → true

$$\mathbf{E}\left[X\right] = \mathbf{E}\left[\sum_{i=1}^{n} [C_i]\right] = \sum_{i=1}^{n} \mathbf{E}\left[C_i\right] = \sum_{i=1}^{n} \frac{7}{8} = 7n/8$$

当我们用随机算法时，用E来评估
cost

# Randomly assigning values

$\Phi$ is a MAX-3-SAT instance

RANDOM-ASSIGNMENT($\Phi$)
  for each variable $x_i$ of $\Phi$
    choose $x_i \in \{0,1\}$ by flipping fair coin
  return assignment

**Thm.:** RANDOM-ASSIGNMENT is a $8/7$-approximation algorithm.

*Proof:* Let $\Phi = C_1 \wedge \ldots \wedge C_n$. Consider $C_i = \ell_1 \vee \ell_2 \vee \ell_3$.

Clause $C_i$ not satisfied $\iff \neg\ell_1 \wedge \neg\ell_2 \wedge \neg\ell_3$.

$$\mathbf{Pr}\left[\neg C_i\right] = \mathbf{Pr}\left[\neg\ell_1\right] \cdot \mathbf{Pr}\left[\neg\ell_2\right] \cdot \mathbf{Pr}\left[\neg\ell_2\right] = \left(\tfrac{1}{2}\right)^3 = 1/8$$

variables in $C_i$ chosen independently

Linearity of expectation

$$\mathbf{Pr}\left[C_i\right] = 1 - \mathbf{Pr}\left[\neg C_i\right] = 1 - 1/8 = 7/8$$

$X := \sum_{i=1}^{n} [C_i] = $ #satisfied clauses

$$\mathbf{E}\left[X\right] = \mathbf{E}\left[\sum_{i=1}^{n} [C_i]\right] = \sum_{i=1}^{n} \mathbf{E}\left[C_i\right] = \sum_{i=1}^{n} \frac{7}{8} = 7n/8$$

Approximation ratio: $\boxed{\dfrac{\mathcal{C}^*}{\mathcal{C}}} = \dfrac{\mathcal{C}^*}{7n/8} \leq$

maximization

# Randomly assigning values

$\Phi$ is a MAX-3-SAT instance

RANDOM-ASSIGNMENT($\Phi$)
  for each variable $x_i$ of $\Phi$
    choose $x_i \in \{0,1\}$ by flipping fair coin
  return assignment

**Thm.:** RANDOM-ASSIGNMENT is a $8/7$-approximation algorithm.

*Proof:* Let $\Phi = C_1 \wedge \ldots \wedge C_n$. Consider $C_i = \ell_1 \vee \ell_2 \vee \ell_3$.

Clause $C_i$ not satisfied $\iff \neg\ell_1 \wedge \neg\ell_2 \wedge \neg\ell_3$.

$$\mathbf{Pr}\left[\neg C_i\right] = \mathbf{Pr}\left[\neg \ell_1\right] \cdot \mathbf{Pr}\left[\neg \ell_2\right] \cdot \mathbf{Pr}\left[\neg \ell_2\right] = \left(\tfrac{1}{2}\right)^3 = 1/8$$

variables in $C_i$ chosen independently

Linearity of expectation

$$\mathbf{Pr}\left[C_i\right] = 1 - \mathbf{Pr}\left[\neg C_i\right] = 1 - 1/8 = 7/8$$

$X := \sum_{i=1}^{n} [C_i] = \#$satisfied clauses

$$\mathbf{E}\left[X\right] = \mathbf{E}\left[\sum_{i=1}^{n} [C_i]\right] = \sum_{i=1}^{n} \mathbf{E}\left[C_i\right] = \sum_{i=1}^{n} \frac{7}{8} = 7n/8$$

Approximation ratio: $\frac{\mathcal{C}^*}{\mathcal{C}} = \frac{\mathcal{C}^*}{7n/8} \le \frac{n}{7n/8} = 8/7$

# Bonus: Derandomization

**Goal:** Find deterministic alg. that satisfies $7n/8$ clauses.

# Bonus: Derandomization

**Goal:** Find deterministic alg. that satisfies $7n/8$ clauses.

**Recall:** $\mathbf{E}\left[X\right] = 7n/8$, where $X := \sum_{i=1}^{n} \left[C_i\right] = \#$satisfied clauses

# Bonus: Derandomization

**Goal:** Find deterministic alg. that satisfies $7n/8$ clauses.

**Recall:** $\mathbf{E}[X] = 7n/8$, where $X := \sum_{i=1}^{n} [C_i] = $ #satisfied clauses

Assignment guaranteed to exist by *the probabilistic method*.

# Bonus: Derandomization

**Goal:** Find deterministic alg. that satisfies $7n/8$ clauses.

**Recall:** $\mathbf{E}[X] = 7n/8$, where $X := \sum_{i=1}^{n} [C_i] =$ #satisfied clauses

Assignment guaranteed to exist by *the probabilistic method*.

Clause $C$ with $3$ literals: $\mathbf{Pr}[C] = 1 - (\frac{1}{2})^3 = 7/8$.

# Bonus: Derandomization

**Goal:** Find deterministic alg. that satisfies $7n/8$ clauses.

**Recall:** $\mathbf{E}\left[X\right] = 7n/8$, where $X := \sum_{i=1}^{n} \left[C_i\right] = \#$satisfied clauses

Assignment guaranteed to exist by *the probabilistic method*.

Clause $C$ with $3$ literals: $\mathbf{Pr}\left[C\right] = 1 - (\frac{1}{2})^3 = 7/8$.

Clause $C$ with $2$ literals: $\mathbf{Pr}\left[C\right] = 1 - (\frac{1}{2})^2 = 3/4$.

# Bonus: Derandomization

**Goal:** Find deterministic alg. that satisfies $7n/8$ clauses.

**Recall:** $\mathbf{E}[X] = 7n/8$, where $X := \sum_{i=1}^{n} [C_i] =$ #satisfied clauses

Assignment guaranteed to exist by *the probabilistic method*.

Clause $C$ with $3$ literals: $\mathbf{Pr}[C] = 1 - (\frac{1}{2})^3 = 7/8$.

Clause $C$ with $2$ literals: $\mathbf{Pr}[C] = 1 - (\frac{1}{2})^2 = 3/4$.

Clause $C$ with $1$ literal: $\mathbf{Pr}[C] = 1 - \frac{1}{2} = 1/2$.

# Bonus: Derandomization

**Goal:** Find deterministic alg. that satisfies $7n/8$ clauses.

**Recall:** $\mathbf{E}[X] = 7n/8$, where $X := \sum_{i=1}^{n} [C_i] = $ #satisfied clauses

Assignment guaranteed to exist by *the probabilistic method*.

Clause $C$ with $3$ literals: $\mathbf{Pr}[C] = 1 - (\frac{1}{2})^3 = 7/8$.

Clause $C$ with $2$ literals: $\mathbf{Pr}[C] = 1 - (\frac{1}{2})^2 = 3/4$.

Clause $C$ with $1$ literal: $\mathbf{Pr}[C] = 1 - \frac{1}{2} = 1/2$.

Clause $C$ with $0$ literals: $\mathbf{Pr}[C] = 1 - (\frac{1}{2})^0 = 0$.

# Bonus: Derandomization

**Goal:** Find deterministic alg. that satisfies $7n/8$ clauses.

**Recall:** $\mathbf{E}[X] = 7n/8$, where $X := \sum_{i=1}^{n} [C_i] = \#$satisfied clauses

Assignment guaranteed to exist by *the probabilistic method*.

Clause $C$ with 3 literals: $\mathbf{Pr}[C] = 1 - (\frac{1}{2})^3 = 7/8$.

Clause $C$ with 2 literals: $\mathbf{Pr}[C] = 1 - (\frac{1}{2})^2 = 3/4$.

Clause $C$ with 1 literal: $\mathbf{Pr}[C] = 1 - \frac{1}{2} = 1/2$.

Clause $C$ with 0 literals: $\mathbf{Pr}[C] = 1 - (\frac{1}{2})^0 = 0$.

DETERMINISTIC-ASSIGNMENT($\Phi$)

for $i = 1, \ldots, \boxed{m}$ $m = \#$variables in $\Phi$

  $x_i := 0$

  compute $D := \mathbf{E}[X \mid \text{chosen values of } x_1, \ldots, x_i]$

  if $D < 7n/8$

    $x_i := 1$

return assignment

# Bonus: Derandomization

**Goal:** Find deterministic alg. that satisfies $7n/8$ clauses.

**Recall:** $\mathbf{E}[X] = 7n/8$, where $X := \sum_{i=1}^{n} [C_i] = \#$satisfied clauses

Assignment guaranteed to exist by *the probabilistic method*.

Clause $C$ with $3$ literals: $\mathbf{Pr}[C] = 1 - (\frac{1}{2})^3 = 7/8$.

Clause $C$ with $2$ literals: $\mathbf{Pr}[C] = 1 - (\frac{1}{2})^2 = 3/4$.

Clause $C$ with $1$ literal: $\mathbf{Pr}[C] = 1 - \frac{1}{2} = 1/2$.

Clause $C$ with $0$ literals: $\mathbf{Pr}[C] = 1 - (\frac{1}{2})^0 = 0$.

DETERMINISTIC-ASSIGNMENT($\Phi$)

for $i = 1, \ldots, \boxed{m}$ $\qquad m = \#$variables in $\Phi$

  $x_i := 0$

  compute $D := \mathbf{E}[X \mid \text{chosen values of } x_1, \ldots, x_i]$

  if $D < 7n/8$

    $x_i := 1$

return assignment

Method of conditional probabilities

# Example

DETERMINISTIC-ASSIGNMENT($\Phi$)
  for $i = 1, \ldots, m$
    $x_i := 0$
    compute $D := \mathbf{E}[X \mid \text{chosen values of } x_1, \ldots, x_i]$
    if $D < 7n/8$
      $x_i := 1$
  return assignment

$\boxed{\text{Method of conditional probabilities}}$

$$\Phi = (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee \neg x_4 \vee x_5) \wedge (x_1 \vee \neg x_5 \vee x_6) \wedge \ldots$$

# Example

DETERMINISTIC-ASSIGNMENT($\Phi$)

for $i = 1, \ldots, m$

$x_i := 0$

compute $D := \mathbf{E}[X \mid \text{chosen values of } x_1, \ldots, x_i]$

if $D < 7n/8$

$x_i := 1$

return assignment

$\boxed{\text{Method of conditional probabilities}}$

$$\Phi = \overbrace{(\neg x_1 \vee \neg x_2 \vee x_4)}^{7/8} \wedge \overbrace{(x_1 \vee \neg x_4 \vee x_5)}^{7/8} \wedge \overbrace{(x_1 \vee \neg x_5 \vee x_6)}^{7/8} \wedge \ldots$$

# Example

DETERMINISTIC-ASSIGNMENT($\Phi$)
  for $i = 1, \ldots, m$
    $x_i := 0$
    compute $D := \mathbf{E}[X \mid \text{chosen values of } x_1, \ldots, x_i]$
    if $D < 7n/8$
      $x_i := 1$
  return assignment

Method of conditional probabilities

$$\Phi = \overbrace{(\neg x_1 \vee \neg x_2 \vee x_4)}^{7/8} \wedge \overbrace{(x_1 \vee \neg x_4 \vee x_5)}^{7/8} \wedge \overbrace{(x_1 \vee \neg x_5 \vee x_6)}^{7/8} \wedge \ldots$$

$x_1 := 0$:

$$\Phi = (1 \vee \neg x_2 \vee x_4) \wedge (0 \vee \neg x_4 \vee x_5) \wedge (0 \vee \neg x_5 \vee x_6) \wedge \ldots$$

# Example

DETERMINISTIC-ASSIGNMENT($\Phi$)

for $i = 1, \ldots, m$

$x_i := 0$

compute $D := \mathbf{E}[X \mid$ chosen values of $x_1, \ldots, x_i]$

if $D < 7n/8$

$x_i := 1$

return assignment

Method of conditional probabilities

$$\Phi = \overbrace{(\neg x_1 \vee \neg x_2 \vee x_4)}^{7/8} \wedge \overbrace{(x_1 \vee \neg x_4 \vee x_5)}^{7/8} \wedge \overbrace{(x_1 \vee \neg x_5 \vee x_6)}^{7/8} \wedge \ldots$$

$x_1 := 0$:

$$\Phi = \overbrace{(1 \vee \neg x_2 \vee x_4)}^{1} \wedge \overbrace{(0 \vee \neg x_4 \vee x_5)}^{3/4} \wedge \overbrace{(0 \vee \neg x_5 \vee x_6)}^{3/4} \wedge \ldots$$

# Example

DETERMINISTIC-ASSIGNMENT($\Phi$)
  for $i = 1, \ldots, m$
    $x_i := 0$
    compute $D := \mathbf{E}[X \mid \text{chosen values of } x_1, \ldots, x_i]$
    if $D < 7n/8$
      $x_i := 1$
  return assignment

Method of conditional probabilities

$$\Phi = \overbrace{(\neg x_1 \vee \neg x_2 \vee x_4)}^{7/8} \wedge \overbrace{(x_1 \vee \neg x_4 \vee x_5)}^{7/8} \wedge \overbrace{(x_1 \vee \neg x_5 \vee x_6)}^{7/8} \wedge \ldots$$

$x_1 := 0$:

$$\Phi = \overbrace{(1 \vee \neg x_2 \vee x_4)}^{1} \wedge \overbrace{(0 \vee \neg x_4 \vee x_5)}^{3/4} \wedge \overbrace{(0 \vee \neg x_5 \vee x_6)}^{3/4} \wedge \ldots$$

$x_1 := 1$

$$\Phi = (0 \vee \neg x_2 \vee x_4) \wedge (1 \vee \neg x_4 \vee x_5) \wedge (1 \vee \neg x_5 \vee x_6) \wedge \ldots$$

# Example

DETERMINISTIC-ASSIGNMENT($\Phi$)
  for $i = 1, \ldots, m$
    $x_i := 0$
    compute $D := \mathbf{E}\left[X \mid \text{chosen values of } x_1, \ldots, x_i\right]$
    if $D < 7n/8$
      $x_i := 1$
  return assignment

Method of conditional probabilities

$$\Phi = \overbrace{(\neg x_1 \vee \neg x_2 \vee x_4)}^{7/8} \wedge \overbrace{(x_1 \vee \neg x_4 \vee x_5)}^{7/8} \wedge \overbrace{(x_1 \vee \neg x_5 \vee x_6)}^{7/8} \wedge \ldots$$

$x_1 := 0$:

$$\Phi = \overbrace{(1 \vee \neg x_2 \vee x_4)}^{1} \wedge \overbrace{(0 \vee \neg x_4 \vee x_5)}^{3/4} \wedge \overbrace{(0 \vee \neg x_5 \vee x_6)}^{3/4} \wedge \ldots$$

$x_1 := 1$

$$\Phi = \overbrace{(0 \vee \neg x_2 \vee x_4)}^{3/4} \wedge \overbrace{(1 \vee \neg x_4 \vee x_5)}^{1} \wedge \overbrace{(1 \vee \neg x_5 \vee x_6)}^{1} \wedge \ldots$$

# Example

DETERMINISTIC-ASSIGNMENT($\Phi$)
  for $i = 1, \ldots, m$
    $x_i := 0$
    compute $D := \mathbf{E}\left[X \mid \text{chosen values of } x_1, \ldots, x_i\right]$
    if $D < 7n/8$
      $x_i := 1$
  return assignment

$\boxed{\text{Method of conditional probabilities}}$

$$\Phi = \overbrace{(\neg x_1 \vee \neg x_2 \vee x_4)}^{7/8} \wedge \overbrace{(x_1 \vee \neg x_4 \vee x_5)}^{7/8} \wedge \overbrace{(x_1 \vee \neg x_5 \vee x_6)}^{7/8} \wedge \ldots$$

$x_1 := 0$:

$$\Phi = \overbrace{(1 \vee \neg x_2 \vee x_4)}^{1} \wedge \overbrace{(0 \vee \neg x_4 \vee x_5)}^{3/4} \wedge \overbrace{(0 \vee \neg x_5 \vee x_6)}^{3/4} \wedge \ldots$$

$x_1 := 1$

$$\Phi = \overbrace{(0 \vee \neg x_2 \vee x_4)}^{3/4} \wedge \overbrace{(1 \vee \neg x_4 \vee x_5)}^{1} \wedge \overbrace{(1 \vee \neg x_5 \vee x_6)}^{1} \wedge \ldots$$

$x_2 := 0$

$$\Phi = \overbrace{(0 \vee 1 \vee x_4)}^{1} \wedge \overbrace{(1 \vee \neg x_4 \vee x_5)}^{1} \wedge \overbrace{(1 \vee \neg x_5 \vee x_6)}^{1} \wedge \ldots$$

# Bonus info

DETERMINISTIC-ASSIGNMENT($\Phi$)
for $i = 1, \ldots, m$
  $x_i := 0$
  compute $D := \mathbf{E}\left[X \mid \text{chosen values of } x_1, \ldots, x_i\right]$
  if $D < 7n/8$
    $x_i := 1$
return assignment

By work of Håstad, it is NP-hard to approximate within $8/7 - \varepsilon$ for all $\varepsilon > 0$, so this very simple algorithm is essentially optimal, unless P=NP.

# Vertex Cover

**Def.:** Let $G = (V, E)$ be a graph. A set $V' \subseteq V$ of vertices is a *vertex cover* if for all $uv \in E$, we have $u \in V'$ or $v \in V'$.

# Vertex Cover

**Def.:** Let $G = (V, E)$ be a graph. A set $V' \subseteq V$ of vertices is a *vertex cover* if for all $uv \in E$, we have $u \in V'$ or $v \in V'$.

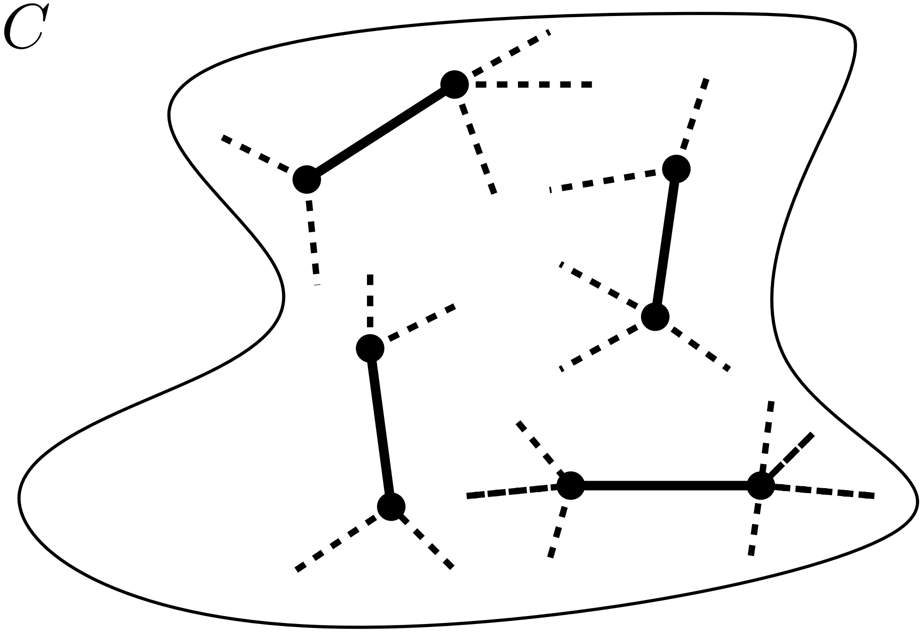
NP-hard!

# Vertex Cover

**Def.:** Let $G = (V, E)$ be a graph. A set $V' \subseteq V$ of vertices is a *vertex cover* if for all $uv \in E$, we have $u \in V'$ or $v \in V'$.

NP-hard!

APPROX-VERTEX-COVER($G$)
  $C := \emptyset$
  while $E(G) \neq \emptyset$
    choose $uv \in E(G)$
    $C := C \cup \{u, v\}$
    remove all edges incident on $u$ or $v$ from $E(G)$
  return $C$



$$\frac{C}{C^*} \leq 2$$

# Weighted Vertex Cover

**Def.:** Let $G = (V, E)$ be a graph. A set $V' \subseteq V$ of vertices is a *vertex cover* if for all $uv \in E$, we have $u \in V'$ or $v \in V'$.

# Weighted Vertex Cover

**Def.:** Let $G = (V, E)$ be a graph. A set $V' \subseteq V$ of vertices is a *vertex cover* if for all $uv \in E$, we have $u \in V'$ or $v \in V'$.

**Now:** We are given weight $w(v) > 0$ for each $v \in V$.

**Goal:** Find vertex cover $C$ with minimum

$$w(C) = \sum_{v \in C} w(v).$$

instead of minimizing the number of vertices we choose, we want to minimize the sum of the weights in the cover we chose.

# Weighted Vertex Cover

**Def.:** Let $G = (V, E)$ be a graph. A set $V' \subseteq V$ of vertices is a *vertex cover* if for all $uv \in E$, we have $u \in V'$ or $v \in V'$.

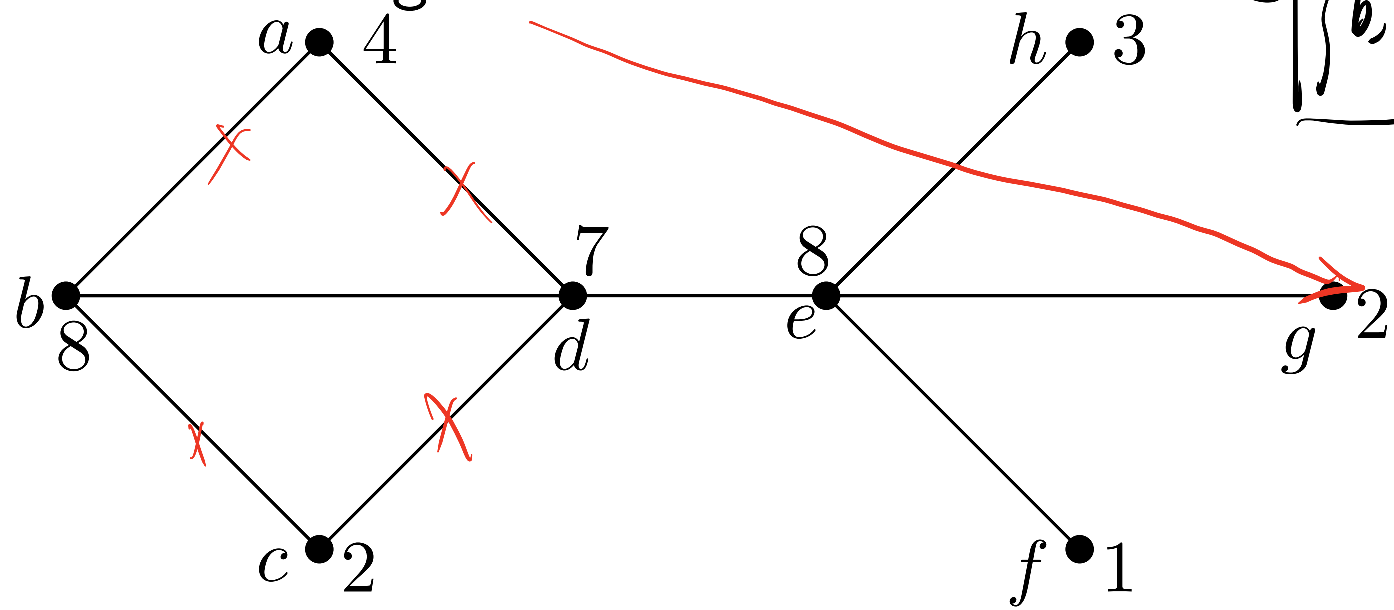
**Now:** We are given weight $w(v) > 0$ for each $v \in V$.

**Goal:** Find vertex cover $C$ with minimum

$$w(C) = \sum_{v \in C} w(v).$$

**Exercise:** Find minimum (unweighted) vertex cover and then minimum weighted vertex cover.

$\boxed{\{b, d, e\}}$




$\{a, c, d, h, f, g\}$

# Weighted Vertex Cover

**Def.:** Let $G = (V, E)$ be a graph. A set $V' \subseteq V$ of vertices is a *vertex cover* if for all $uv \in E$, we have $u \in V'$ or $v \in V'$.

**Now:** We are given weight $w(v) > 0$ for each $v \in V$.

**Goal:** Find vertex cover $C$ with minimum

$$w(C) = \sum_{v \in C} w(v).$$

0-1-integer program (IP):

$x_v \in \{0, 1\}$, $\forall v \in V$ $\qquad$ $(x_v = 1 \Leftrightarrow v \in C)$

$x_u + x_v \geq 1$, $\forall uv \in E$ $\qquad$ (edge $uv$ covered)

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

# LP-relaxation

0-1-integer program (IP):

$x_v \in \{0,1\}$, $\forall v \in V$ $\qquad (x_v = 1 \Leftrightarrow v \in C)$

$x_u + x_v \geq 1$, $\forall uv \in E$ $\qquad$ (edge $uv$ covered)

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

IP is still NP-complete!

but we can

# LP-relaxation

$0$-$1$-integer program (IP):

$x_v \in \{0,1\}$, $\forall v \in V$ $\qquad (x_v = 1 \Leftrightarrow v \in C)$

$x_u + x_v \geq 1$, $\forall uv \in E$ $\qquad$ (edge $uv$ covered)

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

IP is NP-complete!

**LP-relaxation:** replace $x_v \in \{0,1\}$ with $0 \leq x_v \leq 1$. Result:

# LP-relaxation

0-1-integer program (IP):

$x_v \in \{0,1\}$, $\forall v \in V$ $\qquad (x_v = 1 \Leftrightarrow v \in C)$

$x_u + x_v \geq 1$, $\forall uv \in E$ $\qquad$ (edge $uv$ covered)

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

IP is NP-complete!

**LP-relaxation:** replace $x_v \in \{0,1\}$ with $0 \leq x_v \leq 1$. Result:

$0 \leq x_v \leq 1$, $\forall v \in V$

$x_u + x_v \geq 1$, $\forall uv \in E$

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

# LP-relaxation

0-1-integer program (IP):

$x_v \in \{0,1\}$, $\forall v \in V$ $\quad\quad (x_v = 1 \Leftrightarrow v \in C)$

$x_u + x_v \geq 1$, $\forall uv \in E$ $\quad\quad$ (edge $uv$ covered)

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

IP is NP-complete!

**LP-relaxation:** replace $x_v \in \{0,1\}$ with $0 \leq x_v \leq 1$. Result:

$0 \leq x_v \leq 1$, $\forall v \in V$

$x_u + x_v \geq 1$, $\forall uv \in E$

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

Relaxed solution can be smaller, not larger

# Algorithm

**LP:**

$$0 \le x_v \le 1,\ \forall v \in V$$

$$x_u + x_v \ge 1,\ \forall uv \in E$$

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

APPROX-MIN-WEIGHT-VC$(G, W)$:

Compute opt. sol. $\bar{x}$ to LP

return $C := \{v \in V \mid \bar{x}_v \ge 1/2\}$

# Algorithm

**LP:**

$$0 \le x_v \le 1, \ \forall v \in V$$

$$x_u + x_v \ge 1, \ \forall uv \in E$$

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

APPROX-MIN-WEIGHT-VC$(G, W)$:

Compute opt. sol. $\bar{x}$ to LP

return $C := \{v \in V \mid \bar{x}_v \ge 1/2\}$

**Theorem:** Alg. is a polynomial-time $2$-approximation algorithm for minimum-weight vertex cover.

# Algorithm

**LP:**

$$0 \leq x_v \leq 1, \forall v \in V$$
$$x_u + x_v \geq 1, \forall uv \in E$$

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

APPROX-MIN-WEIGHT-VC($G, W$):
  Compute opt. sol. $\bar{x}$ to LP
  return $C := \{v \in V \mid \bar{x}_v \geq 1/2\}$

**Theorem:** Alg. is a polynomial-time $2$-approximation algorithm for minimum-weight vertex cover.

**Need to prove:** 1) Polynomial time. 2) Alg. produces feasible solution (i.e., $C$ is a vertex cover). 3) $\frac{w(C)}{w(C^*)} \leq 2$.

# Algorithm

**LP:**

$$0 \le x_v \le 1, \ \forall v \in V$$
$$x_u + x_v \ge 1, \ \forall uv \in E$$

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

APPROX-MIN-WEIGHT-VC$(G, W)$:

Compute opt. sol. $\bar{x}$ to LP

return $C := \{v \in V \mid \bar{x}_v \ge 1/2\}$

**Theorem:** Alg. is a polynomial-time 2-approximation algorithm for minimum-weight vertex cover.

**Need to prove:** 1) Polynomial time. 2) Alg. produces feasible solution (i.e., $C$ is a vertex cover). 3) $\frac{w(C)}{w(C^*)} \le 2$.

1) ✓ → 因为这一步LP最多 polynomial time，所以整个就 poly

# Algorithm

**LP:**

$$0 \leq x_v \leq 1, \ \forall v \in V$$
$$x_u + x_v \geq 1, \ \forall uv \in E$$

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

APPROX-MIN-WEIGHT-VC$(G, W)$:
Compute opt. sol. $\bar{x}$ to LP
return $C := \{v \in V \mid \bar{x}_v \geq 1/2\}$

**Theorem:** Alg. is a polynomial-time $2$-approximation algorithm for minimum-weight vertex cover.

**Need to prove:** 1) Polynomial time. 2) Alg. produces feasible solution (i.e., $C$ is a vertex cover). 3) $\frac{w(C)}{w(C^*)} \leq 2$.

1) ✓

2) $uv \in E \Rightarrow \bar{x}_u + \bar{x}_v \geq 1 \Rightarrow \bar{x}_u \geq \frac{1}{2} \vee \bar{x}_v \geq \frac{1}{2} \Rightarrow u \in C \vee v \in C$. ✓

The sum of the two in our solution is at least 1

# Algorithm

**LP:**

$$0 \leq x_v \leq 1, \forall v \in V$$
$$x_u + x_v \geq 1, \forall uv \in E$$

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

APPROX-MIN-WEIGHT-VC$(G, W)$:

Compute opt. sol. $\bar{x}$ to LP

return $C := \{v \in V \mid \bar{x}_v \geq 1/2\}$

**Need to prove:** 3) $\frac{w(C)}{w(C^*)} \leq 2$.

# Algorithm

**LP:**

$0 \leq x_v \leq 1, \ \forall v \in V$

$x_u + x_v \geq 1, \ \forall uv \in E$

$\bar{x}$ from LP

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

APPROX-MIN-WEIGHT-VC$(G, W)$:

Compute opt. sol. $\bar{x}$ to LP

return $C := \{v \in V \mid \bar{x}_v \geq 1/2\}$

**Need to prove:** 3) $\frac{w(C)}{w(C^*)} \leq 2.$

Let $z^* := \sum_{v \in V} \bar{x}_v w(v)$, recall $z^* \leq w(C^*)$.

cost of our optimal solution

因为 linear program is a relaxation of the integer program, then it at most as big as the cost of the optimal solution.

# Algorithm

**LP:**

$$0 \le x_v \le 1, \ \forall v \in V$$
$$x_u + x_v \ge 1, \ \forall uv \in E$$

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

APPROX-MIN-WEIGHT-VC$(G, W)$:

Compute opt. sol. $\bar{x}$ to LP

return $C := \{v \in V \mid \bar{x}_v \ge 1/2\}$

**Need to prove:** 3) $\frac{w(C)}{w(C^*)} \le 2$.

Let $z^* := \sum_{v \in V} \bar{x}_v w(v)$, recall $z^* \le w(C^*)$.

Evaluate the cost of our produced solution.

$$w(C) = \sum_{v \in C} w(v) = \sum_{v \in V} w(v)[v \in C] = \sum_{v \in V} w(v)[\bar{x}_v \ge \tfrac{1}{2}]$$

# Algorithm

**LP:**

$$0 \le x_v \le 1,\ \forall v \in V$$
$$x_u + x_v \ge 1,\ \forall uv \in E$$

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

APPROX-MIN-WEIGHT-VC$(G, W)$:

Compute opt. sol. $\bar{x}$ to LP

return $C := \{v \in V \mid \bar{x}_v \ge 1/2\}$

**Need to prove:** 3) $\frac{w(C)}{w(C^*)} \le 2.$

Let $z^* := \sum_{v \in V} \bar{x}_v w(v)$, recall $z^* \le w(C^*)$.

$$w(C) = \sum_{v \in C} w(v) = \sum_{v \in V} w(v)[v \in C] = \sum_{v \in V} w(v) \overset{\le 2\bar{x}_v}{\boxed{[\bar{x}_v \ge \tfrac{1}{2}]}}$$

1 when $v \in C$

$[\bar{x}_v \ge \frac{1}{2}] = 0 \Rightarrow [\bar{x}_v \ge \frac{1}{2}] = 0 \le 2\bar{x}_v$

$[\bar{x}_v \ge \frac{1}{2}] = 1 \Rightarrow \bar{x}_v \ge \frac{1}{2} \Rightarrow 2\bar{x}_v \ge 1$

# Algorithm

**LP:**

$$0 \le x_v \le 1,\ \forall v \in V$$
$$x_u + x_v \ge 1,\ \forall uv \in E$$

$$\text{minimize} \sum_{v \in V} w(v) x_v$$

APPROX-MIN-WEIGHT-VC$(G, W)$:

Compute opt. sol. $\bar{x}$ to LP

return $C := \{v \in V \mid \bar{x}_v \ge 1/2\}$

**Need to prove:** 3) $\frac{w(C)}{w(C^*)} \le 2.$

Let $z^* := \sum_{v \in V} \bar{x}_v w(v)$, recall $z^* \le w(C^*)$.

$$w(C) = \sum_{v \in C} w(v) = \sum_{v \in V} w(v)[v \in C] = \sum_{v \in V} w(v) \underbrace{\boxed{[\bar{x}_v \ge \tfrac{1}{2}]}}_{\le 2\bar{x}_v}$$

$$\le 2 \sum_{v \in V} w(v) \bar{x}_v = 2z^* \le 2w(C^*) \Rightarrow \frac{w(C)}{w(C^*)} \le 2. \quad \checkmark$$

# Reflection and methodology

How can we prove $w(C)/w(C^*) \leq 2$ when we don't know $w(C^*)$?

Answer: By proving $w(C) \leq 2z^*$ and $|z^*| \leq w(C^*)$.

# Reflection and methodology

How can we prove $w(C)/w(C^*) \leq 2$ when we don't know $w(C^*)$?

Answer: By proving $w(C) \leq 2z^*$ and $|z^*| \leq w(C^*)$.

General technique: Find a parameter $\square$ such that $C \leq \rho \cdot \square$ and $\square \leq C^*$.

For weighted vertex cover: $\square = z^*$ and $\rho = 2$.

# Approximation schemes

**Polynomial-time approximation scheme (PTAS):**
Approximation algorithm that takes instance $I$ of an optimization problem $P$ and $\varepsilon > 0$ as input. For any fixed $\varepsilon$ works as $(1+\varepsilon)$-approximation algorithm for $P$.

# 近似方案 Approximation schemes

a whole family of algorithms one for each $\varepsilon$.

A higher order algorithm.

**Polynomial-time approximation scheme (PTAS):**
Approximation algorithm that takes instance $I$ of an optimization problem $P$ and $\varepsilon > 0$ as input. For any fixed $\varepsilon$ works as $(1+\varepsilon)$-approximation algorithm for $P$.

**Ex:** Runtime $O(2^{1/\varepsilon} \cdot n^3)$ or $O(n^{1/\varepsilon})$ or $O(n \log n/\varepsilon^2)$.

Example

# Approximation schemes

**Polynomial-time approximation scheme (PTAS):**
Approximation algorithm that takes instance $I$ of an optimization problem $P$ and $\varepsilon > 0$ as input. For any fixed $\varepsilon$ works as $(1+\varepsilon)$-approximation algorithm for $P$.
**Ex:** Runtime $O(2^{1/\varepsilon} \cdot n^3)$ or $O(n^{1/\varepsilon})$ or $O(n \log n / \varepsilon^2)$.

因为这个是 exponential in $\frac{1}{\varepsilon}$

$n^{\frac{1}{\varepsilon}}$

**Fully polynomial-time approximation scheme (FPTAS):**
PTAS with runtime polynomial in $1/\varepsilon$ and the size of $I$.

# Approximation schemes

**Polynomial-time approximation scheme (PTAS):**
Approximation algorithm that takes instance $I$ of an optimization problem $P$ and $\varepsilon > 0$ as input. For any fixed $\varepsilon$ works as $(1+\varepsilon)$-approximation algorithm for $P$.
**Ex:** Runtime $O(2^{1/\varepsilon} \cdot n^3)$ or $O(n^{1/\varepsilon})$ or $O(n \log n/\varepsilon^2)$.

**Fully polynomial-time approximation scheme (FPTAS):**
PTAS with runtime polynomial in $1/\varepsilon$ and the size of $I$.
**Ex:** Runtime $O(n \log n/\varepsilon^2)$.

# SUBSET-SUM EFPTAS

**Input:** Set $S = \{x_1, \ldots, x_n\} \subset \mathbb{N}$, and $t \in \mathbb{N}$.
**Goal:** Find $U \subset S$ s.t. $\sum_{x \in U} x \leq t$ with maximum $\sum_{x \in U} x$.

# SUBSET-SUM

**Input:** Set $S = \{x_1, \ldots, x_n\} \subset \mathbb{N}$, and $t \in \mathbb{N}$.
**Goal:** Find $U \subset S$ s.t. $\sum_{x \in U} x \leq t$ with maximum $\sum_{x \in U} x$.
**Example:** $S = \{1, 4, 5\}$, $t = 8$.

# SUBSET-SUM

**Input:** Set $S = \{x_1, \ldots, x_n\} \subset \mathbb{N}$, and $t \in \mathbb{N}$.
**Goal:** Find $U \subset S$ s.t. $\sum_{x \in U} x \leq t$ with maximum $\sum_{x \in U} x$.
**Example:** $S = \{1, 4, 5\}$, $t = 8$.

NP-complete to decide if $\exists U \subset S : \sum_{x \in U} = t$.

尽可能大但不超过t.

# SUBSET-SUM

**Input:** Set $S = \{x_1, \ldots, x_n\} \subset \mathbb{N}$, and $t \in \mathbb{N}$.
**Goal:** Find $U \subset S$ s.t. $\sum_{x \in U} x \leq t$ with maximum $\sum_{x \in U} x$.
**Example:** $S = \{1, 4, 5\}$, $t = 8$. $\to$ 6
NP-complete to decide if $\exists U \subset S : \sum_{x \in U} = t$.

**Abstract exact alg.:**

for $k = 1, 2, \ldots, n$
  compute $L_k := \{\sum_{x \in U} x \mid U \subset \{x_1, \ldots, x_k\} \boxed{\wedge} \sum_{x \in U} x \leq t\}$.
return $\max L_n$

# SUBSET-SUM

**Input:** Set $S = \{x_1, \ldots, x_n\} \subset \mathbb{N}$, and $t \in \mathbb{N}$.
**Goal:** Find $U \subset S$ s.t. $\sum_{x \in U} x \leq t$ with maximum $\sum_{x \in U} x$.
**Example:** $S = \{1, 4, 5\}$, $t = 8$.

NP-complete to decide if $\exists U \subset S : \sum_{x \in U} = t$.

**Abstract exact alg.:**

for $k = 1, 2, \ldots, n$
  compute $L_k := \left\{\sum_{x \in U} x \mid U \subset \{x_1, \ldots, x_k\} \wedge \sum_{x \in U} x \leq t\right\}$.
return $\max L_n$

**Note:** $L_k \subset L_{k-1} \cup (L_{k-1} + x_k)$.

# SUBSET-SUM

**Input:** Set $S = \{x_1, \ldots, x_n\} \subset \mathbb{N}$, and $t \in \mathbb{N}$.
**Goal:** Find $U \subset S$ s.t. $\sum_{x \in U} x \leq t$ with maximum $\sum_{x \in U} x$.
**Example:** $S = \{1, 4, 5\}$, $t = 8$.

1+4
1+5

NP-complete to decide if $\exists U \subset S : \sum_{x \in U} = t$.

**Abstract exact alg.:**

for $k = 1, 2, \ldots, n$
  compute $L_k := \{\sum_{x \in U} x \mid U \subset \{x_1, \ldots, x_k\} \wedge \sum_{x \in U} x \leq t\}$.
  return $\max L_n$

**Note:** $L_k \subset L_{k-1} \cup (L_{k-1} + x_k)$.

$L_k$ 要么 $= L_{k-1}$ (当 $+x_k > t$), 要么 $L_k$ $L_{k-1}+x_k$ 当 $x_k \leq t$,

$L_k = \{k$个数值的和的集合$\}$

EXACT-SUBSET-SUM$(S, t)$
  $L_0 = [0]$
  for $k = 1, \ldots, n$
    $L_k$=MERGE-LISTS$(L_{k-1}, L_{k-1} + x_k)$
    remove from $L_k$ duplicates and elements $> t$
  return last$(L_n)$

# SUBSET-SUM

EXACT-SUBSET-SUM$(S, t)$
$L_0 = [0]$
for $k = 1, \ldots, n$
$L_k$=MERGE-LISTS$(L_{k-1}, L_{k-1} + x_k)$
remove duplicates and elm.s $> t$
return last$(L_n)$

**Example:** $S = \{1, 4, 5\}$, $t = 8$.

$L_k$ will contain all the subsets we can make with the numbers $x_1, \ldots, x_k$ that are not too large

# SUBSET-SUMEXACT-SUBSET-SUM($S, t$)

$L_0 = [0]$
for $k = 1, \ldots, n$
$L_k$=MERGE-LISTS($L_{k-1}, L_{k-1} + x_k$)
remove duplicates and elm.s $> t$
return last($L_n$)

**Example:** $S = \{1, 4, 5\}$, $t = 8$.

$L_0 = [0]$

# SUBSET-SUM

EXACT-SUBSET-SUM($S, t$)
  $L_0 = [0]$
  for $k = 1, \ldots, n$
    $L_k$=MERGE-LISTS($L_{k-1}, L_{k-1} + x_k$)
    remove duplicates and elm.s $> t$
  return last($L_n$)

**Example:** $S = \{1, 4, 5\}$, $t = 8$.

$x_k$ 从小到大加的

$L_0 = [0]$

$L_1 = L_0 \cup (L_0 + 1) = [0] \cup [1] = [0, 1]$

# SUBSET-SUM

EXACT-SUBSET-SUM$(S, t)$
  $L_0 = [0]$
  for $k = 1, \ldots, n$
    $L_k$=MERGE-LISTS$(L_{k-1}, L_{k-1} + x_k)$
    remove duplicates and elm.s $> t$
  return last$(L_n)$

**Example:** $S = \{1, 4, 5\}$, $t = 8$.

$L_0 = [0]$

$L_1 = L_0 \cup (L_0 + 1) = [0] \cup [1] = [0, 1]$

$L_2 = \underline{L_1} \cup (L_1 + 4) = [\underline{0, 1}] \cup [4, 5] = [0, 1, 4, 5]$

# SUBSET-SUM

EXACT-SUBSET-SUM($S, t$)
    $L_0 = [0]$
    for $k = 1, \ldots, n$
        $L_k$=MERGE-LISTS($L_{k-1}, L_{k-1} + x_k$)
        remove duplicates and elm.s $> t$
    return last($L_n$)

**Example:** $S = \{1, 4, 5\}$, $t = 8$.

$L_0 = [0]$

$L_1 = L_0 \cup (L_0 + 1) = [0] \cup [1] = [0, 1]$

$L_2 = L_1 \cup (L_1 + 4) = [0, 1] \cup [4, 5] = [0, 1, 4, 5]$

$L_3 = L_2 \cup (L_2 + 5) = [0, 1, 4, 5] \cup [\cancel{5}, 6, \cancel{9}, \cancel{10}] = [0, 1, 4, 5, \boxed{6}]$

# SUBSET-SUM

EXACT-SUBSET-SUM$(S, t)$
  $L_0 = [0]$
  for $k = 1, \ldots, n$
    $L_k$=MERGE-LISTS$(L_{k-1}, L_{k-1} + x_k)$
    remove duplicates and elm.s $> t$
  return last$(L_n)$

**Example:** $S = \{1, 4, 5\}$, $t = 8$.

$L_0 = [0]$

$L_1 = L_0 \cup (L_0 + 1) = [0] \cup [1] = [0, 1]$

$L_2 = L_1 \cup (L_1 + 4) = [0, 1] \cup [4, 5] = [0, 1, 4, 5]$

$L_3 = L_2 \cup (L_2 + 5) = [0, 1, 4, 5] \cup [\cancel{5}, 6, \cancel{9}, \cancel{10}] = [0, 1, 4, 5, \boxed{6}]$

**Running time:** Computing $L_k$: $O(|\underline{L_{k-1}}|)$.

Total: $O\left(\sum_{k=1}^{n} |L_k|\right)$

+ Remove

$L_k \cup L_{k-1} \cup (L_{k-1} + x_k)$

所以 $L_k$ 要进行与 $L_{k-1}$ 次

# SUBSET-SUM

```
EXACT-SUBSET-SUM(S, t)
    L_0 = [0]
    for k = 1, ..., n
        L_k = MERGE-LISTS(L_{k-1}, L_{k-1} + x_k)
        remove duplicates and elm.s > t
    return last(L_n)
```

**Example:** $S = \{1, 4, 5\}$, $t = 8$.

$L_0 = [0]$

$L_1 = L_0 \cup (L_0 + 1) = [0] \cup [1] = [0, 1]$

$L_2 = L_1 \cup (L_1 + 4) = [0, 1] \cup [4, 5] = [0, 1, 4, 5]$

$L_3 = L_2 \cup (L_2 + 5) = [0, 1, 4, 5] \cup [\cancel{5}, 6, \cancel{9}, \cancel{10}] = [0, 1, 4, 5, \boxed{6}]$

**Running time:** Computing $L_k$: $O(|L_{k-1}|)$.

Total: $O\left(\sum_{k=1}^{n} |L_k|\right) = O(nt)$

Because we have n of these, and each of them can contain at most t numbers because ... away everything that is more than t.

we take away

# SUBSET-SUM

EXACT-SUBSET-SUM($S, t$)
  $L_0 = [0]$
  for $k = 1, \ldots, n$
    $L_k$=MERGE-LISTS($L_{k-1}, L_{k-1} + x_k$)
    remove duplicates and elm.s $> t$
  return last($L_n$)

**Example:** $S = \{1, 4, 5\}$, $t = 8$.

$L_0 = [0]$

$L_1 = L_0 \cup (L_0 + 1) = [0] \cup [1] = [0, 1]$

$L_2 = L_1 \cup (L_1 + 4) = [0, 1] \cup [4, 5] = [0, 1, 4, 5]$

$L_3 = L_2 \cup (L_2 + 5) = [0, 1, 4, 5] \cup [\cancel{5}, 6, \cancel{9}, \cancel{10}] = [0, 1, 4, 5, \boxed{6}]$

**Running time:** Computing $L_k$: $O(|L_{k-1}|)$.

Total: $O\left(\sum_{k=1}^{n} |L_k|\right) = O(nt) = O(n2^{\log t})$ EXPONENTIAL !!

Represent t in binary

exponential function in the size of t.

! how we represent numbers in the input.

# Trimming

| EXACT-SUBSET-SUM$(S, t)$ | **Idea:** Trim list |
|---|---|
| $L_0 = [0]$<br>for $k = 1, \ldots, n$<br>$L_k$=MERGE-LISTS$(L_{k-1}, L_{k-1} + x_k)$<br>remove duplicates and elm.s $> t$<br>return last$(L_n)$ | $L \subset \{0, 1, \ldots, t\}$ with parameter $\delta > 0$: if we keep $s \in L$, then remove $(s, (1 + \delta)s]$. |

instead of computing all these lists exactly, then we use some parameter $\delta$ to trim them so that we don't get numbers that are too close to each other in the lists.

then the list will be short and then it will run in polynomial time.

# Trimming

EXACT-SUBSET-SUM($S, t$)
  $L_0 = [0]$
  for $k = 1, \ldots, n$
    $L_k$=MERGE-LISTS($L_{k-1}, L_{k-1} + x_k$)
    remove duplicates and elm.s $> t$
  return last($L_n$)

**Idea:** Trim list $L \subset \{0, 1, \ldots, t\}$ with parameter $\delta > 0$: if we keep $s \in L$, then remove $(s, (1+\delta)s]$.

**Example:** $L = [0, 9, 10, 11, 12, 13, 16]$, $\delta = 0.1$.

# Trimming

| EXACT-SUBSET-SUM$(S,t)$ | **Idea:** Trim list |
|---|---|
| $L_0 = [0]$ | $L \subset \{0,1,\ldots,t\}$ with |
| for $k = 1,\ldots,n$ | parameter $\delta > 0$: if we keep |
| $L_k$=MERGE-LISTS$(L_{k-1}, L_{k-1} + x_k)$ | $s \in L$, then remove |
| remove duplicates and elm.s $> t$ | $(s,(1+\delta)s]$. |
| return last$(L_n)$ | |

**Example:** $L = [0, 9, 10, \cancel{11}, 12, 13, 16]$, $\delta = 0.1$.

[9, 9.9]

# Trimming

EXACT-SUBSET-SUM($S, t$)
  $L_0 = [0]$
  for $k = 1, \ldots, n$
    $L_k$=MERGE-LISTS($L_{k-1}, L_{k-1} + x_k$)
    remove duplicates and elm.s $> t$
  return last($L_n$)

**Idea:** Trim list $L \subset \{0, 1, \ldots, t\}$ with parameter $\delta > 0$: if we keep $s \in L$, then remove $(s, (1 + \delta)s]$.

**Example:** $L = [0, 9, 10, \cancel{11}, 12, \cancel{13}, 16]$, $\delta = 0.1$.

[10, 11]

[12, 13.2]

# Trimming

EXACT-SUBSET-SUM$(S, t)$
  $L_0 = [0]$
  for $k = 1, \dots, n$
    $L_k$=MERGE-LISTS$(L_{k-1}, L_{k-1} + x_k)$
    remove duplicates and elm.s $> t$
  return last$(L_n)$

**Idea:** Trim list $L \subset \{0, 1, \dots, t\}$ with parameter $\delta > 0$: if we keep $s \in L$, then remove $(s, (1+\delta)s]$.

**Example:** $L = [0, 9, 10, \cancel{11}, 12, \cancel{13}, 16]$, $\delta = 0.1$.

TRIM$(L = [\overbrace{s_1, \dots, s_m}^{\text{input}}], \delta)$
  $L' = [s_1]$
  for $i = 2, \dots, m$
    if $s_i >$ last$(L') \cdot (1 + \delta)$
      $L' = L' \cup [s_i]$
  return $L'$

# Trimming

EXACT-SUBSET-SUM($S, t$)
  $L_0 = [0]$
  for $k = 1, \ldots, n$
    $L_k$=MERGE-LISTS($L_{k-1}, L_{k-1} + x_k$)
    remove duplicates and elm.s $> t$
  return last($L_n$)

**Idea:** Trim list $L \subset \{0, 1, \ldots, t\}$ with parameter $\delta > 0$: if we keep $s \in L$, then remove $(s, (1+\delta)s]$.

**Example:** $L = [0, 9, 10, \cancel{11}, 12, \cancel{13}, 16]$, $\delta = 0.1$.

TRIM($L = [s_1, \ldots, s_m], \delta$)
  $L' = [s_1]$
  for $i = 2, \ldots, m$
    if $s_i >$ last$(L') \cdot (1+\delta)$
      $L' = L' \cup [s_i]$
  return $L'$

APPROX-SUBSET-SUM($S, t, \varepsilon$)
  $L'_0 = [0]$
  for $k = 1, \ldots, n$                Approximation scheme
    $L'_k$=MERGE-LISTS($L'_{k-1}, L'_{k-1} + x_k$)
    $L'_k$=TRIM($L'_k, \varepsilon/2n$)
    remove duplicates and elm.s $> t$
  return last($L'_n$)

# Theorem

**Thm.:** The alg. is an FPTAS.

APPROX-SUBSET-SUM($S, t, \varepsilon$)
  $L'_0 = [0]$
  for $k = 1, \ldots, n$
    $L'_k$=MERGE-LISTS($L'_{k-1}, L'_{k-1} + x_k$)
    $L'_k$=TRIM($L'_k, \varepsilon/2n$)
    remove duplicates and elm.s $> t$
  return last($L'_n$)

# Theorem

**Thm.:** The alg. is an FPTAS.

*Proof:* Feasibility: Opt. is $s_{\max} = \text{last}(L_n) \leq t$ and $\text{last}(L'_n) \leq \text{last}(L_n)$.

未修剪的

it produces subsets which don't exceeds t when you add them together.

APPROX-SUBSET-SUM$(S, t, \varepsilon)$
- $L'_0 = [0]$
- for $k = 1, \ldots, n$
  - $L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$
  - $L'_k$=TRIM$(L'_k, \varepsilon/2n)$
  - remove duplicates and elm.s $> t$
- return last$(L'_n)$

# Theorem

**Thm.:** The alg. is an FPTAS.

*Proof:* Feasibility: Opt. is $s_{\max} = \text{last}(L_n) \le t$ and $\text{last}(L'_n) \le \text{last}(L_n)$.
Approx. ratio: Assume we trim with $\boxed{\delta}$.

APPROX-SUBSET-SUM$(S, t, \varepsilon)$
  $L'_0 = [0]$
  for $k = 1, \ldots, n$
    $L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$
    $L'_k$=TRIM$(L'_k, \boxed{\varepsilon/2n})$
    remove duplicates and elm.s $> t$
  return last$(L'_n)$

$(1+\varepsilon)$ Approximation

# Theorem

**Thm.:** The alg. is an FPTAS.

*Proof:* Feasibility: Opt. is $s_{\max} = \text{last}(L_n) \le t$ and $\text{last}(L'_n) \le \text{last}(L_n)$.
Approx. ratio: Assume we trim with $\boxed{\delta}$.

APPROX-SUBSET-SUM$(S, t, \varepsilon)$
- $L'_0 = [0]$
- for $k = 1, \ldots, n$
  - $L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$
  - $L'_k$=TRIM$(L'_k, \boxed{\varepsilon/2n})$
  - remove duplicates and elm.s $> t$
- return last$(L'_n)$

From exercise: $\forall s \in L_k \exists s' \in L'_k : s' \le s \le (1+\delta)^k s'$.

# Theorem

**Thm.:** The alg. is an FPTAS.

APPROX-SUBSET-SUM$(S, t, \varepsilon)$
  $L'_0 = [0]$
  for $k = 1, \ldots, n$
    $L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$
    $L'_k$=TRIM$(L'_k, \boxed{\varepsilon/2n})$
    remove duplicates and elm.s $> t$
  return last$(L'_n)$

*Proof:* Feasibility: Opt. is $s_{\max} = \text{last}(L_n) \leq t$ and last$(L'_n) \leq$ last$(L_n)$.
Approx. ratio: Assume we trim with $\boxed{\delta}$.

From exercise: $\forall s \in L_k \exists s' \in L'_k : s' \leq \boxed{s \leq (1+\delta)^k s'}$.

$\Rightarrow \frac{s}{s'} \leq (1+\delta)^k$

$\Downarrow$

We have numbers in our trimed list which are not too much difference from the untrimed list.

# Theorem

**Thm.:** The alg. is an FPTAS.

APPROX-SUBSET-SUM$(S, t, \varepsilon)$

$L'_0 = [0]$

for $k = 1, \ldots, n$

$L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$

$L'_k$=TRIM$(L'_k, \boxed{\varepsilon/2n})$

remove duplicates and elm.s $> t$

return last$(L'_n)$

*Proof:* Feasibility: Opt. is $s_{\max} =$ last$(L_n) \leq t$ and last$(L'_n) \leq$ last$(L_n)$.

Approx. ratio: Assume we trim with $\boxed{\delta}$.

From exercise: $\forall s \in L_k \exists s' \in L'_k : s' \leq \boxed{s \leq (1+\delta)^k s'}$.

$$\Rightarrow \frac{s}{s'} \leq (1+\delta)^k$$

From exercise, there is $s' \in L'_n$ such that $\frac{s_{\max}}{s'} \leq (1+\delta)^n$.

# Theorem

**Thm.:** The alg. is an FPTAS.

APPROX-SUBSET-SUM$(S, t, \varepsilon)$
  $L'_0 = [0]$
  for $k = 1, \ldots, n$
    $L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$
    $L'_k$=TRIM$(L'_k, \boxed{\varepsilon/2n})$
    remove duplicates and elm.s $> t$
  return last$(L'_n)$

*Proof:* Feasibility: Opt. is $s_{\max} = \text{last}(L_n) \leq t$ and last$(L'_n) \leq$ last$(L_n)$.
Approx. ratio: Assume we trim with $\boxed{\delta}$.

From exercise: $\forall s \in L_k \exists s' \in L'_k : s' \leq \boxed{s \leq (1+\delta)^k s'}$.

$$\Rightarrow \frac{s}{s'} \leq (1+\delta)^k$$

From exercise, there is $s' \in L'_n$ such that $\frac{s_{\max}}{s'} \leq (1+\delta)^n$.

$$\text{Approximation ratio: } \frac{s_{\max}}{\text{last}(L'_n)} \leq \boxed{\frac{s_{\max}}{s'}} \leq (1+\delta)^n.$$

最大的

# Theorem

**Thm.:** The alg. is an FPTAS.

APPROX-SUBSET-SUM$(S, t, \varepsilon)$
$L'_0 = [0]$
for $k = 1, \ldots, n$
$L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$
$L'_k$=TRIM$(L'_k, \boxed{\varepsilon/2n})$
remove duplicates and elm.s $> t$
return last$(L'_n)$

*Proof:* Feasibility: Opt. is $s_{\max} = \text{last}(L_n) \leq t$ and last$(L'_n) \leq$ last$(L_n)$.
Approx. ratio: Assume we trim with $\boxed{\delta}$.

From exercise: $\forall s \in L_k \exists s' \in L'_k : s' \leq \boxed{s \leq (1+\delta)^k s'}$.

$\Rightarrow \frac{s}{s'} \leq (1+\delta)^k$

want $\leq 1 + \varepsilon$ !

From exercise, there is $s' \in L'_n$ such that $\frac{s_{\max}}{s'} \leq (1+\delta)^n$.

$$\text{Approximation ratio: } \frac{s_{\max}}{\text{last}(L'_n)} \leq \frac{s_{\max}}{s'} \leq \boxed{(1+\delta)^n}.$$

# Theorem

**Thm.:** The alg. is an FPTAS.

*Proof:* Feasibility: Opt. is $s_{\max} = \text{last}(L_n) \leq t$ and $\text{last}(L'_n) \leq \text{last}(L_n)$.
Approx. ratio: Assume we trim with $\boxed{\delta}$.

APPROX-SUBSET-SUM$(S, t, \varepsilon)$
- $L'_0 = [0]$
- for $k = 1, \ldots, n$
  - $L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$
  - $L'_k$=TRIM$(L'_k, \boxed{\varepsilon/2n})$
  - remove duplicates and elm.s $> t$
- return last$(L'_n)$

From exercise: $\forall s \in L_k \exists s' \in L'_k : s' \leq \boxed{s \leq (1+\delta)^k s'}$.

$\Rightarrow \frac{s}{s'} \leq (1+\delta)^k$

want $\leq 1 + \varepsilon$ !

From exercise, there is $s' \in L'_n$ such that $\frac{s_{\max}}{s'} \leq (1+\delta)^n$.

Approximation ratio: $\dfrac{s_{\max}}{\text{last}(L'_n)} \leq \dfrac{s_{\max}}{s'} \leq \boxed{(1+\delta)^n}$.

**Claim:** $(1+\delta)^n \leq 1 + 2n\delta$ if $2n\delta \leq 1$.

# Theorem

**Thm.:** The alg. is an FPTAS.

APPROX-SUBSET-SUM$(S, t, \varepsilon)$
$L'_0 = [0]$
for $k = 1, \ldots, n$
$L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$
$L'_k$=TRIM$(L'_k, \boxed{\varepsilon/2n})$
remove duplicates and elm.s $> t$
return last$(L'_n)$

*Proof:* Feasibility: Opt. is $s_{\max} = \text{last}(L_n) \leq t$ and last$(L'_n) \leq$ last$(L_n)$.
Approx. ratio: Assume we trim with $\boxed{\delta}$.

From exercise: $\forall s \in L_k \exists s' \in L'_k : s' \leq \boxed{s \leq (1+\delta)^k s'}$.

$$\Rightarrow \frac{s}{s'} \leq (1+\delta)^k$$

want $\leq 1 + \varepsilon$ !

From exercise, there is $s' \in L'_n$ such that $\frac{s_{\max}}{s'} \leq (1+\delta)^n$.

Approximation ratio: $\dfrac{s_{\max}}{\text{last}(L'_n)} \leq \dfrac{s_{\max}}{s'} \leq \boxed{(1+\delta)^n}$.

**Claim:** $(1+\delta)^n \leq 1 + 2n\delta$ if $2n\delta \leq 1$.
Induction: $(1+\delta)^0 = 1 = 1 + 2 \cdot 0 \cdot \delta$. ✓

# Theorem

**Thm.:** The alg. is an FPTAS.

APPROX-SUBSET-SUM($S, t, \varepsilon$)
  $L'_0 = [0]$
  for $k = 1, \ldots, n$
    $L'_k$=MERGE-LISTS($L'_{k-1}, L'_{k-1} + x_k$)
    $L'_k$=TRIM($L'_k, \boxed{\varepsilon/2n}$)
    remove duplicates and elm.s $> t$
  return last($L'_n$)

*Proof:* Feasibility: Opt. is $s_{\max} = \text{last}(L_n) \leq t$ and $\text{last}(L'_n) \leq \text{last}(L_n)$.
Approx. ratio: Assume we trim with $\boxed{\delta}$.

From exercise: $\forall s \in L_k \exists s' \in L'_k : s' \leq \boxed{s \leq (1+\delta)^k s'}$.

$\Rightarrow \frac{s}{s'} \leq (1+\delta)^k$

want $\leq 1 + \varepsilon$ !

From exercise, there is $s' \in L'_n$ such that $\frac{s_{\max}}{s'} \leq (1+\delta)^n$.

$$\text{Approximation ratio: } \frac{s_{\max}}{\text{last}(L'_n)} \leq \frac{s_{\max}}{s'} \leq \boxed{(1+\delta)^n}.$$

**Claim:** $(1+\delta)^n \leq 1 + 2n\delta$ if $2n\delta \leq 1$.

Induction: $(1+\delta)^0 = 1 = 1 + 2 \cdot 0 \cdot \delta$. ✓

$(1+\delta)^n = (1+\delta)^{n-1}(1+\delta) \leq (1 + 2(n-1)\delta)(1+\delta)$

$= 1 + 2n\delta - 2\delta + \delta + \delta \cdot 2(n-1)\delta$

use induction hypothesis

# Theorem

**Thm.:** The alg. is an FPTAS.

```
APPROX-SUBSET-SUM(S, t, ε)
    L'_0 = [0]
    for k = 1, ..., n
        L'_k = MERGE-LISTS(L'_{k-1}, L'_{k-1} + x_k)
        L'_k = TRIM(L'_k, [ε/2n])
        remove duplicates and elm.s > t
    return last(L'_n)
```

*Proof:* Feasibility: Opt. is $s_{\max} = \text{last}(L_n) \le t$ and $\text{last}(L'_n) \le \text{last}(L_n)$.

Approx. ratio: Assume we trim with $\boxed{\delta}$.

From exercise: $\forall s \in L_k \exists s' \in L'_k : s' \le \boxed{s \le (1+\delta)^k s'}$.

$\Rightarrow \frac{s}{s'} \le (1+\delta)^k$

want $\le 1 + \varepsilon$ !

From exercise, there is $s' \in L'_n$ such that $\frac{s_{\max}}{s'} \le (1+\delta)^n$.

$$\text{Approximation ratio: } \frac{s_{\max}}{\text{last}(L'_n)} \le \frac{s_{\max}}{s'} \le \boxed{(1+\delta)^n}.$$

**Claim:** $(1+\delta)^n \le 1 + 2n\delta$ if $2n\delta \le 1$.

Induction: $(1+\delta)^0 = 1 = 1 + 2 \cdot 0 \cdot \delta$. ✓

$(1+\delta)^n = (1+\delta)^{n-1}(1+\delta) \le (1 + 2(n-1)\delta)(1+\delta)$

$= 1 + 2n\delta - 2\delta + \delta + \delta \cdot \underbrace{\boxed{2(n-1)\delta}}_{<1}$

use induction hypothesis

# Theorem

**Thm.:** The alg. is an FPTAS.

*Proof:* Feasibility: Opt. is $s_{\max} = \text{last}(L_n) \leq t$ and $\text{last}(L'_n) \leq \text{last}(L_n)$.
Approx. ratio: Assume we trim with $\boxed{\delta}$.

APPROX-SUBSET-SUM($S, t, \varepsilon$)
- $L'_0 = [0]$
- for $k = 1, \ldots, n$
  - $L'_k$=MERGE-LISTS($L'_{k-1}, L'_{k-1} + x_k$)
  - $L'_k$=TRIM($L'_k, \boxed{\varepsilon/2n}$)
  - remove duplicates and elm.s $> t$
- return last($L'_n$)

From exercise: $\forall s \in L_k \exists s' \in L'_k : s' \leq \boxed{s \leq (1+\delta)^k s'}$.

$\Rightarrow \frac{s}{s'} \leq (1+\delta)^k$

From exercise, there is $s' \in L'_n$ such that $\frac{s_{\max}}{s'} \leq (1+\delta)^n$.

want $\leq 1 + \varepsilon$ !

$$\text{Approximation ratio: } \frac{s_{\max}}{\text{last}(L'_n)} \leq \frac{s_{\max}}{s'} \leq \boxed{(1+\delta)^n}.$$

**Claim:** $(1+\delta)^n \leq 1 + 2n\delta$ if $2n\delta \leq 1$.
Induction: $(1+\delta)^0 = 1 = 1 + 2 \cdot 0 \cdot \delta$. ✓
$(1+\delta)^n = (1+\delta)^{n-1}(1+\delta) \leq (1 + 2(n-1)\delta)(1+\delta)$
$= 1 + 2n\delta - 2\delta + \delta + \delta \cdot \underbrace{\boxed{2(n-1)\delta}}_{<1} \leq 1 + 2n\delta$.

use induction hypothesis

# Theorem

**Thm.:** The alg. is an FPTAS.

APPROX-SUBSET-SUM$(S, t, \varepsilon)$

$L'_0 = [0]$
for $k = 1, \ldots, n$
  $L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$
  $L'_k$=TRIM$(L'_k, \boxed{\varepsilon/2n})$
  remove duplicates and elm.s $> t$
return last$(L'_n)$

*Proof:* Feasibility: Opt. is $s_{\max} =$ last$(L_n) \le t$ and last$(L'_n) \le$ last$(L_n)$.
Approx. ratio: Assume we trim with $\boxed{\delta}$.

From exercise: $\forall s \in L_k \exists s' \in L'_k : s' \le \boxed{s \le (1+\delta)^k s'}$.

$\Rightarrow \frac{s}{s'} \le (1+\delta)^k$

want $\le 1 + \varepsilon$ !

From exercise, there is $s' \in L'_n$ such that $\frac{s_{\max}}{s'} \le (1+\delta)^n$.

Approximation ratio: $\dfrac{s_{\max}}{\text{last}(L'_n)} \le \dfrac{s_{\max}}{s'} \le \boxed{(1+\delta)^n}$.

**Claim:** $(1+\delta)^n \le 1 + 2n\delta$ if $2n\delta \le 1$.

$\boxed{\delta := \varepsilon/2n \Rightarrow 1 + 2n\delta \le 1 + \varepsilon \checkmark}$

Induction: $(1+\delta)^0 = 1 = 1 + 2 \cdot 0 \cdot \delta$. ✓
$(1+\delta)^n = (1+\delta)^{n-1}(1+\delta) \le (1 + 2(n-1)\delta)(1+\delta)$
$= 1 + 2n\delta - 2\delta + \delta + \delta \cdot \underbrace{\boxed{2(n-1)\delta}}_{<1} < 1 + 2n\delta.$

use induction hypothesis

# Running time

**Thm.:** The alg. is an FPTAS.

Running time:

$$O\left(\sum_{k=1}^{n} |L'_k|\right).$$

Sum of the primed list

length

APPROX-SUBSET-SUM$(S, t, \varepsilon)$
  $L'_0 = [0]$
  for $k = 1, \ldots, n$
    $L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$
    $L'_k$=TRIM$(L'_k, \varepsilon/2n)$
    remove duplicates and elm.s $> t$
  return last$(L'_n)$

# Running time

**Thm.:** The alg. is an FPTAS.

```
APPROX-SUBSET-SUM(S, t, ε)
  L'_0 = [0]
  for k = 1, ..., n
    L'_k = MERGE-LISTS(L'_{k-1}, L'_{k-1} + x_k)
    L'_k = TRIM(L'_k, ε/2n)
    remove duplicates and elm.s > t
  return last(L'_n)
```

Running time:

$$O\left(\sum_{k=1}^{n} |L'_k|\right).$$

Claim: $|L'_k| = O(\frac{n \log t}{\varepsilon})$.

# Running time

**Thm.:** The alg. is an FPTAS.

Running time:

$$O\left(\sum_{k=1}^{n} |L'_k|\right).$$

APPROX-SUBSET-SUM$(S, t, \varepsilon)$
- $L'_0 = [0]$
- for $k = 1, \ldots, n$
  - $L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$
  - $L'_k$=TRIM$(L'_k, \varepsilon/2n)$
  - remove duplicates and elm.s $> t$
- return last$(L'_n)$

Claim: $|L'_k| = O(\frac{n \log t}{\varepsilon})$.

Recall $\delta = \varepsilon/2n$.

Let $L'_k = [0, s_0, s_1, \ldots, s_m]$. Then

$t \geq s_m > (1+\delta)s_{m-1} > \ldots > (1+\delta)^m s_0 \geq (1+\delta)^m$.

# Running time

**Thm.:** The alg. is an FPTAS.

Running time:

$$O\left(\sum_{k=1}^{n} |L'_k|\right).$$

APPROX-SUBSET-SUM$(S, t, \varepsilon)$
- $L'_0 = [0]$
- for $k = 1, \ldots, n$
  - $L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$
  - $L'_k$=TRIM$(L'_k, \varepsilon/2n)$
  - remove duplicates and elm.s $> t$
- return last$(L'_n)$

Claim: $|L'_k| = O(\frac{n \log t}{\varepsilon})$.

Recall $\delta = \varepsilon/2n$.

Let $L'_k = [0, s_0, s_1, \ldots, s_m]$. Then

$t \geq s_m > (1+\delta)s_{m-1} > \ldots > (1+\delta)^m s_0 \geq (1+\delta)^m$.

因为int

So $m < \log_{1+\delta} t = \frac{\ln t}{\ln(1+\delta)}$.

# Running time

**Thm.:** The alg. is an FPTAS.

Running time:

$$O\left(\sum_{k=1}^{n} |L'_k|\right).$$

APPROX-SUBSET-SUM$(S, t, \varepsilon)$
  $L'_0 = [0]$
  for $k = 1, \ldots, n$
    $L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$
    $L'_k$=TRIM$(L'_k, \varepsilon/2n)$
    remove duplicates and elm.s $> t$
  return last$(L'_n)$

Claim: $|L'_k| = O(\frac{n \log t}{\varepsilon})$.

Recall $\delta = \varepsilon/2n$.

Let $L'_k = [0, s_0, s_1, \ldots, s_m]$. Then
$t \geq s_m > (1+\delta)s_{m-1} > \ldots > (1+\delta)^m s_0 \geq (1+\delta)^m$.

So $m < \log_{1+\delta} t = \frac{\ln t}{\ln(1+\delta)}$.

CLRS eq. (3.17): if $\delta > -1$: $\delta \geq \ln(1+\delta) \geq \frac{\delta}{1+\delta}$.

# Running time

**Thm.:** The alg. is an FPTAS.

APPROX-SUBSET-SUM$(S, t, \varepsilon)$
$L'_0 = [0]$
for $k = 1, \ldots, n$
$L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$
$L'_k$=TRIM$(L'_k, \varepsilon/2n)$
remove duplicates and elm.s $> t$
return last$(L'_n)$

Running time:

$$O\left(\sum_{k=1}^{n} |L'_k|\right).$$

Claim: $|L'_k| = O(\frac{n \log t}{\varepsilon})$.

Recall $\delta = \varepsilon/2n$.

Let $L'_k = [0, s_0, s_1, \ldots, s_m]$. Then
$t \geq s_m > (1+\delta)s_{m-1} > \ldots > (1+\delta)^m s_0 \geq (1+\delta)^m$.

So $m < \log_{1+\delta} t = \frac{\ln t}{\ln(1+\delta)}$.

CLRS eq. (3.17): if $\delta > -1$: $\delta \geq \ln(1+\delta) \geq \frac{\delta}{1+\delta}$.

So $m < \frac{\ln t}{\ln(1+\delta)} \leq \frac{\ln t}{\frac{\delta}{1+\delta}} = \frac{(1+\delta)\ln t}{\delta} \leq \frac{2\ln t}{\delta} = \frac{4n \ln t}{\varepsilon}$.

# Running time

**Thm.:** The alg. is an FPTAS.

APPROX-SUBSET-SUM$(S, t, \varepsilon)$
  $L'_0 = [0]$
  for $k = 1, \ldots, n$
    $L'_k$=MERGE-LISTS$(L'_{k-1}, L'_{k-1} + x_k)$
    $L'_k$=TRIM$(L'_k, \varepsilon/2n)$
    remove duplicates and elm.s $> t$
  return last$(L'_n)$

Running time:

$$O\left(\sum_{k=1}^{n} |L'_k|\right).$$

Claim: $|L'_k| = O(\frac{n \log t}{\varepsilon})$.

Recall $\delta = \varepsilon/2n$.

Let $L'_k = [0, s_0, s_1, \ldots, s_m]$. Then

$t \geq s_m > (1+\delta)s_{m-1} > \ldots > (1+\delta)^m s_0 \geq (1+\delta)^m$.

So $m < \log_{1+\delta} t = \frac{\ln t}{\ln(1+\delta)}$.

CLRS eq. (3.17): if $\delta > -1$: $\delta \geq \ln(1+\delta) \geq \frac{\delta}{1+\delta}$.

So $m < \frac{\ln t}{\ln(1+\delta)} \leq \frac{\ln t}{\frac{\delta}{1+\delta}} = \frac{(1+\delta)\ln t}{\delta} \leq \frac{2\ln t}{\delta} = \frac{4n \ln t}{\varepsilon}$.

Total running time: $O\left(\sum_{k=1}^{n} |L'_k|\right) = O\left(\frac{n^2 \ln t}{\varepsilon}\right)$.

poly in $n$ and $\log t$

polynomial in

input size

and in $\frac{1}{\varepsilon}$

FPTAS