Approximation Algorithms

Bin Wang

School of Software
Tsinghua University

May 31, 2022

Outline

1. Overview
2. Vertex-cover problem
3. TSP
4. Set-covering problem
5. Randomization and LP
6. Subset-sum problem

Performance ratios for approximation algorithms

Definition

We say that an algorithm for a problem has an ***approximation ratio*** of $\rho(n)$ if, for any input of size n, the cost C of the solution produced by the algorithm is within a factor of $\rho(n)$ of the cost C^* of an optimal solution:

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

Performance ratios for approximation algorithms

Definition

- An ***approximation scheme*** for an optimization problem is an approximation algorithm that takes as input not only an instance of the problem, but also a value $\epsilon > 0$ such that for any fixed ϵ, the scheme is a $(1 + \epsilon)$-approximation algorithm.

Performance ratios for approximation algorithms

Definition

- We say that an approximation scheme is a ***polynomial-time approximation scheme*** if for any fixed $\epsilon > 0$, the scheme runs in time polynomial in the size n of its input instance.

Performance ratios for approximation algorithms

Definition

- We say that an approximation scheme is a ***fully polynomial-time approximation scheme*** if it is an approximation scheme and its running time is polynomial both in $1/\epsilon$ and in the size n of the input instance.

  For example, the scheme might have a running time of $O((1/\epsilon)^2 n^3)$.

The vertex-cover problem

Definition

- A ***vertex cover*** of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if (u, v) is an edge of G, then either $u \in V'$ or $v \in V'$ (or both).
- The ***vertex-cover problem*** is to find a vertex cover of minimum size in a given undirected graph. We call such a vertex cover an ***optimal vertex cover***.

The vertex-cover problem

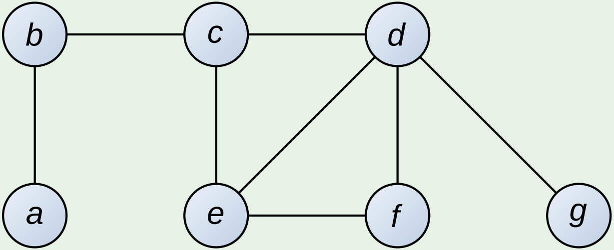
APPROX-VERTEX-COVER(G)

$C = \emptyset$
$E' = G.E$
while $E' \neq \emptyset$
  let (u, v) be an arbitrary edge of E'
  $C = C \cup \{u, v\}$
  remove from E' every edge incident on either u or v
return C

Skip Example

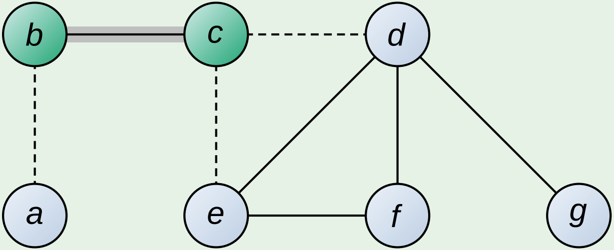
The vertex-cover problem

Example



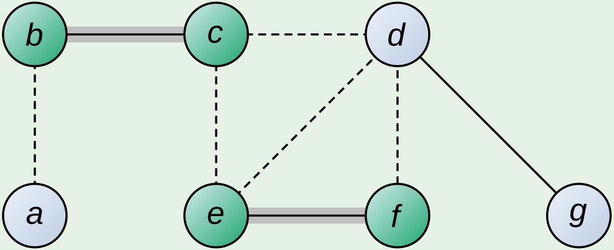
The vertex-cover problem

Example



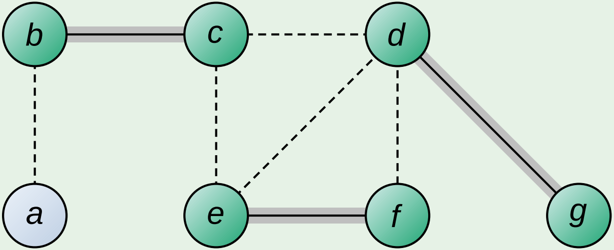
The vertex-cover problem

Example



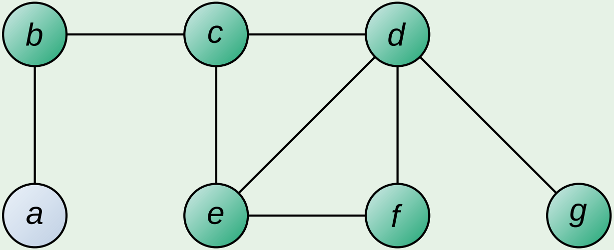
The vertex-cover problem

Example



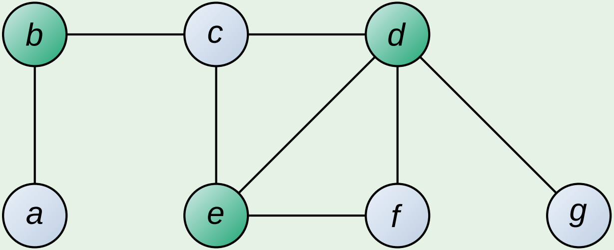
The vertex-cover problem

Example



The vertex-cover problem

Example



The vertex-cover problem

Theorem 35.1

APPROX-VERTEX-COVER is a polynomial-time 2-approximation algorithm.

Proof.

- The running time of this algorithm is $O(V + E)$, using adjacency lists to represent E'.

The vertex-cover problem

Theorem 35.1

APPROX-VERTEX-COVER is a polynomial-time 2-approximation algorithm.

Proof.

- The running time of this algorithm is $O(V + E)$, using adjacency lists to represent E'.

The vertex-cover problem

Proof.

- Let A denote the set of edges that were picked in line 4 of APPROX-VERTEX-COVER. Thus we have $|C^*| \geq |A|$.
- $|C| = 2|A| \leq 2|C^*|$

□

The traveling-salesman problem

Definition

$$\textbf{TSP} = \{\langle G, c, k\rangle : G = (V, E) \text{ is a complete graph, } c \text{ is a function from } V \times V \to \mathbb{Z},\ k \in \mathbb{Z}, \text{ and } G \text{ has a traveling-salesman tour with cost at most } k\}$$

TSP with triangle inequality

Triangle inequality

- Let $c(A)$ denote the total cost of the edges in the subset $A \subseteq E$:

$$c(A) = \sum_{(u,v) \in A} c(u, v).$$

TSP with triangle inequality

Triangle inequality

- We say that cost function c satisfies the ***triangle inequality*** if for all vertices $u, v, w \in V$,

$$c(u,w) \leq c(u,v) + c(v,w).$$

TSP with triangle inequality

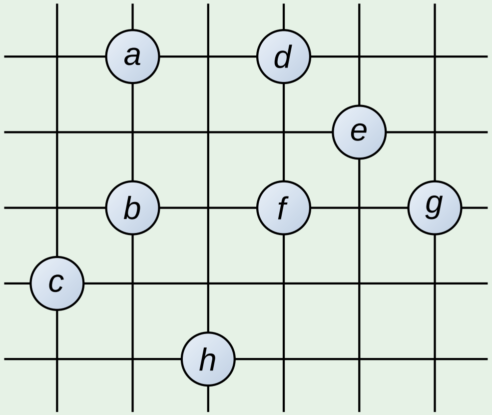
APPROX-TSP-TOUR(G, c)

1 select a vertex $r \in G.V$ to be a "root" vertex
2 compute a minimum spanning tree T for G
  from root r using MST-PRIM(G, c, r)
3 let H be the list of vertices, ordered according
  to a preorder tree walk of T
4 **return** the hamiltonian cycle H

Skip Example

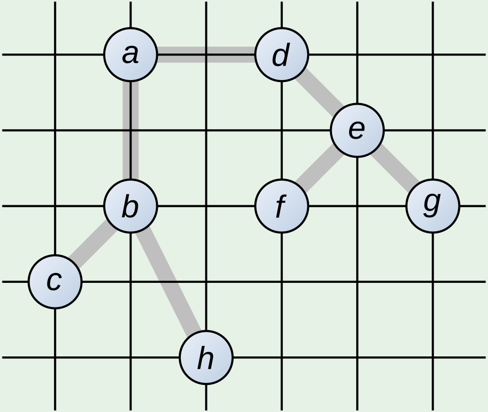
TSP with triangle inequality

Example



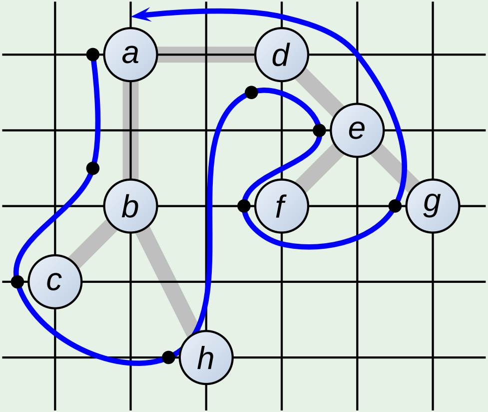
TSP with triangle inequality

Example



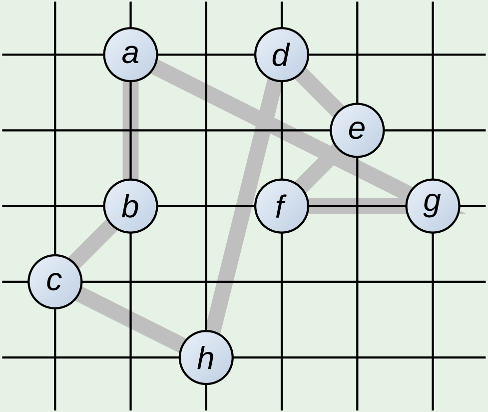
TSP with triangle inequality

Example



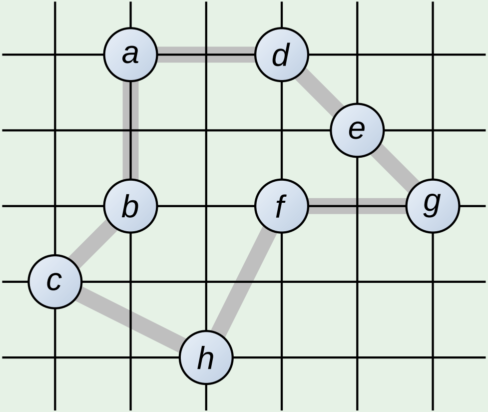
TSP with triangle inequality

Example



TSP with triangle inequality

Example



TSP with triangle inequality

Theorem 35.2

APPROX-TSP-TOUR is a polynomial-time 2-approximation algorithm for the traveling-salesman problem with the triangle inequality.

TSP with triangle inequality

Proof.

- The running time of APPROX-TSP-TOUR is $\Theta(V^2)$.
- $c(T) \leq c(H^*)$
- A ***full walk*** of T lists the vertices when they are first visited and also whenever they are returned to after a visit to a subtree. Let us call this walk W .

$$c(W) = 2c(T) \leq 2c(H^*).$$

$\square$

TSP with triangle inequality

Proof.

- The running time of APPROX-TSP-TOUR is $\Theta(V^2)$.
- $c(T) \le c(H^*)$
- A ***full walk*** of T lists the vertices when they are first visited and also whenever they are returned to after a visit to a subtree. Let us call this walk W.

$$c(W) = 2c(T) \le 2c(H^*).$$

$\square$

TSP with triangle inequality

Proof.

- The running time of APPROX-TSP-TOUR is $\Theta(V^2)$.
- $c(T) \leq c(H^*)$
- A ***full walk*** of T lists the vertices when they are first visited and also whenever they are returned to after a visit to a subtree. Let us call this walk W.

$$c(W) = 2c(T) \leq 2c(H^*).$$

□

TSP with triangle inequality

Proof.

- The running time of APPROX-TSP-TOUR is $\Theta(V^2)$.
- $c(T) \leq c(H^*)$
- A ***full walk*** of T lists the vertices when they are first visited and also whenever they are returned to after a visit to a subtree. Let us call this walk W .
  $c(H) \leq c(W) = 2c(T) \leq 2c(H^*)$.

□

The general TSP

Theorem 35.3

If $P \neq NP$, then for any constant $\rho \geq 1$, there is ***no*** polynomial-time approximation algorithm with approximation ratio ρ for the general traveling-salesman problem.

The general TSP

Proof.

- Suppose to the contrary that for some number $\rho \geq 1$, there is a polynomial-time approximation algorithm A with approximation ratio ρ. Without loss of generality, we assume that ρ is an integer, by rounding it up if necessary.
- Let $G = (V, E)$ be an instance of the hamiltonian-cycle problem.

The general TSP

Proof.

- Let $G' = (V, E')$ be the complete graph on V ; that is, $E' = \{(u,v) : u, v \in V \text{ and } u \neq v\}$.
- Assign an integer cost to each edge in E' as follows:

$$c(u,v) = \begin{cases} 1 & \text{if } (u,v) \in E \\ \rho|V| + 1 & \text{otherwise.} \end{cases}$$

- $(\rho|V| + 1) + (|V| - 1) = \rho|V| + |V| > \rho|V|$

$\square$

The general TSP

Proof.

- Let $G' = (V, E')$ be the complete graph on V ; that is, $E' = \{(u, v) : u, v \in V \text{ and } u \neq v\}$.
- Assign an integer cost to each edge in E' as follows:

$$c(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ \rho|V| + 1 & \text{otherwise.} \end{cases}$$

- $(\rho|V| + 1) + (|V| - 1) = \rho|V| + |V| > \rho|V|$

□

The general TSP

Proof.

- Let $G' = (V, E')$ be the complete graph on V ; that is, $E' = \{(u, v) : u, v \in V \text{ and } u \neq v\}$.
- Assign an integer cost to each edge in E' as follows:

$$c(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ \rho|V| + 1 & \text{otherwise.} \end{cases}$$

- $(\rho|V| + 1) + (|V| - 1) = \rho|V| + |V| > \rho|V|$ $\square$

The set-covering problem

Definition

- An instance $(X, \mathcal{F})$ of the ***set-covering problem*** consists of a finite set X and a family $\mathcal{F}$ of subsets of X, such that every element of X belongs to at least one subset in :

$$X = \bigcup_{S \in \mathcal{F}} S.$$

The set-covering problem

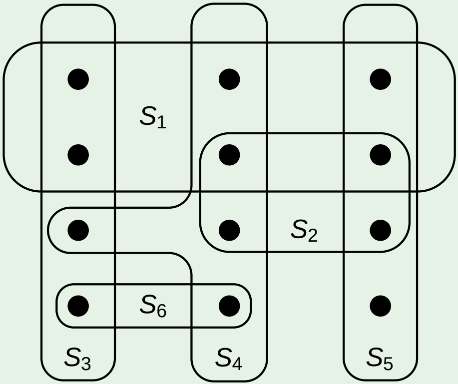
Definition

- The problem is to find a ***minimum-size*** subset $\mathcal{C} \subseteq \mathcal{F}$ whose members cover all of X:

$$X = \bigcup_{S \in \mathcal{C}} S.$$

The set-covering problem

Example



A greedy approximation algorithm

GREEDY-SET-COVER$(X, \mathcal{F})$

```
U = X
C = ∅
while U ≠ ∅
    select an S ∈ F
        that maximizes |S ∩ U|
    U = U − S
    C = C ∪ {S}
return C
```

$U = X$
$\mathcal{C} = \emptyset$
while $U \neq \emptyset$
  select an $S \in \mathcal{F}$ that maximizes $|S \cap U|$
  $U = U - S$
  $\mathcal{C} = \mathcal{C} \cup \{S\}$
return $\mathcal{C}$

A greedy approximation algorithm

Theorem 35.4

GREEDY-SET-COVER is a polynomial-time $\rho(n)$-approximation algorithm, where $\rho(n) = H(\max\{|S| : S \in \mathcal{F}\})$.
We define $H(d) = \sum_{i=1}^{d} 1/i$ and $H(0) = 0$.

A greedy approximation algorithm

Proof.

- The algorithm GREEDY-SET-COVER can easily be implemented to run in time polynomial in $|X|$ and $|\mathcal{F}|$. For example, there is an implementation that runs in time $O(|X||\mathcal{F}|\min(|X|, |\mathcal{F}|))$.

A greedy approximation algorithm

Proof.

- Let S_i denote the ith subset selected by GREEDY-SET-COVER; the algorithm incurs a cost of 1 when it adds S_i to $\mathcal{C}$. We spread this cost of selecting S_i evenly among the elements covered for the first time by S_i.

A greedy approximation algorithm

Proof.

- Let c_x denote the cost allocated to element x, for each $x \in X$. If x is covered for the first time by S_i, then

$$c_x = \frac{1}{|S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|}$$

A greedy approximation algorithm

Proof.

- At each step of the algorithm, 1 unit of cost is assigned, and so $|\mathcal{C}| = \sum_{x \in X} c_x$.
- The cost assigned to the optimal cover is $\sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x$,

A greedy approximation algorithm

Proof.

- and since each $x \in X$ is in at least once set $S \in \mathcal{C}^*$, we have $|\mathcal{C}| = \sum_{x \in X} c_x \leq \sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x$.
- The remainder of the proof rests on the following key inequality, which we shall prove shortly. For any set S belonging to the family , $\sum_{x \in S} c_x \leq H(|S|)$.
- $|\mathcal{C}| \leq \sum_{S \in \mathcal{C}^*} H(|S|) \leq |\mathcal{C}^*| \cdot H(\max |S|)$.

A greedy approximation algorithm

Proof.

- Consider any set $S \in \mathcal{F}$ and $i = 1, 2, \ldots, |\mathcal{C}|$, and let $u_i = |S - (S_1 \cup S_2 \cup \ldots \cup S_i)|$ be the number of elements in S remaining uncovered after $S_1, S_2, \ldots, S_i$ have been selected by the algorithm.

A greedy approximation algorithm

Proof.

- We define $u_0 = |S|$ to be the number of elements of S, which are all initially uncovered. Let k be the least index such that $u_k = 0$, so that each element in S is covered by at least one of the sets $S_1, S_2, \ldots, S_k$.
- Then, $u_{i-1} \geq u_i$, and $u_{i-1} - u_i$ elements of S are covered for the first time by S_i.

A greedy approximation algorithm

Proof.

- Thus

$$\sum_{x \in S} c_x = \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot \frac{1}{|S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|}$$

A greedy approximation algorithm

Proof.

- Observe that

$$\begin{aligned} & |S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})| \\ \geq & |S - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})| \\ = & u_{i-1}, \end{aligned}$$

  because the greedy choice of S_i guarantees that S cannot cover more new elements that S_i does.

A greedy approximation algorithm

Proof.

- Consequently, we obtain

$$\begin{aligned}
\sum_{x\in S} c_x &\leq \sum_{i=1}^{k}(u_{i-1}-u_i)\cdot\frac{1}{u_{i-1}} = \sum_{i=1}^{k}\sum_{j=u_i+1}^{u_{i-1}}\frac{1}{u_{i-1}} \\
&\leq \sum_{i=1}^{k}\sum_{j=u_i+1}^{u_{i-1}}\frac{1}{j} \qquad (\textit{because } j\leq u_{i-1}) \\
&= \sum_{i=1}^{k}\left(\sum_{j=1}^{u_{i-1}}\frac{1}{j}-\sum_{j=1}^{u_i}\frac{1}{j}\right)
\end{aligned}$$

A greedy approximation algorithm

Proof.

- Thus,

$$
\begin{aligned}
\sum_{x \in S} c_x &\leq \sum_{i=1}^{k} (H(u_{i-1}) - H(u_i)) \\
&= H(u_0) - H(u_k) \\
&= H(u_0) \\
&= H(|S|).
\end{aligned}
$$

$\square$

A greedy approximation algorithm

Corollary 35.5

GREEDY-SET-COVER is a polynomial-time $(\ln |X| + 1)$-approximation algorithm

Randomization

Definition

We say that a randomized algorithm for a problem has an ***approximation ratio*** of $\rho(n)$ if, for any input of size n, the ***expected*** cost C of the solution produced by the algorithm is within a factor of $\rho(n)$ of the cost C^* of an optimal solution:

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

Randomization

Definition

We also call a randomized algorithm that achieves an approximation ratio of $\rho(n)$ a $\rho(\mathbf{n})$ ***randomized approximation algorithm***.

Randomization

Theorem 35.6

Given an instance of ***MAX-3-CNF*** satisfiability with n variables $x_1, x_2, \ldots, x_n$ and m clauses, the ***randomized algorithm*** that independently sets each variable to 1 with probability $1/2$ and to 0 with probability $1/2$ is a randomized $8/7$-approximation algorithm.

Randomization

Proof.

For $i = 1, 2, \ldots, m$, we define the indicator random variable

$$Y_i = I\{\text{ clause } i \text{ is satisfied }\}$$

Let Y be the number of satisfied clauses overall, so that $Y = Y_1 + Y_2 + \cdots + Y_m$.

Randomization

Proof.

For $i = 1, 2, \ldots, m$, we define the indicator random variable

$$\begin{aligned} Y_i &= I\{ \text{ clause } i \text{ is satisfied } \} \\ &= 1 - 1/8 \end{aligned}$$

Let Y be the number of satisfied clauses overall, so that $Y = Y_1 + Y_2 + \cdots + Y_m$.

Randomization

Proof.

Then we have

$$E[Y] = E\left[\sum_{i=1}^{m} Y_i\right]$$
$$= \sum_{i=1}^{m} E[Y_i] = \sum_{i=1}^{m} 7/8 = 7m/8$$

Hence the approximation ratio is at most $m/(7m/8) = 8/7$. $\square$

Randomization

Proof.

Then we have

$$
\begin{aligned}
E[Y] &= E\left[\sum_{i=1}^{m} Y_i\right] \\
&= \sum_{i=1}^{m} E[Y_i] = \sum_{i=1}^{m} 7/8 = 7m/8
\end{aligned}
$$

Hence the approximation ratio is at most $m/(7m/8) = 8/7$. $\square$

Linear programming

Definition

In the ***minimum-weight vertex-cover problem***, we are given an undirected graph $G = (V, E)$ in which each vertex $v \in V$ has an associated positive weight $w(v)$.
For any vertex cover $V' \subseteq V$, we define the weight of the vertex cover $w(V') = \sum_{v \in V'} w(v)$.
The goal is to find a vertex cover of minimum weight.

Linear programming

0-1 integer program

The following $0-1$ integer program is used to find a minimum-weight vertex cover:

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

subject to

$$\begin{aligned} x(u) + x(v) &\geq 1 && \text{for each } (u, v) \in E \\ x(v) &\in \{0, 1\} && \text{for each } v \in V. \end{aligned}$$

Linear programming

linear-programming relaxation

Suppose, we remove the constraint that $x(v) \in \{0, 1\}$ and replace it by $0 \leq x(v) \leq 1$.

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

subject to

$$\begin{array}{rcll} x(u) + x(v) & \geq & 1 & \text{for each } (u, v) \in E \\ x(v) & \leq & 1 & \text{for each } v \in V \\ x(v) & \geq & 0 & \text{for each } v \in V \end{array}$$

Linear programming

APPROX-MIN-WEIGHT-VC(G, w)

$C = \emptyset$
compute $\overline{x}$, an optimal solution
        to the linear program.
for each $v \in V$
        if $\overline{x}(v) \geq 1/2$
                $C = C \cup \{v\}$
return C

Linear programming

Theorem 35.7

Algorithm APPROX-MIN-WEIGHT-VC is a polynomial-time 2-approximation algorithm for the minimum-weight vertex-cover problem.

Linear programming

Proof.

Let C^* be an optimal solution to the minimum-weight vertex-cover problem, and let z^* be the value of an optimal solution to the linear program.

Since an optimal vertex cover is a feasible solution to the linear program, we have $z^* \leq w(C^*)$.

Linear programming

Proof.

$$
\begin{aligned}
z^* &= \sum_{v \in V} w(v)\bar{x}(v) \\
&\geq \sum_{v \in V : \bar{x}(v) \geq 1/2} w(v)\bar{x}(v)
\end{aligned}
$$

Linear programming

Proof.

$$z^* \geq \sum_{v \in V : \bar{x}(v) \geq 1/2} w(v) \cdot \frac{1}{2}$$

$$= \sum_{v \in C} w(v) \cdot \frac{1}{2} = \frac{1}{2} w(C)$$

Thus, $w(C) \leq 2z^* \leq 2w(C^*)$. $\square$

An exponential-time exact algorithm

```
EXACT-SUBSET-SUM(S, t)
n = |S|
L_0 = ⟨0⟩
for each i = 1 to n
        L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
        remove from L_i every element that
                is greater than t
return the largest element in L_n
```

An exponential-time exact algorithm

Example

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

$$
\begin{aligned}
P_1 &= \{0, 1\}, \\
P_2 &= \{0, 1, 4, 5\}, \\
P_3 &= \{0, 1, 4, 5, 6, 9, 10\},
\end{aligned}
$$

A fully polynomial-time approximation scheme

A trimming scheme

- We use a trimming parameter δ such that $0 < \delta < 1$.
- To ***trim*** a list L by δ means to remove as many elements from L as possible,

A fully polynomial-time approximation scheme

A trimming scheme

- For every element y that was removed from L, there is an element z still in L' that approximates y, that is,

$$\frac{y}{1+\delta} \leq z \leq y.$$

- $\delta = 0.1, L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle \Rightarrow$, $L' = \langle 10, 12, 15, 20, 23, 29 \rangle$.

A fully polynomial-time approximation scheme

$\text{TRIM}(L, \delta)$

```
m = |L|
L' = ⟨y_1⟩
last = y_1
for i = 2 to m
        if y_i > last · (1 + δ)
                append y_i onto the end of L'
                last = y_i
return L'
```

A fully polynomial-time approximation scheme

$\textsc{Approx-Subset-Sum}(S, t, \epsilon)$

$n = |S|$
$L_0 = \langle 0 \rangle$
for $i = 1$ **to** n
    $L_i = \textsc{Merge-Lists}(L_{i-1}, L_{i-1} + x_i)$
    $L_i = \textsc{Trim}(L_i, \epsilon/2n)$
    remove from L_i every element that is greater than t
let z^* be the largest value in L_n
return z^*

A fully polynomial-time approximation scheme

Example

Suppose we have the instance $S = \langle 104, 102, 201, 101 \rangle$ with $t = 308$ and $\epsilon = 0.40$. The trimming parameter δ is $\epsilon/8 = 0.05$.

A fully polynomial-time approximation scheme

Example

APPROX-SUBSET-SUM computes the following values on the indicated lines:

$$
\begin{aligned}
\text{line } 2: \quad L_0 &= \langle 0 \rangle \\
\\
\text{line } 4: \quad L_1 &= \langle 0, 104 \rangle \\
\text{line } 5: \quad L_1 &= \langle 0, 104 \rangle \\
\text{line } 6: \quad L_1 &= \langle 0, 104 \rangle
\end{aligned}
$$

A fully polynomial-time approximation scheme

Example

$$\begin{aligned}
\text{line } 4: \quad L_2 &= \langle 0, 102, 104, 206 \rangle \\
\text{line } 5: \quad L_2 &= \langle 0, 102, 206 \rangle \\
\text{line } 6: \quad L_2 &= \langle 0, 102, 206 \rangle
\end{aligned}$$

$$\begin{aligned}
\text{line } 4: \quad L_3 &= \langle 0, 102, 201, 206, 303, 407 \rangle \\
\text{line } 5: \quad L_3 &= \langle 0, 102, 201, 303, 407 \rangle \\
\text{line } 6: \quad L_3 &= \langle 0, 102, 201, 303 \rangle
\end{aligned}$$

A fully polynomial-time approximation scheme

Example

$$
\begin{aligned}
\text{line } 4: \quad L_4 &= \langle 0, 101, 102, 201, 203, 302, 303, 404 \rangle \\
\text{line } 5: \quad L_4 &= \langle 0, 101, 201, 302, 404 \rangle \\
\text{line } 6: \quad L_4 &= \langle 0, 101, 201, 302 \rangle
\end{aligned}
$$

A fully polynomial-time approximation scheme

Theorem 35.8

APPROX-SUBSET-SUM is a fully polynomial-time approximation scheme for the subset-sum problem.

A fully polynomial-time approximation scheme

Proof.

Let $y^* \in P_n$ denote an optimal solution to the subset-sum problem. By induction on i, it can be shown that for every element y in P_i that is at most t, there is a $z \in L_i$ such that

$$\frac{y}{(1+\epsilon/2n)^i} \leq z \leq y\,.$$

A fully polynomial-time approximation scheme

Proof.

Thus

$$
\begin{aligned}
\frac{y^*}{z^*} &\leq \left(1+\frac{\epsilon}{2n}\right)^n \\
&\leq e^{\epsilon/2} \qquad \left(\lim_{n\to\infty}(1+x/n)^n = e^x\right) \\
&\leq 1+\epsilon/2+(\epsilon/2)^2 \quad (1+x<e^x<1+x+x^2) \\
&\leq 1+\epsilon
\end{aligned}
$$

A fully polynomial-time approximation scheme

Proof.

After trimming, successive elements z and z' of L_i must have the relationship $z'/z > 1 + \epsilon/2n$. The number of elements in each list L_i is at most

A fully polynomial-time approximation scheme

Proof.

$$\log_{1+\epsilon/2n} t + 2 = \frac{\ln t}{\ln(1+\epsilon/2n)} + 2$$

$$\leq \frac{2n(1+\epsilon/2n)\ln t}{\epsilon} + 2 \qquad \left(\frac{x}{1+x} \leq \ln(1+x)\right)$$

$$\leq \frac{3n \ln t}{\epsilon} + 2$$

$\square$