CSE 548: Analysis of Algorithms

Lecture 12
(Approximation Algorithms)

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Fall 2015

Approximation Ratio

Consider an optimization problem P in which each potential solution has a positive cost.

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

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

We call A a $\rho(n)$-approximation algorithm.

Approximation Ratio

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

Maximization problem: $0 < C \leq C^*$, and the ratio $\frac{C^*}{C}$ gives the factor by which the cost of an optimal solution is larger than the cost of the approximate solution.

Minimization problem: $0 < C^* \leq C$, and the ratio $\frac{C}{C^*}$ gives the factor by which the cost of the approximate solution is larger than the cost of an optimal solution.

Approximation Scheme

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.

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.

An approximation scheme is a *fully polynomial-time approximation scheme* if it is an approximation scheme and its running time is polynomial in both $\frac{1}{\epsilon}$ and the size n of the input instance.

Vertex Cover

A *vertex cover* of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$, then either $u \in V'$ or $v \in V'$ (or both).

The size of a vertex cover is the number of vertices in it.

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*. This problem is the optimization version of an NP-complete decision problem.

Even though we do not know how to find an optimal vertex cover in polynomial time, we can efficiently find one that is near-optimal.

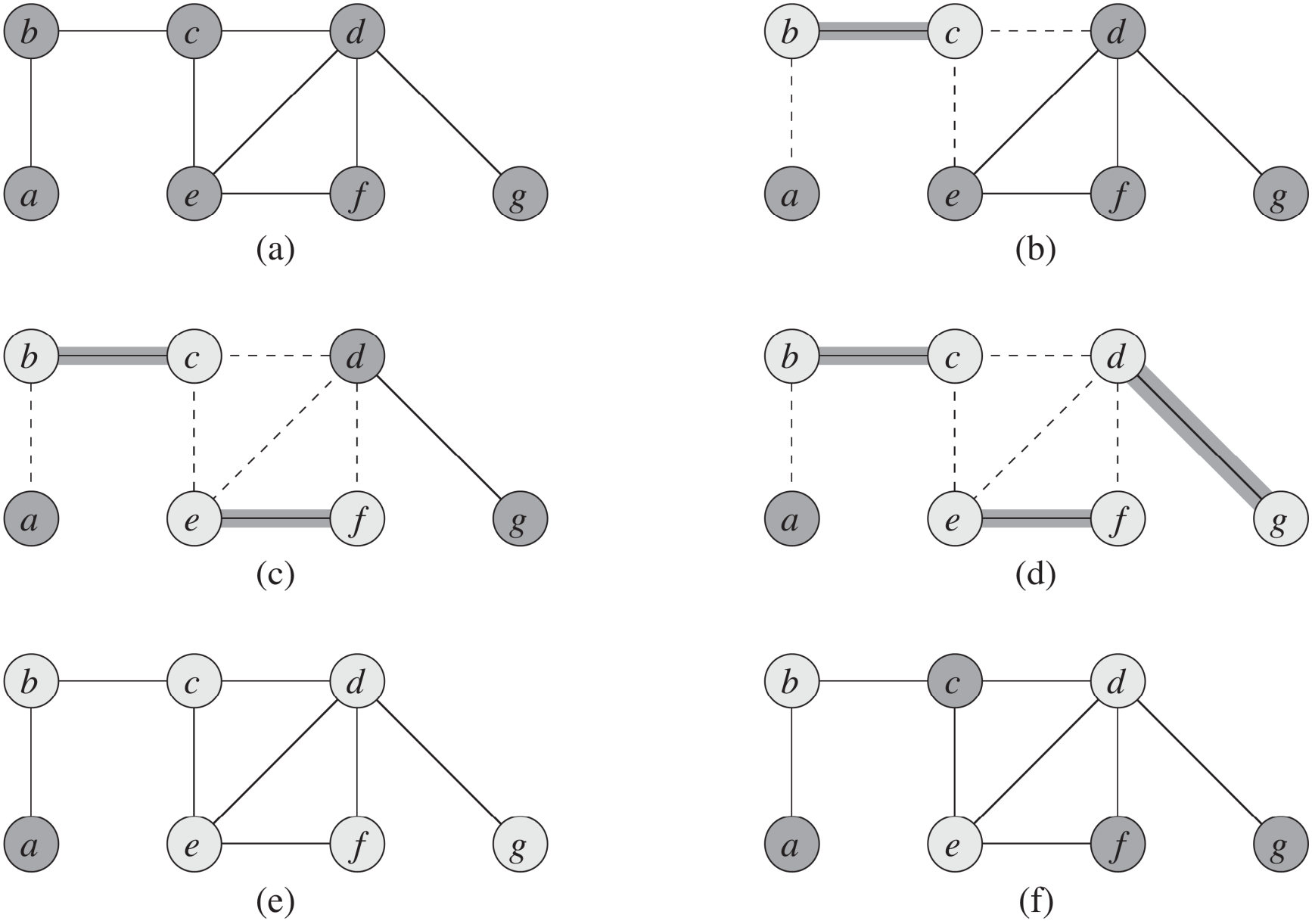
Vertex Cover

Input: Undirected graph $G = (V, E)$ with vertex set V and edge set E.

Output: A vertex cover $C \subseteq G.V$ which is not necessarily optimal.

```
APPROX-VERTEX-COVER ( G = (V, E) )
1.    C ← ∅
2.    E' ← G.E
3.    while E' ≠ ∅ do
4.        let (u, v) be an arbitrary edge of E'
5.        C ← C ∪ {u, v}
6.        remove from E' every edge incident on either u or v
7.    return C
```

Vertex Cover



Source: "Introduction to Algorithms" (3rd edition) by Cormen, Leiserson, Rivest and Stein.

Vertex Cover

Theorem 1: APPROX-VERTEX-COVER is a polynomial-time 2-approximation algorithm for the Vertex Cover problem.

Proof: Let $n = |G.V|$ and $m = |G.E|$. Then running time is clearly $O(n + m)$ assuming E' is represented as an adjacency list.

Since lines 3—6 iterate until every edge in $G.E$ is covered by some vertex in C, the algorithm returns a vertex cover of G in C.

Let A be the set of edges that was picked by line 4. In order to cover the edges in A, any vertex cover — in particular, an optimal cover C^* — must include at least one endpoint of each edge in A. But once an edge is picked in line 4, all other edges incident on its endpoints are removed from E' in line 6. Thus, no two edges in A are covered by the same vertex from C^*, and hence,

$$|C^*| \geq |A|.$$

Vertex Cover

Theorem 1: APPROX-VERTEX-COVER is a polynomial-time 2-approximation algorithm for the Vertex Cover problem.

Proof: … … … no two edges in A are covered by the same vertex from C^*, and hence,

$$|C^*| \geq |A|.$$

Each execution of line 4 picks an edge for which neither of its endpoints are already in C, and hence,

$$|C| = 2|A|.$$

Combining the two relationships above, we get:

$$|C| = 2|A| \leq 2|C^*|.$$

The Traveling Salesman Problem

In the *Traveling Salesman Problem* (TSP) we are given a complete undirected graph $G = (V, E)$ with a nonnegative integer cost $c(u, v)$ associated with each edge $(u, v) \in G.E$.

The goal is to find a *Hamiltonian cycle* (a tour) of G with minimum cost.

We say that the cost function c satisfies the *triangle inequality* provided the following holds for $\forall u, v, w \in G.V$:

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

The TSP problem is NP-complete even if we require the cost function to satisfy the triangle inequality.

The Traveling Salesman Problem with Triangle Inequality

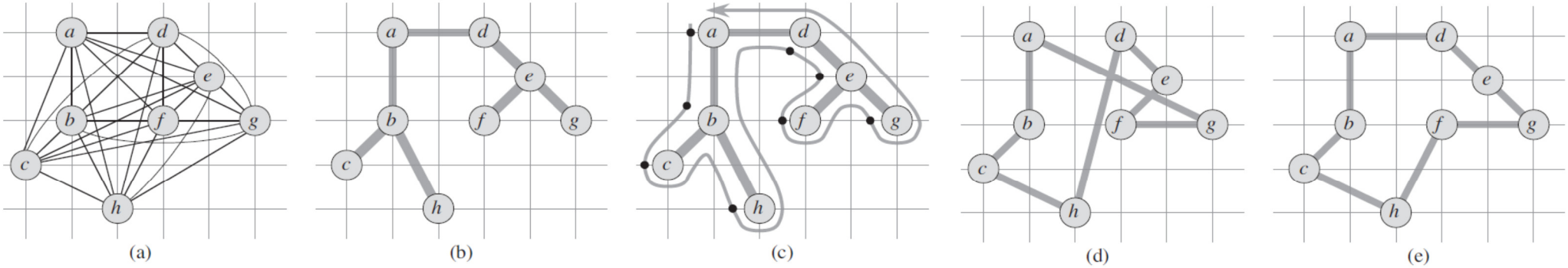
Input: Complete undirected graph $G = (V, E)$ with nonnegative integer cost $c(u, v)$ associated with each edge $(u, v) \in G.E$. The cost function c satisfies the triangle inequality, i.e., for $\forall u, v, w \in G.V$: $c(u, w) \leq c(u, v) + c(v, w)$.

Output: A TSP tour of $G.V$ with cost at most 2 times that of optimal.

APPROX-TSP-TOUR ($G = (V, E)$, 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
3. *let* H *be a list of vertices, ordered according to when they are first visited in a preorder tree walk of* T
4. *return the Hamiltonian cycle* H

The Traveling Salesman Problem with Triangle Inequality



The operation of APPROX-TSP-TOUR:

(a) A complete undirected graph. Vertices lie on intersections of integer grid lines. For example, f is one unit to the right and two units up from h. The cost function between two points is the ordinary Euclidean distance.

(b) A minimum spanning tree T of the complete graph. Vertex a is the root vertex. Only edges in the minimum spanning tree are shown.

(c) A walk of T, starting at a. A full walk of the tree visits the vertices in the order $a, b, c, b, h, b, a, d, e, f, e, g, e, d, a$. A preorder walk of T lists a vertex just when it is first encountered, as indicated by the dot next to each vertex, yielding the ordering a, b, c, h, d, e, f, g.

(d) A tour obtained by visiting the vertices in the order given by the preorder walk, which is the tour H returned by APPROX-TSP-TOUR. Its total cost is approximately 19.074.

(e) An optimal tour H^* for the original complete graph. Its total cost is approximately 14.715.

Source: "Introduction to Algorithms" (3rd edition) by Cormen, Leiserson, Rivest and Stein.

The Traveling Salesman Problem with Triangle Inequality

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

Proof: Since a minimum spanning tree T and its preorder traversal can be found in polynomial time, APPROX-TSP-TOUR is a polynomial-time algorithm.

Let H^* be an optimal tour of the given set of vertices. We obtain a spanning tree by deleting any edge from a tour, and each edge cost is nonnegative. Hence, assuming that $c(A)$ denotes the total cost of the edges in any given subset $A \in G.E$, we have:

$$c(T) \leq c(H^*) \qquad (1)$$

The Traveling Salesman Problem with Triangle Inequality

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

Proof: … … …

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 full walk W . The full walk of our example gives the order

$$a, b, c, b, h, b, a, d, e, f, e, g, e, d, a.$$

Since the full walk traverses every edge of T exactly twice, we have (extending our definition of the cost c in the natural manner to handle multisets of edges):

$$c(W) = 2c(T) \qquad (2)$$

The Traveling Salesman Problem with Triangle Inequality

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

Proof: … … …

Combining (1) and (2), we have:

$$c(W) \leq 2c(H^*) \qquad (3)$$

Unfortunately, W is generally not a tour, since it visits some vertices more than once. By the triangle inequality, however, we can delete a visit to any vertex from W and the cost does not increase. By repeatedly applying this operation, we can remove from W all but the first visit to each vertex. In our example, this leaves the ordering:

$$a, b, c, h, d, e, f, g.$$

The Traveling Salesman Problem with Triangle Inequality

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

Proof: … … …

This ordering is the same as that obtained by a preorder walk of the tree T. Let H be the cycle corresponding to this preorder walk. It is a *Hamiltonian cycle*, since every vertex is visited exactly once, and in fact it is the cycle computed by APPROX-TSP-TOUR. Since H is obtained by deleting vertices from the full walk W, we have

$$c(H) \leq c(W) \qquad (4)$$

Combining inequalities (3) and (4) gives:

$$c(H) \leq 2c(H^*).$$

The General Traveling Salesman Problem (i.e., without Triangle Inequality)

Theorem 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.

Proof: The proof is by contradiction.

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.

We will show how to use A to solve instances of the Hamiltonian-cycle problem in polynomial time. Since we know that Hamiltonian-cycle problem is NP-complete (see Theorem 34.13 of CLRS, 3rd ed), Theorem 3 implies that if we can solve it in polynomial time, then $P = NP$.

The General Traveling Salesman Problem (i.e., without Triangle Inequality)

Theorem 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.

Proof: … … …

Let $G = (V, E)$ be an instance of the Hamiltonian-cycle problem. We wish to determine efficiently whether G contains a Hamiltonian cycle by making use of the hypothesized approximation algorithm A.

We turn G into an instance of the traveling-salesman problem as follows. Let $G' = (V, E')$ be the complete graph on V; that is,

$$E' = \{(u, v): u, v \in V \; and \; u \neq v\}.$$

The General Traveling Salesman Problem (i.e., without Triangle Inequality)

Theorem 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.

Proof: … … …

Assign an integer cost to each edge in E' as follows:

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

We can create representations of G' and c from a representation of G in time polynomial in $|V|$ and $|E|$.

The General Traveling Salesman Problem (i.e., without Triangle Inequality)

Theorem 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.

Proof: … … …

Now, consider the traveling-salesman problem (G', c). If the original graph G has a Hamiltonian cycle H, then the cost function c assigns to each edge of H a cost of 1, and so (G', c) contains a tour of cost $|V|$.

On the other hand, if G does not contain a Hamiltonian cycle, then any tour of G' must use some edge not in E. But any tour that uses an edge not in E has a cost of at least

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

The General Traveling Salesman Problem (i.e., without Triangle Inequality)

Theorem 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.

Proof: … … …

Because edges not in G are so costly, there is a gap of at least $\rho|V|$ between the cost of a tour that is a Hamiltonian cycle in G (cost $|V|$) and the cost of any other tour (cost at least $\rho|V| + |V|$). Therefore, the cost of a tour that is not a Hamiltonian cycle in G is at least a factor of $\rho + 1$ greater than the cost of a tour that is a Hamiltonian cycle in G.

The General Traveling Salesman Problem (i.e., without Triangle Inequality)

Theorem 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.

Proof: … … …

Now, suppose that we apply the approximation algorithm A to the traveling salesman problem (G', c). Because A is guaranteed to return a tour of cost no more than ρ times the cost of an optimal tour, if G contains a Hamiltonian cycle, then A must return it. If G has no Hamiltonian cycle, then A returns a tour of cost more than $\rho|V|$. Therefore, we can use A to solve the Hamiltonian-cycle problem in polynomial time.

Set Covering Problem

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

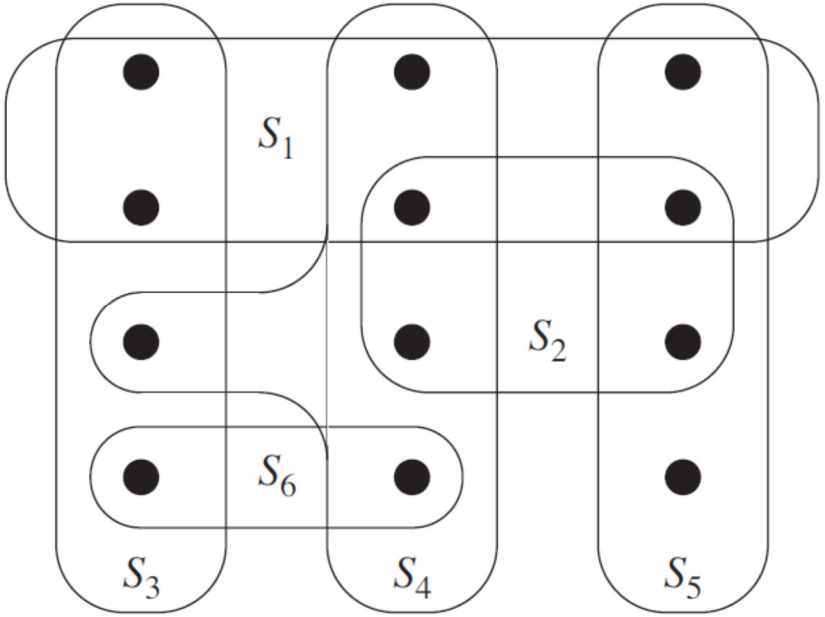
$$X = \bigcup_{S \in F} S .$$

We say that a subset $S \in F$ *covers* its elements. The problem is to find a minimum size subset $C \subseteq F$ whose members cover all of X:

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

We say that any C satisfying the equation above *covers* X.

Set Covering Problem



An instance (X, F) of the set-covering problem, where X consists of the 12 black points and $F = \{S_1, S_2, S_3, S_4, S_5, S_6\}$. A minimum-size set cover is $C = \{S_3, S_4, S_5\}$ with size 3. The greedy algorithm produces a cover of size 4 by selecting either the sets S_1, S_4, S_5, and S_3 or the sets S_1, S_4, S_5, and S_6, in order.

Source: "Introduction to Algorithms" (3rd edition) by Cormen, Leiserson, Rivest and Stein.

Set Covering Problem

Input: A finite set X and a family F of subsets of X, such that every element of X belongs to at least one subset in F.

Output: A set $C \subseteq F$ covering X which is not necessarily optimal.

```
GREEDY-SET-COVER ( X, F )
1.    U ← X
2.    C ← ∅
3.    while U ≠ ∅ do
4.        select an S ∈ F that maximizes |S ∩ U|
5.        U ← U − S
6.        C ← C ∪ {S}
7.    return C
```

Set Covering Problem

Theorem 4: GREEDY-SET-COVER is a polynomial-time $\rho(n)$-approximation algorithm, where $\rho(n) = H(\max\{|S|: S \in F\})$, $H(d) = \sum_{i=1}^{d} \frac{1}{i}$ and $H(0) = 0$.

Proof: GREEDY-SET-COVER clearly runs in polynomial time.

To show that GREEDY-SET-COVER is a $\rho(n)$-approximation algorithm, we assign a cost of 1 to each set selected by the algorithm, distribute this cost over the elements covered for the first time, and then use these costs to derive the desired relationship between the size of an optimal set cover C^* and the size of the set cover C returned by the algorithm.

Let S_i denote the i-th subset selected by GREEDY-SET-COVER; a cost of 1 is incurred when S_i is added to C. We spread this cost of selecting S_i evenly among the elements covered for the first time by S_i.

Set Covering Problem

Theorem 4: GREEDY-SET-COVER is a polynomial-time $\rho(n)$-approximation algorithm, where $\rho(n) = H(\max\{|S|: S \in F\})$, $H(d) = \sum_{i=1}^{d} \frac{1}{i}$ and $H(0) = 0$.

Proof: … … …

Let c_x denote the cost allocated to element x, for each $x \in X$. Each element is assigned a cost only once, when it is covered for the first time. If x is covered for the first time by S_i, then

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

Each step of the algorithm assigns 1 unit of cost, and so

$$|C| = \sum_{x \in X} c_x \qquad (1)$$

Set Covering Problem

Theorem 4: GREEDY-SET-COVER is a polynomial-time $\rho(n)$-approximation algorithm, where $\rho(n) = H(\max\{|S|: S \in F\})$, $H(d) = \sum_{i=1}^{d} \frac{1}{i}$ and $H(0) = 0$.

Proof: … … …

Each $x \in X$ is in at least one set in the optimal cover C^*, and so

$$\sum_{S \in C^*} \sum_{x \in S} c_x \geq \sum_{x \in X} c_x \qquad (2)$$

Combining (1) and (2) we have

$$|C| \leq \sum_{S \in C^*} \sum_{x \in S} c_x \qquad (3)$$

Set Covering Problem

Theorem 4: GREEDY-SET-COVER is a polynomial-time $\rho(n)$-approximation algorithm, where $\rho(n) = H(\max\{|S|: S \in F\})$, $H(d) = \sum_{i=1}^{d} \frac{1}{i}$ and $H(0) = 0$.

Proof: … … …

The remainder of the proof rests on the following key inequality, which we will prove shortly. For any set S belonging to the family F,

$$\sum_{x \in S} c_x \leq H(|S|) \qquad (4)$$

From (3) and (4) we get the following which proves the theorem.

$$|C| \leq \sum_{S \in C^*} H(|S|) \leq |C^*| \cdot H(\max\{|S|: S \in F\})$$

Set Covering Problem

Theorem 4: GREEDY-SET-COVER is a polynomial-time $\rho(n)$-approximation algorithm, where $\rho(n) = H(\max\{|S|: S \in F\})$, $H(d) = \sum_{i=1}^{d} \frac{1}{i}$ and $H(0) = 0$.

Proof: … … …

All that remains is to prove inequality (4). Consider any set $S \in F$ and any $i = 1,2, \ldots, |C|$, and let

$$u_i = |S - (S_1 \cup S_2 \cup \cdots \cup S_i)|$$

be the number of elements in S that remain uncovered after the algorithm has selected sets $S_1, S_2, \ldots, S_i$.

We define $u_0 = |S|$ to be the number of elements of S, which are all initially uncovered.

Set Covering Problem

Theorem 4: GREEDY-SET-COVER is a polynomial-time $\rho(n)$-approximation algorithm, where $\rho(n) = H(\max\{|S|: S \in F\})$, $H(d) = \sum_{i=1}^{d} \frac{1}{i}$ and $H(0) = 0$.

Proof: … … …

Let k be the least index such that $u_k = 0$, so that every element in S is covered by at least one of the sets $S_1, S_2, \ldots, S_k$ and some element in S is uncovered by $S_1 \cup S_2 \cup \cdots \cup S_{k-1}$. 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, for $i = 1,2, \ldots, k$. Thus,

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

Set Covering Problem

Theorem 4: GREEDY-SET-COVER is a polynomial-time $\rho(n)$-approximation algorithm, where $\rho(n) = H(\max\{|S|: S \in F\})$, $H(d) = \sum_{i=1}^{d} \frac{1}{i}$ and $H(0) = 0$.

Proof: … … …

Observe that

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

because the greedy choice of S_i guarantees that S cannot cover more new elements than S_i does (otherwise, the algorithm would have chosen S instead of S_i). Consequently, we obtain

$$\sum_{x \in S} c_x \leq \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot \frac{1}{u_{i-1}}.$$

Set Covering Problem

Theorem 4: GREEDY-SET-COVER is a polynomial-time $\rho(n)$-approximation algorithm, where $\rho(n) = H(\max\{|S|: S \in F\})$, $H(d) = \sum_{i=1}^{d} \frac{1}{i}$ and $H(0) = 0$.

Proof: … … …

$$\sum_{x \in S} c_x \leq \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot \frac{1}{u_{i-1}} = \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{u_{i-1}} \leq \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{j}$$

$$= \sum_{i=1}^{k} \left(\sum_{j=1}^{u_{i-1}} \frac{1}{j} - \sum_{j=1}^{u_i} \frac{1}{j} \right) = \sum_{i=1}^{k} \big(H(u_{i-1}) - H(u_i) \big) = H(u_0) - H(u_k)$$

$$= H(u_0) - H(0) = H(u_0) = H(|S|),$$

which completes the proof of inequality (4).

Set Covering Problem

Corollary 4: GREEDY-SET-COVER is a polynomial-time $(\ln|X| + 1)$-approximation algorithm.

Proof: Since we know $\sum_{i=1}^{n} \frac{1}{k} \leq \ln n + 1$, corollary 4 directly follows from Theorem 4.

Subset Sum Problem

An instance (S, t) of the *subset sum problem* consists of a set $S = \{x_1, x_2, \ldots, x_n\}$ of positive integers, and a positive integer t. This decision problem asks whether there exists a subset of S that adds up exactly to the target value t. This problem is NP-complete.

The optimization problem associated with this decision problem asks you to find a subset of $S = \{x_1, x_2, \ldots, x_n\}$ whose sum is as large as possible but not larger than t.

Subset Sum: An Exponential-Time Exact Algorithm

Input: A set $S = \{x_1, x_2, \ldots, x_n\}$ of positive integers, and a positive integer t.

Output: Sum of the elements of a subset of S whose elements sum up to the largest value not exceeding t.

```
EXACT-SUBSET-SUM ( S, t )
1.    n ← |S|
2.    L_0 ← ⟨0⟩
3.    for i ← 1 to n do
4.        L_i ← MERGE-LISTS ( L_{i−1}, L_{i−1} + x_i )
5.        remove from L_i every element that is greater than t
6.    return the largest element in L_n
```

If L is a list of positive integers and x is another positive integer, then we let $L + x$ denote the list of integers derived from L by increasing each element of L by x. For example, if $L = \{1,2,3,5,9\}$, then $L + 2 = \{3,4,5,7,11\}$.

$\text{MERGE}-\text{LISTS}(L, L')$ returns the sorted list by merging its two sorted input lists L and L' with duplicates removed. It runs in time $O(|L| + |L'|)$.

Subset Sum:
A Fully Polynomial-Time Approximation Scheme

Input: A list $L = \langle y_1, y_2, \ldots, y_m \rangle$ of numbers sorted into monotonically increasing order, and a trimming parameter δ with $0 < \delta < 1$.

Output: A list L' obtained by removing as many elements from L as possible such that for every element y that is removed from L there is an item z still in L' satisfying $\frac{y}{1+\delta} \leq z \leq y$.

```
TRIM ( L, δ )
1.    m ← |L|
2.    L' ← ⟨y_1⟩
3.    last ← y_1
4.    for i ← 2 to m do
5.        if y_i > last · (1 + δ) then
6.            append y_i onto the end of L'
7.            last ← y_i
8.    return L'
```

Example: If $\delta = 0.1$ and $L = \langle 10,11,12,15,20,21,22,23,24,29 \rangle$ then $L' = \langle 10,12,15,20,23,29 \rangle$.

Subset Sum:
A Fully Polynomial-Time Approximation Scheme

Input: A set $S = \{x_1, x_2, \ldots, x_n\}$ of n integers (in arbitrary order), a target integer t, and an "approximation parameter" ϵ, where $0 < \epsilon < 1$.

Output: It returns a subset sum whose value is within a $1 + \epsilon$ factor of the optimal.

APPROX-SUBSET-SUM (S, t, ϵ)

1. $n \leftarrow |S|$
2. $L_0 \leftarrow \langle 0 \rangle$
3. *for* $i \leftarrow 1$ *to* n *do*
4. $\quad L_i \leftarrow$ *MERGE-LISTS* ($L_{i-1}, L_{i-1} + x_i$)
5. $\quad L_i \leftarrow$ *TRIM* ($L_i, \frac{\epsilon}{2n}$)
6. $\quad$ *remove from* L_i *every element that is greater than* t
7. *let* z^* *be the largest value in* L_n
8. *return* z^*

Subset Sum:
A Fully Polynomial-Time Approximation Scheme

Example Instance:

$S = \{104, 102, 201, 101\}$,

$t = 308$, $\epsilon = 0.4$, and $\delta = \frac{\epsilon}{2n} = \frac{0.4}{8}$.

Execution:

line 2: $L_0 = \langle 0 \rangle$,

line 4: $L_1 = \langle 0, 104 \rangle$,
line 5: $L_1 = \langle 0, 104 \rangle$,
line 6: $L_1 = \langle 0, 104 \rangle$,

line 4: $L_2 = \langle 0, 102, 104, 206 \rangle$,
line 5: $L_2 = \langle 0, 102, 206 \rangle$,
line 6: $L_2 = \langle 0, 102, 206 \rangle$,

line 4: $L_3 = \langle 0, 102, 201, 206, 303, 407 \rangle$,
line 5: $L_3 = \langle 0, 102, 201, 303, 407 \rangle$,
line 6: $L_3 = \langle 0, 102, 201, 303 \rangle$,

line 4: $L_4 = \langle 0, 101, 102, 201, 203, 302, 303, 404 \rangle$,
line 5: $L_4 = \langle 0, 101, 201, 302, 404 \rangle$,
line 6: $L_4 = \langle 0, 101, 201, 302 \rangle$.

APPROX-SUBSET-SUM (S, t, ϵ)

1. $n \leftarrow |S|$
2. $L_0 \leftarrow \langle 0 \rangle$
3. *for* $i \leftarrow 1$ *to* n *do*
4. $\quad L_i \leftarrow$ *MERGE-LISTS* ($L_{i-1}, L_{i-1} + x_i$)
5. $\quad L_i \leftarrow$ *TRIM* ($L_i, \frac{\epsilon}{2n}$)
6. $\quad$ *remove from* L_i *every element that is greater than* t
7. *let* z^* *be the largest value in* L_n
8. *return* z^*

The algorithm returns $z^* = 302$ as the answer which within $\epsilon = 40\%$ of the optimal

$$307 = 104 + 102 + 101;$$

in fact, it is within 2%.

Subset Sum

Theorem 4: APPROX-SUBSET-SUM is a fully polynomial-time approximation scheme for the subset sum problem.

Proof: Let P_i denote the set of all values obtained by selecting a (possibly empty) subset of $\{x_1, x_2, \ldots, x_i\}$ and summing its members.

The operations of trimming L_i in line 5 and removing from L_i every element that is greater than t maintain the property that every element of L_i is also a member of P_i . Therefore, the value z^* returned in line 8 is indeed the sum of some subset of S.

Let $y^* \in P_n$ denote an optimal solution to the subset-sum problem. Then, from line 6, we know that $z^* \leq y^*$. Now we need to show:

(i) $\frac{y^*}{z^*} \leq 1 + \epsilon$, and

(ii) running time of this algorithm is polynomial in both $\frac{1}{\epsilon}$ and the size of the input.

Subset Sum

Theorem 4: APPROX-SUBSET-SUM is a fully polynomial-time approximation scheme for the subset sum problem.

Proof: … … …

One can show that (see Exercise 35.5-2 of CLRS), for every element y in P_i that is at most t, there exists an element $z \in L_i$ such that

$$\frac{y}{\left(1+\frac{\epsilon}{2n}\right)^i} \leq z \leq y \qquad (1)$$

Inequality (1) must hold for $y^* \in P_n$, and therefore there exists an element $z \in L_n$ such that

$$\frac{y^*}{\left(1+\frac{\epsilon}{2n}\right)^n} \leq z \leq y^* \qquad (2)$$

Subset Sum

Theorem 4: APPROX-SUBSET-SUM is a fully polynomial-time approximation scheme for the subset sum problem.

Proof: … … …

Thus

$$\frac{y^*}{z} \leq \left(1 + \frac{\epsilon}{2n}\right)^n \tag{3}$$

Since there exists an element $z \in L_n$ fulfilling inequality (3), the inequality must hold for z^*, which is the largest value in L_n; that is,

$$\frac{y^*}{z^*} \leq \left(1 + \frac{\epsilon}{2n}\right)^n \tag{4}$$

But

$$\lim_{n\to\infty} \left(1 + \frac{\epsilon}{2n}\right)^n = e^{\frac{\epsilon}{2}} \tag{5}$$

Subset Sum

Theorem 4: APPROX-SUBSET-SUM is a fully polynomial-time approximation scheme for the subset sum problem.

Proof: … … …

One can also show that (see Exercise 35.5-3 of CLRS)

$$\frac{d}{dn}\left(1+\frac{\epsilon}{2n}\right)^{n} > 0 \qquad (6)$$

Therefore, we have

$$\left(1+\frac{\epsilon}{2n}\right)^{n} \leq e^{\frac{\epsilon}{2}} \leq 1+\frac{\epsilon}{2}+\left(\frac{\epsilon}{2}\right)^{2} < 1+\epsilon$$

Hence, combining with inequality (4), we get

$$\frac{y^{*}}{z^{*}} \leq \left(1+\frac{\epsilon}{2n}\right)^{n} < 1+\epsilon$$

Subset Sum

Theorem 4: APPROX-SUBSET-SUM is a fully polynomial-time approximation scheme for the subset sum problem.

Proof: … … …

To show that APPROX-SUBSET-SUM is a fully polynomial-time approximation scheme, we derive a bound on the length of L_i. After trimming, successive elements z' and z of L_i must have the relationship $\frac{z'}{z} > 1 + \frac{\epsilon}{2n}$. Each list, therefore, contains the value 0, possibly the value 1, and up to $\left\lfloor \log_{1+\frac{\epsilon}{2n}} t \right\rfloor$ additional values. The number of elements in each list L_i is at most

$$2 + \log_{1+\epsilon/2n} t = 2 + \frac{\ln t}{\ln\left(1 + \frac{\epsilon}{2n}\right)} \leq 2 + \frac{2n\left(1 + \frac{\epsilon}{2n}\right)\ln t}{\epsilon} < \frac{3n \ln t}{\epsilon} + 2$$

Subset Sum

Theorem 4: APPROX-SUBSET-SUM is a fully polynomial-time approximation scheme for the subset sum problem.

Proof: … … …

The bound is polynomial in the size of the input—which is the number of bits $\lg t$ needed to represent t plus the number of bits needed to represent the set S, which is in turn polynomial in n — and in $\frac{1}{\epsilon}$. Since the running time of APPROX-SUBSET-SUM is polynomial in the lengths of the L_i , we conclude that APPROX-SUBSET-SUM is a fully polynomial-time approximation scheme.