Approximation algorithms

#2

Map for today

- By the end of today you will be able to:
  - Explain how to get a log n approximation for set cover
  - Explain how to get a 2 approximation for weighted vertex cover with rounding of ILP
  - Define the terms PTAS and FPTAS
  - Explain the FPTAS of subset sum
  - Improve the bound on set cover

Set Cover

- *Definition*: Given a finite set X and subsets of X, find the minimum number of these subsets whose union is X.

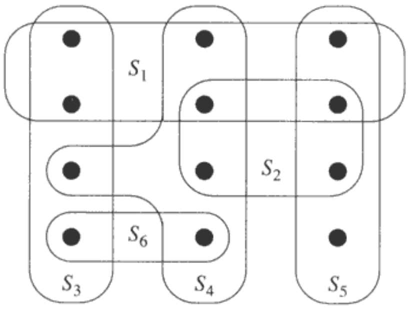
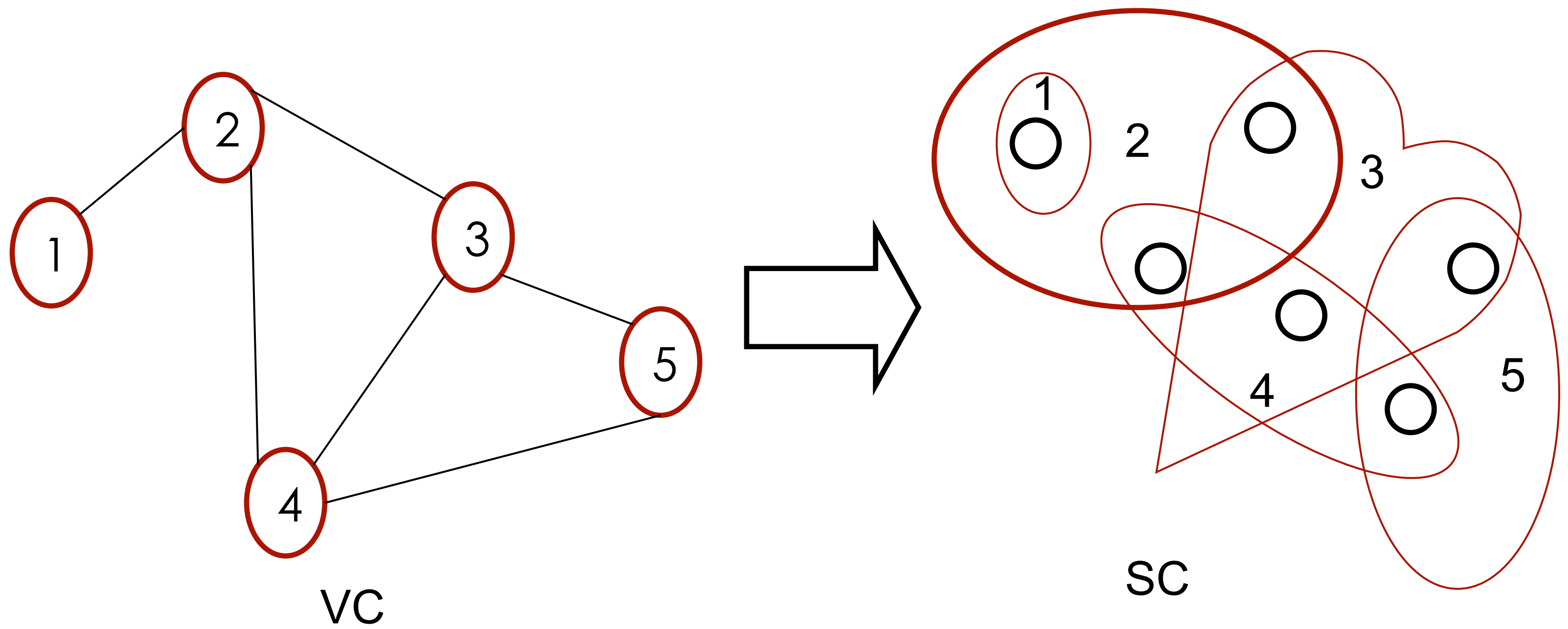


Figure 35.3 An instance $(X, \mathcal{F})$ of the set-covering problem, where X consists of the 12 black points and $\mathcal{F} = \{S_1, S_2, S_3, S_4, S_5, S_6\}$. A minimum-size set cover is $\mathcal{C} = \{S_3, S_4, S_5\}$. The greedy algorithm produces a cover of size 4 by selecting the sets S_1, S_4, S_5, and S_3 in order.

Vertex cover -> Set cover



Greedy-set-cover(X,F)

$C \leftarrow \phi$
$U \leftarrow X$
while $U \neq \phi$ **do**
    select $S \in F$ that maximizes $|S \cap U|$ O(|F|·|X|)
    $C \leftarrow C \cup \{S\}$
    $U \leftarrow U - S$
return C

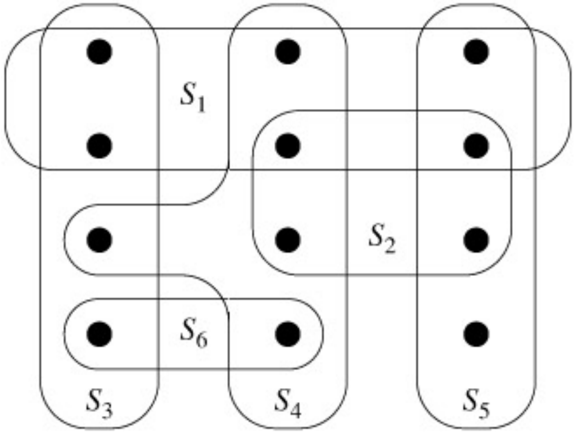
min{|X|,|F|}

Question: What is the running time of the algorithm?

Answer: min{|X|,|F|}*O(|F|·|X|)

Our plan

- Clearly polynomial time
- We will show that the Approximation ratio is O(log n)



Observation

- Let k = OPT, E_t be the set of elements not yet covered after step t, ($E_0 = E$) .
- E_t can be covered with no more than k sets.
- Greedy-Set-Cover always picks the largest set of E_t in step $t + 1$.

Question: Why is that?

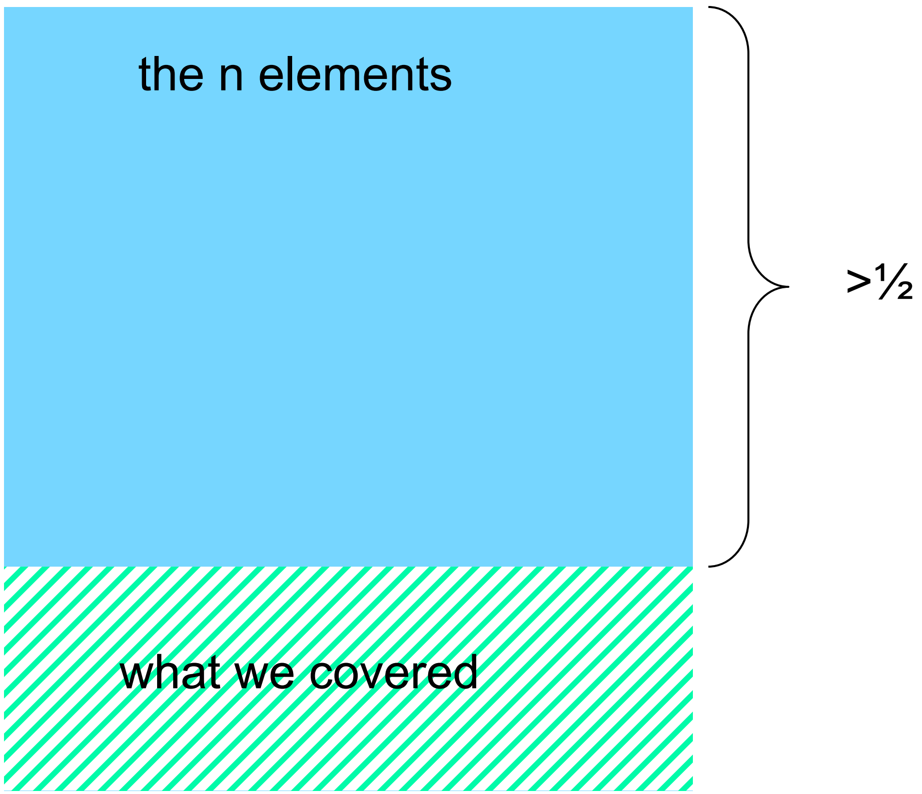
Number example

- Say we had 100 points that could be covered by 10 subsets
- **Question**: After selecting 10 subsets using the algorithm, how many elements (at most) are not covered?

After k iterations

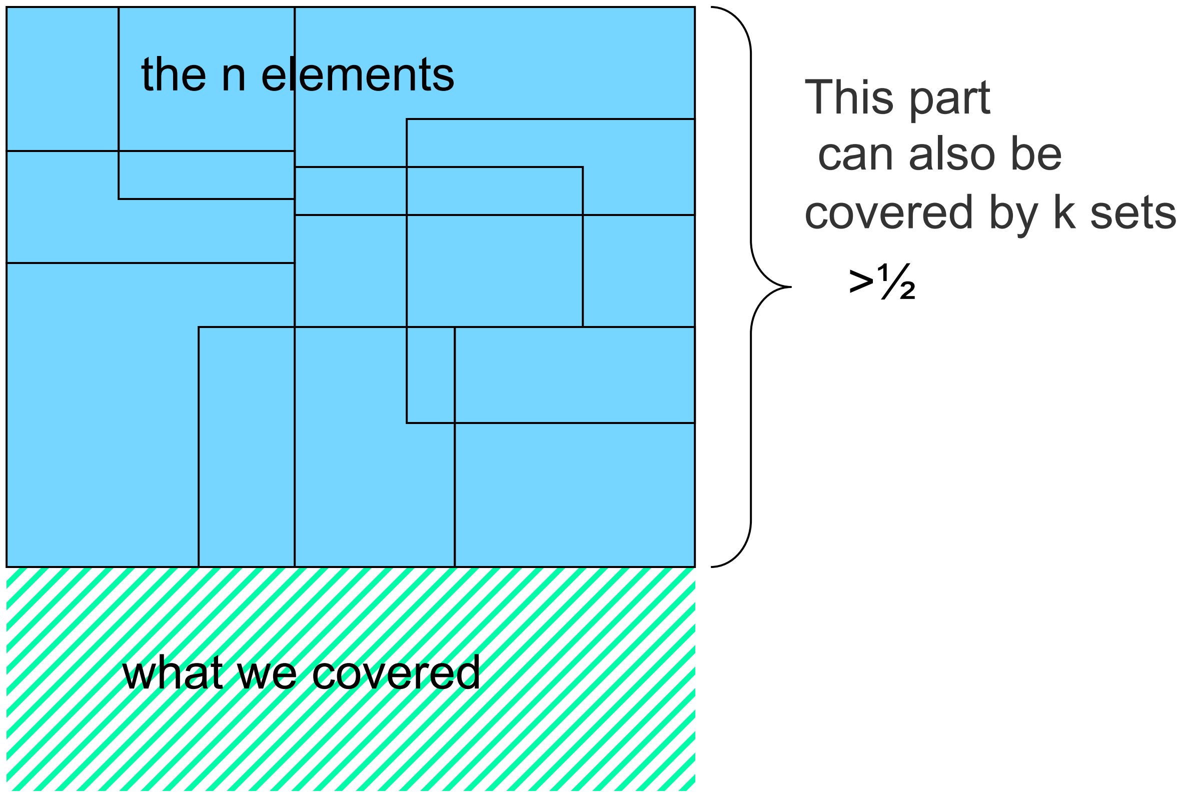
Claim: after k iterations the algorithm covered at least ½ of the elements.

Suppose it doesn't and observe the situation after k iterations:



After k iterations

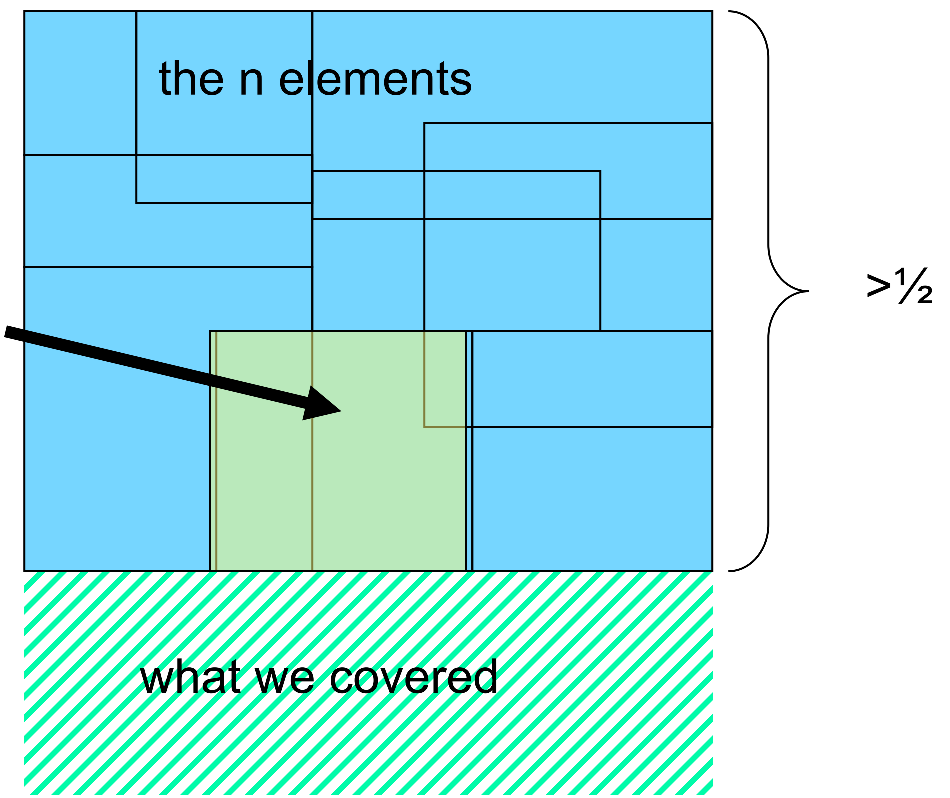
Claim: after k iterations the algorithm covered at least ½ of the elements.



After k iterations

Claim: after k iterations the algorithm covered at least ½ of the elements.

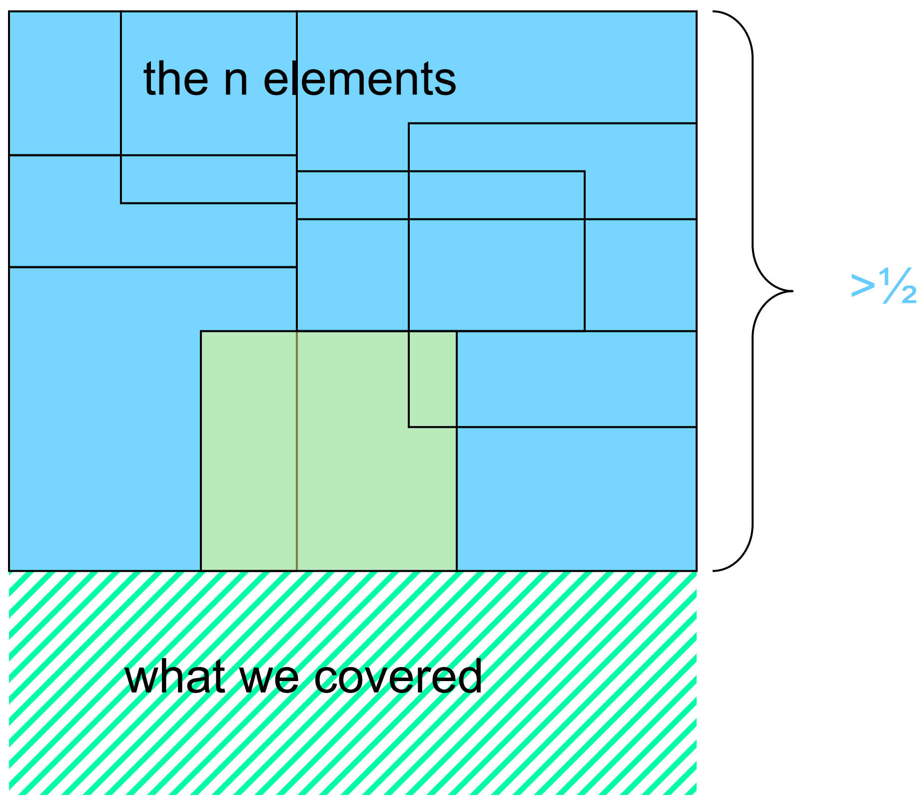
there must be a set not chosen yet, whose size is at least ½n·1/k



After k iterations

Claim: after k iterations the algorithm covered at least ½ of the elements.

and the claim is proven!



Thus in each of the k iterations we've covered at least ½n·1/k new elements, contradiction.

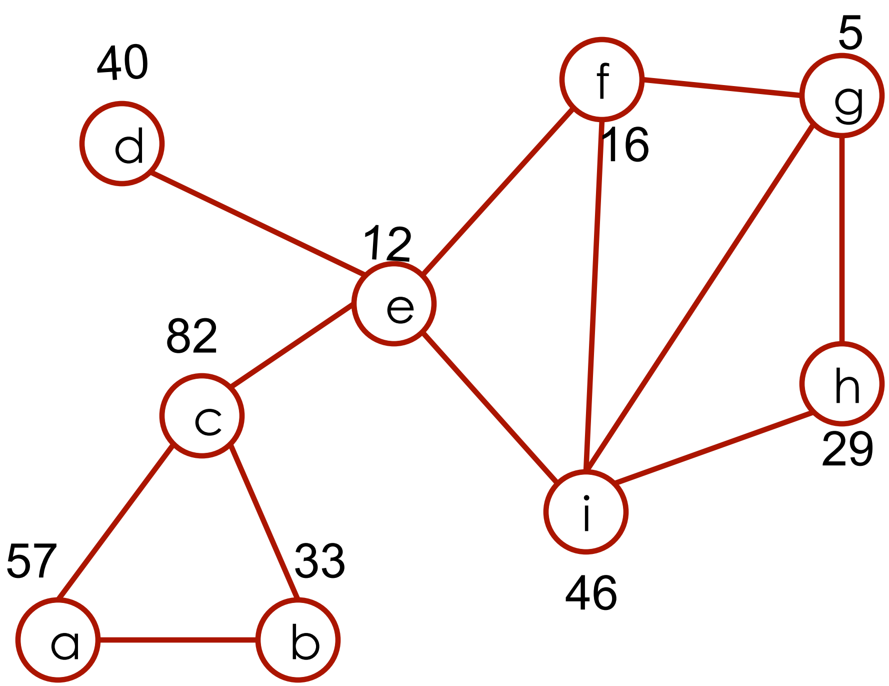
Conclusion of the claim

- Remember - for every subset of the elements , k is still a size of a set cover
- after k iterations the algorithm covered at least ½ of the elements.
- **Question**: How can we now show that the bound is log(n)?
- After **k logn** iterations all the n elements must be covered
- i.e, choosing k logn sets vs. k in the optimum
- The log n bound is guaranteed ■

Weighted vertex cover

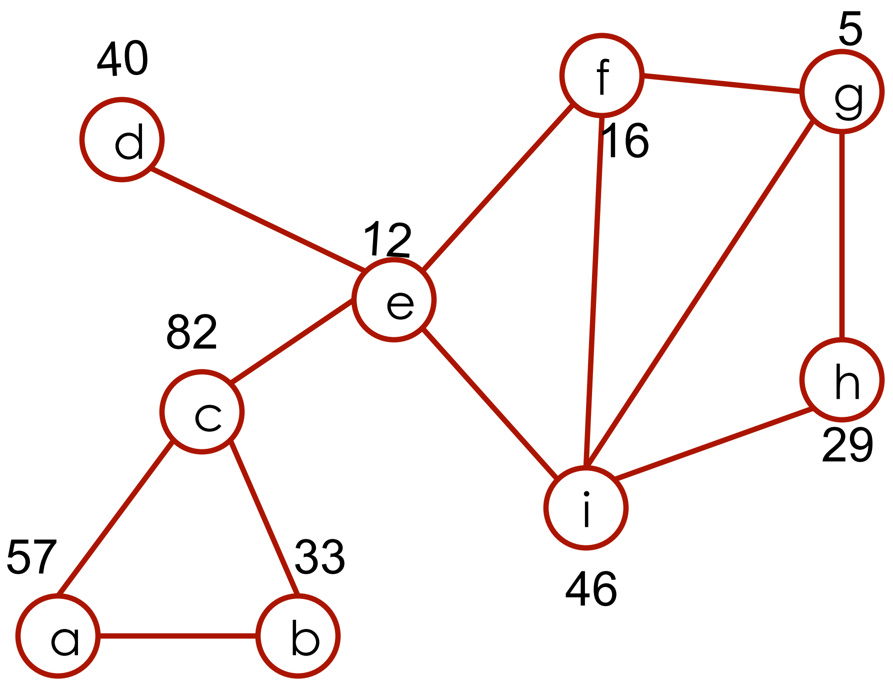
- Still looking for a vertex cover
- Minimizing the sum of the vertices(the graph is now weighted)
- problem is NP-hard

Question: why?



A factor 2 approx. algorithm

- Previous approx. Algorithm doesn't work
- Express as a 0-1 integer program



LP for min-weighted-vertex-cover

$$
\begin{array}{llll}
\text{OPT-VC} = & \text{Minimize} & \sum_{v \in V} w(v) X(v) & \\
\text{subject to} & & X(u) + X(v) \geq 1 & \forall e = (u,v) \in E \\
\text{where} & & X(u) \in \{0,1\} & \forall u \in V
\end{array}
$$

- for every vertex v: x(v)=1 or 0,
- 1 – v is in the V.C , otherwise 0
- Forcing one the endpoints of every edge to be in the V.C

LP for min-weighted-vertex-cover

$$
\begin{array}{rccc}
\text{OPT-VC} \ = & \text{Minimize} & \sum_{v \in V} w(v) X(v) & \\
\text{subject to} & & X(u) + X(v) \geq 1 & \forall e = (u,v) \in E \\
\text{where} & & X(u) \in \{0,1\} & \forall u \in V
\end{array}
$$

- Still NP-hard
- Convert to a ***linear*** program
  - relax the {0,1} constraint. I.E

$0 \leq x(v) \geq 1$

The 2-approximation algorithm

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

$C \leftarrow \emptyset$
compute $\bar{x}$, an optimal solution to the linear program in lines (35.15)–(35.18)
for each $v \in V$
  do if $\bar{x}(v) \geq 1/2$
    then $C \leftarrow C \cup \{v\}$
return C

- **Question**: Which two things should I prove now?
- LP is solved in polynomial time
- The rest of the algorithm runs in linear time
- The value of an optimal solution to this LP ≤ the optimal solution for the 0-1 program
  - We only **relaxed** a constraint on a **minimzation** problem

Terminology

- Z*- the value returned by the relaxed LP
- C – The solution outputted by APPROX-MIN-WEIGHT-VC(G,w)
- W(C)- the value of APPROX-MIN-WEIGHT-VC(G,w)
- C* - an optimal solution to the problem (the subset of vertices)
- W(C*)-the value of the optimal solution
- Remember: $W(C) \geq W(C^*)$

APPROX-MIN-WEIGHT-VC(G,w) is a 2-approximation

- WE need to prove that : 1.C is a vertex cover, and 2. $W(C) \leq 2W(C^*)$
- **Question**: how do I show that C is a vertex-cover?
- 1. We include in the vertex cover each vertex v for which $x(v) \geq 0.5$
  - Each edge $e = (u,v) \in E$ $X(u)+X(v) \geq 1$
    → either X(u) or X(v) is at least 0.5
  - Thus the solution returned is a vertex cover
    - it may still be a bit more costly than the optimum

Z^*- value of the relaxed LP C –set returned by APPROX-MIN-WEIGHT-VC W(C)- the value of APPROX-MIN-WEIGHT-VC(G,w) C^* - optimal set $W(C^*)$-the value of the optimal set

APPROX-MIN-WEIGHT-VC(G,w) is a 2-approximation (cont.)

2. We first note $z^* \leq W(C^*)$
   - Since the optimal solution is a feasible solution for the relaxed LP
   - All we need to prove: $W(C) \leq 2W(C^*)$
     - In other words: $2z^* \geq W(C)$
   - *Each $v \in C$ contributes to C at most twice as much as it did for z^**

   $\rightarrow W(C) \leq 2z^* \leq 2W(C^*)$

Z^*- value of the relaxed LP C –set returned by APPROX-MIN-WEIGHT-VC $W(C)$- the value of APPROX-MIN-WEIGHT-VC(G,w) C^* - optimal set $W(C^*)$-the value of the optimal set

PTAS and FPTAS

- A problem L has a **polynomial-time approximation scheme (PTAS)** if for any fixed $\varepsilon > 0$, L has a polynomial-time $(1+\varepsilon)$-approximation algorithm
- The idea of an approximation scheme is to be able to get better and better approximation ratios by expending more computation time

PTAS and FPTAS

- The running time of a PTAS is required to be polynomial in *n* for every fixed ε but can be different for different ε .
- algorithm running in time

  $O(n^{1/\varepsilon})$ or even $O(n^{\exp(1/\varepsilon)})$ counts as a PTAS.
- L has a **full polynomial-time approximation scheme (FPTAS)** if it has a PTAS that runs in time polynomial both in $(1/\varepsilon)$ *and* in the size of the input.

Question

- So what is different from all the algorithms we have seen so far and PTAS\FPTAS algorithms?

- Answer: In all of the algorithms we have seen, you can not choose the precision of the answer. Wev'e seen 2-approximation (V.C, Weighted V.C Euclidian TSP) and log-n approximation (subset-sum)

The Subset Sum Problem

- Problem definition
  - Input: a finite set S and a target t
  - find a subset $S' \subseteq S$ whose elements sum to t
- All possible sums
  - $S = \{x_1, x_2, .., x_n\}$
  - L_i = set of all possible sums of $\{x_1, x_2, .., x_i\}$
- Example
  - $S = \{1, 4, 5\}$
  - $L_1 = \{0, 1\}$
  - $L_2 = \{0, 1, 4, 5\} = L_1 \cup (L_1 + x_2)$
  - $L_3 = \{0, 1, 4, 5, 6, 9, 10\} = L_2 \cup (L_2 + x_3)$
- $L_i = L_{i-1} \cup (L_{i-1}+x_i)$

Subset Sum-exp. algorithm

- Given a finite set S and a target t, find a subset S' ⊆ S whose elements sum to t

x + T adds x to each element in the set T

```
T = {0};
for each x in S {
    T = union(T, x+T);
    remove elements from T that exceed t;
}
return largest element in T;
```

Potential doubling at each step

Complexity $O(2^n)$

- **Question**: What is the complexity of this algorithm?

Trimming:

- To reduce the size of the set T at each stage, we apply a trimming process.
- For example, if z and y are consecutive elements and $(1-\delta)y \le z < y$, then remove y.
- If $\delta=0.1$, $\{10,11,12,15,20,21,22,23,24,29\}$

  $\Rightarrow \{10, \quad 12,15,20, \quad\quad 23, \quad 29\}$

Subset Sum with Trimming:

- Incorporate trimming in the previous algorithm:

```
T = {0};
for each x in S {
     T = union(T, x+T);
     T = trim(δ, T);
     remove elements from T that exceed t;
}
return largest element in T;
```

$0 \leq \delta \leq 1/n$

- Trimming only eliminates values, it doesn't create new ones.
- The final result is still the sum of a subset of S that is less than t.

Subset Sum – Trim

- Reduce the size of a list by "trimming"
  - L: An original list
  - L': The list after trimming L
  - δ: trimming parameter, [0..1]
  - y: an element that is removed from L
  - z: corresponding (representing) element in L' (also in L)
  - $(y-z)/y \leq \delta$
  - $(1-\delta)y \leq z \leq y$

Trim

- Example
  - L = {10, 11, 12, 15, 20, 21, 22, 23, 24, 29}
  - $\delta = 0.1$
  - L' = {10, 12, 15, 20, 23, 29}
  - 11 is represented by 10. $(11-10)/11 \leq 0.1$
  - 21, 22 are represented by 20. $(21-20)/21 \leq 0.1$
  - 24 is represented by 23. $(24-23)/24 \leq 0.1$

Trim – the code

- Trim(L, δ)   // L: $y_1, y_2, .., y_m$

1. L' = $\{y_1\}$
2. last = y_1 // most recent element z in L' which represent elements in L
3. for i = 2 to m do
4.   if last < $(1-\delta)\ y_i$ then   // $(1-\delta)y \le z \le y$
5.   // y_i is appended into L' when it cannot be represented by last
6.   append y_i onto the end of L'
7.   last = y_i
8. return L'

L = {10, 11, 12, 15, 20, 21, 22, 23, 24, 29} δ = 0.1,

L' = {10, 12, 15, 20, 23, 29}

Subset Sum – Approximate Algorithm

- **Approx_subset_sum(S, t, e)** // $S=x_1,x_2,\dots,x_n$
  1. $L_0 = \{0\}$
  2. for i = 1 to n do
  3. $L_i = L_{i-1} \cup (L_{i-1}+x_i)$
  4. $L_i = \text{Trim}(L_i, \varepsilon/n)$
  5. Remove elements that are greater than t from L_i
  6. return the largest element in L_n

Example:

Input:

L = {104, 102, 201, 101}, t=308, ε=0.20, $\delta = \varepsilon/n$=0.05

Question: What is returned? What is the optimal?

$L_0 = \{0\}$

$L_1 = \{0, 104\}$

$L_2 = \{0, 102, 104, 206\}$

After trimming 104: $L_2 = \{0, 102, 206\}$

$L_3 = \{0, 102, 201, 206, 303, 407\}$

After trimming 206: $L_3 = \{0, 102, 201, 303, 407\}$

After removing 407: $L_3 = \{0, 102, 201, 303\}$

$L_4 = \{0, 101, 102, 201, 203, 302, 303, 404\}$

After trimming 102, 203, 303: $L_4 = \{0, 101, 201, 302, 404\}$

After removing 404: $L_4 = \{0, 101, 201, 302\}$

Return 302 (=201+101)

Optimal answer is 104+102+101=307

Choosing δ

- δ needs to be:
  - small enough to compensate for *n* accumulating errors
  - large enough so that $(1/\delta)$ is polynomial in (n/ε).
- An appropriate value: $\delta=\varepsilon/n$

Approx_subset_sum(S, t, e) is an FPTAS

- We now prove the following 2 claims:
  1. $C^*(1-\varepsilon) \leq C$
  2. The approximation algorithm is fully polynomial

Approximation ratio correctness - Intuition

- At each stage, **values in the trimmed** T are within a factor somewhere between $(1-\delta)$ and 1 of the corresponding values in the untrimmed T.

- By induction, the **final result (after n iterations)** is within a factor somewhere between $(1-\delta)^n$ and 1 of the result produced by the original algorithm.

$C^*(1-\varepsilon) \le C$

- Proof
  - $\forall y \in L \ \exists z \in L'$ such that:

    $$(1-\varepsilon/n)y \le z \le y$$

  - $\forall y \in L_i \ \exists z \in L'_i$ such that

    $$(1-\varepsilon/n)^i y \le z \le y$$

  - If y^* is an optimal solution in L_n then there is a corresponding z in L_n'

    $$(1-\varepsilon/n)^n y^* \le z \le y^*$$

$\rightarrow 1\backslash e$ When $n \rightarrow \infty$

S = {x1,...,xn} - a set of n integer positive numbers **t**- target number δ: trimming parameter, [0..1]
L: An original list L': L after trimming **y**: element removed from L **z**: representing element y in L' (also in L) L_i - the sorted list of *all sums* of *all subsets* of $\{x_1, x_2, \ldots, x_i\}$ that do not exceed the target value *t*. δ trimming factor

$C^*(1-\varepsilon) \le C$ Proof (cont.)

- If y^* is an optimal solution in L_n then there is a corresponding z in L_n'

$$(1-\varepsilon/n)^n y^* \le z \le y^*$$

- $(1-\varepsilon/n)^n$ is increasing and therefore:
  - $(1-\varepsilon) < (1-\varepsilon/n)^n$
  - $(1-\varepsilon)\, y^* \le (1-\varepsilon/n)^n y^* \le z$
    - $(1-\varepsilon)\, y^* \le z$
  - So the value z returned is not smaller than $1-\varepsilon$ times the optimal solution y^*

S = {x1,…,xn} - a set of n integer positive numbers **t**- target number δ: trimming parameter, [0..1]
L: An original list L': L after trimming y: element removed from L **z**: representing element y in L' (also in L) L_i - the sorted list of *all sums* of *all subsets* of $\{x_1, x_2, \ldots, x_i\}$ that do not exceed the target value *t*. δ trimming factor

Running in polynomial time- intuition

- Running time of the i'th iteration - O(|Li|).
- $x_{i,}\ x_{i+1} \in T$ successive elements
- $0 \leq x_{i,}\ x_{i+1} \leq t$ and $x_{i+1} / x_i \geq (1\text{-}\delta)$
- The maximum number of elements in T is:

$$\log_{(1/(1-\delta))} t \leq (\log t / \delta).$$

Question: Why is that?

Example: elements in T is at least 2, and all of the values 0-1024:
0,1,2,4,8,16,32,64,128,256,512,1024

S = {x1,...,xn} - a set of n integer positive numbers **t**- target number δ: trimming parameter, [0..1]
L: An original list L': L after trimming y: element removed from L **z**: representing element y in L' (also in L) **L_i** - the sorted list of *all sums* of *all subsets* of $\{x_1, x_2, \ldots, x_i\}$ that do not exceed the target value *t*. δ trimming factor

The approximation algorithm is fully polynomial

- Proof
  - Successive elements z and z' in L_i' must maintain:

    $z/z' = 1/(1-\varepsilon/n)$

    i.e, they differ by a factor of at least $1/(1-\varepsilon/n)$
  - $|L_i|$ is at most

    $\log_{1/(1-\varepsilon/n)} t$

    $= \ln t / (-\ln(1-\varepsilon/n))$

    Change of base

    $\le (\ln t) / (-(-\varepsilon/n))$

    since $x/(1+x) \le \ln(1+x) \le x$, for $x > -1$, $x = -\varepsilon/n$

    $\le (n \ln t) / \varepsilon$
  - $|L_i|$ is polynomial, and so is the running time

S = {x1,...,xn} - a set of n integer positive numbers **t**- target number δ: trimming parameter, [0..1]
L: An original list L': L after trimming y: element removed from L **z**: representing element y in L' (also in L) L_i - the sorted list of *all sums* of *all subsets* of $\{x_1, x_2, \ldots, x_i\}$ that do not exceed the target value *t.* δ trimming factor

Improving the analysis for set cover

- Currently: $\rho(n) \leq \log_2 n$
- More careful analysis yields approximation ratio no larger than:

$$\ln(|X|)+1$$

- More precisely, not greater than $H(|S|)$, where S is the largest of the subsets of X, and H(i) is the harmonic sum:

$$H(i) = 1 + \tfrac{1}{2} + 1/3 + \tfrac{1}{4} + \ldots + 1/I$$

- We turn to prove the tight ratio-bound

Tight Ratio-Bound

Claim: The greedy algorithm approximates the optimal set-cover within factor

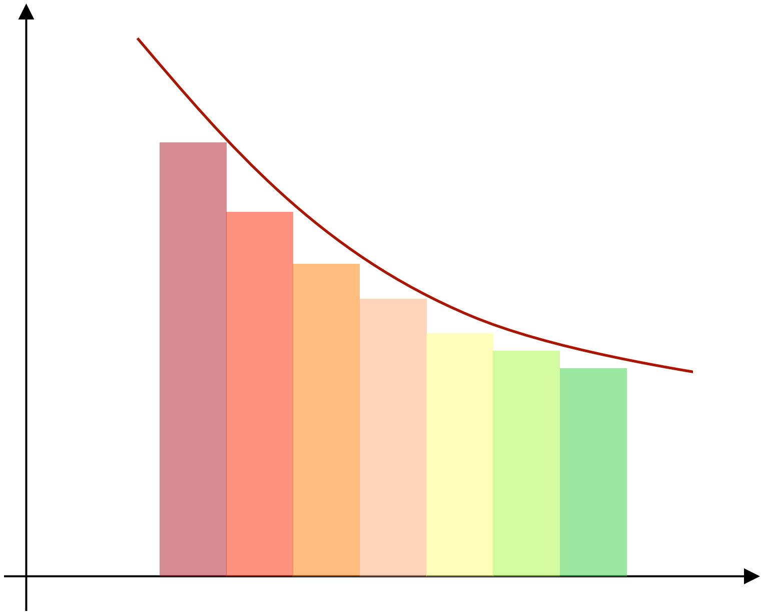
$$H(\max\{\ |S| : S \in F\ \})$$

Where H(d) is the d-th harmonic number:

$$H(d) \stackrel{def}{=} \sum_{i=1}^{d} \frac{1}{i}$$

H(d) illustrated

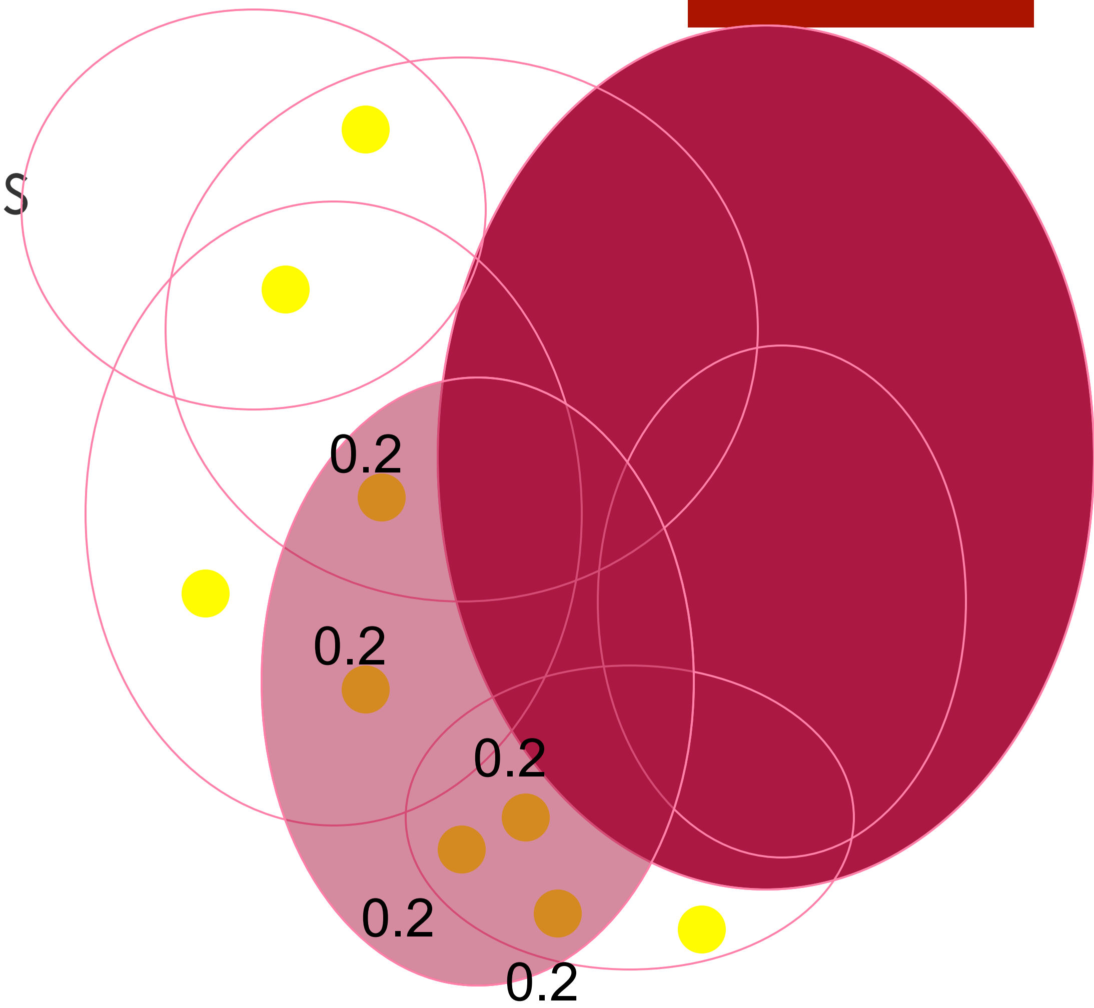
$$\sum_{k=1}^{n}\frac{1}{k} = \sum_{k=2}^{n}\frac{1}{k} + 1 \leq \int_{1}^{n}\frac{1}{x}dx + 1 = \ln(n) + 1$$



Claim's Proof

- Whenever the algorithm chooses a set, charge 1.

- Split the cost between all covered vertices.



Analysis

- That is, we charge every element $x \in X$ with

$$c_x \overset{\text{def}}{=} \frac{1}{|S_i - (S_1 \cup \ldots \cup S_{i-1})|}$$

- Where S_i is the first set which covers x.

Note

- Since at every selection we assign 1 unit of cost

$$|C| = \sum_{x \in X} c_x$$

- And since every element is in at least one set in C*

$$\sum_{s \in C^*} \sum_{x \in S} c_x \geq \sum_{x \in X} c_x$$

- And so

$$\sum_{s \in C^*} \sum_{x \in S} c_x \geq |C|$$

Our mission

$$\sum_{s \in C^*} \sum_{x \in S} c_x \geq |C|$$

Bound the contribution of each group by H(d)

Bounding to cost for every set

Lemma: For every $S \in F$,

$$\sum_{x \in S} c_x \le H(|S|)$$

Number of members of S left uncovered after i iterations

Proof: Fix an $S \in F$. For any i, Define

$$u_i \overset{def}{=} |S - (S_1 \cup \ldots \cup S_i)|$$

Let k be the smallest index, for which $u_k=0$.

$\forall 1 \le i \le k : S_i$ covers $u_{i-1}-u_i$ elements from S

Lemma

$u_0 = |S|$

$$\sum_{x \in S} c_x \le H(|S|)$$

■

Analysis

Now we can finally complete our analysis:

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

$\blacksquare$