

# Approximation Algorithms II

## Lecture 8

September 19, 2014

# Part I

## Max Exact 3SAT

# 3SAT revisited

- ① Instance of **3SAT** is a boolean formula.
- ② Example:  $F = (x_1 + x_2 + x_3)(x_4 + \overline{x_1} + x_2)$ .
- ③ Decision problem = is the formula has a satisfiable assignment.
- ④ Optimization version:

## Max 3SAT

**Instance:** A collection of clauses:  $C_1, \dots, C_m$ .

**Question:** Find the assignment to  $x_1, \dots, x_n$  that satisfies the maximum number of clauses.

- ⑤ Max 3SAT is **NP-Hard**
- ⑥ Max 3SAT is a *maximization problem*.

# 3SAT revisited

- ① Instance of **3SAT** is a boolean formula.
- ② Example:  $F = (x_1 + x_2 + x_3)(x_4 + \overline{x_1} + x_2)$ .
- ③ Decision problem = is the formula has a satisfiable assignment.
- ④ Optimization version:

## Max 3SAT

**Instance:** A collection of clauses:  $C_1, \dots, C_m$ .

**Question:** Find the assignment to  $x_1, \dots, x_n$  that satisfies the maximum number of clauses.

- ⑤ Max 3SAT is **NP-Hard**
- ⑥ Max 3SAT is a *maximization problem*.

# 3SAT revisited

- ① Instance of **3SAT** is a boolean formula.
- ② Example:  $F = (x_1 + x_2 + x_3)(x_4 + \overline{x_1} + x_2)$ .
- ③ Decision problem = is the formula has a satisfiable assignment.
- ④ Optimization version:

## Max 3SAT

**Instance:** A collection of clauses:  $C_1, \dots, C_m$ .

**Question:** Find the assignment to  $x_1, \dots, x_n$  that satisfies the maximum number of clauses.

- ⑤ **Max 3SAT** is **NP-Hard**
- ⑥ **Max 3SAT** is a *maximization problem*.

# 3SAT revisited

- ① Instance of **3SAT** is a boolean formula.
- ② Example:  $F = (x_1 + x_2 + x_3)(x_4 + \overline{x_1} + x_2)$ .
- ③ Decision problem = is the formula has a satisfiable assignment.
- ④ Optimization version:

## Max 3SAT

**Instance:** A collection of clauses:  $C_1, \dots, C_m$ .

**Question:** Find the assignment to  $x_1, \dots, x_n$  that satisfies the maximum number of clauses.

- ⑤ **Max 3SAT** is **NP-Hard**
- ⑥ **Max 3SAT** is a *maximization problem*.

# Some definitions

## Definition

Algorithm **Alg** for a maximization problem achieves an approximation factor  $\alpha \leq 1$  if for all inputs, we have:

$$\frac{\text{Alg}(G)}{\text{Opt}(G)} \geq \alpha.$$

*randomized algorithm*: it is allowed to consult with a source of random numbers in making decisions.

## Definition (Linearity of expectations.)

Given two random variables  $X, Y$  (not necessarily independent, we have that  $\mathbf{E}[X + Y] = \mathbf{E}[X] + \mathbf{E}[Y]$ .

# Some definitions

## Definition

Algorithm **Alg** for a maximization problem achieves an approximation factor  $\alpha \leq 1$  if for all inputs, we have:

$$\frac{\text{Alg}(G)}{\text{Opt}(G)} \geq \alpha.$$

**randomized algorithm**: it is allowed to consult with a source of random numbers in making decisions.

## Definition (Linearity of expectations.)

Given two random variables  $X, Y$  (not necessarily independent, we have that  $\mathbf{E}[X + Y] = \mathbf{E}[X] + \mathbf{E}[Y]$ .



# Some definitions

## Definition

Algorithm **Alg** for a maximization problem achieves an approximation factor  $\alpha \leq 1$  if for all inputs, we have:

$$\frac{\text{Alg}(G)}{\text{Opt}(G)} \geq \alpha.$$

**randomized algorithm**: it is allowed to consult with a source of random numbers in making decisions.

## Definition (Linearity of expectations.)

Given two random variables  $X, Y$  (not necessarily independent, we have that  $\mathbf{E}[X + Y] = \mathbf{E}[X] + \mathbf{E}[Y]$ .

# Approximating Max3SAT

## Theorem

*Expected  $(7/8)$ -approximation to **Max 3SAT** in polynomial time.  
 $F$  has  $m$  clauses  $\implies$  generated assignment satisfies  $(7/8)m$  clauses in expectation.*

## Proof

- ①  $x_1, \dots, x_n$ :  $n$  variables used.
- ② Randomly and independently assign 0/1 values to  $x_1, \dots, x_n$ .
- ③  $Y_i$ : indicator variable is 1  $\iff$   $i$ th clause in instance is satisfied.
- ④  $Y = \sum_{i=1}^m Y_i$ : # clauses satisfied.

# Approximating Max3SAT - proof continued

## Proof continued:

- ① Claim:  $\mathbf{E}[Y] = (7/8)m$ ,  $m$  = number of clauses.

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

by linearity of expectation.

- ②  $\Pr[Y_i = 0] = \frac{1}{2} * \frac{1}{2} * \frac{1}{2} = \frac{1}{8}. \implies \Pr[Y_i = 1] = \frac{7}{8},$

$$\mathbf{E}[Y_i] = \Pr[Y_i = 0] * 0 + \Pr[Y_i = 1] * 1 = \frac{7}{8}.$$

$$\mathbf{E}[\# \text{ of clauses sat}] = \mathbf{E}[Y] = \sum_{i=1}^m \mathbf{E}[Y_i] = (7/8)m. \quad \blacksquare$$

# Approximating Max3SAT - proof continued

## Proof continued:

- ① Claim:  $\mathbf{E}[Y] = (7/8)m$ ,  $m$  = number of clauses.

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

by linearity of expectation.

②  $\Pr[Y_i = 0] = \frac{1}{2} * \frac{1}{2} * \frac{1}{2} = \frac{1}{8}. \implies \Pr[Y_i = 1] = \frac{7}{8},$

$$\mathbf{E}[Y_i] = \Pr[Y_i = 0] * 0 + \Pr[Y_i = 1] * 1 = \frac{7}{8}.$$

$$\mathbf{E}[\# \text{ of clauses sat}] = \mathbf{E}[Y] = \sum_{i=1}^m \mathbf{E}[Y_i] = (7/8)m. \quad \blacksquare$$

# Approximating Max3SAT - proof continued

## Proof continued:

- ① Claim:  $\mathbf{E}[Y] = (7/8)m$ ,  $m$  = number of clauses.

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

by linearity of expectation.

- ②  $\Pr[Y_i = 0] = \frac{1}{2} * \frac{1}{2} * \frac{1}{2} = \frac{1}{8}. \implies \Pr[Y_i = 1] = \frac{7}{8},$

$$\mathbf{E}[Y_i] = \Pr[Y_i = 0] * 0 + \Pr[Y_i = 1] * 1 = \frac{7}{8}.$$

$$\mathbf{E}[\# \text{ of clauses sat}] = \mathbf{E}[Y] = \sum_{i=1}^m \mathbf{E}[Y_i] = (7/8)m. \quad \blacksquare$$

# Approximating Max3SAT - proof continued

## Proof continued:

- ① Claim:  $\mathbf{E}[Y] = (7/8)m$ ,  $m$  = number of clauses.

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

by linearity of expectation.

②  $\Pr[Y_i = 0] = \frac{1}{2} * \frac{1}{2} * \frac{1}{2} = \frac{1}{8}. \implies \Pr[Y_i = 1] = \frac{7}{8},$

$$\mathbf{E}[Y_i] = \Pr[Y_i = 0] * 0 + \Pr[Y_i = 1] * 1 = \frac{7}{8}.$$

$$\mathbf{E}[\# \text{ of clauses sat}] = \mathbf{E}[Y] = \sum_{i=1}^m \mathbf{E}[Y_i] = (7/8)m. \quad \blacksquare$$

# Approximating Max3SAT - proof continued

## Proof continued:

- ① Claim:  $\mathbf{E}[Y] = (7/8)m$ ,  $m$  = number of clauses.

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

by linearity of expectation.

- ②  $\Pr[Y_i = 0] = \frac{1}{2} * \frac{1}{2} * \frac{1}{2} = \frac{1}{8}. \implies \Pr[Y_i = 1] = \frac{7}{8},$

$$\mathbf{E}[Y_i] = \Pr[Y_i = 0] * 0 + \Pr[Y_i = 1] * 1 = \frac{7}{8}.$$

$$\mathbf{E}[\# \text{ of clauses sat}] = \mathbf{E}[Y] = \sum_{i=1}^m \mathbf{E}[Y_i] = (7/8)m. \quad \blacksquare$$

# Approximating Max3SAT

## Concluding remarks

- ① Algorithm quality independent of opt...
- ② Algorithm is oblivious.
- ③ ? proved that one can do no better; that is, for any constant  $\epsilon > 0$ , one can not approximate **3SAT** in polynomial time (unless **P** = **NP**) to within a factor of  $7/8 + \epsilon$ .
- ④ Amazing that a trivial algorithm like the above is essentially optimal!



# Approximating Max3SAT

## Concluding remarks

- 1 Algorithm quality independent of opt...
- 2 Algorithm is oblivious.
- 3 ? proved that one can do no better; that is, for any constant  $\epsilon > 0$ , one can not approximate **3SAT** in polynomial time (unless **P** = **NP**) to within a factor of  $7/8 + \epsilon$ .
- 4 Amazing that a trivial algorithm like the above is essentially optimal!

# Approximating Max3SAT

## Concluding remarks

- ① Algorithm quality independent of opt...
- ② Algorithm is oblivious.
- ③ ? proved that one can do no better; that is, for any constant  $\epsilon > 0$ , one can not approximate **3SAT** in polynomial time (unless **P** = **NP**) to within a factor of  $7/8 + \epsilon$ .
- ④ Amazing that a trivial algorithm like the above is essentially optimal!

# Approximating Max3SAT

## Concluding remarks

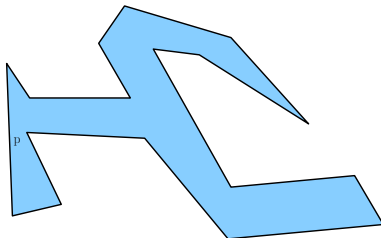
- ① Algorithm quality independent of opt...
- ② Algorithm is oblivious.
- ③ ? proved that one can do no better; that is, for any constant  $\epsilon > 0$ , one can not approximate **3SAT** in polynomial time (unless **P** = **NP**) to within a factor of  $7/8 + \epsilon$ .
- ④ Amazing that a trivial algorithm like the above is essentially optimal!

# Biographical Notes

The **Max 3SAT** remains hard in the “easier” variant of **MAX 2SAT** version, where every clause has **2** variables. It is known to be **NP-Hard** and approximable within **1.0741** ?, and is not approximable within **1.0476** ?. Notice, that the fact that **MAX 2SAT** is hard to approximate is surprising as **2SAT** can be solved in polynomial time (!).

# Guarding an Art Gallery

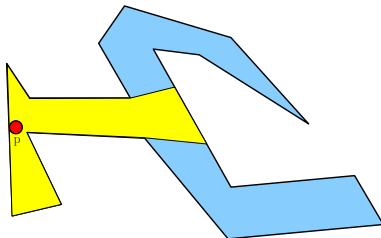
Set cover in the real world



- 1 Given: floor plan of an art gallery.
- 2 **Target:** Place min # guards that see the whole polygon.
- 3 **Visibility polygon** at  $p$ : region inside polygon that  $p$  can see.
- 4 Example of **Set Cover**.
- 5 **NP-Hard**, no approximation currently known.

# Guarding an Art Gallery

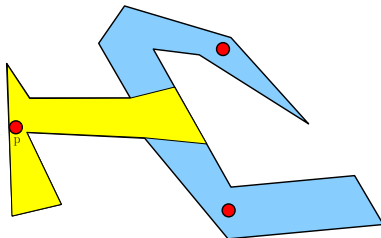
Set cover in the real world



- 1 Given: floor plan of an art gallery.
- 2 **Target:** Place min # guards that see the whole polygon.
- 3 *Visibility polygon* at  $p$ : region inside polygon that  $p$  can see.
- 4 Example of **Set Cover**.
- 5 **NP-Hard**, no approximation currently known.

# Guarding an Art Gallery

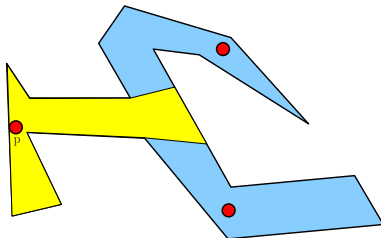
Set cover in the real world



- 1 Given: floor plan of an art gallery.
- 2 **Target:** Place min # guards that see the whole polygon.
- 3 **Visibility polygon** at  $p$ : region inside polygon that  $p$  can see.
- 4 Example of **Set Cover**.
- 5 **NP-Hard**, no approximation currently known.

# Guarding an Art Gallery

Set cover in the real world

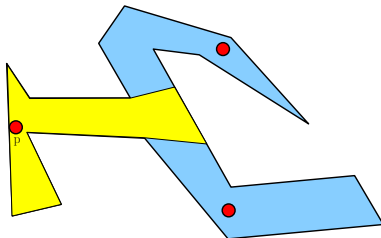


- 1 Given: floor plan of an art gallery.
- 2 **Target:** Place min # guards that see the whole polygon.
- 3 **Visibility polygon** at  $p$ : region inside polygon that  $p$  can see.
- 4 Example of **Set Cover**.
- 5 **NP-Hard**, no approximation currently known.



# Guarding an Art Gallery

Set cover in the real world



- 1 Given: floor plan of an art gallery.
- 2 **Target:** Place min # guards that see the whole polygon.
- 3 **Visibility polygon** at  $p$ : region inside polygon that  $p$  can see.
- 4 Example of **Set Cover**.
- 5 **NP-Hard**, no approximation currently known.

## Set Cover

**Instance:**  $(S, \mathcal{F})$ :

$S$  - a set of  $n$  elements

$\mathcal{F}$  - a family of subsets of  $S$ , s.t.

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

**Question:** The set  $\mathcal{X} \subseteq \mathcal{F}$  such that  $\mathcal{X}$  contains as few sets as possible, and  $\mathcal{X}$  covers  $S$ . Formally,  $\bigcup_{X \in \mathcal{X}} X = S$ .

$S$ : *ground set*

$(S, \mathcal{F})$ : *set system* or a *hypergraph*.

**Set Cover** is a minimization problem. **NP-Hard**.

# Set cover

## Example

### Example

Consider set  $S = \{1, 2, 3, 4, 5\}$  and the family of subsets

$$\mathcal{F} = \{\{1, 2, 3\}, \{2, 5\}, \{1, 4\}, \{4, 5\}\}.$$

Smallest cover of  $S$  is  $\mathcal{X}_{opt} = \{\{1, 2, 3\}, \{4, 5\}\}.$

# Set cover

## Greedy algorithm

```
GreedySetCover( $S, \mathcal{F}$ )  
   $X_0 \leftarrow \emptyset, \quad U_0 \leftarrow S, \quad i \leftarrow 0$   
  while  $U_i$  is not empty do  
     $Y_i \leftarrow$  set in  $\mathcal{F}$  covering largest  
      # of elements in  $U_i$   
     $X_{i+1} \leftarrow X_i \cup \{Y_i\}$   
     $U_{i+1} \leftarrow U_i \setminus Y_i$   
     $i \leftarrow i + 1$   
  
  return  $X_i$ .
```

- 1  $S$ : set of  $n$  elements.
- 2  $\mathcal{F}$ :  $m$  sets.
- 3 Size of input  $\Omega(m + n)$  (and  $O(mn)$ ).

# Set cover – Greedy algorithm

## Analysis

- ①  $\mathcal{X}_{opt} = \{V_1, \dots, V_k\} \subseteq \mathcal{F}$ : optimal solution.
- ②  $U_i$ : elements not covered in beginning of  $i$ th iteration.
- ③  $U_1 = \mathbf{S}$ .
- ④  $Y_i$ : set added to the cover in  $i$ th iteration.
- ⑤  $\alpha_i = |Y_i \cap U_i|$ : # of new elements being covered.

### Claim

We have  $\alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_k \geq \dots \geq \alpha_m$ .

### Proof.

If  $\alpha_i < \alpha_{i+1}$  then  $Y_{i+1}$  covers more elements than  $Y_i$  and we can exchange between them, and get a better set. A contradiction.  $\square$

# Set cover – Greedy algorithm

## Analysis

- ①  $\mathcal{X}_{opt} = \{V_1, \dots, V_k\} \subseteq \mathcal{F}$ : optimal solution.
- ②  $U_i$ : elements not covered in beginning of  $i$ th iteration.
- ③  $U_1 = \mathbf{S}$ .
- ④  $Y_i$ : set added to the cover in  $i$ th iteration.
- ⑤  $\alpha_i = |Y_i \cap U_i|$ : # of new elements being covered.

## Claim

We have  $\alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_k \geq \dots \geq \alpha_m$ .

## Proof.

If  $\alpha_i < \alpha_{i+1}$  then  $Y_{i+1}$  covers more elements than  $Y_i$  and we can exchange between them, and get a better set. A contradiction.  $\square$

# Set cover – Greedy algorithm

## Analysis

- ①  $\mathcal{X}_{opt} = \{V_1, \dots, V_k\} \subseteq \mathcal{F}$ : optimal solution.
- ②  $U_i$ : elements not covered in beginning of  $i$ th iteration.
- ③  $U_1 = S$ .
- ④  $Y_i$ : set added to the cover in  $i$ th iteration.
- ⑤  $\alpha_i = |Y_i \cap U_i|$ : # of new elements being covered.

## Claim

We have  $\alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_k \geq \dots \geq \alpha_m$ .

## Proof.

If  $\alpha_i < \alpha_{i+1}$  then  $Y_{i+1}$  covers more elements than  $Y_i$  and we can exchange between them, and get a better set. A contradiction.  $\square$

# Set cover – Greedy algorithm

## Analysis continued

### Claim

$\alpha_i \geq |U_i| / k$ . Equivalently:  $|U_{i+1}| \leq (1 - 1/k) |U_i|$ .

### Proof.

- ①  $k$ : Size of optimal solution.
- ② Opt solution:  $\mathcal{O} = \{O_1, \dots, O_k\}$  covers ground set  $S$ .
- ③  $\implies \forall i \quad U_i \subseteq S \subseteq \bigcup_{i=1}^k O_i$  elements of  $U_i$ .
- ④  $\implies$  one set of opt covers  $\geq |U_i| / k$  of  $U_i$ .
- ⑤ greedy algorithm picks set  $Y_i$  with max cover.
- ⑥  $\implies Y_i$  covers  $\alpha_i \geq |U_i| / k$  (prev. not covered) elements.
- ⑦  $|U_{i+1}| = |U_i| - \alpha_i \leq (1 - 1/k) |U_i|$ .





# Set cover – Greedy algorithm

## Analysis continued

### Claim

$\alpha_i \geq |U_i|/k$ . Equivalently:  $|U_{i+1}| \leq (1 - 1/k) |U_i|$ .

### Proof.

- ①  $k$ : Size of optimal solution.
- ② Opt solution:  $\mathcal{O} = \{O_1, \dots, O_k\}$  covers ground set  $S$ .
- ③  $\implies \forall i \quad U_i \subseteq S \subseteq \bigcup_{i=1}^k O_i$  elements of  $U_i$ .
- ④  $\implies$  one set of opt covers  $\geq |U_i|/k$  of  $U_i$ .
- ⑤ greedy algorithm picks set  $Y_i$  with max cover.
- ⑥  $\implies Y_i$  covers  $\alpha_i \geq |U_i|/k$  (prev. not covered) elements.
- ⑦  $|U_{i+1}| = |U_i| - \alpha_i \leq (1 - 1/k) |U_i|$ .



# Set cover – Greedy algorithm

## Analysis continued

### Claim

$\alpha_i \geq |U_i| / k$ . Equivalently:  $|U_{i+1}| \leq (1 - 1/k) |U_i|$ .

### Proof.

- ①  $k$ : Size of optimal solution.
- ② Opt solution:  $\mathcal{O} = \{O_1, \dots, O_k\}$  covers ground set  $S$ .
- ③  $\implies \forall i \quad U_i \subseteq S \subseteq \bigcup_{i=1}^k O_i$  elements of  $U_i$ .
- ④  $\implies$  one set of opt covers  $\geq |U_i| / k$  of  $U_i$ .
- ⑤ greedy algorithm picks set  $Y_i$  with max cover.
- ⑥  $\implies Y_i$  covers  $\alpha_i \geq |U_i| / k$  (prev. not covered) elements.
- ⑦  $|U_{i+1}| = |U_i| - \alpha_i \leq (1 - 1/k) |U_i|$ .



# Set cover – Greedy algorithm

## Analysis continued

### Claim

$\alpha_i \geq |U_i| / k$ . Equivalently:  $|U_{i+1}| \leq (1 - 1/k) |U_i|$ .

### Proof.

- ①  $k$ : Size of optimal solution.
- ② Opt solution:  $\mathcal{O} = \{O_1, \dots, O_k\}$  covers ground set  $S$ .
- ③  $\implies \forall i \quad U_i \subseteq S \subseteq \bigcup_{i=1}^k O_i$  elements of  $U_i$ .
- ④  $\implies$  one set of opt covers  $\geq |U_i| / k$  of  $U_i$ .
- ⑤ greedy algorithm picks set  $Y_i$  with max cover.
- ⑥  $\implies Y_i$  covers  $\alpha_i \geq |U_i| / k$  (prev. not covered) elements.
- ⑦  $|U_{i+1}| = |U_i| - \alpha_i \leq (1 - 1/k) |U_i|$ .



# Set cover – Greedy algorithm

## Analysis continued

### Claim

$\alpha_i \geq |U_i| / k$ . Equivalently:  $|U_{i+1}| \leq (1 - 1/k) |U_i|$ .

### Proof.

- ①  $k$ : Size of optimal solution.
- ② Opt solution:  $\mathcal{O} = \{O_1, \dots, O_k\}$  covers ground set  $S$ .
- ③  $\implies \forall i \quad U_i \subseteq S \subseteq \bigcup_{i=1}^k O_i$  elements of  $U_i$ .
- ④  $\implies$  one set of opt covers  $\geq |U_i| / k$  of  $U_i$ .
- ⑤ greedy algorithm picks set  $Y_i$  with max cover.
- ⑥  $\implies Y_i$  covers  $\alpha_i \geq |U_i| / k$  (prev. not covered) elements.
- ⑦  $|U_{i+1}| = |U_i| - \alpha_i \leq (1 - 1/k) |U_i|$ .



# Set cover – Greedy algorithm

## Analysis continued

### Claim

$\alpha_i \geq |U_i|/k$ . Equivalently:  $|U_{i+1}| \leq (1 - 1/k) |U_i|$ .

### Proof.

- ①  $k$ : Size of optimal solution.
- ② Opt solution:  $\mathcal{O} = \{O_1, \dots, O_k\}$  covers ground set  $S$ .
- ③  $\implies \forall i \quad U_i \subseteq S \subseteq \bigcup_{i=1}^k O_i$  elements of  $U_i$ .
- ④  $\implies$  one set of opt covers  $\geq |U_i|/k$  of  $U_i$ .
- ⑤ greedy algorithm picks set  $Y_i$  with max cover.
- ⑥  $\implies Y_i$  covers  $\alpha_i \geq |U_i|/k$  (prev. not covered) elements.
- ⑦  $|U_{i+1}| = |U_i| - \alpha_i \leq (1 - 1/k) |U_i|$ .



# Set cover – Greedy algorithm

## Analysis continued

### Claim

$\alpha_i \geq |U_i| / k$ . Equivalently:  $|U_{i+1}| \leq (1 - 1/k) |U_i|$ .

### Proof.

- ①  $k$ : Size of optimal solution.
- ② Opt solution:  $\mathcal{O} = \{O_1, \dots, O_k\}$  covers ground set  $S$ .
- ③  $\implies \forall i \quad U_i \subseteq S \subseteq \bigcup_{i=1}^k O_i$  elements of  $U_i$ .
- ④  $\implies$  one set of opt covers  $\geq |U_i| / k$  of  $U_i$ .
- ⑤ greedy algorithm picks set  $Y_i$  with max cover.
- ⑥  $\implies Y_i$  covers  $\alpha_i \geq |U_i| / k$  (prev. not covered) elements.
- ⑦  $|U_{i+1}| = |U_i| - \alpha_i \leq (1 - 1/k) |U_i|$ .



# Set cover – Greedy algorithm

Analysis continued

Using the claim

$$|U_i| \leq (1 - 1/k) |U_{i-1}| \leq (1 - 1/k)^i |U_0| = (1 - 1/k)^i n.$$

Useful Fact

$$1 - x \leq e^{-x}.$$

# Set cover – Greedy algorithm

Analysis continued

Using the claim

$$|U_i| \leq (1 - 1/k) |U_{i-1}| \leq (1 - 1/k)^i |U_0| = (1 - 1/k)^i n.$$

Useful Fact

$$1 - x \leq e^{-x}.$$



# Set cover – Greedy algorithm

Analysis continued

Using the claim

$$|U_i| \leq (1 - 1/k) |U_{i-1}| \leq (1 - 1/k)^i |U_0| = (1 - 1/k)^i n.$$

Useful Fact

$$1 - x \leq e^{-x}.$$

# Set cover – Greedy algorithm

Analysis continued

## Theorem

**GreedySetCover**( $\mathbf{S}, \mathcal{F}$ ) generates a cover of  $\mathbf{S}$  using at most  $O(k \log n)$  sets of  $\mathcal{F}$ ,  $k$ : size of the cover in opt solution.  $n = |\mathbf{S}|$

## Proof.

In what round  $M$  is  $U_M$  empty?

$$\begin{aligned} \text{For } M = \lceil 2k \ln n \rceil: |U_M| &\leq \left(1 - \frac{1}{k}\right)^M n \leq \exp\left(-\frac{1}{k}M\right) n \\ &= \exp\left(-\frac{\lceil 2k \ln n \rceil}{k}\right) n \leq \exp(-2 \ln n) n = \frac{1}{n} < 1, \end{aligned}$$

$$\implies |U_M| = 0$$

$\implies$  Algorithm terminates before reaching  $M$ th iteration. □

# Set cover – Greedy algorithm

Analysis continued

## Theorem

**GreedySetCover**( $\mathbf{S}, \mathcal{F}$ ) generates a cover of  $\mathbf{S}$  using at most  $O(k \log n)$  sets of  $\mathcal{F}$ ,  $k$ : size of the cover in opt solution.  $n = |\mathbf{S}|$

## Proof.

In what round  $M$  is  $U_M$  empty?

$$\begin{aligned} \text{For } M = \lceil 2k \ln n \rceil: |U_M| &\leq \left(1 - \frac{1}{k}\right)^M n \leq \exp\left(-\frac{1}{k}M\right) n \\ &= \exp\left(-\frac{\lceil 2k \ln n \rceil}{k}\right) n \leq \exp(-2 \ln n) n = \frac{1}{n} < 1, \end{aligned}$$

$$\implies |U_M| = 0$$

$\implies$  Algorithm terminates before reaching  $M$ th iteration. □

# Set cover – Greedy algorithm

Analysis continued

## Theorem

**GreedySetCover**( $\mathbf{S}, \mathcal{F}$ ) generates a cover of  $\mathbf{S}$  using at most  $O(k \log n)$  sets of  $\mathcal{F}$ ,  $k$ : size of the cover in opt solution.  $n = |\mathbf{S}|$

## Proof.

In what round  $M$  is  $U_M$  empty?

$$\begin{aligned} \text{For } M = \lceil 2k \ln n \rceil: |U_M| &\leq \left(1 - \frac{1}{k}\right)^M n \leq \exp\left(-\frac{1}{k}M\right) n \\ &= \exp\left(-\frac{\lceil 2k \ln n \rceil}{k}\right) n \leq \exp(-2 \ln n) n = \frac{1}{n} < 1, \end{aligned}$$

$$\implies |U_M| = 0$$

$\implies$  Algorithm terminates before reaching  $M$ th iteration. □

# Set cover – Greedy algorithm

Analysis continued

## Theorem

**GreedySetCover**( $\mathbf{S}, \mathcal{F}$ ) generates a cover of  $\mathbf{S}$  using at most  $O(k \log n)$  sets of  $\mathcal{F}$ ,  $k$ : size of the cover in opt solution.  $n = |\mathbf{S}|$

## Proof.

In what round  $M$  is  $U_M$  empty?

$$\begin{aligned} \text{For } M = \lceil 2k \ln n \rceil: |U_M| &\leq \left(1 - \frac{1}{k}\right)^M n \leq \exp\left(-\frac{1}{k}M\right) n \\ &= \exp\left(-\frac{\lceil 2k \ln n \rceil}{k}\right) n \leq \exp(-2 \ln n) n = \frac{1}{n} < 1, \end{aligned}$$

$$\implies |U_M| = 0$$

$\implies$  Algorithm terminates before reaching  $M$ th iteration. □

# Set cover – Greedy algorithm

Analysis continued

## Theorem

**GreedySetCover**( $\mathbf{S}, \mathcal{F}$ ) generates a cover of  $\mathbf{S}$  using at most  $O(k \log n)$  sets of  $\mathcal{F}$ ,  $k$ : size of the cover in opt solution.  $n = |\mathbf{S}|$

## Proof.

In what round  $M$  is  $U_M$  empty?

$$\begin{aligned} \text{For } M = \lceil 2k \ln n \rceil: |U_M| &\leq \left(1 - \frac{1}{k}\right)^M n \leq \exp\left(-\frac{1}{k}M\right) n \\ &= \exp\left(-\frac{\lceil 2k \ln n \rceil}{k}\right) n \leq \exp(-2 \ln n) n = \frac{1}{n} < 1, \end{aligned}$$

$$\implies |U_M| = 0$$

$\implies$  Algorithm terminates before reaching  $M$ th iteration. □

# Set cover – Greedy algorithm

Analysis continued

## Theorem

**GreedySetCover**( $\mathbf{S}, \mathcal{F}$ ) generates a cover of  $\mathbf{S}$  using at most  $O(k \log n)$  sets of  $\mathcal{F}$ ,  $k$ : size of the cover in opt solution.  $n = |\mathbf{S}|$

## Proof.

In what round  $M$  is  $U_M$  empty?

$$\begin{aligned} \text{For } M = \lceil 2k \ln n \rceil: |U_M| &\leq \left(1 - \frac{1}{k}\right)^M n \leq \exp\left(-\frac{1}{k}M\right) n \\ &= \exp\left(-\frac{\lceil 2k \ln n \rceil}{k}\right) n \leq \exp(-2 \ln n) n = \frac{1}{n} < 1, \end{aligned}$$

$$\implies |U_M| = 0$$

$\implies$  Algorithm terminates before reaching  $M$ th iteration. □

# Set cover – Greedy algorithm

Analysis continued

## Theorem

**GreedySetCover**( $\mathbf{S}, \mathcal{F}$ ) generates a cover of  $\mathbf{S}$  using at most  $O(k \log n)$  sets of  $\mathcal{F}$ ,  $k$ : size of the cover in opt solution.  $n = |\mathbf{S}|$

## Proof.

In what round  $M$  is  $U_M$  empty?

$$\begin{aligned} \text{For } M = \lceil 2k \ln n \rceil: |U_M| &\leq \left(1 - \frac{1}{k}\right)^M n \leq \exp\left(-\frac{1}{k}M\right) n \\ &= \exp\left(-\frac{\lceil 2k \ln n \rceil}{k}\right) n \leq \exp(-2 \ln n) n = \frac{1}{n} < 1, \end{aligned}$$

$$\implies |U_M| = 0$$

$\implies$  Algorithm terminates before reaching  $M$ th iteration. □



# Set cover – Greedy algorithm

Analysis continued

## Theorem

**GreedySetCover**( $\mathbf{S}, \mathcal{F}$ ) generates a cover of  $\mathbf{S}$  using at most  $O(k \log n)$  sets of  $\mathcal{F}$ ,  $k$ : size of the cover in opt solution.  $n = |\mathbf{S}|$

## Proof.

In what round  $M$  is  $U_M$  empty?

$$\begin{aligned} \text{For } M = \lceil 2k \ln n \rceil: |U_M| &\leq \left(1 - \frac{1}{k}\right)^M n \leq \exp\left(-\frac{1}{k}M\right) n \\ &= \exp\left(-\frac{\lceil 2k \ln n \rceil}{k}\right) n \leq \exp(-2 \ln n) n = \frac{1}{n} < 1, \end{aligned}$$

$$\implies |U_M| = 0$$

$\implies$  Algorithm terminates before reaching  $M$ th iteration. □

# Set cover – Greedy algorithm

Analysis continued

## Theorem

**GreedySetCover**( $\mathbf{S}, \mathcal{F}$ ) generates a cover of  $\mathbf{S}$  using at most  $O(k \log n)$  sets of  $\mathcal{F}$ ,  $k$ : size of the cover in opt solution.  $n = |\mathbf{S}|$

## Proof.

In what round  $M$  is  $U_M$  empty?

$$\begin{aligned}\text{For } M = \lceil 2k \ln n \rceil: |U_M| &\leq \left(1 - \frac{1}{k}\right)^M n \leq \exp\left(-\frac{1}{k}M\right) n \\ &= \exp\left(-\frac{\lceil 2k \ln n \rceil}{k}\right) n \leq \exp(-2 \ln n) n = \frac{1}{n} < 1, \\ \implies |U_M| &= 0\end{aligned}$$

$\implies$  Algorithm terminates before reaching  $M$ th iteration. □

# Set cover – Greedy algorithm

## Analysis continued

### Theorem

**GreedySetCover**( $\mathbf{S}, \mathcal{F}$ ) generates a cover of  $\mathbf{S}$  using at most  $O(k \log n)$  sets of  $\mathcal{F}$ ,  $k$ : size of the cover in opt solution.  $n = |\mathbf{S}|$

### Proof.

In what round  $M$  is  $U_M$  empty?

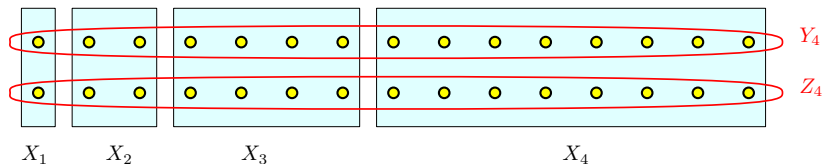
$$\begin{aligned} \text{For } M = \lceil 2k \ln n \rceil: |U_M| &\leq \left(1 - \frac{1}{k}\right)^M n \leq \exp\left(-\frac{1}{k}M\right) n \\ &= \exp\left(-\frac{\lceil 2k \ln n \rceil}{k}\right) n \leq \exp(-2 \ln n) n = \frac{1}{n} < 1, \end{aligned}$$

$$\implies |U_M| = 0$$

$\implies$  Algorithm terminates before reaching  $M$ th iteration. □

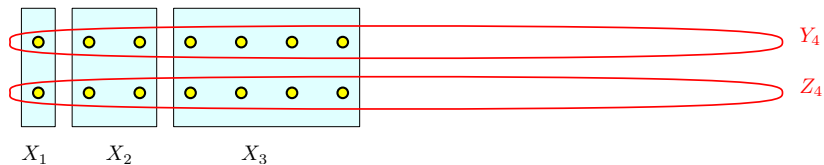
# Set cover – Greedy algorithm

## Lower bound



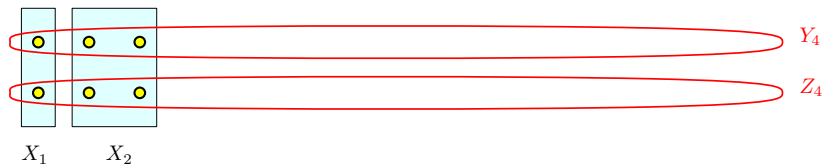
# Set cover – Greedy algorithm

## Lower bound



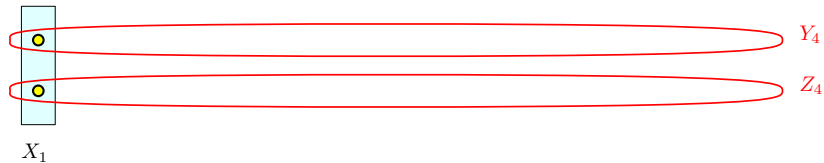
# Set cover – Greedy algorithm

## Lower bound



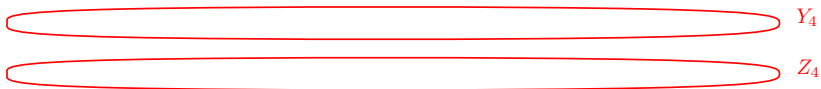
# Set cover – Greedy algorithm

## Lower bound



# Set cover – Greedy algorithm

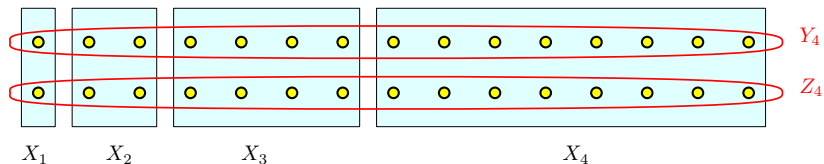
## Lower bound





# Set cover – Greedy algorithm

## Lower bound



## Lemma

Let  $n = 2^{i+1} - 2$ .  $\exists$  instance of **Set Cover** of  $n$  elements.  
Optimal cover is by two sets.

**GreedySetCover** would use  $i = \lfloor \lg n \rfloor$  sets.

**GreedySetCover** is a  $\Theta(\log n)$  approximation to **SetCover**.

## Weighted Set Cover

**Instance:**  $(\mathbf{S}, \mathcal{F}, \rho)$ :

$\mathbf{S}$ : a set of  $n$  elements

$\mathcal{F}$ : family subsets of  $\mathbf{S}$ , s.t.  $\bigcup_{X \in \mathcal{F}} X = \mathbf{S}$ .

$\rho(\cdot)$ : A price function assigning price to each set in  $\mathcal{F}$ .

**Question:** The set  $\mathcal{X} \subseteq \mathcal{F}$ , such that  $\mathcal{X}$  covers  $\mathbf{S}$ . Formally,  $\bigcup_{X \in \mathcal{X}} X = \mathbf{S}$ , and  $\rho(\mathcal{X}) = \sum_{X \in \mathcal{X}} \rho(X)$  is minimized.

- 1 **WGreedySetCover**: repeatedly picks set that pays the least cover each element it covers.
- 2  $X \in \mathcal{F}$  covered  $t$  new elements, then the *average price* it pays per element  $\beta(X) = \rho(X) / t$ .
- 3 **WGreedySetCover**: picks the set with the lowest average price.

## Weighted Set Cover

**Instance:**  $(\mathbf{S}, \mathcal{F}, \rho)$ :

$\mathbf{S}$ : a set of  $n$  elements

$\mathcal{F}$ : family subsets of  $\mathbf{S}$ , s.t.  $\bigcup_{X \in \mathcal{F}} X = \mathbf{S}$ .

$\rho(\cdot)$ : A price function assigning price to each set in  $\mathcal{F}$ .

**Question:** The set  $\mathcal{X} \subseteq \mathcal{F}$ , such that  $\mathcal{X}$  covers  $\mathbf{S}$ . Formally,  $\bigcup_{X \in \mathcal{X}} X = \mathbf{S}$ , and  $\rho(\mathcal{X}) = \sum_{X \in \mathcal{X}} \rho(X)$  is minimized.

- 1 **WGreedySetCover**: repeatedly picks set that pays the least cover each element it covers.
- 2  $X \in \mathcal{F}$  covered  $t$  new elements, then the *average price* it pays per element  $\beta(X) = \rho(X) / t$ .
- 3 **WGreedySetCover**: picks the set with the lowest average price.

## Weighted Set Cover

**Instance:**  $(\mathbf{S}, \mathcal{F}, \rho)$ :

$\mathbf{S}$ : a set of  $n$  elements

$\mathcal{F}$ : family subsets of  $\mathbf{S}$ , s.t.  $\bigcup_{X \in \mathcal{F}} X = \mathbf{S}$ .

$\rho(\cdot)$ : A price function assigning price to each set in  $\mathcal{F}$ .

**Question:** The set  $\mathcal{X} \subseteq \mathcal{F}$ , such that  $\mathcal{X}$  covers  $\mathbf{S}$ . Formally,  $\bigcup_{X \in \mathcal{X}} X = \mathbf{S}$ , and  $\rho(\mathcal{X}) = \sum_{X \in \mathcal{X}} \rho(X)$  is minimized.

- 1 **WGreedySetCover**: repeatedly picks set that pays the least cover each element it covers.
- 2  $X \in \mathcal{F}$  covered  $t$  new elements, then the *average price* it pays per element  $\beta(X) = \rho(X) / t$ .
- 3 **WGreedySetCover**: picks the set with the lowest average price.

## Weighted Set Cover

**Instance:**  $(\mathbf{S}, \mathcal{F}, \rho)$ :

$\mathbf{S}$ : a set of  $n$  elements

$\mathcal{F}$ : family subsets of  $\mathbf{S}$ , s.t.  $\bigcup_{X \in \mathcal{F}} X = \mathbf{S}$ .

$\rho(\cdot)$ : A price function assigning price to each set in  $\mathcal{F}$ .

**Question:** The set  $\mathcal{X} \subseteq \mathcal{F}$ , such that  $\mathcal{X}$  covers  $\mathbf{S}$ . Formally,  $\bigcup_{X \in \mathcal{X}} X = \mathbf{S}$ , and  $\rho(\mathcal{X}) = \sum_{X \in \mathcal{X}} \rho(X)$  is minimized.

- 1 **WGreedySetCover**: repeatedly picks set that pays the least cover each element it covers.
- 2  $X \in \mathcal{F}$  covered  $t$  new elements, then the *average price* it pays per element  $\beta(X) = \rho(X) / t$ .
- 3 **WGreedySetCover**: picks the set with the lowest average price.

# Weighted set cover – greedy algorithm

## Analysis

- ①  $U_i$ : set of elements not covered in beginning  $i$ th iteration.
- ②  $U_1 = S$ .
- ③  $\text{Opt}$ : optimal solution.
- ④ *average optimal cost*:  $\beta_i = \rho(\text{Opt}) / |U_i|$ ,

# Weighted set cover – greedy algorithm

## Analysis – continued

### Lemma

(A)  $\beta_1 \leq \beta_2 \leq \dots$ .

(B) For  $i < j$ , we have if  $|U_j| > |U_i|/2$  then  $2\beta_i > \beta_j$ .

### Proof.

(A)  $\beta_i = \frac{\rho(\text{Opt})}{|U_i|}$ :  $\rho(\text{Opt})$  is constant and  $|U_i|$  can only decrease

(B)  $|U_j| > |U_i|/2 \implies 2/|U_i| > 1/|U_j| \implies$   
 $2\rho(\text{Opt})/|U_i| > \rho(\text{Opt})/|U_j| \implies 2\beta_i > \beta_j$  □

# Weighted set cover – greedy algorithm

## Analysis – continued

### Lemma

(A)  $\beta_1 \leq \beta_2 \leq \dots$ .

(B) For  $i < j$ , we have if  $|U_j| > |U_i|/2$  then  $2\beta_i > \beta_j$ .

### Proof.

(A)  $\beta_i = \frac{\rho(\text{Opt})}{|U_i|}$ :  $\rho(\text{Opt})$  is constant and  $|U_i|$  can only decrease

(B)  $|U_j| > |U_i|/2 \implies 2/|U_i| > 1/|U_j| \implies$   
 $2\rho(\text{Opt})/|U_i| > \rho(\text{Opt})/|U_j| \implies 2\beta_i > \beta_j$  □



# Weighted set cover – greedy algorithm

## Analysis – continued

### Lemma

(A)  $\beta_1 \leq \beta_2 \leq \dots$ .

(B) For  $i < j$ , we have if  $|U_j| > |U_i|/2$  then  $2\beta_i > \beta_j$ .

### Proof.

(A)  $\beta_i = \frac{\rho(\text{Opt})}{|U_i|}$ :  $\rho(\text{Opt})$  is constant and  $|U_i|$  can only decrease

(B)  $|U_j| > |U_i|/2 \implies 2/|U_i| > 1/|U_j| \implies$   
 $2\rho(\text{Opt})/|U_i| > \rho(\text{Opt})/|U_j| \implies 2\beta_i > \beta_j$  □

# Weighted set cover – greedy algorithm

## Analysis – continued

### Lemma

(A)  $\beta_1 \leq \beta_2 \leq \dots$ .

(B) For  $i < j$ , we have if  $|U_j| > |U_i|/2$  then  $2\beta_i > \beta_j$ .

### Proof.

(A)  $\beta_i = \frac{\rho(\text{Opt})}{|U_i|}$ :  $\rho(\text{Opt})$  is constant and  $|U_i|$  can only decrease

(B)  $|U_j| > |U_i|/2 \implies 2/|U_i| > 1/|U_j| \implies$   
 $2\rho(\text{Opt})/|U_i| > \rho(\text{Opt})/|U_j| \implies 2\beta_i > \beta_j$  □

# Weighted set cover – greedy algorithm

## Analysis – continued

### Lemma

$\beta_i = \rho(\text{Opt}) / |U_i|$ : average optimal cost per uncovered element.

Let  $\text{Opt} = \{X_1, \dots, X_m\}$ , and  $s_j = |U_i \cap X_j|$ .

Then  $\exists X_j \in \text{Opt}$  with lower average cost:  $\rho(X_j) / s_j \leq \beta_i$ .

Proof.

$$\min_{j=1}^m \frac{\rho(X_j)}{s_j} \leq \frac{\sum_{j=1}^m \rho(X_j)}{\sum_{j=1}^m s_j} = \frac{\rho(\text{Opt})}{\sum_{j=1}^m s_j} \leq \frac{\rho(\text{Opt})}{|U_i|} = \beta_i.$$



# Weighted set cover – greedy algorithm

## Analysis – continued

### Lemma

$\beta_i = \rho(\text{Opt}) / |U_i|$ : average optimal cost per uncovered element.

Let  $\text{Opt} = \{X_1, \dots, X_m\}$ , and  $s_j = |U_i \cap X_j|$ .

Then  $\exists X_j \in \text{Opt}$  with lower average cost:  $\rho(X_j) / s_j \leq \beta_i$ .

### Proof.

$$\min_{j=1}^m \frac{\rho(X_j)}{s_j} \leq \frac{\sum_{j=1}^m \rho(X_j)}{\sum_{j=1}^m s_j} = \frac{\rho(\text{Opt})}{\sum_{j=1}^m s_j} \leq \frac{\rho(\text{Opt})}{|U_i|} = \beta_i.$$



# Weighted set cover – greedy algorithm

## Analysis – continued

### Lemma

$\beta_i = \rho(\text{Opt}) / |U_i|$ : average optimal cost per uncovered element.

Let  $\text{Opt} = \{X_1, \dots, X_m\}$ , and  $s_j = |U_i \cap X_j|$ .

Then  $\exists X_j \in \text{Opt}$  with lower average cost:  $\rho(X_j) / s_j \leq \beta_i$ .

### Proof.

$$\min_{j=1}^m \frac{\rho(X_j)}{s_j} \leq \frac{\sum_{j=1}^m \rho(X_j)}{\sum_{j=1}^m s_j} = \frac{\rho(\text{Opt})}{\sum_{j=1}^m s_j} \leq \frac{\rho(\text{Opt})}{|U_i|} = \beta_i.$$



# Weighted set cover – greedy algorithm

## Analysis – continued

### Lemma

$\beta_i = \rho(\text{Opt}) / |U_i|$ : average optimal cost per uncovered element.

Let  $\text{Opt} = \{X_1, \dots, X_m\}$ , and  $s_j = |U_i \cap X_j|$ .

Then  $\exists X_j \in \text{Opt}$  with lower average cost:  $\rho(X_j) / s_j \leq \beta_i$ .

### Proof.

$$\min_{j=1}^m \frac{\rho(X_j)}{s_j} \leq \frac{\sum_{j=1}^m \rho(X_j)}{\sum_{j=1}^m s_j} = \frac{\rho(\text{Opt})}{\sum_{j=1}^m s_j} \leq \frac{\rho(\text{Opt})}{|U_i|} = \beta_i.$$



# Weighted set cover – greedy algorithm

## Analysis – continued

### Lemma

$\beta_i = \rho(\text{Opt}) / |U_i|$ : average optimal cost per uncovered element.

Let  $\text{Opt} = \{X_1, \dots, X_m\}$ , and  $s_j = |U_i \cap X_j|$ .

Then  $\exists X_j \in \text{Opt}$  with lower average cost:  $\rho(X_j) / s_j \leq \beta_i$ .

### Proof.

$$\min_{j=1}^m \frac{\rho(X_j)}{s_j} \leq \frac{\sum_{j=1}^m \rho(X_j)}{\sum_{j=1}^m s_j} = \frac{\rho(\text{Opt})}{\sum_{j=1}^m s_j} \leq \frac{\rho(\text{Opt})}{|U_i|} = \beta_i.$$



### Main Point

Greedy pays at most  $\beta_i$  per element in round  $i$ .

# Weighted set cover – greedy algorithm

## Analysis – continued

### Lemma

*$k$ : first iteration  $|U_k| \leq n/2$ .*

*Total price of sets picked in iterations  $1 \dots k - 1$ , is  $\leq 2\rho(\text{Opt})$ .*

### Proof.

- ①  $|U_j| > |U_1|/2$  for  $j = 2, \dots, k - 1$ ,
- ② Earlier we showed: if  $|U_j| > |U_1|/2$  then  $2\beta_1 > \beta_j$ .
- ③  $\beta_j = \frac{\rho(\text{Opt})}{|U_j|}$  and  $|U_1| = n$   
 $\Rightarrow 2\rho(\text{Opt})/n > \beta_j$  for  $j = 1, \dots, k - 1$
- ④ We showed greedy pays at most  $\beta_j$  per element in round  $j$   
 $\Rightarrow$  in rounds  $j = 1, \dots, k - 1$  greedy paid at most twice what opt paid per element.





# Weighted set cover – greedy algorithm

## Analysis – continued

### Lemma

*$k$ : first iteration  $|U_k| \leq n/2$ .*

*Total price of sets picked in iterations  $1 \dots k - 1$ , is  $\leq 2\rho(\text{Opt})$ .*

### Proof.

- ①  $|U_j| > |U_1|/2$  for  $j = 2, \dots, k - 1$ ,
- ② Earlier we showed: if  $|U_j| > |U_1|/2$  then  $2\beta_1 > \beta_j$ .
- ③  $\beta_j = \frac{\rho(\text{Opt})}{|U_j|}$  and  $|U_1| = n$   
 $\Rightarrow 2\rho(\text{Opt})/n > \beta_j$  for  $j = 1, \dots, k - 1$
- ④ We showed greedy pays at most  $\beta_j$  per element in round  $j$   
 $\Rightarrow$  in rounds  $j = 1, \dots, k - 1$  greedy paid at most twice what opt paid per element.



# Weighted set cover – greedy algorithm

## Analysis – continued

### Lemma

*$k$ : first iteration  $|U_k| \leq n/2$ .*

*Total price of sets picked in iterations  $1 \dots k - 1$ , is  $\leq 2\rho(\text{Opt})$ .*

### Proof.

- ①  $|U_j| > |U_1|/2$  for  $j = 2, \dots, k - 1$ ,
- ② Earlier we showed: if  $|U_j| > |U_1|/2$  then  $2\beta_1 > \beta_j$ .
- ③  $\beta_j = \frac{\rho(\text{Opt})}{|U_j|}$  and  $|U_1| = n$   
 $\Rightarrow 2\rho(\text{Opt})/n > \beta_j$  for  $j = 1, \dots, k - 1$
- ④ We showed greedy pays at most  $\beta_j$  per element in round  $j$   
 $\Rightarrow$  in rounds  $j = 1, \dots, k - 1$  greedy paid at most twice what opt paid per element.



# Weighted set cover – greedy algorithm

## Analysis – continued

### Lemma

$k$ : first iteration  $|U_k| \leq n/2$ .

Total price of sets picked in iterations  $1 \dots k - 1$ , is  $\leq 2\rho(\text{Opt})$ .

### Proof.

- ①  $|U_j| > |U_1|/2$  for  $j = 2, \dots, k - 1$ ,
- ② Earlier we showed: if  $|U_j| > |U_1|/2$  then  $2\beta_1 > \beta_j$ .
- ③  $\beta_j = \frac{\rho(\text{Opt})}{|U_j|}$  and  $|U_1| = n$   
 $\Rightarrow 2\rho(\text{Opt})/n > \beta_j$  for  $j = 1, \dots, k - 1$
- ④ We showed greedy pays at most  $\beta_j$  per element in round  $j$   
 $\Rightarrow$  in rounds  $j = 1, \dots, k - 1$  greedy paid at most twice what opt paid per element.



# Weighted set cover – greedy algorithm

## The result

### Theorem

**WGreedySetCover** computes a  $O(\log n)$  approximation to the optimal weighted set cover solution.

### Proof.

- 1 By Lemma: **WGreedySetCover** paid at most twice the Opt price to cover half the elements.
- 2 Now, repeat the argument on the remaining uncovered elements.
- 3 After  $O(\log n)$  such halving steps, all sets covered.
- 4 In each halving step, **WGreedySetCover** paid at most twice the opt cost.



# Weighted set cover – greedy algorithm

## The result

### Theorem

**WGreedySetCover** computes a  $O(\log n)$  approximation to the optimal weighted set cover solution.

### Proof.

- 1 By Lemma: **WGreedySetCover** paid at most twice the Opt price to cover half the elements.
- 2 Now, repeat the argument on the remaining uncovered elements.
- 3 After  $O(\log n)$  such halving steps, all sets covered.
- 4 In each halving step, **WGreedySetCover** paid at most twice the opt cost.



# Weighted set cover – greedy algorithm

## The result

### Theorem

**WGreedySetCover** computes a  $O(\log n)$  approximation to the optimal weighted set cover solution.

### Proof.

- 1 By Lemma: **WGreedySetCover** paid at most twice the Opt price to cover half the elements.
- 2 Now, repeat the argument on the remaining uncovered elements.
- 3 After  $O(\log n)$  such halving steps, all sets covered.
- 4 In each halving step, **WGreedySetCover** paid at most twice the opt cost.



# Weighted set cover – greedy algorithm

## The result

### Theorem

**WGreedySetCover** computes a  $O(\log n)$  approximation to the optimal weighted set cover solution.

### Proof.

- 1 By Lemma: **WGreedySetCover** paid at most twice the Opt price to cover half the elements.
- 2 Now, repeat the argument on the remaining uncovered elements.
- 3 After  $O(\log n)$  such halving steps, all sets covered.
- 4 In each halving step, **WGreedySetCover** paid at most twice the opt cost.



# Part II

## Clustering



# Clustering

- ① ***unsupervised learning***: Given examples, partition them into classes of similar examples.
- ② Example: Given webpage  $X$  about “The reality dysfunction”, find all webpages on this topic (or closely related topics).
- ③ Webpage about “All quiet on the western front” should be in the same group as webpage as “Storm of steel”.
- ④ Hope: All such webpages of interest in same cluster as  $X$ , if the clustering is good.

# Clustering

- ① ***unsupervised learning***: Given examples, partition them into classes of similar examples.
- ② Example: Given webpage **X** about “The reality dysfunction”, find all webpages on this topic (or closely related topics).
- ③ Webpage about “All quiet on the western front” should be in the same group as webpage as “Storm of steel”.
- ④ Hope: All such webpages of interest in same cluster as **X**, if the clustering is good.

# Clustering

- ① ***unsupervised learning***: Given examples, partition them into classes of similar examples.
- ② Example: Given webpage **X** about “The reality dysfunction”, find all webpages on this topic (or closely related topics).
- ③ Webpage about “All quiet on the western front” should be in the same group as webpage as “Storm of steel”.
- ④ Hope: All such webpages of interest in same cluster as **X**, if the clustering is good.

# Clustering

- ① ***unsupervised learning***: Given examples, partition them into classes of similar examples.
- ② Example: Given webpage  $X$  about “The reality dysfunction”, find all webpages on this topic (or closely related topics).
- ③ Webpage about “All quiet on the western front” should be in the same group as webpage as “Storm of steel”.
- ④ Hope: All such webpages of interest in same cluster as  $X$ , if the clustering is good.

# Clustering – similarity measure

- 1 Input: A set of examples (points in high dim).
- 2 Example:
  - 1 Webpage  $W$ :  $i$ th coordinate to 1 if the word  $w_i$  appears in  $W$ . We have 10,000 words care about.  $W$  interpreted as a point  $\in \{0, 1\}^{10,000}$ .
  - 2 Let  $X$  be the resulting set of  $n$  points in  $d$  dimensions.
- 3 Need similarity measure.
- 4 For example, Euclidean distance between points, where

$$\|p - q\| = \sqrt{\sum_{i=1}^d (p_i - q_i)^2},$$

where  $p = (p_1, \dots, p_d)$  and  $q = (q_1, \dots, q_d)$ .

# Clustering – similarity measure

- ① Input: A set of examples (points in high dim).
- ② Example:
  - ① Webpage  $W$ :  $i$ th coordinate to  $1$  if the word  $w_i$  appears in  $W$ .  
We have **10,000** words care about.  
 $W$  interpreted as a point  $\in \{0, 1\}^{10,000}$ .
  - ② Let  $X$  be the resulting set of  $n$  points in  $d$  dimensions.
- ③ Need similarity measure.
- ④ For example, Euclidean distance between points, where

$$\|p - q\| = \sqrt{\sum_{i=1}^d (p_i - q_i)^2},$$

where  $p = (p_1, \dots, p_d)$  and  $q = (q_1, \dots, q_d)$ .

# Clustering – similarity measure

- ① Input: A set of examples (points in high dim).
- ② Example:
  - ① Webpage  $W$ :  $i$ th coordinate to  $1$  if the word  $w_i$  appears in  $W$ .  
We have **10,000** words care about.  
 $W$  interpreted as a point  $\in \{0, 1\}^{10,000}$ .
  - ② Let  $X$  be the resulting set of  $n$  points in  $d$  dimensions.
- ③ Need similarity measure.
- ④ For example, Euclidean distance between points, where

$$\|p - q\| = \sqrt{\sum_{i=1}^d (p_i - q_i)^2},$$

where  $p = (p_1, \dots, p_d)$  and  $q = (q_1, \dots, q_d)$ .

# Clustering – similarity measure

- ① Input: A set of examples (points in high dim).
- ② Example:
  - ① Webpage  $W$ :  $i$ th coordinate to  $1$  if the word  $w_i$  appears in  $W$ .  
We have **10,000** words care about.  
 $W$  interpreted as a point  $\in \{0, 1\}^{10,000}$ .
  - ② Let  $X$  be the resulting set of  $n$  points in  $d$  dimensions.
- ③ Need similarity measure.
- ④ For example, Euclidean distance between points, where

$$\|p - q\| = \sqrt{\sum_{i=1}^d (p_i - q_i)^2},$$

where  $p = (p_1, \dots, p_d)$  and  $q = (q_1, \dots, q_d)$ .



# Clustering – k center clustering

## k center clustering problem

$P$ : set of  $n$  cities, and distances between them.

Build  $k$  hospitals, s.t. max dist city from its closest hospital is min.

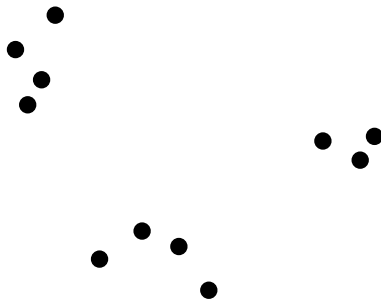
# Clustering – k center clustering

## k center clustering problem

$P$ : set of  $n$  cities, and distances between them.

Build  $k$  hospitals, s.t. max dist city from its closest hospital is min.

Example:  $k = 3$



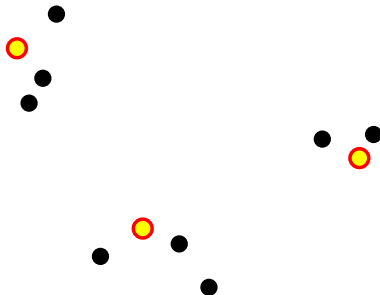
# Clustering – k center clustering

## k center clustering problem

$P$ : set of  $n$  cities, and distances between them.

Build  $k$  hospitals, s.t. max dist city from its closest hospital is min.

Example:  $k = 3$



# Clustering – price

- ① *price* of clustering of  $P$  by  $S$  is

$$\nu(P, S) = \max_{p \in P} d(p, S)$$

- ② *k-center* problem.

- ① Find  $S \subseteq P$  s.t.  $|S| = k$  and  $\nu(P, S)$  minimized.
- ② Equivalently, find  $k$  smallest discs centered at input points...
- ③ ... cover all the points of  $P$ .

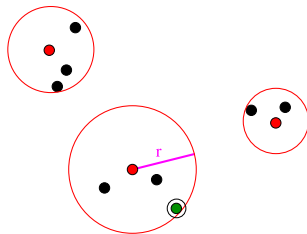
# Clustering – price

- ① **price** of clustering of  $P$  by  $S$  is

$$\nu(P, S) = \max_{p \in P} d(p, S)$$

- ②  **$k$ -center** problem.

- ① Find  $S \subseteq P$  s.t.  $|S| = k$  and  $\nu(P, S)$  minimized.
- ② Equivalently, find  $k$  smallest discs centered at input points...
- ③ ... cover all the points of  $P$ .



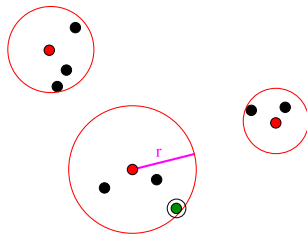
# Clustering – price

- ① **price** of clustering of  $P$  by  $S$  is

$$\nu(P, S) = \max_{p \in P} d(p, S)$$

- ②  **$k$ -center** problem.

- ① Find  $S \subseteq P$  s.t.  $|S| = k$  and  $\nu(P, S)$  minimized.
- ② Equivalently, find  $k$  smallest discs centered at input points...
- ③ ... cover all the points of  $P$ .



# k Center Clustering

- ①  $k$ -center clustering is **NP-Hard**...
- ② ...even to approximate within a factor of (roughly) **1.8**.
- ③ Formal definition...

## $k$ -center clustering

**Instance:** A set  $P$  of  $n$  points, a distance function  $d(p, q)$ , for  $p, q \in P$ , satisfying the triangle inequality, and a parameter  $k$ .

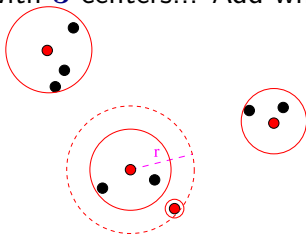
**Question:** Find the subset  $S$  that realizes

$$r_{opt}(P, k) = \min_{S \subseteq P, |S|=k} \nu(P, S),$$

$$\text{where } \nu(P, S) = \max_{p \in P} d(p, S)$$

# k-center clustering - approximation

- ① Current solution with **3** centers... Add which center?

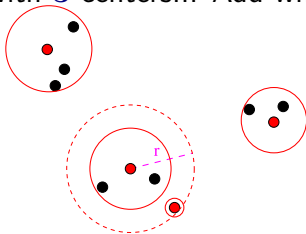


- ② ...use bottleneck point.
- ③ Point furthest away from centers.
- ④ Find a new center that better serves this bottleneck point.
- ⑤ ...make it the next center.



# k-center clustering - approximation

- ① Current solution with 3 centers... Add which center?



- ② ...use bottleneck point.  
③ Point furthest away from centers.  
④ Find a new center that better serves this bottleneck point.  
⑤ ...make it the next center.

# k-center clustering - approximation algorithm

**AprxKCenter**( $P$ ,  $k$ )

$P = \{p_1, \dots, p_n\}$

$S = \{p_1\}$ ,  $u_1 \leftarrow p_1$

**while**  $|S| < k$  **do**

$i \leftarrow |S|$

**for**  $j = 1 \dots n$  **do**

$d_j \leftarrow \min(d_j, d(p_j, u_i))$

$r_{i+1} \leftarrow \max(d_1, \dots, d_n)$

$u_{i+1} \leftarrow$  point of  $P$  realizing  $r_i$

$S \leftarrow S \cup \{u_{i+1}\}$

**return**  $S$

# k-center clustering - approximation algorithm

- ① Running time of **AprxKCenter** is  $O(nk)$
- ②  $r_{i+1}$ : the (minimum) radius of the  $i$  balls centered at  $u_1, \dots, u_i$  covering  $P$ .
- ③  $\exists p \in P: d(p, \{u_1, \dots, u_i\}) = r_{i+1}$ .
- ④ Imagine run **AprxKCenter** one additional iteration.  
... so  $r_{k+1}$  is well defined.

# k-center clustering approximation algorithm

## Analysis

### Lemma

$$r_2 \geq \dots \geq r_k \geq r_{k+1}.$$

### Proof...

### Observation

*The radius of the clustering generated by **AprxKCenter** is  $r_{k+1}$ .*

# k-center clustering approximation algorithm

## Analysis – continued

### Lemma

$r_{k+1} \leq 2r_{opt}(P, k)$ .     $r_{opt}(P, k)$ : radius of the opt with  $k$  balls.

### Proof:

- ①  $D_1, \dots, D_k$ :  $k$  discs in opt sol.
- ②  $S$ :  $k$  centers computed by **AprxKCenter**.
- ③ Suppose every disk  $D_i$  contains at least one point of  $S$ ...
- ④ Then  $\forall p \in P$  distance to  $S$  is  $\leq 2r_{opt}(P, k)$ . That is,

$$d(p, u) \leq d(p, q) + d(q, u) \leq 2r_{opt}$$

# k-center clustering approximation algorithm

## Analysis – continued

### Lemma

$r_{k+1} \leq 2r_{opt}(P, k)$ .     $r_{opt}(P, k)$ : radius of the opt with  $k$  balls.

### Proof:

- ①  $D_1, \dots, D_k$ :  $k$  discs in opt sol.
- ②  $S$ :  $k$  centers computed by **AprxKCenter**.
- ③ Suppose every disk  $D_i$  contains at least one point of  $S$ ...
- ④ Then  $\forall p \in P$  distance to  $S$  is  $\leq 2r_{opt}(P, k)$ . That is,

$$d(p, u) \leq d(p, q) + d(q, u) \leq 2r_{opt}$$

# k-center clustering approximation algorithm

## Analysis – continued

### Lemma

$r_{k+1} \leq 2r_{opt}(P, k)$ .  $r_{opt}(P, k)$ : radius of the opt with  $k$  balls.

### Proof:

- ①  $D_1, \dots, D_k$ :  $k$  discs in opt sol.
- ②  $S$ :  $k$  centers computed by **AprxKCenter**.
- ③ Suppose every disk  $D_i$  contains at least one point of  $S$ ...
- ④ Then  $\forall p \in P$  distance to  $S$  is  $\leq 2r_{opt}(P, k)$ . That is,

$$d(p, u) \leq d(p, q) + d(q, u) \leq 2r_{opt}$$

# k-center clustering approximation algorithm

## Analysis – continued

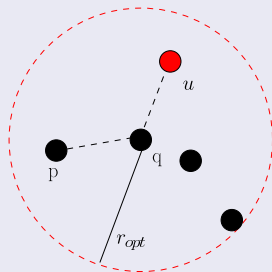
### Lemma

$r_{k+1} \leq 2r_{opt}(P, k)$ .  $r_{opt}(P, k)$ : radius of the opt with  $k$  balls.

### Proof:

- ①  $D_1, \dots, D_k$ :  $k$  discs in opt sol.
- ②  $\mathbf{S}$ :  $k$  centers computed by **AprxKCenter**.
- ③ Suppose every disk  $D_i$  contains at least one point of  $\mathbf{S}$ ...
- ④ Then  $\forall p \in P$  distance to  $\mathbf{S}$  is  $\leq 2r_{opt}(P, k)$ . That is,

$$d(p, u) \leq d(p, q) + d(q, u) \leq 2r_{opt}$$



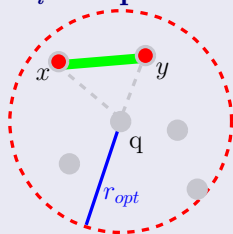


# k-center clustering approximation algorithm

## Analysis – continued

### Proof continued

- ❶ Otherwise,  $\exists x, y \in S$  contained in same ball  $D_i$  of  $\text{Opt}$ .
- ❷ Let  $D_i$  be centered at a point  $q$ .
- ❸ Claim:  $d(x, y) \geq r_{k+1}$ .
- ❹ Suppose  $u_\alpha = x$ ,  $u_\beta = y$ ,  $\alpha < \beta$ .
- ❺  $d(x, y) \geq d(y, \{u_1, \dots, u_{\beta-1}\}) \geq r_\beta$
- ❻ By lemma  $r_\beta \geq r_{k+1}$ .  $\implies$  claim holds
- ❼ By triangle inequality:  
$$r_{k+1} \leq d(x, y) \leq d(x, q) + d(q, y) \leq 2r_{\text{opt}}.$$

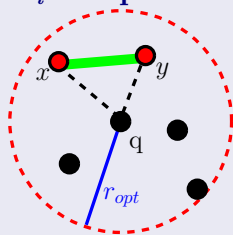


# k-center clustering approximation algorithm

## Analysis – continued

### Proof continued

- ❶ Otherwise,  $\exists x, y \in \mathbf{S}$  contained in same ball  $D_i$  of  $\mathbf{Opt}$ .
- ❷ Let  $D_i$  be centered at a point  $q$ .
- ❸ Claim:  $d(x, y) \geq r_{k+1}$ .
- ❹ Suppose  $u_\alpha = x$ ,  $u_\beta = y$ ,  $\alpha < \beta$ .
- ❺  $d(x, y) \geq d(y, \{u_1, \dots, u_{\beta-1}\}) \geq r_\beta$
- ❻ By lemma  $r_\beta \geq r_{k+1}$ .  $\implies$  claim holds
- ❼ By triangle inequality:  
$$r_{k+1} \leq d(x, y) \leq d(x, q) + d(q, y) \leq 2r_{opt}.$$

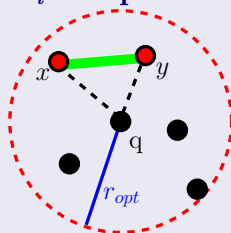


# k-center clustering approximation algorithm

## Analysis – continued

### Proof continued

- ❶ Otherwise,  $\exists x, y \in S$  contained in same ball  $D_i$  of  $\text{Opt}$ .
- ❷ Let  $D_i$  be centered at a point  $q$ .
- ❸ Claim:  $d(x, y) \geq r_{k+1}$ .
- ❹ Suppose  $u_\alpha = x$ ,  $u_\beta = y$ ,  $\alpha < \beta$ .
- ❺  $d(x, y) \geq d(y, \{u_1, \dots, u_{\beta-1}\}) \geq r_\beta$
- ❻ By lemma  $r_\beta \geq r_{k+1}$ .  $\implies$  claim holds
- ❼ By triangle inequality:  
$$r_{k+1} \leq d(x, y) \leq d(x, q) + d(q, y) \leq 2r_{\text{opt}}.$$

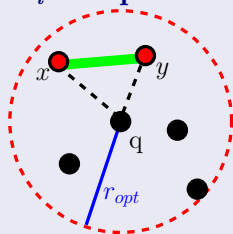


# k-center clustering approximation algorithm

## Analysis – continued

### Proof continued

- ❶ Otherwise,  $\exists x, y \in S$  contained in same ball  $D_i$  of  $\text{Opt}$ .
- ❷ Let  $D_i$  be centered at a point  $q$ .
- ❸ Claim:  $d(x, y) \geq r_{k+1}$ .
- ❹ Suppose  $u_\alpha = x$ ,  $u_\beta = y$ ,  $\alpha < \beta$ .
- ❺  $d(x, y) \geq d(y, \{u_1, \dots, u_{\beta-1}\}) \geq r_\beta$
- ❻ By lemma  $r_\beta \geq r_{k+1}$ .  $\implies$  claim holds
- ❼ By triangle inequality:  
$$r_{k+1} \leq d(x, y) \leq d(x, q) + d(q, y) \leq 2r_{\text{opt}}.$$

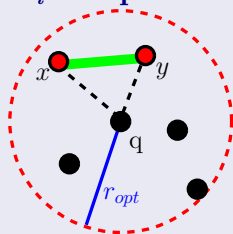


# k-center clustering approximation algorithm

## Analysis – continued

### Proof continued

- ① Otherwise,  $\exists x, y \in S$  contained in same ball  $D_i$  of  $\text{Opt}$ .
- ② Let  $D_i$  be centered at a point  $q$ .
- ③ Claim:  $d(x, y) \geq r_{k+1}$ .
- ④ Suppose  $u_\alpha = x$ ,  $u_\beta = y$ ,  $\alpha < \beta$ .
- ⑤  $d(x, y) \geq d(y, \{u_1, \dots, u_{\beta-1}\}) \geq r_\beta$
- ⑥ By lemma  $r_\beta \geq r_{k+1}$ .  $\implies$  claim holds
- ⑦ By triangle inequality:  
$$r_{k+1} \leq d(x, y) \leq d(x, q) + d(q, y) \leq 2r_{\text{opt}}.$$

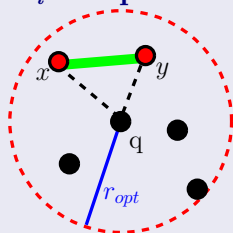


# k-center clustering approximation algorithm

## Analysis – continued

### Proof continued

- ① Otherwise,  $\exists x, y \in \mathbf{S}$  contained in same ball  $D_i$  of  $\mathbf{Opt}$ .
- ② Let  $D_i$  be centered at a point  $q$ .
- ③ Claim:  $d(x, y) \geq r_{k+1}$ .
- ④ Suppose  $u_\alpha = x$ ,  $u_\beta = y$ ,  $\alpha < \beta$ .
- ⑤  $d(x, y) \geq d(y, \{u_1, \dots, u_{\beta-1}\}) \geq r_\beta$
- ⑥ By lemma  $r_\beta \geq r_{k+1}$ .  $\implies$  claim holds
- ⑦ By triangle inequality:  
$$r_{k+1} \leq d(x, y) \leq d(x, q) + d(q, y) \leq 2r_{opt}.$$



# $k$ -center clustering approximation algorithm

## Theorem

*One can approximate the  $k$ -center clustering up to a factor of two, in time  $O(nk)$ .*

## Proof.

**AprxKCenter**: approximation algorithm.

The approximation quality guarantee follows from the above lemma, since the furthest point of  $P$  from the  $k$ -centers computed is  $r_{k+1}$ , which is guaranteed to be at most  $2r_{opt}$ .  $\square$

# Part III

## Subset Sum



# Subset Sum

## Subset Sum

**Instance:**  $X = \{x_1, \dots, x_n\}$  –  $n$  integer positive numbers,  
 $t$  – target number

**Question:**  $\exists$  subset of  $X$  s.t. sum of its elements is  $t$ ?

Assume  $x_1, \dots, x_n$  are all  $\leq n$ . Then this problem can be solved in

- (A) The problem is still **NP-Hard**, so probably exponential time.
- (B)  $O(n^3)$ .
- (C)  $2^{O(\log^2 n)}$ .
- (D)  $O(n \log n)$ .
- (E) None of the above.

# Subset Sum

## Subset Sum

**Instance:**  $X = \{x_1, \dots, x_n\}$  –  $n$  integer positive numbers,  
 $t$  – target number

**Question:**  $\exists$  subset of  $X$  s.t. sum of its elements is  $t$ ?

$M$ : Max value  
input numbers.

```
SolveSubsetSum ( $X, t, M$ )  
     $b[0 \dots Mn]$  – boolean array init to false.  
    //  $b[x]$  is true if  $x$  can be realized  
    // by a subset of  $X$ .  
     $b[0] \leftarrow \text{true}$ .  
    for  $i = 1, \dots, n$  do  
        for  $j = Mn$  down to  $x_i$  do  
             $b[j] \leftarrow b[j] \vee b[j - x_i]$   
  
    return  $b[t]$ 
```

# Subset Sum

## Subset Sum

**Instance:**  $X = \{x_1, \dots, x_n\}$  –  $n$  integer positive numbers,  
 $t$  – target number

**Question:**  $\exists$  subset of  $X$  s.t. sum of its elements is  $t$ ?

$M$ : Max value  
input numbers.

```
SolveSubsetSum ( $X, t, M$ )  
     $b[0 \dots Mn]$  – boolean array init to false.  
    //  $b[x]$  is true if  $x$  can be realized  
    // by a subset of  $X$ .  
     $b[0] \leftarrow \text{true}$ .  
    for  $i = 1, \dots, n$  do  
        for  $j = Mn$  down to  $x_i$  do  
             $b[j] \leftarrow b[j - x_i] \vee b[j]$   
  
    return  $b[t]$ 
```

# Subset Sum

## Subset Sum

**Instance:**  $X = \{x_1, \dots, x_n\}$  –  $n$  integer positive numbers,  
 $t$  – target number

**Question:**  $\exists$  subset of  $X$  s.t. sum of its elements is  $t$ ?

$M$ : Max value  
input numbers.

```
SolveSubsetSum ( $X, t, M$ )  
     $b[0 \dots Mn]$  – boolean array init to false.  
    //  $b[x]$  is true if  $x$  can be realized  
    // by a subset of  $X$ .  
     $b[0] \leftarrow \mathbf{true}$ .  
    for  $i = 1, \dots, n$  do  
        for  $j = Mn$  down to  $x_i$  do  
             $b[j] \leftarrow B[j - x_i] \vee B[j]$   
  
    return  $B[t]$ 
```

# Subset Sum

## Subset Sum

**Instance:**  $X = \{x_1, \dots, x_n\}$  –  $n$  integer positive numbers,  
 $t$  – target number

**Question:**  $\exists$  subset of  $X$  s.t. sum of its elements is  $t$ ?

$M$ : Max value  
input numbers.

R.T.  $O(Mn^2)$ .

**SolveSubsetSum** ( $X, t, M$ )

$b[0 \dots Mn]$  – boolean array init to **false**.  
//  $b[x]$  is **true** if  $x$  can be realized  
// by a subset of  $X$ .

$b[0] \leftarrow \mathbf{true}$ .

**for**  $i = 1, \dots, n$  **do**

**for**  $j = Mn$  down to  $x_i$  **do**  
         $b[j] \leftarrow b[j - x_i] \vee b[j]$

**return**  $b[t]$

# Subset Sum

Efficient algorithm???

- 1 Algorithm solving **Subset Sum** in  $O(Mn^2)$ .
- 2  $M$  might be prohibitly large...
- 3 if  $M = 2^n \implies$  algorithm is not polynomial time.
- 4 **Subset Sum** is **NPC**.
- 5 Still want to solve quickly even if  $M$  huge.
- 6 Optimization version:

## Subset Sum Optimization

**Instance:**  $(X, t)$ : A set  $X$  of  $n$  positive integers, and a target number  $t$ .

**Question:** The largest number  $\gamma_{\text{opt}}$  one can represent as a subset sum of  $X$  which is smaller or equal to  $t$ .

# Subset Sum

Efficient algorithm???

- 1 Algorithm solving **Subset Sum** in  $O(Mn^2)$ .
- 2  $M$  might be prohibitly large...
- 3 if  $M = 2^n \implies$  algorithm is not polynomial time.
- 4 **Subset Sum** is **NPC**.
- 5 Still want to solve quickly even if  $M$  huge.
- 6 Optimization version:

## Subset Sum Optimization

**Instance:**  $(X, t)$ : A set  $X$  of  $n$  positive integers, and a target number  $t$ .

**Question:** The largest number  $\gamma_{\text{opt}}$  one can represent as a subset sum of  $X$  which is smaller or equal to  $t$ .

# Subset Sum

Efficient algorithm???

- 1 Algorithm solving **Subset Sum** in  $O(Mn^2)$ .
- 2  $M$  might be prohibitly large...
- 3 if  $M = 2^n \implies$  algorithm is not polynomial time.
- 4 **Subset Sum** is **NPC**.
- 5 Still want to solve quickly even if  $M$  huge.
- 6 Optimization version:

## Subset Sum Optimization

**Instance:**  $(X, t)$ : A set  $X$  of  $n$  positive integers, and a target number  $t$ .

**Question:** The largest number  $\gamma_{\text{opt}}$  one can represent as a subset sum of  $X$  which is smaller or equal to  $t$ .



# Subset Sum

## 2-approximation

### Lemma

- ①  $(X, t)$ ; Given instance of **Subset Sum**.  $\gamma_{\text{opt}} \leq t$ : Opt.
- ②  $\implies$  Compute legal subset with sum  $\geq \gamma_{\text{opt}}/2$ .
- ③ Running time  $O(n \log n)$ .

### Proof.

- ① Sort numbers in  $X$  in decreasing order.
- ② Greedily - add numbers from largest to smallest (if possible).
- ③  $s$ : Generates sum.
- ④  $u$ : First rejected number.  $s'$ : sum before rejection.
- ⑤  $s' > u > 0$ ,  $s' < t$ , and  $s' + u > t \implies$   
 $t < s' + u < s' + s' = 2s' \implies s' \geq t/2$ .

# Subset Sum

## 2-approximation

### Lemma

- ①  $(X, t)$ ; Given instance of **Subset Sum**.  $\gamma_{\text{opt}} \leq t$ : Opt.
- ②  $\implies$  Compute legal subset with sum  $\geq \gamma_{\text{opt}}/2$ .
- ③ Running time  $O(n \log n)$ .

### Proof.

- ① Sort numbers in  $X$  in decreasing order.
- ② Greedily - add numbers from largest to smallest (if possible).
- ③  $s$ : Generates sum.
- ④  $u$ : First rejected number.  $s'$ : sum before rejection.
- ⑤  $s' > u > 0$ ,  $s' < t$ , and  $s' + u > t \implies$   
 $t < s' + u < s' + s' = 2s' \implies s' \geq t/2$ .

# Subset Sum

## 2-approximation

### Lemma

- ①  $(X, t)$ ; Given instance of **Subset Sum**.  $\gamma_{\text{opt}} \leq t$ : Opt.
- ②  $\implies$  Compute legal subset with sum  $\geq \gamma_{\text{opt}}/2$ .
- ③ Running time  $O(n \log n)$ .

### Proof.

- ① Sort numbers in  $X$  in decreasing order.
- ② Greedily - add numbers from largest to smallest (if possible).
- ③  $s$ : Generates sum.
- ④  $u$ : First rejected number.  $s'$ : sum before rejection.
- ⑤  $s' > u > 0$ ,  $s' < t$ , and  $s' + u > t \implies$   
 $t < s' + u < s' + s' = 2s' \implies s' \geq t/2$ .

# Subset Sum

## 2-approximation

### Lemma

- ①  $(X, t)$ ; Given instance of **Subset Sum**.  $\gamma_{\text{opt}} \leq t$ : Opt.
- ②  $\implies$  Compute legal subset with sum  $\geq \gamma_{\text{opt}}/2$ .
- ③ Running time  $O(n \log n)$ .

### Proof.

- ① Sort numbers in  $X$  in decreasing order.
- ② Greedily - add numbers from largest to smallest (if possible).
- ③  $s$ : Generates sum.
- ④  $u$ : First rejected number.  $s'$ : sum before rejection.
- ⑤  $s' > u > 0$ ,  $s' < t$ , and  $s' + u > t \implies$   
 $t < s' + u < s' + s' = 2s' \implies s' \geq t/2$ .

# Subset Sum

## 2-approximation

### Lemma

- ①  $(X, t)$ ; Given instance of **Subset Sum**.  $\gamma_{\text{opt}} \leq t$ : Opt.
- ②  $\implies$  Compute legal subset with sum  $\geq \gamma_{\text{opt}}/2$ .
- ③ Running time  $O(n \log n)$ .

### Proof.

- ① Sort numbers in  $X$  in decreasing order.
- ② Greedily - add numbers from largest to smallest (if possible).
- ③  $s$ : Generates sum.
- ④  $u$ : First rejected number.  $s'$ : sum before rejection.
- ⑤  $s' > u > 0$ ,  $s' < t$ , and  $s' + u > t \implies$   
 $t < s' + u < s' + s' = 2s' \implies s' \geq t/2$ .

# Polynomial Time Approximation Schemes

## Definition (PTAS)

**PROB**: Maximization problem.

$\epsilon > 0$ : approximation parameter.

$\mathcal{A}(I, \epsilon)$  is a *polynomial time approximation scheme* (**PTAS**) for **PROB**:

- ①  $\forall I: (1 - \epsilon) |\mathbf{opt}(I)| \leq |\mathcal{A}(I, \epsilon)| \leq |\mathbf{opt}(I)|,$
- ②  $|\mathbf{opt}(I)|$ : opt price,
- ③  $|\mathcal{A}(I, \epsilon)|$ : price of solution of  $\mathcal{A}$ .
- ④  $\mathcal{A}$  running time polynomial in  $n$  for fixed  $\epsilon$ .

For minimization problem:

$$|\mathbf{opt}(I)| \leq |\mathcal{A}(I, \epsilon)| \leq (1 + \epsilon) |\mathbf{opt}(I)|.$$

# Polynomial Time Approximation Schemes

## Definition (PTAS)

**PROB**: Maximization problem.

$\varepsilon > 0$ : approximation parameter.

$\mathcal{A}(I, \varepsilon)$  is a *polynomial time approximation scheme* (**PTAS**) for **PROB**:

- ①  $\forall I: (1 - \varepsilon) |\mathbf{opt}(I)| \leq |\mathcal{A}(I, \varepsilon)| \leq |\mathbf{opt}(I)|,$
- ②  $|\mathbf{opt}(I)|$ : opt price,
- ③  $|\mathcal{A}(I, \varepsilon)|$ : price of solution of  $\mathcal{A}$ .
- ④  $\mathcal{A}$  running time polynomial in  $n$  for fixed  $\varepsilon$ .

For minimization problem:

$$|\mathbf{opt}(I)| \leq |\mathcal{A}(I, \varepsilon)| \leq (1 + \varepsilon) |\mathbf{opt}(I)|.$$

# Polynomial Time Approximation Schemes

- ① Example: Approximation algorithm with running time  $O(n^{1/\epsilon})$  is a **PTAS**.  
Algorithm with running time  $O(1/\epsilon^n)$  is not.
- ② Fully polynomial...

## Definition (FPTAS)

An approximation algorithm is *fully polynomial time approximation scheme* (**FPTAS**) if it is a **PTAS**, and its running time is polynomial both in  $n$  and  $1/\epsilon$ .

- ③ Example: **PTAS** with running time  $O(n^{1/\epsilon})$  is not a **FPTAS**.
- ④ Example: **PTAS** with running time  $O(n^2/\epsilon^3)$  is a **FPTAS**.



# Polynomial Time Approximation Schemes

- ① Example: Approximation algorithm with running time  $O(n^{1/\epsilon})$  is a **PTAS**.  
Algorithm with running time  $O(1/\epsilon^n)$  is not.
- ② Fully polynomial...

## Definition (FPTAS)

An approximation algorithm is **fully polynomial time approximation scheme** (**FPTAS**) if it is a **PTAS**, and its running time is polynomial both in  $n$  and  $1/\epsilon$ .

- ③ Example: **PTAS** with running time  $O(n^{1/\epsilon})$  is not a **FPTAS**.
- ④ Example: **PTAS** with running time  $O(n^2/\epsilon^3)$  is a **FPTAS**.

# Polynomial Time Approximation Schemes

- ① Example: Approximation algorithm with running time  $O(n^{1/\epsilon})$  is a **PTAS**.  
Algorithm with running time  $O(1/\epsilon^n)$  is not.
- ② Fully polynomial...

## Definition (FPTAS)

An approximation algorithm is **fully polynomial time approximation scheme** (**FPTAS**) if it is a **PTAS**, and its running time is polynomial both in  $n$  and  $1/\epsilon$ .

- ③ Example: **PTAS** with running time  $O(n^{1/\epsilon})$  is not a **FPTAS**.
- ④ Example: **PTAS** with running time  $O(n^2/\epsilon^3)$  is a **FPTAS**.

# Polynomial Time Approximation Schemes

- ① Example: Approximation algorithm with running time  $O(n^{1/\epsilon})$  is a **PTAS**.  
Algorithm with running time  $O(1/\epsilon^n)$  is not.
- ② Fully polynomial...

## Definition (FPTAS)

An approximation algorithm is **fully polynomial time approximation scheme** (**FPTAS**) if it is a **PTAS**, and its running time is polynomial both in  $n$  and  $1/\epsilon$ .

- ③ Example: **PTAS** with running time  $O(n^{1/\epsilon})$  is not a **FPTAS**.
- ④ Example: **PTAS** with running time  $O(n^2/\epsilon^3)$  is a **FPTAS**.

# Approximating Subset Sum

## Subset Sum Approx

**Instance:**  $(X, t, \varepsilon)$ : A set  $X$  of  $n$  positive integers, a target number  $t$ , and parameter  $\varepsilon > 0$ .

**Question:** A number  $z$  that one can represent as a subset sum of  $X$ , such that  $(1 - \varepsilon)\gamma_{\text{opt}} \leq z \leq \gamma_{\text{opt}} \leq t$ .

# Approximating Subset Sum

Looking again at the exact algorithm

**ExactSubsetSum**( $S$ ,  $t$ )

$n \leftarrow |S|$

$P_0 \leftarrow \{0\}$

**for**  $i = 1 \dots n$  **do**

$P_i \leftarrow P_{i-1} \cup (P_{i-1} + x_i)$

Remove from  $P_i$  all elements  $> t$

**return** largest element in  $P_n$

①  $S = \{a_1, \dots, a_n\}$

$$x + S = \{a_1 + x, a_2 + x, \dots, a_n + x\}$$

② Lists might explode in size.

# Trim the lists...

```
Trim( $L', \delta$ )  
   $L \leftarrow \text{Sort}(L')$   
   $L = \langle y_1 \dots y_m \rangle$   
   $\text{curr} \leftarrow y_1$   
   $L_{\text{out}} \leftarrow \{y_1\}$   
  for  $i = 2 \dots m$  do  
    if  $y_i > \text{curr} \cdot (1 + \delta)$   
      Append  $y_i$  to  $L_{\text{out}}$   
       $\text{curr} \leftarrow y_i$   
  return  $L_{\text{out}}$ 
```

## Definition

For two positive real numbers  $z \leq y$ , the number  $y$  is a  $\delta$ -approximation to  $z$  if

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

## Observation

If  $x \in L'$  then there exists a number  $y \in L_{\text{out}}$  such that  $y \leq x \leq y(1 + \delta)$ , where  $L_{\text{out}} \leftarrow \text{Trim}(L', \delta)$ .

# Trim the lists...

**Trim**( $L', \delta$ )

$L \leftarrow \text{Sort}(L')$

$L = \langle y_1 \dots y_m \rangle$

$\text{curr} \leftarrow y_1$

$L_{\text{out}} \leftarrow \{y_1\}$

**for**  $i = 2 \dots m$  **do**

**if**  $y_i > \text{curr} \cdot (1 + \delta)$

        Append  $y_i$  to  $L_{\text{out}}$

$\text{curr} \leftarrow y_i$

**return**  $L_{\text{out}}$

**ApproxSubsetSum**( $S, t$ )

//  $S = \{x_1, \dots, x_n\}$ ,

//  $x_1 \leq x_2 \leq \dots \leq x_n$

$n \leftarrow |S|$ ,  $L_0 \leftarrow \{0\}$ ,  $\delta = \varepsilon/2n$

**for**  $i = 1 \dots n$  **do**

$E_i \leftarrow L_{i-1} \cup (L_{i-1} + x_i)$

$L_i \leftarrow \text{Trim}(E_i, \delta)$

    Remove from  $L_i$  elems  $> t$ .

**return** largest element in  $L_n$

# Analysis

- ①  $E_i$  list generated by algorithm in  $i$ th iteration.
- ②  $P_i$ : list of numbers (no trimming).

## Claim

For any  $x \in P_i$  there exists  $y \in L_i$  such that  $y \leq x \leq (1 + \delta)^i y$ .

## Proof

- ① If  $x \in P_1$  then follows by observation above.
- ② If  $x \in P_{i-1} \implies$  (induction)  $\exists y' \in L_{i-1}$  s.t.  
 $y' \leq x \leq (1 + \delta)^{i-1} y'$ .
- ③ By observation  $\exists y \in L_i$  s.t.  $y \leq y' \leq (1 + \delta)y$ , As such,

$$y \leq y' \leq x \leq (1 + \delta)^{i-1} y' \leq (1 + \delta)^i y.$$



# Analysis

- ①  $E_i$  list generated by algorithm in  $i$ th iteration.
- ②  $P_i$ : list of numbers (no trimming).

## Claim

For any  $x \in P_i$  there exists  $y \in L_i$  such that  $y \leq x \leq (1 + \delta)^i y$ .

## Proof

- ① If  $x \in P_1$  then follows by observation above.
- ② If  $x \in P_{i-1} \implies$  (induction)  $\exists y' \in L_{i-1}$  s.t.  
 $y' \leq x \leq (1 + \delta)^{i-1} y'$ .
- ③ By observation  $\exists y \in L_i$  s.t.  $y \leq y' \leq (1 + \delta)y$ , As such,

$$y \leq y' \leq x \leq (1 + \delta)^{i-1} y' \leq (1 + \delta)^i y.$$

# Analysis

- ①  $E_i$  list generated by algorithm in  $i$ th iteration.
- ②  $P_i$ : list of numbers (no trimming).

## Claim

For any  $x \in P_i$  there exists  $y \in L_i$  such that  $y \leq x \leq (1 + \delta)^i y$ .

## Proof

- ① If  $x \in P_1$  then follows by observation above.
- ② If  $x \in P_{i-1} \implies$  (induction)  $\exists y' \in L_{i-1}$  s.t.  
 $y' \leq x \leq (1 + \delta)^{i-1} y'$ .
- ③ By observation  $\exists y \in L_i$  s.t.  $y \leq y' \leq (1 + \delta)y$ , As such,

$$y \leq y' \leq x \leq (1 + \delta)^{i-1} y' \leq (1 + \delta)^i y.$$

# Proof continued

## Proof continued

- ① If  $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$ , for some  $\alpha \in P_{i-1}$ .
- ② By induction,  $\exists \alpha' \in L_{i-1}$  s.t.  $\alpha' \leq \alpha \leq (1 + \delta)^{i-1} \alpha'$ .
- ③ Thus,  $\alpha' + x_i \in E_i$ .
- ④  $\exists x' \in L_i$  s.t.  $x' \leq \alpha' + x_i \leq (1 + \delta)x'$ .
- ⑤ Thus,  $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1 + \delta)^{i-1} \alpha' + x_i \leq (1 + \delta)^{i-1}(\alpha' + x_i) \leq (1 + \delta)^i x'$ . ■

# Proof continued

## Proof continued

- ① If  $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$ , for some  $\alpha \in P_{i-1}$ .
- ② By induction,  $\exists \alpha' \in L_{i-1}$  s.t.  $\alpha' \leq \alpha \leq (1 + \delta)^{i-1} \alpha'$ .
- ③ Thus,  $\alpha' + x_i \in E_i$ .
- ④  $\exists x' \in L_i$  s.t.  $x' \leq \alpha' + x_i \leq (1 + \delta)x'$ .
- ⑤ Thus,  $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1 + \delta)^{i-1} \alpha' + x_i \leq (1 + \delta)^{i-1}(\alpha' + x_i) \leq (1 + \delta)^i x'$ . ■


# Proof continued

## Proof continued

- ① If  $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$ , for some  $\alpha \in P_{i-1}$ .
- ② By induction,  $\exists \alpha' \in L_{i-1}$  s.t.  $\alpha' \leq \alpha \leq (1 + \delta)^{i-1} \alpha'$ .
- ③ Thus,  $\alpha' + x_i \in E_i$ .
- ④  $\exists x' \in L_i$  s.t.  $x' \leq \alpha' + x_i \leq (1 + \delta)x'$ .
- ⑤ Thus,  $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1 + \delta)^{i-1} \alpha' + x_i \leq (1 + \delta)^{i-1}(\alpha' + x_i) \leq (1 + \delta)^i x'$ . ■

# Proof continued

## Proof continued

- ① If  $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$ , for some  $\alpha \in P_{i-1}$ .
- ② By induction,  $\exists \alpha' \in L_{i-1}$  s.t.  $\alpha' \leq \alpha \leq (1 + \delta)^{i-1} \alpha'$ .
- ③ Thus,  $\alpha' + x_i \in E_i$ .
- ④  $\exists x' \in L_i$  s.t.  $x' \leq \alpha' + x_i \leq (1 + \delta)x'$ .
- ⑤ Thus,  $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1 + \delta)^{i-1} \alpha' + x_i \leq (1 + \delta)^{i-1}(\alpha' + x_i) \leq (1 + \delta)^i x'$ . 


# Proof continued

## Proof continued

- ① If  $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$ , for some  $\alpha \in P_{i-1}$ .
- ② By induction,  $\exists \alpha' \in L_{i-1}$  s.t.  $\alpha' \leq \alpha \leq (1 + \delta)^{i-1} \alpha'$ .
- ③ Thus,  $\alpha' + x_i \in E_i$ .
- ④  $\exists x' \in L_i$  s.t.  $x' \leq \alpha' + x_i \leq (1 + \delta)x'$ .
- ⑤ Thus,  $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1 + \delta)^{i-1} \alpha' + x_i \leq (1 + \delta)^{i-1}(\alpha' + x_i) \leq (1 + \delta)^i x'$ . ■

# Proof continued


## Proof continued

- ① If  $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$ , for some  $\alpha \in P_{i-1}$ .
- ② By induction,  $\exists \alpha' \in L_{i-1}$  s.t.  $\alpha' \leq \alpha \leq (1 + \delta)^{i-1} \alpha'$ .
- ③ Thus,  $\alpha' + x_i \in E_i$ .
- ④  $\exists x' \in L_i$  s.t.  $x' \leq \alpha' + x_i \leq (1 + \delta) x'$ .
- ⑤ Thus,  $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1 + \delta)^{i-1} \alpha' + x_i \leq (1 + \delta)^{i-1} (\alpha' + x_i) \leq (1 + \delta)^i x'$ . 



# Proof continued

## Proof continued

- ① If  $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$ , for some  $\alpha \in P_{i-1}$ .
- ② By induction,  $\exists \alpha' \in L_{i-1}$  s.t.  $\alpha' \leq \alpha \leq (1 + \delta)^{i-1} \alpha'$ .
- ③ Thus,  $\alpha' + x_i \in E_i$ .
- ④  $\exists x' \in L_i$  s.t.  $x' \leq \alpha' + x_i \leq (1 + \delta)x'$ .
- ⑤ Thus,  $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1 + \delta)^{i-1} \alpha' + x_i \leq (1 + \delta)^{i-1}(\alpha' + x_i) \leq (1 + \delta)^i x'$ . 

# Proof continued

## Proof continued

- ① If  $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$ , for some  $\alpha \in P_{i-1}$ .
- ② By induction,  $\exists \alpha' \in L_{i-1}$  s.t.  $\alpha' \leq \alpha \leq (1 + \delta)^{i-1} \alpha'$ .
- ③ Thus,  $\alpha' + x_i \in E_i$ .
- ④  $\exists x' \in L_i$  s.t.  $x' \leq \alpha' + x_i \leq (1 + \delta)x'$ .
- ⑤ Thus,  $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1 + \delta)^{i-1} \alpha' + x_i \leq (1 + \delta)^{i-1} (\alpha' + x_i) \leq (1 + \delta)^i x'$ . ■

# Running time of ApproxSubsetSum

## Lemma

For  $x \in [0, 1]$ , it holds  $\exp(x/2) \leq (1 + x)$ .

## Lemma

For  $0 < \delta < 1$ , and  $x \geq 1$ , we have

$$\log_{1+\delta} x \leq \frac{2 \ln x}{\delta} = O\left(\frac{\ln x}{\delta}\right).$$

See notes for a proof of lemmas.

# Running time of ApproxSubsetSum

## Observation

*In a list generated by **Trim**, for any number  $x$ , there are no two numbers in the trimmed list between  $x$  and  $(1 + \delta)x$ .*

## Lemma

$|L_i| = O\left((n/\varepsilon^2) \log n\right)$ , for  $i = 1, \dots, n$ .

# Running time of ApproxSubsetSum

## Proof.

- ①  $L_{i-1} + x_i \subseteq [x_i, ix_i]$ .
- ② Trimming  $L_{i-1} + x_i$  results in list of size

$$\log_{1+\delta} \frac{ix_i}{x_i} = O\left(\frac{\ln i}{\delta}\right) = O\left(\frac{\ln n}{\delta}\right),$$

- ③ Now,  $\delta = \varepsilon/2n$ , and

$$\begin{aligned} |L_i| &\leq |L_{i-1}| + O\left(\frac{\ln n}{\delta}\right) \leq |L_{i-1}| + O\left(\frac{n \ln n}{\varepsilon}\right) \\ &= O\left(\frac{n^2 \log n}{\varepsilon}\right). \end{aligned}$$



# Running time of ApproxSubsetSum

## Proof.

- ①  $L_{i-1} + x_i \subseteq [x_i, ix_i]$ .
- ② Trimming  $L_{i-1} + x_i$  results in list of size

$$\log_{1+\delta} \frac{ix_i}{x_i} = O\left(\frac{\ln i}{\delta}\right) = O\left(\frac{\ln n}{\delta}\right),$$

- ③ Now,  $\delta = \varepsilon/2n$ , and

$$\begin{aligned} |L_i| &\leq |L_{i-1}| + O\left(\frac{\ln n}{\delta}\right) \leq |L_{i-1}| + O\left(\frac{n \ln n}{\varepsilon}\right) \\ &= O\left(\frac{n^2 \log n}{\varepsilon}\right). \end{aligned}$$

□

# Running time of ApproxSubsetSum

## Lemma

The running time of **ApproxSubsetSum** is  $O\left(\frac{n^3}{\epsilon} \log^2 n\right)$ .

## Proof.

- ① Running time of **ApproxSubsetSum** dominated by total length of  $L_1, \dots, L_n$ .
- ② Above lemma implies  $\sum_i |L_i| = O\left(\frac{n^3}{\epsilon} \log n\right)$ .
- ③ **Trim** sorts lists.  $i$ th iteration R.T.  $O(|L_i| \log |L_i|)$ .
- ④ Overall, R.T.  $O(\sum_i |L_i| |L_i|) = O\left(\frac{n^3}{\epsilon} \log^2 n\right)$ .



# ApproxSubsetSum

## Theorem

**ApproxSubsetSum** returns  $u \leq t$ , s.t.  $\frac{\gamma_{\text{opt}}}{1+\epsilon} \leq u \leq \gamma_{\text{opt}} \leq t$ ,

$\gamma_{\text{opt}}$ : opt solution.

Running time is  $O((n^3/\epsilon) \log^2 n)$ .

## Proof.

- ① Running time from above.
- ②  $\gamma_{\text{opt}} \in P_n$ : optimal solution.
- ③  $\exists z \in L_n$ , such that  $z \leq \text{opt} \leq (1 + \delta)^n z$
- ④  $(1 + \delta)^n = (1 + \epsilon/2n)^n \leq \exp(\frac{\epsilon}{2}) \leq 1 + \epsilon$ , since  $1 + x \leq e^x$  for  $x \geq 0$ .
- ⑤  $\gamma_{\text{opt}}/(1 + \epsilon) \leq z \leq \text{opt} \leq t$ .





# ApproxSubsetSum

## Theorem

**ApproxSubsetSum** returns  $u \leq t$ , s.t.  $\frac{\gamma_{\text{opt}}}{1+\epsilon} \leq u \leq \gamma_{\text{opt}} \leq t$ ,

$\gamma_{\text{opt}}$ : opt solution.

Running time is  $O((n^3/\epsilon) \log^2 n)$ .

## Proof.

- ① Running time from above.
- ②  $\gamma_{\text{opt}} \in P_n$ : optimal solution.
- ③  $\exists z \in L_n$ , such that  $z \leq \text{opt} \leq (1 + \delta)^n z$
- ④  $(1 + \delta)^n = (1 + \epsilon/2n)^n \leq \exp(\frac{\epsilon}{2}) \leq 1 + \epsilon$ , since  $1 + x \leq e^x$  for  $x \geq 0$ .
- ⑤  $\gamma_{\text{opt}}/(1 + \epsilon) \leq z \leq \text{opt} \leq t$ .



# ApproxSubsetSum

## Theorem

**ApproxSubsetSum** returns  $u \leq t$ , s.t.  $\frac{\gamma_{\text{opt}}}{1+\epsilon} \leq u \leq \gamma_{\text{opt}} \leq t$ ,

$\gamma_{\text{opt}}$ : opt solution.

Running time is  $O((n^3/\epsilon) \log^2 n)$ .

## Proof.

- ① Running time from above.
- ②  $\gamma_{\text{opt}} \in P_n$ : optimal solution.
- ③  $\exists z \in L_n$ , such that  $z \leq \text{opt} \leq (1 + \delta)^n z$
- ④  $(1 + \delta)^n = (1 + \epsilon/2n)^n \leq \exp(\frac{\epsilon}{2}) \leq 1 + \epsilon$ , since  $1 + x \leq e^x$  for  $x \geq 0$ .
- ⑤  $\gamma_{\text{opt}}/(1 + \epsilon) \leq z \leq \text{opt} \leq t$ .



# ApproxSubsetSum

## Theorem

**ApproxSubsetSum** returns  $u \leq t$ , s.t.  $\frac{\gamma_{\text{opt}}}{1+\epsilon} \leq u \leq \gamma_{\text{opt}} \leq t$ ,

$\gamma_{\text{opt}}$ : opt solution.

Running time is  $O((n^3/\epsilon) \log^2 n)$ .

## Proof.

- ① Running time from above.
- ②  $\gamma_{\text{opt}} \in P_n$ : optimal solution.
- ③  $\exists z \in L_n$ , such that  $z \leq \text{opt} \leq (1 + \delta)^n z$
- ④  $(1 + \delta)^n = (1 + \epsilon/2n)^n \leq \exp(\frac{\epsilon}{2}) \leq 1 + \epsilon$ , since  $1 + x \leq e^x$  for  $x \geq 0$ .
- ⑤  $\gamma_{\text{opt}}/(1 + \epsilon) \leq z \leq \text{opt} \leq t$ .



# ApproxSubsetSum

## Theorem

**ApproxSubsetSum** returns  $u \leq t$ , s.t.  $\frac{\gamma_{\text{opt}}}{1+\epsilon} \leq u \leq \gamma_{\text{opt}} \leq t$ ,

$\gamma_{\text{opt}}$ : opt solution.

Running time is  $O((n^3/\epsilon) \log^2 n)$ .

## Proof.

- ① Running time from above.
- ②  $\gamma_{\text{opt}} \in P_n$ : optimal solution.
- ③  $\exists z \in L_n$ , such that  $z \leq \text{opt} \leq (1 + \delta)^n z$
- ④  $(1 + \delta)^n = (1 + \epsilon/2n)^n \leq \exp(\frac{\epsilon}{2}) \leq 1 + \epsilon$ , since  $1 + x \leq e^x$  for  $x \geq 0$ .
- ⑤  $\gamma_{\text{opt}}/(1 + \epsilon) \leq z \leq \text{opt} \leq t$ .









