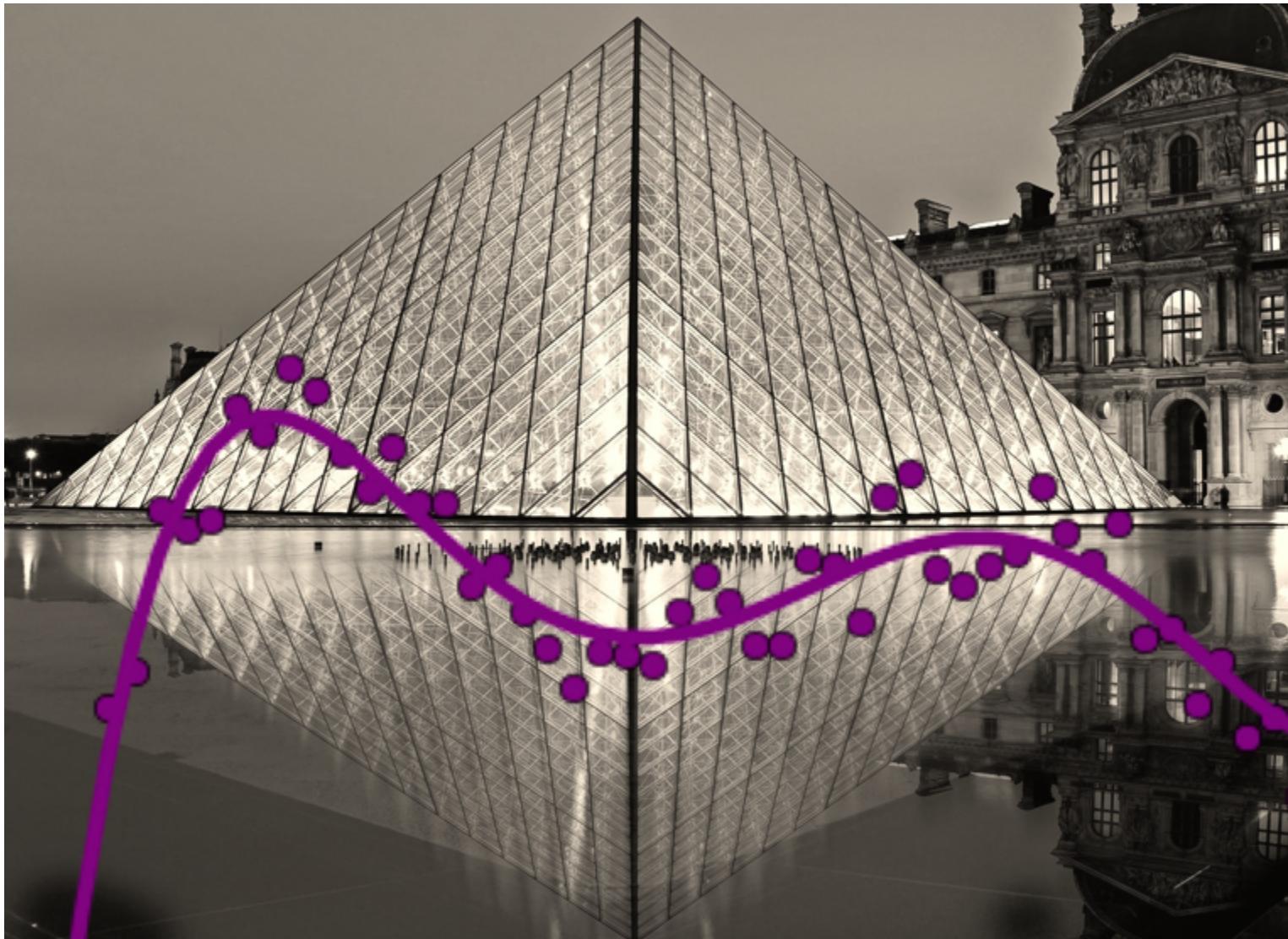


# Approximation Algorithms



# Combinatorial optimization:

## Scheduling classes

## Planning delivery routes for trucks

“  
**Most are NP-hard**

| Lundi  | Mardi  | Mercredi  | Jeudi  | Vendredi   |
|--|--|---|--|--|
| TD proba 1<br>Systèmes dynamiques<br>Topo 1  | Algèbre 1<br>Proba 2<br>Logique  | Topo 1<br>Algèbre 2                                     | TD Topo 1<br>EDP cours<br>TD Algèbre 1<br>TD EDP                                     | Structures et Algorithmes Aléatoires (A Bouillard)<br>8h30-12h15 Cours et TD Salle R<br>Début :  |
| Langages de programmation et compilation (JC Fillatre) 13h15-15h15 cours salle W Début : 28 sept | TD Algèbre 1<br>TD proba 2<br>Système Digital (J. Vuillemin)<br>13h15-17h cours TD (peut-être 2) | TD Statistiques<br>Algèbre 1<br>Statistiques<br>Proba 1 | Langages formels (D. Vergnaud) 13h15-17h cours TD<br>TD Systèmes<br>Dynamiques<br>GL | Algorithmique et programmation (C Mathieu) 13h15-15h15 cours salle UV Début : 15h15-17h00<br>TD 1 Algo salle W<br>TD 2 Compil salle Info 4<br>TD Algèbre 2 |
| 15h15-17h00<br>TD 1 compil<br>salle Info 4<br>TD 2 Algo<br>salle W                               | Algèbre 2<br>salle UV<br>Début : 29-sept   | Modélisation et Simulation Numérique<br>Thé du DMA      | Séminaires DI<br>17h00-19h00<br>salle Henri Cartan                                   |  |
| GT<br>TD Logique   |  |   |  |  |

What's that?



# Dealing with NP-hard problems

- Give up?
- Roll up our sleeves?
- Try something, hoping for luck?
- Do a rough but good enough job

In polynomial time, we find a solution whose value is provably within a "small" factor of the optimal solution

# Designing approximation algorithms

- Real-life problems are too complex
- Study idealized problems
- Theorems: new algorithmic and structural insights



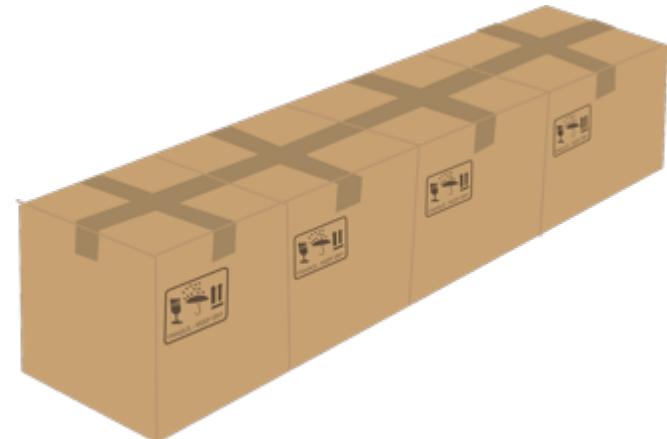
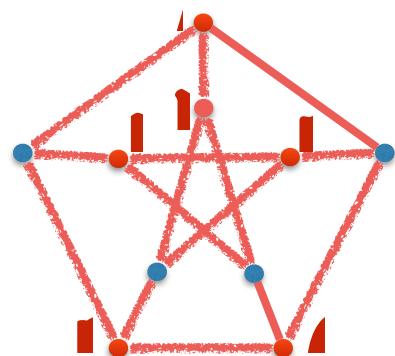
# The interface with real life



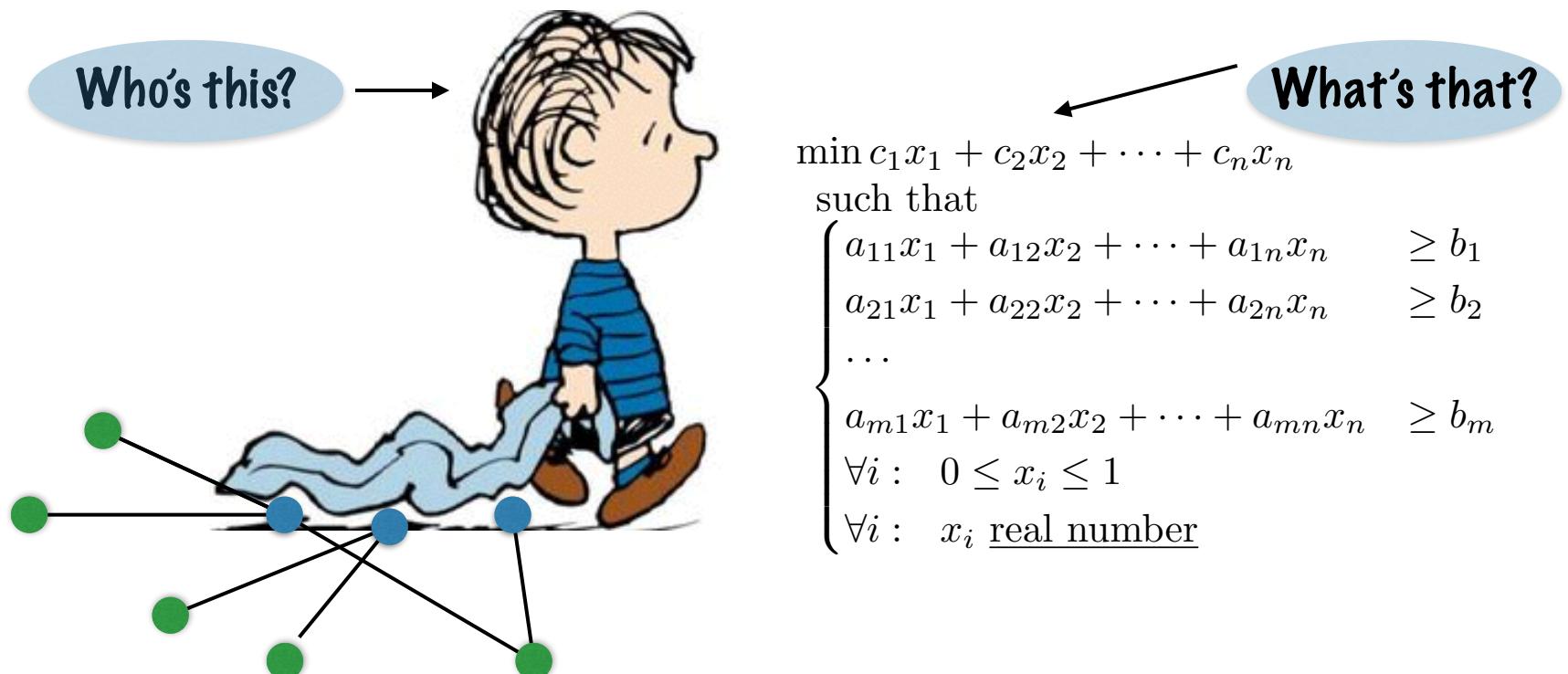
# A course with two parts

## Approximation algorithms, Part I

- Vertex cover
- Knapsack
- Bin packing
- Set cover
- Multiway cut

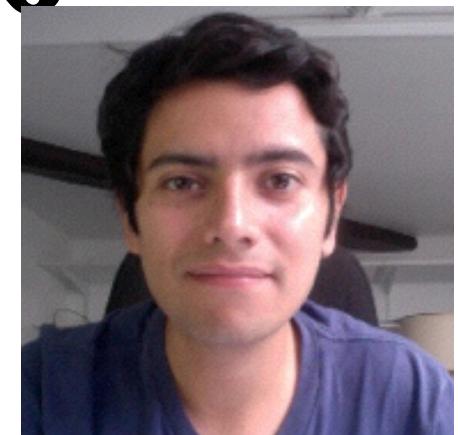


# Approximation algorithms, vertex cover, and linear programming



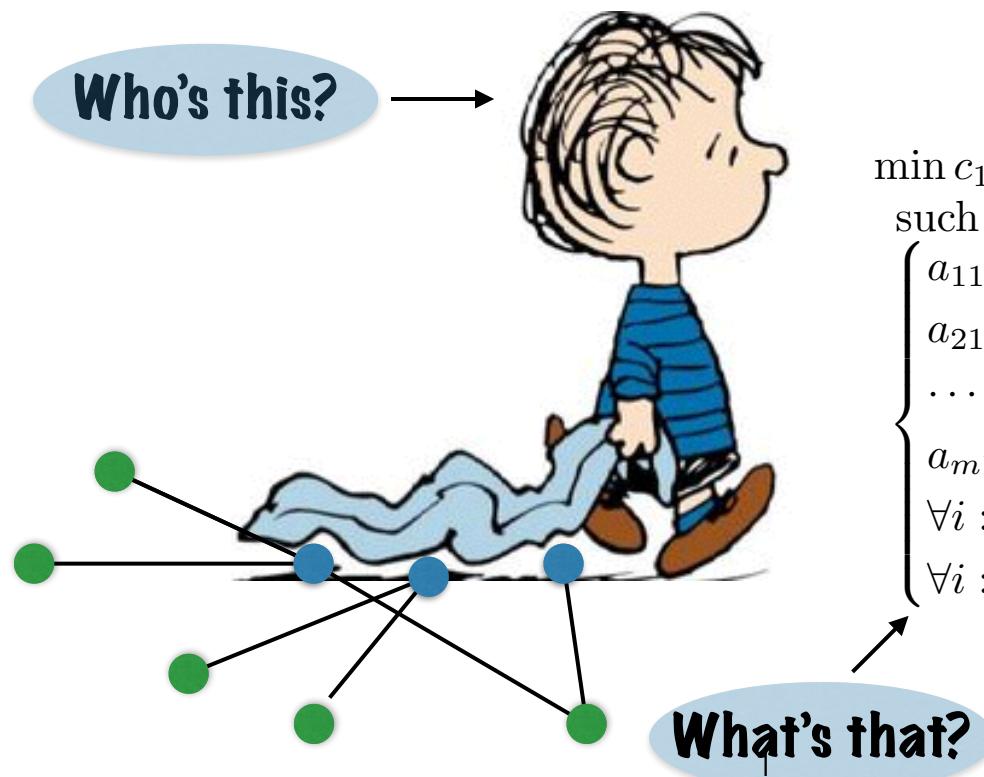
# Teaching staff behind the scenes

- **Vincent Cohen-Addad**
- **Frederik Mallmann-Trenn**
- **Victor Verdugo**



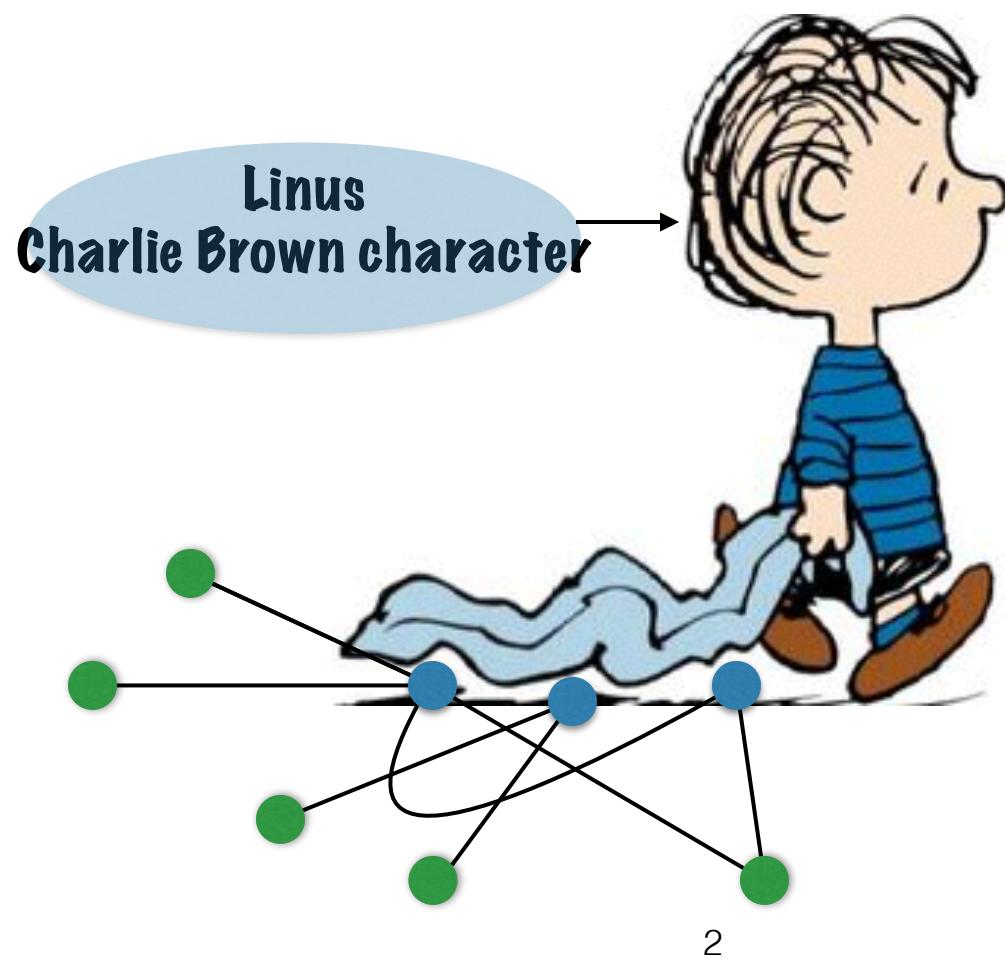
**Film director: Nordine Méziane**  
**Cameraman: Jovanny Parvedy**

# Approximation algorithms, vertex cover, and linear programming



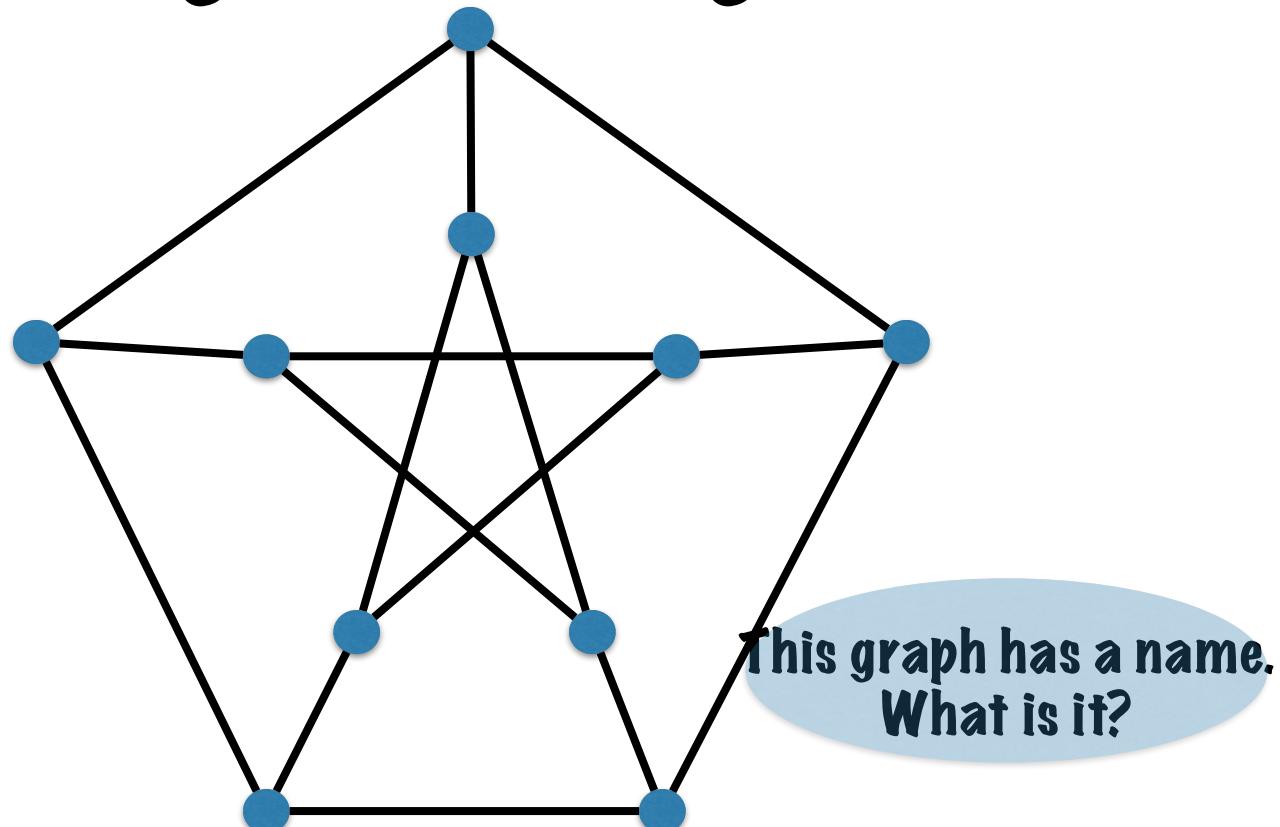
$$\begin{aligned} & \min c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{such that} \quad & \left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \geq b_m \end{array} \right. \\ & \forall i : 0 \leq x_i \leq 1 \\ & \forall i : x_i \text{ real number} \end{aligned}$$

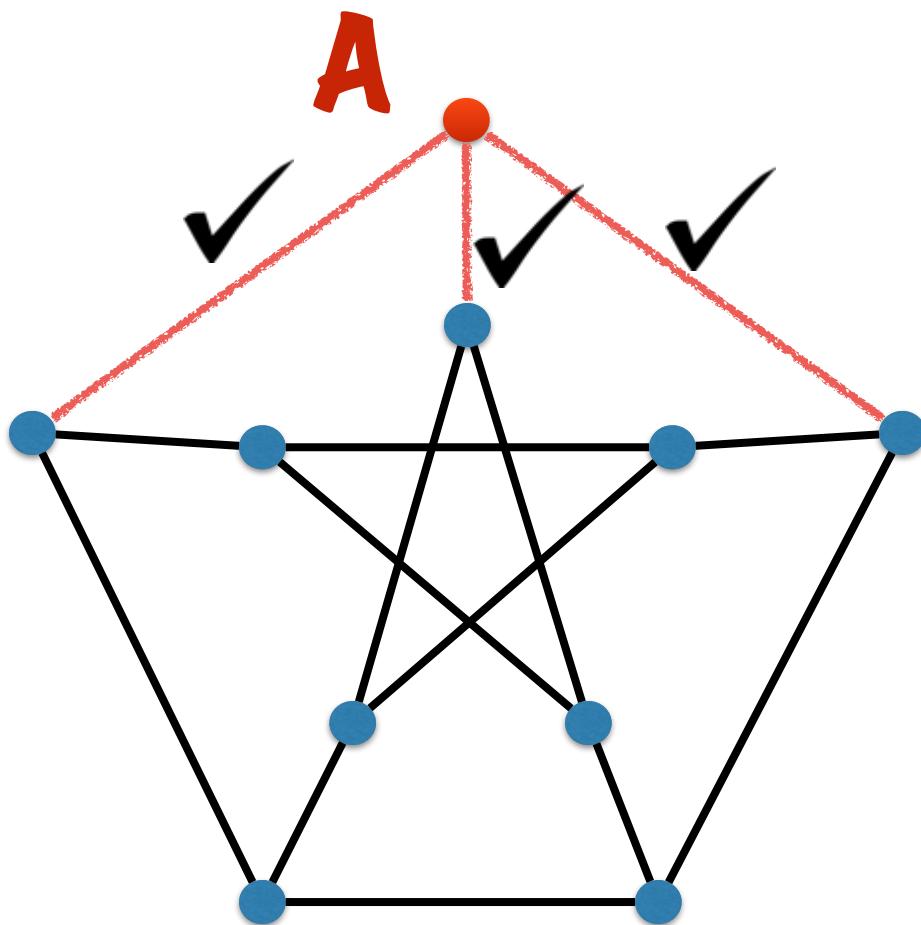
# Vertex Cover

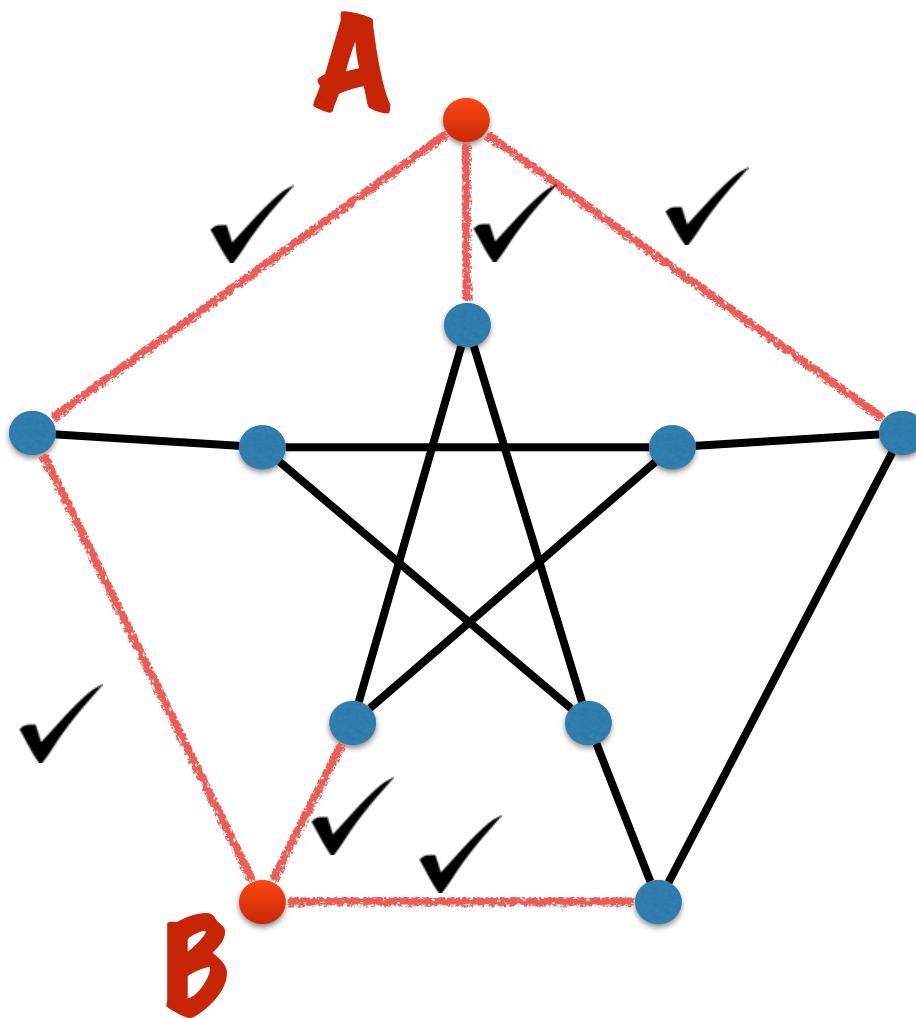


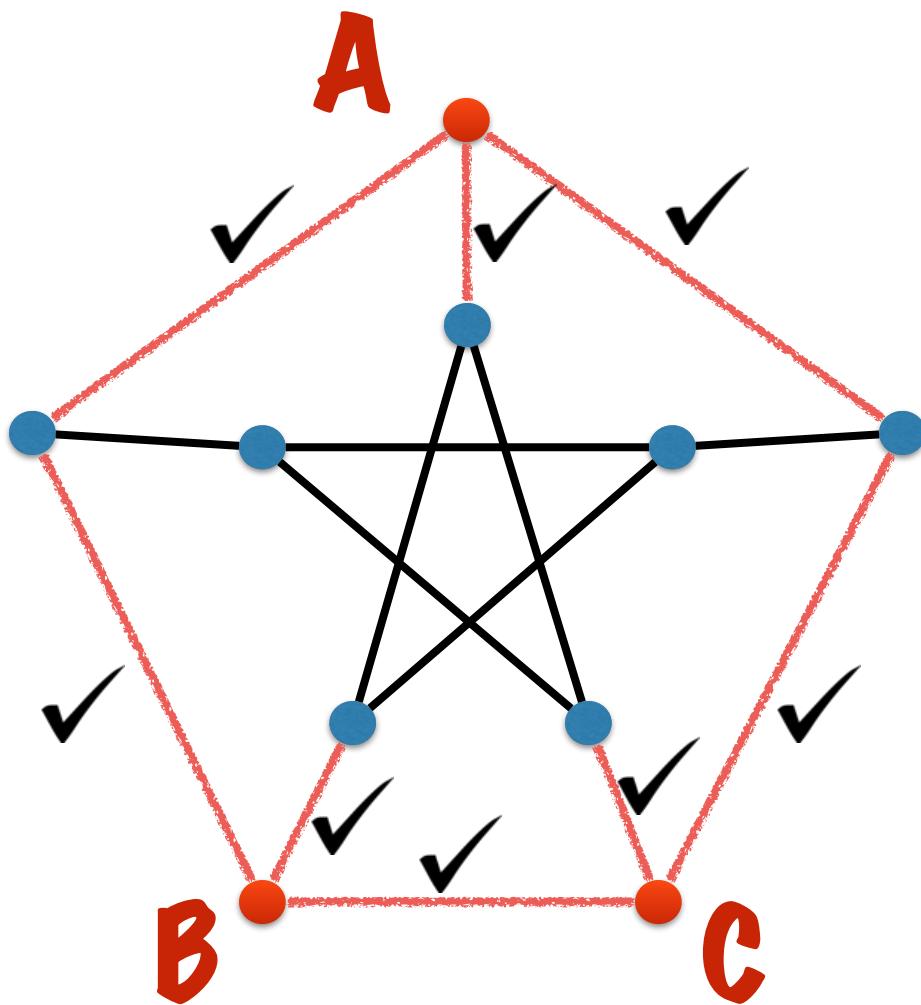
# Vertex cover

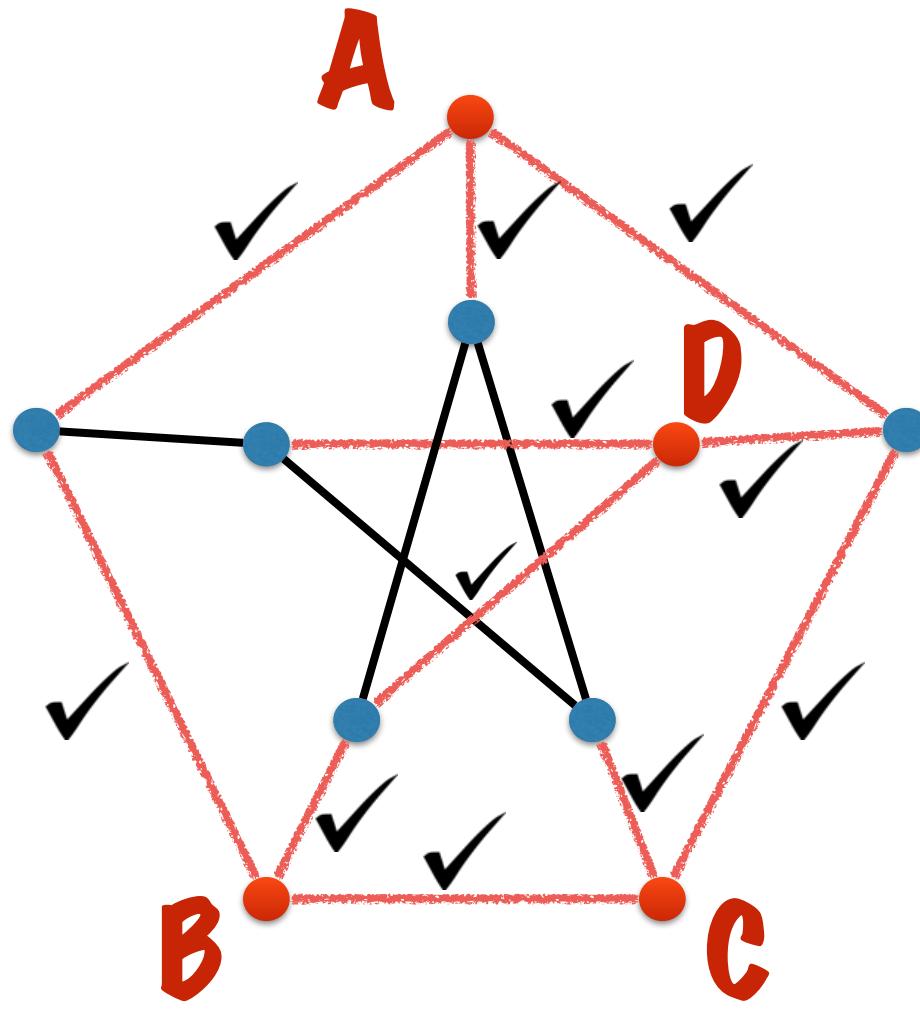
Given graph with vertex weights,  
cover edges with lightest vertices

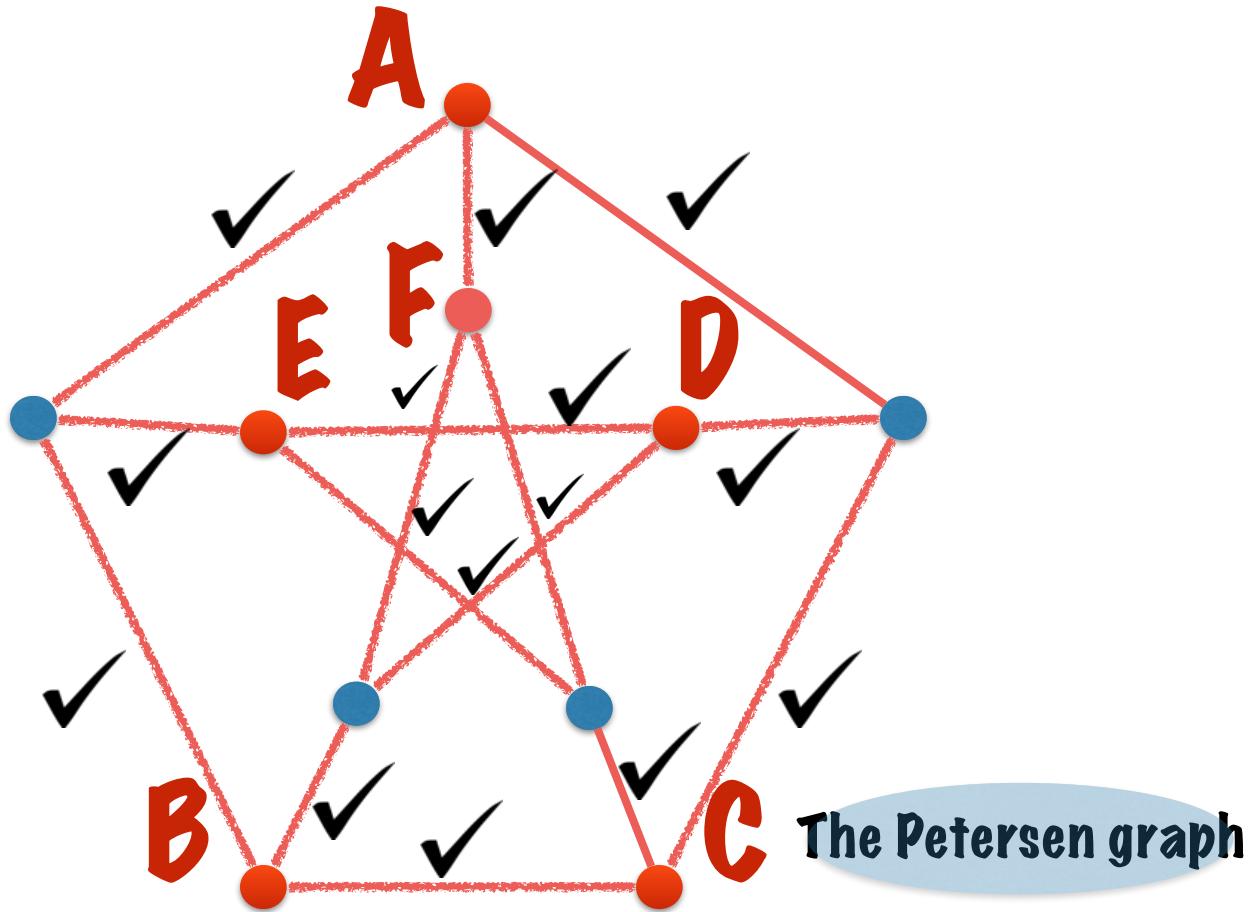




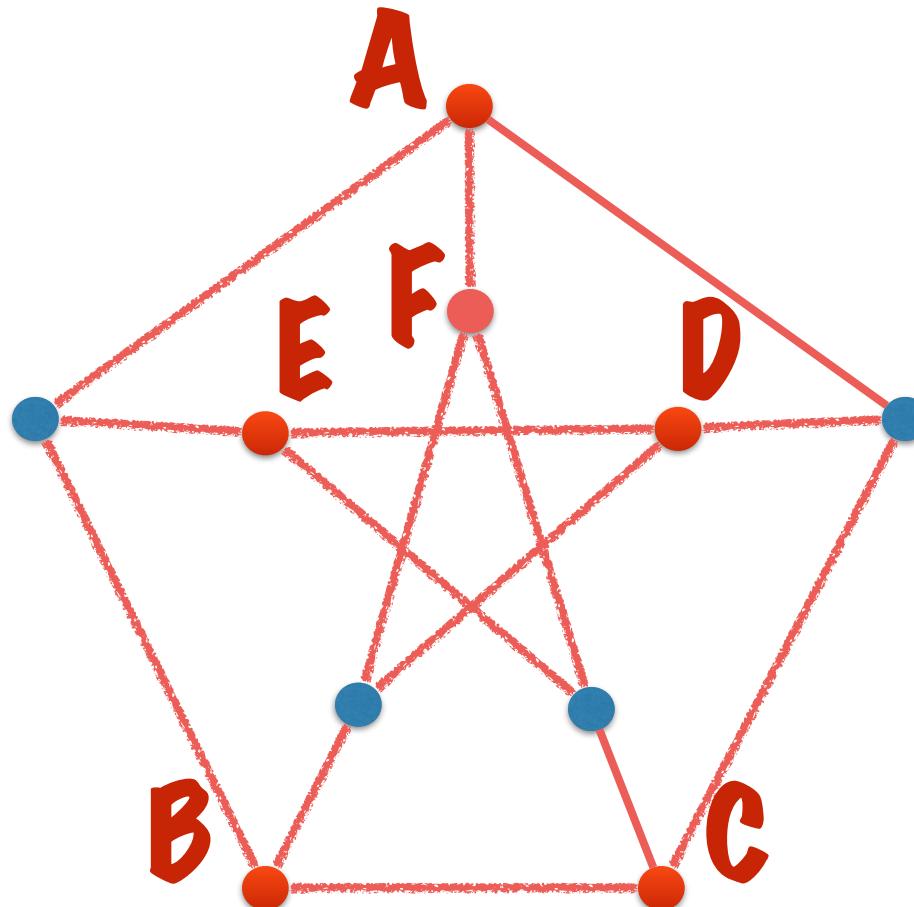






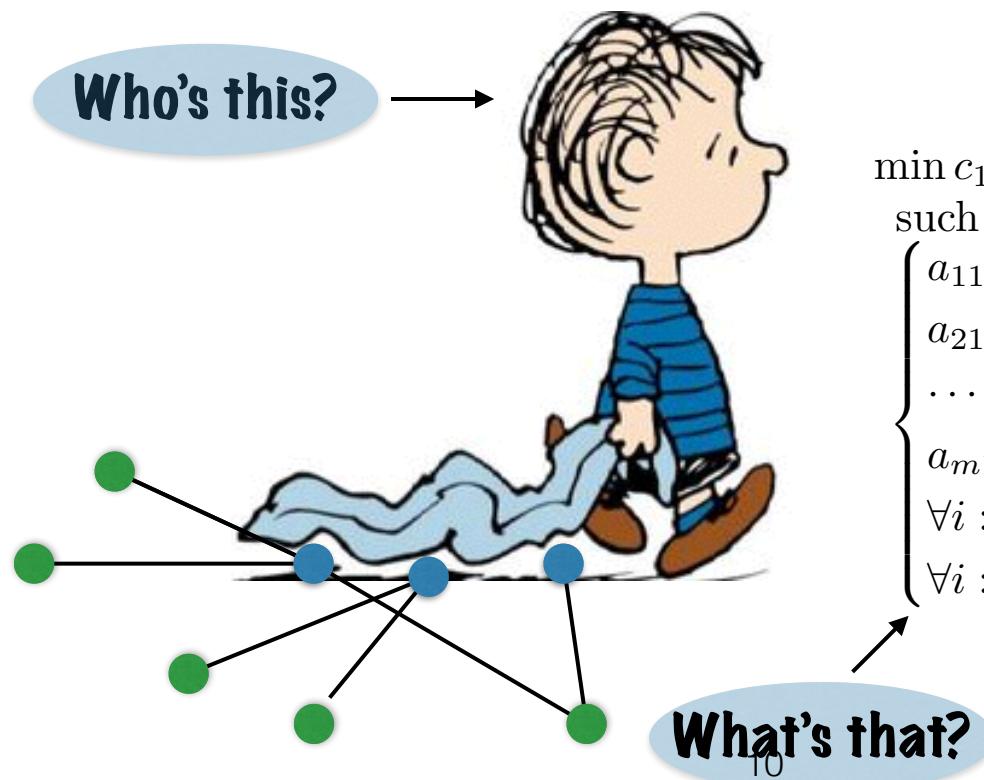


**Vertex weights: 1.**  
**Cover edges with 6 vertices.**  
**Optimal: cannot cover with 5**



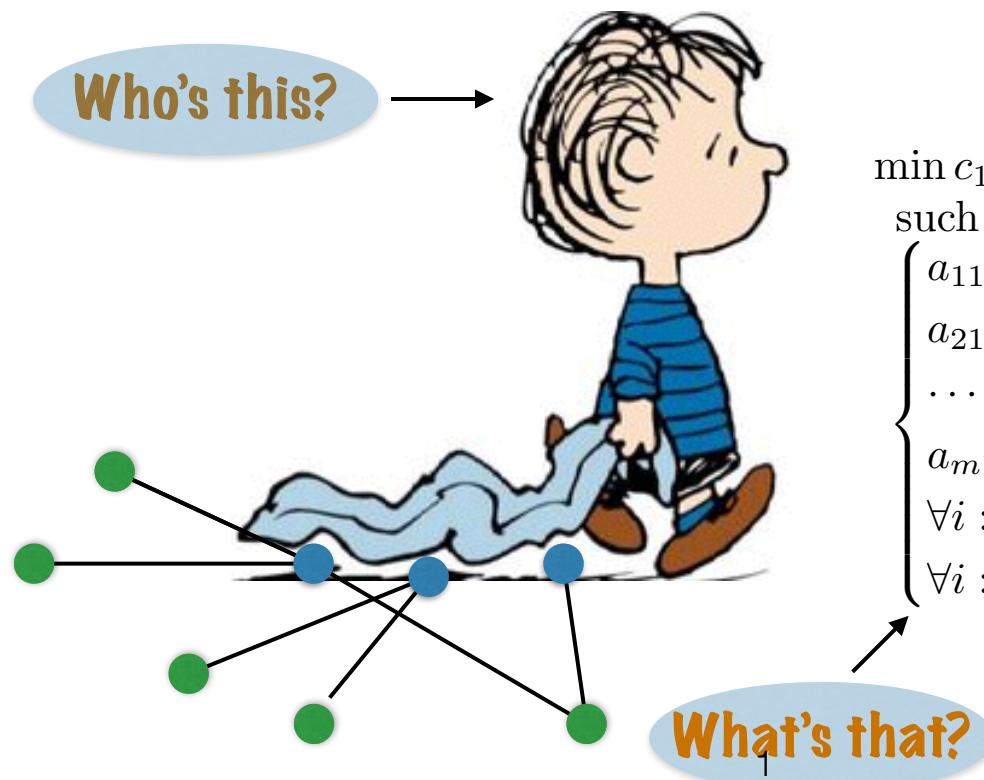
Given graph with vertex weights,  
cover edges with lightest vertices

# Approximation algorithms, vertex cover, and linear programming



$$\begin{aligned} & \min c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{such that} \\ & \left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \geq b_m \end{array} \right. \\ & \forall i : 0 \leq x_i \leq 1 \\ & \forall i : x_i \text{ real number} \end{aligned}$$

# Approximation algorithms, vertex cover, and linear programming



$$\begin{aligned} & \min c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{such that} \quad & \left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \geq b_m \end{array} \right. \\ & \forall i : 0 \leq x_i \leq 1 \\ & \forall i : x_i \text{ real number} \end{aligned}$$

# Variables

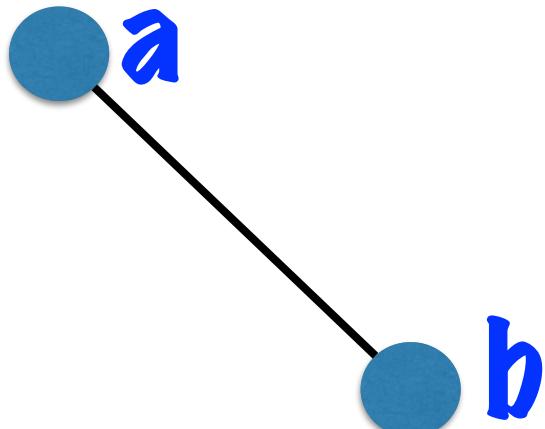
$\{a, b\} \in E :$

a or b must be in cover



$$x_a = \begin{cases} 1 & \text{if } a \text{ in cover} \\ 0 & \text{otherwise} \end{cases}$$

# Constraints



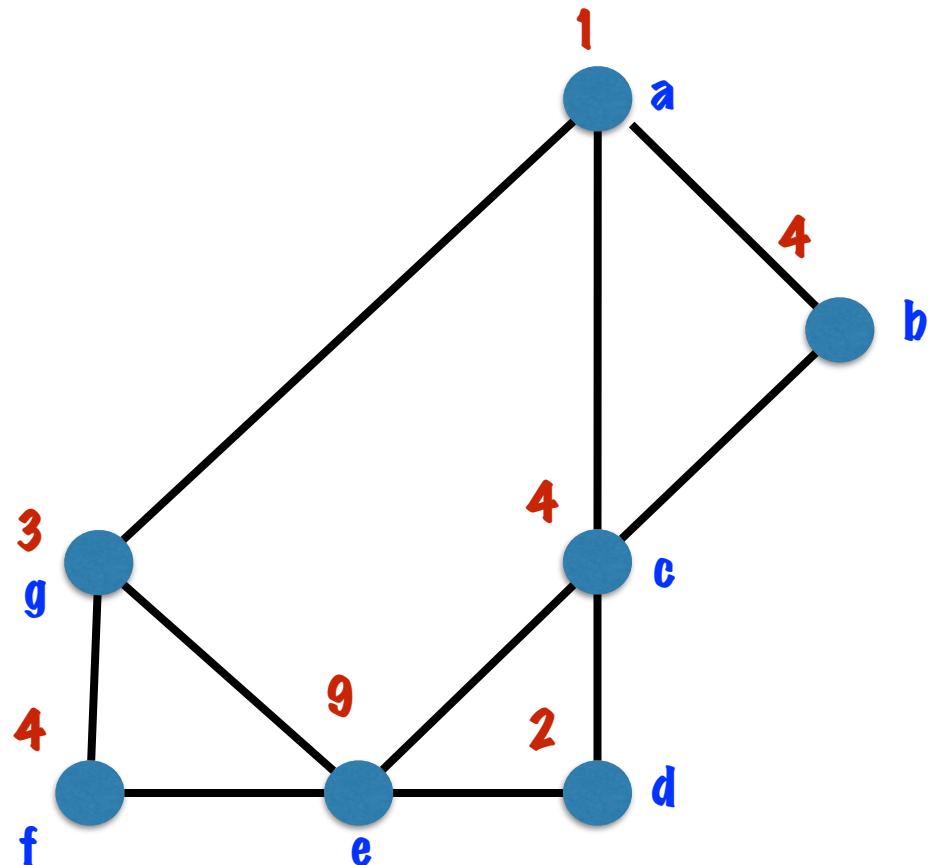
| $x_a$ | $x_b$ | edge covered? | $x_a + x_b$ |
|-------|-------|---------------|-------------|
| 0     | 0     | no            | 0           |
| 1     | 0     | yes           | 1           |
| 0     | 1     | yes           | 1           |
| 1     | 1     | yes           | 2           |

$\{a, b\}$  covered



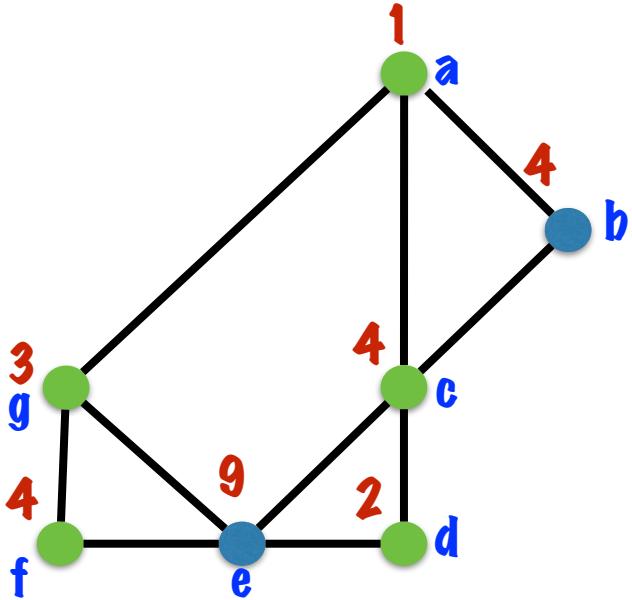
$$x_a + x_b \geq 1$$

# Objective



$$\left\{ \begin{array}{ll} x_a + x_b & \geq 1 \\ x_a + x_c & \geq 1 \\ x_a + x_g & \geq 1 \\ x_b + x_c & \geq 1 \\ x_c + x_d & \geq 1 \\ x_c + x_e & \geq 1 \\ x_d + x_e & \geq 1 \\ x_e + x_f & \geq 1 \\ x_e + x_g & \geq 1 \\ x_f + x_g & \geq 1 \end{array} \right.$$

$$\min x_a + 4x_b + 4x_c + 2x_d + 9x_e + 4x_f + 3x_g$$



$\{a, c, d, f, g\}$  vertex cover

$$\left\{ \begin{array}{l} x_a + x_b \geq 1 \\ x_a + x_c \geq 1 \\ x_a + x_g \geq 1 \\ x_b + x_c \geq 1 \\ x_c + x_d \geq 1 \\ x_c + x_e \geq 1 \\ x_d + x_e \geq 1 \\ x_e + x_f \geq 1 \\ x_e + x_g \geq 1 \\ x_f + x_g \geq 1 \end{array} \right.$$

$$\min x_a + 4x_b + 4x_c + 2x_d + 9x_e + 4x_f + 3x_g$$

$$x_a = x_c = x_d = x_f = x_g = 1$$

$$x_b = x_e = 0$$

satisfies all constraints

value = 14

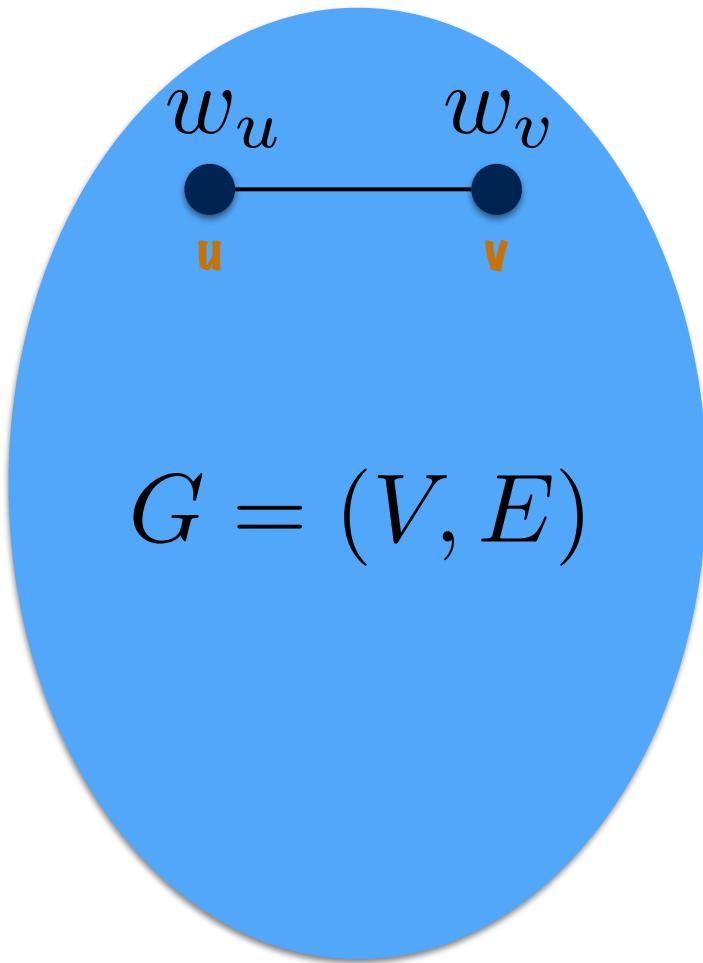
# in general

Constraints:

$$\forall u \in V : x_u = 0 \text{ or } 1$$

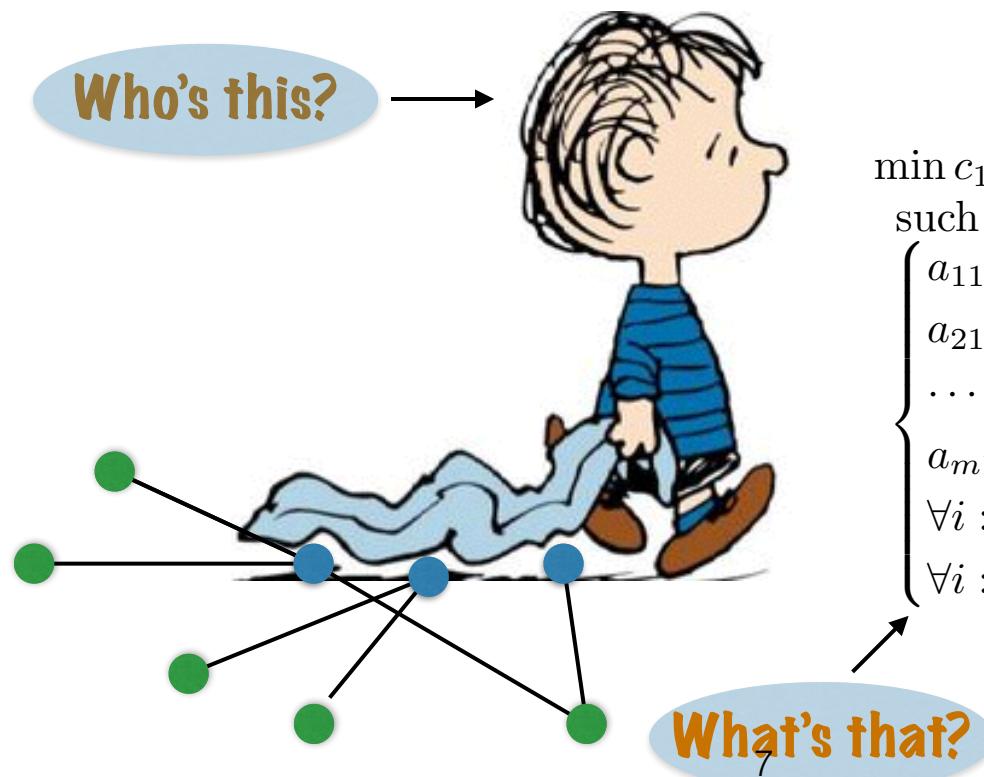
$$\forall \{u, v\} \in E : x_u + x_v \geq 1$$

Objective:  $\min \sum_u w_u x_u$



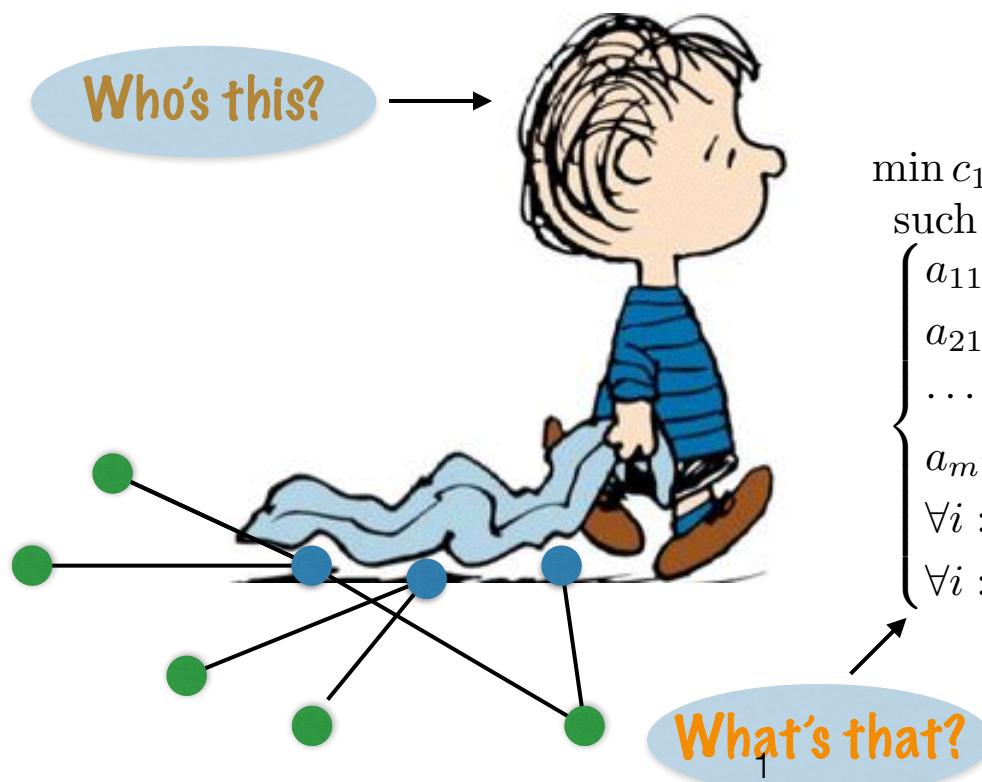
# integer program

# Approximation algorithms, vertex cover, and linear programming



$$\begin{aligned} & \min c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{such that} \quad & \left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \geq b_m \end{array} \right. \\ & \forall i : 0 \leq x_i \leq 1 \\ & \forall i : x_i \text{ real number} \end{aligned}$$

# Approximation algorithms, vertex cover, and linear programming



$$\begin{aligned} & \min c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{such that} \\ & \left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \geq b_m \end{array} \right. \\ & \forall i : 0 \leq x_i \leq 1 \\ & \forall i : x_i \text{ real number} \end{aligned}$$

# Integer program

$$\min c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$$

such that

$$\left\{ \begin{array}{ll} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & \geq b_m \\ \forall i : 0 \leq x_i \leq 1 \\ \forall i : x_i \text{ integer} \end{array} \right.$$

NP-hard

# Linear program

$$\min c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$$

such that

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \geq b_m \\ \forall i : 0 \leq x_i \leq 1 \\ \forall i : x_i \text{ real number} \end{array} \right.$$

**polynomial time**

# Two ways to present

$$\min c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$$

such that

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & \geq b_m \\ \forall i : 0 \leq x_i \leq 1 \\ \forall i : x_i \text{ real number} \end{cases}$$

## Linear program

$$\begin{array}{l} \min c \cdot x \text{ such that} \\ \begin{cases} Ax \geq b \\ x \in [0, 1]^n \\ x \text{ vector of } \mathbb{R}^n \end{cases} \end{array}$$

## Same linear program

# Integer vs. linear programs

IP

$$\min c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$$

such that

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & \geq b_m \\ \forall i : 0 \leq x_i \leq 1 \\ \forall i : x_i \text{ integer} \end{cases}$$

LP

$$\min c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$$

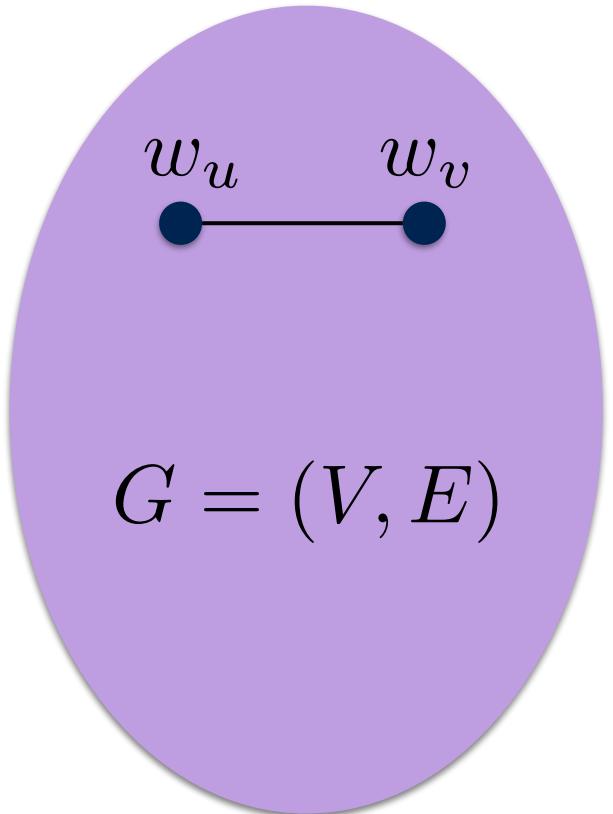
such that

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & \geq b_m \\ \forall i : 0 \leq x_i \leq 1 \\ \forall i : x_i \text{ real number} \end{cases}$$

NP-hard

polynomial time

# Vertex cover linear program



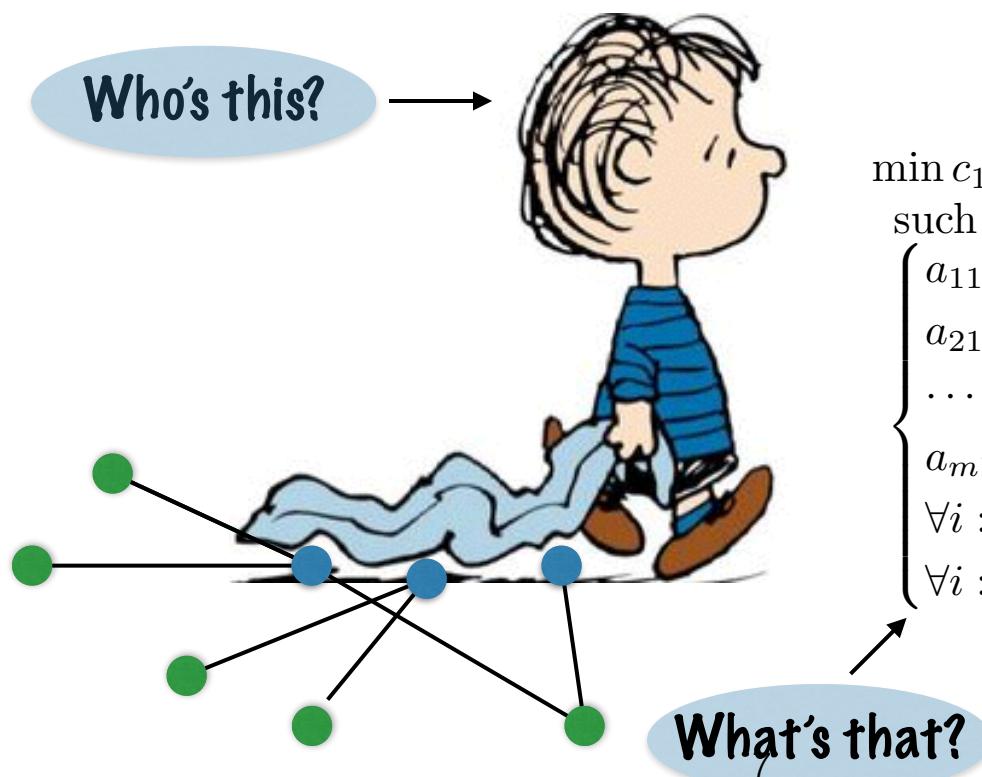
Constraints:

$$\forall u \in V : 0 \leq x_u \leq 1$$

$$\forall \{u, v\} \in E : x_u + x_v \geq 1$$

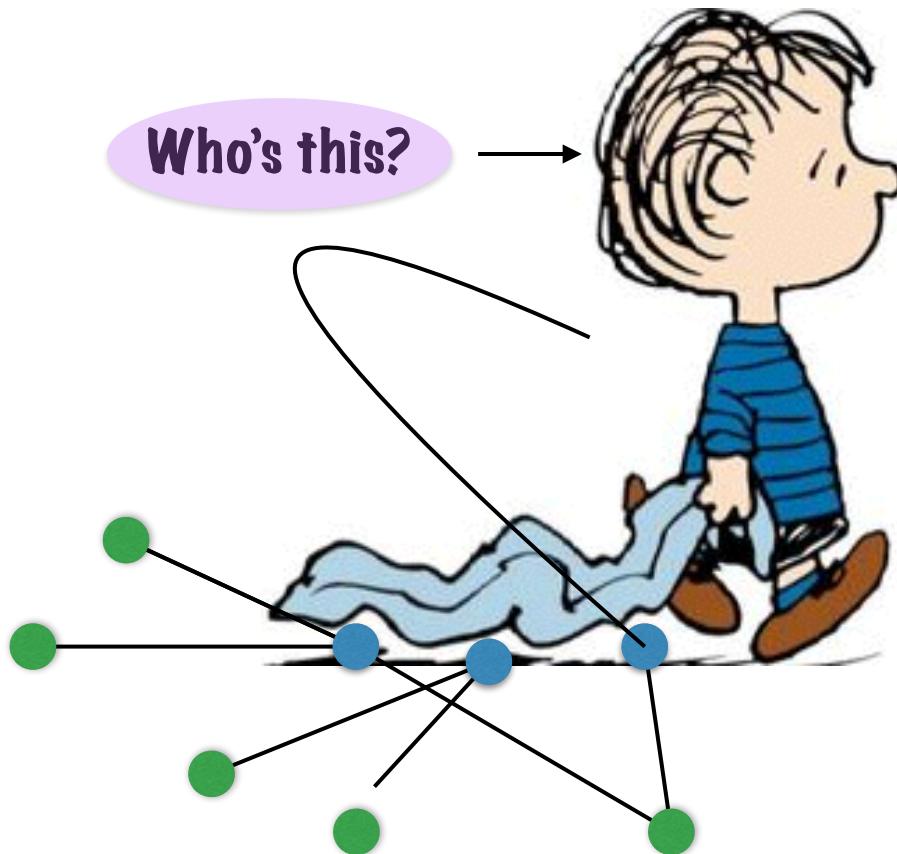
Objective:  $\min \sum_u w_u x_u$

# Approximation algorithms, vertex cover, and linear programming



$$\begin{aligned} & \min c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ & \text{such that} \\ & \left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \geq b_m \\ \forall i : 0 \leq x_i \leq 1 \\ \forall i : x_i \text{ real number} \end{array} \right. \end{aligned}$$

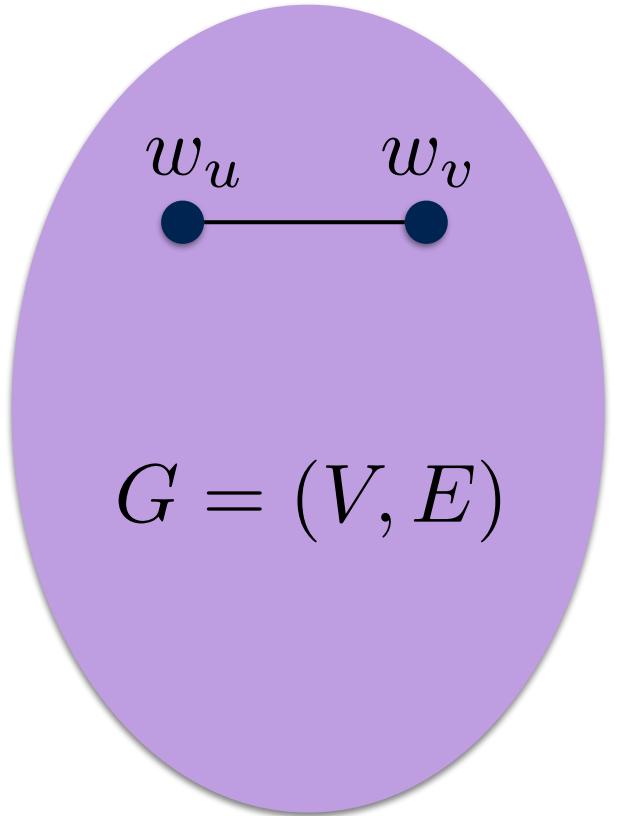
# Approximation algorithms, vertex cover, and linear programming



$$\begin{aligned} & \min c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{such that } & \left\{ \begin{array}{ll} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & \geq b_2 \\ \dots & \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & \geq b_m \end{array} \right. \\ & \forall i : 0 \leq x_i \leq 1 \\ & \forall i : x_i \text{ real number} \end{aligned}$$

What's that?

# Using the LP (1/3)



Constraints:

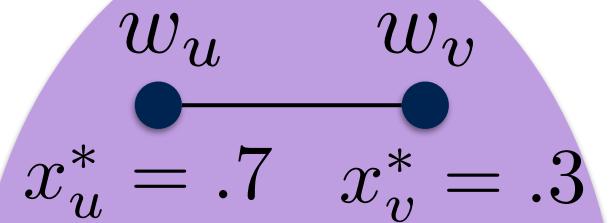
$$\forall u \in V : 0 \leq x_u \leq 1$$

$$\forall \{u, v\} \in E : x_u + x_v \geq 1$$

Objective:  $\min \sum_u w_u x_u$

# Using the LP (2/3)

## 1. Solving the LP



$$G = (V, E)$$

$\implies (x_u^*)_{u \in V}$  such that

$$\forall u \in V : 0 \leq x_u^* \leq 1$$

$$\forall \{u, v\} \in E : x_u^* + x_v^* \geq 1$$

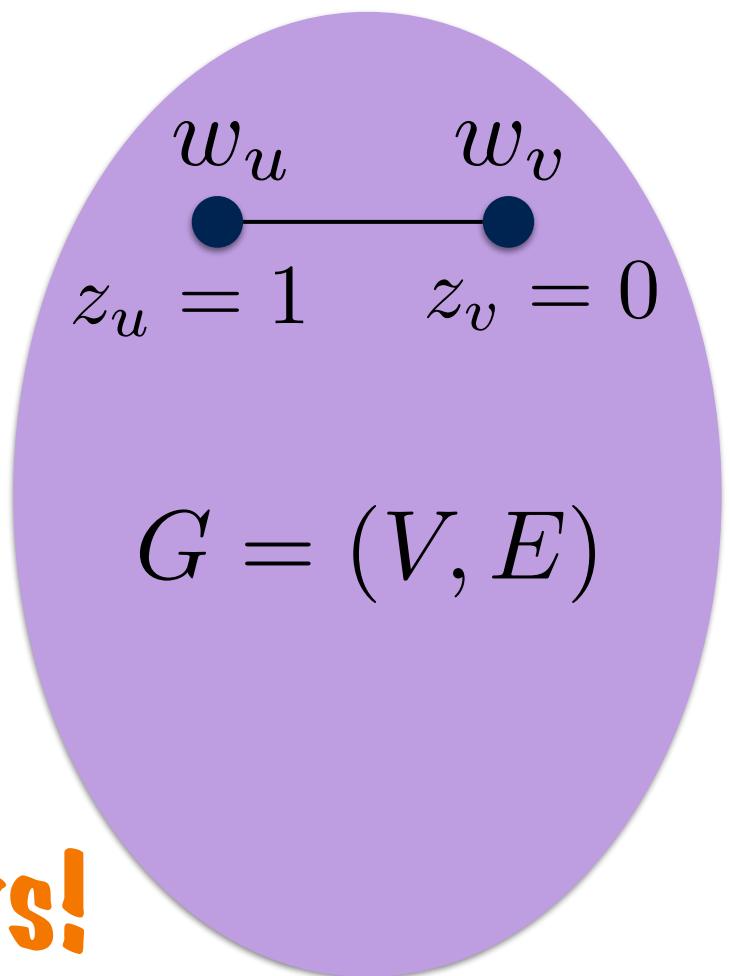
$$\sum_u w_u x_u^* \text{ minimum}$$

# Using the LP (3/3)

## 2. Rounding the LP solution

$\implies (z_u)_{u \in V}$  defined by

$$z_u = \begin{cases} 1 & \text{if } x_u^* \geq .5 \\ 0 & \text{otherwise} \end{cases}$$



# We are back to integers!

# Runtime

## 1. Solve the LP

$(x_u^*)_{u \in V}$  such that

$$\forall u \in V : 0 \leq x_u^* \leq 1 \quad \longleftarrow$$

$$\forall \{u, v\} \in E : x_u^* + x_v^* \geq 1$$

$$\sum_u w_u x_u^* \text{ minimum}$$

Polynomial time

## 2. Round the LP solution

$\implies (z_u)_{u \in V}$  defined by

$$z_u = \begin{cases} 1 & \text{if } x_u^* \geq .5 \\ 0 & \text{otherwise} \end{cases}$$

Linear time

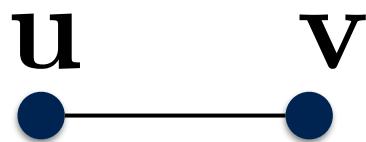
## 3. Output

$\{u \in V \text{ such that } z_u = 1\}$

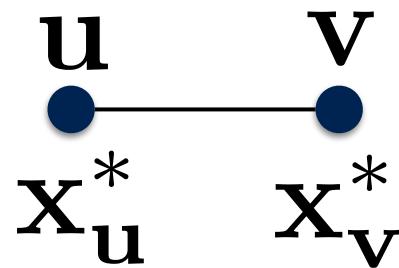
# Correctness

... is it a vertex cover?

# Does output cover all edges?

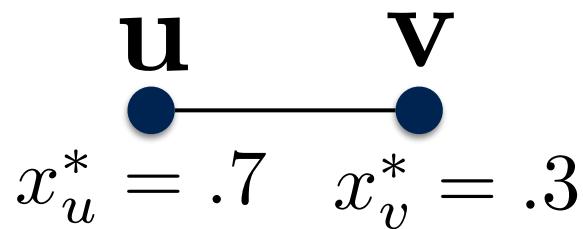


$$\{u, v\} : x_u^* + x_v^* \geq 1$$



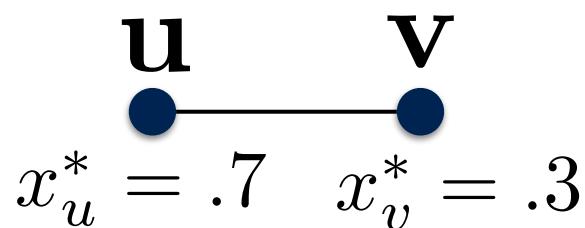
# Does output cover all edges?

$$x_u^* + x_v^* \geq 1$$



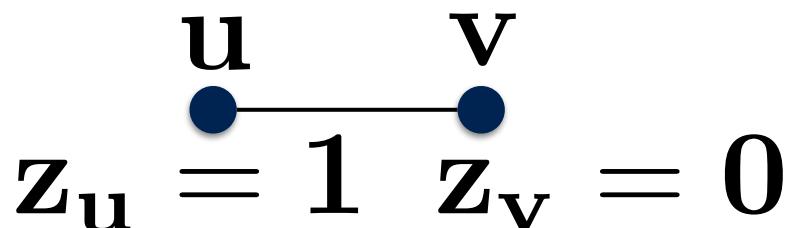
$$x_u^* \geq .5 \text{ or } x_v^* \geq .5$$

# Does output cover all edges?



$\implies (z_u)_{u \in V}$  defined by  
$$z_u = \begin{cases} 1 & \text{if } x_u^* \geq .5 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{x}_u^* \geq .5 \text{ or } \mathbf{x}_v^* \geq .5$$

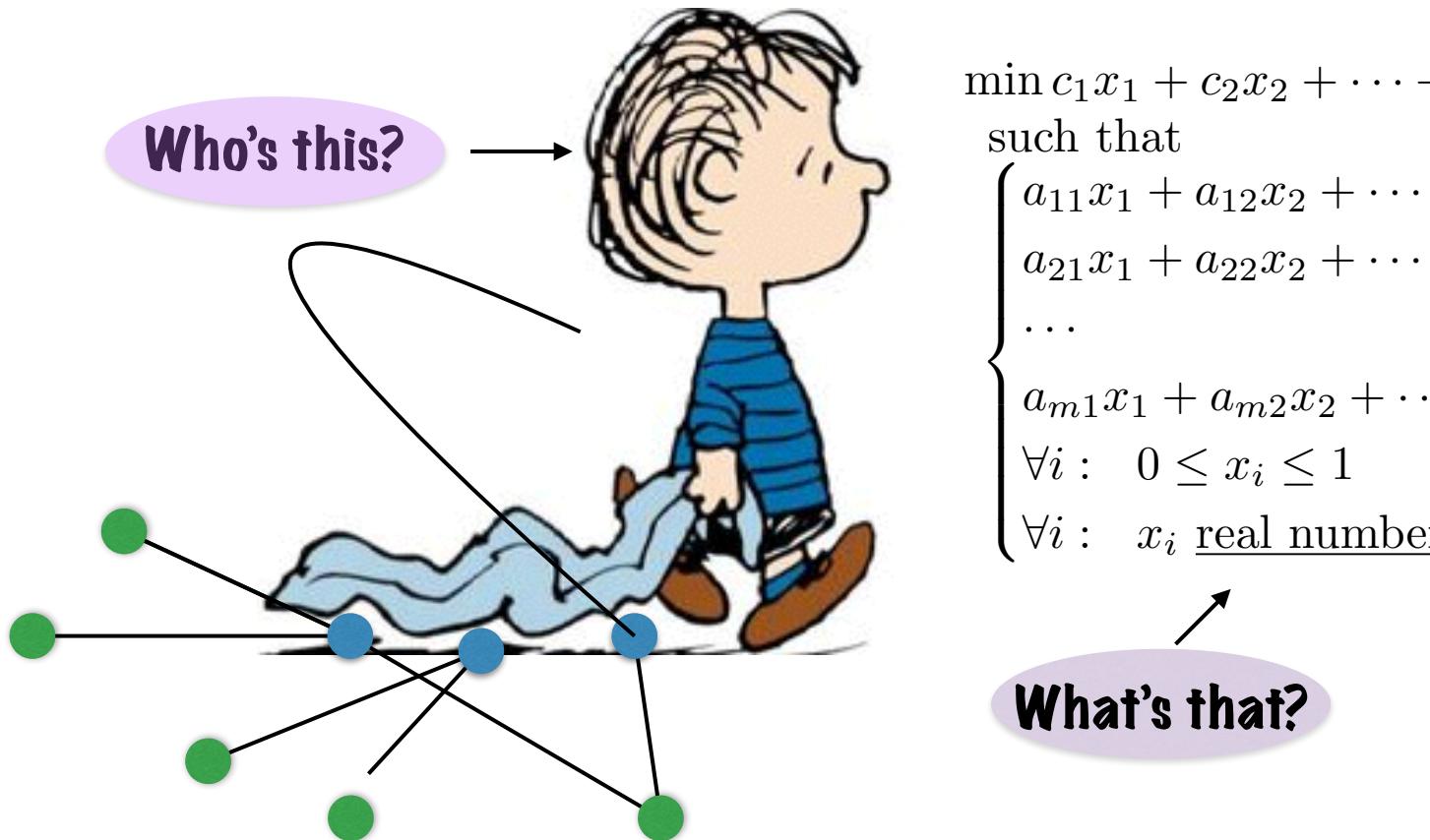


u is in output



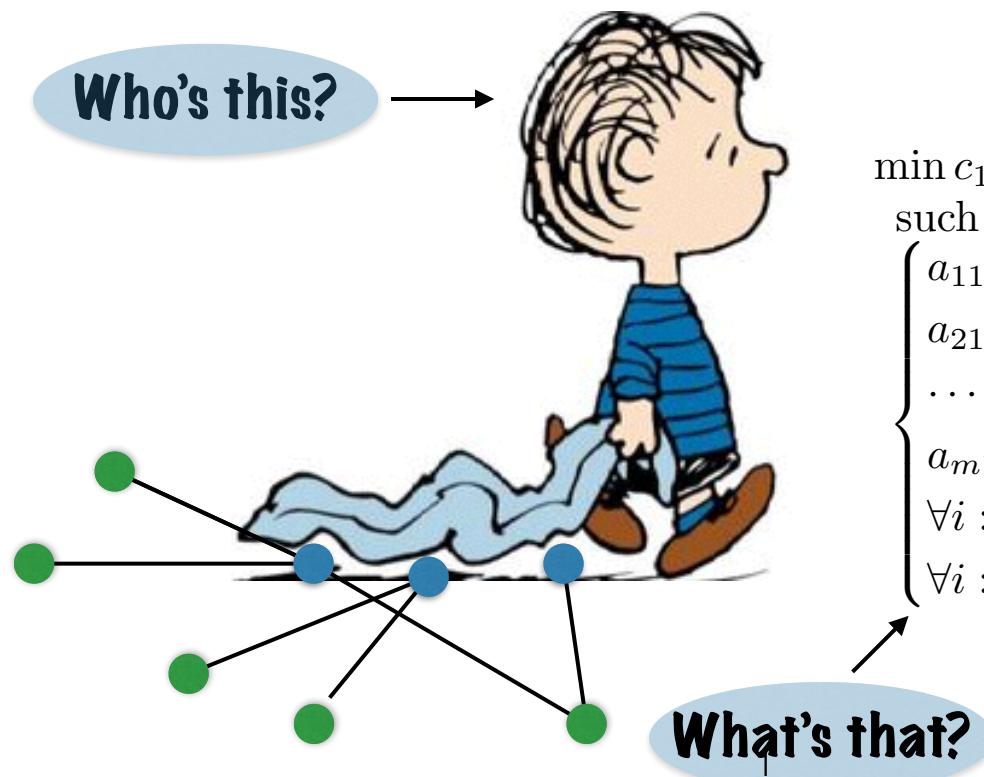
$$z_u = 1 \text{ or } z_v = 1$$

# Approximation algorithms, vertex cover, and linear programming



$$\begin{aligned} & \min c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{such that} \quad & \left\{ \begin{array}{ll} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & \geq b_m \\ \forall i : 0 \leq x_i \leq 1 \\ \forall i : x_i \text{ real number} \end{array} \right. \end{aligned}$$

# Approximation algorithms, vertex cover, and linear programming



$$\begin{aligned} & \min c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{such that} \quad & \left\{ \begin{array}{ll} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & \geq b_m \end{array} \right. \\ & \forall i : 0 \leq x_i \leq 1 \\ & \forall i : x_i \text{ real number} \end{aligned}$$

# Quality of output?

**2. Round the LP solution**

$$z_u \in \{0, 1\}$$

**3. Output**  $\{u : z_u = 1\}$

**Output cost** =  $\sum_u w_u z_u$

**1. Solve the LP**

$$(\mathbf{x}_u^*)$$

**2. Round the LP solution**

$$z_u = \begin{cases} 1 & \text{if } \mathbf{x}_u^* \geq 1/2 \\ 0 & \text{otherwise} \end{cases}$$

**Observe:**  $z_u \leq 2x_u^*$

# 1. Solve the LP

IP min:  $x(u)=1$   
iff  $u$  in optimum  
vertex cover

$$\min \sum_u w_u x_u$$

$$x_u + x_v \geq 1$$

$$x_u \in \{0, 1\}$$

LP min:  $x^*(u)$

$$\min \sum_u w_u x_u^*$$

$$x_u^* + x_v^* \geq 1$$

$$0 \leq x_u^* \leq 1$$

The LP is a relaxation of the IP

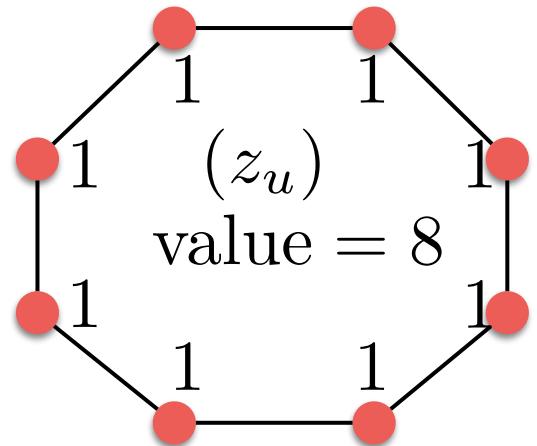
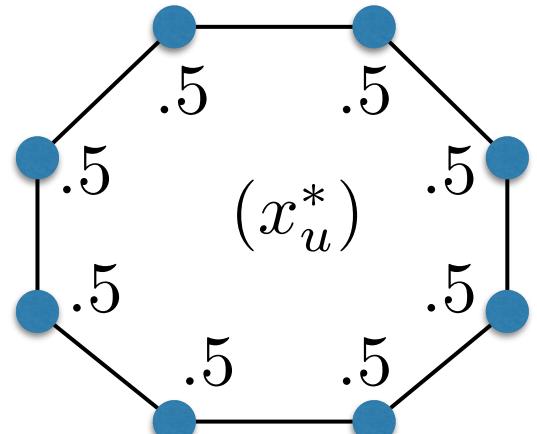
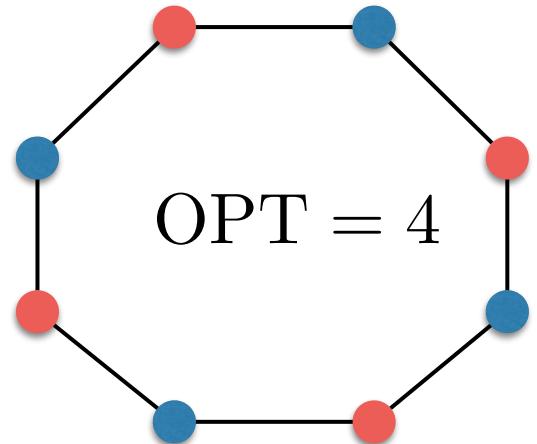
$$\sum_u w_u x_u^* \leq \text{OPT}$$

# Combine

$$\begin{aligned}\text{Output cost} &= \sum_u w_u z_u \\ &\leq 2 \sum_u w_u x_u^* \\ &\leq 2 \text{OPT}\end{aligned}$$

Thm: output is a vertex cover  
of value at most 2 OPT

Is the analysis tight?



# How good is that?

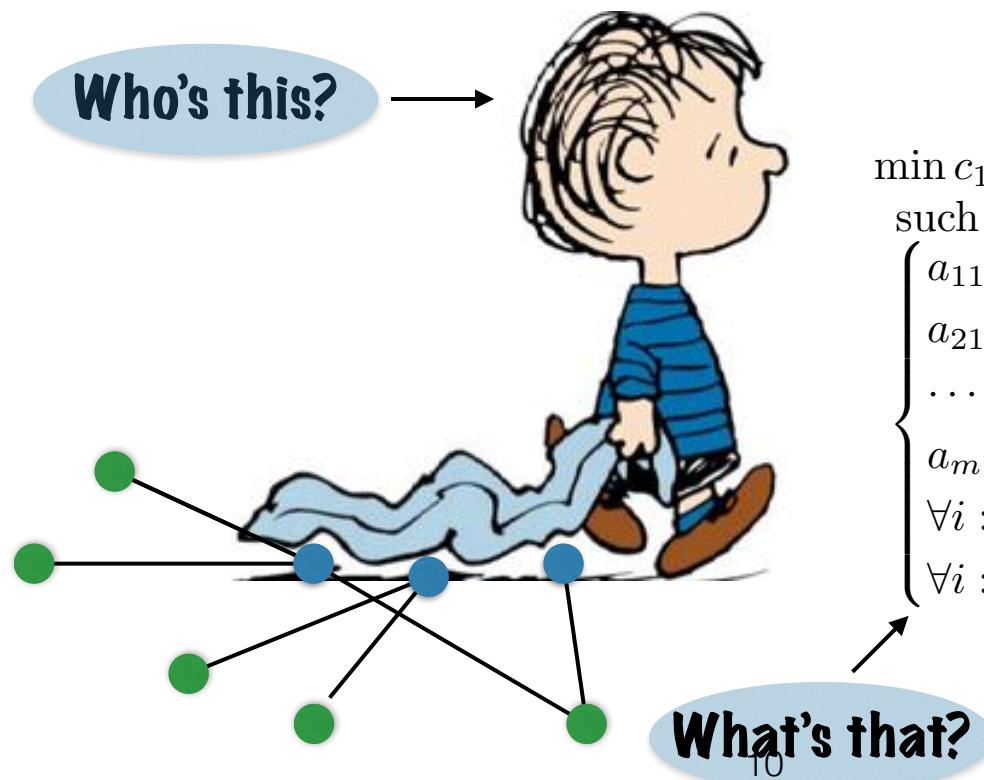
Typical performance  
(hearsay):  
within 10% of optimum

# How do we know?

Can compare output  
value to

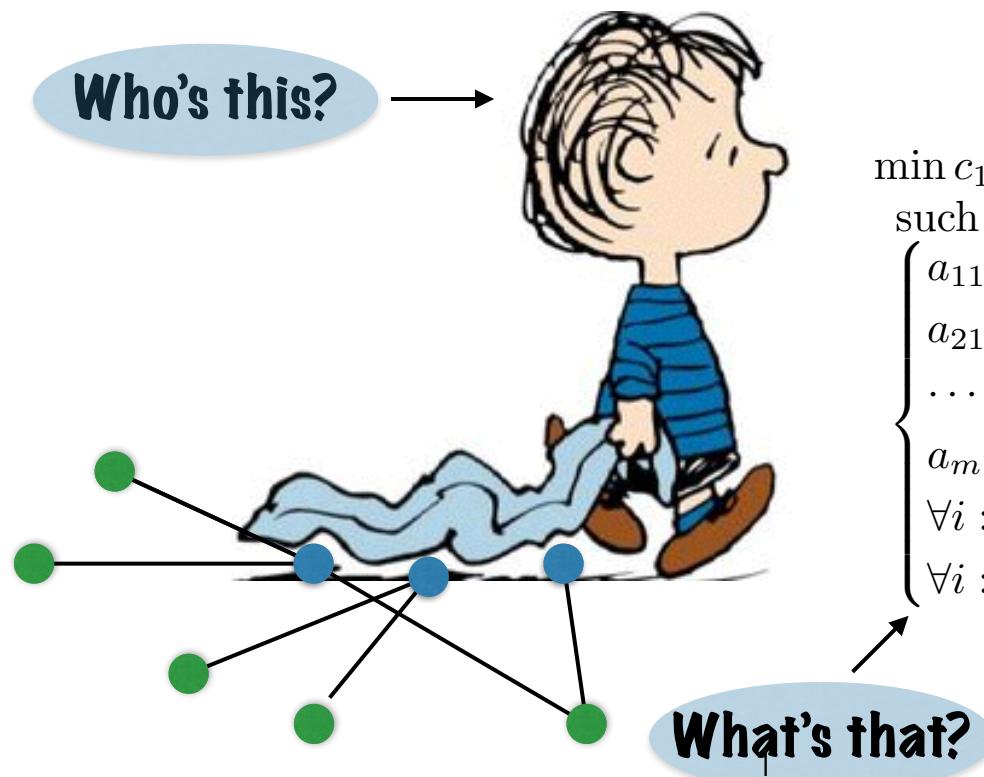
$$\sum_u w_u x_u^*$$

# Approximation algorithms, vertex cover, and linear programming



$$\begin{aligned} & \min c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{such that} \quad & \left\{ \begin{array}{ll} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & \geq b_m \end{array} \right. \\ & \forall i : 0 \leq x_i \leq 1 \\ & \forall i : x_i \text{ real number} \end{aligned}$$

# Approximation algorithms, vertex cover, and linear programming



$$\begin{aligned} & \min c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{such that} \quad & \left\{ \begin{array}{ll} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & \geq b_m \end{array} \right. \\ & \forall i : 0 \leq x_i \leq 1 \\ & \forall i : x_i \text{ real number} \end{aligned}$$

# The method

# **Meta approximation algorithm**

- Find IP
- Solve LP relaxation
- Round solution to integers

# Analysis

- correct: does it satisfy conditions?
- efficient: polynomial runtime?
- good: value of output solution within factor of optimal value?

# How good is it?

Method

- Output can be related to LP value

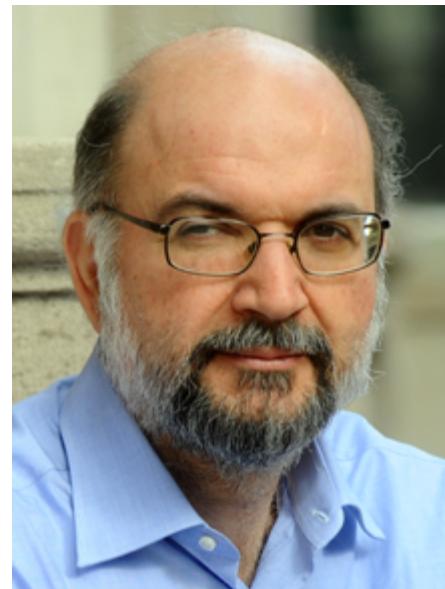
- OPT can be bounded by LP value

Combine

Message: for analysis,  
focus on LP value.



**Karp (1972)**  
**NP-complete**

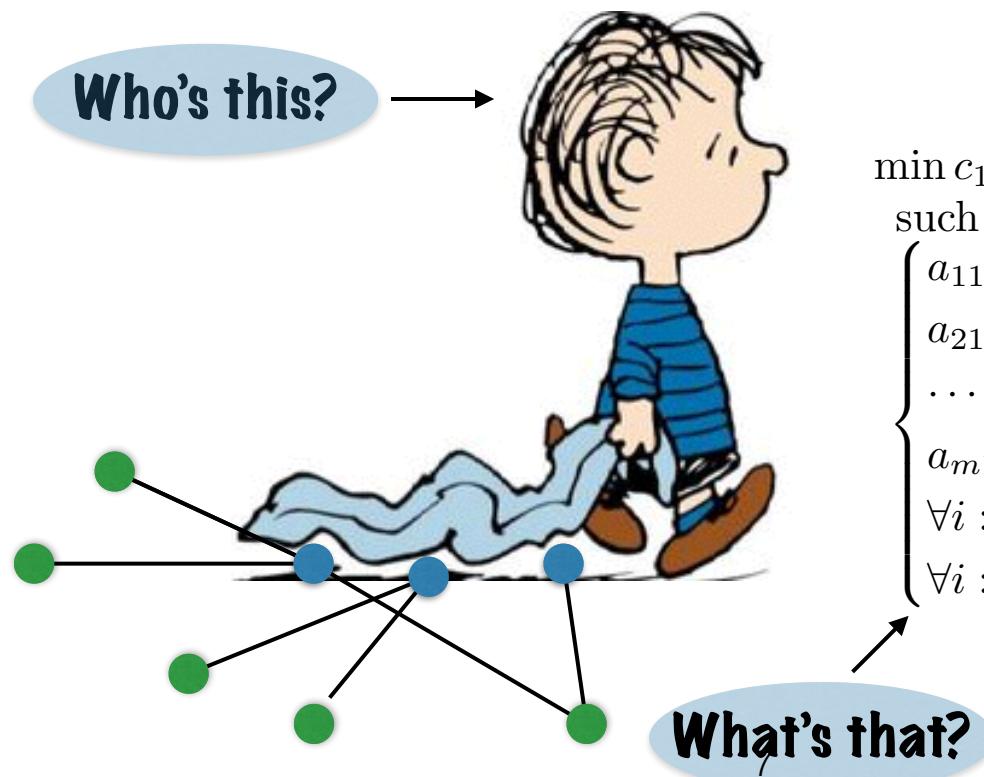


**Khot  
Regev  
(2003)**  
**Conditional  $<2$  hard**

**Papadimitriou  
Yannakakis:**  
**1.0001 hard (1991)**

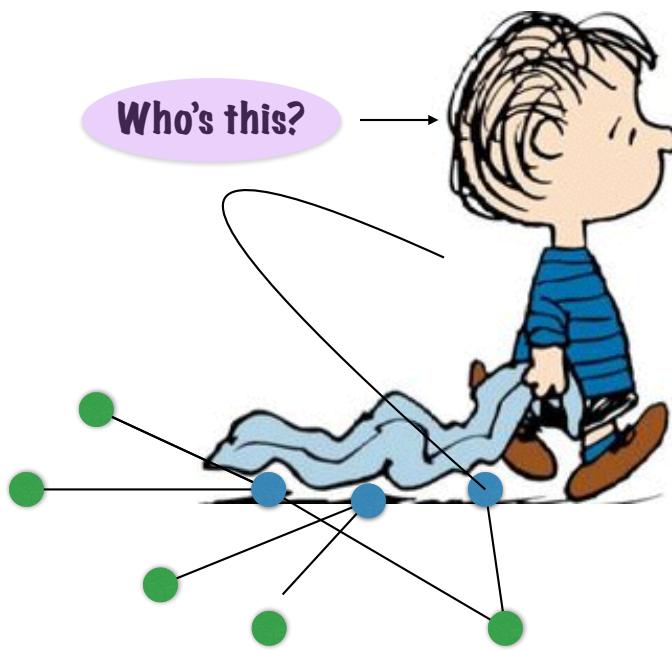
**Dinur  
Safra:**  
**1.36 hard (2002)**

# Approximation algorithms, vertex cover, and linear programming



$$\begin{aligned} & \min c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{such that} \quad & \left\{ \begin{array}{ll} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & \geq b_m \end{array} \right. \\ & \forall i : 0 \leq x_i \leq 1 \\ & \forall i : x_i \text{ real number} \end{aligned}$$

# Approximation algorithms, vertex cover, and linear programming



$$\begin{aligned} & \min c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{such that} \quad & \left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \geq b_m \\ \forall i : 0 \leq x_i \leq 1 \\ \forall i : x_i \text{ real number} \end{array} \right. \end{aligned}$$

What's that?

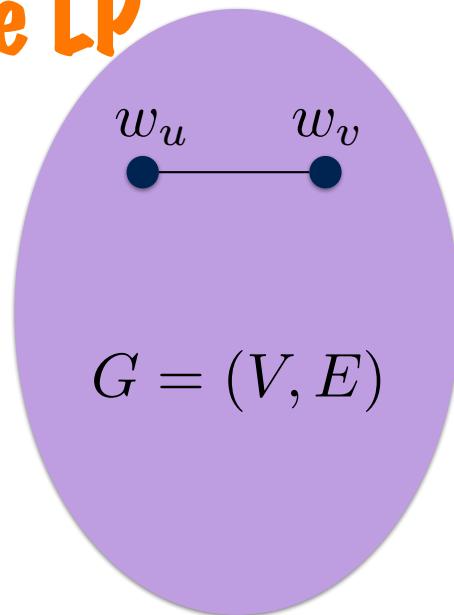
## Property of the LP

Constraints:

$$\forall u \in V : 0 \leq x_u \leq 1$$

$$\forall \{u, v\} \in E : x_u + x_v \geq 1$$

Objective:  $\min \sum_u w_u x_u$



**Theorem:**

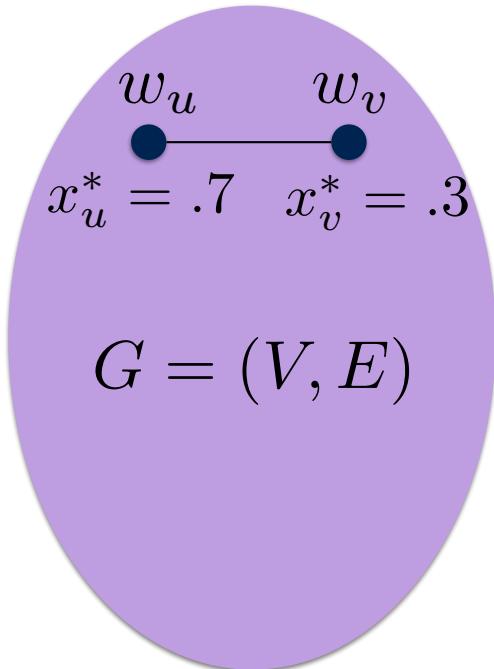
**there exists an optimal solution**

**s.t. every coordinate is in  $\{0, .5, 1\}$**

**and there is a polynomial-time algorithm to construct it**

## 1. Solve the LP

$\implies (x_u^*)_{u \in V}$  such that  
 $\forall u \in V : 0 \leq x_u^* \leq 1$   
 $\forall \{u, v\} \in E : x_u^* + x_v^* \geq 1$   
 $\sum_u w_u x_u^*$  minimum



## 2. Freeze all variables with value in {0, .5, 1}

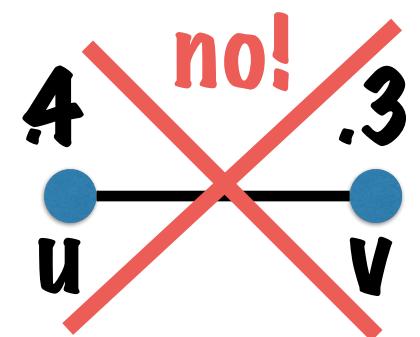
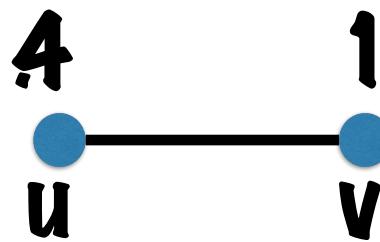
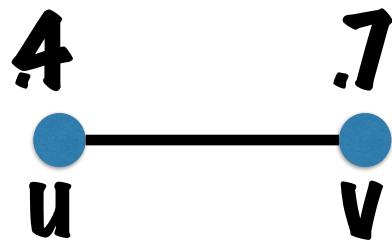
### 3. While some variables are not frozen

$$L = \{u : .5 < x_u^* < 1\}$$

$$S = \{u : 0 < x_u^* < .5\}$$

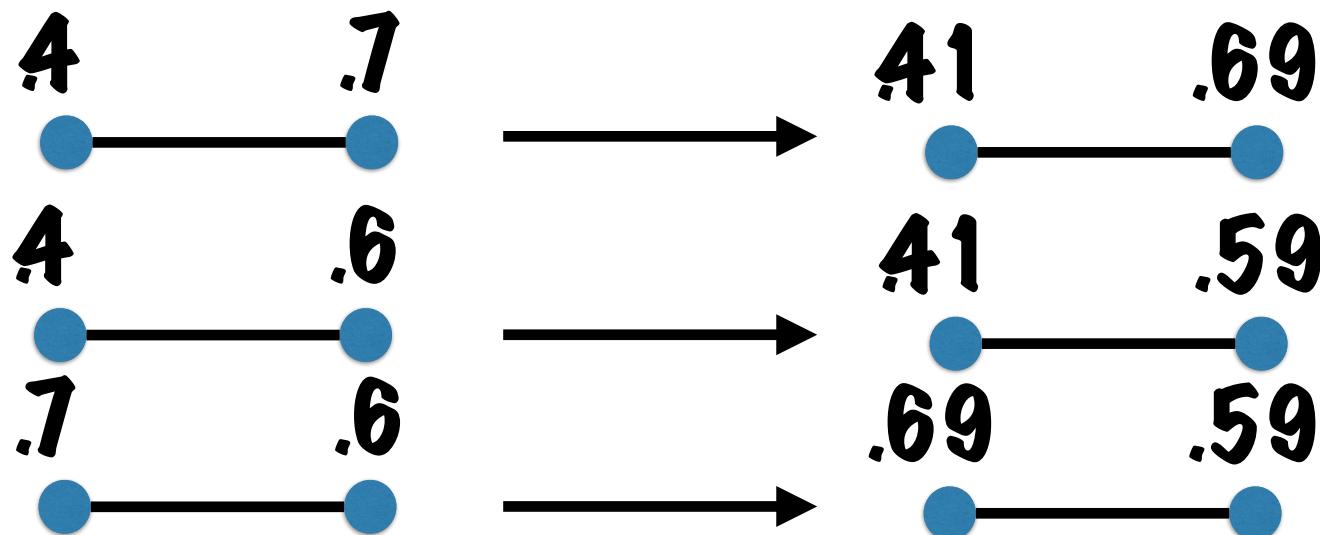
Observe:  
if  $u$  is in  $S$  and  $uv$  is an edge  
then

$v$  is in  $L$  or  $x_v^* = 1$



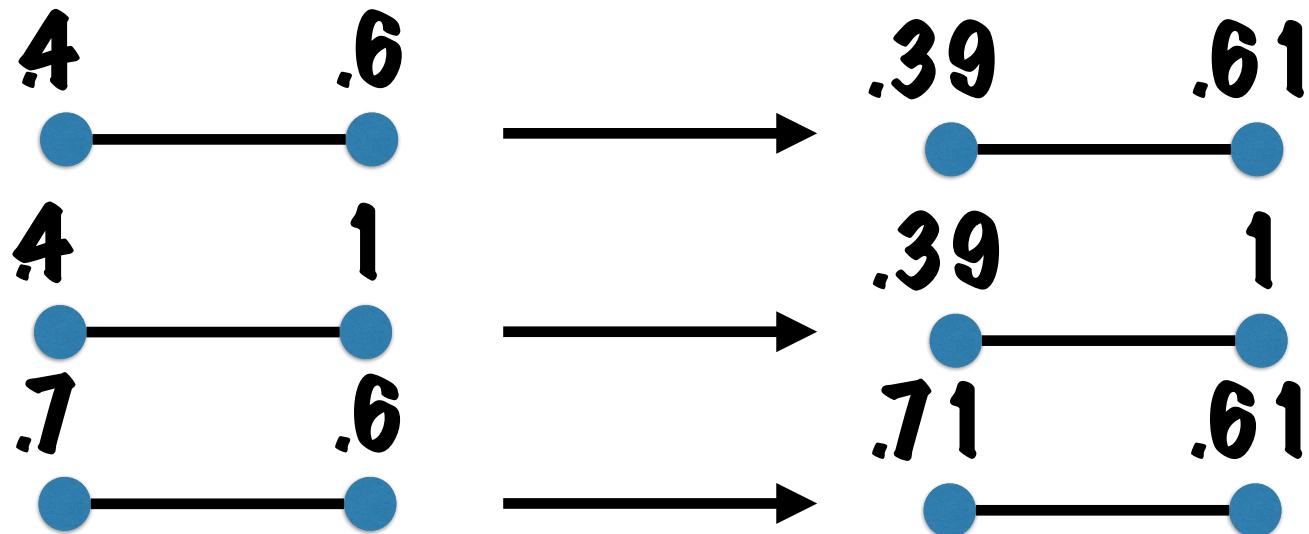
$$\mathbf{y}_u = \begin{cases} x_u^* & \text{if } u \text{ frozen} \\ x_u^* + \epsilon & \text{if } u \in S \\ x_u^* - \epsilon & \text{if } u \in L \end{cases}$$

**Observe: for  $\epsilon$  small, it is still feasible.**



$$z_u = \begin{cases} x_u^* & \text{if } u \text{ frozen} \\ x_u^* - \epsilon & \text{if } u \in S \\ x_u^* + \epsilon & \text{if } u \in L \end{cases}$$

**Observe: for  $\epsilon$  small, it is still feasible.**



$$\mathbf{y}_u = \begin{cases} x_u^* & \text{if } u \text{ frozen} \\ x_u^* + \epsilon & \text{if } u \in S \\ x_u^* - \epsilon & \text{if } u \in L \end{cases} \quad \mathbf{z}_u = \begin{cases} x_u^* & \text{if } u \text{ frozen} \\ x_u^* - \epsilon & \text{if } u \in S \\ x_u^* + \epsilon & \text{if } u \in L \end{cases}$$

**Since  $\mathbf{y}$  feasible and  $\mathbf{x}^*$  optimal:**  $\sum w_u y_u \geq \sum w_u x_u^*$

**Since  $\mathbf{z}$  feasible and  $\mathbf{x}^*$  optimal:**  $\sum w_u z_u \geq \sum w_u x_u^*$

**But observe:**  $(\sum w_u y_u + \sum w_u z_u)/2 = \sum w_u x_u^*$

**So:**

$$\sum_u w_u y_u = \sum_u w_u z_u = \sum_u x_u^*$$

**y and z are also optimal solutions**

increase  $\epsilon$  until something happens:

$$y_u = \begin{cases} x_u^* & \text{if } u \text{ frozen} \\ x_u^* + \epsilon & \text{if } u \in S \\ x_v^* - \epsilon & \text{if } u \in L \end{cases}$$

reaches .5

reaches .5

$$z_u = \begin{cases} x_u^* & \text{if } u \text{ frozen} \\ x_u^* - \epsilon & \text{if } u \in S \\ x_v^* + \epsilon & \text{if } u \in L \end{cases}$$

reaches 0

reaches 1

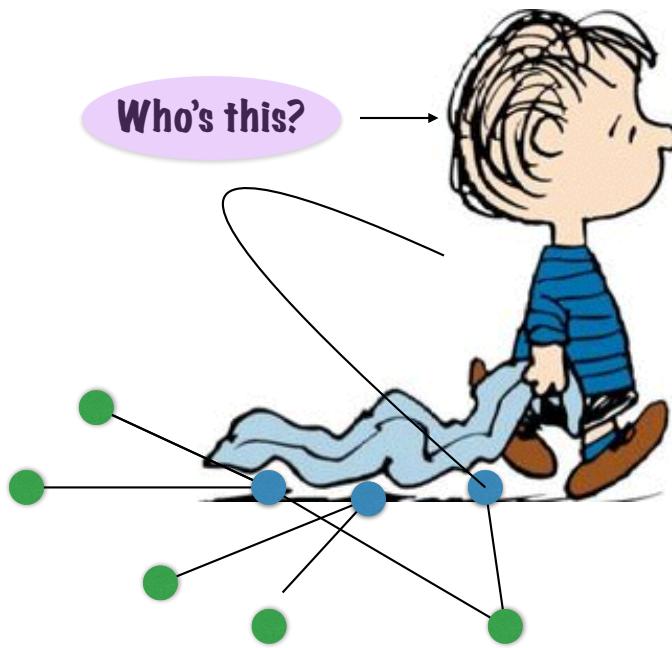
**Freeze** the variable that reached 0, .5, or 1

$$x^* \leftarrow y \text{ or } z$$

Repeat...

QED

# Approximation algorithms, vertex cover, and linear programming



$$\begin{aligned} & \min c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{such that} \quad & \left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \geq b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \geq b_m \\ \forall i : 0 \leq x_i \leq 1 \\ \forall i : x_i \text{ real number} \end{array} \right. \end{aligned}$$

What's that?

# Knapsack and rounding



The knapsack problem: what  
should you put in your  
knapsack?

# Weight Value



.3  
2



.15  
4  
3



.29



3.5  
1



.23  
.23



4  
3



.23  
2



2.5  
5



6.1



.05  
1



.9  
2



10

# The knapsack problem

Given: capacity  $B$  knapsack,  $n$  items,  
item  $i$  has size  $s_i$  and value  $v_i$

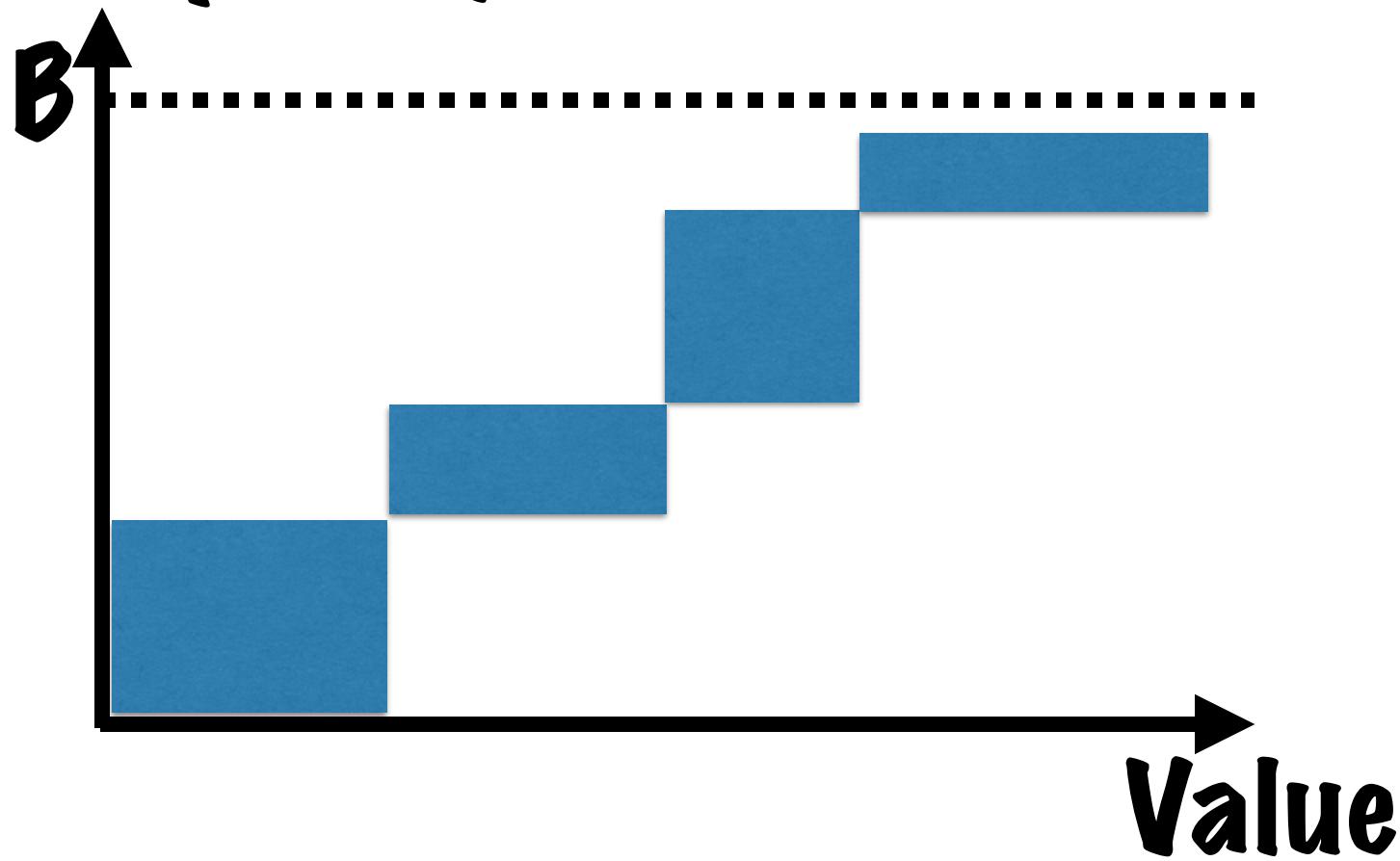
Place some items in knapsack  
Maximize value

NP-hard



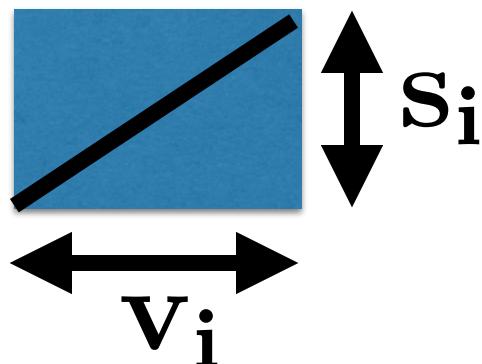
**A general recipe:  
for intuition,  
graphical representation**

**Size, capacity**



**Desire: small size, high value**

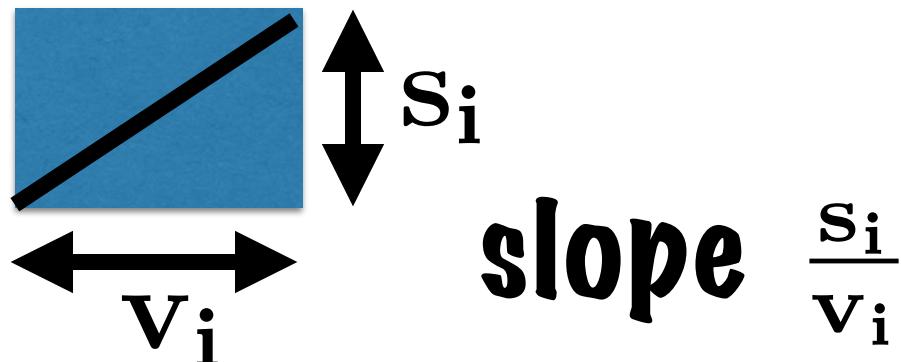
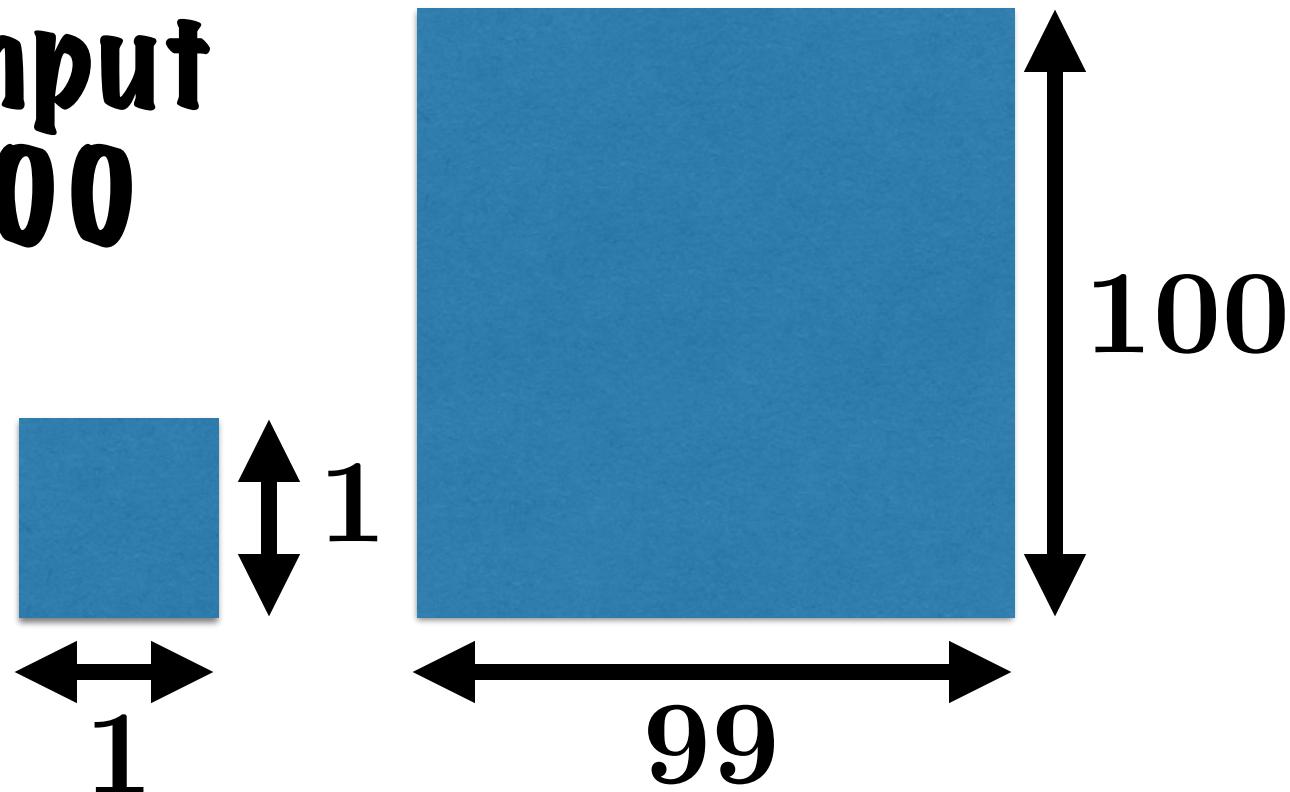
**Naive greedy algorithm:**  
take by order of  
increasing size/value



**slope**  $\frac{s_i}{v_i}$

# How good is Greedy?

Bad input  
 $B=100$



Greedy is bad

**Another general recipe:  
for intuition,  
try special cases**

If all items have the same size...

Greedy is good.

If all items have the same value...

Greedy is good.

If all items have size=value...

# Knapsack and rounding



# Knapsack and rounding

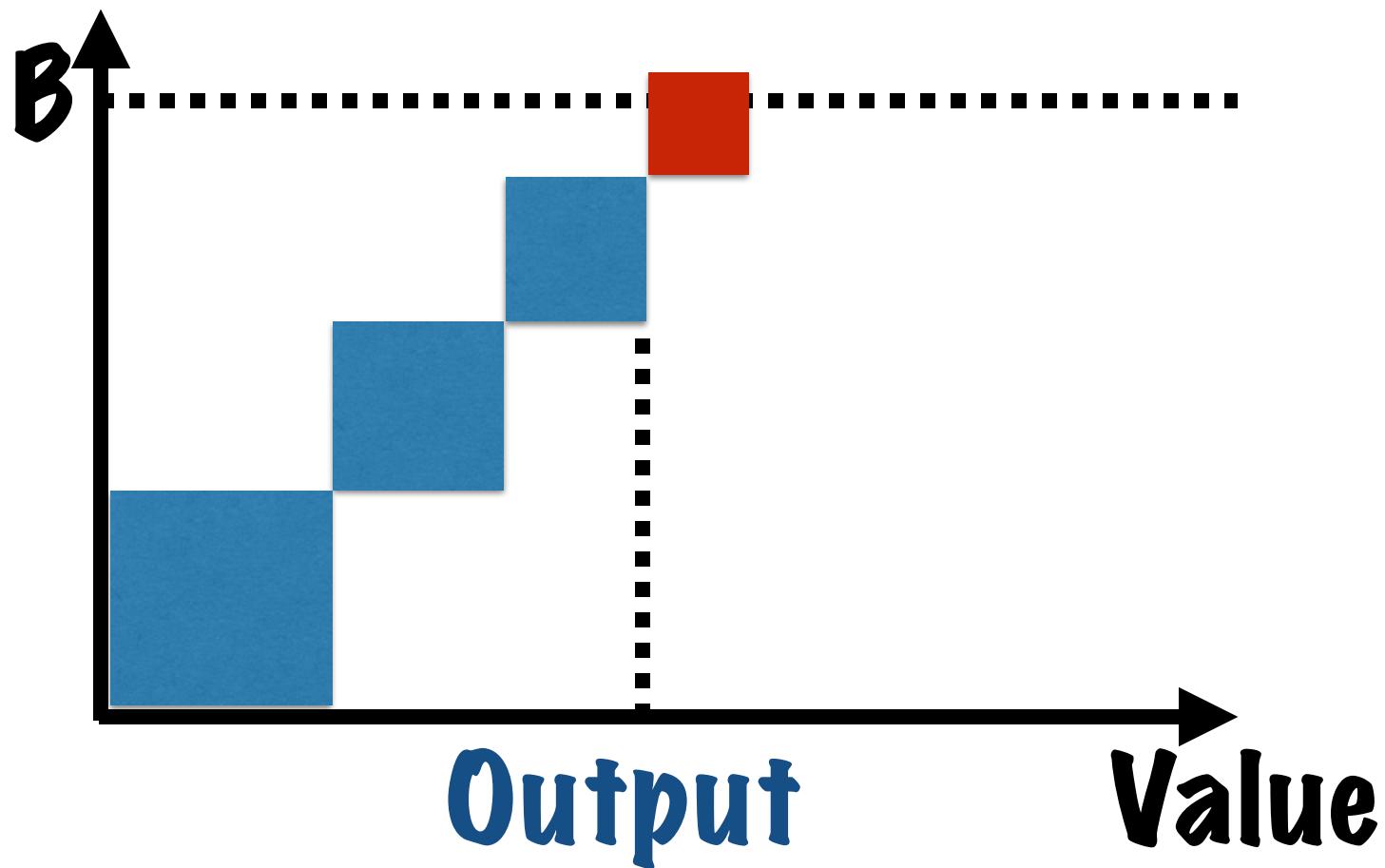


# A greedy algorithm for special case size=value

Order items by decreasing value.

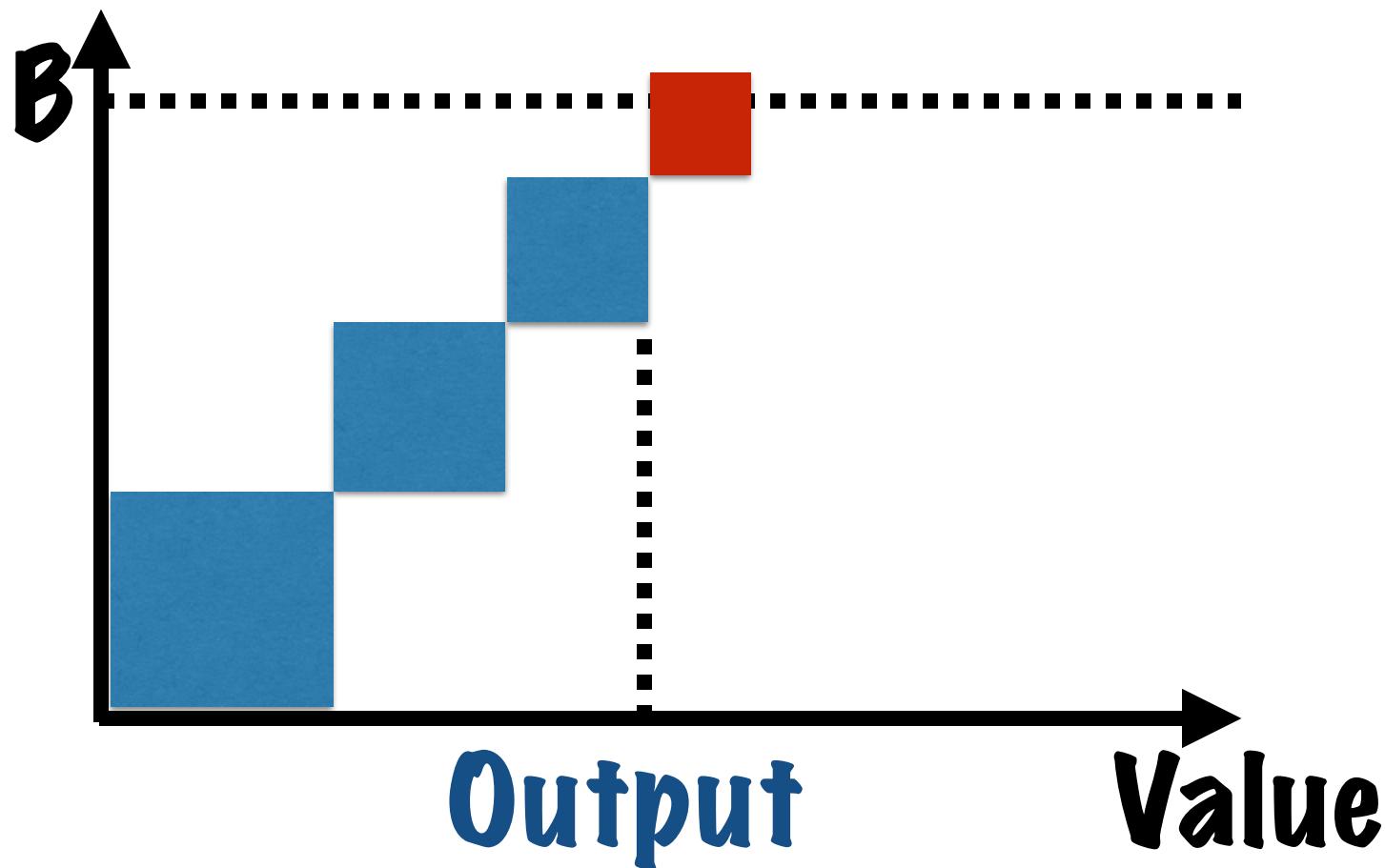
How good is that?

**Observe:**  $\text{OPT} \leq B$

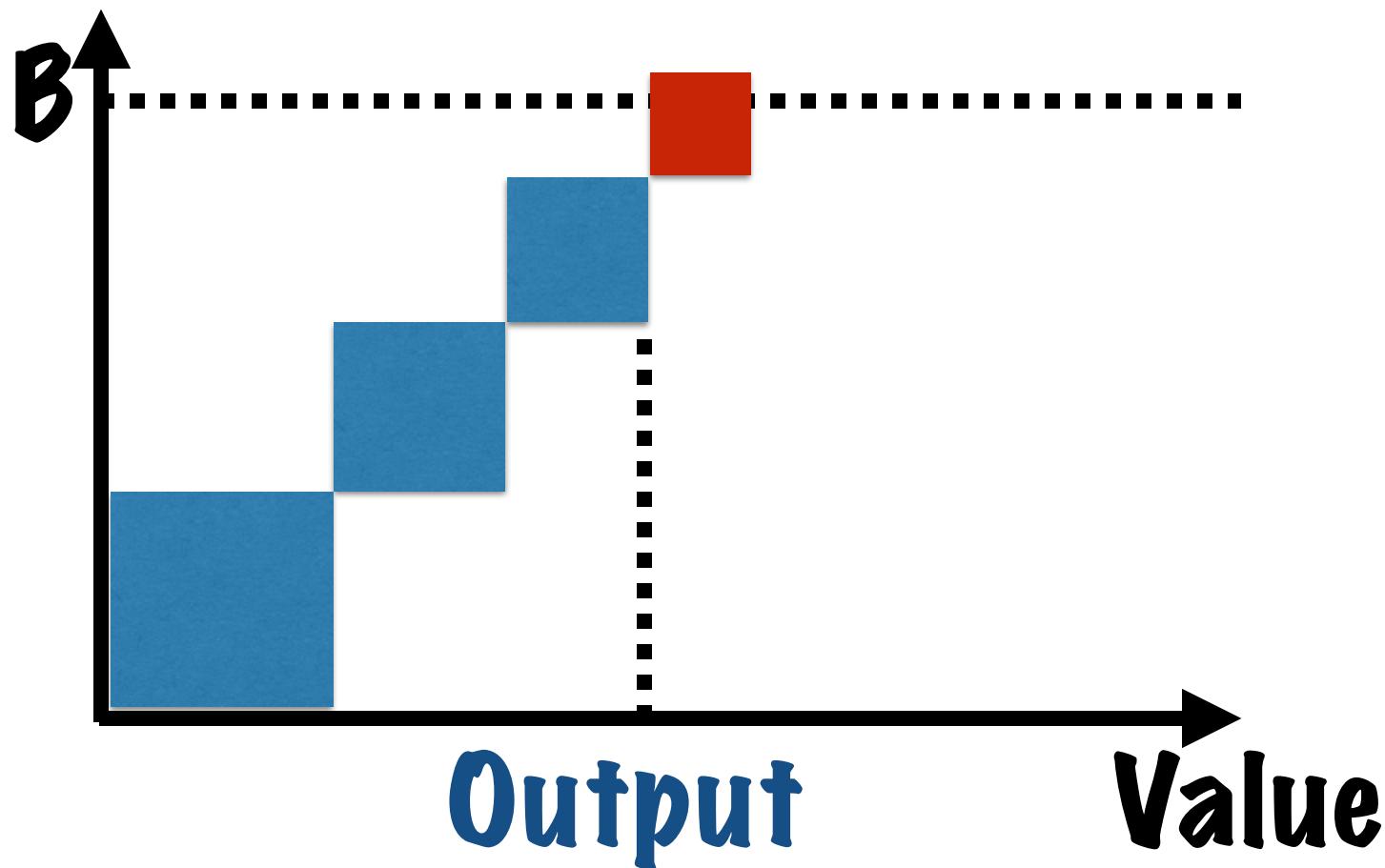


**Observe: Output+1 item > B**

**Can assume:**  
**Output has at least 1 item**



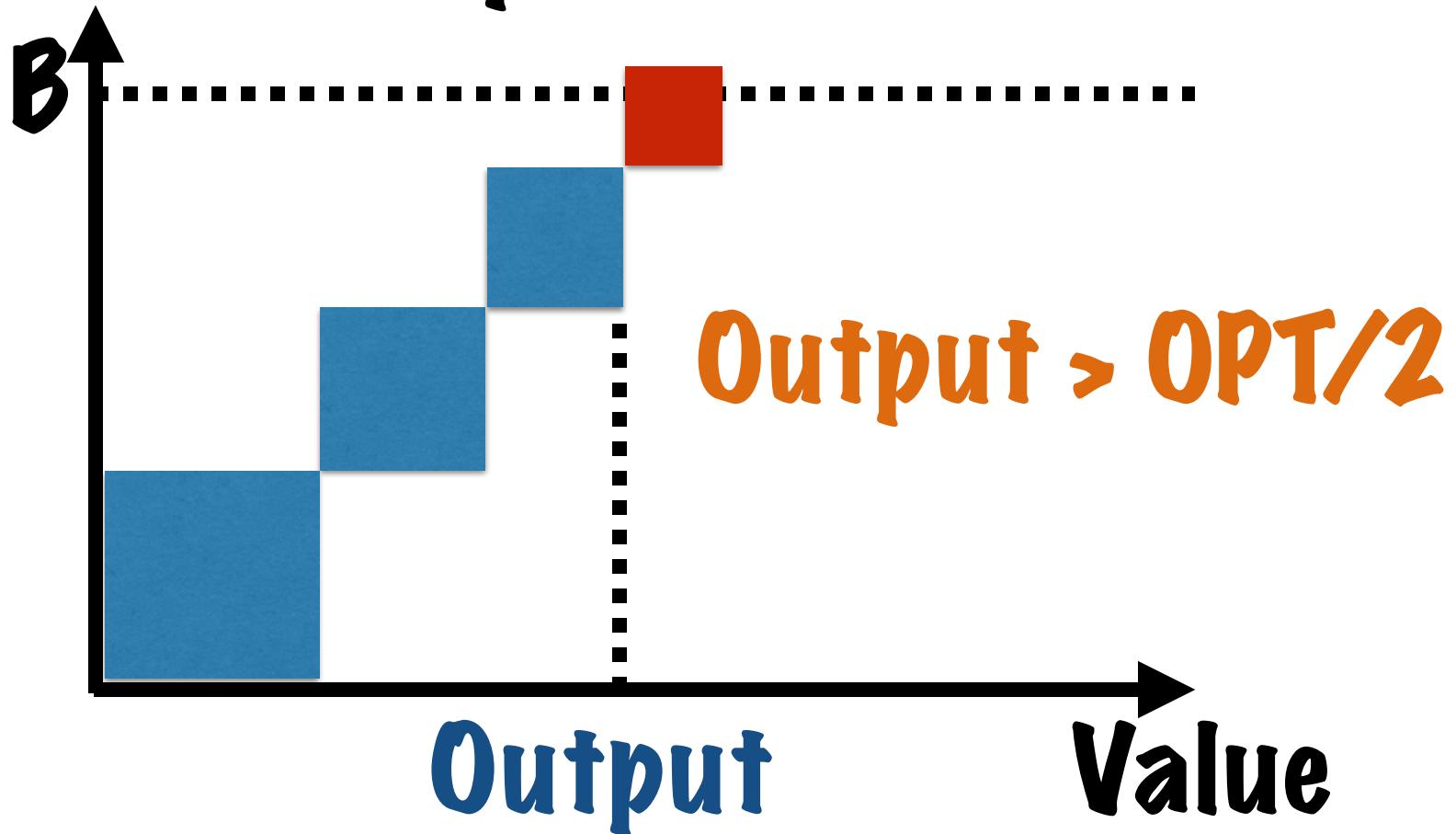
**Observe: first item in output  
is better than  
item not in output**



Combine: Output + red item > B

First output item > red item

Output > B/2



**Theorem:**  
in special case size=value,  
greedy is a 2-approximation.

**Can we do better?**

# Knapsack and rounding



# Knapsack and rounding



# **Knapsack special special case**

**All items have size = value**

$$\in \{1, 2, \dots, B\}$$

**and in addition**

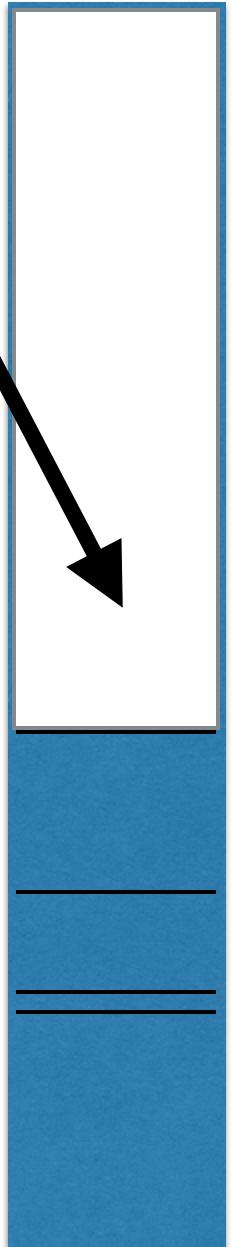
**B is a "small" integer**

# Dynamic programming

add  
stuff  
here

interface

Given partial solution  
for first  $i$  items,  
what to remember  
to complete solution optimally?



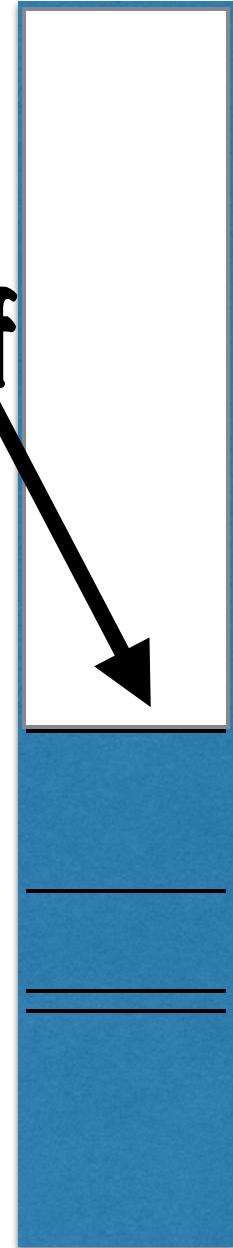
# Dynamic programming

Q: What to remember?

A: remember  
total value so far

$A[i, v]$ =whether  
 $v$  achievable with  
subset of first  $i$  items

add  
stuff  
here



**Q:**  $v$  achievable with  
subset of first  $i$  items iff...

**A:** ...it depends on  
**whether** subset contains  $i$

If not:

$v$  must be reached  
with first  $i-1$  items

If yes:

$v - v_i$  must be reached  
with first  $i-1$  items

**A[i,v]=whether  
v achievable with  
subset of first i items**

$A[i, v] =$   
 $A[i - 1, v]$  or  
 $((v \geq v_i) \text{ and } A[i - 1, v - v_i])$

# Algorithm

```
For  $v = 0 \dots B$  :  $A[1, v] \leftarrow \text{false}$ 
 $A[1, v_1] \leftarrow \text{true}, A[1, 0] \leftarrow \text{true}$ 
For  $i = 2 \dots n$ ,
    For  $v = 0 \dots v_i - 1$  :  $A[i, v] \leftarrow A[i - 1, v]$ 
    For  $v = v_i \dots B$  :
         $A[i, v] \leftarrow A[i - 1, v] \text{ or } A[i - 1, v - v_i]$ 
Output  $\max\{v : A[n, v] \text{ is true}\}$ 
```

# Knapsack and rounding



# Knapsack and rounding



**Less special special case:  
values are small integers**

**All values  
 $\in \{1, 2, \dots, N\}$**   
**N: "small" integer**

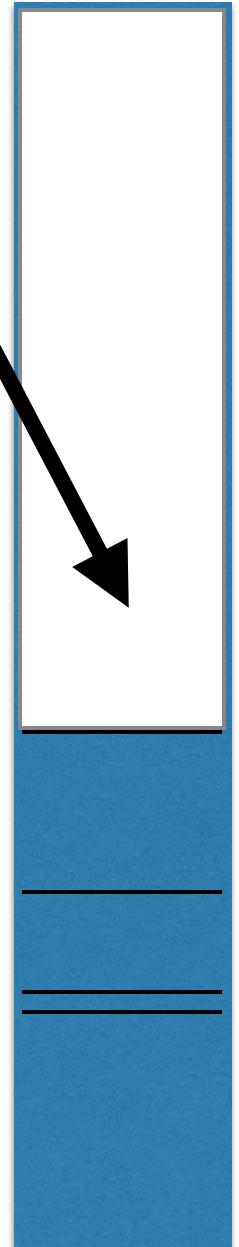
**Extend previous ideas**

# Dynamic programming

add  
stuff  
here

interface

Given partial solution  
for first  $i$  items,  
what to remember  
to complete solution optimally?



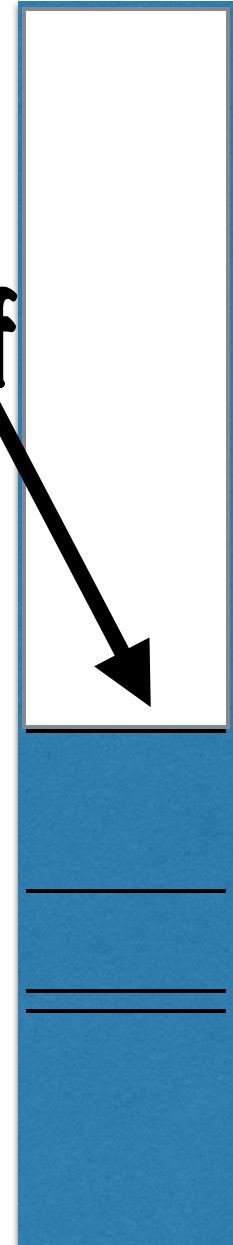
# Dynamic programming

Q: What to remember?

$A[i, v]$ =whether  
~~v achievable with~~  
subset of first i items

$A[i, v]$ =must  
remember size

add  
stuff  
here

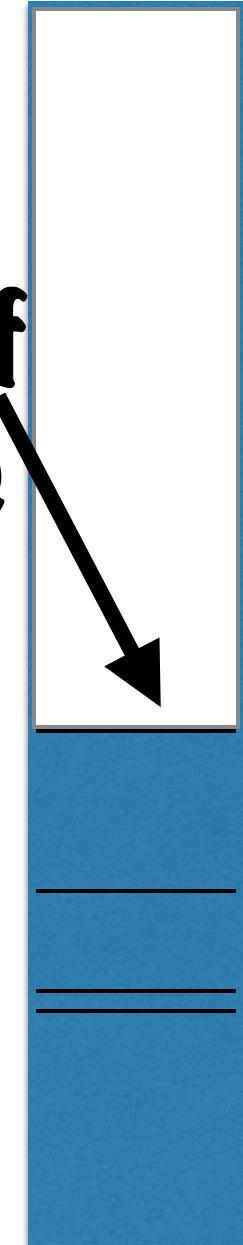


# Dynamic programming

## Q: What to remember?

$A[i, v]$  = min size achievable  
for subset of first  $i$  items  
of value  $v$

add  
stuff  
here



**Q:**  $v, s$  achievable with subset of first  $i$  items iff...

**A:** ...it depends on whether subset contains  $i$

If not:

$v, s$  reached with first  $i-1$  items

If yes:

$v - v_i, s - s_i$  reached with first  $i-1$  items

# Dynamic program

$A[i, v]$ =min size achievable  
for subset of first  $i$  items  
of value  $v$

If  $v \geq v_i$   
then  $A[i, v] =$   
 $\min(A[i - 1, v],$   
 $A[i - 1, v - v_i] + s_i)$   
else ...

For  $v = 0 \dots nN$  :  $A[1, v] \leftarrow B + 1$

$A[1, v_1] \leftarrow s_1, A[1, 0] \leftarrow 0$

For  $i = 2 \dots n$ ,

For  $v = 0 \dots v_i - 1$  :  $A[i, v] \leftarrow A[i - 1, v]$

For  $v = v_i, v_i + 1, \dots, nN$  :

$A[i, v] \leftarrow \min(A[i - 1, v], A[i - 1, v - v_i] + s_i)$

Output  $\max\{v : A[n, v] \leq B\}$

**Runtime:  $O(n^2N)$**

# Q: What's the main idea?

For  $v = 0 \dots nN$  :  $A[1, v] \leftarrow B + 1$

$A[1, v_1] \leftarrow s_1, A[1, 0] \leftarrow 0$

For  $i = 2 \dots n$ ,

For  $v = 0 \dots v_i - 1$  :  $A[i, v] \leftarrow A[i - 1, v]$

For  $v = v_i, v_i + 1, \dots, nN$  :

$A[i, v] \leftarrow \min(A[i - 1, v], A[i - 1, v - v_i] + s_i)$

Output  $\max\{v : A[n, v] \leq B\}$



# A: The definition of $A[i, v]$

dynamic program key step =  
DP table definition

# Knapsack and rounding

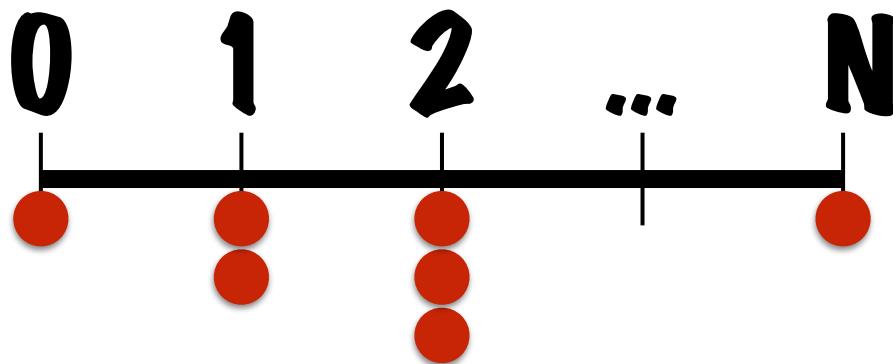


# Knapsack and rounding



# General case: algorithm

We already have an algorithm  
when values  
are small integers



Idea:  
modify input  
so that values are  
small integers

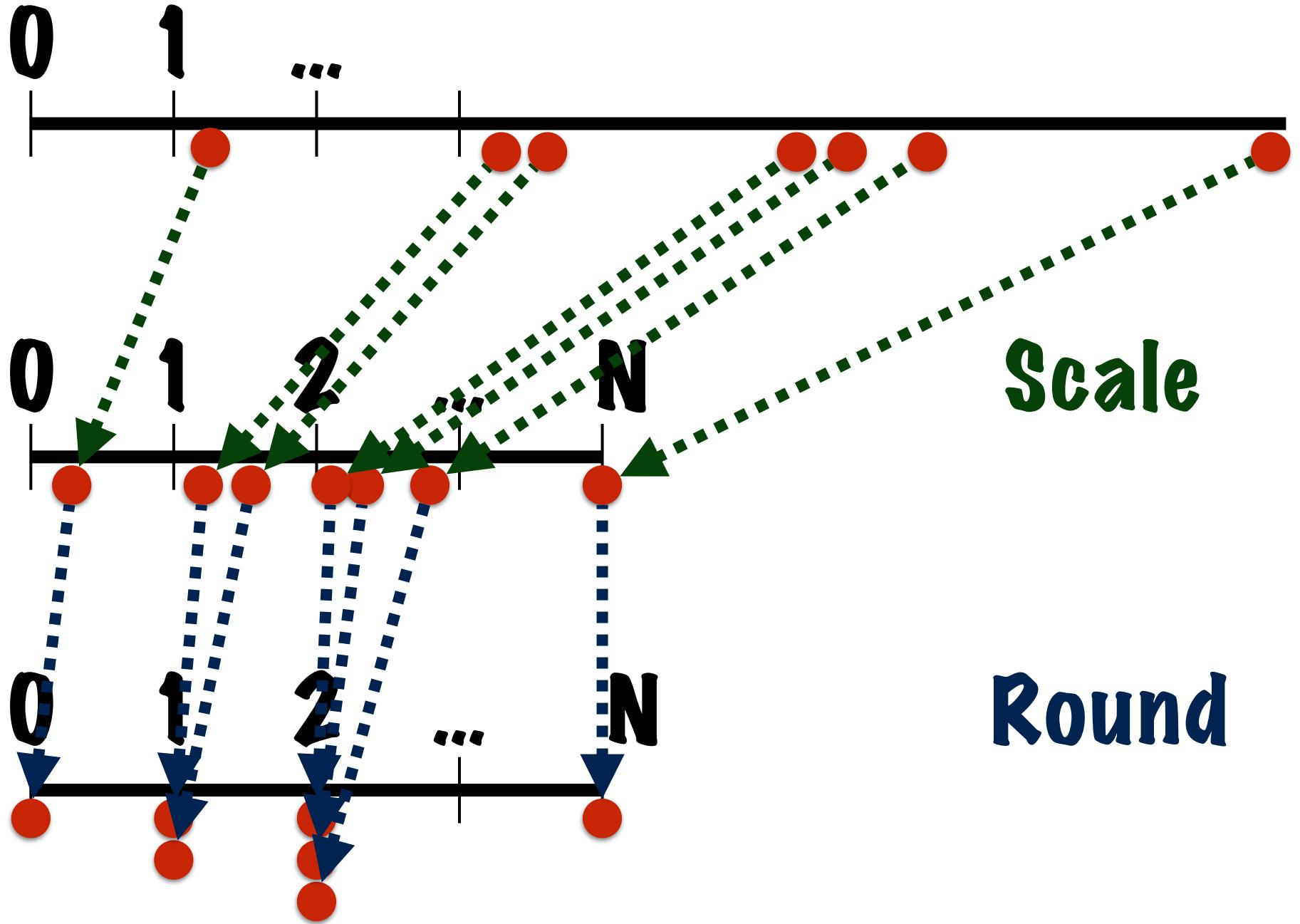
**Q : How to modify input  
so that values are  
small integers?**

**A : Scale and round!**

**Discard items that don't fit**

1. **Multiply each value by something so that values are small**
2. **Round values to integers**

**Dynamic program for the scaled and rounded problem**



Scale

Round

**Effect of scaling:  
Max scaled value = N**

**Multiply by**  $\alpha = \frac{N}{\max v_i}$

**How do we pick N?**

- **small enough that the runtime is polynomial**
- **large enough that the resulting set of items has value close to the optimum**

# Algorithm Scale

1. Discard items not fitting
2. Let  $N=100 n$
3. Let  $v'_i \leftarrow \lfloor v_i \times \frac{N}{\max_j v_j} \rfloor$
4. Apply DP to  $B, (s_i, v'_i)_i$
5. Output corresponding items

# Runtime

For  $v = 0 \dots nN : A[1, v] \leftarrow B + 1$

$A[1, v_1] \leftarrow s_1, A[1, 0] \leftarrow 0$

For  $i = 2 \dots n,$

For  $v = 0 \dots v_i - 1 : A[i, v] \leftarrow A[i - 1, v]$

For  $v = v_i, v_i + 1, \dots, nN :$

$A[i, v] \leftarrow \min(A[i - 1, v], A[i - 1, v - v_i] + s_i)$

Output  $\max\{v : A[n, v] \leq B\}$

$$N = 100 \times n \implies n^2 N = 100 \times n^3$$

# Knapsack and rounding



# Knapsack and rounding



# Analysis

- **S:** output items
- **DP:** S optimal for **scaled rounded** input
- **S\***: optimal items
- **Scaling:** S\* optimal for **scaled unrounded** input

## Relate S for original and modified input value

output value



$$\text{Value}(S) = \sum_S v_i$$

$$= \frac{1}{\alpha} \sum_S (\alpha v_i)$$

$$\geq \frac{1}{\alpha} \sum_S v'_i$$

value for scaled&rounded input

# Relate $S$ to $S^*$ on modified input

$$\sum_S v'_i \geq \sum_{S^*} v'_i$$



$S$  optimal for scaled&rounded

# Relate $S^*$ for original and modified input value

$$v'_i > \alpha v_i - 1$$



effect of rounding

Sum:

$$\sum_{S^*} v'_i > \alpha \sum_{S^*} v_i - n$$

## Combine and substitute:

$$\text{Value}(S) \geq \frac{1}{\alpha} \sum_S v'_i$$

$$\geq \frac{1}{\alpha} \sum_{S^*} v'_i$$

$$\geq \frac{1}{\alpha} [\alpha \sum_{S^*} v_i - n]$$

$$= \text{OPT} - \frac{n}{\alpha}$$

$$= \text{OPT} - \frac{n \times \max v_i}{N}$$

# Lower bound OPT

Discarded items that don't fit:

$$\text{OPT} \geq \max v_i$$

# Wrapping up

$$\text{Value}(S) \geq \text{OPT} - \frac{n}{N} \text{OPT}$$

$$N = 100 \times n$$

$$\text{Value}(S) \geq .99 \times \text{OPT}$$

**Theorem: Solution to knapsack  
with value at least .99 OPT  
and runtime  $O(\text{poly}(n))$**

# Knapsack and rounding



# Knapsack and rounding



# Approximation schemes

# The general algorithm

1. Let  $N=1000 n$
2. Let  $v'_i \leftarrow \lfloor v_i \times \frac{N}{\max_j v_j} \rfloor$
3. Apply DP to  $B, (s_i, v'_i)_i$
4. Output corresponding items

**Theorem:** this gives a solution  
to knapsack  
with value at least .999 OPT  
and with runtime  $O(\text{poly}(n))$

**Approximation scheme : family of  
algorithms:**

**One for each  $\epsilon > 0$**   
**runtime polynomial in input**  
**output value is near-optimal:**

$$|\text{Value}(\text{Output}) - \text{OPT}| \leq \epsilon \times \text{OPT}$$

**Theorem: knapsack  
has an approximation scheme**

# How $\epsilon$ comes in

**Q:** The smaller  $\epsilon$ , the closer to OPT. Why not let it go to 0 and find the exact OPT in  $O(\text{poly}(n))$ ?

**A:** Runtime increases as  
 $\epsilon$  goes to zero.

Runtime has N with  $N=n/\epsilon$  so  
it goes to infinity.

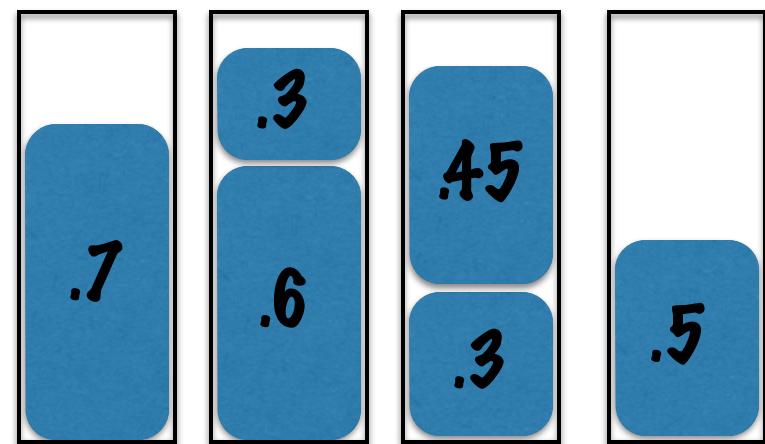
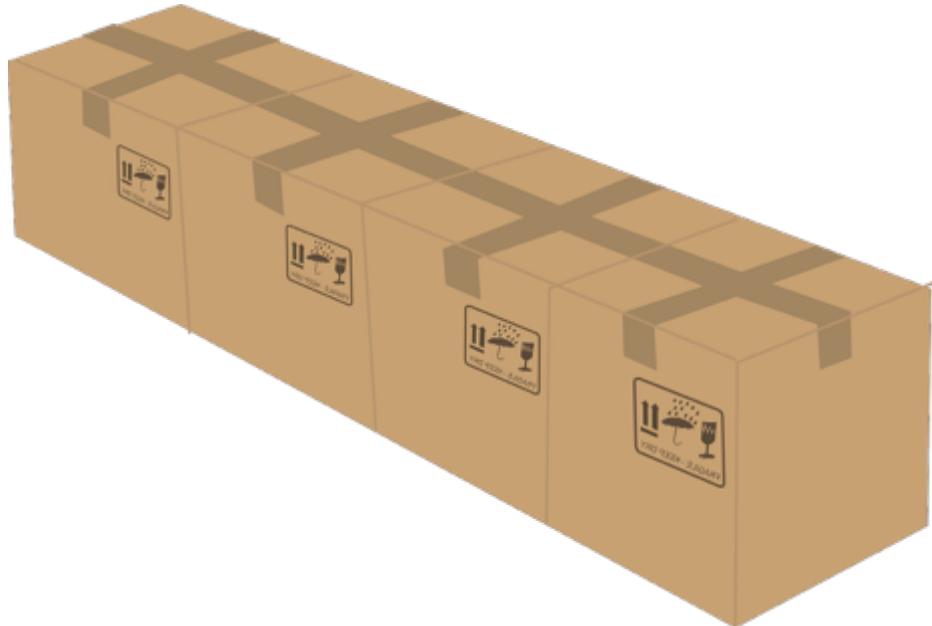
## **Method:**

- 1. Simplify the input**
- 2. Design algorithm for “simple” inputs**

# Knapsack and rounding

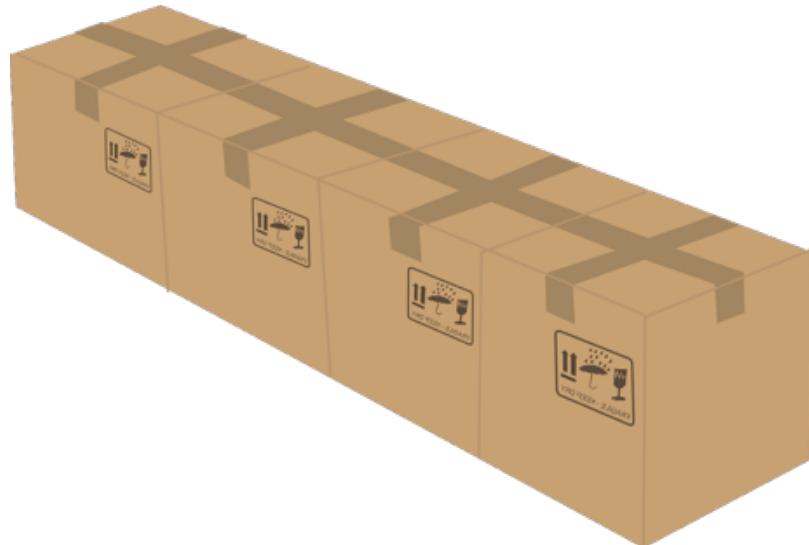


# Bin packing, linear programming and rounding



# The bin packing problem

Pack your items  
using  
as few bins as possible



**Given n items  
item i has size  $s_i < 1$   
pack items into the fewest  
unit capacity bins**

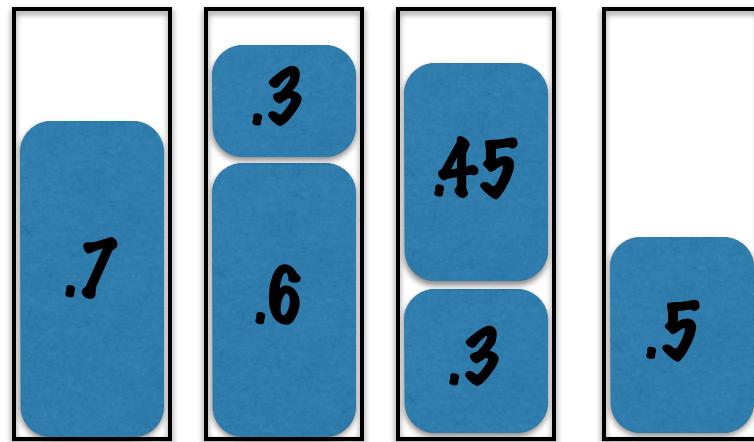


# The Next Fit algorithm

One bin at a time:  
If next item does not fit,  
close the bin and  
open a new bin

“Next Fit” algorithm

$s1=.7$   
 $s2=.6$   
 $s3=.4$   
 $s4=.3$   
 $s5=.45$   
 $s6=.5$



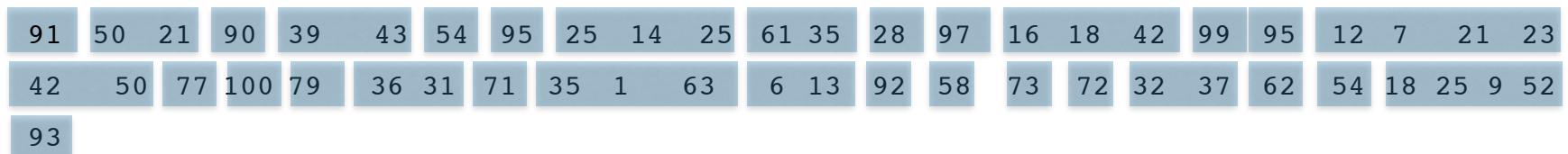
can this instance  
be packed better?

# How good is Next Fit?

Capacity 100. Items:

|     |    |    |    |    |    |    |    |    |    |    |    |    |    |
|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 91  | 50 | 21 | 90 | 39 | 43 | 54 | 95 | 25 | 14 | 25 | 61 | 35 |    |
| 28  | 97 | 16 | 18 | 42 | 99 | 95 | 12 | 7  | 21 | 23 | 42 | 50 | 77 |
| 100 | 79 | 36 | 31 | 71 | 35 | 1  | 63 | 6  | 13 | 92 | 58 | 73 | 72 |
| 32  | 37 | 62 | 54 | 18 | 25 | 9  | 52 | 93 |    |    |    |    |    |

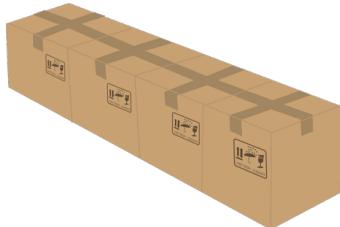
Next Fit:



used 31 bins

|    |    |    |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 91 | 50 | 21 | 90  | 39 | 43 | 54 | 95 | 25 | 14 | 25 | 61 | 35 | 28 | 97 | 16 | 18 | 42 | 99 | 95 | 12 | 7  | 21 | 23 |    |
| 42 | 50 | 77 | 100 | 79 | 36 | 31 | 71 | 35 | 1  | 63 | 6  | 13 | 92 | 58 | 73 | 72 | 32 | 37 | 62 | 54 | 18 | 25 | 9  | 52 |
| 93 |    |    |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**bin 7:  $(25+14+25)$**   
**next item: 61, but**  
 **$(25+14+25) + 61 > 100$**   
**so, close bin 7, open bin 8,**  
**put item 61 in bin 8.**



|    |    |    |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 91 | 50 | 21 | 90  | 39 | 43 | 54 | 95 | 25 | 14 | 25 | 61 | 35 | 28 | 97 | 16 | 18 | 42 | 99 | 95 | 12 | 7  | 21 | 23 |    |
| 42 | 50 | 77 | 100 | 79 | 36 | 31 | 71 | 35 | 1  | 63 | 6  | 13 | 92 | 58 | 73 | 72 | 32 | 37 | 62 | 54 | 18 | 25 | 9  | 52 |
| 93 |    |    |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

In general:

(items in bin  $2i-1$ ) + next item  $> 100$

(items in bins  $2i-1$  or  $2i$ )  $> 100$

31 bins: total item sizes  $> 15 * 100$



|    |    |    |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 91 | 50 | 21 | 90  | 39 | 43 | 54 | 95 | 25 | 14 | 25 | 61 | 35 | 28 | 97 | 16 | 18 | 42 | 99 | 95 | 12 | 7  | 21 | 23 |    |
| 42 | 50 | 77 | 100 | 79 | 36 | 31 | 71 | 35 | 1  | 63 | 6  | 13 | 92 | 58 | 73 | 72 | 32 | 37 | 62 | 54 | 18 | 25 | 9  | 52 |
| 93 |    |    |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

In general:  
**k bins by Next Fit**  
**Total item sizes > (k-1)/2 \* 100**



|    |    |    |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 91 | 50 | 21 | 90  | 39 | 43 | 54 | 95 | 25 | 14 | 25 | 61 | 35 | 28 | 97 | 16 | 18 | 42 | 99 | 95 | 12 | 7  | 21 | 23 |    |
| 42 | 50 | 77 | 100 | 79 | 36 | 31 | 71 | 35 | 1  | 63 | 6  | 13 | 92 | 58 | 73 | 72 | 32 | 37 | 62 | 54 | 18 | 25 | 9  | 52 |
| 93 |    |    |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

# What about OPT?

Total item sizes <  $\text{OPT} * 100$



|    |    |    |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 91 | 50 | 21 | 90  | 39 | 43 | 54 | 95 | 25 | 14 | 25 | 61 | 35 | 28 | 97 | 16 | 18 | 42 | 99 | 95 | 12 | 7  | 21 | 23 |    |
| 42 | 50 | 77 | 100 | 79 | 36 | 31 | 71 | 35 | 1  | 63 | 6  | 13 | 92 | 58 | 73 | 72 | 32 | 37 | 62 | 54 | 18 | 25 | 9  | 52 |
| 93 |    |    |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Combining:**  
**k bins by Next Fit**  
 $\text{OPT} * 100 > (k-1)/2 * 100$   
 $\#(\text{bins of Next Fit}) < 2 * \text{OPT} + 1$



**Asymptotic 2 approximation**

# Is this tight?

Example with  
 $\text{OPT}=501$  bins,  
 $\text{Next Fit}=1000$  bins

# What about non-asymptotic?

Distinguishing between  
 $\text{OPT}=2$  and  $\text{OPT}=3$   
is NP-hard

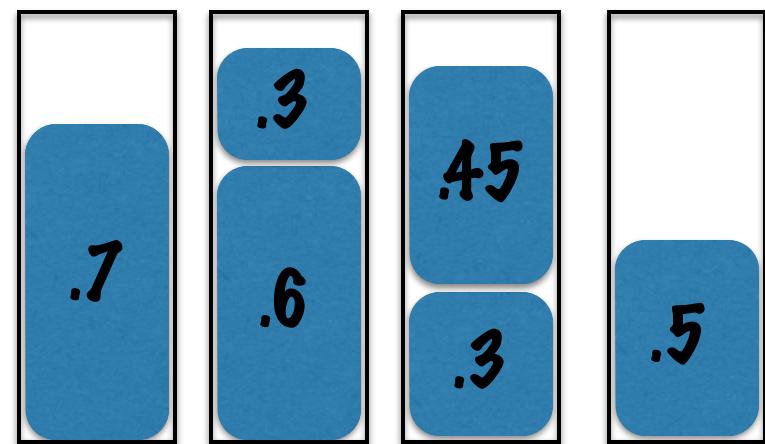
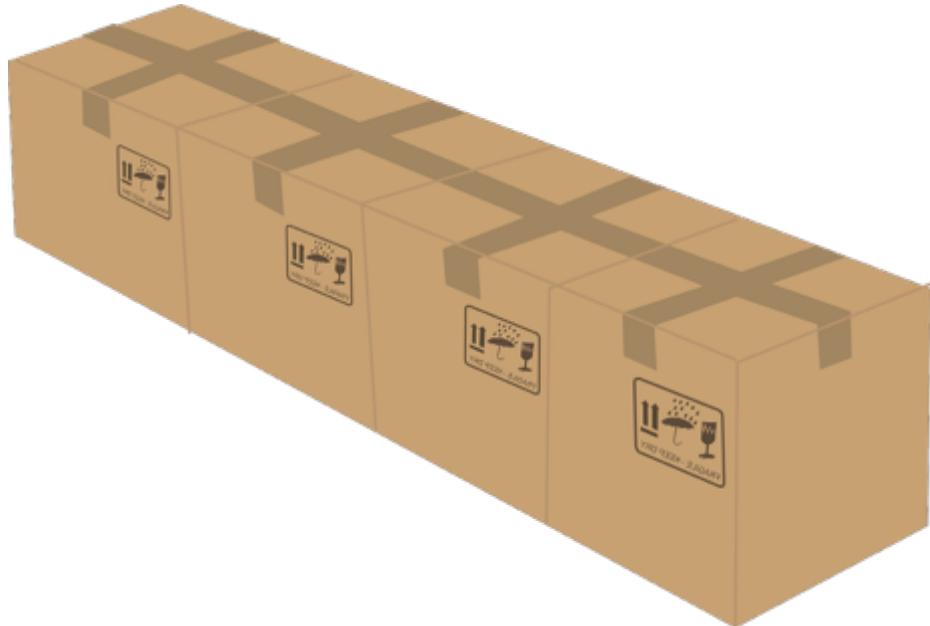
# What have we learned?

1. Crude algorithms can give good bounds

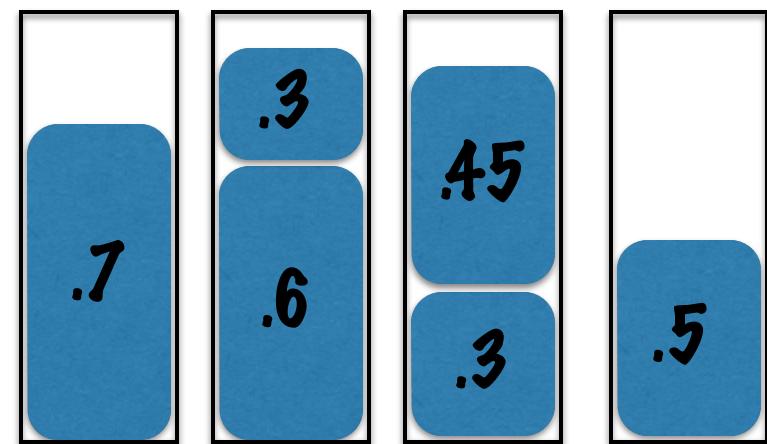
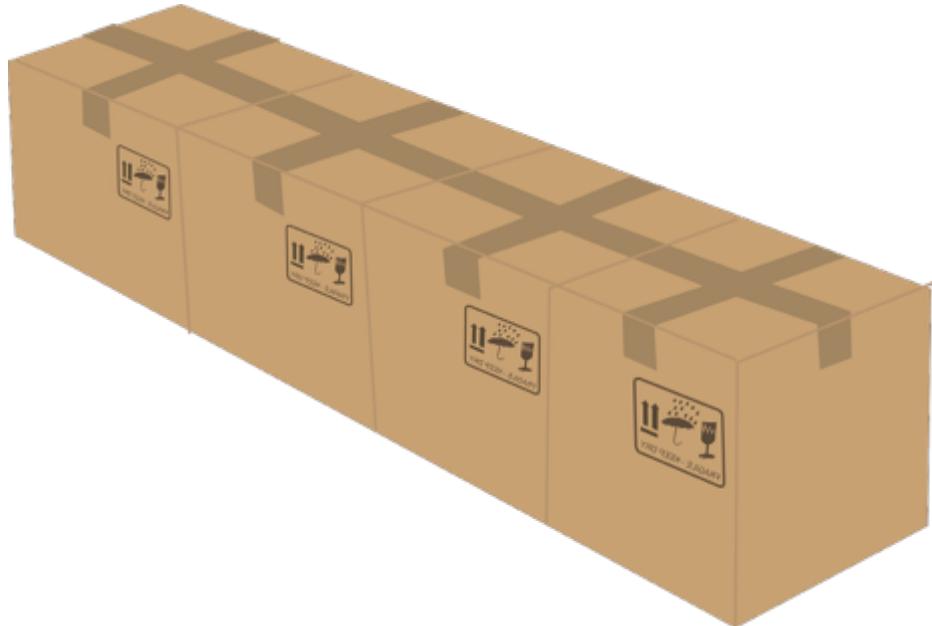
**Message:** first try the simplest algorithm

2. Analysis: for intuition, first execute it on some concrete examples

# Bin packing, linear programming and rounding



# Bin packing, linear programming and rounding



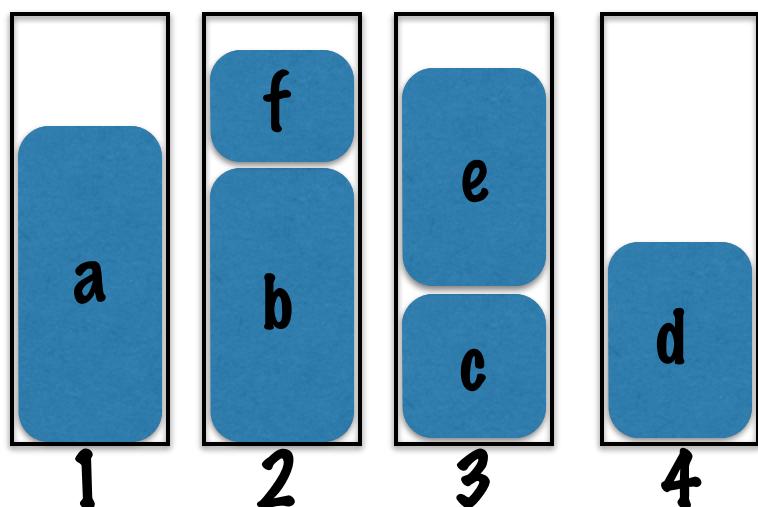
Can we do better than  
Next Fit?

First tool: linear programming  
relaxation

# An integer program

Given  $n$  items and  $K$  unit bins,  
is there a packing?

**Variables:**  $x_{ij} \in \{0, 1\}$   
 $x_{ij} = 1$  iff item  $i$  is placed in bin  $j$



$x_{a1} = x_{b2} =$   
 $x_{f2} = x_{c3} =$   
 $x_{e3} = x_{d4} = 1,$   
 $x_{ij} = 0$  otherwise

# An integer program

Constraint: every item  
must go somewhere

Item b must go into bin 1,2,3 or 4

$$x_{b1} + x_{b2} + x_{b3} + x_{b4} = 1$$

# An integer program

Constraint: must not exceed bin capacity

Item sizes in bin  $j$  sum to at most 1.

$$x_{aj}s_a + x_{bj}s_b + \dots + x_{fj}s_f \leq 1$$

# Integer program

$n$  items,  $K$  bins

$x_{ij}$  = whether item  $i$  goes into bin  $j$

$$\forall i : \sum_j x_{ij} = 1$$

$$\forall j : \sum_i x_{ij} s_i \leq 1$$

$$\forall i, j : x_{ij} \in \{0, 1\}$$

feasible iff items can be packed into  $K$  bins

# Linear programming relaxation

$n$  items,  $K$  bins

$$\forall i : \sum_j x_{ij} = 1$$

$$\forall j : \sum_i x_{ij} s_i \leq 1$$

$$\forall i, j : 0 \leq x_{ij} \leq 1$$

**Algorithm:** use the LP relaxation  
to pack items (somehow)

**How good is the relaxation?**

# A bad example

$n$  items,  $K$  bins

$$\forall i : \sum_j x_{ij} = 1$$

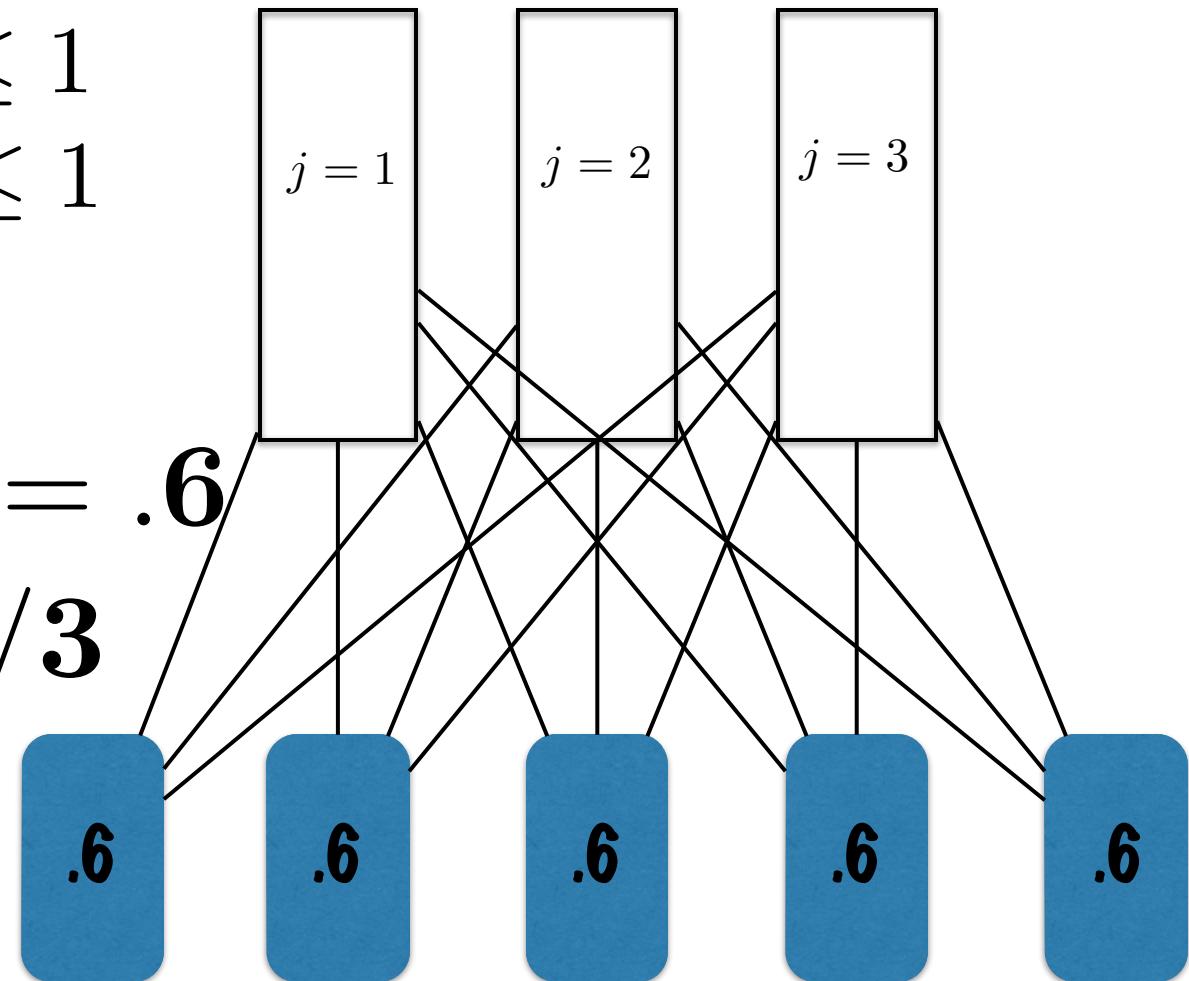
$$\forall j : \sum_i x_{ij} s_i \leq 1$$

$$\forall i, j : 0 \leq x_{ij} \leq 1$$

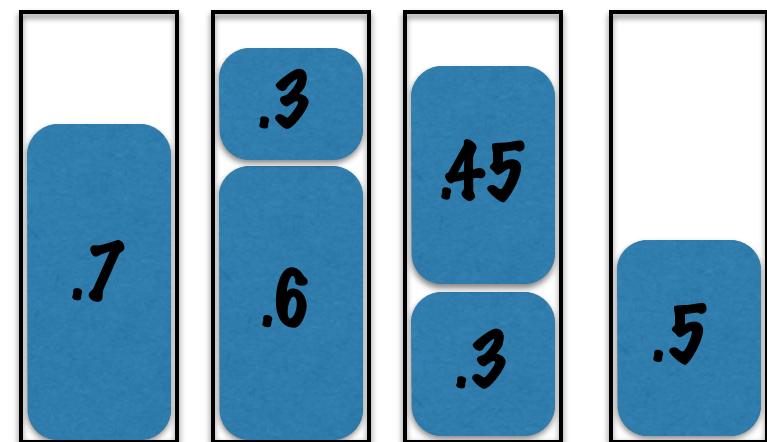
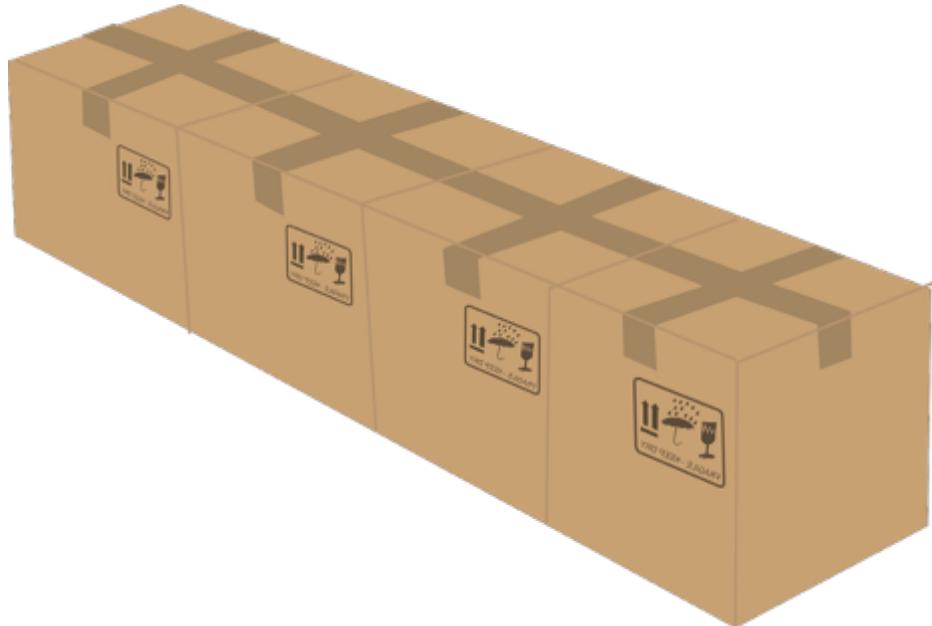
$$s_1 = \dots = s_5 = .6$$

$$\forall i, j \quad x_{ij} = 1/3$$

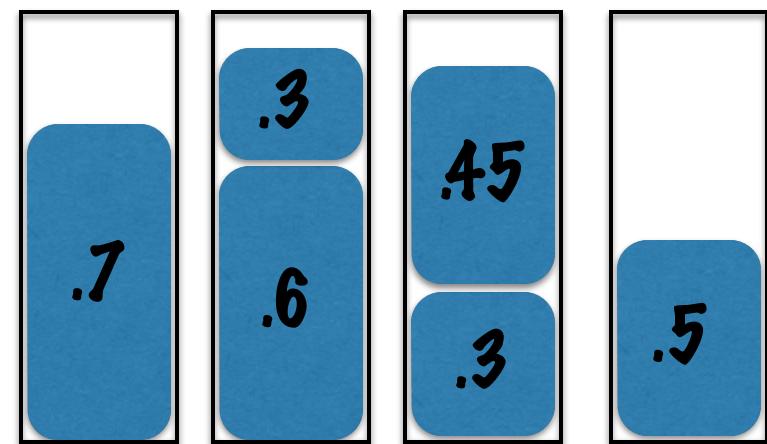
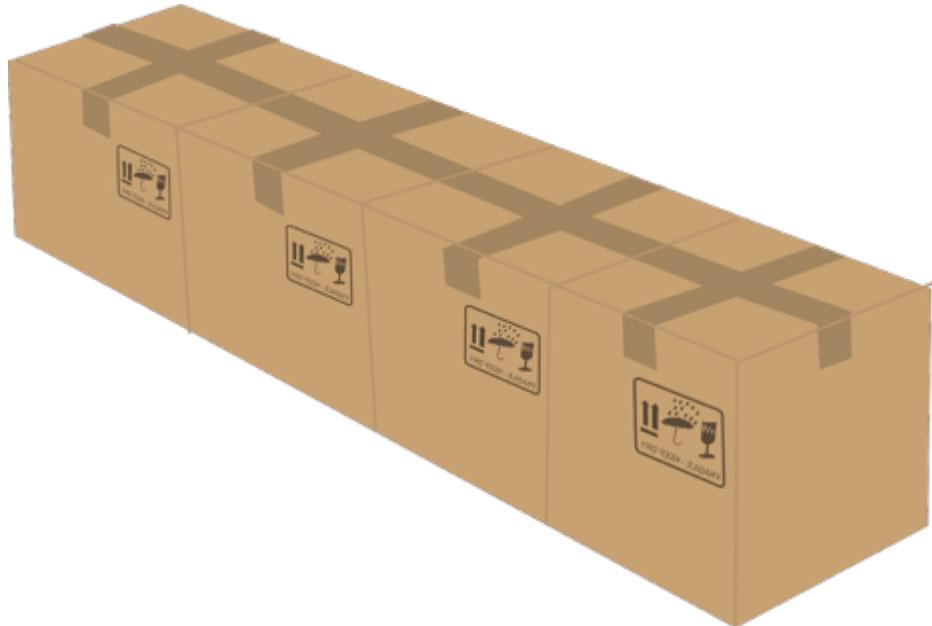
**LP: 3 bins**  
**OPT: 5 bins**



# Bin packing, linear programming and rounding



# Bin packing, linear programming and rounding



**Remember:**  
**To analyze output vs. OPT,  
focus on LP value...**

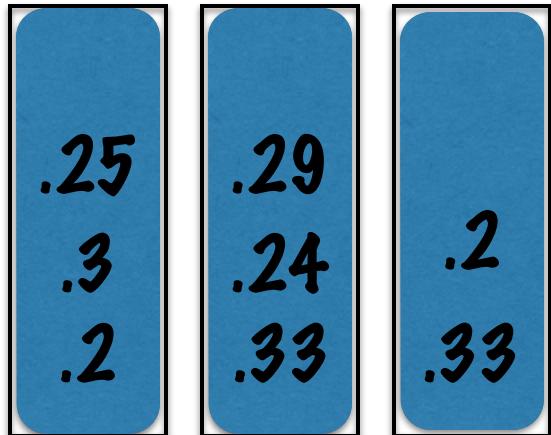


**Try next meta-tool:  
special cases**

What if items are smaller than  
1/3\*capacity?

.2,.3,.25,.33,.24,.29,.33,.2,...

What does Next Fit do?



$$.2 + .3 + .25 = .75$$

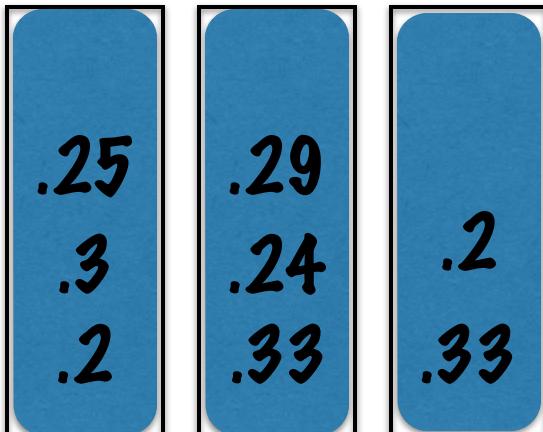
$$.33 + .24 + .29 = .86$$

$$.33 + .2 = .53$$

The next item will fit in bin 3:  
 $.53 + (\text{something less than } 1/3) < 1$

Next Fit when items are smaller than  
 $1/3 * \text{capacity}$

Bin filled to  $< 2/3$  : next item fits  
so:  
only close bin when filled to  $> 2/3$



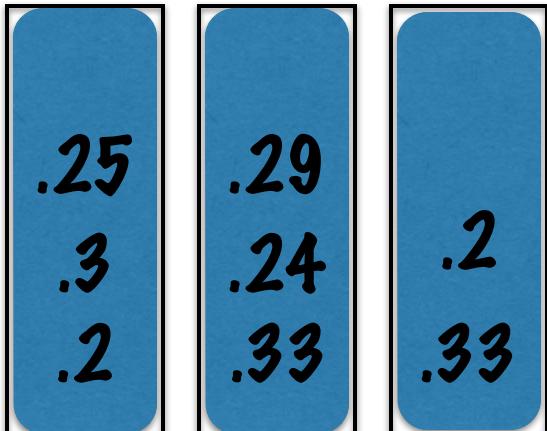
All bins except last  
are filled to  $> 2/3$

Next Fit when items are smaller than  
 $1/3 * \text{capacity}$

All bins except last  
are filled to  $> 2/3$

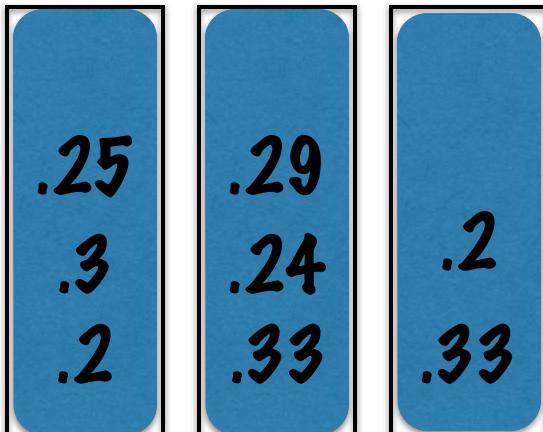
Total size  $> 2/3 * (\#\text{bins} - 1)$

But Total size  $< \text{OPT}$



Combine:  
 $\#\text{bins} < 3/2 * \text{OPT} + 1$

**Theorem:**  
when items are smaller than  
 $1/3 * \text{capacity}$ ,  
Next Fit uses at most  
 $1 + (3/2)$  OPT bins



# Message

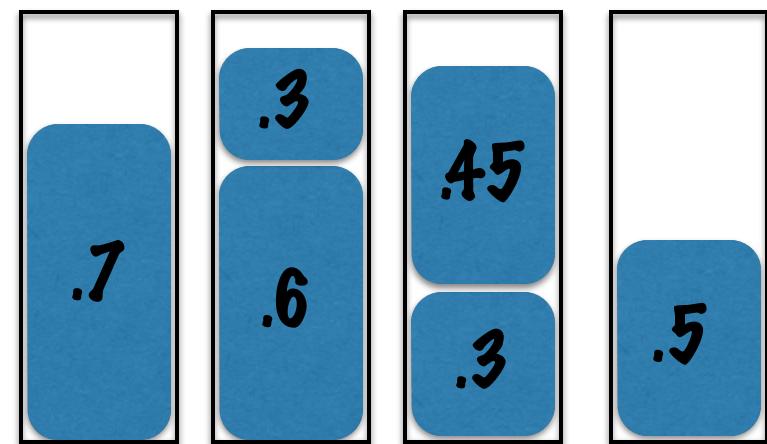
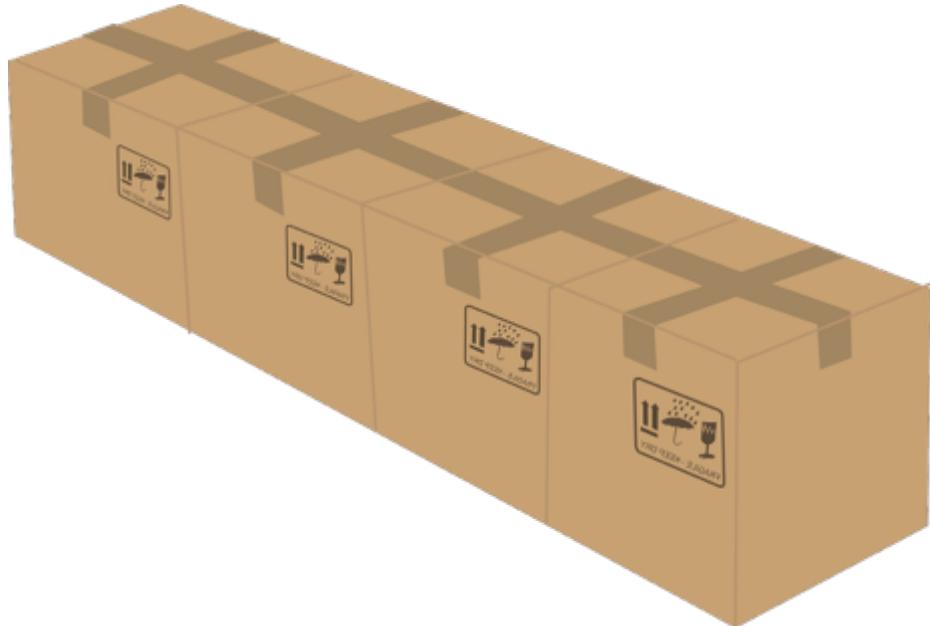
1. From example to structural observation
2. With observation, upper bound algorithm
3. With different argument, lower bound OPT
4. Combine

.25  
.3  
.2

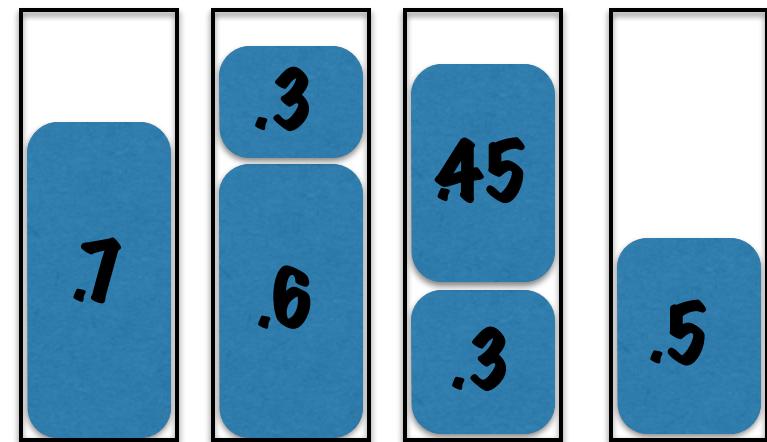
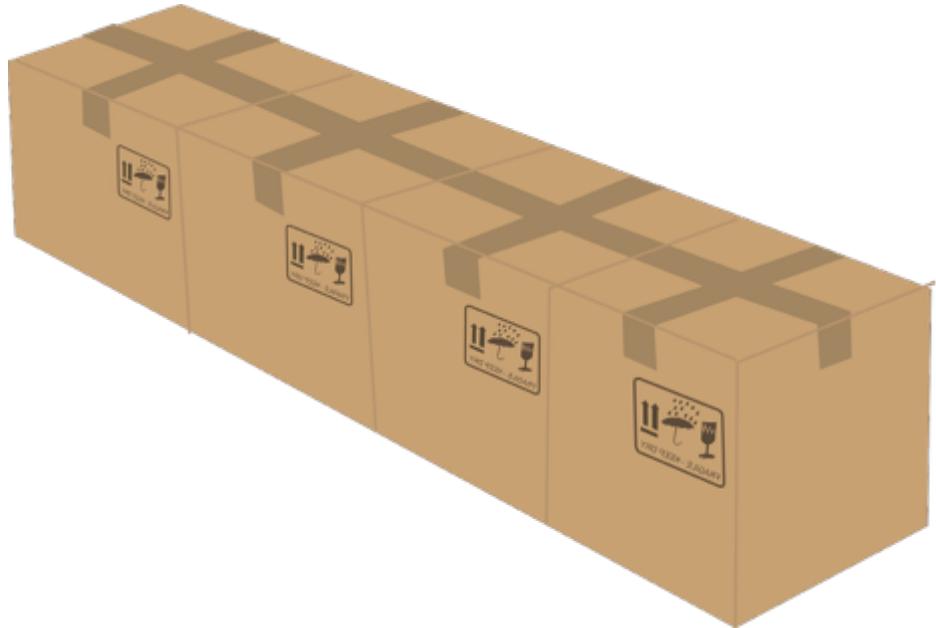
.29  
.24  
.33

.2  
.33

# Bin packing, linear programming and rounding



# Bin packing, linear programming and rounding



# Meta-tool: special cases

Large items

**Special special case**

**Large items,  
few distinct sizes**

# Example

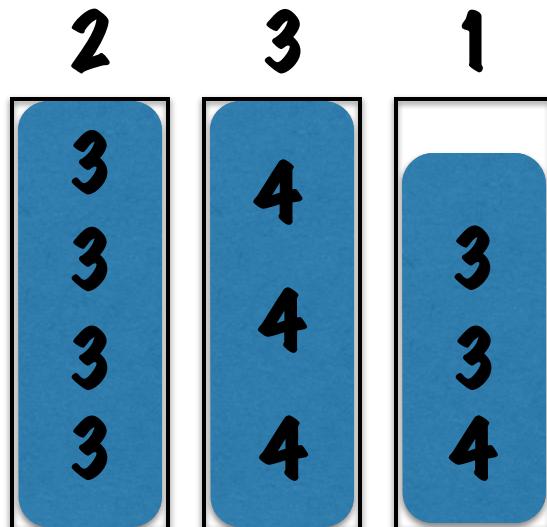
Bin capacity 12

sizes: { 3, 4 }

10 items of size 3,  
10 items of size 4.

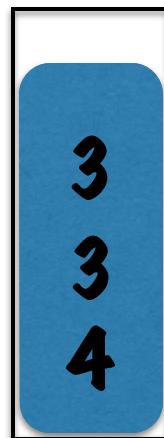
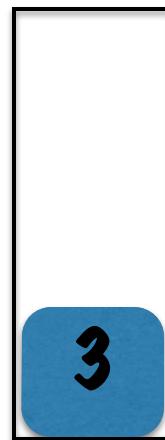
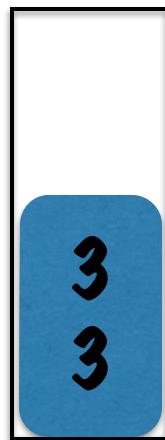
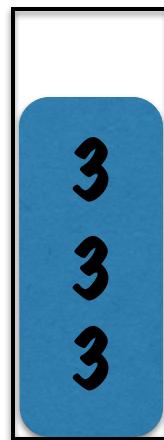
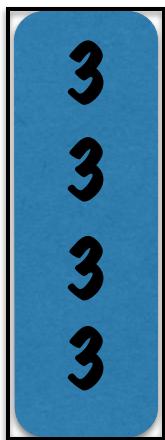
**Bin capacity 12  
sizes: { 3, 4 }**

**10 items of size 3,  
10 items of size 4.**

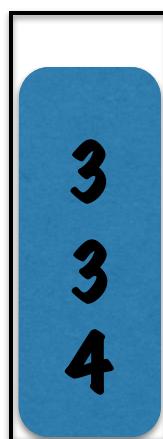
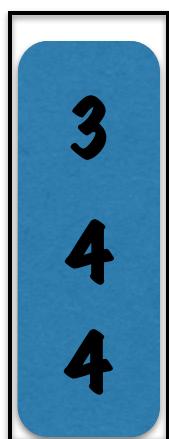
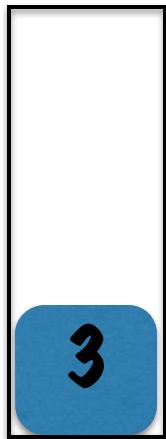
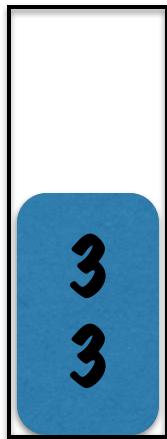
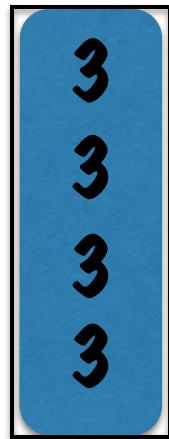




Observe:  
few configurations



Large items,  
few distinct sizes  
 $C=\{\text{configurations}\}$



In configuration  $c$   
size  $s$  occurs  
 $a_{s,c}$  times

# Integer program

Input:  $S = \{size\}$

number of items of size  $s$ :  $n_s$

Output:  $C = \{configurations\}$

number of bins in configuration  $c$ :  $x_c$

Constraints:  $\sum_c a_{s,c} x_c \geq n_s$

Number of bins:  $\sum_c x_c$

integer

If size > capacity/10 then:  
< 10 items per configuration

If < 10 sizes then:  
<  $10^{10}$  configurations

Solve LP relaxation  
10 constraints,  $10^{10}$  variables  
Round up to nearest integer

#bins < OPT +  $10^{10}$

**(Exhaustive search also ok)**

For every  $(x_c)_{c \in C} \in \{0, 1, \dots, n\}^{|C|}$

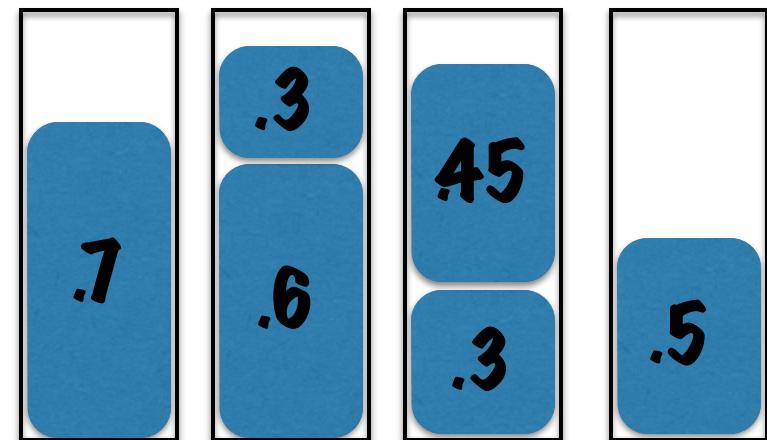
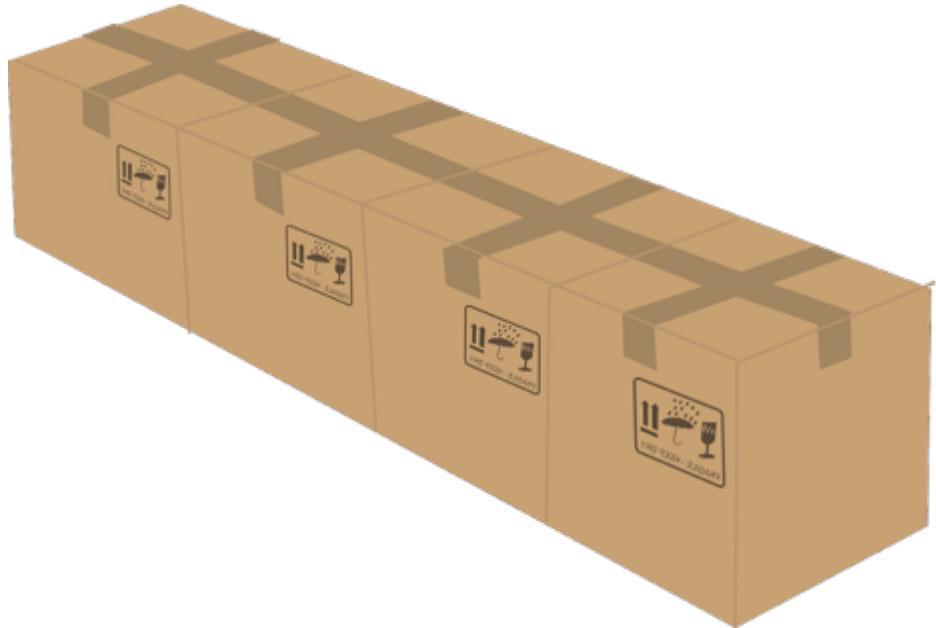
Check whether, for every size  $s$ ,  
enough slots for items of size  $s$

Output solution with min #bins

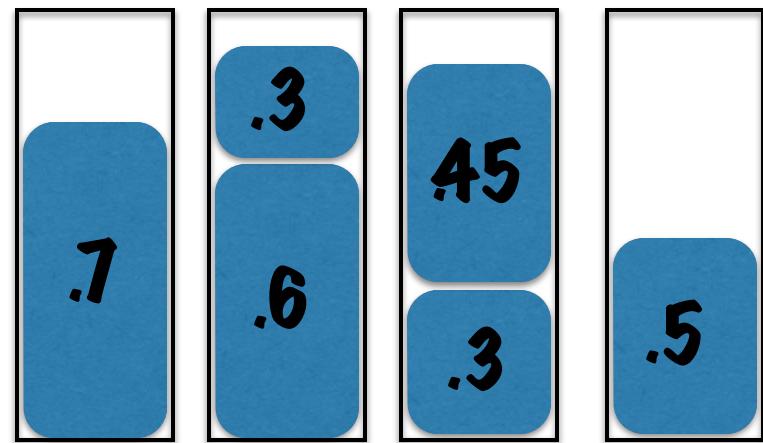
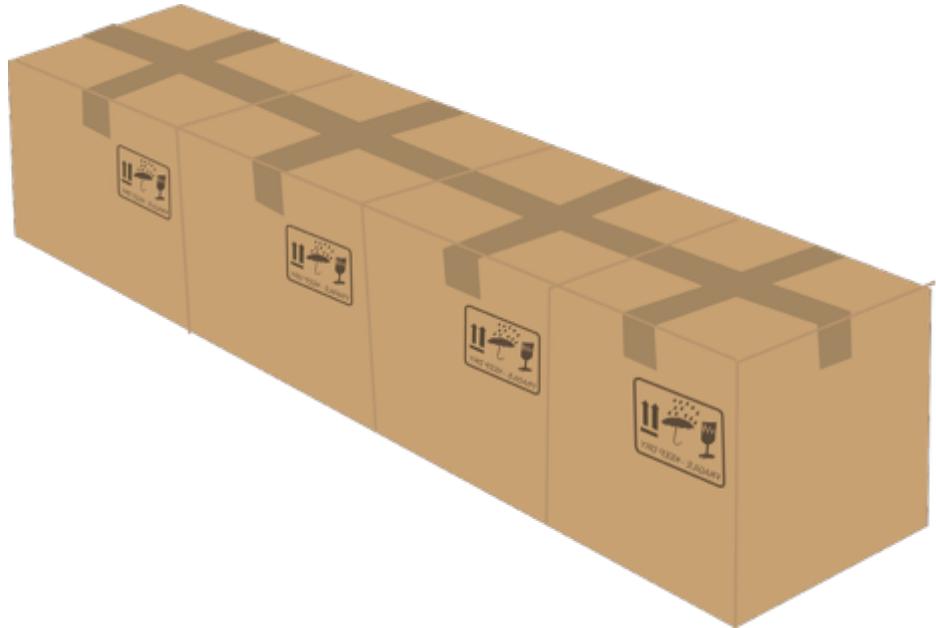
**Runtime if size>capacity/10:**

$$|S| \times n^{|S|^10}$$

# Bin packing, linear programming and rounding



# Bin packing, linear programming and rounding



**Less special special case**

**Large items,  
many sizes**

Idea: Rounding

Round sizes  
Reduce to special case

## How to round?

Capacity = 100 & Min size > 10:

Attempt:

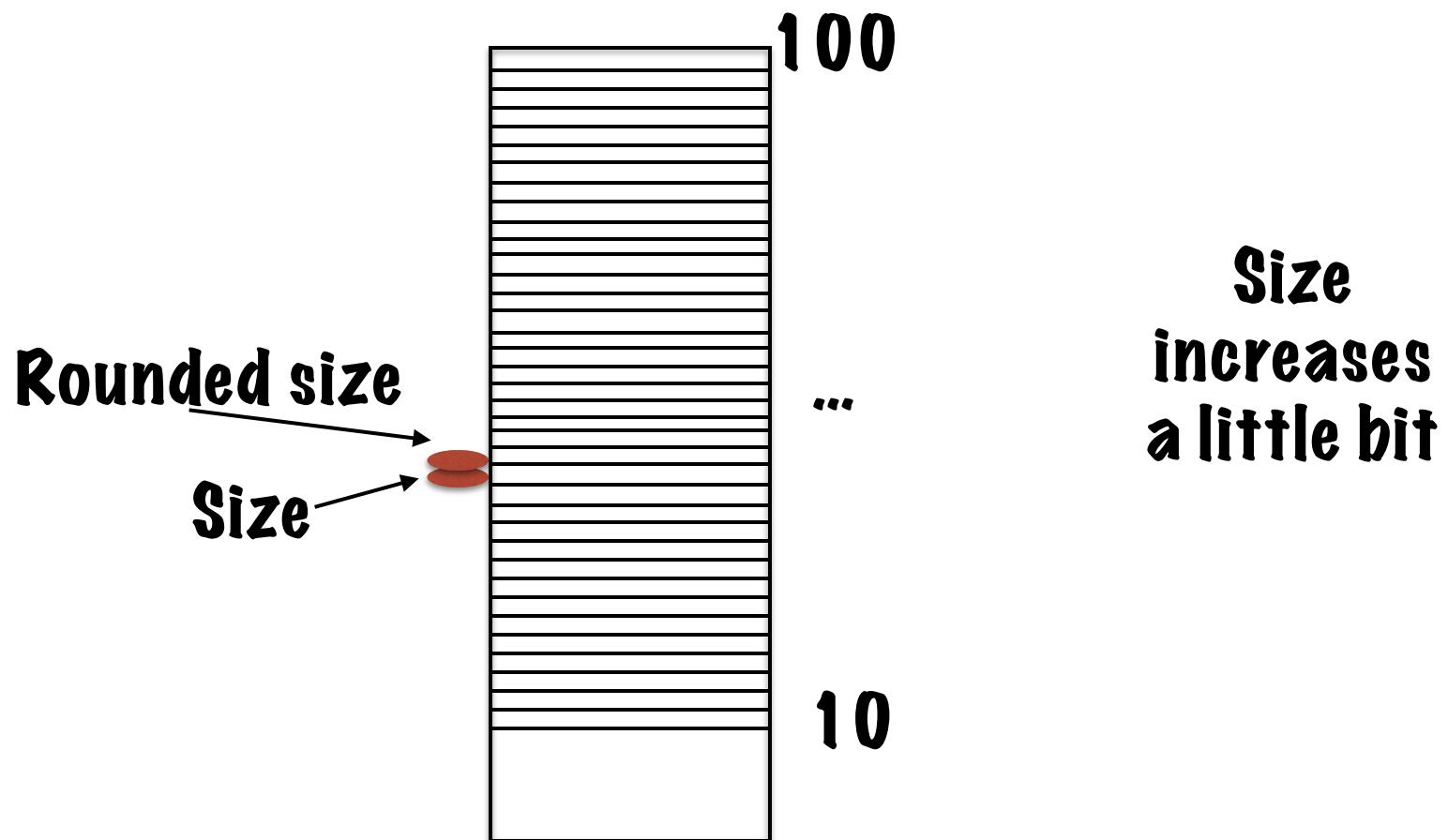
Round sizes **up** to nearest integer

Solve rounded problem

Observe:

It's a solution to original problem

# But how good is it?



How much does OPT change?

**Input:**

Capacity 100

50 items with size  $100 * (1/3)$

50 items with size  $100 * (2/3)$

**OPT=50**

**Rounded input:**

Capacity 100

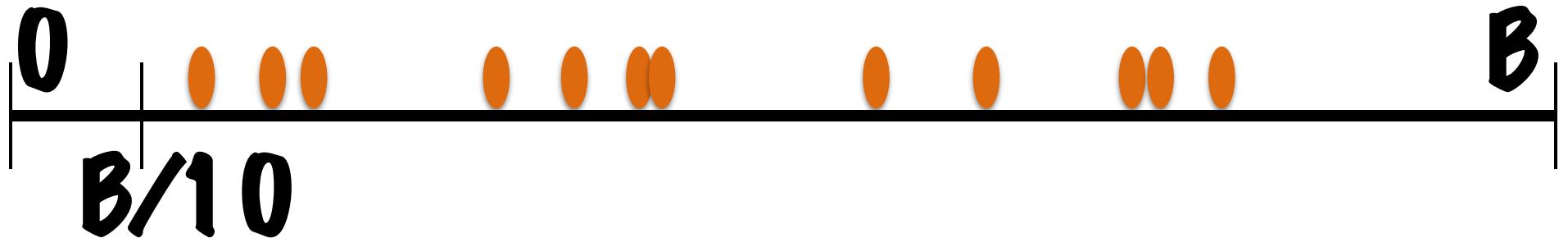
50 items with size 34

50 items with size 67

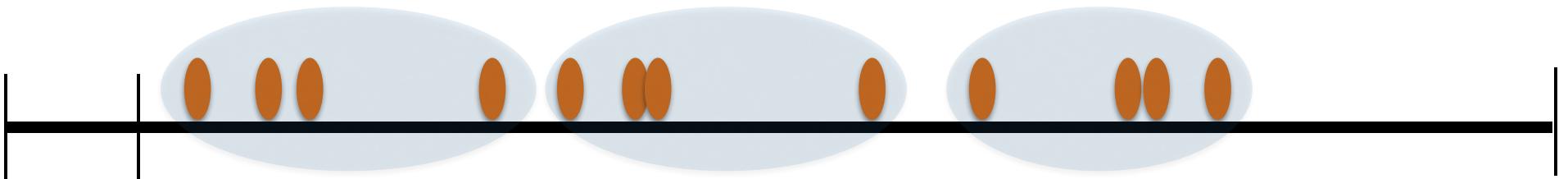
**OPT'=75**

This rounding fails

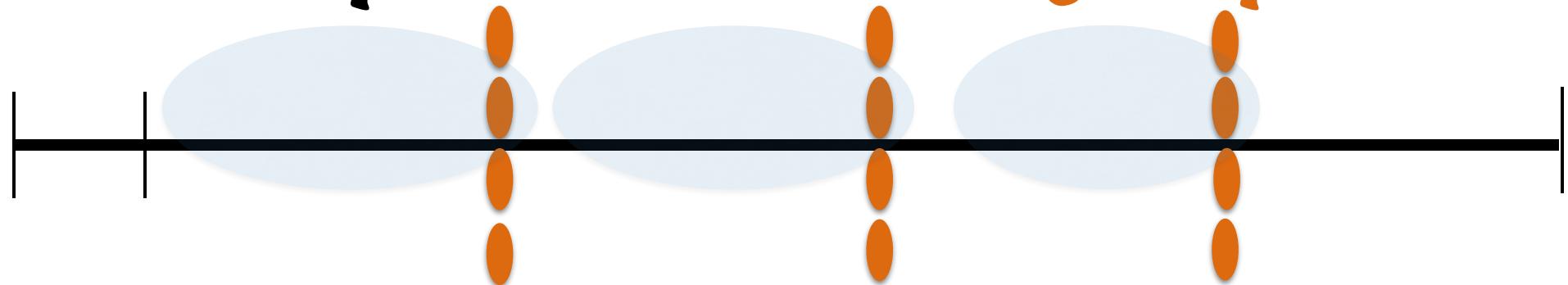
# **Adaptive rounding**



**Make groups of equal cardinality**



**Round up to max size in group**



# Algorithm - large items

Assume: sizes > capacity \*  $\epsilon$

Sort sizes

Make groups of cardinality  $n \times \epsilon^2$

Round up to max size in group

Solve rounded problem

Output corresponding packing

**Observe: output is a packing**

**Observe: all sizes are  $> \text{Capacity} \times \epsilon$**

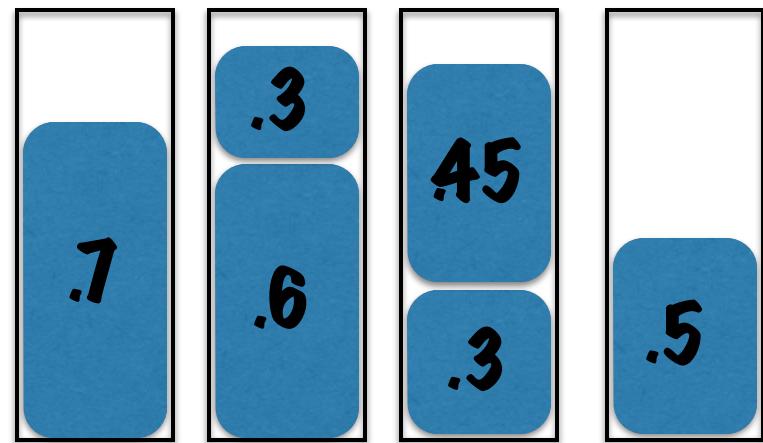
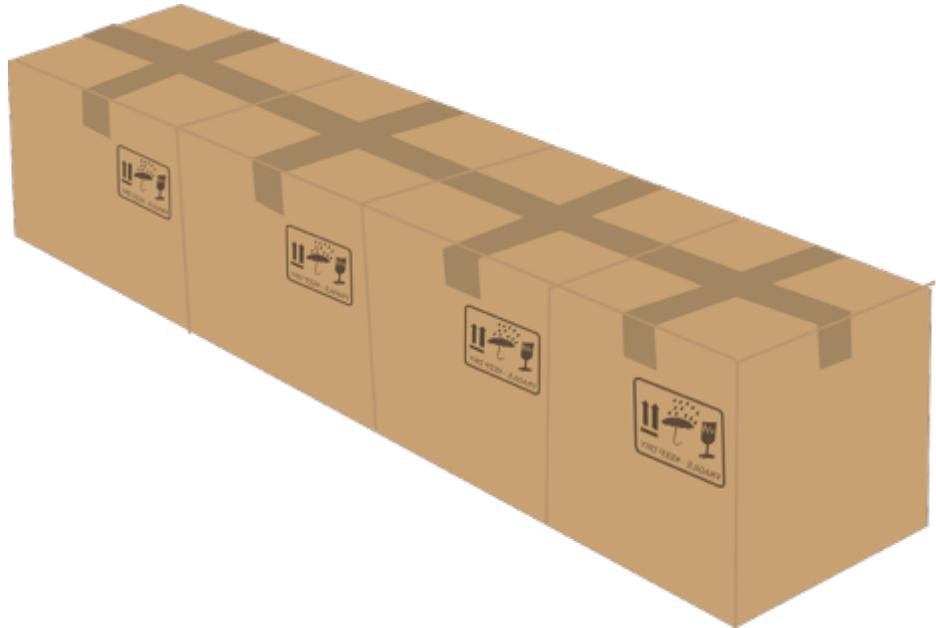
**Observe: #distinct sizes  $< 1/\epsilon^2$**

**Runtime : polynomial**

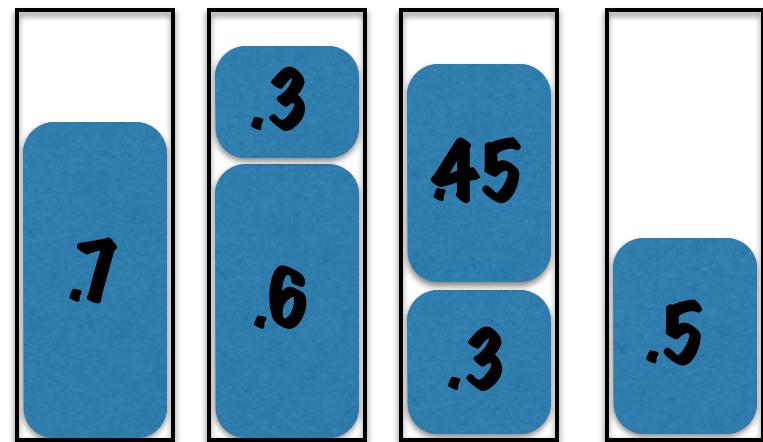
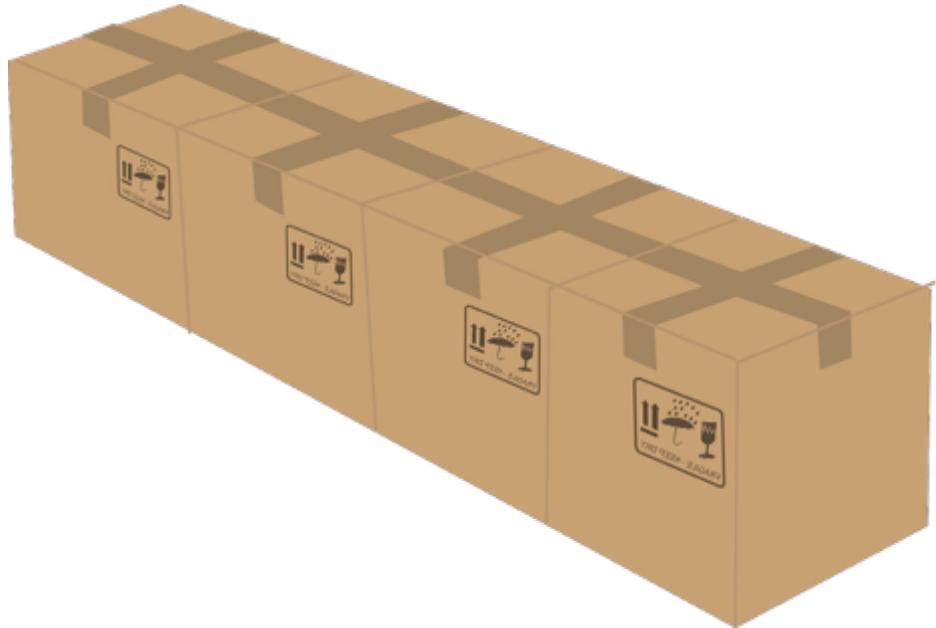
But how good is it?

$$\text{Value}(\text{Output}) < \text{OPT} * (1 + O(\epsilon))$$

# Bin packing, linear programming and rounding



# Bin packing, linear programming and rounding



# Algorithm - large items

Assume: sizes > capacity \*  $\epsilon$

Sort sizes

Make groups of cardinality  $n \times \epsilon^2$

Round up to max size in group

Solve rounded problem U

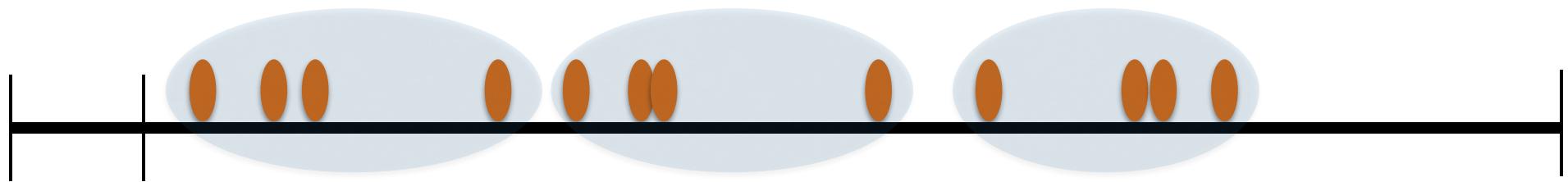
Output corresponding packing

**But how good is it?**

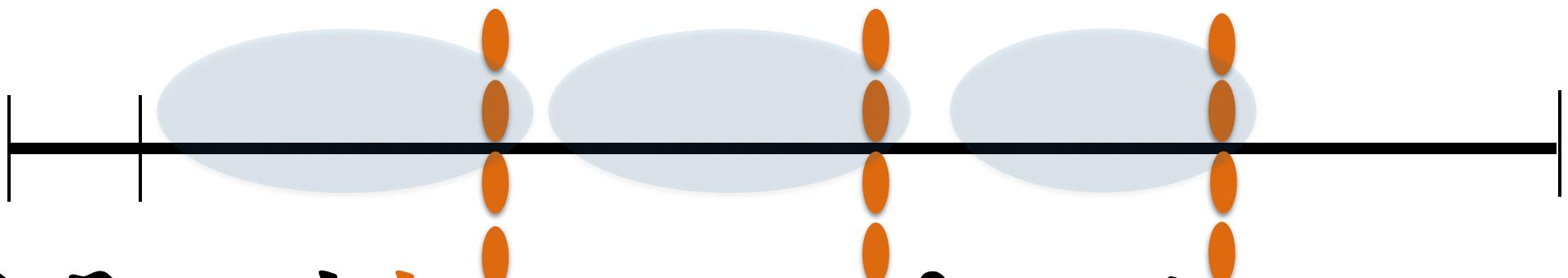
**Value(Output) = OPT(U)**

**Must relate OPT(U) to OPT(I)**

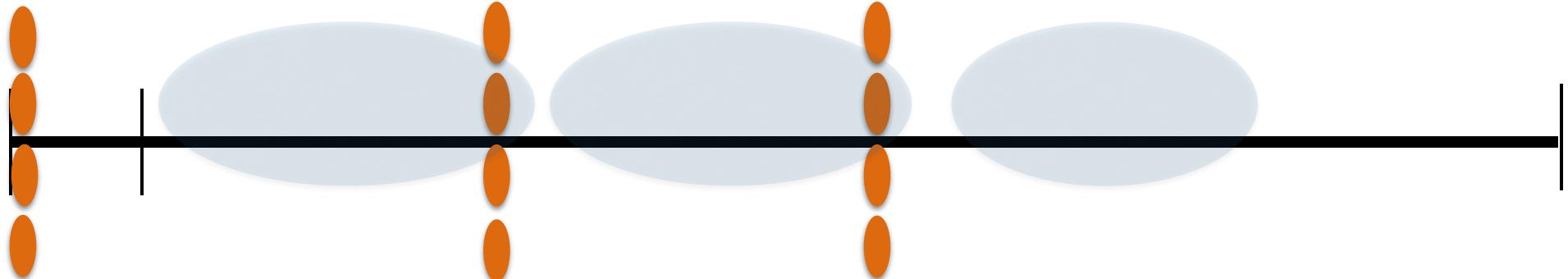
**I: Input**



**U: Round up: max of group**



**D: Round down: max of previous group**



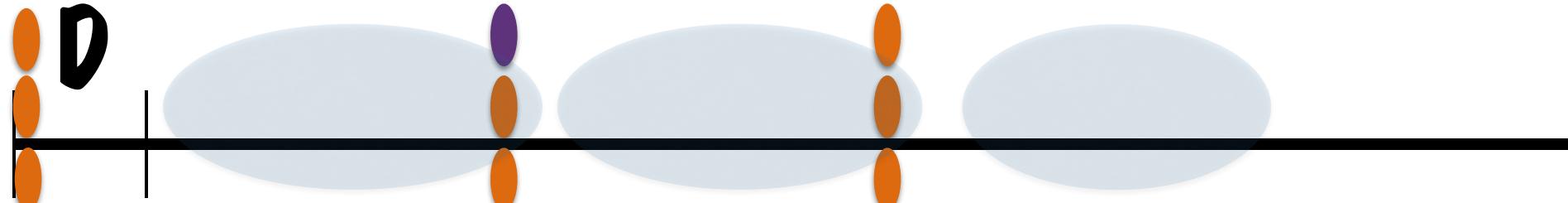
**U**



**up**

**down**

**D**

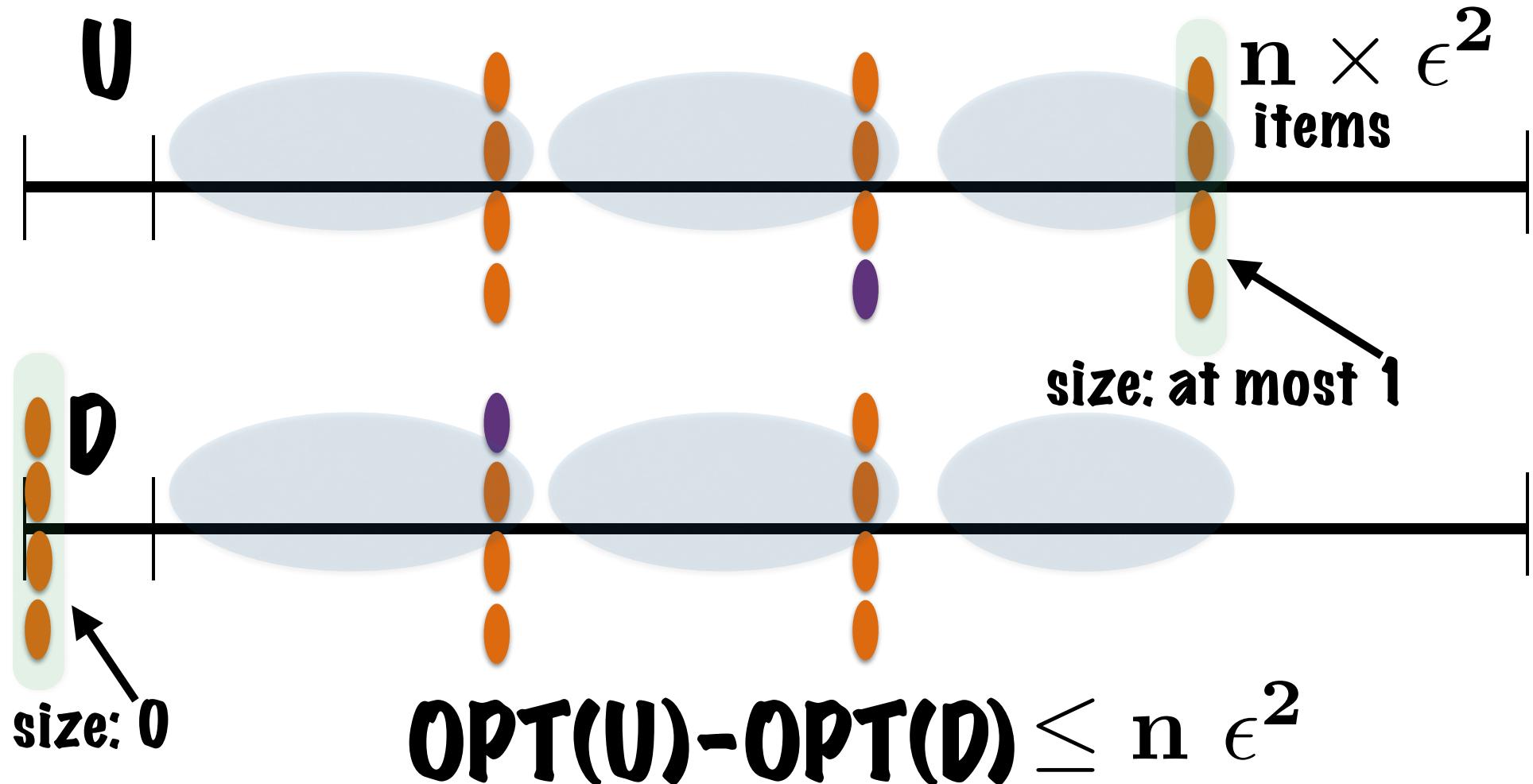


# Relating input to rounded input

Observe:  
Increasing sizes  
can only increase OPT

$$\text{OPT}(D) \leq \text{OPT}(I) \leq \text{OPT}(U)$$

# U and D are similar!



**Combine:**

$$\text{OPT}(D) \leq \text{OPT}(I) \leq \text{OPT}(U)$$

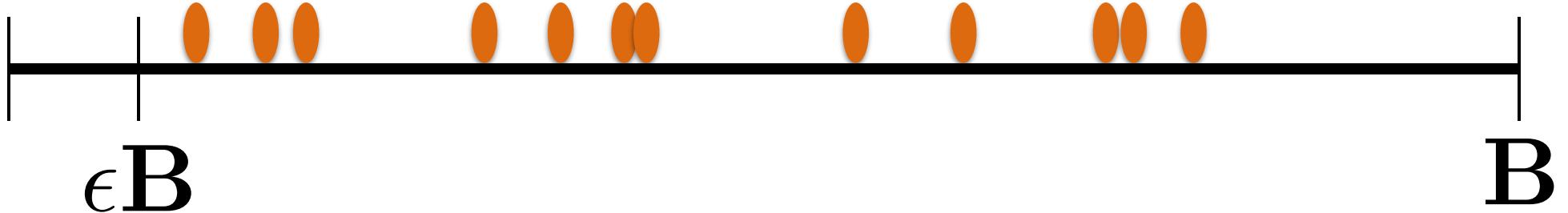
$$\text{OPT}(U) - \text{OPT}(D) \leq n \epsilon^2$$

---

$$\text{OPT}(U) \leq \text{OPT}(I) + n \epsilon^2$$

**Additive error**  $n \epsilon^2$

## Lower bound OPT



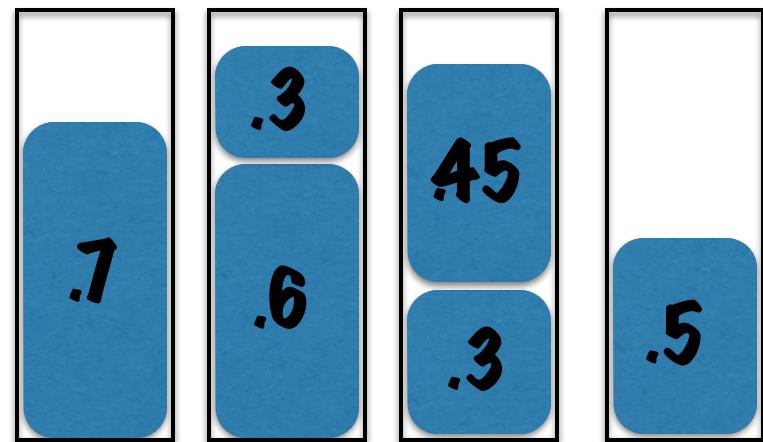
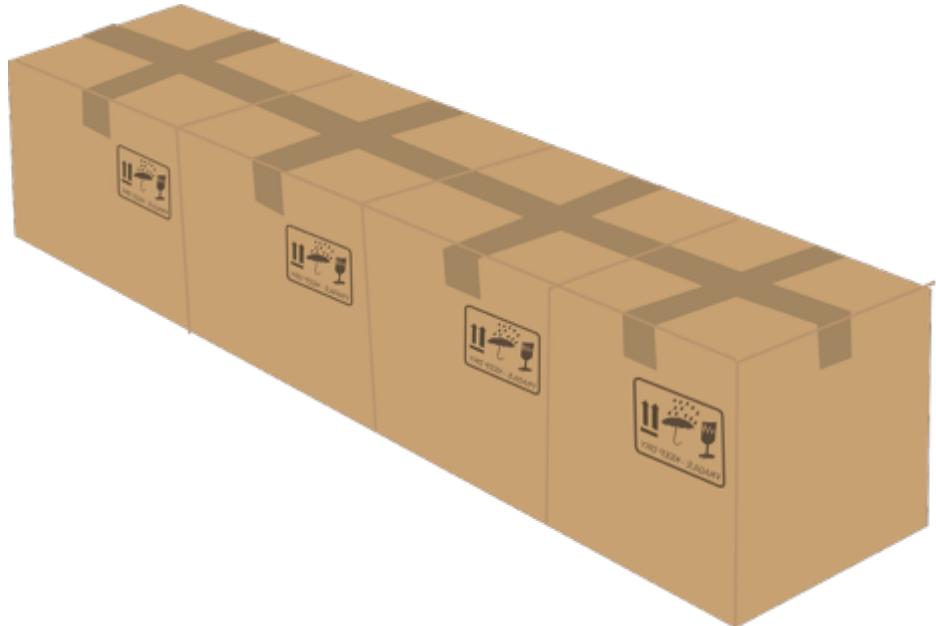
**n items**  
**max #items per bin:  $1/\epsilon$**   
—————  
**min #bins:  $\epsilon n$**

$$n\epsilon^2 \leq \epsilon \times (n\epsilon) \leq \epsilon \text{ OPT}$$

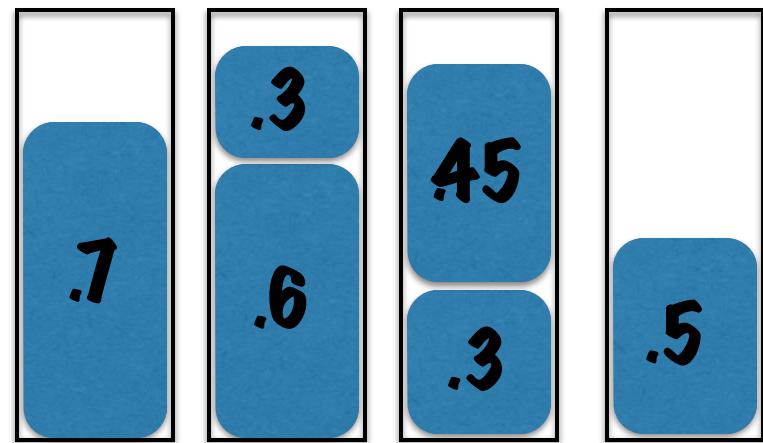
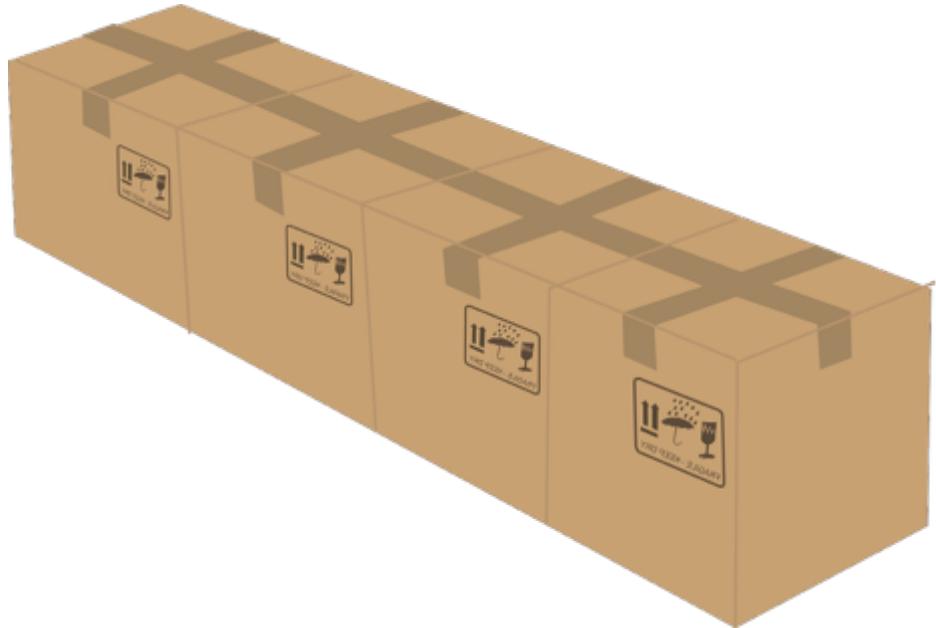
# Theorem

When all sizes are  $> \epsilon B$   
algorithm, in polynomial time  
gives packing s.t.  
 $\text{Value}(\text{Output}) < \text{OPT} * (1 + \epsilon)$

# Bin packing, linear programming and rounding



# Bin packing, linear programming and rounding



# General algorithm

Set aside: sizes < cap. \*  $\epsilon$  (small)

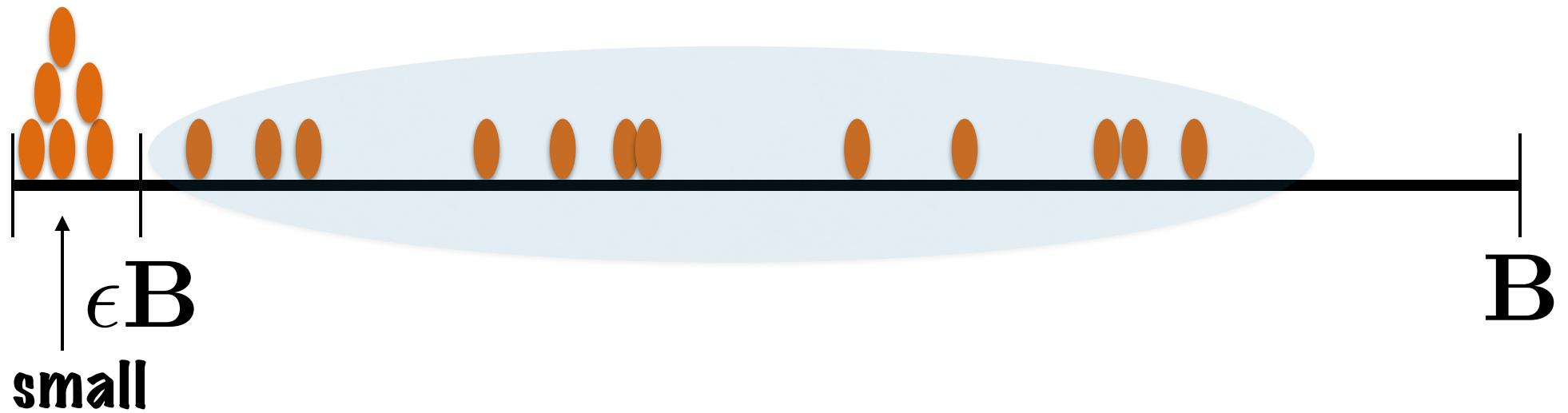
Sort remaining sizes

Make groups of cardinality  $n \times \epsilon^2$

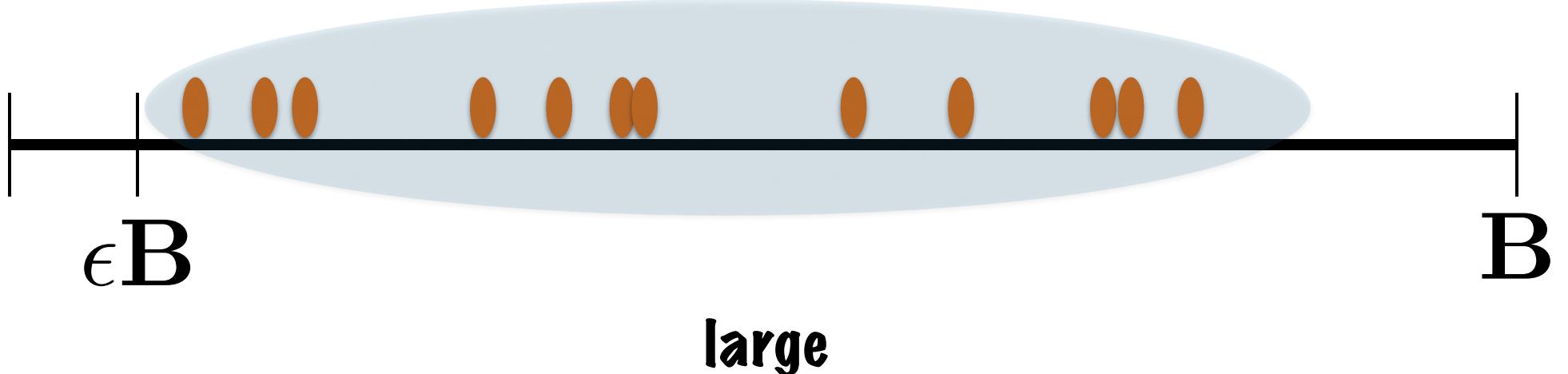
Round up to max size in group

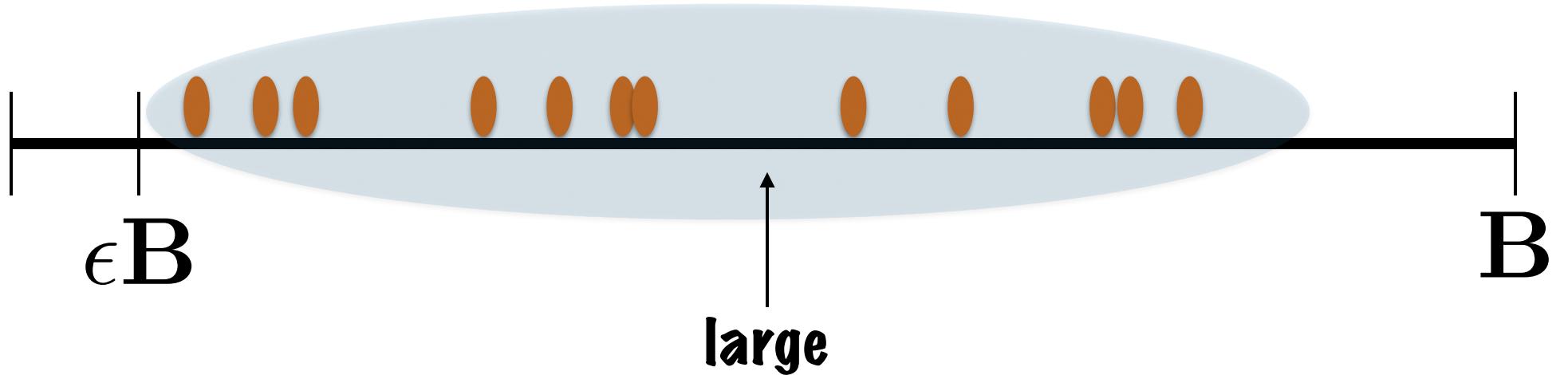
Solve rounded problem  $U$

Greedily add small sizes

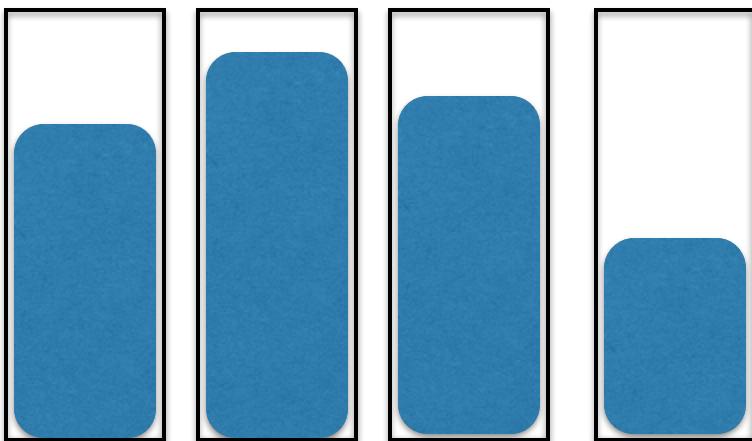


Set aside: sizes  $< B^\epsilon$  (small)

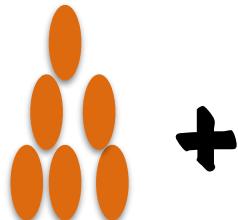




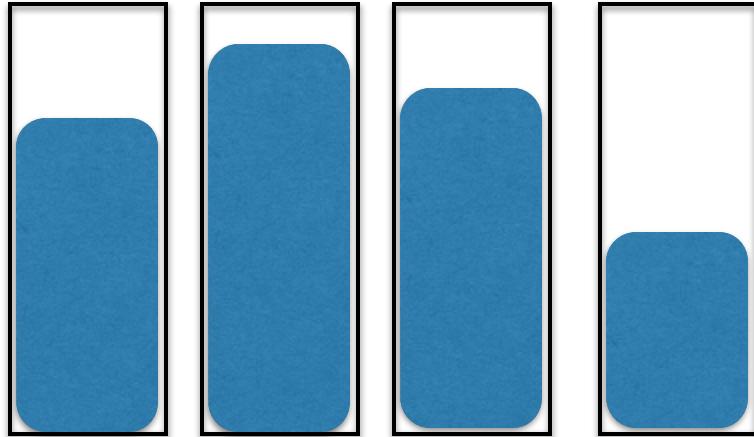
**Solve for remaining sizes**



**Packing of  
large items**

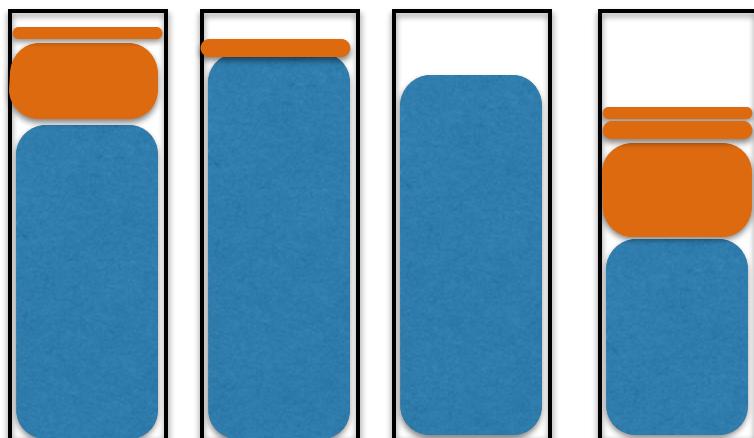


+  
small  
items



Packing of  
large items

Greedily add small sizes



# Analysis

Input  $I = S \cup L$

# Case 1

## No new bins opened by S: then

$$\begin{aligned}\text{Value}(\text{Output}) &= \\ \text{Value}(\text{packing of } L) &\leq (1 + \epsilon) \cdot \text{OPT}(L) \\ &\leq (1 + \epsilon) \cdot \text{OPT}(I)\end{aligned}$$

## Case 2

**Some new bin opened by S:  
then all bins except last  
are filled to B times**

$$\geq 1 - \epsilon$$

$$(1/B) \sum s_i \geq (\# \text{bins} - 1)(1 - \epsilon)$$
$$(1/B) \sum s_i \leq \text{OPT}$$

---

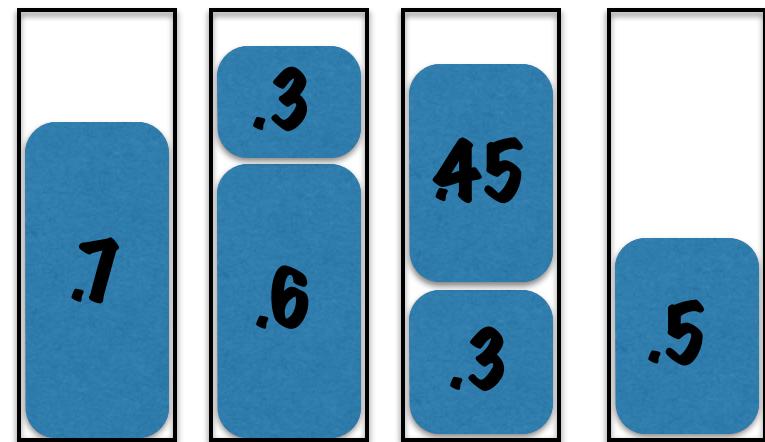
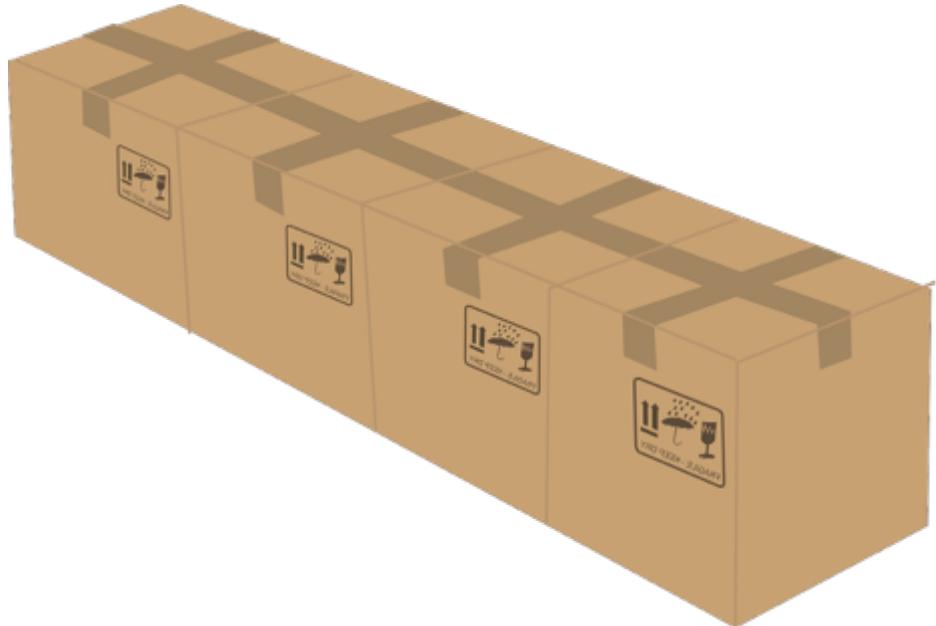
$$\text{Value(Output)} \leq \frac{1}{1-\epsilon} \text{OPT} + 1$$

# Theorem

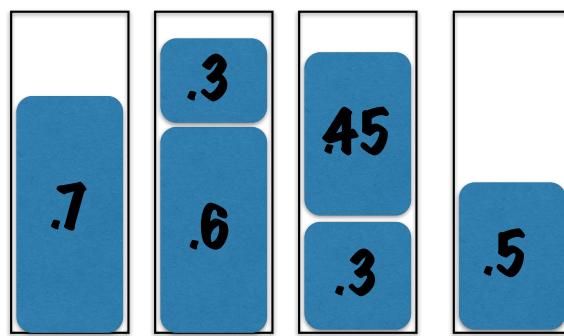
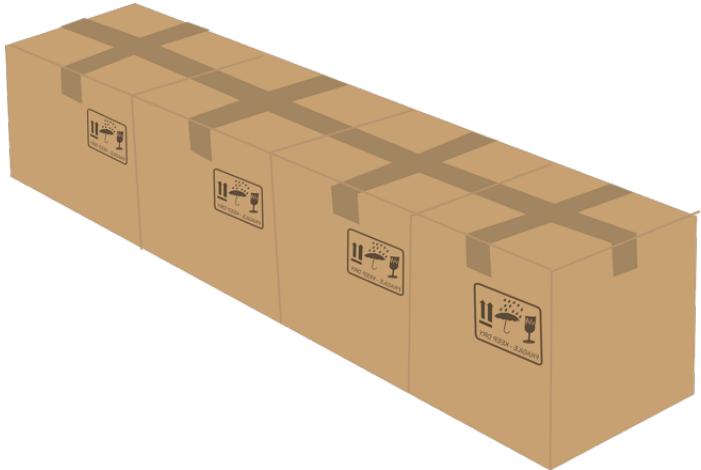
Algorithm, in polynomial time  
gives packing s.t.  
 $\text{Value}(\text{Output}) <$

$$\text{OPT}(1 + O(\epsilon)) + 1$$

# Bin packing, linear programming and rounding



# Bin packing, linear programming and rounding



# Techniques

## Adaptive rounding of input

# **Bin packing variants**

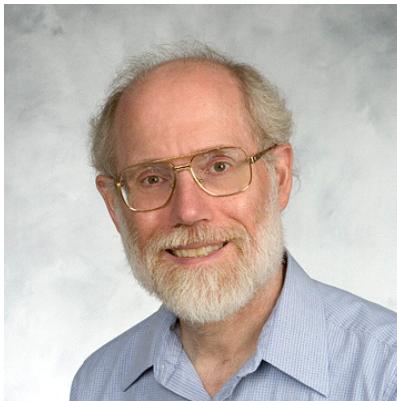
**Two-dimensional,  
three-dimensional**

**Online**

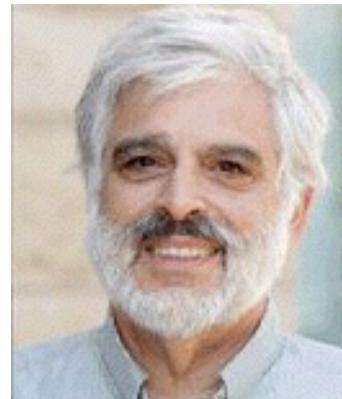


**Karp (1972)  
NP-complete**

**The pioneers**

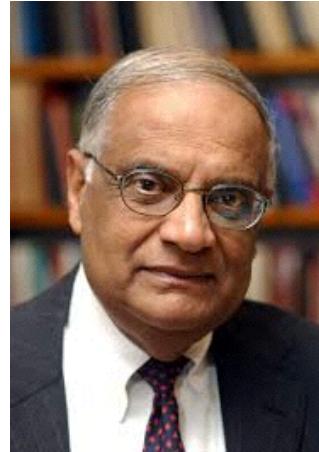


**David S. Johnson  
Constant factor  
approximation algorithms  
Jeffrey D. Ullman**





**Richard Karp**



**Narendra Karmarkar**

**Average case analysis**



**David S. Johnson**



**L. McGeoch**

...



**Wenceslas Fernandez de la Vega**



**George S. Lueker**

$$\text{OPT}(1 + O(\epsilon)) + 1$$



**Narendra Karmarkar**

**Richard Karp**

$$\text{OPT} + O(\log^2 \text{OPT})$$



**Asymptotic approximation schemes**

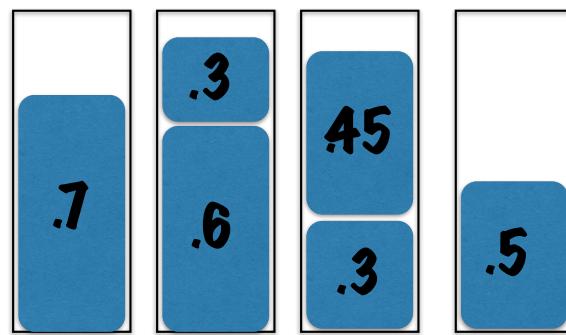
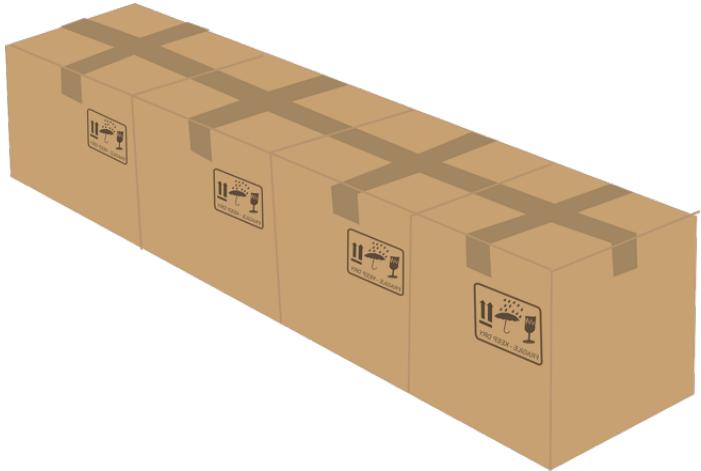


# Rebecca Hoberg Thomas Rothvoss

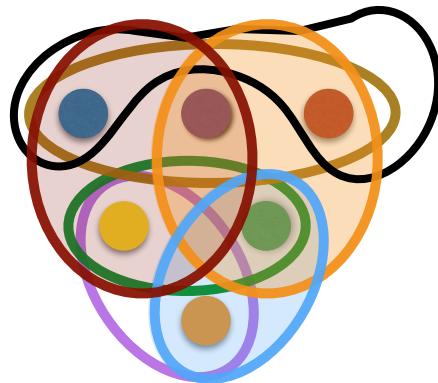
## OPT + O(log OPT)

**Asymptotic approximation schemes**

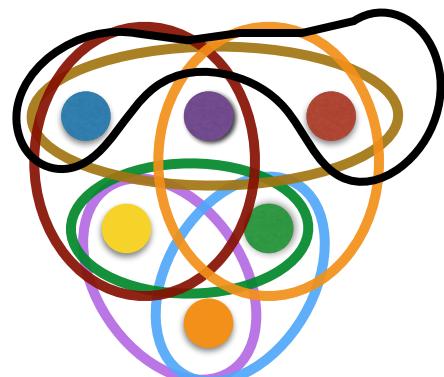
# Bin packing, linear programming and rounding



# Set cover, linear programming and randomized rounding



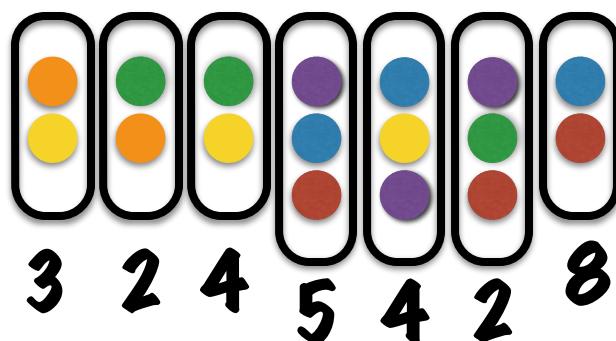
# The set cover problem



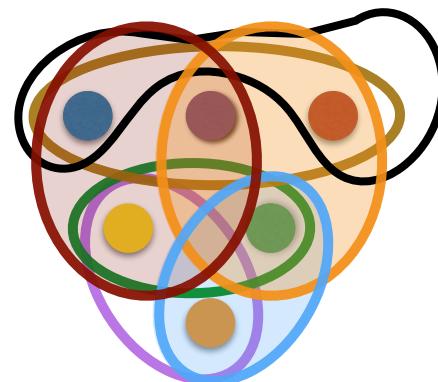
Elements



Subsets with costs



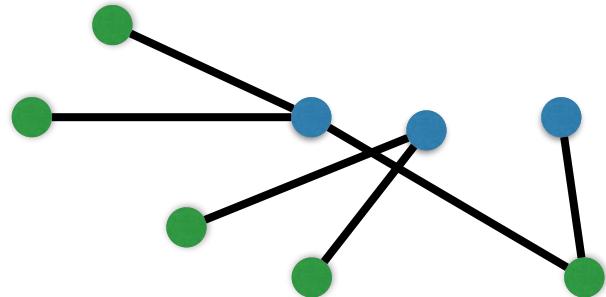
Choose subsets  
Cover elements  
at min cost



$$\text{Cost} = 4 + 2 + 2 = 8$$

Does this ring a bell?

# Vertex cover



Cover all edges  
with the fewest  
vertices

edges = elements  
vertices = sets

Integer program  
for  
Set cover

**Variable for subset S:**

$$x_S = 1$$

**iff S in cover**

**Constraint for element i:**

$$\sum_{S: e \in S} x_S \geq 1$$

**Objective:**

$$\min \sum_S c_S x_S$$

# Linear programming relaxation

$$\min \sum_S c_S x_S$$

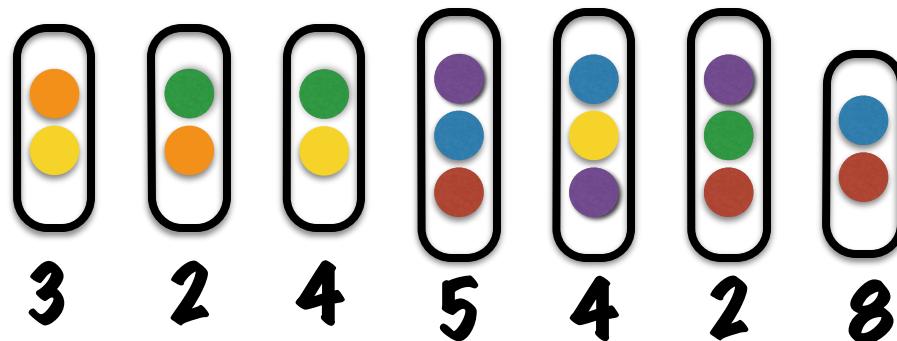
such that

$$\begin{cases} \sum_{S: e \in S} x_S \geq 1 & \forall e \\ 0 \leq x_S \leq 1 & \forall S \end{cases}$$

# Rounding

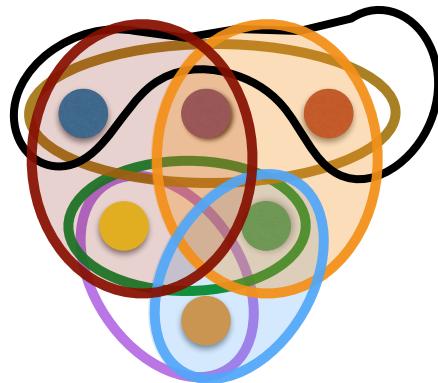
Like vertex cover:  
round to 1 iff  $x_u \geq 1/2$

Fails

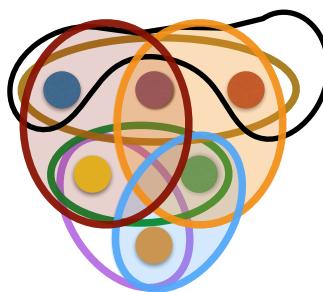


$$x_S : \frac{1}{3}, \frac{2}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{2}, \frac{1}{3}$$

# Set cover, linear programming and randomized rounding



# Set cover, linear programming and randomized rounding



## Linear programming relaxation

$$\min \sum_S c_S x_S$$

such that

$$\begin{cases} \sum_{S:e \in S} x_S \geq 1 & \forall e \\ 0 \leq x_S \leq 1 & \forall S \end{cases}$$

How do we round the LP solution?

# Randomized Rounding: An algorithm

$x_i = .9$  : should probably go to 1

$x_i = .1$  : should probably go to 0

Cannot fix a threshold

Idea: randomized rounding

$x_i = .8 \implies$  round to 1  
w.p. 80%

# New rounding algorithm

For each set  $S$   
with probability  $x_S$   
put  $S$  in the cover

# **Analysis**

**Is it efficient?  
Is the output a cover?  
How good is it?**

# **Analysis**

**Is it efficient?  
Yes**

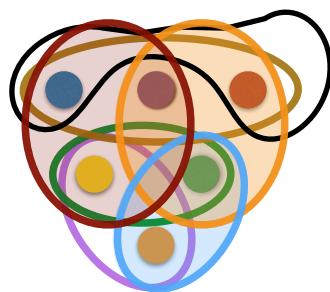
# **Analysis**

**Is the output a cover?  
Maybe, maybe not**

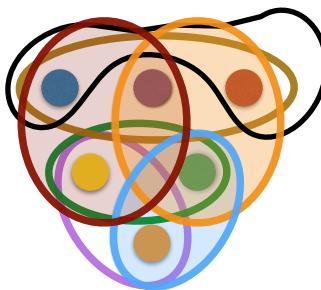
# Analysis

**How good is it?**  
*It depends...*

# Set cover, linear programming and randomized rounding



# Set cover, linear programming and randomized rounding



**How good is it?**  
**It depends...**

$$\text{Value}(\text{Output}) = \sum_{S \text{ in cover}} c_S$$

**How good is it *on average*?**

$$E[\sum_S 1(S \text{ in cover})c_S] = ?$$

# Linearity of expectation

$$E[A + B] = E[A] + E[B]$$

$$\begin{aligned} E\left[\sum_S 1(S \text{ in cover})c_S\right] &= \\ \sum_S E[1(S \text{ in cover})c_S] \end{aligned}$$

$$E[\lambda X] = \lambda E[X]$$

$$\begin{aligned} \sum_S E[1(S \text{ in cover}) c_S] &= \\ \sum_S E[1(S \text{ in cover})] c_S \end{aligned}$$

$$\begin{aligned} E[1(S \text{ in } \text{cover})] &= \\ \Pr[S \text{ in } \text{cover})] \end{aligned}$$

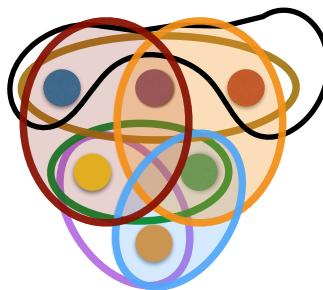
$$\Pr[\mathbf{S} \text{ in cover}] = \mathbf{x_S}$$

**Together**

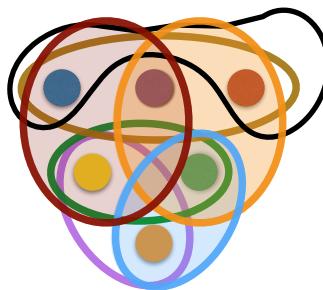
$$E[Value(Output)] = \sum_S x_S c_S$$

**Value of the linear program!**

# Set cover, linear programming and randomized rounding



# Set cover, linear programming and randomized rounding



**Is the output a cover?  
Maybe, maybe not**

**Number of elements covered:**

$$\sum_e 1(e \text{ covered})$$

**On average:**

$$\sum_e \Pr[e \text{ covered}]$$

**Consider an element  $e$ .  
With what probability  
is it covered by output?**

$\Pr[\mathbf{e \text{ covered}}] =$   
 $\Pr[\text{there exists } S \text{ in output:}$   
 $\mathbf{e \in S}]$

$\Pr[\text{there exists } S \text{ in output:}$   
 $e \in S] =$   
 $1 - \Pr[\text{for all } S \text{ containing } e:$   
 $S \text{ not in output}]$

# Independence

If independence:

$$\Pr[A \text{ and } B] = \Pr[A] \times \Pr[B]$$

$\Pr[\text{for all } S \text{ containing } e:$

$S \text{ not in output}] =$

$$\prod_{S:e \in S} \Pr[S \text{ not in output}]$$

$$\Pr[S \text{ not in output}] = 1 - x_S$$

# Together

$$\Pr[\text{e covered}] = 1 - \prod_{S:e \in S} (1 - x_S)$$

# Algebra

$$X = e^{\ln X}$$

$$\ln(XY) = \ln X + \ln Y$$

$$\prod_{S:e \in S} (1 - x_S) = e^{\sum_{S:e \in S} \ln(1 - x_S)}$$

# Algebra

$$\ln(1 - X) \leq -X$$

$$e^{\sum_{S:e \in S} \ln(1-x_S)} \leq e^{-\sum_{S:e \in S} x_S}$$

# Use LP constraint

$$\sum_{S:e \in S} x_S \geq 1$$

$$e^{-\sum_{S:e \in S} x_S} \leq e^{-1}$$

# Combining

$$\Pr[e \text{ covered}] \geq 1 - 1/e$$

**Average number of  
elements covered:**

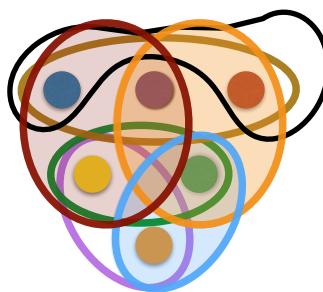
$$\#(\text{elements})(1 - 1/e)$$

$$1 - 1/e = 0.63\dots$$

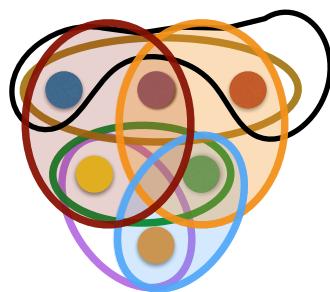
# Recap

**Randomized rounding gives  
collection of sets  
with average cost  
at most OPT  
and covering on average  
63% of the elements.**

# Set cover, linear programming and randomized rounding



# Set cover, linear programming and randomized rounding



# Getting a set cover

Idea: repeat!

# Randomized rounding algorithm

$n = \# \text{elements}$

Repeat  $\ln(n) + 3$  times

For each  $S$

Put  $S$  in cover w.pr.  $x(S)$   
(if not there already)

Note:  $e^3 = 20.0\dots$

# Cost

In expectation:  
at most  
 $(\ln(n) + 3)OPT$

# Correctness

$$\Pr[\text{cover}] = 1 - \Pr[\text{not cover}]$$

$$\begin{aligned}\Pr[\text{not cover}] &= \\ \Pr[\exists \text{ element not covered}] &\leq \\ \sum_e \Pr[e \text{ not covered}] &\end{aligned}$$

**For one element e  
and for one iteration**

$$\Pr[e \text{ not covered}] < 1/e$$

**For one element e  
and for all iterations together**

$$\Pr[e \text{ not covered}] < (1/e)^{\ln(n)+3} = \frac{1}{e^3 n}$$

$$\sum_e \Pr[e \text{ not covered}] < n \frac{1}{e^3 n} = \frac{1}{e^3} < 0.05$$

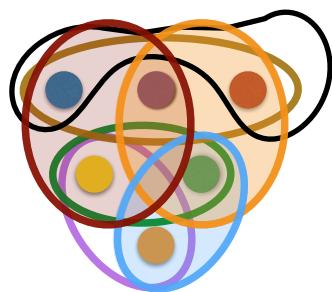
**So:**

$$\Pr[\text{cover}] > 0.95$$

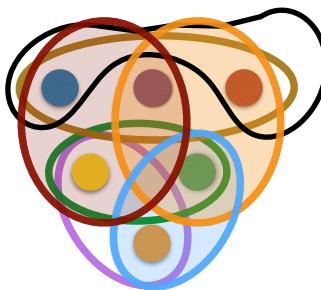
# **Result**

**Iterated randomized rounding gives  
collection of sets  
that is a set cover with  
probability 95% and  
with average cost  
at most  $(\ln(n)+3)$  OPT.**

# Set cover, linear programming and randomized rounding



# Set cover, linear programming and randomized rounding



# Result so far

Iterated randomized rounding gives collection of sets

that is a set cover with probability 95%  
and with average cost at most  $(\ln(n)+3)$  OPT.

Not guaranteed!

Not guaranteed!

**Q: What if you want the output  
to always be a set cover?**

**Desired result:**  
algorithm that gives  
collection of sets  
that **is** a set cover  
and with **average**  
cost at most  $O(\ln(n))$  OPT.

**Guaranteed!**



**Obvious solution:**  
Replace  
“repeat  $\ln(n)+3$  times”  
by  
“repeat until you have a set cover.”

# Equivalent algorithms

Repeat  
For each  $S$   
Put  $S$  in cover w.pr.  $x(S)$   
(if not there yet)  
Until you have a set cover

Repeat  
Choose  $S$  w.pr.  $x_S / \sum_{S'} x(S')$   
Put  $S$  in cover  
(if not there yet)  
Until you have a set cover

# Sample-and-iterate algorithm

**Repeat**

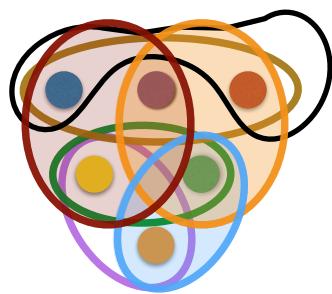
**Choose  $S$  w.pr.  $x_S / \sum_{S'} x(S')$**

**Put  $S$  in cover**

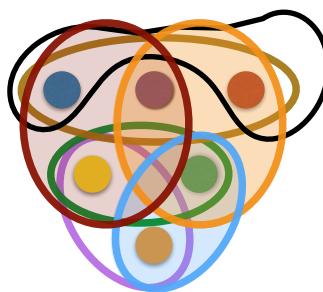
**(if not there yet)**

**Until you have a set cover**

# Set cover, linear programming and randomized rounding



# Set cover, linear programming and randomized rounding



# Sample-and-iterate algorithm

**Repeat**

**Choose  $S$  w.pr.  $x_S / \sum_{S'} x(S')$**

**Put  $S$  in cover**

**(if not there yet)**

**Until you have a set cover**

**Repeat**

**Choose  $S$  w.pr.  $x_S / \sum_{S'} x(S')$**

**Put  $S$  in cover**

**(if not there yet)**

**Until you have a set cover**

**Analysis**

### **1. Expected cost of output**

$T = \text{\#iterations (stopping time)}$

$C_t = \text{cost of set chosen at iteration } t$

**Cost of output:**  $\sum_{t=1}^T C_t$

**Repeat**

**Choose  $S$  w.pr.  $x_S / \sum_{S'} x(S')$**

**Put  $S$  in cover**

**(if not there yet)**

**Until you have a set cover**

**Cost of output:**  $\sum_{t=1}^T C_t$

**Expected cost of output:**  $E[\sum_{t=1}^T C_t]$

**NB:** cannot exchange  $E[ ]$  and summation here!

# **Wald's equation**

**T stopping time**

**X<sub>t</sub> random variable**

**If X<sub>t</sub> bounded from above:**

$$\text{then } E\left[\sum_{t \leq T} X_t\right] \leq \mu E[T]$$

**NB: can now exchange E[] and summation!**

**Expected cost of set chosen at iteration t:**

$$E[C_t] = \sum_S \Pr[S \text{ chosen}] c_S = \frac{\sum_S c_S x_S}{\sum_S x_S} = \mu$$

**Expected cost of output:**  $E[T]\mu = E[T] \frac{\sum_S c_S x_S}{\sum_S x_S}$

**Next problem: What is  $E[T]$ ?**

Repeat

Choose  $S$  w.pr.  $x_S / \sum_{S'} x(S')$

Put  $S$  in cover

(if not there yet)

Until you have a set cover

More  
notations

## 2. Expected number of iterations

$n_t = \#\text{elts not yet covered after } t \text{ iterations}$

$$n_0 = n, n_T = 0, n_{T-1} \geq 1$$

# Wald's equation for dependent decrements

Consider a random decreasing sequence

$$n_0, n_1, \dots, n_T,$$

where  $T$  is a stopping time. If

$$E[n_t - n_{t+1} | n_t] \geq f(n_t),$$

where  $f$  is an increasing function, then

$$E[T] \leq 1 + E\left[\int_{n_{T-1}}^{n_0} \frac{1}{f(z)} dz\right].$$

To bound  $E[T]$ , analyze  
#elts not yet covered after  $t$  iterations

**Analyze**  $n_t$

**Analyze**  $E[n_t - n_{t+1} | n_t]$ : elts covered in next iteration

**Fix an element**  $e$  among the  $n_t$

$\Pr[e \text{ covered in next iteration}] =$

$\Pr[S \text{ chosen contains } e] =$

$$\frac{\sum_{S:e \in S} x_S}{\sum_{S'} x_{S'}} \geq \\ 1 / \sum_{S'} x_{S'}$$

**Sum over**  $e$

$$E[n_t - n_{t+1} | n_t] \geq f(n_t)$$

$$E[n_t - n_{t+1} | n_t] \geq n_t / \sum_{S'} x_{S'}$$

## Wald's equation for dependent decrements

Consider a random decreasing sequence

$$n_0, n_1, \dots, n_T,$$

where T is a stopping time. If

$$E[n_t - n_{t+1} | n_t] \geq f(n_t),$$

where f is an increasing function, then

$$E[T] \leq 1 + E\left[\int_{n_{T-1}}^{n_0} \frac{1}{f(z)} dz\right].$$

## Application:

Stopping time  $E[T] \leq 1 + \int_1^n \frac{\sum_S x_S}{z} dz = 1 + \sum_S x_S \ln(n)$

# Together

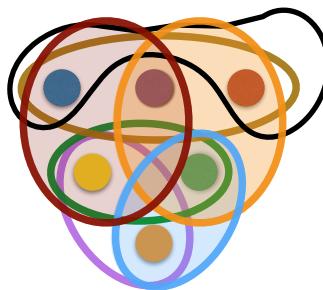
## Expected cost of output:

$$(1 + (\sum_S x_S) \ln(n)) \frac{\sum_S c_S x_S}{\sum_S x_S} \leq (1 + \ln(n)) \text{ OPT}$$

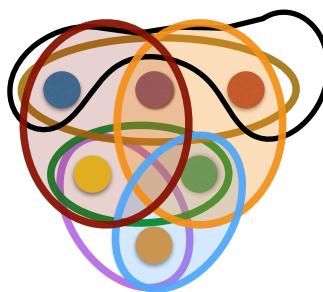
### Result:

the algorithm gives  
a collection of sets that is a set cover  
with average cost at most  $(1 + \ln(n)) \text{ OPT}$ .

# Set cover, linear programming and randomized rounding



# Set cover, linear programming and randomized rounding



# Result

The sample-and-iterate algorithm gives a collection of sets that is a set cover with average cost at most  $(1 + \ln(n)) \text{ OPT}$ .

# **A more efficient algorithm**

**Linear programming takes  
polynomial time  
but  
is often slower than  
combinatorial algorithms**

# **Greedy**

**Repeat**

**Choose  $S$  maximizing  $\#(\text{new elts covered})/c_S$**

**Put  $S$  in cover**

**Until you have a set cover**

**Result:**  
Greedy also gives  
a collection of sets  
that **is** a set cover  
and with  
**cost at most  $(1 + \ln(n)) \text{ OPT.}$**

Can we do better?

No:

It is NP-hard to obtain (in polynomial time) a  
better-than- $\ln(n)$  approximation  
for set cover



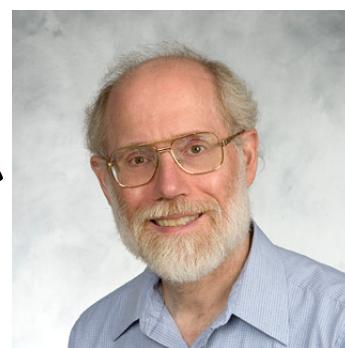
**Uri  
Feige**



**Vašek  
Chvátal**



**László  
Lovász**



**David  
Johnson**

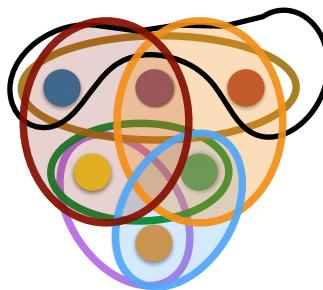


**Neal  
Young**

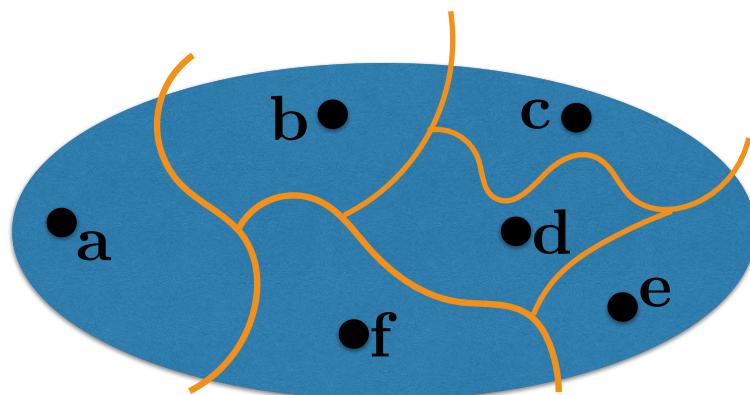
# What have we learned?

- Famous problem: set cover
- Concept: Randomization
- Algorithmic technique: Randomized rounding
- Analysis tool: Linearity of expectation
- Laying out an analysis: slow & steady, orderly - like hiking up a mountain

# Set cover, linear programming and randomized rounding



# Multiway cut, linear programming and randomized rounding



Let  $k$  be a fixed integer.

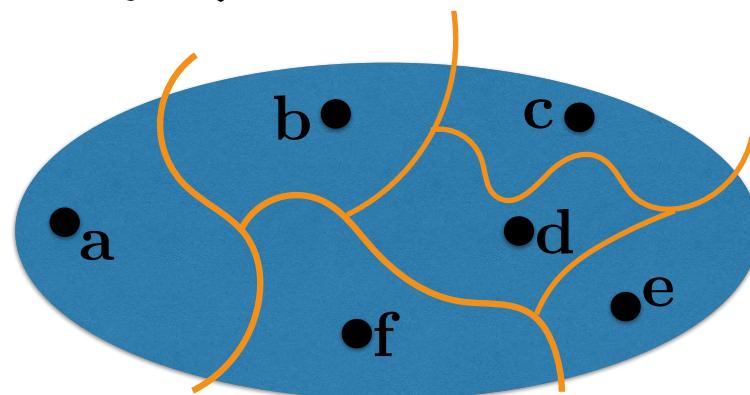
**Given:** graph  $G$  with edge weights,  
 $k$  vertices called "terminals"

**Find:** subset  $F$  of edges such that

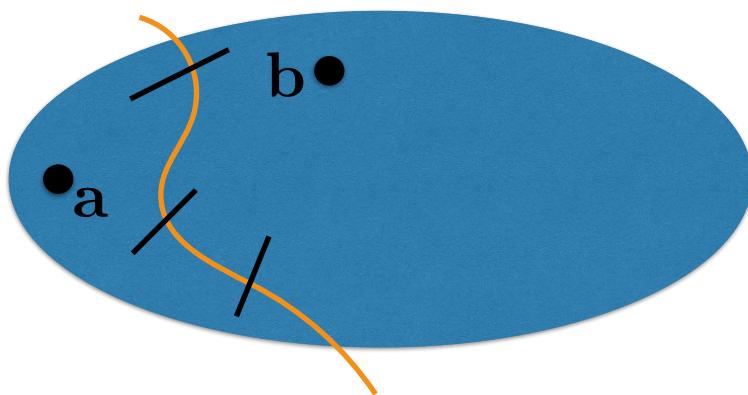
Removing  $F$  disconnects the  
terminals from one another

**Goal:** minimize weight of  $F$

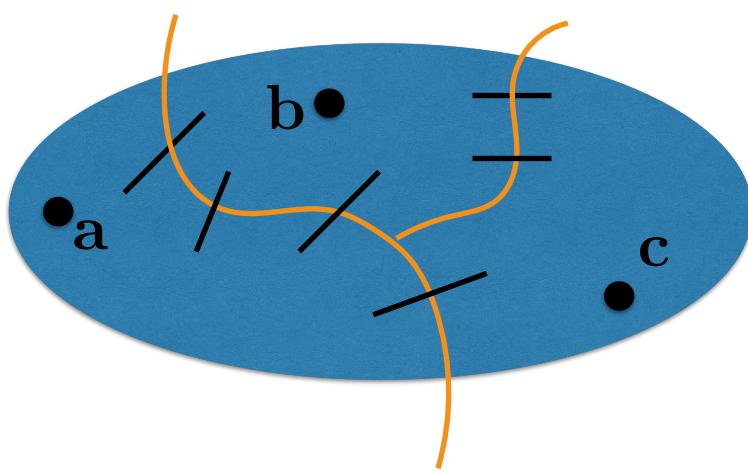
## Problem



**$k=2$**   
**min cut  
in P**



**$k=3$**   
**NP-hard**



# Simple algorithm for k=3

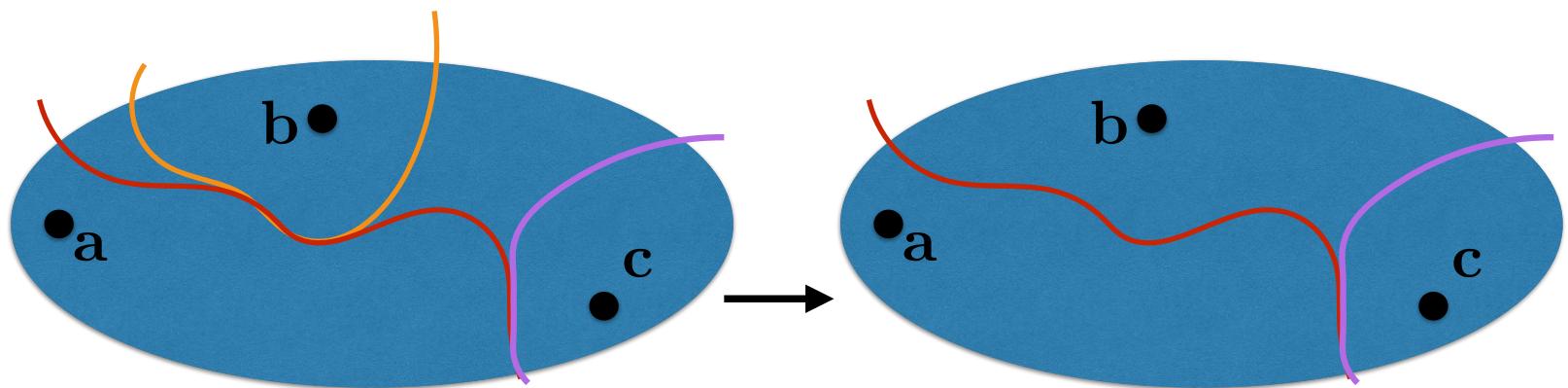
Terminals: a,b,c

Output the two smallest of

{ $\text{Mincut}(a, \{b, c\})$ ,

$\text{Mincut}(b, \{c, a\})$ ,

$\text{Mincut}(c, \{a, b\})$ }



## Analysis

- It takes polynomial time...
- It's a correct multiway cut:  
a,b,c are separated from one another
- But how good is it?

## Cost of output

Output is at most

$$(2/3) (\text{Mincut}(a,bc) + \text{Mincut}(b,ac) + \text{Mincut}(c,ab))$$

- OPT is at least  $\text{Mincut}(a,bc)$
- OPT is at least  $\text{Mincut}(b,ac)$
- OPT is at least  $\text{Mincut}(c,ab)$

So OPT is at least

$$(\text{Mincut}(a,bc) + \text{Mincut}(b,ac) + \text{Mincut}(c,ab))/3$$

Alg is a 2 approximation

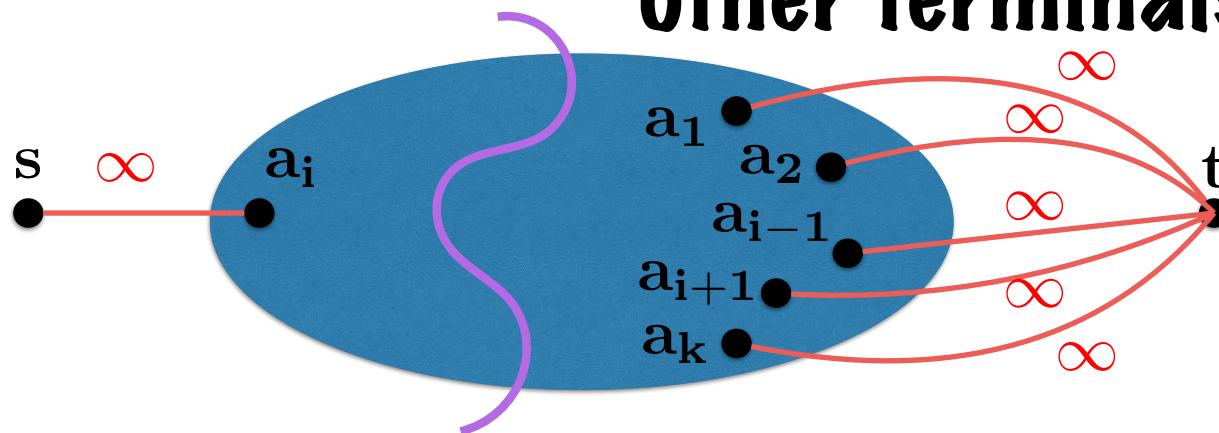
**Extension: algorithm for k**

**Terminals:  $a_1, a_2, \dots, a_k$**

**Output the  $k-1$  smallest of**

**{ $\text{Mincut}(a_i, \{a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_k\})$ :  $i=1, 2, \dots, k$ }**

**To compute the min cut separation of  $a_i$  from the other terminals:**



**$\text{Mincut}(s, t)$**

# **Analysis of output**

**The  $k-1$  smallest cuts cost  
less than a random choice of  $k-1$ , so**

$$\text{Cost}(\text{Output}) \leq \frac{k-1}{k} \sum_i \text{Mincut}(a_i, \{a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_k\})$$

# Analysis of OPT

Let  $F(i) = \{\text{edges of OPT that separate } a_i \text{ from some } a_j\}$

Consider  $e$  in OPT

$e$  separates some  $a_i$  from some  $a_j$   
 $e$  belongs to  $F(i)$  and to  $F(j)$

so

$$\sum_i \text{Cost}(F_i) = 2\text{OPT}$$

$F(i) = \{ \text{edges of OPT that separate } a_i \text{ from some } a_j \}$

$F(i)$  separates  $a_i$   
from all other terminals  
so

$\text{Cost}(F_i) \geq$   
 $\text{Mincut}(a_i, \{a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_k\})$

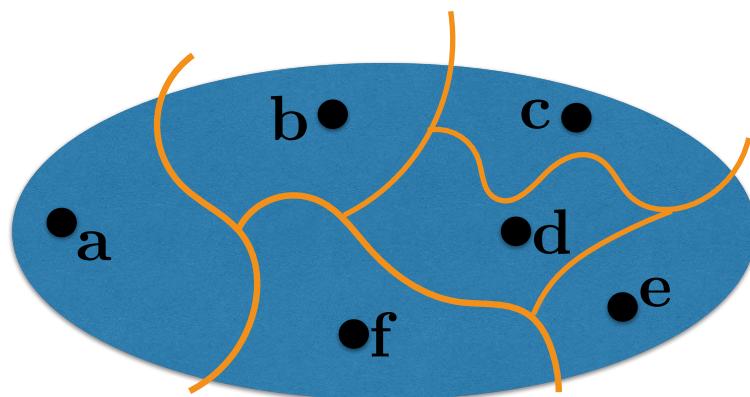
# Together

$$\begin{aligned} & \text{Cost(Output)} \\ & \leq \frac{k-1}{k} \sum_i \text{Cost}(F_i) \\ & \leq 2\left(1 - \frac{1}{k}\right) \text{OPT} \end{aligned}$$

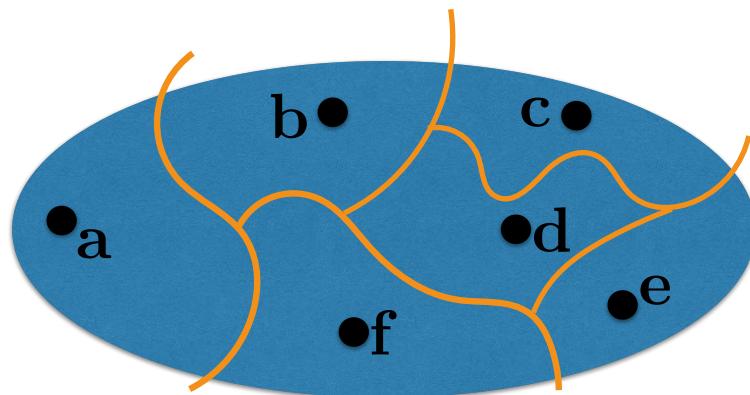
**k=2: Optimal**

**k=3: it's a 4/3 approximation**

# Multiway cut, linear programming and randomized rounding



# Multiway cut, linear programming and randomized rounding

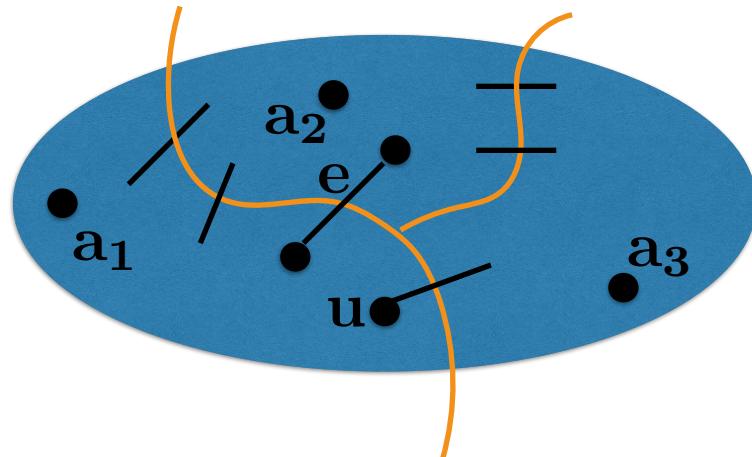


## Variab‌les

$x_{u,i} = 1$  iff  
vertex  $u$  belongs to the  
cluster  
of terminal  $a_i$ .

$z_{e,i} = 1$  iff  
removal of edge  $e$   
separates  $a_i$  from  
some other terminal

## IP model



## IP model

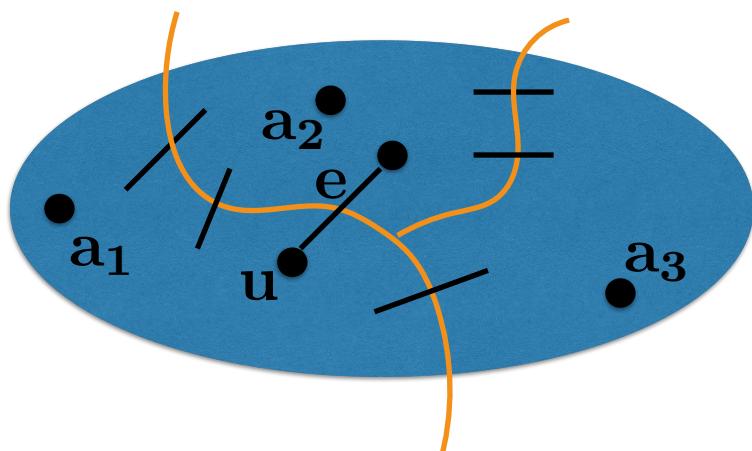
### Constraints

$$x_{u,i}, z_{e,i} \in \{0, 1\}$$

$x_{a_i,i} = 1$  ( $a_i$  belongs to its own cluster)

$\sum_i x_{u,i} = 1$  ( $u$  belongs to some cluster)

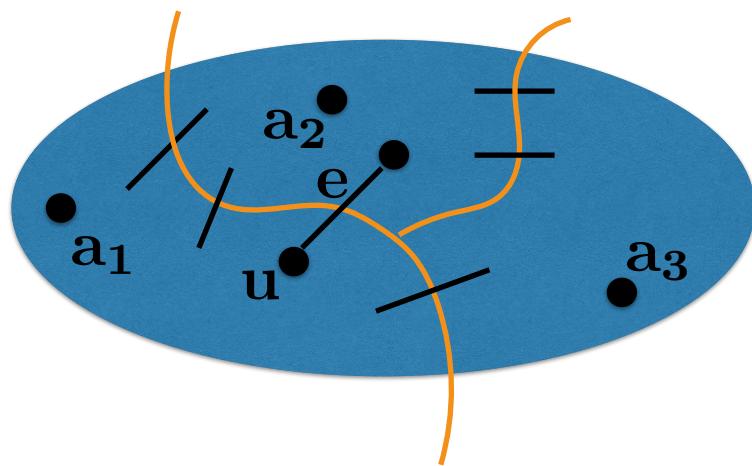
$z_{uv,i} = 1 \quad \text{iff} \quad x_{u,i} \neq x_{v,i} \quad ???$



$z_{uv,i} = 1 \quad \text{iff} \quad x_{u,i} \neq x_{v,i}$  IP model

## Objective

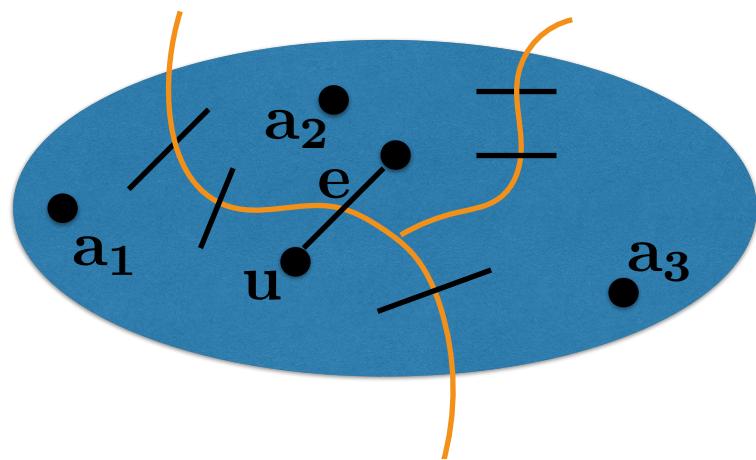
$$\min (1/2) \sum_{e,i} c_e z_{e,i}$$



$$z_{uv,i} = 1 \quad \text{iff} \quad x_{u,i} \neq x_{v,i} \quad \text{IP model}$$

How do we express that constraint?

$$\frac{z_{e,i} \geq |x_{u,i} - x_{v,i}|}{\begin{aligned} z_{e,i} &\geq x_{u,i} - x_{v,i} \\ z_{e,i} &\geq x_{v,i} - x_{u,i} \end{aligned}}$$



# IP model

$$x_{u,i}, z_{e,i} \in \{0, 1\}$$

$$x_{a_i,i} = 1$$

$$\sum_i x_{u,i} = 1$$

$$z_{e,i} \geq x_{u,i} - x_{v,i}$$

$$z_{e,i} \geq x_{v,i} - x_{u,i}$$

$$\min (1/2) \sum_{e,i} c_e z_{e,i}$$

## Linear programming relaxation

$$x_{u,i}, z_{e,i} \in \{0, 1\} \longrightarrow 0 \leq x_{u,i}, z_{e,i} \leq 1$$

$$x_{a_i,i} = 1$$

$$\sum_i x_{u,i} = 1$$

$$z_{e,i} \geq x_{u,i} - x_{v,i}$$

$$z_{e,i} \geq x_{v,i} - x_{u,i}$$

$$\min (1/2) \sum_{e,i} c_e z_{e,i}$$

## A geometric interpretation

### Variables

$x_{u,i} = 1$  iff

**vertex u belongs  
to the cluster  
of terminal ai.**

Vector  $\mathbf{x}_u = (x_{u,i})_i$

One dimension for each terminal

$$\sum_i x_{u,i} = \sum_i |x_{u,i}| = |\mathbf{x}_u|_1$$

## A geometric interpretation

Vector  $\mathbf{x}_u = (x_{u,i})_i$

One dimension for each terminal

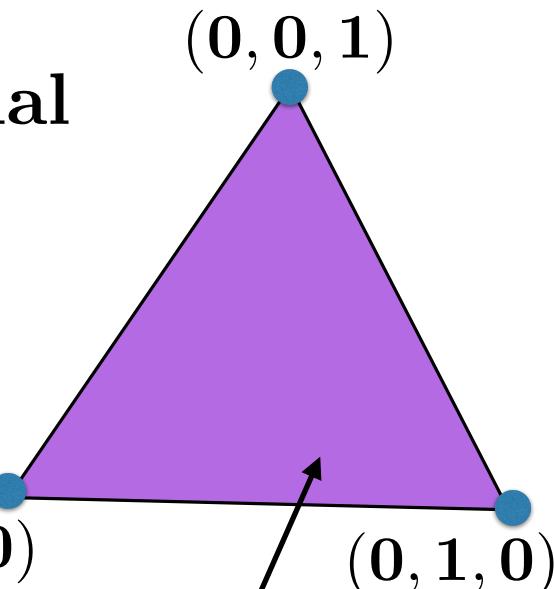
$$\sum_i x_{u,i} = \sum_i |x_{u,i}| = |\mathbf{x}_u|_1$$

$$|\mathbf{x}_u|_1 = 1 \quad \forall u$$

$$\mathbf{x}_u \in [0, 1]^k \quad \forall u$$

$$\mathbf{x}_{a_i} = (0, \dots, 0, 1, 0, \dots, 0) \quad \begin{cases} (1, 0, 0) \\ (0, 1, 0) \end{cases}$$
$$\left\{ \begin{array}{l} x_1 + x_2 + x_3 = 1 \\ x_i \geq 0 \end{array} \right.$$

$$\min \quad (1/2) \sum_{uv \in E} c_{uv} |\mathbf{x}_u - \mathbf{x}_v|_1$$



$$\min \quad (1/2) \sum_{uv \in E} c_{uv} |x_u - x_v|_1$$

$$x_{a_3} = (0, 0, 1)$$

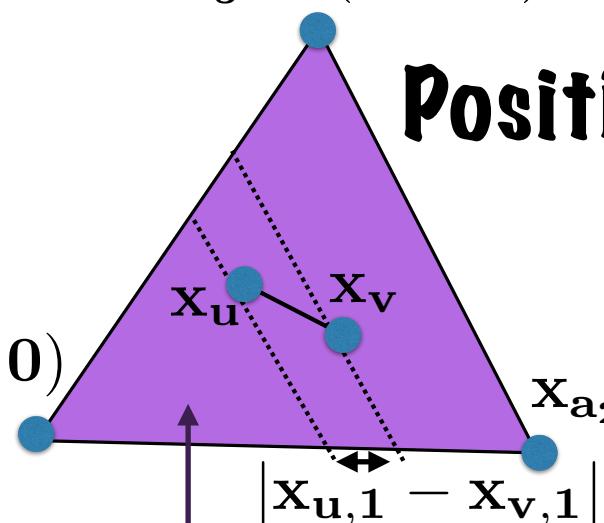
**k=3:**

**Position graph vertices  
in triangle**

$$x_{a_1} = (1, 0, 0)$$

$$x_{a_2} = (0, 1, 0)$$

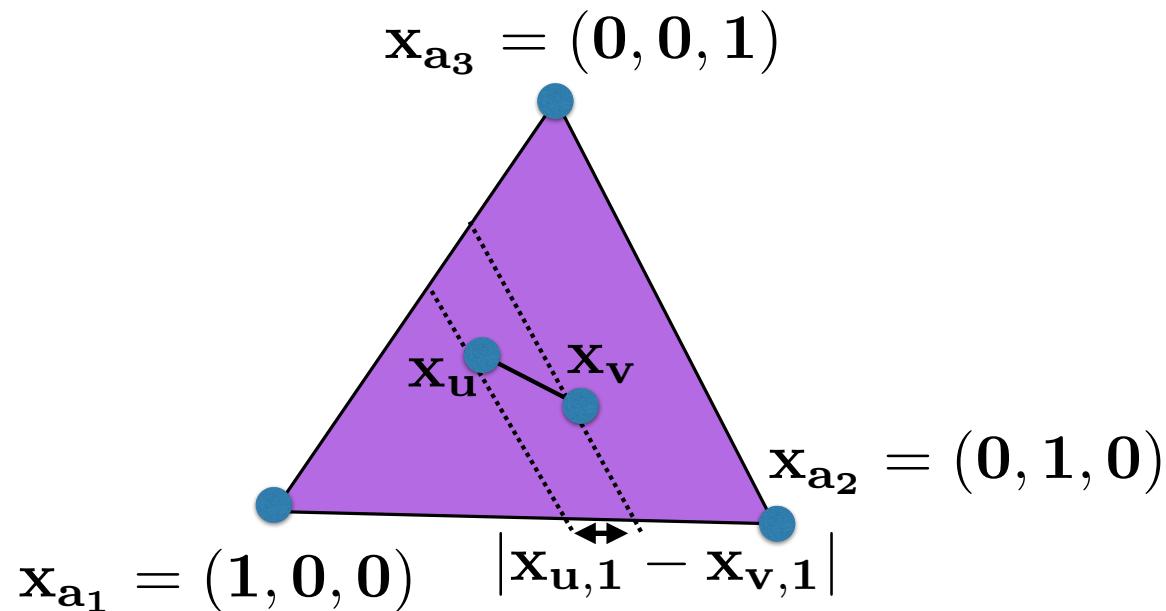
$$\left\{ \begin{array}{l} x_1 + x_2 + x_3 = 1 \\ x_i \geq 0 \end{array} \right.$$



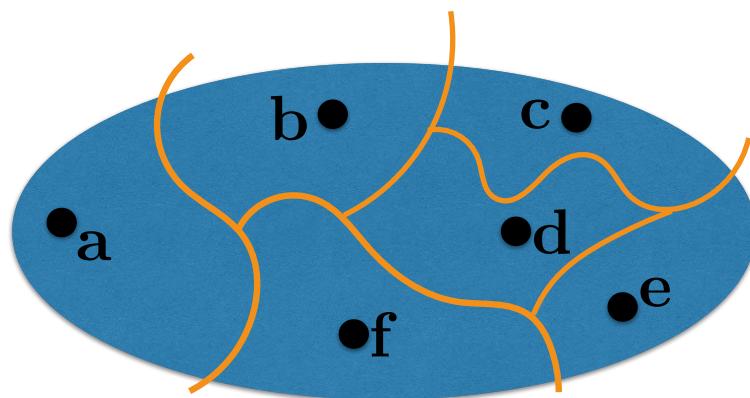
**to minimize  
weighted lengths of  
projections on sides**

## A geometric interpretation ( $k=3$ )

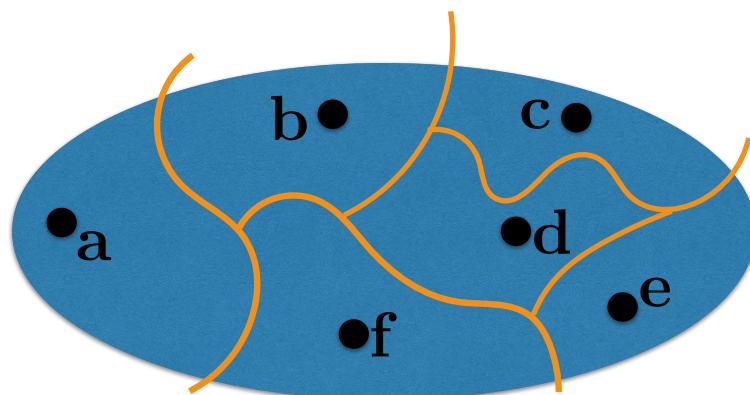
Position graph vertices in triangle  
to minimize weighted lengths of  
projections of graph edges on the sides



# Multiway cut, linear programming and randomized rounding

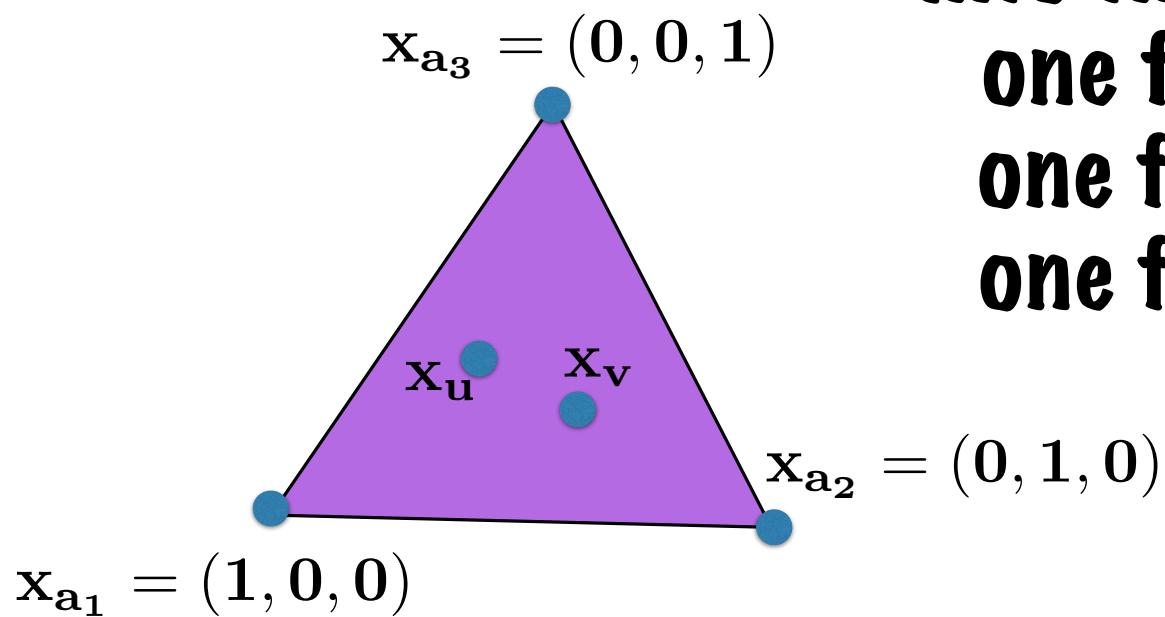


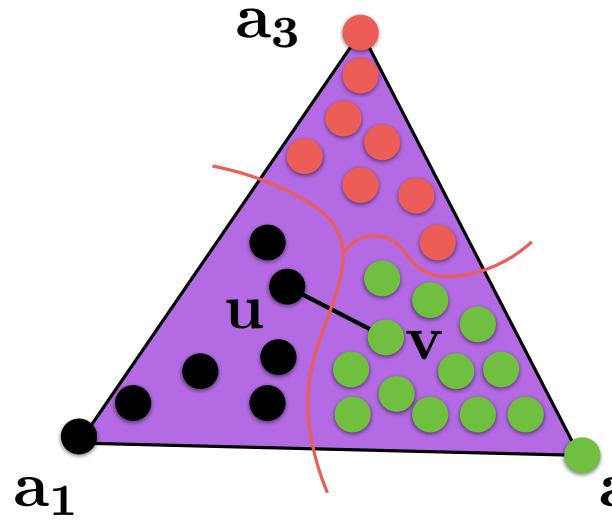
# Multiway cut, linear programming and randomized rounding



## How do we round?

**Partition triangle  
into three areas:  
one for (0,0,1)  
one for (1,0,0)  
one for (0,1,0)**





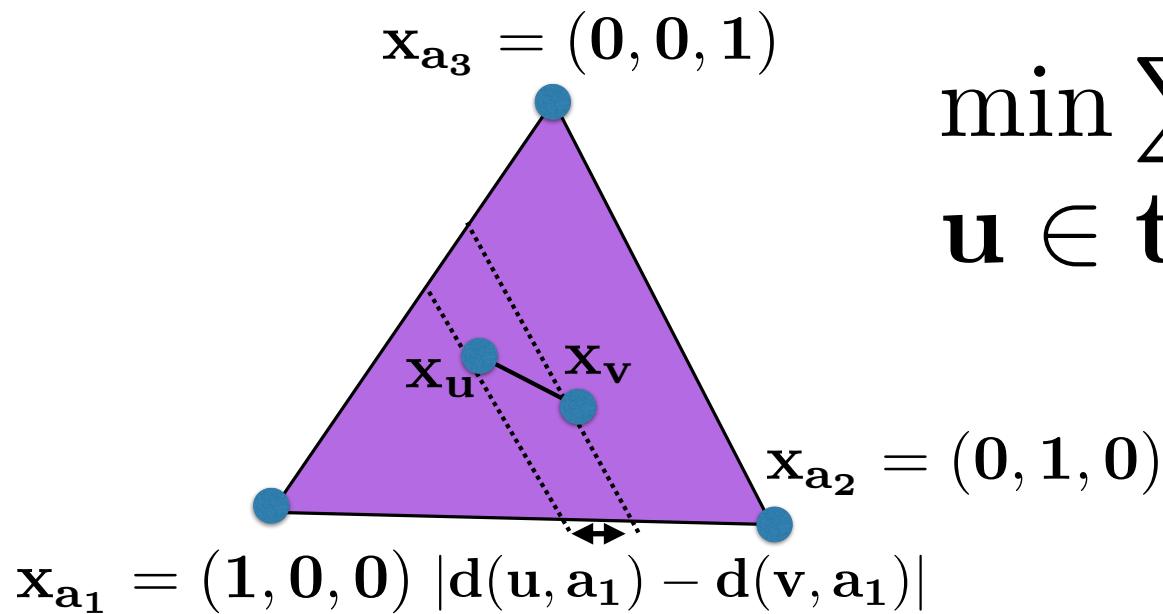
How do we round?

Vertices • go with  $a_1$   
Vertices • go with  $a_2$   
Vertices • go with  $a_3$   
Pay cost of edges across

Good rounding = small cost choice of partition of triangle into three areas

## LP relaxation

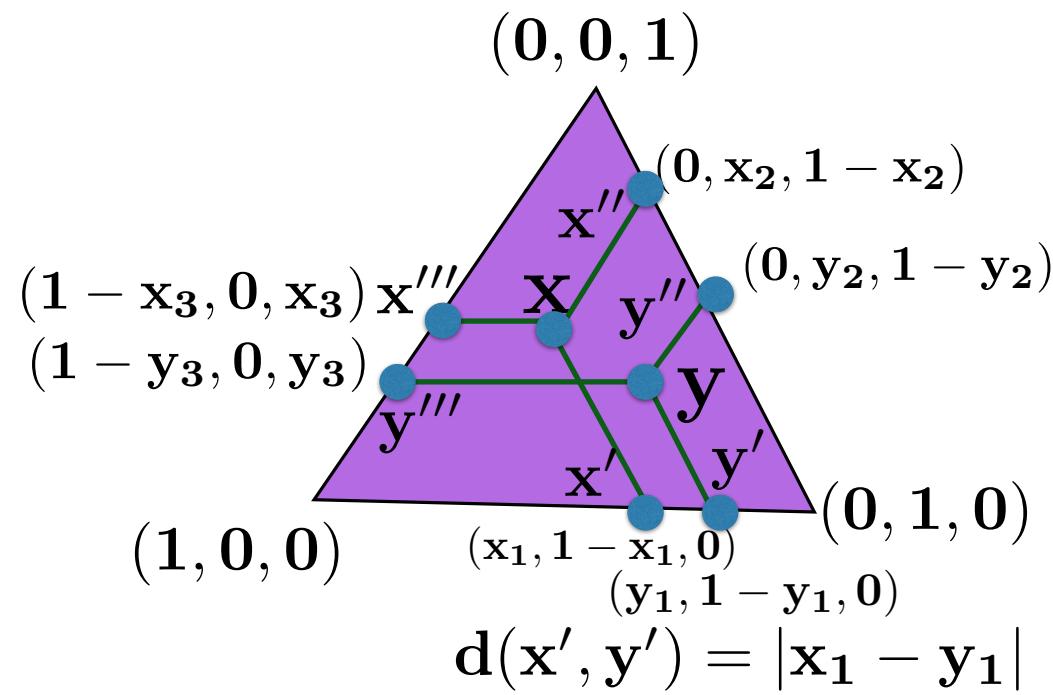
Place vertices in triangle, min lengths of edge projections on sides



$$d(u, v) = \frac{1}{2}|x_u - x_v|_1$$

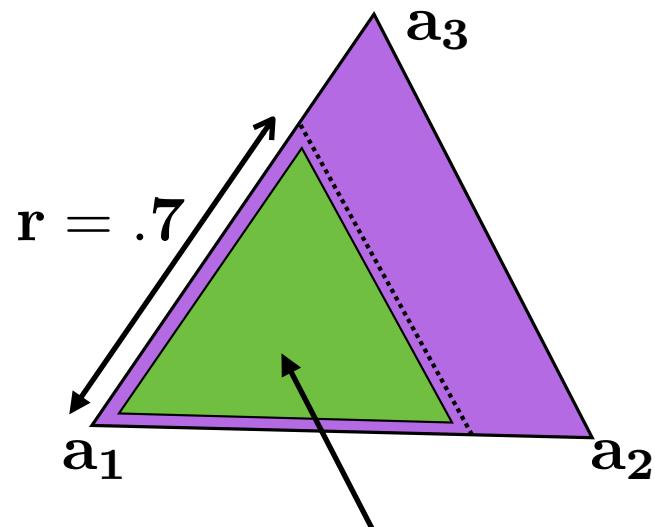
$$\min \sum_{uv \in E} c_{uv} d(u, v) : \\ u \in \text{triangle} \quad \forall u$$

How do we round?



$$\begin{aligned}
 d(u, v) &= \frac{1}{2}(|x_1 - y_1| + |x_2 - y_2| + |x_3 - y_3|) \\
 &= \frac{1}{2}(d(x', y') + d(x'', y'') + d(x''', y'''))
 \end{aligned}$$

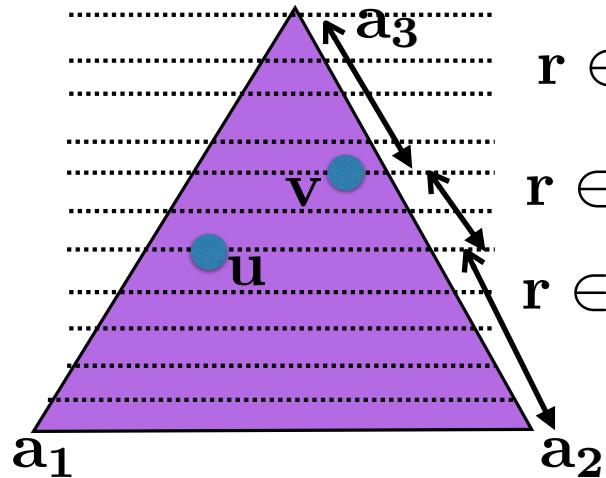
## Using balls in $\ell_1$ metric



$$d(u, v) = \frac{1}{2}|u - v|_1$$
$$d(a_i, a_j) = 1$$

$$B(a_1, r) = \{u : d(a_1, u) \leq r\}$$

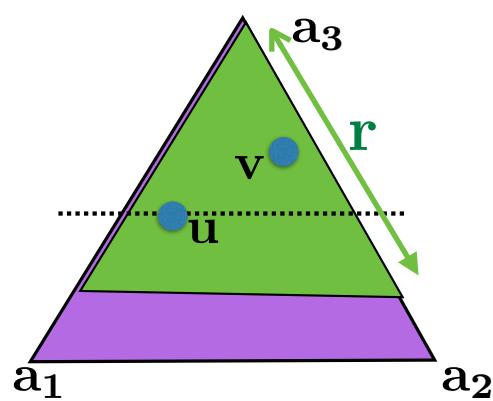
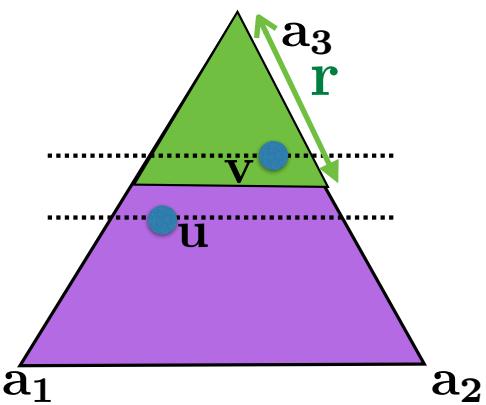
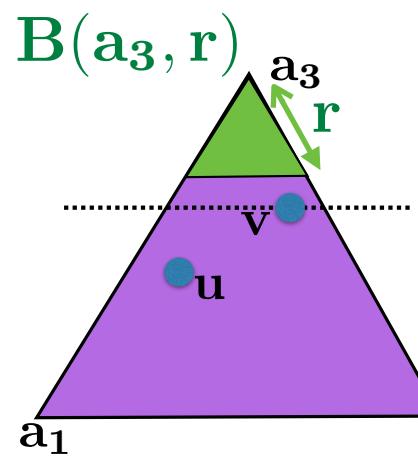
Pick random  $r$ , assign  $B(a_3, r)$  to  $a_3$



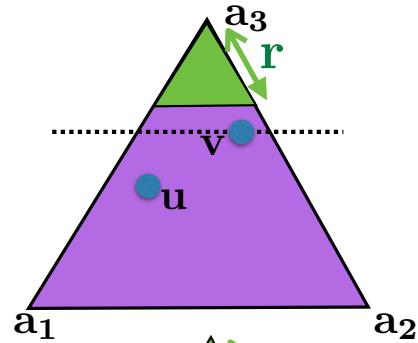
$$r \in [0, d(v, a_3)]$$

$$r \in [d(v, a_3), d(u, a_3)]$$

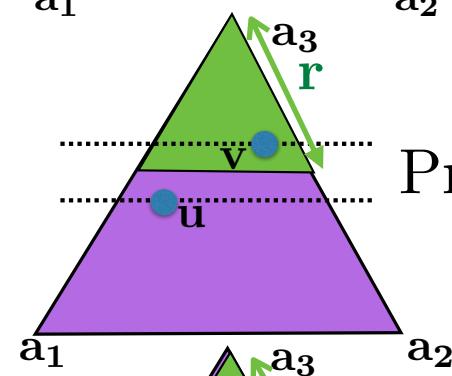
$$r \in [d(u, a_3), 1]$$



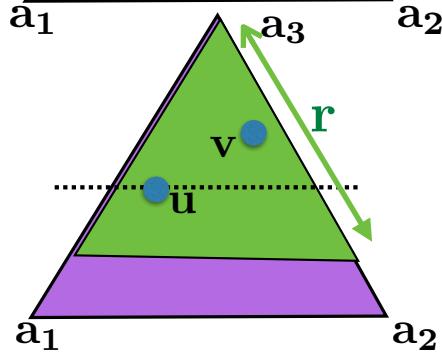
Pick random  $r$ , assign  $B(a_3, r)$  to  $a_3$



$$\Pr[u, v \notin B(a_3, r)] = \Pr[r < d(a_3, v)] = d(a_3, v)$$



$$\begin{aligned} \Pr[u, v \text{ separated by } B(a_3, r)] &= d(a_3, u) - d(a_3, v) \\ &= |\mathbf{u}_3 - \mathbf{v}_3| \end{aligned}$$



$$\Pr[u, v \in B(a_3, r)] = 1 - d(a_3, u)$$

**Consider terminals in random order**

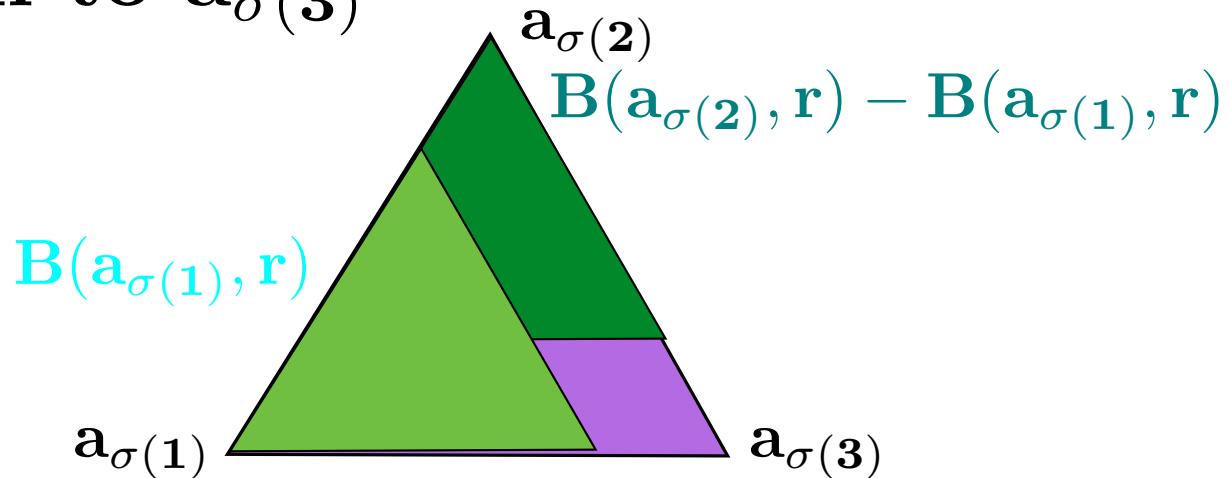
$$\mathbf{a}_{\sigma(1)}, \mathbf{a}_{\sigma(2)}, \mathbf{a}_{\sigma(3)}$$

**Assigning u**

if  $d(\mathbf{a}_{\sigma(1)}, \mathbf{u}) < r$  then assign to  $\mathbf{a}_{\sigma(1)}$

else if  $d(\mathbf{a}_{\sigma(2)}, \mathbf{u}) < r$  then assign to  $\mathbf{a}_{\sigma(2)}$

else assign to  $\mathbf{a}_{\sigma(3)}$



## Full Algorithm for 3-way cut

Solve relaxation: embed vertices in triangle

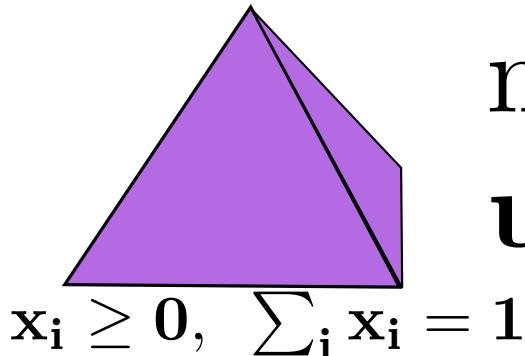
Pick random permutation of terminals

- assign to **first** terminal a all vertices in  $B(a,r)$  where  $r$  is random uniform in  $[0,1]$
- assign to **second** terminal b all unassigned vertices in  $B(b,r)$
- assign to **third** terminal remaining vertices

## Algorithm

$$d(u, v) = \frac{1}{2} |x_u - x_v|_1$$

**LP relaxation: embed vertices in k-simplex  
with terminals at corners**



$$\min \sum_{uv \in E} c_{uv} d(u, v) : \\ u \in \text{k-simplex} \quad \forall u$$

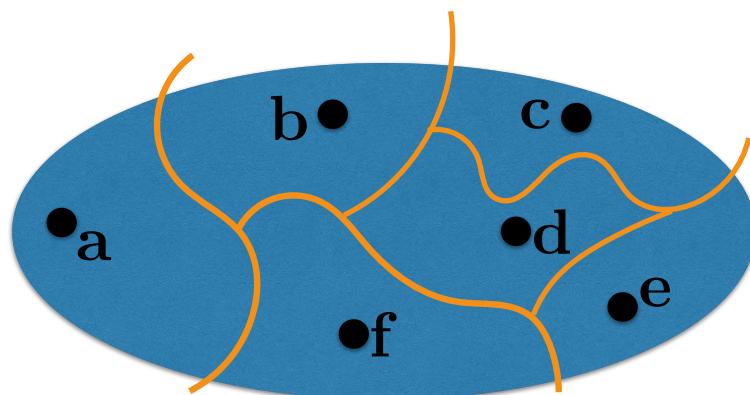
**random ordering**  $a_{\sigma(1)}, a_{\sigma(2)}, \dots, a_{\sigma(k)}$   $r \in [0, 1]$

for  $i = 1, \dots, k - 1$  :

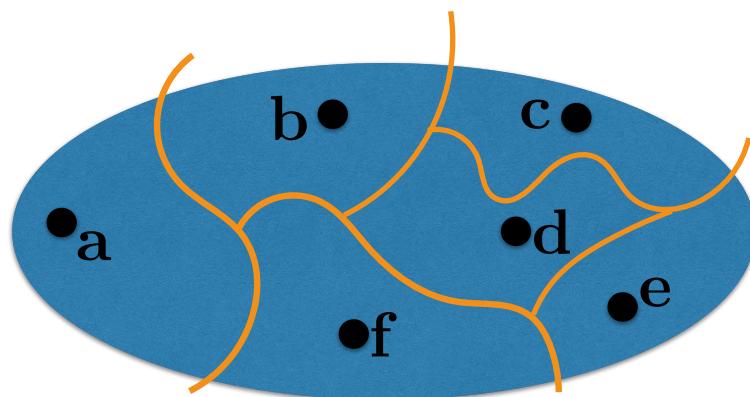
    assign to  $a_{\sigma(i)}$  unassigned vertices of  $B(a_{\sigma(i)}, r)$

    assign rest to  $a_{\sigma(k)}$

# Multiway cut, linear programming and randomized rounding



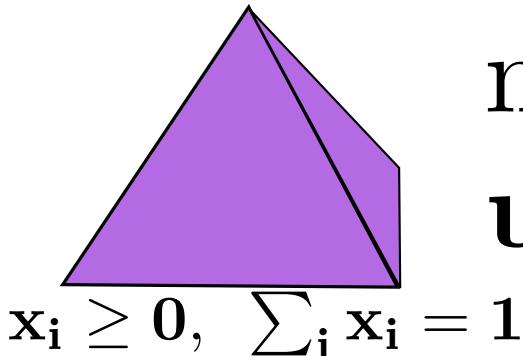
# Multiway cut, linear programming and randomized rounding



## Algorithm

$$d(u, v) = \frac{1}{2} |x_u - x_v|_1$$

**LP relaxation: embed vertices in k-simplex  
with terminals at corners**



$$\min \sum_{uv \in E} c_{uv} d(u, v) : \\ u \in \text{k-simplex} \quad \forall u$$

**random ordering**  $a_{\sigma(1)}, a_{\sigma(2)}, \dots, a_{\sigma(k)}$   $r \in [0, 1]$

for  $i = 1, \dots, k - 1$  :

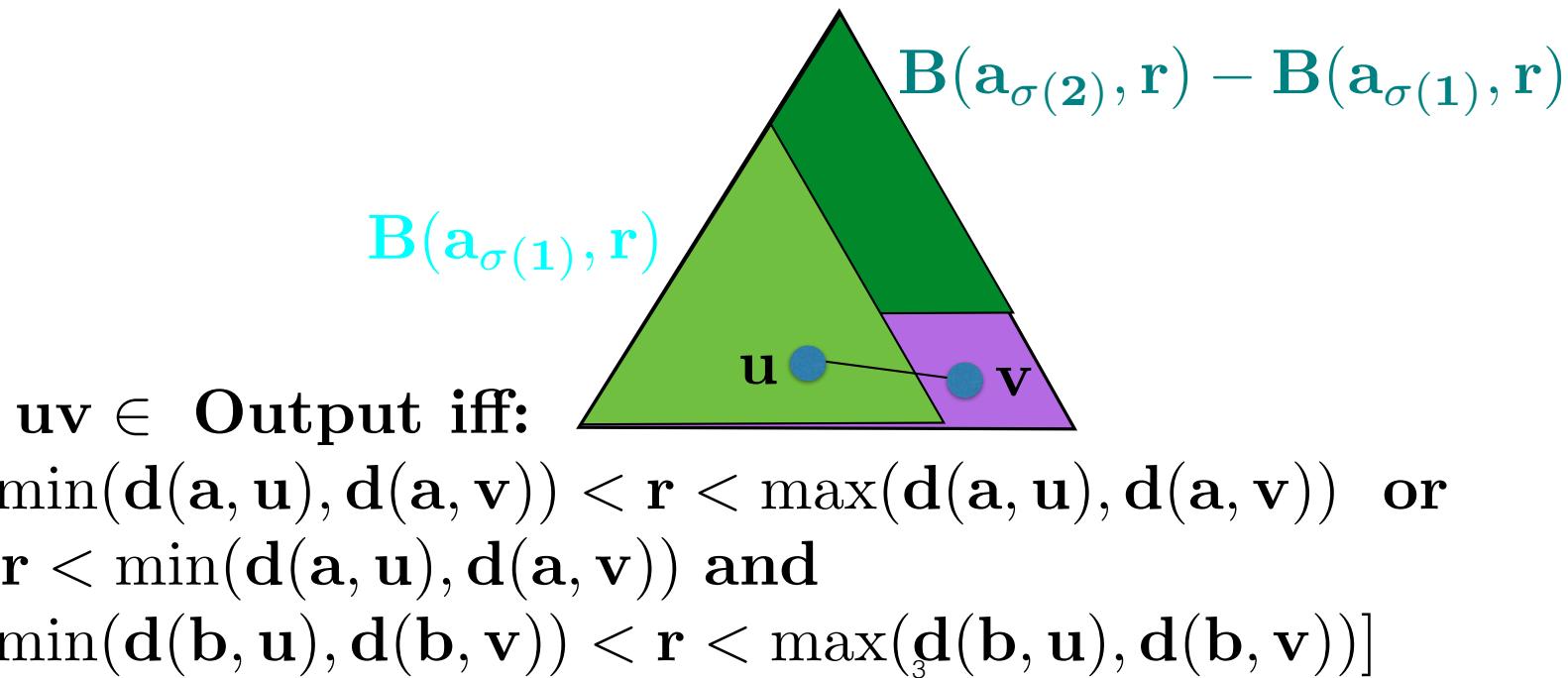
    assign to  $a_{\sigma(i)}$  unassigned vertices of  $B(a_{\sigma(i)}, r)$

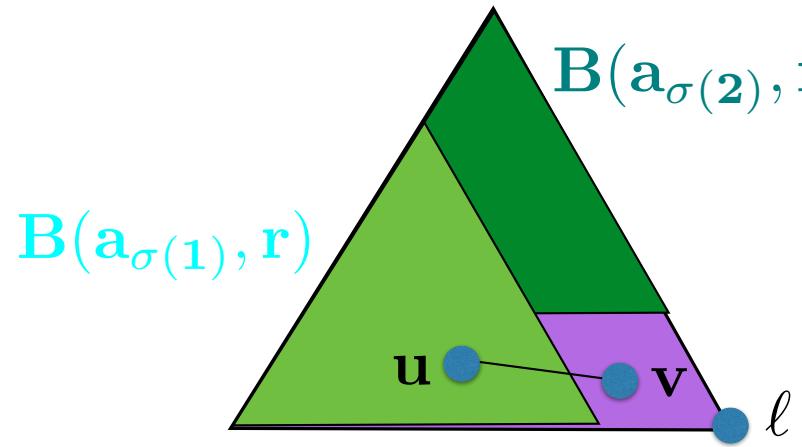
    assign rest to  $a_{\sigma(k)}$

# Analysis

$$E[Output] = \sum_{uv \in E} c_{uv} \Pr(uv \in Output)$$

$$a = a_{\sigma(1)}, b = a_{\sigma(2)}, c = a_{\sigma(3)}$$





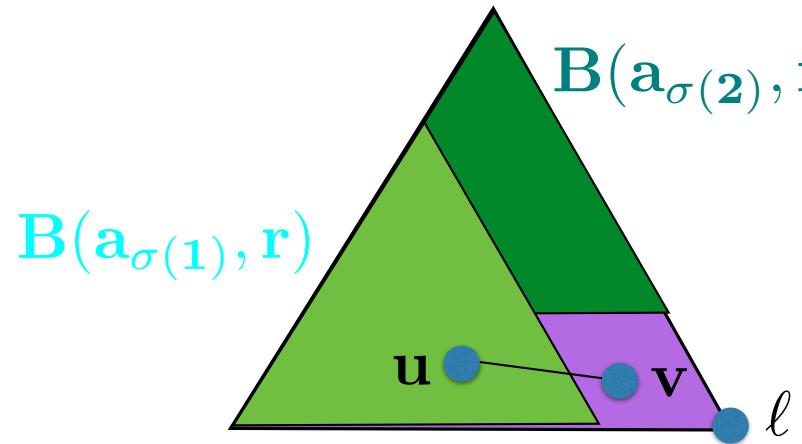
$$B(a_{\sigma(2)}, r) - B(a_{\sigma(1)}, r)$$

**uv ∈ Output:**  
 $a_i$  first to catch  $u$  or  $v$   
 and catches exactly one

$$\ell = \arg \min \{ \min(d(a_i, u), d(a_i, v)) \}$$

$i \neq \ell : i$  precedes  $\ell$  and  $B(a_i, r)$  separates them

$$\frac{1}{2} |d(a_i, u) - d(a_i, v)|$$



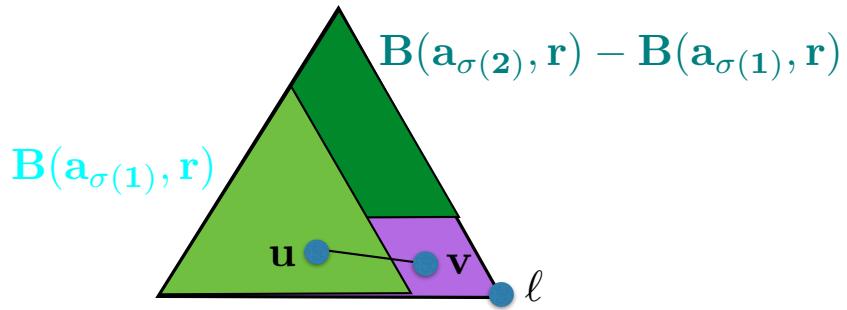
$$B(a_{\sigma(2)}, r) - B(a_{\sigma(1)}, r)$$

$uv \in \text{Output}:$   
 $a_i$  first to catch  $u$  or  $v$   
 and catches exactly one

$$\ell = \arg \min \{ \min(d(a_i, u), d(a_i, v)) \}$$

$i = \ell : \ell$  not last and  $B(a_\ell, r)$  separates them

$$(1 - \frac{1}{k}) |d(a_\ell, u) - d(a_\ell, v)|$$

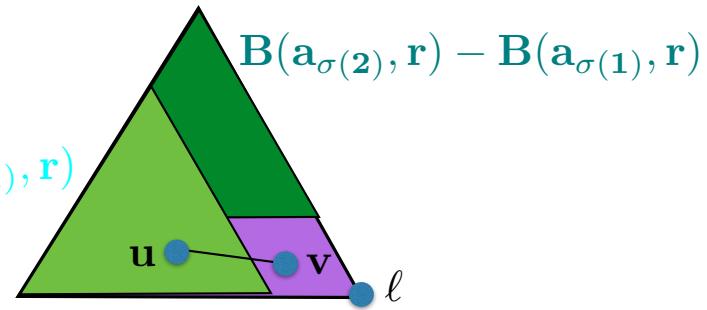


Together

$$(1 - \frac{1}{k})|d(a_\ell, u) - d(a_\ell, v)| + \sum_{i \neq \ell} \frac{1}{2}|d(a_i, u) - d(a_i, v)|$$

$$= \frac{1}{2}(1 - \frac{2}{k})|d(a_\ell, u) - d(a_\ell, v)| + \sum_i \frac{1}{2}|d(a_i, u) - d(a_i, v)|$$

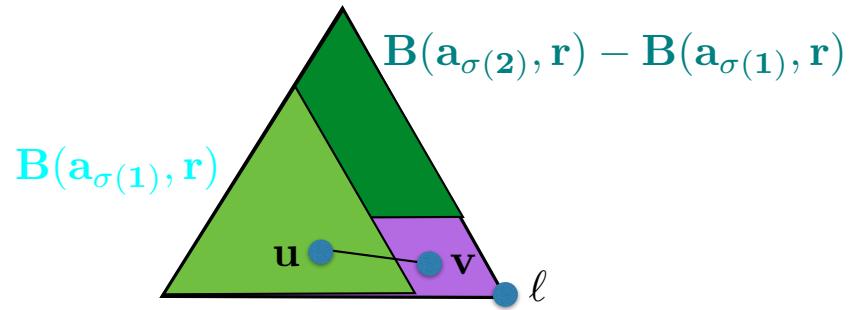
$$= \frac{1}{2}(1 - \frac{2}{k})|u_\ell - v_\ell| + \sum_i \frac{1}{2}|u_i - v_i|$$



$$\sum_{\mathbf{u}} \mathbf{u}_i = \sum_{\mathbf{i}} \mathbf{v}_i = 1$$

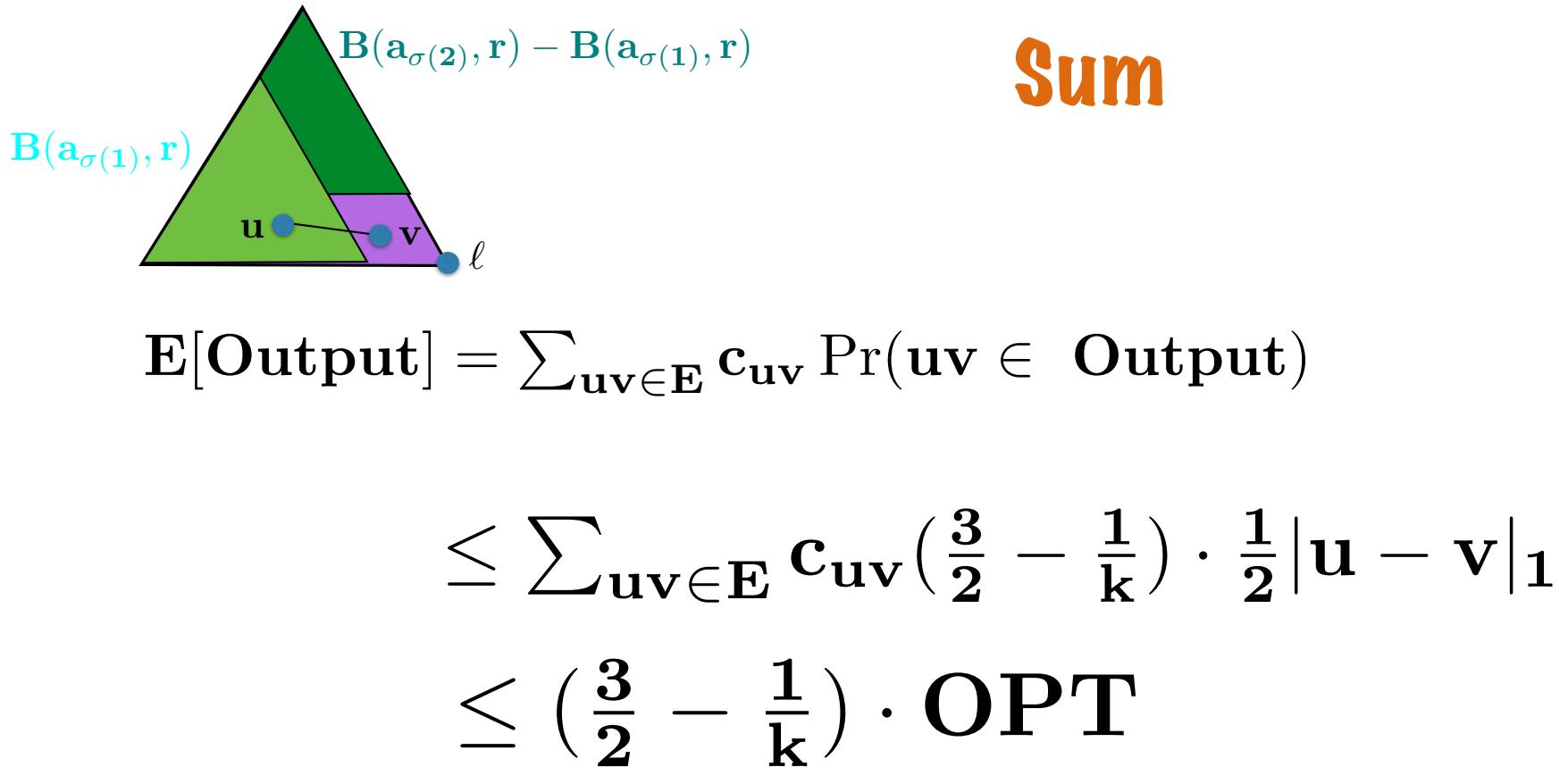
**so**

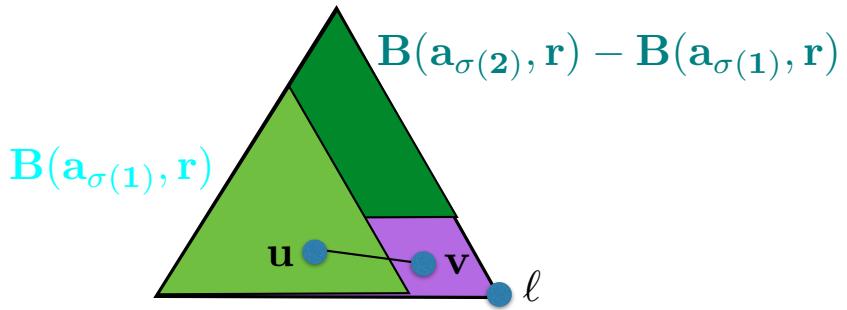
$$|\mathbf{u}_\ell - \mathbf{v}_\ell| \leq \frac{1}{2} \sum_{\mathbf{i}} |\mathbf{u}_i - \mathbf{v}_i|$$



Together

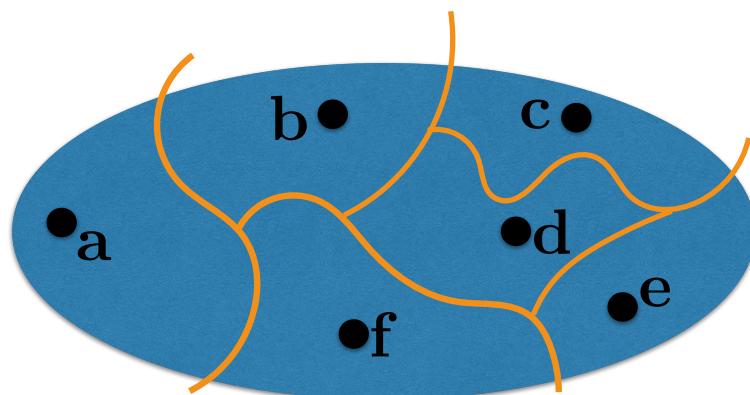
$$\begin{aligned}
 & \left(\frac{1}{2} - \frac{1}{k}\right) |\mathbf{u}_\ell - \mathbf{v}_\ell| + \sum_i \frac{1}{2} |\mathbf{u}_i - \mathbf{v}_i| \\
 & \leq \sum_i \left(\frac{3}{4} - \frac{1}{2k}\right) |\mathbf{u}_i - \mathbf{v}_i| \\
 & = \left(\frac{3}{2} - \frac{1}{k}\right) \cdot \frac{1}{2} |\mathbf{u} - \mathbf{v}|_1
 \end{aligned}$$



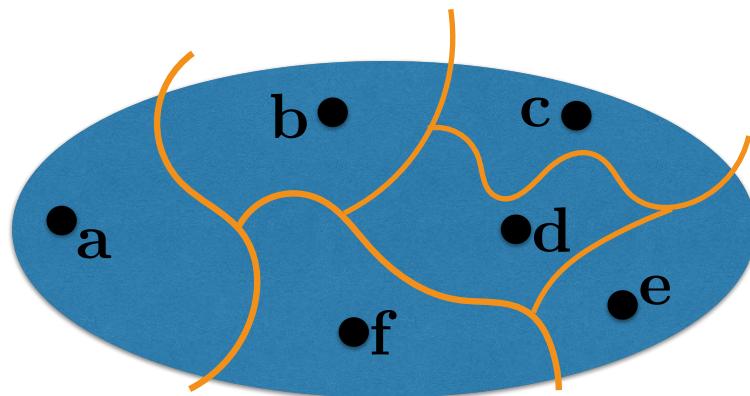


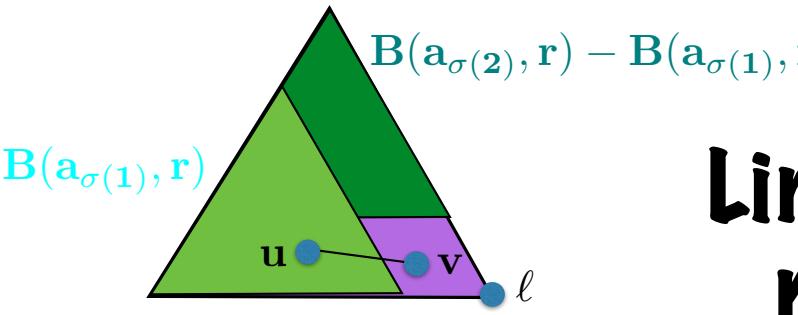
Linear programming and  
randomized rounding  
give a  $3/2 - 1/k$   
approximation  
for multicut

# Multiway cut, linear programming and randomized rounding



# Multiway cut, linear programming and randomized rounding





Linear programming and  
randomized rounding  
give a  $3/2 - 1/k$   
approximation  
for multicut

Can we do better?

**k=3: 12/11**  
**by using LP to find the rounding!**

**APX-hard: cannot get  $1 + \epsilon$**

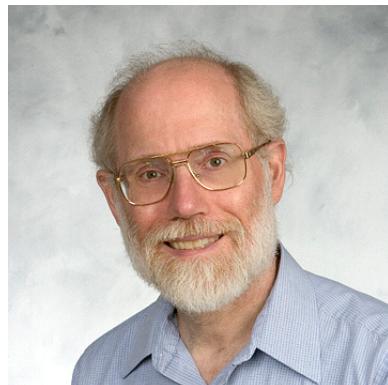
The story behind the story

## **Applications:**

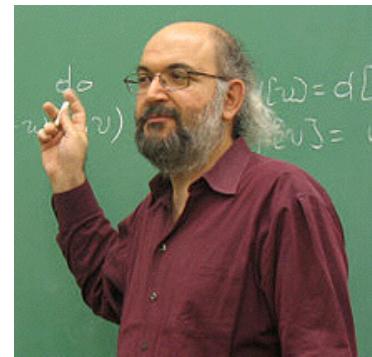
- “minimization of communication costs in parallel computing systems...”
- assigning program modules to processors ...
- partitioning files among the nodes of a network...
- assigning users to base computers in a multicomputer environment...
- partitioning the elements of a circuit into the subcircuits that will go on different chips”



**Elias Dalhaus**



**David Johnson**



**Mihalis  
Yannakakis**



**Christos Papadimitriou**



**Paul Seymour**

**APX hardness  
approx with min cuts**



**Gruia Calinescu**



**Howard Karloff**

**Geometric embedding**

$$3/2 - 1/k$$



**Yuval Rabani**



**David Karger**



**Cliff Stein**



**Philip Klein**



**Neal Young**



**Mikkel Thorup**

**12/11  
rounding by  
linear programming**

## Techniques

rounding input

linear programming relaxation

randomized rounding

probabilistic analysis techniques

geometric interpretation

## Problems

Vertex cover

Knapsack

Bin packing

Set cover

Multiway cut

## Approximation algorithms, Part II

LP duality

primal dual algorithms

semi-definite programming

# Multiway cut, linear programming and randomized rounding

