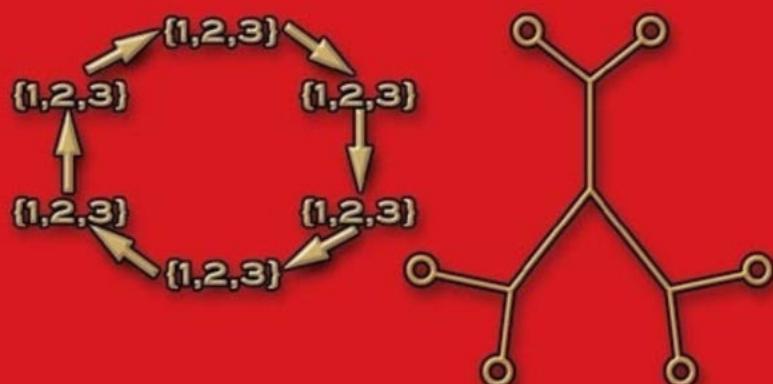
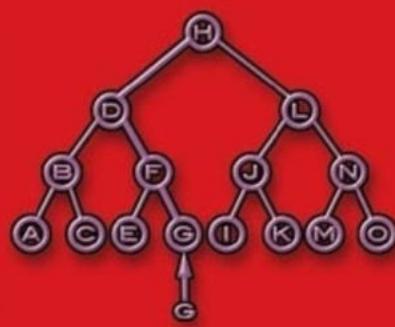
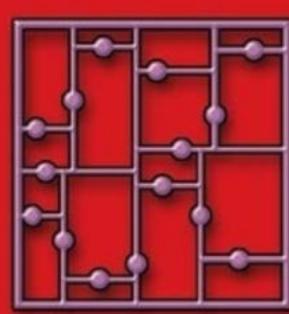
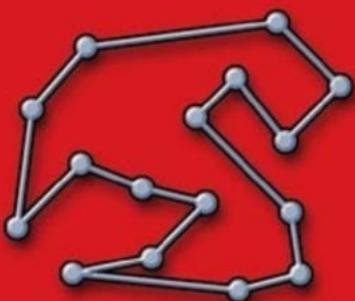


# Algorithm Design



XYZXYZ\$  
YZXYZ\$  
ZXYZ\$  
XYZ\$  
YZ\$  
Z\$  
\$



# Program

Lecture 1 Stable Matching and other representative problems

Lecture 2 Overview of Algorithm Analysis

Lecture 4 Greedy Algorithms

Lecture 5 Network Flow: Ford and Fulkerson, Capacity Scaling, Shortest Augmenting Paths

Lecture 6 Perfect Matching, Circulations, Improved algorithm for bipartite matching

Lecture 7 Dynamic Programming

Lecture 9 Basics of NP-Completeness

Lecture 10 Polynomial Time Reductions

# Program

Lecture 11 Basics of approximation algorithms

Lecture 12 Basics of approximation algorithms

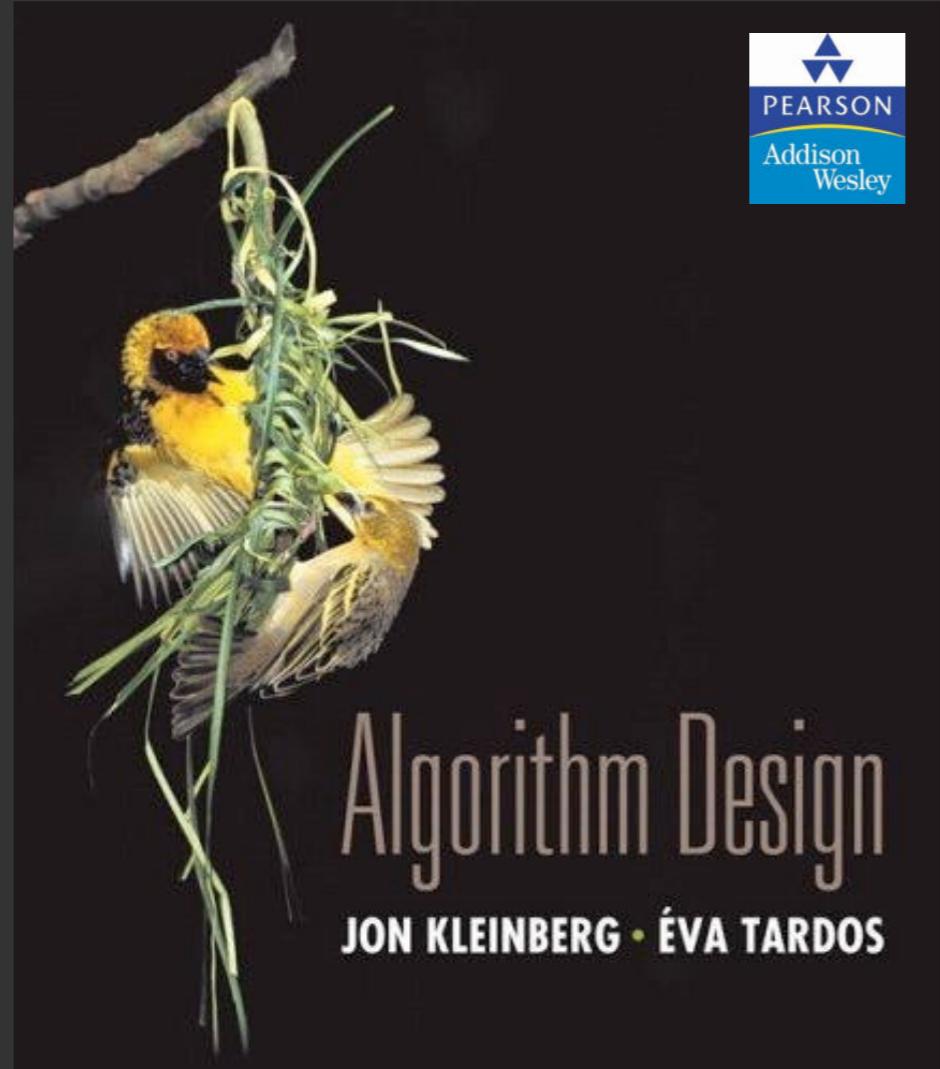
Lecture 13 Linear Programming based approximation algorithms

Lecture 14 Randomized Algorithms

Lecture 15 Introduction to Game Theory and Equilibria Computation

Lecture 16 Computing Equilibria in Zero-Sum Games and Efficiency of Equilibria

Lecture 17 Selfish Routing and Price of Anarchy



## 1. REPRESENTATIVE PROBLEMS

---

- ▶ *stable matching*
- ▶ *five representative problems*

Lecture slides by Kevin Wayne

Copyright © 2005 Pearson–Addison Wesley

<http://www.cs.princeton.edu/~wayne/kleinberg-tardos>

# Matching med-school students to hospitals

---

**Goal.** Given a set of preferences among hospitals and med-school students, design a **self-reinforcing** admissions process.

**Unstable pair.** Hospital  $h$  and student  $s$  form an **unstable pair** if both:

- $h$  prefers  $s$  to one of its admitted students.
- $s$  prefers  $h$  to assigned hospital.

**Stable assignment.** Assignment with no unstable pairs.

- Natural and desirable condition.
- Individual self-interest prevents any hospital–student side deal.



# Stable matching problem: input

**Input.** A set of  $n$  hospitals  $H$  and a set of  $n$  students  $S$ .

- Each hospital  $h \in H$  ranks students.
- Each student  $s \in S$  ranks hospitals.

one student per hospital (for now)

favorite  
↓  
least favorite

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

**hospitals' preference lists**

favorite  
↓  
least favorite

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

**students' preference lists**

# Perfect matching

---

Def. A **matching**  $M$  is a set of ordered pairs  $h-s$  with  $h \in H$  and  $s \in S$  s.t.

- Each hospital  $h \in H$  appears in at most one pair of  $M$ .
- Each student  $s \in S$  appears in at most one pair of  $M$ .

Def. A matching  $M$  is **perfect** if  $|M| = |H| = |S| = n$ .

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

a perfect matching  $M = \{ A-Z, B-Y, X-X \}$

## Unstable pair

---

**Def.** Given a perfect matching  $M$ , hospital  $h$  and student  $s$  form an **unstable pair** if both:

- $h$  prefers  $s$  to matched student.
- $s$  prefers  $h$  to matched hospital.

**Key point.** An unstable pair  $h-s$  could each improve by joint action.

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

A-Y is an **unstable pair**

# Stable matching problem

---

Def. A **stable matching** is a perfect matching with no unstable pairs.

**Stable matching problem.** Given the preference lists of  $n$  hospitals and  $n$  students, find a stable matching (if one exists).

- Natural, desirable, and self-reinforcing condition.
- Individual self-interest prevents any hospital–student pair from breaking commitment.

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

a stable matching  $M = \{ A-X, B-Y, C-Z \}$

# Stable roommate problem

---

Q. Do stable matchings always exist?

A. Not obvious a priori.

## Stable roommate problem.

- $2n$  people; each person ranks others from 1 to  $2n - 1$ .
- Assign roommate pairs so that no unstable pairs.

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	
A	B	C	D	<b>no perfect matching is stable</b>
B	C	A	D	$A-B, C-D \Rightarrow B-C$ unstable
C	A	B	D	$A-C, B-D \Rightarrow A-B$ unstable
D	A	B	C	$A-D, B-C \Rightarrow A-C$ unstable

Observation. Stable matchings need not exist.

# Gale–Shapley deferred acceptance algorithm

An intuitive method that **guarantees** to find a stable matching.



**GALE–SHAPLEY** (*preference lists for hospitals and students*)

**INITIALIZE**  $M$  to empty matching.

**WHILE** (some hospital  $h$  is unmatched and hasn't proposed to every student)

$s \leftarrow$  first student on  $h$ 's list to whom  $h$  has not yet proposed.

**IF** ( $s$  is unmatched)

Add  $h-s$  to matching  $M$ .

**ELSE IF** ( $s$  prefers  $h$  to current partner  $h'$ )

Replace  $h'-s$  with  $h-s$  in matching  $M$ .

**ELSE**

$s$  rejects  $h$ .

**RETURN** stable matching  $M$ .

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

# Gale-Shapley demo

---

men's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

Victor proposes to Bertha

women's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

# Gale-Shapley demo

men's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

women's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

Victor proposes to Bertha

Bertha accepts  
(since previously unmatched)

# Gale-Shapley demo

men's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

Wyatt proposes to Diane

women's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

# Gale-Shapley demo

men's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

women's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

Wyatt proposes to Diane

Diane accepts  
(since previously unmatched)

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

Xavier proposes to Bertha

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

# Gale-Shapley demo

men's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

women's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

Xavier proposes to Bertha

Bertha accepts  
(and dumps Victor)

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

Victor proposes to Amy

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

# Gale-Shapley demo

men's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

women's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

Victor proposes to Amy

Amy accepts  
(since previously unmatched)

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

**Yancey proposes to Amy**

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

# Gale-Shapley demo

men's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

women's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

Yancey proposes to Amy

Amy rejects  
(since she prefers Victor)

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

Yancey proposes to Diane

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

# Gale-Shapley demo

men's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

women's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

Yancey proposes to Diane

Dianne accepts  
(and dumps Wyatt)

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

Wyatt proposes to Bertha

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

# Gale-Shapley demo

men's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

women's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

Wyatt proposes to Bertha  
Bertha rejects  
(since she prefers Xavier)

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

Wyatt proposes to Amy

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

Wyatt proposes to Amy  
Amy rejects  
(since she prefers Victor)

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

Wyatt proposes to Clare

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

Wyatt proposes to Clare

Clare accepts  
(since previously unmatched)

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

Zeus proposes to Bertha

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

**Zeus proposes to Bertha**

**Bertha rejects  
(since she prefers Xavier)**

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

**Zeus proposes to Diane**

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

# Gale-Shapley demo

men's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

women's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

Zeus proposes to Diane

Diane accepts  
(and dumps Yancey)

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

Yancey proposes to Clare

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

Yancey proposes to Clare

Clare rejects  
(since she prefers Wyatt)

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

**Yancey proposes to Bertha**

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

**Yancey proposes to Bertha**

**Bertha rejects  
(since she prefers Xavier)**

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

Yancey proposes to Erika

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

# Gale-Shapley demo

---

**men's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

**women's preference list**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

Yancey proposes to Erika

Erika accepts  
(since previously unmatched)

# Gale-Shapley demo

men's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

women's preference list

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

STOP  
(stable matching)

## Proof of correctness: termination

---

**Observation 1.** Hospitals propose to students in decreasing order of preference.

**Observation 2.** Once a student is matched, the student never becomes unmatched; only “trades up.”

**Claim.** Algorithm terminates after at most  $n^2$  iterations of while loop.

**Pf.** Each time through the while loop a hospital proposes to a new student. There are only  $n^2$  possible proposals. ▀

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Atlanta	A	B	C	D	E
Boston	B	C	D	A	E
Chicago	C	D	A	B	E
Dallas	D	A	B	C	E
Eugene	A	B	C	D	E

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>
Val	W	X	Y	Z	V
Wayne	X	Y	Z	V	W
Xavier	Y	Z	V	W	X
Yolanda	Z	V	W	X	Y
Zeus	V	W	X	Y	Z

$n(n-1) + 1$  proposals required

## Proof of correctness: perfection

---

**Claim.** Gale–Shapley produces a matching.

**Pf.** Hospital proposes only if unmatched; student

**Claim.** In Gale–Shapley matching, all hospitals get matched.

**Pf.** [by contradiction]

- Suppose, for sake of contradiction, that some hospital  $h \in H$  is not matched upon termination of Gale–Shapley algorithm.
- Then some student, say  $s \in S$ , is not matched upon termination.
- By Observation 2,  $s$  was never proposed to.
- But,  $h$  proposes to every student, since  $h$  ends up unmatched.

**Claim.** In Gale–Shapley matching, all students get matched.

**Pf.**

- By previous claim, all  $n$  hospitals get matched.
- Thus, all  $n$  students get matched. ▀

## Proof of correctness: stability

---

**Claim.** In Gale–Shapley matching  $M^*$ , there are no unstable pairs.

**Pf.** Suppose that  $M^*$  does not contain the pair  $h-s$ .

- Case 1:  $h$  never proposed to  $s$ .

$\Rightarrow h$  prefers its Gale–Shapley partner  $s'$  to  $s$ .  
hospitals propose in  
decreasing order  
of preference

$\Rightarrow h-s$  is not unstable.

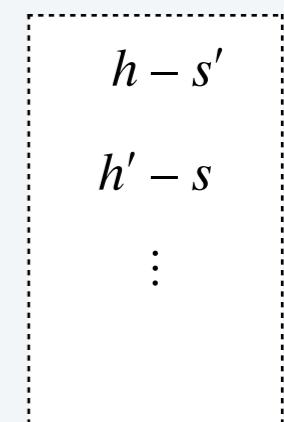
- Case 2:  $h$  proposed to  $s$ .

$\Rightarrow s$  rejected  $h$  (right away or later)

$\Rightarrow s$  prefers Gale–Shapley partner  $h'$  to  $h$ .

$\Rightarrow h-s$  is not unstable.

- In either case, the pair  $h-s$  is not unstable. ▀



Gale–Shapley matching  $M^*$

# Summary

---

**Stable matching problem.** Given  $n$  hospitals and  $n$  students, and their preferences, find a stable matching if one exists.

**Theorem.** [Gale–Shapley 1962] The Gale–Shapley algorithm guarantees to find a stable matching for **any** problem instance.

- Q. How to implement Gale–Shapley algorithm efficiently?
- Q. If multiple stable matchings, which one does Gale–Shapley find?

## COLLEGE ADMISSIONS AND THE STABILITY OF MARRIAGE

D. GALE\* AND L. S. SHAPLEY, Brown University and the RAND Corporation

**1. Introduction.** The problem with which we shall be concerned relates to the following typical situation: A college is considering a set of  $n$  applicants of which it can admit a quota of only  $q$ . Having evaluated their qualifications, the admissions office must decide which ones to admit. The procedure of offering admission only to the  $q$  best-qualified applicants will not generally be satisfactory, for it cannot be assumed that all who are offered admission will accept. Accordingly, in order for a college to receive  $q$  acceptances, it will generally have to offer to admit more than  $q$  applicants. The problem of determining how many and which ones to admit requires some rather involved guesswork. It may not be known (a) whether a given applicant has also applied elsewhere; if this is known it may not be known (b) how he ranks the colleges to which he has applied; even if this is known it will not be known (c) which of the other colleges will offer to admit him. A result of all this uncertainty is that colleges can expect only that the entering class will come reasonably close in numbers to the desired quota, and be reasonably close to the attainable optimum in quality.

# Efficient implementation

---

Efficient implementation. We describe an  $O(n^2)$  time implementation.

Representing hospitals and students. Index hospitals and students  $1, \dots, n$ .

Representing the matching.

- Maintain a list of free hospitals (in a stack or queue).
- Maintain two arrays  $student[h]$  and  $hospital[s]$ .
  - if  $h$  matched to  $s$ , then  $student[h] = s$  and  $hospital[s] = h$
  - use value 0 to designate that hospital or student is unmatched

Hospitals proposing.

- For each hospital, maintain a list of students, ordered by preference.
- For each hospital, maintain a pointer to students in list for next proposal.

## Efficient implementation (continued)

---

### Students rejecting/accepting.

- Does student  $s$  prefer hospital  $h$  to hospital  $h'$  ?
- For each student, create **inverse** of preference list of hospitals.
- Constant time access for each query after  $O(n)$  preprocessing.

<b>pref[]</b>	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>
	8	3	7	1	4	5	6	2

student prefers hospital 3 to 6  
since  $\text{inverse}[3] < \text{inverse}[6]$

<b>inverse[]</b>	1	2	3	4	5	6	7	8
	4 <sup>th</sup>	8 <sup>th</sup>	2 <sup>nd</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	3 <sup>rd</sup>	1 <sup>st</sup>

```
for i = 1 to n
    inverse[pref[i]] = i
```

# Understanding the solution

---

For a given problem instance, there may be several stable matchings.

- Do all executions of Gale–Shapley yield the same stable matching?
- If so, which one?

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

an instance with two stable matchings:  $S = \{ A-X, B-Y, C-Z \}$  and  $S' = \{ A-Y, B-X, C-Z \}$

## Understanding the solution

---

**Def.** Student  $s$  is a **valid partner** for hospital  $h$  if there exists any stable matching in which  $h$  and  $s$  are matched.

**Ex.**

- Both Xavier and Yolanda are valid partners for Atlanta.
- Both Xavier and Yolanda are valid partners for Boston.
- Zeus is the only valid partner for Chicago.

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

an instance with two stable matchings:  $S = \{ A-X, B-Y, C-Z \}$  and  $S' = \{ A-Y, B-X, C-Z \}$

## Understanding the solution

---

**Def.** Student  $s$  is a **valid partner** for hospital  $h$  if there exists any stable matching in which  $h$  and  $s$  are matched.

**Hospital-optimal assignment.** Each hospital receives best valid partner.

- Is it perfect?
- Is it stable?

**Claim.** All executions of Gale–Shapley yield **hospital-optimal** assignment.

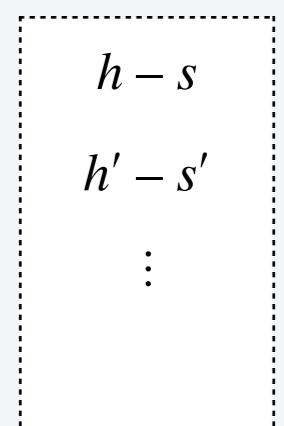
**Corollary.** Hospital-optimal assignment is a stable matching!

# Hospital optimality

**Claim.** Gale–Shapley matching  $S^*$  is hospital-optimal.

**Pf.** [by contradiction]

- Suppose a hospital is matched with student other than best valid partner.
- Hospitals propose in decreasing order of preference  
     $\Rightarrow$  some hospital is rejected by valid partner during Gale–Shapley.
- Let  $h$  be first such hospital, and let  $s$  be the first valid student that rejects  $h$ .
- Let  $M$  be a stable matching where  $h$  and  $s$  are matched.
- When  $s$  rejects  $h$  in Gale–Shapley,  $s$  forms (or re-affirms) commitment to a hospital, say  $h'$ .  
 $\Rightarrow$   $s$  prefers  $h'$  to  $h$ .
- Let  $s'$  be partner of  $h'$  in  $M$ .
- $h'$  had not been rejected by any valid partner (including  $s'$ ) at the point when  $h$  is rejected by  $s$ . ← because this is the first rejection by a valid partner
- Thus,  $h'$  had not yet proposed to  $s'$  when  $h'$  proposed to  $s$ .  
 $\Rightarrow$   $h'$  prefers  $s$  to  $s'$ .
- Thus  $h - s'$  is unstable in  $S$ , a contradiction. ▀



stable matching  $M$

# Student pessimality

---

Q. Does hospital-optimality come at the expense of the students?

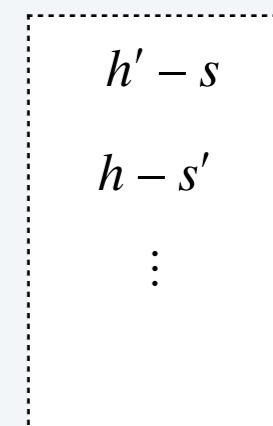
A. Yes.

**Student-pessimal assignment.** Each student receives worst valid partner.

**Claim.** Gale–Shapley finds **student-pessimal** stable matching  $M^*$ .

Pf. [by contradiction]

- Suppose  $h-s$  matched in  $M^*$  but  $h$  is not the worst valid partner for  $s$ .
- There exists stable matching  $M$  in which  $s$  is paired with a hospital, say  $h'$ , whom  $s$  prefers less than  $h$ .  
     $\Rightarrow s$  prefers  $h$  to  $h'$ .
- Let  $s'$  be the partner of  $h$  in  $M$ . By hospital-optimality,  $s$  is the best valid partner for  $h$ .  
     $\Rightarrow h$  prefers  $s$  to  $s'$ .
- Thus,  $h-s$  is an unstable pair in  $M$ , a contradiction. ▀



# Deceit: Machiavelli meets Gale-Shapley

Q. Can there be an incentive to misrepresent your preference list?

- Assume you know hospital's propose-and-reject algorithm will be run.
- Assume preference lists of all other participants are known.

Fact. No, for any hospital; yes, for some students.

**hospitals' preference lists**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>
A	X	Y	Z
B	Y	X	Z
C	X	Y	Z

**students' preference lists**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>
X	B	A	C
Y	A	B	C
Z	A	B	C

**X lies**

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>
X	B	(C)	(A)
Y	A	B	C
Z	A	B	C

## Extensions

---

Extension 1. Some participants declare others as unacceptable.

Extension 2. Some hospitals have more than one position.

Extension 3. Unequal number of positions and students.

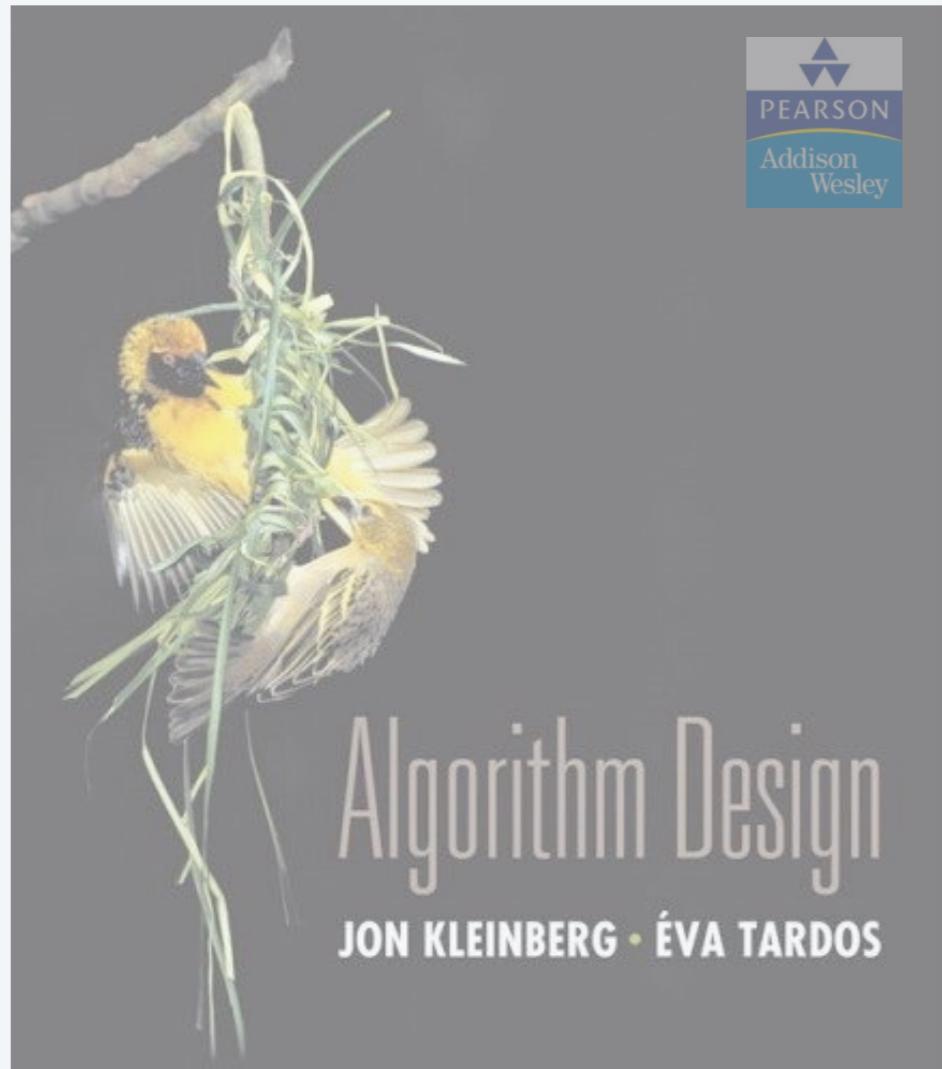
more than 43K med-school  
students; only 31K positions

med-school student  
unwilling to work  
in Cleveland

Def. Matching  $M$  is **unstable** if there is a hospital  $h$  and student  $s$  such that:

- $h$  and  $s$  are acceptable to each other; and
- Either  $s$  is unmatched, or  $s$  prefers  $h$  to assigned hospital; and
- Either  $h$  does not have all its places filled, or  $h$  prefers  $s$  to at least one of its assigned students.

- ISOLATE UNDERLYING STRUCTURE OF PROBLEM.
- CREATE USEFUL AND EFFICIENT ALGORITHMS.



## 1. REPRESENTATIVE PROBLEMS

---

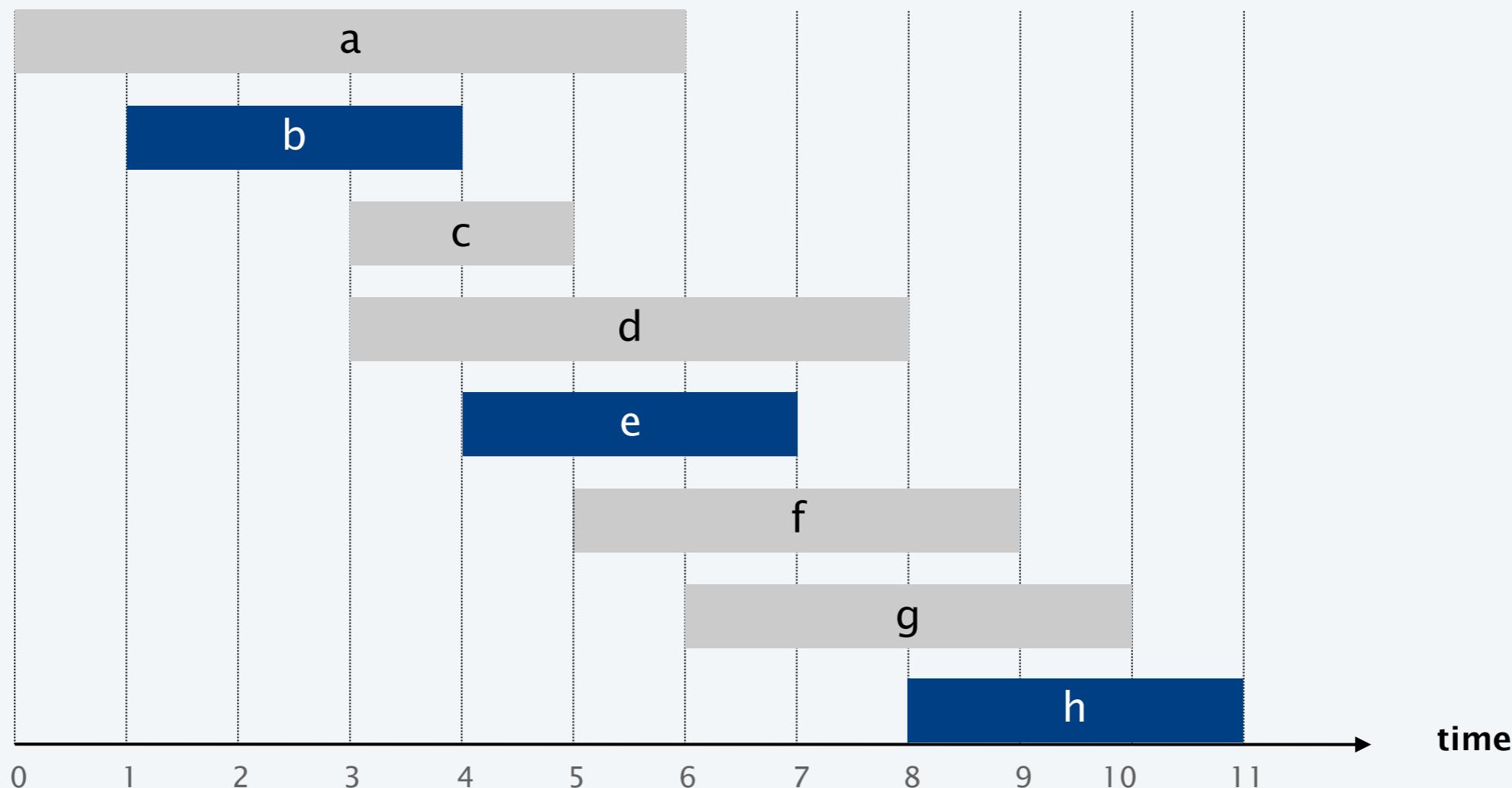
- ▶ *stable matching*
- ▶ *five representative problems*

# Interval scheduling

**Input.** Set of jobs with start times and finish times.

**Goal.** Find maximum cardinality subset of mutually **compatible** jobs.

jobs don't overlap

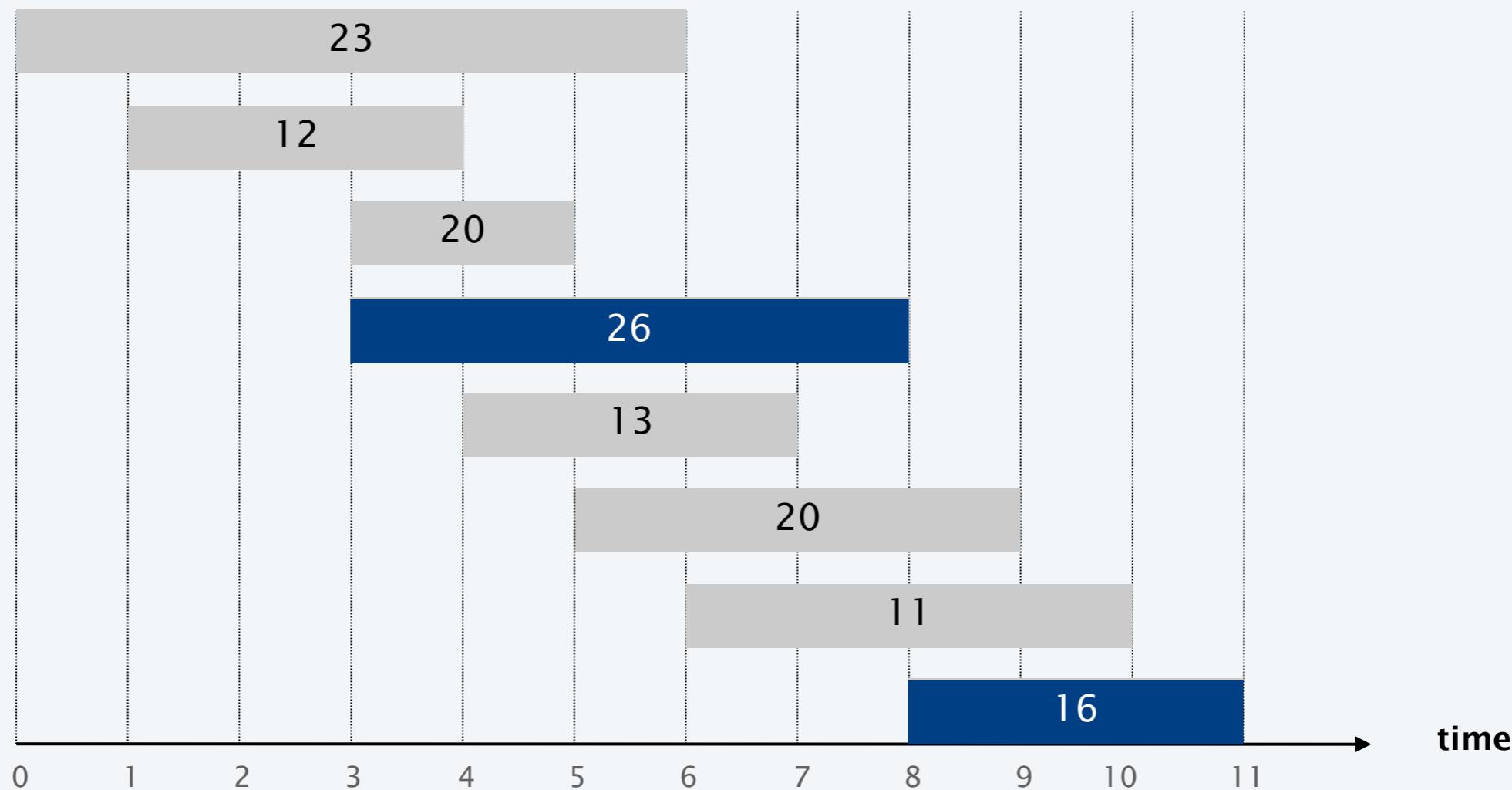


# Weighted interval scheduling

---

**Input.** Set of jobs with start times, finish times, and weights.

**Goal.** Find **maximum weight** subset of mutually compatible jobs.

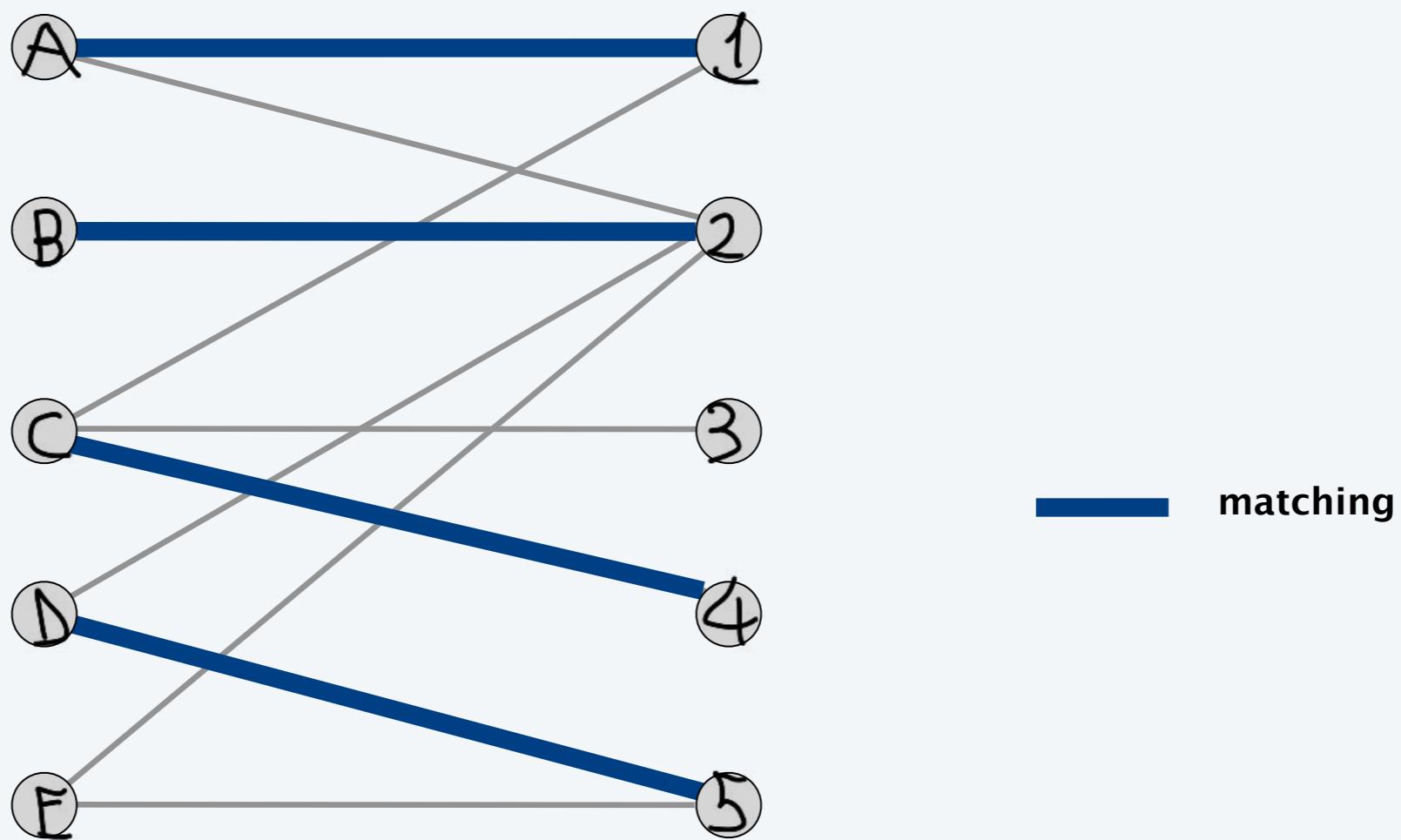


# Bipartite matching

---

**Problem.** Given a bipartite graph  $G = (L \cup R, E)$ , find a max cardinality matching.

**Def.** A subset of edges  $M \subseteq E$  is a **matching** if each node appears in exactly one edge in  $M$ .

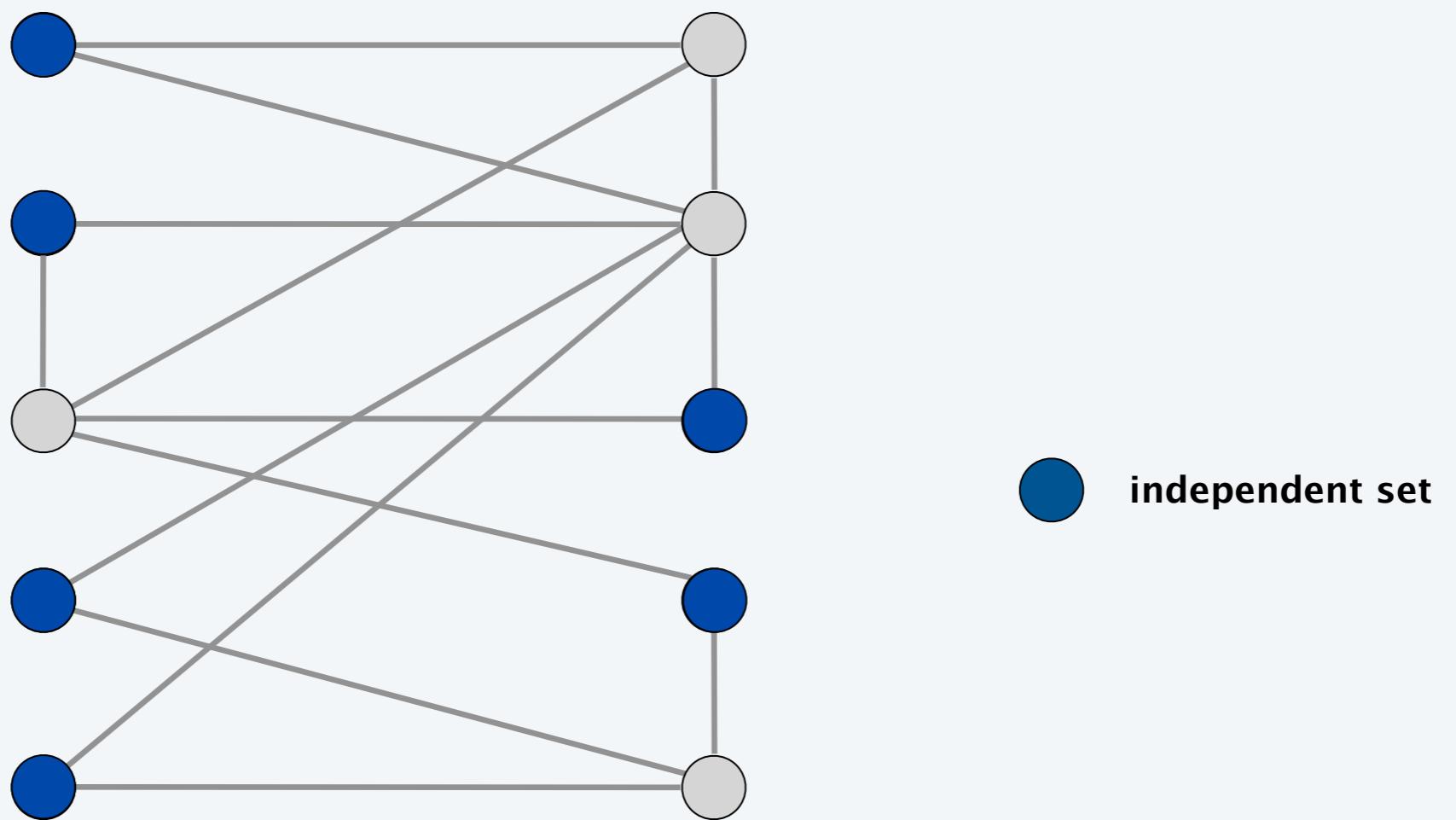


## Independent set

---

Problem. Given a graph  $G = (V, E)$ , find a max cardinality independent set.

Def. A subset  $S \subseteq V$  is **independent** if for every  $(u, v) \in E$ , either  $u \notin S$  or  $v \notin S$  (or both). **SUBSET OF NODES SUCH THAT NO TWO JOINED BY AN EDGE.**



# Competitive facility location

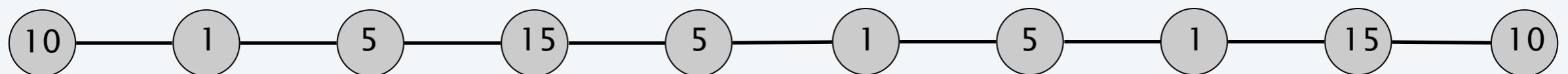
---

**Input.** Graph with weight on each node.

**Game.** Two competing players alternate in selecting nodes.

Not allowed to select a node if any of its neighbors have been selected.

**Goal.** Select a **maximum weight** subset of nodes.



Second player can guarantee 20, but not 25.

## Five representative problems

---

Variations on a theme: independent set.

Interval scheduling:  $O(n \log n)$  greedy algorithm.

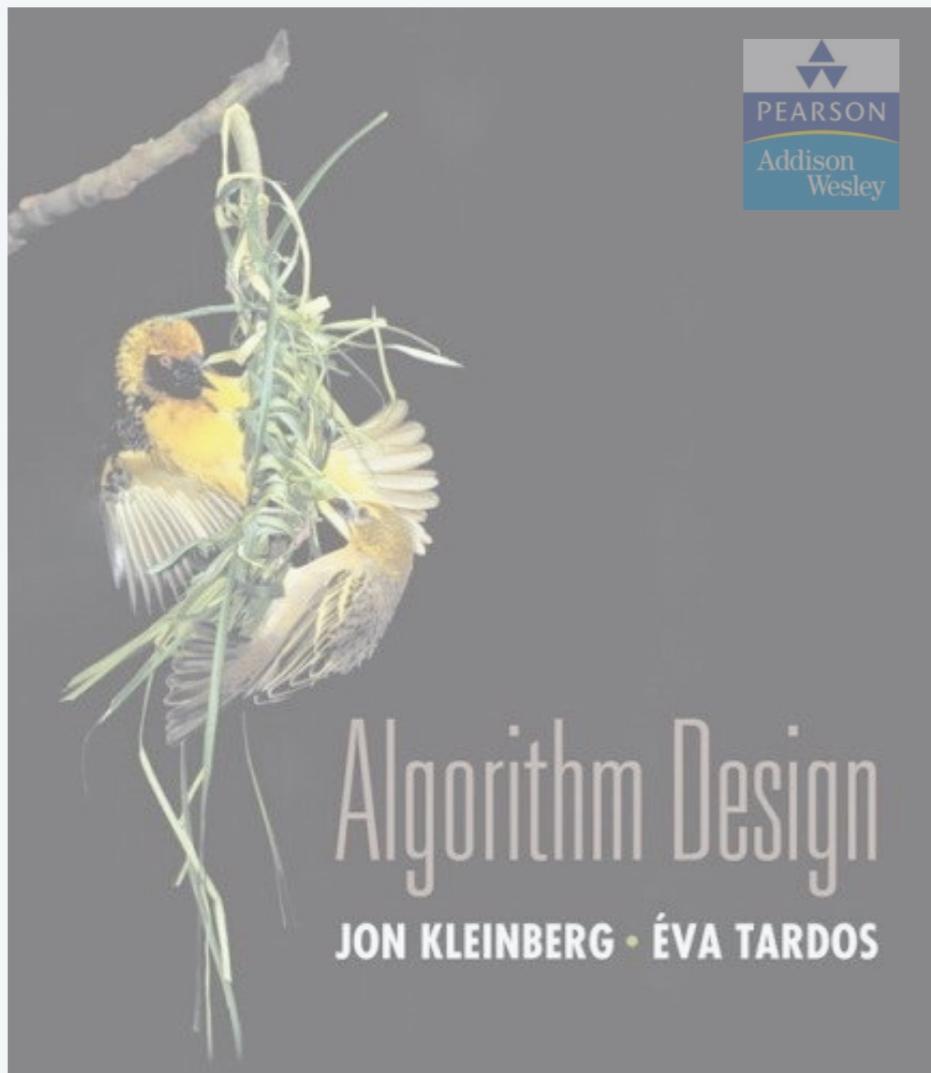
Weighted interval scheduling:  $O(n \log n)$  dynamic programming algorithm.

Bipartite matching:  $O(n^k)$  max-flow based algorithm.

Independent set: **NP**-complete.

Competitive facility location: **PSPACE**-complete.

fot analyse algorithm we will use asymptotic analyse and worst case analyse



## 2. ALGORITHM ANALYSIS

---

- ▶ *computational tractability*
- ▶ *asymptotic order of growth*
- ▶ *survey of common running times*

**Brute force** → enumerate all possible solution

---

**Brute force.** For many nontrivial problems, there is a natural brute-force search algorithm that checks every possible solution.

- Typically takes  $2^n$  time or worse for inputs of size  $n$ .
- Unacceptable in practice.



## Polynomial running time

Desirable scaling property. When the input size doubles, the algorithm should slow down by at most some constant factor  $C$ .



Def. An algorithm is poly-time if the above scaling property holds.

There exist constants  $c > 0$  and  $d > 0$  such that,  
for every input of size  $n$ , the running time of the algorithm  
is bounded above by  $c n^d$  primitive computational steps.

← choose  $C = 2^d$



von Neumann  
(1953)



Nash  
(1955)



Gödel  
(1956)



Cobham  
(1964)



Edmonds  
(1965)



Rabin  
(1966)

# Polynomial running time

We say that an algorithm is **efficient** if it has a polynomial running time.

**Justification.** It really works in practice! especially if the input size is large

- In practice, the poly-time algorithms that people develop have low constants and low exponents. desirable
- Breaking through the exponential barrier of brute force typically exposes some crucial structure of the problem.

**Exceptions.** Some poly-time algorithms do have high constants

and/or exponents, and/or are useless in practice.

Q. Which would you prefer  $20n^{120}$  vs.  $n^{1 + 0.02 \ln n}$  ?

↓  
is polynomial but is a huge  
number! want  $c$  as small

there are some exponential time  
algorithm that in practice work  
very well, because the input  
instances where the algorithm  
is exponential are extremely rare.

for much case is smaller  
than  $n^{100}$

Map graphs in polynomial time

Mikkel Thorup\*

Department of Computer Science, University of Copenhagen  
Universitetsparken 1, DK-2100 Copenhagen East, Denmark  
mthorup@diku.dk

## Abstract

Chen, Grigni, and Papadimitriou (WADS'97 and STOC'98) have introduced a modified notion of planarity, where two faces are considered adjacent if they share at least one point. The corresponding abstract graphs are called map graphs. Chen et.al. raised the question of whether map graphs can be recognized in polynomial time. They showed that the decision problem is in NP and presented a polynomial time algorithm for the special case where we allow at most 4 faces to intersect in any point — if only 3 are allowed to intersect in a point, we get the usual planar graphs.

Chen et.al. conjectured that map graphs can be recognized in polynomial time, and in this paper, their conjecture is settled affirmatively.

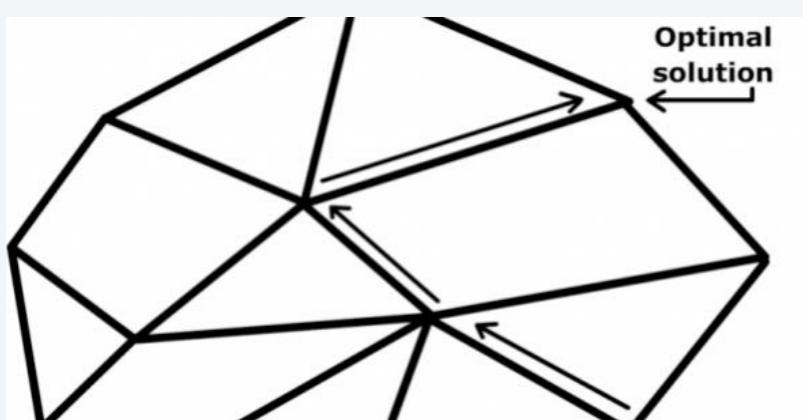
# Worst-case analysis

---

Worst case. Running time guarantee for any input of size  $n$ .

- Generally captures efficiency in practice.
- Draconian view, but hard to find effective alternative.

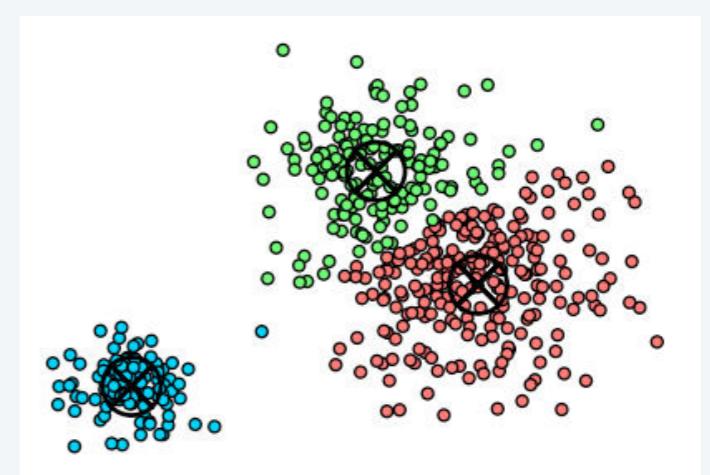
Exceptions. Some exponential-time algorithms are used widely in practice because the worst-case instances seem to be rare.



simplex algorithm



Linux grep  
matching algorithm



k-means algorithm

## Types of analyses

---

Worst case. Running time guarantee for any input of size  $n$ .

Ex. Heapsort requires at most  $2n \log_2 n$  compares to sort  $n$  elements.

Probabilistic. Expected running time of a randomized algorithm.

Ex. The expected number of compares to quicksort  $n$  elements is  $\sim 2n \ln n$ .

Amortized. Worst-case running time for any sequence of  $n$  operations.

Ex. Starting from an empty stack, any sequence of  $n$  push and pop operations takes  $O(n)$  primitive computational steps using a resizing array.

→ some input where algorithm work poorly are very rare in practice

Average-case. Expected running time for a random input of size  $n$ .

Ex. The expected number of character compares performed by 3-way radix quicksort on  $n$  uniformly random strings is  $\sim 2n \ln n$ .

Also. Smoothed analysis, competitive analysis, ...

# Why it matters

**Table 2.1** The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds  $10^{25}$  years, we simply record the algorithm as taking a very long time.

	$n$	$n \log_2 n$	$n^2$	$n^3$	$1.5^n$	$2^n$	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	$10^{25}$ years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	$10^{17}$ years	very long
$n = 1,000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10,000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100,000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1,000,000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

WE WANT POLYNOMIAL TIME, SECONDS!

EXPONENTIAL TIME IS BAD

asymptotic analyses focus on  
giving rough estimation of  
the running time of the



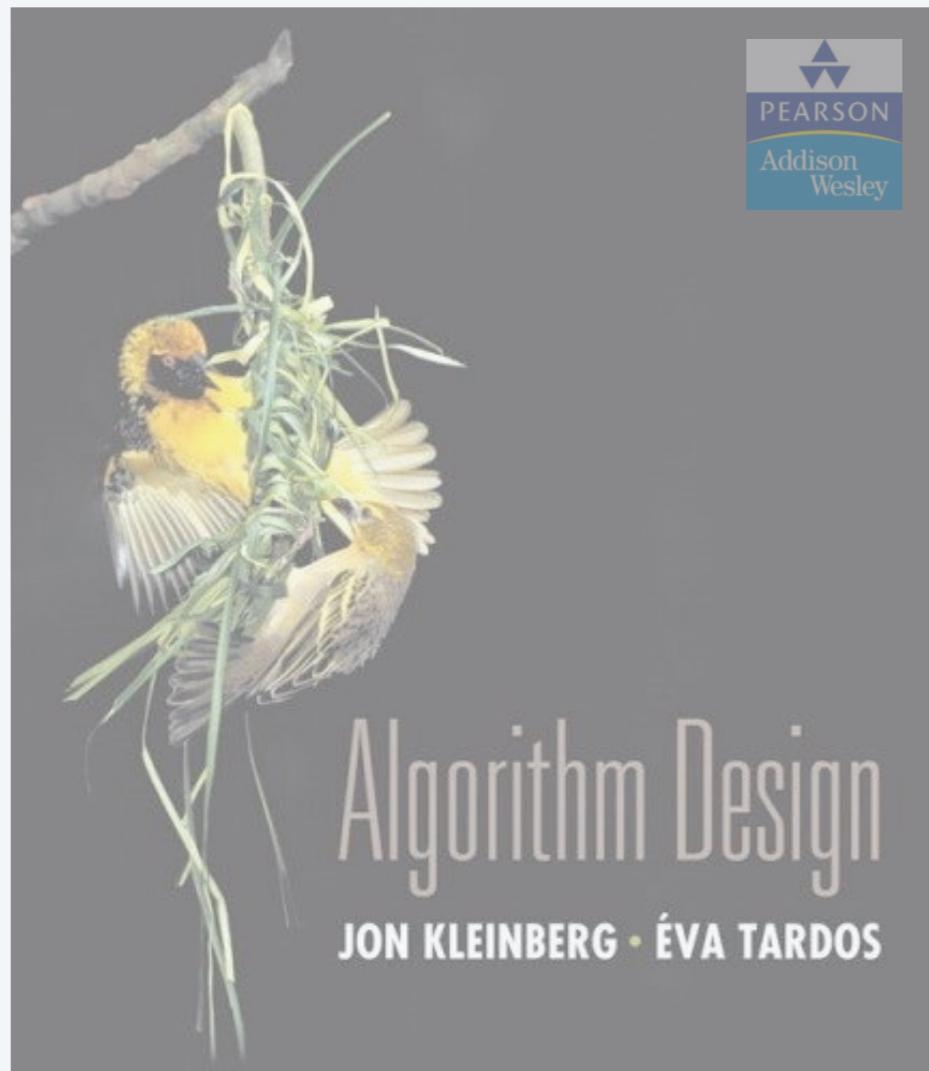
## 2. ALGORITHM ANALYSIS algorithm

- ▶ computational tractability
- ▶ asymptotic order of growth
- ▶ survey of common running times

define family of  
function that describe  
running time of  
the algorithm

We want to assess the running time of an algorithm only looking to some few important operation (dominant operation).

And characterized the running time of the algorithm for value of the input size  
that are big > for small value running time we assume is small

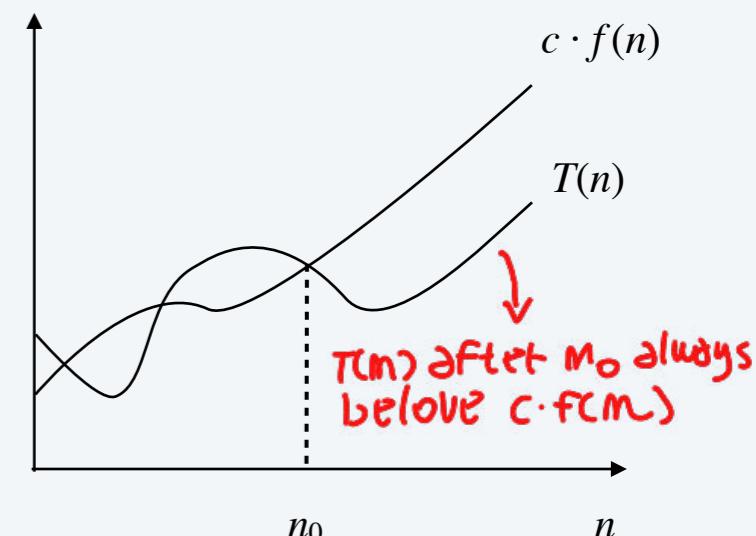


**Big-Oh notation**: used to define upper bounds on running time of an algorithm

Upper bounds.  $T(n)$  is  $O(f(n))$  if there exist constants  $c > 0$  and  $n_0 \geq 0$  such that  $T(n) \leq c \cdot f(n)$  for all  $n \geq n_0$ .

Ex.  $T(n) = 32n^2 + 17n + 1$ .

- $T(n)$  is  $O(n^2)$ . ← choose  $c = 50, n_0 = 1$
- $T(n)$  is also  $O(n^3)$ .
- $T(n)$  is neither  $O(n)$  nor  $O(n \log n)$ .



Typical usage. Insertion sort makes  $O(n^2)$  compares to sort  $n$  elements.

Alternate definition.  $T(n)$  is  $O(f(n))$  if  $\limsup_{n \rightarrow \infty} \frac{T(n)}{f(n)} < \infty$ .

The number of bits for describe the input in numerical algorithm is  $\log(n)$

## Notational abuses

$n$  often is the number of input elements, not always

**Equals sign.**  $O(f(n))$  is a set of functions, but computer scientists often write  $T(n) = O(f(n))$  instead of  $T(n) \in O(f(n))$ .

input size not always  $n$ !  
↑

**Ex.** Consider  $f(n) = 5n^3$  and  $g(n) = 3n^2$ .

- We have  $f(n) = O(n^3) = g(n)$ .
- Thus,  $f(n) = g(n)$ .  $\times$  **False !!**

the number of bits for  
representative number  $n$  is  $\log(n)$



**Domain.** The domain of  $f(n)$  is typically the natural numbers  $\{0, 1, 2, \dots\}$ .

- Sometimes we restrict to a subset of the natural numbers.  
Other times we extend to the reals.

**Non-negative functions.** When using big-Oh notation, we assume that the functions involved are (asymptotically) non-negative.

**Bottom line.** OK to abuse notation; not OK to misuse it.

nothing better is possible is very complicated

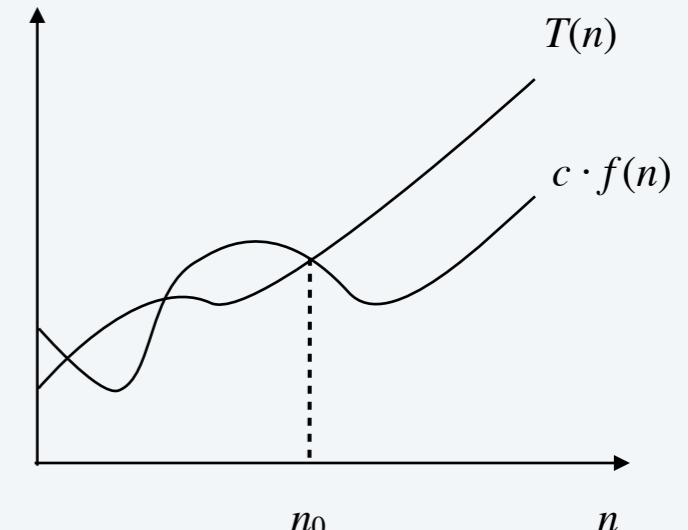
## Big-Omega notation

→ bounding  $T(n)$  from below

Lower bounds.  $T(n)$  is  $\Omega(f(n))$  if there exist constants  $c > 0$  and  $n_0 \geq 0$  such that  $T(n) \geq c \cdot f(n)$  for all  $n \geq n_0$ .

Ex.  $T(n) = 32n^2 + 17n + 1$ .

- $T(n)$  is both  $\Omega(n^2)$  and  $\Omega(n)$ . ← choose  $c = 32, n_0 = 1$
- $T(n)$  is neither  $\Omega(n^3)$  nor  $\Omega(n^3 \log n)$ .



Typical usage. Any compare-based sorting algorithm requires  $\Omega(n \log n)$  compares in the worst case.

Meaningless statement. Any compare-based sorting algorithm requires at least  $O(n \log n)$  compares in the worst case.

## Big-Theta notation

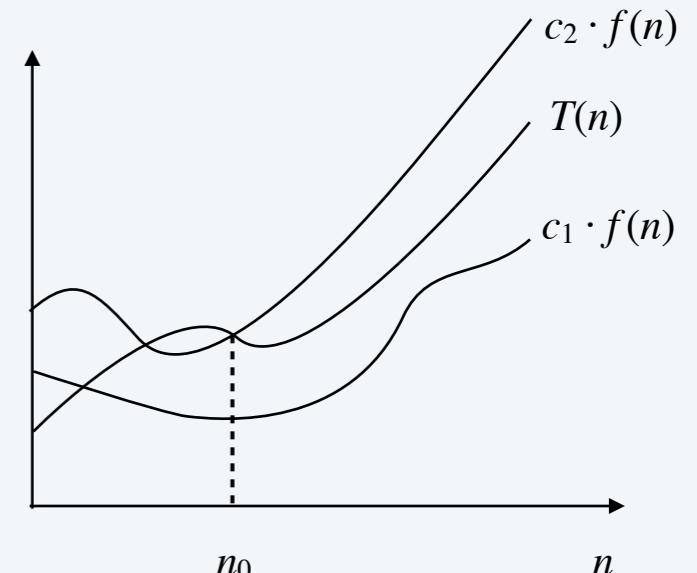
---

**Tight bounds.**  $T(n)$  is  $\Theta(f(n))$  if there exist constants  $c_1 > 0$ ,  $c_2 > 0$ , and  $n_0 \geq 0$  such that  $c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$  for all  $n \geq n_0$ .

---

**Ex.**  $T(n) = 32n^2 + 17n + 1$ .

- $T(n)$  is  $\Theta(n^2)$ . ← choose  $c_1 = 32$ ,  $c_2 = 50$ ,  $n_0 = 1$
- $T(n)$  is neither  $\Theta(n)$  nor  $\Theta(n^3)$ .



**Typical usage.** Mergesort makes  $\Theta(n \log n)$  compares to sort  $n$  elements.

## Useful facts

---

**Proposition.** If  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c > 0$ , then  $f(n)$  is  $\Theta(g(n))$ .

---

**Pf.** By definition of the limit, there exists  $n_0$  such that for all  $n \geq n_0$

$$\frac{1}{2}c \leq \frac{f(n)}{g(n)} \leq 2c$$

- Thus,  $f(n) \leq 2c g(n)$  for all  $n \geq n_0$ , which implies  $f(n)$  is  $O(g(n))$ .
- Similarly,  $f(n) \geq \frac{1}{2}c g(n)$  for all  $n \geq n_0$ , which implies  $f(n)$  is  $\Omega(g(n))$ .

**Proposition.** If  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$ , then  $f(n)$  is  $O(g(n))$ .

---

## Asymptotic bounds for some common functions

---

**Polynomials.** Let  $T(n) = a_0 + a_1 n + \dots + a_d n^d$  with  $a_d > 0$ . Then,  $T(n)$  is  $\Theta(n^d)$ .

---

Pf. 
$$\lim_{n \rightarrow \infty} \frac{a_0 + a_1 n + \dots + a_d n^d}{n^d} = a_d > 0$$

**Logarithms.**  $\Theta(\log_a n)$  is  $\Theta(\log_b n)$  for any constants  $a, b > 0$ . ← no need to specify base  
(assuming it is a constant)

---

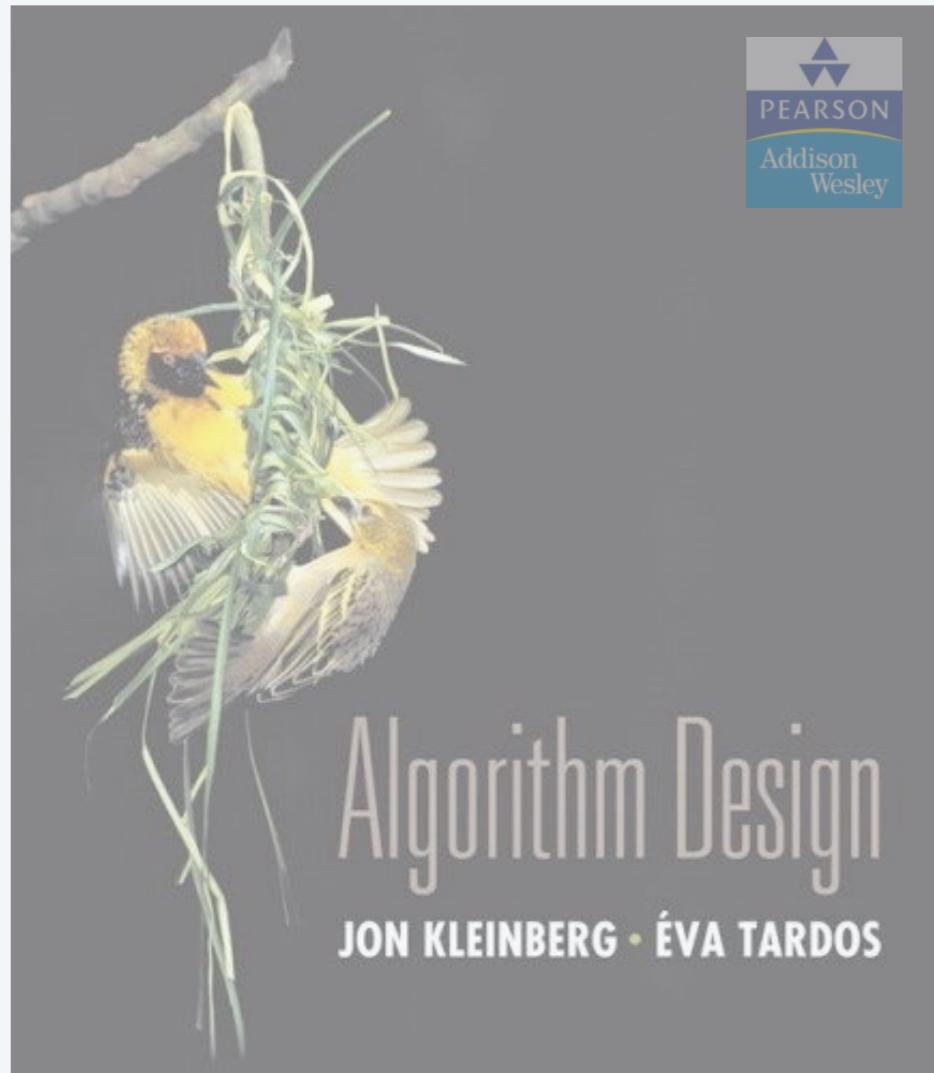
**Logarithms and polynomials.** For every  $d > 0$ ,  $\log n$  is  $O(n^d)$ .

---

**Exponentials and polynomials.** For every  $r > 1$  and every  $d > 0$ ,  $n^d$  is  $O(r^n)$ .

---

Pf. 
$$\lim_{n \rightarrow \infty} \frac{n^d}{r^n} = 0$$
       $m^d \in O(r^m)$



## 2. ALGORITHM ANALYSIS

---

- ▶ *computational tractability*
- ▶ *asymptotic order of growth*
- ▶ *survey of common running times*

## Linear time: $O(n)$

---

Linear time. Running time is proportional to input size.

Computing the maximum. Compute maximum of  $n$  numbers  $a_1, \dots, a_n$ .

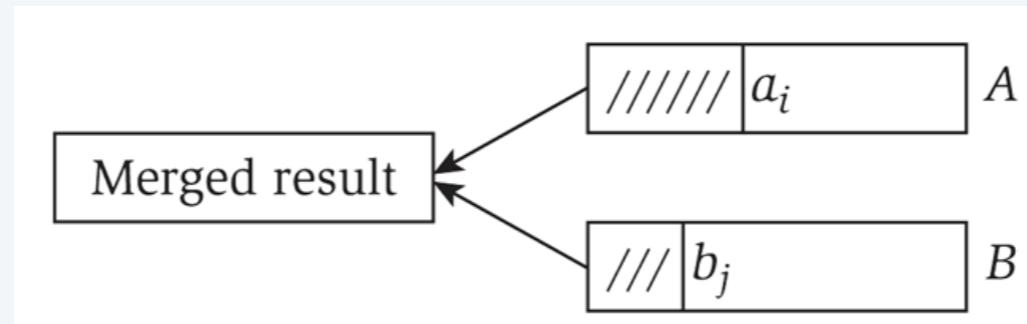
```
max ← a1
for i = 2 to n {
    if (ai > max)
        max ← ai
}
```

~> dominant operation, mtime  
the for cycle

## Linear time: $O(n)$

---

Merge. Combine two sorted lists  $A = a_1, a_2, \dots, a_n$  with  $B = b_1, b_2, \dots, b_n$  into sorted whole.



```
i = 1, j = 1
while (both lists are nonempty) {
    if ( $a_i \leq b_j$ ) append  $a_i$  to output list and increment i
    else                append  $b_j$  to output list and increment j
}
append remainder of nonempty list to output list
```

↑ worst case in time the  
while cycle

Claim. Merging two lists of size  $n$  takes  $O(n)$  time.

Pf. After each compare, the length of output list increases by 1.

## Linearithmic time: $O(n \log n)$

---

$O(n \log n)$  time. Arises in divide-and-conquer algorithms.

Sorting. Mergesort and heapsort are sorting algorithms that perform  $O(n \log n)$  compares.

Largest empty interval. Given  $n$  time-stamps  $x_1, \dots, x_n$  on which copies of a file arrive at a server, what is largest interval when no copies of file arrive?

$O(n \log n)$  solution. Sort the time-stamps. Scan the sorted list in order, identifying the maximum gap between successive time-stamps.

## Quadratic time: $O(n^2)$

---

Ex. Enumerate all pairs of elements.

Closest pair of points. Given a list of  $n$  points in the plane  $(x_1, y_1), \dots, (x_n, y_n)$ , find the pair that is closest to each other.

$O(n^2)$  solution. Try all pairs of points.

```
min ← (x1 - x2)2 + (y1 - y2)2
for i = 1 to n {
    for j = i+1 to n { ~> two for cycle done each n time
        d ← (xi - xj)2 + (yi - yj)2
        if (d < min)
            min ← d
    }
}
```

Remark.  $\Omega(n^2)$  seems inevitable, but this is just an illusion. [see Chapter 5]

## Cubic time: $O(n^3)$

---

**Ex.** Enumerate all triples of elements.

**Set disjointness.** Given  $n$  sets  $S_1, \dots, S_n$  each of which is a subset of  $1, 2, \dots, n$ , is there some pair of these which are disjoint?

**$O(n^3)$  solution.** For each pair of sets, determine if they are disjoint.

```
foreach set  $S_i$  {  
    foreach other set  $S_j$  {  
        foreach element  $p$  of  $S_i$  {  
            determine whether  $p$  also belongs to  $S_j$   
        }  
        if (no element of  $S_i$  belongs to  $S_j$ )  
            report that  $S_i$  and  $S_j$  are disjoint  
    }  
}
```

## Polynomial time: $O(n^k)$

---

Independent set of size  $k$ . Given a graph, are there  $k$  nodes such that no two are joined by an edge?

$\nearrow$   
k is a constant

$O(n^k)$  solution. Enumerate all subsets of  $k$  nodes.

```
foreach subset S of k nodes {  
    check whether S is an independent set  
    if (S is an independent set)  
        report S is an independent set  
    }  
}
```

- Check whether  $S$  is an independent set takes  $O(k^2)$  time.

- Number of  $k$ -element subsets =  $\binom{n}{k} = \frac{n(n-1)(n-2) \times \cdots \times (n-k+1)}{k(k-1)(k-2) \times \cdots \times 1} \leq \frac{n^k}{k!}$

$\nearrow$   
poly-time for  $k=17$ ,  
but not practical

## Exponential time

---

Independent set. Given a graph, what is maximum cardinality of an independent set?

$O(n^2 2^n)$  solution. Enumerate all subsets.

```
S* ← φ
foreach subset S of nodes {
    check whether S is an independent set
    if (S is largest independent set seen so far)
        update S* ← S
}
}
```

## Sublinear time

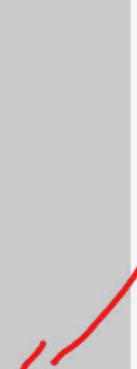
---

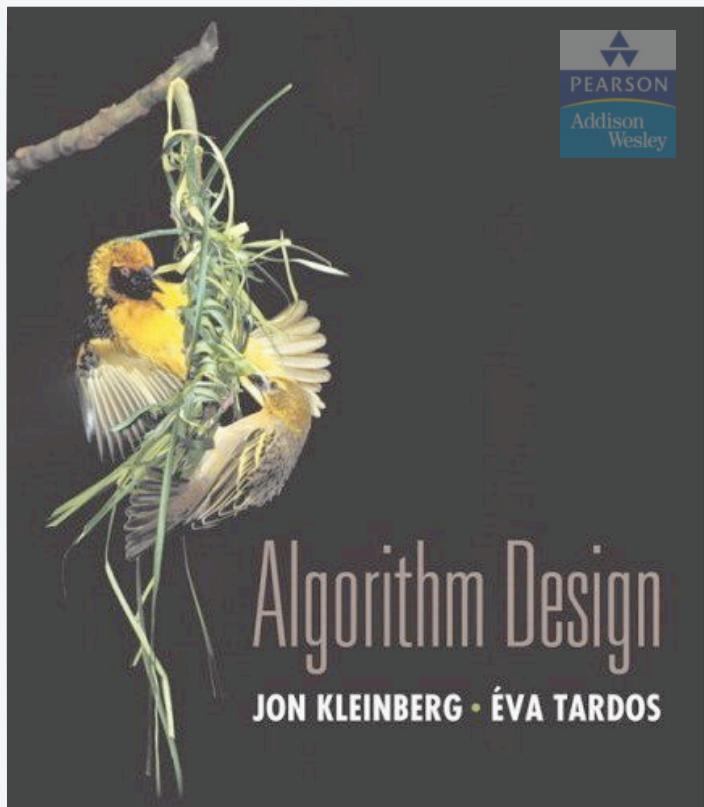
Search in a sorted array. Given a sorted array  $A$  of  $n$  numbers, is a given number  $x$  in the array?

$O(\log n)$  solution. Binary search.

```
lo ← 1, hi ← n
while (lo ≤ hi) {
    mid ← (lo + hi) / 2
    if      (x < A[mid]) hi ← mid - 1
    else if (x > A[mid]) lo ← mid + 1
    else return yes
}
return no
```

divide and conquer





## SECTION 4.1

definition:

- is the simplest possible way of thinking for algorithm design.  
build out solution with a sequence of step and each step we only care about choosing an item that optimize some measure of quality.

## 4. GREEDY ALGORITHMS I

---

- ▶ *coin changing*
- ▶ *interval scheduling*
- ▶ *scheduling to minimize lateness*
- ▶ *optimal caching*

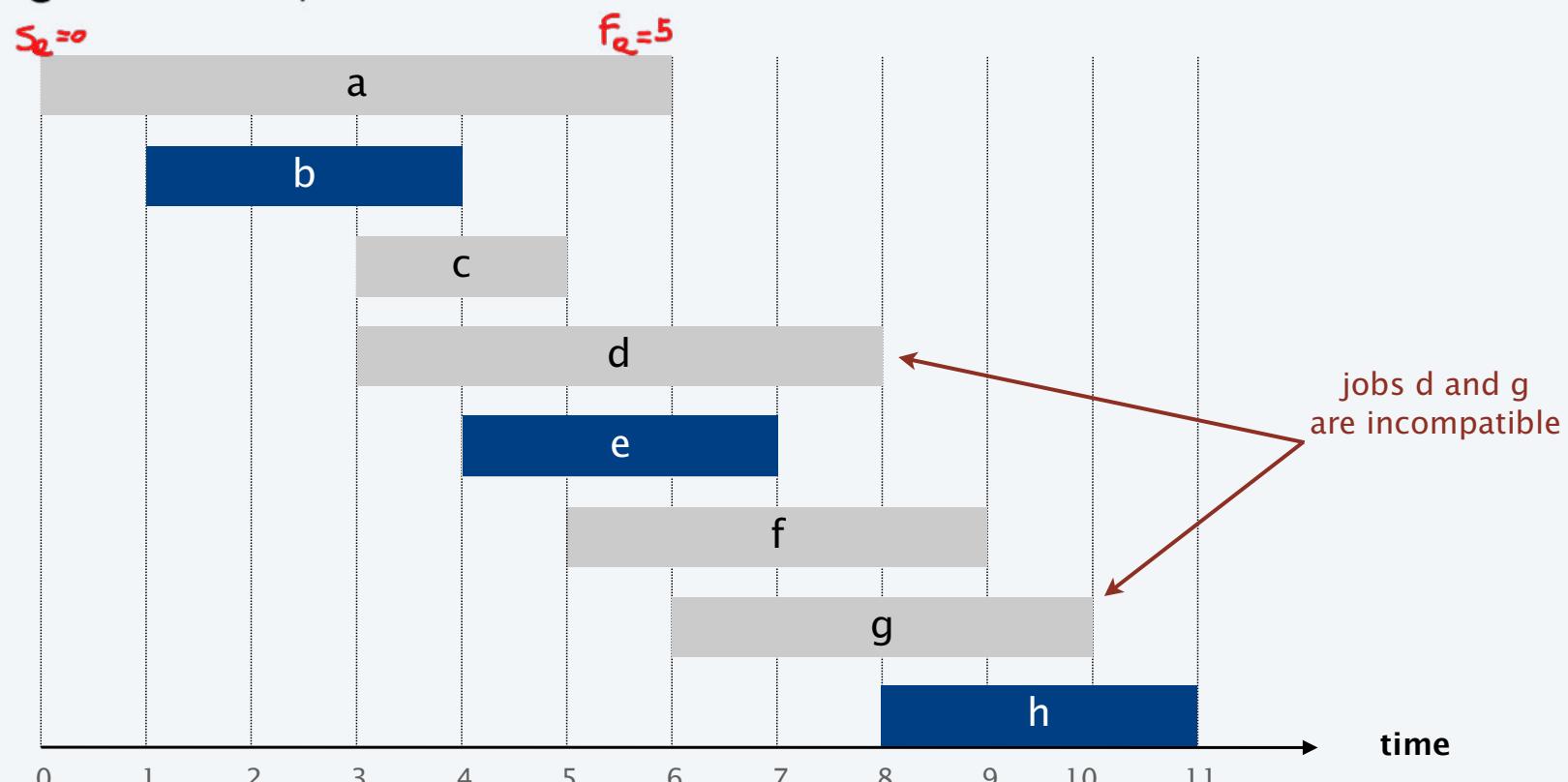
we have a set of possible choices that we can make, there is a measure of quality associated to each possible choice and make that choice that optimize the measure of quality.

in greedy algorithm what we do now will be optimal also in the future, a property that not always is respected in algorithm

# Interval scheduling can be solved optimally by an greedy algorithm

- collection of intervals.
- Job  $j$  starts at  $s_j$  and finishes at  $f_j$ .  $s_j, f_j$  integer values.
- Two jobs **compatible** if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.  
↳ select maximum number of non overlapping intervals on line.

N.B.: this is a special case of maximum independent set on graph. Associate each vertex to an interval and two vertex are connected if the corresponding interval overlap.



## Interval scheduling: greedy algorithms

based on information available that  
is here  $s_i$  and  $f_i$ .

**Greedy template.** Consider jobs in some natural order.

Take each job provided it's compatible with the ones already taken.

- [Earliest start time] Consider jobs in ascending order of  $s_j$ .
- [Earliest finish time] Consider jobs in ascending order of  $f_j$ .
- [Shortest interval] Consider jobs in ascending order of  $f_j - s_j$ .
- [Fewest conflicts] For each job  $j$ , count the number of conflicting jobs  $c_j$ . Schedule in ascending order of  $c_j$ .

# Interval scheduling: greedy algorithms

Greedy template. Consider jobs in some natural order.

Take each job provided it's compatible with the ones already taken.

counterexample for show that is not the optimal strategy

counterexample for earliest start time

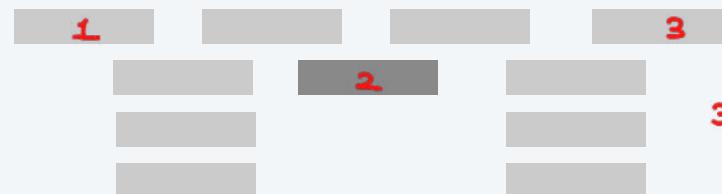


is terrible reject a lot of  
jobs

counterexample for shortest interval



counterexample for fewest conflicts



3 not the optimal solution

## Interval scheduling: earliest-finish-time-first algorithm



EARLIEST-FINISH-TIME-FIRST ( $n, s_1, s_2, \dots, s_n, f_1, f_2, \dots, f_n$ )

SORT jobs by finish time so that  $f_1 \leq f_2 \leq \dots \leq f_n \rightarrow$  Sorting  $O(n \cdot \log n)$

$A \leftarrow \emptyset$  ← set of jobs selected

FOR  $j = 1$  TO  $n$

IF job  $j$  is compatible with  $A \rightarrow s_j \geq f_{j^*}$

$A \leftarrow A \cup \{j\}$

↳ last finish time of a job in  $A$ .

RETURN  $A$

One possible for check optimality, is check algorithm for all possible input, but is impossible, alternative we must demonstrate that not exists a better solution.

Proposition. Can implement earliest-finish-time first in  $O(n \log n)$  time.

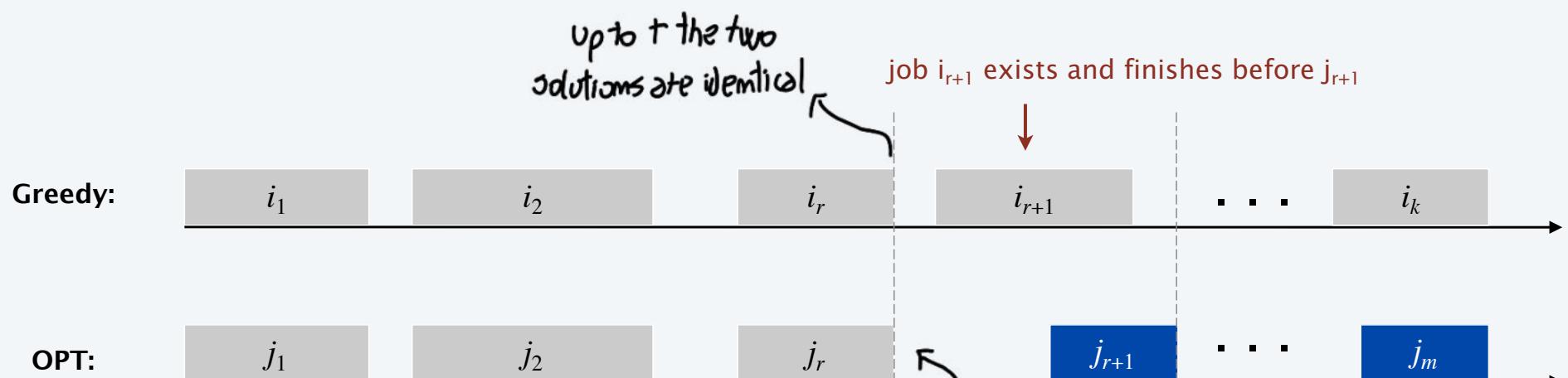
- Keep track of job  $j^*$  that was added last to  $A$ .
- Job  $j$  is compatible with  $A$  iff  $s_j \geq f_{j^*}$ .
- Sorting by finish time takes  $O(n \log n)$  time.

# Interval scheduling: analysis of earliest-finish-time-first algorithm

Theorem. The earliest-finish-time-first algorithm is optimal.

Pf. [by contradiction] and also some induction

- Assume greedy is not optimal, and let's see what happens.
- Let  $i_1, i_2, \dots, i_k$  denote set of jobs selected by greedy.
- Let  $j_1, j_2, \dots, j_m$  denote set of jobs in an optimal solution with  $i_1 = j_1, i_2 = j_2, \dots, i_r = j_r$  for the largest possible value of  $r$ .



if greedy not optimal, there will be a first time that greedy deviate from optimal when greedy slope  $t+1$  interval that is first time that greedy deviate from optimal, but up to  $t$  greedy and optimal are exact equal.

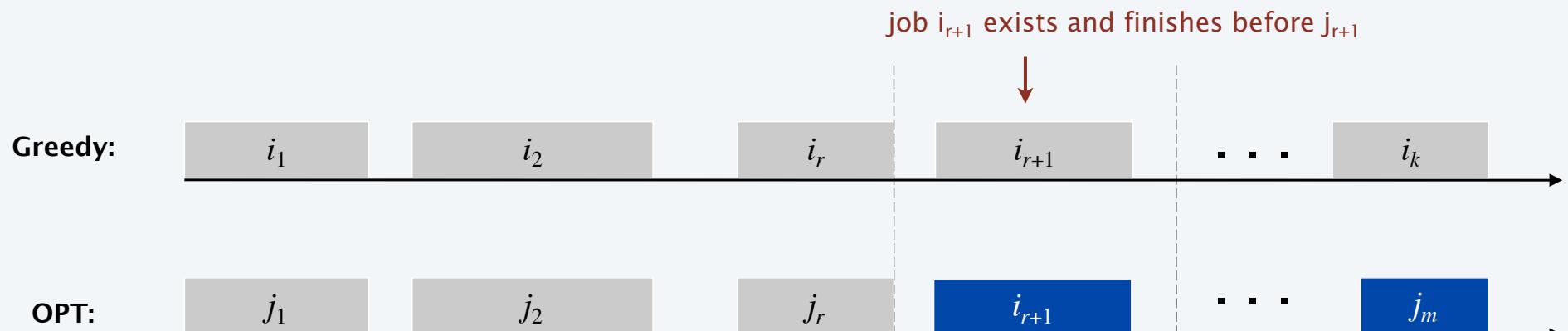
if greedy deviate from optimal, we can align them without losing optimality, greedy in next interval,  $t+1$ , select a finish time that is earlier than finish time of opt., but we know that no interval is selected after this point, so the opt. solution can be replaced with  $i_{r+1}$

# Interval scheduling: analysis of earliest-finish-time-first algorithm

Theorem. The earliest-finish-time-first algorithm is optimal.

Pf. [by contradiction] we change optimal solution with greedy solution without loose optimality

- Assume greedy is not optimal, and let's see what happens.
- Let  $i_1, i_2, \dots, i_k$  denote set of jobs selected by greedy.
- Let  $j_1, j_2, \dots, j_m$  denote set of jobs in an optimal solution with  $i_1 = j_1, i_2 = j_2, \dots, i_r = j_r$  for the largest possible value of  $r$ .



consider the first interval where the two solutions deviate, from before we know  
that exist some finishing time of the previous interval and one of the  
next. opt next finishing time is after the next finish time in greedy, we can anticipate  
interval in opt solution.

solution still feasible and optimal  
(but contradicts maximality of r)

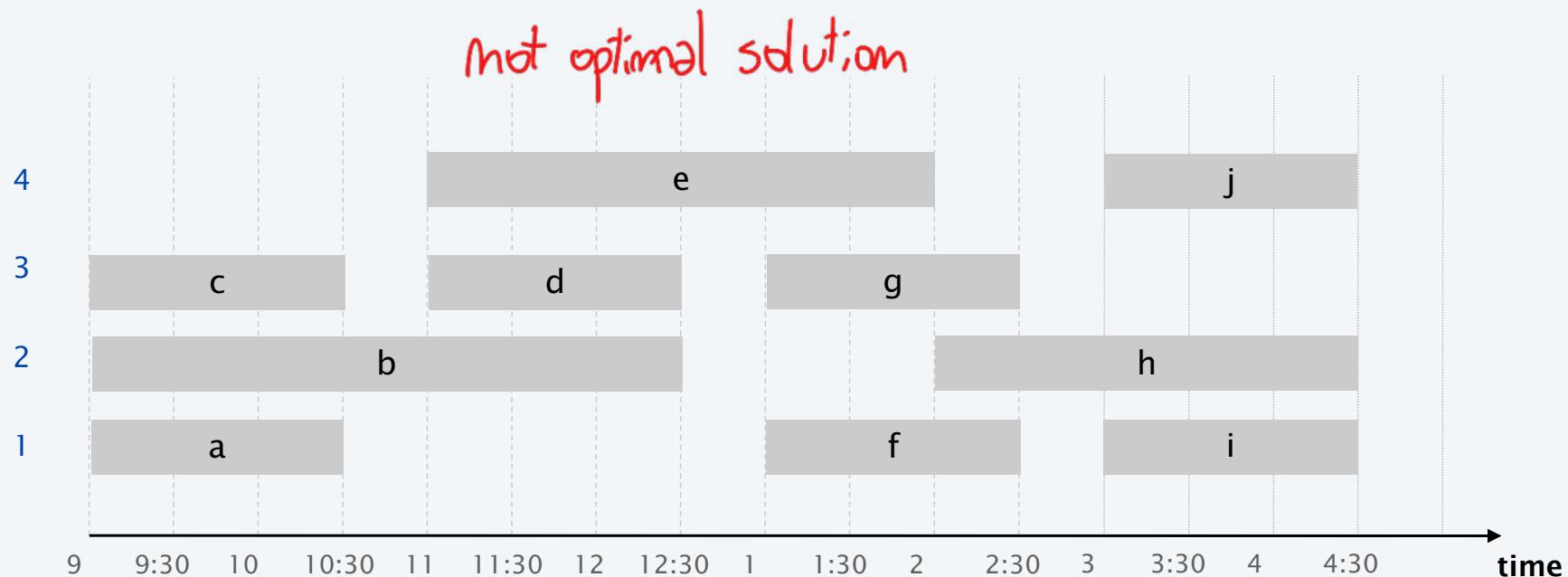
**Interval partitioning**, version of graph coloring problem, assign color to each vertex such that no two adjacent vertex have the same color

## Interval partitioning.

- Lecture  $j$  starts at  $s_j$  and finishes at  $f_j$ .
- Goal: find minimum number of classrooms to schedule all lectures so that no two lectures occur at the same time in the same room.

Without overlapping

Ex. This schedule uses 4 classrooms to schedule 10 lectures.

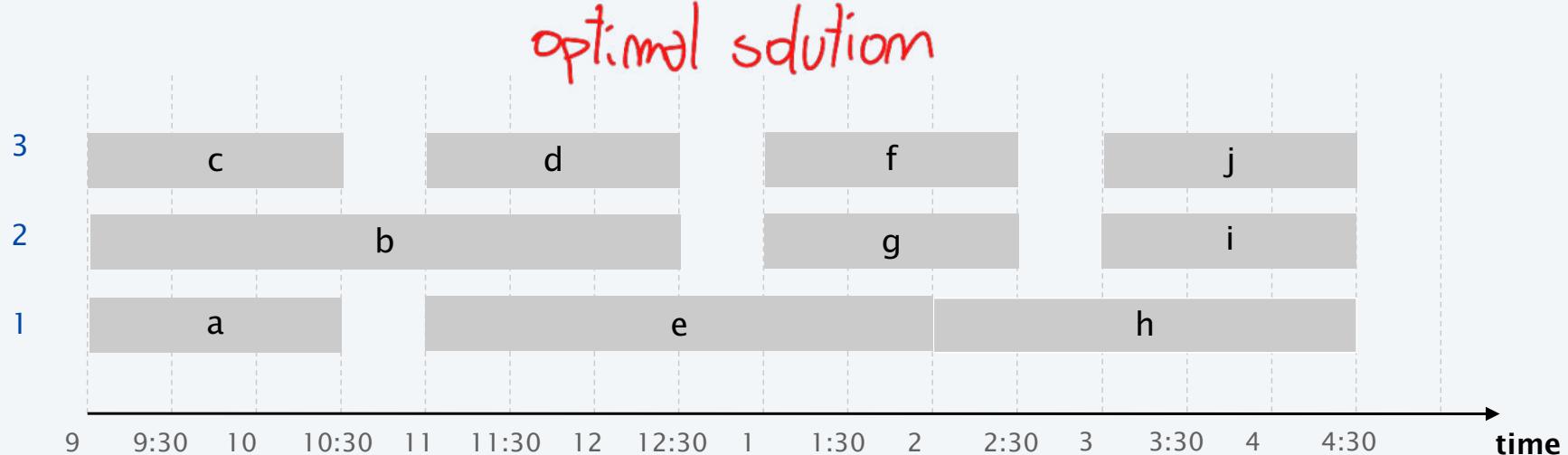


# Interval partitioning

## Interval partitioning.

- Lecture  $j$  starts at  $s_j$  and finishes at  $f_j$ .
- Goal: find minimum number of classrooms to schedule all lectures so that no two lectures occur at the same time in the same room.

Ex. This schedule uses 3 classrooms to schedule 10 lectures.



## Interval partitioning: greedy algorithms

---

**Greedy template.** Consider lectures in some natural order.

Assign each lecture to an available classroom (which one?);  
allocate a new classroom if none are available.

- [Earliest start time] Consider lectures in ascending order of  $s_j$ .
- [Earliest finish time] Consider lectures in ascending order of  $f_j$ .
- [Shortest interval] Consider lectures in ascending order of  $f_j - s_j$ .
- [Fewest conflicts] For each lecture  $j$ , count the number of conflicting lectures  $c_j$ . Schedule in ascending order of  $c_j$ .

## Interval partitioning: greedy algorithms

Greedy template. Consider lectures in some natural order.  
Assign each lecture to an available classroom (which one?);  
allocate a new classroom if none are available.

counterexample for earliest finish time



counterexample for shortest interval



counterexample for fewest conflicts



## Interval partitioning: earliest-start-time-first algorithm



EARLIEST-START-TIME-FIRST ( $n, s_1, s_2, \dots, s_n, f_1, f_2, \dots, f_n$ )

SORT lectures by start time so that  $s_1 \leq s_2 \leq \dots \leq s_n$ .  $\rightarrow$  sorting  $O(n \cdot \log n)$

$d \leftarrow 0$      $\leftarrow$  number of allocated classrooms

FOR  $j = 1$  TO  $n$

    IF lecture  $j$  is compatible with some classroom

        Schedule lecture  $j$  in any such classroom  $k$ .

    ELSE

        Allocate a new classroom  $d + 1$ .

        Schedule lecture  $j$  in classroom  $d + 1$ .

$d \leftarrow d + 1$

RETURN schedule.

For say that this is an optimal solution, show that any solution can not use a less number of classes, there is a lower bound, and this algorithm achieve this lower bound.

## Interval partitioning: earliest-start-time-first algorithm

---

Proposition. The earliest-start-time-first algorithm can be implemented in  $O(n \log n)$  time.

want finish time of each classroom

Pf. Store classrooms in a priority queue (key = finish time of its last lecture).

- To determine whether lecture  $j$  is compatible with some classroom, compare  $s_j$  to key of min classroom  $k$  in priority queue.  $s_j \geq \text{min } K$
- To add lecture  $j$  to classroom  $k$ , increase key of classroom  $k$  to  $f_j$ .
- Total number of priority queue operations is  $O(n)$ .
- Sorting by start time takes  $O(n \log n)$  time. ▀

We can use heap for implement priority queue.

Remark. This implementation chooses the classroom  $k$  whose finish time of its last lecture is the earliest.

We can not use less classroom than the number of classes that need to be schedule at the same time! (lowerbound)

d: number of classes that algorithm allocate

## Interval partitioning: lower bound on optimal solution

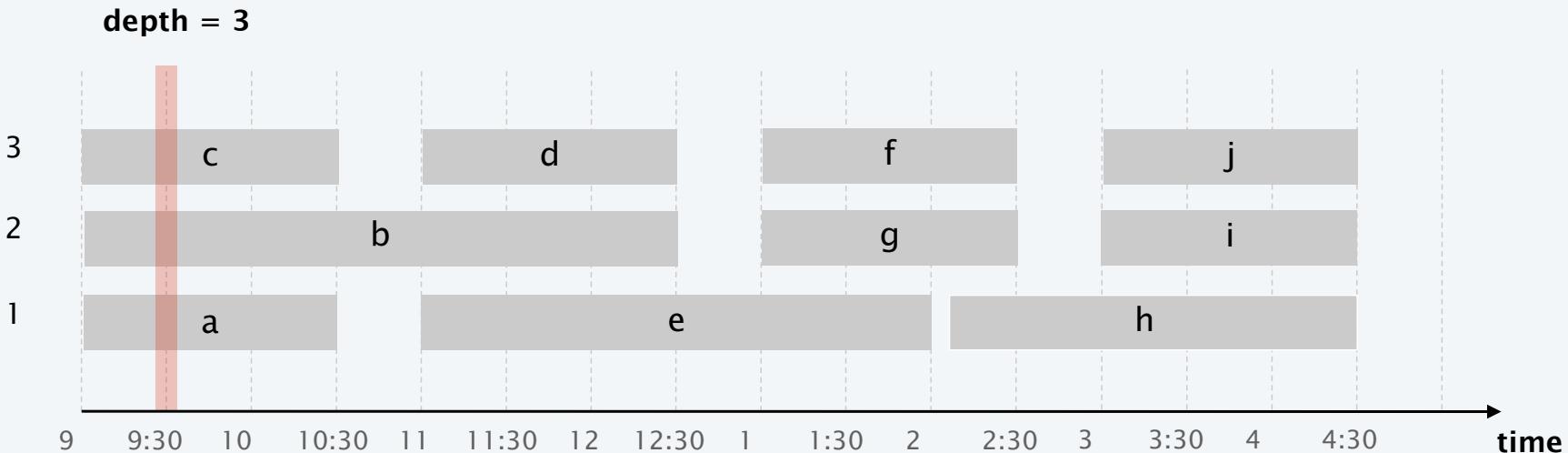
---

Def. The depth of a set of open intervals is the maximum number that contain any given time.

Key observation. Number of classrooms needed  $\geq$  depth.

Q. Does number of classrooms needed always equal depth?

A. Yes! Moreover, earliest-start-time-first algorithm finds one.



## Interval partitioning: analysis of earliest-start-time-first algorithm

---

**Observation.** The earliest-start-time first algorithm never schedules two incompatible lectures in the same classroom.

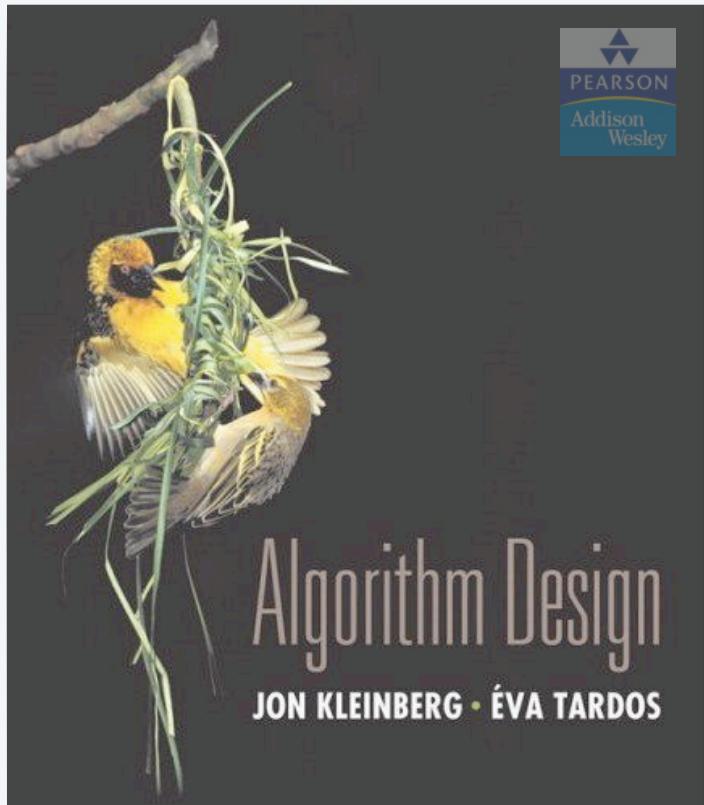
---

**Theorem.** Earliest-start-time-first algorithm is optimal.

Pf.

- Let  $d$  = number of classrooms that the algorithm allocates.
- Classroom  $d$  is opened because we needed to schedule a lecture, say  $j$ ,  
that is incompatible with all  $d - 1$  other classrooms.
- These  $d$  lectures each end after  $s_j$ . ( $d-1$ )
- Since we sorted by start time, all these incompatibilities are caused by  
lectures that start no later than  $s_j$ .
- Thus, we have  $d$  lectures overlapping at time  $s_j + \varepsilon$ .
- Key observation  $\Rightarrow$  all schedules use  $\geq d$  classrooms. ▀

• in interval scheduling if greedy deviate from optimum, then also greedy is optimum.  
• here greedy achieve the minimum value for any possible solution.



## SECTION 4.2

# 4. GREEDY ALGORITHMS I

---

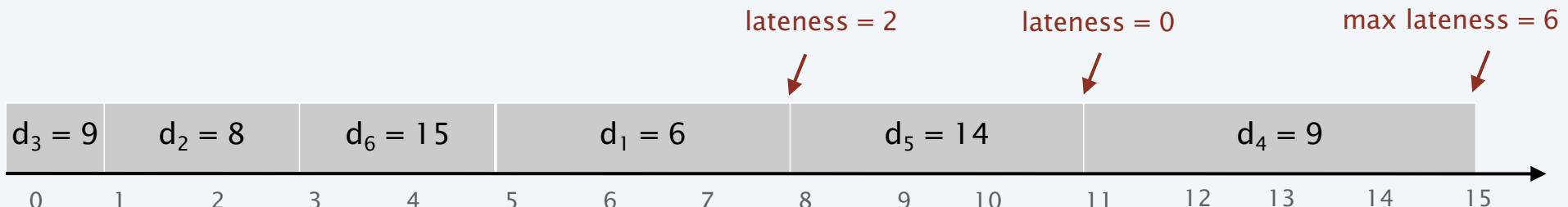
- ▶ *coin changing*
- ▶ *interval scheduling*
- ▶ *scheduling to minimize lateness*
- ▶ *optimal caching*

# Scheduling to minimizing lateness: Schedule intervals in one processor

Minimizing lateness problem.

- Single resource processes one job at a time.
- Job  $j$  requires  $t_j$  units of processing time and is due at time  $d_j$ . *→ time limit without pay penalty*
- If  $j$  starts at time  $s_j$ , it finishes at time  $f_j = s_j + t_j$ .
- Lateness:  $\ell_j = \max \{ 0, f_j - d_j \}$ .
- Goal: schedule all jobs to minimize maximum lateness  $L = \max_j \ell_j$ .

	1	2	3	4	5	6
$t_j$	3	2	1	4	3	2
$d_j$	6	8	9	9	14	15



$$f_3 = 0 + 1 = 1$$

$$\ell = \max \{ 0, 1 - 3 \} = 0$$

$$f_1 = 5 + 3 = 8$$

$$\ell = \max \{ 0, 8 - 6 \} = 2$$

## Minimizing lateness: greedy algorithms

---

**Greedy template.** Schedule jobs according to some natural order.

---

- [Shortest processing time first] Schedule jobs in ascending order of processing time  $t_j$ .
- [Earliest deadline first] Schedule jobs in ascending order of deadline  $d_j$ .
- [Smallest slack] Schedule jobs in ascending order of slack  $d_j - t_j$ .

## Minimizing lateness: greedy algorithms

Greedy template. Schedule jobs according to some natural order.

- [Shortest processing time first] Schedule jobs in ascending order of processing time  $t_j$ .

	1	2
$t_j$	1	10
$d_j$	100	10

counterexample

1 2  $\ell=1$  not optimal  
2 1  $\ell=0$  correct

- [Smallest slack] Schedule jobs in ascending order of slack  $d_j - t_j$ .

	1	2
$t_j$	1	10
$d_j$	2	10

counterexample

2 1  $\ell=9$  not optimal  
1 2  $\ell=1$  correct

## Minimizing lateness: earliest deadline first

EARLIEST-DEADLINE-FIRST ( $n, t_1, t_2, \dots, t_n, d_1, d_2, \dots, d_n$ )

SORT  $n$  jobs so that  $d_1 \leq d_2 \leq \dots \leq d_n$ .  $\rightarrow$  sorting for value of  $d$ !

$t \leftarrow 0$

FOR  $j = 1$  TO  $n$

Assign job  $j$  to interval  $[t, t + t_j]$ .

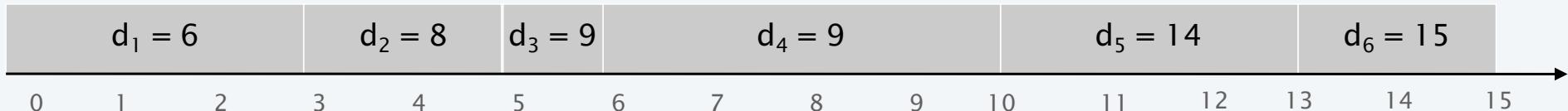
$s_j \leftarrow t ; f_j \leftarrow t + t_j$

$t \leftarrow t + t_j$

$\rightarrow$  simple operations

RETURN intervals  $[s_1, f_1], [s_2, f_2], \dots, [s_n, f_n]$ .

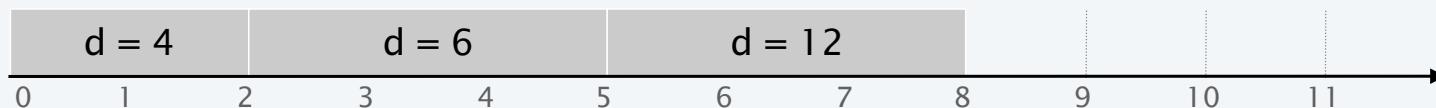
max lateness = 1 before was 6!



## Minimizing lateness: no idle time

→ an optimal schedule have not any idle time, if there is an idle time, can anticipate all the jobs that follow and certainly not decreasing latencies

Observation 1. There exists an optimal schedule with no **idle time**.

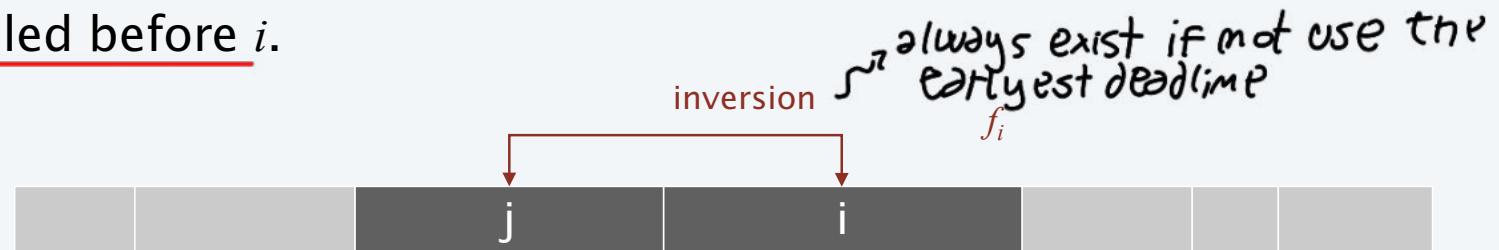


Observation 2. The earliest-deadline-first schedule has no idle time.

## Minimizing lateness: inversions

---

Def. Given a schedule  $S$ , an inversion is a pair of jobs  $i$  and  $j$  such that:  
 $i < j$  but  $j$  scheduled before  $i$ .



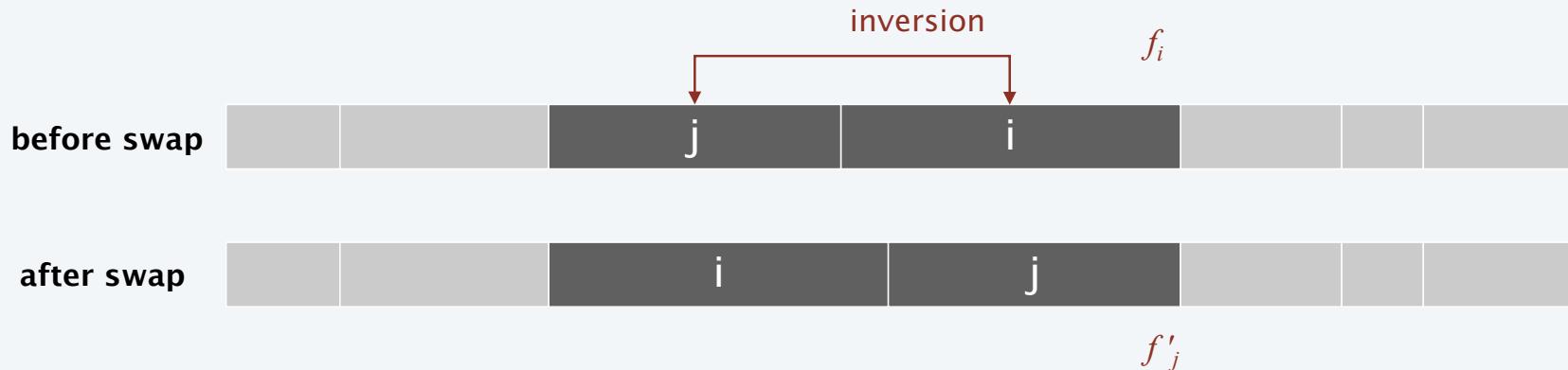
[ as before, we assume jobs are numbered so that  $d_1 \leq d_2 \leq \dots \leq d_n$  ]

Observation 3. The earliest-deadline-first schedule has no inversions.

Observation 4. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.

## Minimizing lateness: inversions

**Def.** Given a schedule  $S$ , an **inversion** is a pair of jobs  $i$  and  $j$  such that:  
 $i < j$  but  $j$  scheduled before  $i$ .



**Claim.** Swapping two adjacent, inverted jobs reduces the number of inversions by one and does not increase the max lateness.

**Pf.** Let  $\ell$  be the lateness before the swap, and let  $\ell'$  be it afterwards.

- $\ell'_k = \ell_k$  for all  $k \neq i, j$ .
- $\ell'_i \leq \ell_i$ . *statt before, is decreasing lateness*
- If job  $j$  is late,  $\ell'_j = f'_j - d_j$  (definition)  
 $= f_i - d_j$  ( $j$  now finishes at time  $f_i$ )  
 $\leq f_i - d_i$  (since  $i$  and  $j$  inverted)  
 $\leq \ell_i$ . (definition)

*↳ not increase lateness before swap*

# Minimizing lateness: analysis of earliest-deadline-first algorithm

---

Theorem. The earliest-deadline-first schedule  $S$  is optimal.

Pf. [by contradiction]

Define  $S^*$  to be an optimal schedule that has the fewest number of inversions, and let's see what happens.

- Can assume  $S^*$  has no idle time.
- If  $S^*$  has no inversions, then  $S = S^*$ .
- If  $S^*$  has an inversion, let  $i-j$  be an adjacent inversion.
- Swapping  $i$  and  $j$ 
  - does not increase the max lateness
  - strictly decreases the number of inversions
- This contradicts definition of  $S^*$  ■

## Greedy analysis strategies

→ interval scheduling we show that greedy algorithms always ahead of the optimum

Greedy algorithm stays ahead. Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.

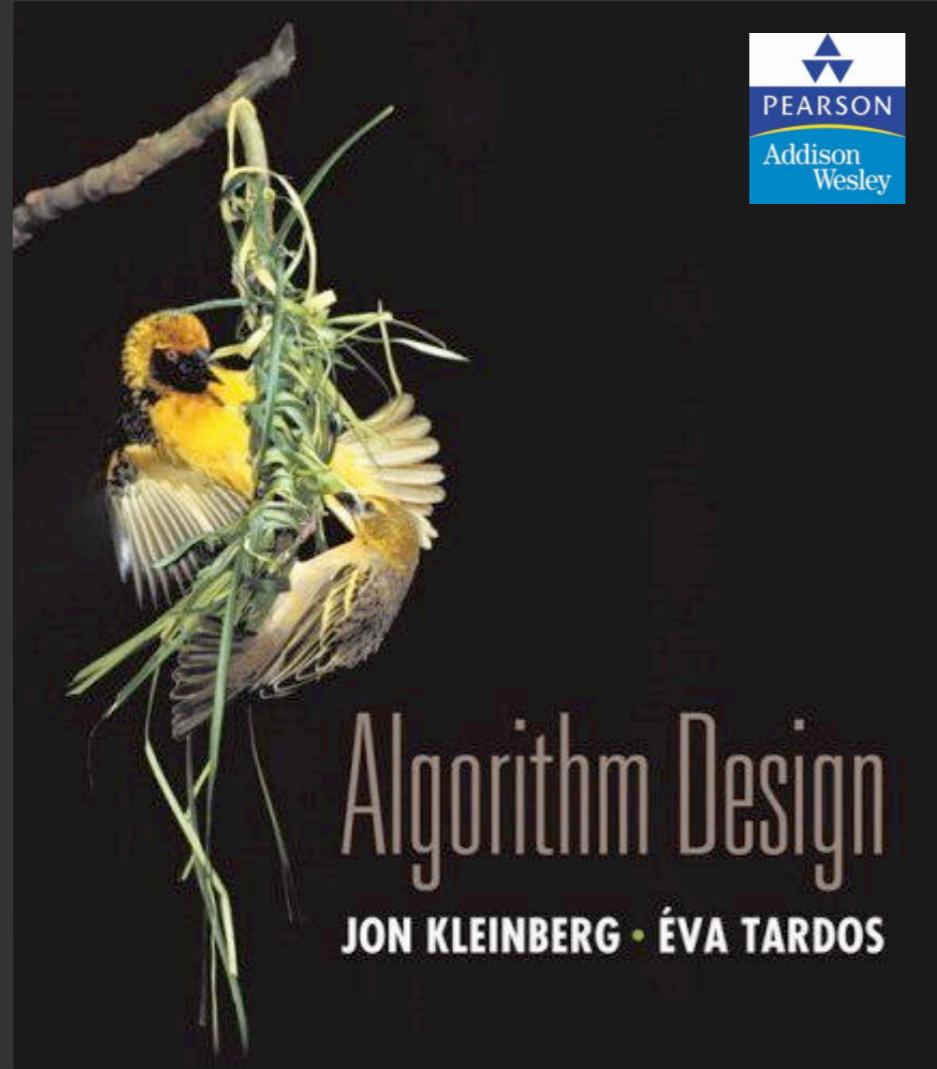
**Structural.** Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound. ~interval partitioning prove that greedy algorithm achieve the best possible result, comes from structure bound of the problem

**Exchange argument.** Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

↳ in minimizing lateness transform any solution in greedy solution making simple exchanging without deteriorate the quality of solution

**Other greedy algorithms.** Gale-Shapley, Kruskal, Prim, Dijkstra, Huffman, ...

here don't assume optimality, only the not decreasing



## 6. DYNAMIC PROGRAMMING I

---

- ▶ *weighted interval scheduling*
- ▶ *segmented least squares*
- ▶ *knapsack problem*
- ▶ *RNA secondary structure*

Lecture slides by Kevin Wayne

Copyright © 2005 Pearson–Addison Wesley

Copyright © 2013 Kevin Wayne

<http://www.cs.princeton.edu/~wayne/kleinberg-tardos>

# Algorithmic paradigms

---

**Greedy.** Build up a solution incrementally, myopically optimizing some local criterion.

**Divide-and-conquer.** Break up a problem into independent subproblems, solve each subproblem, and combine solution to subproblems to form solution to original problem.

**Dynamic programming.** Break up a problem into a series of overlapping subproblems, and build up solutions to larger and larger subproblems.

fancy name for  
caching away intermediate results  
in a table for later reuse

problem is break in some subproblems,  
but this subproblem are not disjoint, is not just  
combine the solution of subproblem in order to get  
the original solution .

if don't implement correctly Dynamic programming always lead to  
a exponential number of subproblems. ↗ need to use a data structure  
to store intermediate solution

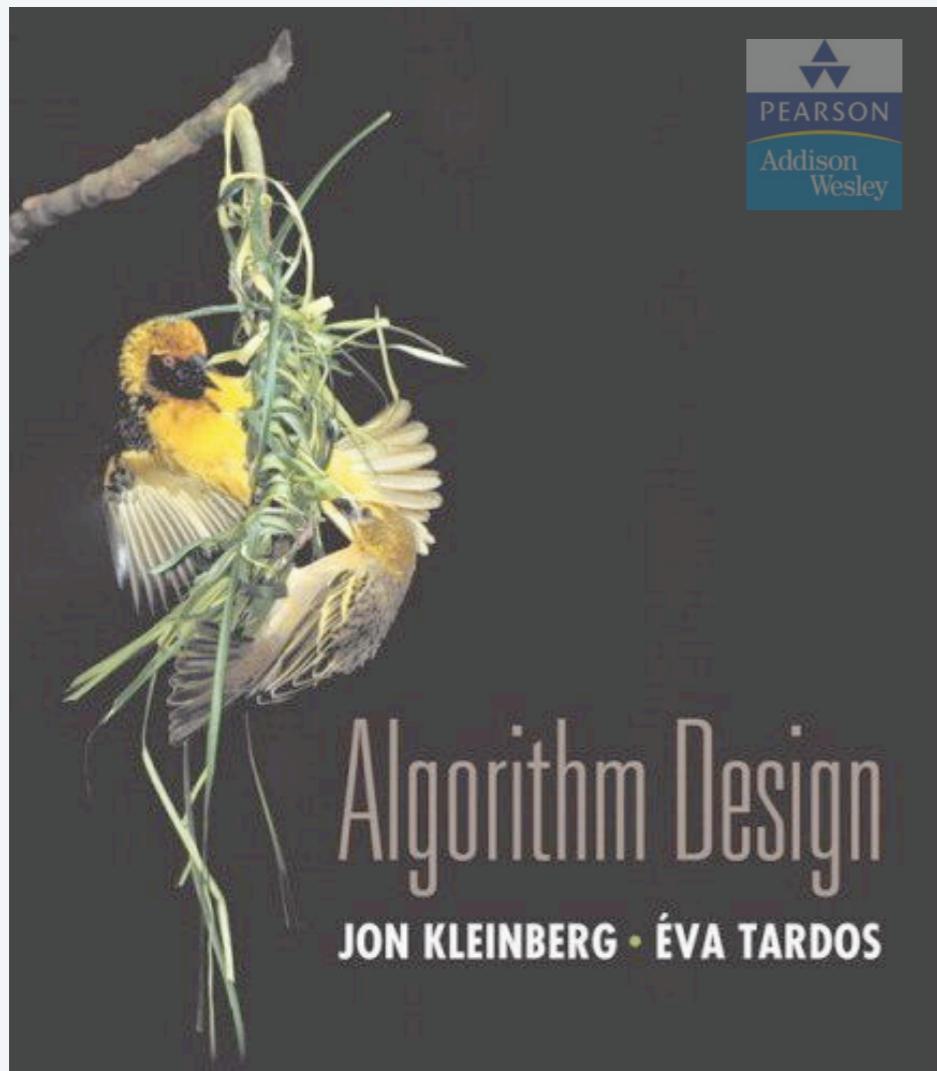
## 6. DYNAMIC PROGRAMMING I

---

- ▶ weighted interval scheduling
- ▶ segmented least squares
- ▶ knapsack problem
- ▶ RNA secondary structure

always start by a definition of the problem  
in term of recursive function, clear  
mathematic definition.

d.p. don't compute the solution, only the  
value of the optimal solution, by that we  
can reconstruct by induction the solution

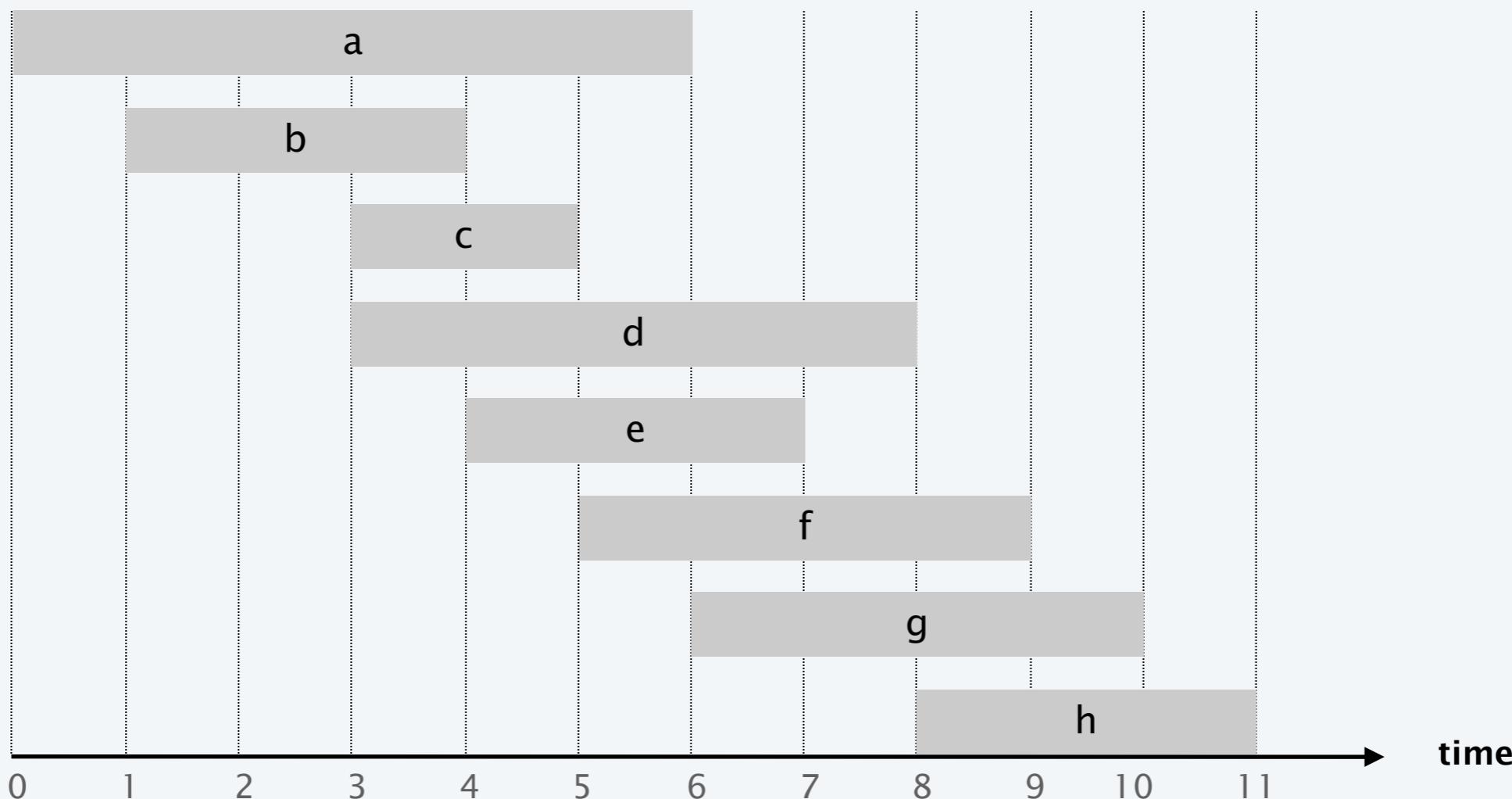


SECTION 6.1–6.2

# Weighted interval scheduling

Weighted interval scheduling problem.

- Job  $j$  starts at  $s_j$ , finishes at  $f_j$ , and has weight or value  $v_j$ .
- Two jobs compatible if they don't overlap.
- Goal: find maximum weight subset of mutually compatible jobs.



# Earliest-finish-time first algorithm

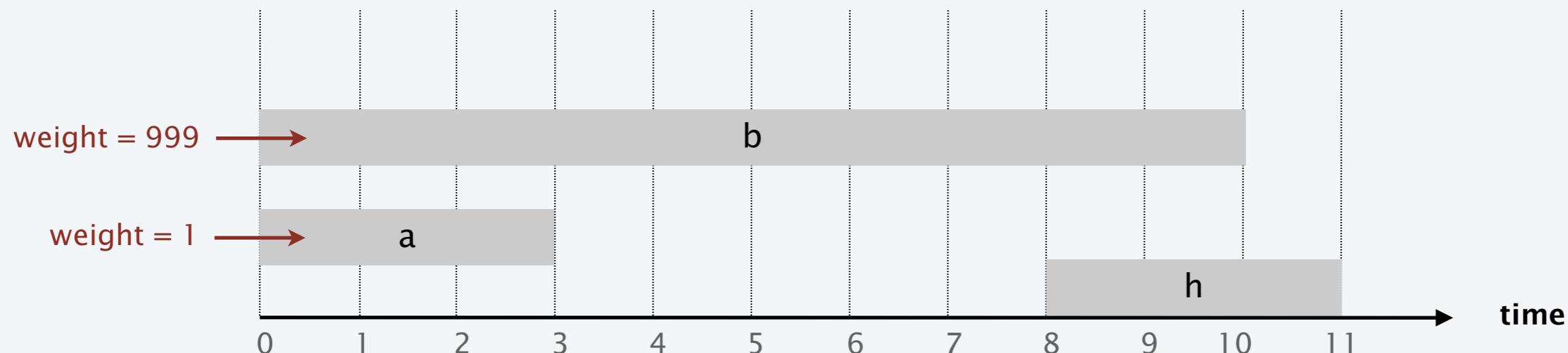
## Earliest finish-time first.

- Consider jobs in ascending order of finish time.
- Add job to subset if it is compatible with previously chosen jobs.

Recall. Greedy algorithm is correct if all weights are 1.

Observation. Greedy algorithm fails spectacularly for weighted version.

↳ not compatible with weighted version



# Weighted interval scheduling

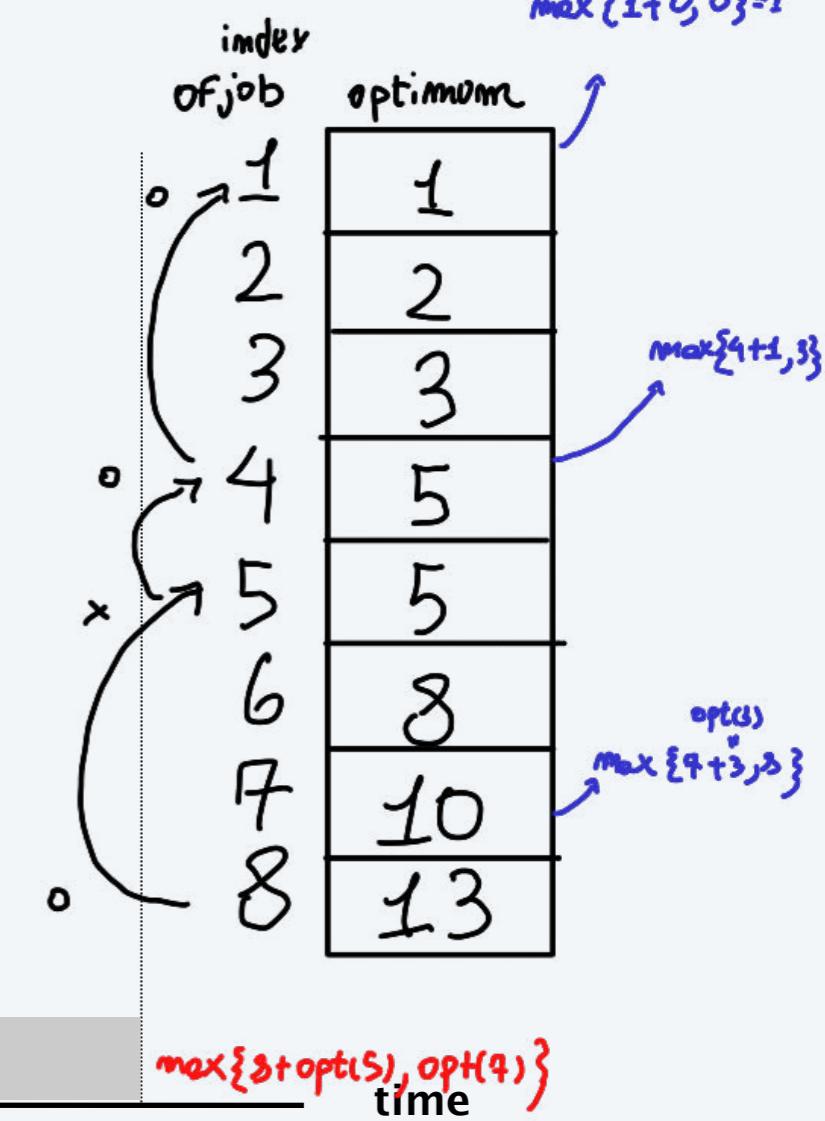
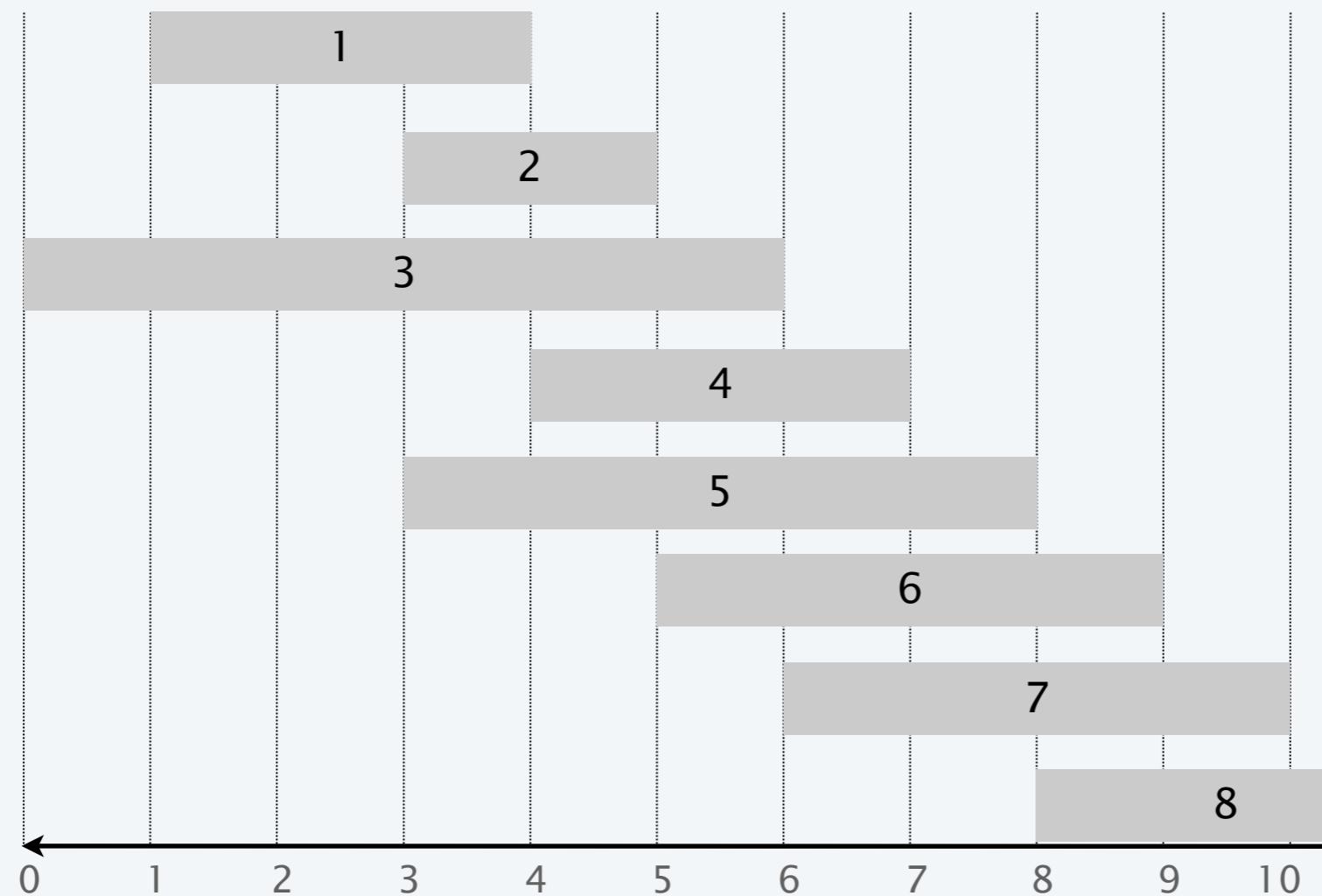
→ must be valid also for unweighted version  
 the solution (all weight = 1)

Notation. Label jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$ .

Def.  $p(j) = \text{largest index } i < j \text{ such that job } i \text{ is compatible with } j$

Ex.  $p(8) = 5, p(7) = 3, p(2) = 0$ .  $p(6) = 2, p(5) = 0, p(4) = 1$

start from  
 ↗ bottom to up



# Dynamic programming: binary choice

analysis: order of  $n$  for execution of dynamic programming algorithm, order of  $n \log n$  for order of finishing time.

Notation.  $OPT(j)$  = value of optimal solution to the problem consisting of job requests  $1, 2, \dots, j$ .

Case 1.  $OPT$  selects job  $j$ .

- Collect profit  $v_j$ . value of job  $j$
- Can't use incompatible jobs  $\{ p(j) + 1, p(j) + 2, \dots, j - 1 \}$ .
- Must include optimal solution to problem consisting of remaining compatible jobs  $1, 2, \dots, p(j)$ .

space complexity of an dynamic programming algorithm is equal to the size of the table used!

Case 2.  $OPT$  does not select job  $j$ .

- Must include optimal solution to problem consisting of remaining compatible jobs  $1, 2, \dots, j - 1$ .

optimal substructure property  
(proof via exchange argument)

where store the intermediate result

↓  
number of subproblem  
to compute opt solution.

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max \{ v_j + OPT(p(j)), OPT(j-1) \} & \text{otherwise} \end{cases}$$

among the 2 solutions want the best  
recursiv function

## Weighted interval scheduling: brute force

not optimal solution use an exponential number of call to Compute-Opt to compute the same value many times

Input:  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ .

Compute  $p[1], p[2], \dots, p[n]$ .

Compute-Opt( $j$ )

if  $j = 0$

return 0.

else

return  $\max(v[j] + \text{Compute-Opt}(p[j]), \text{Compute-Opt}(j-1))$ .

↑

need a data structure to store the opt values for subproblems

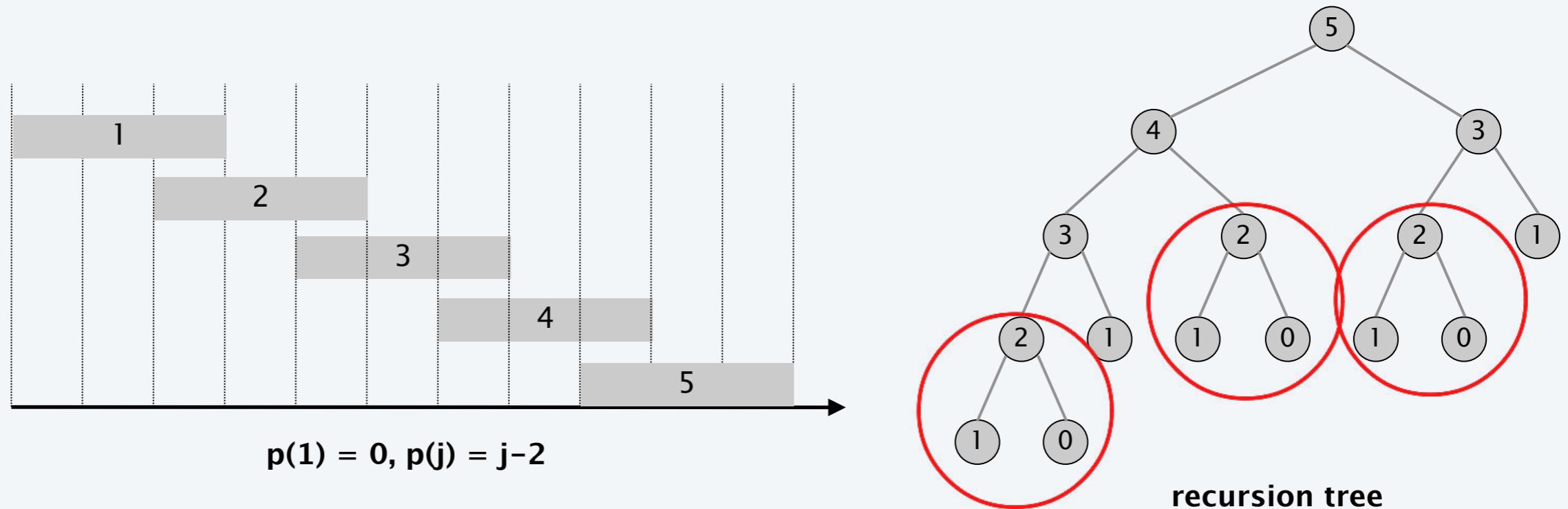
after computed optimal value, we use the data structure to compute the final solution

N.B.: don't recompute the same optimal value many times, Smart reducing of the Subproblems

# Weighted interval scheduling: brute force

**Observation.** Recursive algorithm fails spectacularly because of redundant subproblems  $\Rightarrow$  exponential algorithms.

**Ex.** Number of recursive calls for family of "layered" instances grows like Fibonacci sequence.



*(→ Find the same solution many times, want to store intermediate value)*

## Weighted interval scheduling: memoization

Memoization. Cache results of each subproblem; lookup as needed.

Input:  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ .

Compute  $p[1], p[2], \dots, p[n]$ .

for  $j = 1$  to  $n$

$M[j] \leftarrow$  empty.

$M[0] \leftarrow 0$ .

$\sim$  data structure to store values already  
    computed, a table

M-Compute-Opt( $j$ )

if  $M[j]$  is empty

$M[j] \leftarrow \max(v[j] + M\text{-Compute-Opt}(p[j]), M\text{-Compute-Opt}(j - 1))$ .

return  $M[j]$ .

USE CACHING, store intermediate values

## Weighted interval scheduling: running time

---

Claim. Memoized version of algorithm takes  $O(n \log n)$  time.

- Sort by finish time:  $O(n \log n)$ .
- Computing  $p(\cdot)$ :  $O(n \log n)$  via sorting by start time.  
*execution of dynamic program, using bottom-up*
- M-COMPUTE-OPT( $j$ ): each invocation takes  $O(1)$  time and either
  - (i) returns an existing value  $M[j]$
  - (ii) fills in one new entry  $M[j]$  and makes two recursive calls

*Progress that you make:*

- Progress measure  $\Phi = \#$  nonempty entries of  $M[]$ .
  - initially  $\Phi = 0$ , throughout  $\Phi \leq n$ .
  - (ii) increases  $\Phi$  by 1  $\Rightarrow$  at most  $2n$  recursive calls.  
*Filling on more items, that entries will not generate work in the future  $\rightarrow$  in filling, that require constant time*
- Overall running time of M-COMPUTE-OPT( $n$ ) is  $O(n)$ . ■

Remark.  $O(n)$  if jobs are presorted by start and finish times.

## Weighted interval scheduling: **finding a solution**

---

Q. DP algorithm computes optimal value. How to find solution itself?

A. Make a second pass.

```
Find-Solution(j)
if j = 0
    return ∅.
else if (v[j] + M[p[j]] > M[j-1])
    return {j} ∪ Find-Solution(p[j]).  
else
    return Find-Solution(j-1).
```

Analysis. # of recursive calls  $\leq n \Rightarrow O(n)$ .

## Weighted interval scheduling: bottom-up

simplest way to do  
dynamic program

Bottom-up dynamic programming. Unwind recursion.

best solution

BOTTOM-UP ( $n, s_1, \dots, s_n, f_1, \dots, f_n, v_1, \dots, v_n$ )

Sort jobs by finish time so that  $f_1 \leq f_2 \leq \dots \leq f_n$ .  $O(n \cdot \log n)$

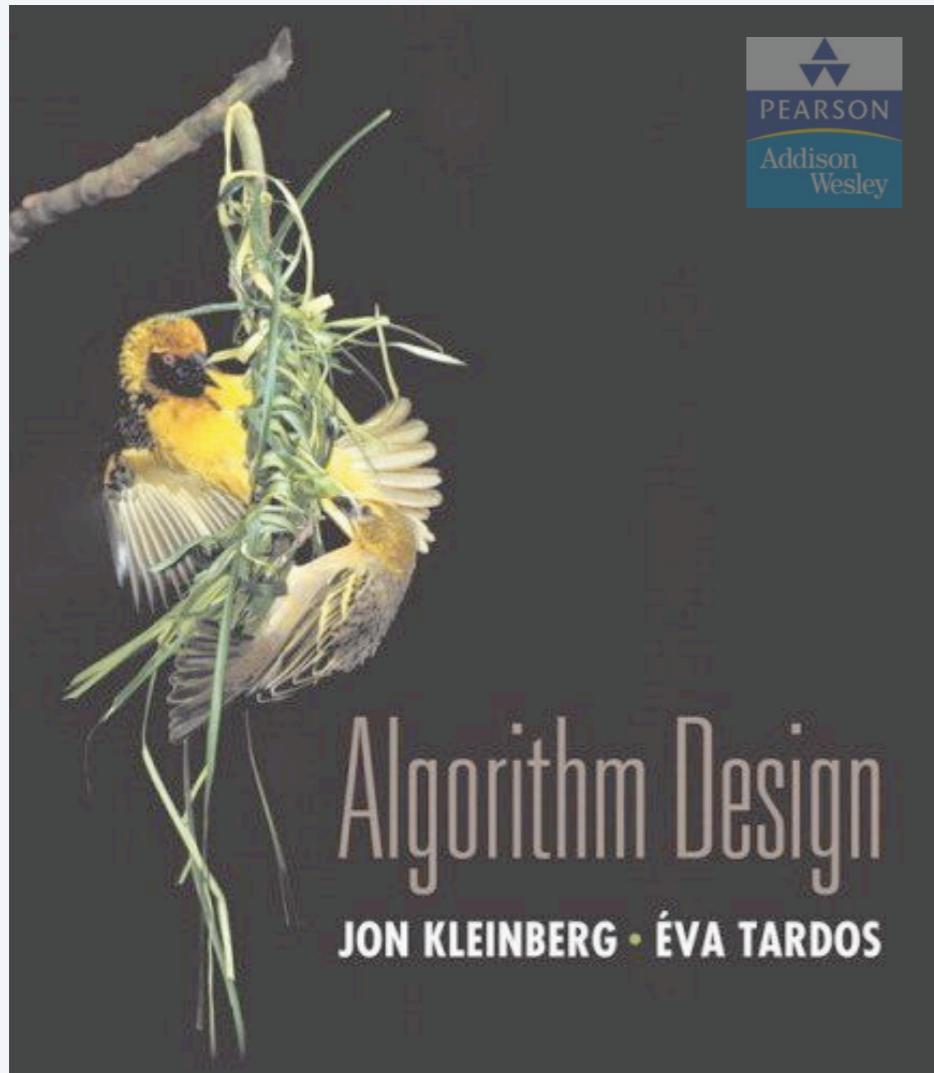
Compute  $p(1), p(2), \dots, p(n)$ .

$M[0] \leftarrow 0$ .

FOR  $j = 1$  TO  $n$

$M[j] \leftarrow \max \{ v_j + M[p(j)], M[j-1] \}$ .

running time  $O(n)$  only access to table, don't recompute anything



## 6. DYNAMIC PROGRAMMING I

---

- ▶ *weighted interval scheduling*
- ▶ *segmented least squares*
- ▶ ***knapsack problem***
- ▶ *RNA secondary structure*

SECTION 6.4

**Knapsack problem** maximizing total value of a set of objects, but every element have a value and a weight

- Given  $n$  objects and a "knapsack."
- Item  $i$  weighs  $w_i > 0$  and has value  $v_i > 0$ .
- Knapsack has capacity of  $W$ . total capacity weight of bag
- Goal: fill knapsack so as to maximize total value.  
but weight doesn't exceed the capacity

Ex.  $\{1, 2, 5\}$  has value 35. not optimal

Ex.  $\{3, 4\}$  has value 40.

Ex.  $\{3, 5\}$  has value 46 (but exceeds weight limit).

$i$	$v_i$	$w_i$
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

knapsack instance  
(weight limit  $W = 11$ )

Greedy by value. Repeatedly add item with maximum  $v_i$ .

Greedy by weight. Repeatedly add item with minimum  $w_i$ .

Greedy by ratio. Repeatedly add item with maximum ratio  $v_i / w_i$ .

Observation. None of greedy algorithms is optimal.

use dynamic programming 24

## Dynamic programming: false start

---

Def.  $OPT(i) = \max$  profit subset of items  $1, \dots, i$ .

Case 1.  $OPT$  does not select item  $i$ .

- $OPT$  selects best of  $\{1, 2, \dots, i-1\}$ .

optimal substructure property  
(proof via exchange argument)

Case 2.  $OPT$  selects item  $i$ .

- Selecting item  $i$  does not immediately imply that we will have to reject other items.
- Without knowing what other items were selected before  $i$ , we don't even know if we have enough room for  $i$ .

Conclusion. Need more subproblems!

# Dynamic programming: adding a new variable

Def.  $OPT(i, w) = \max$  profit subset of items  $1, \dots, i$  with weight limit  $w$ .

Case 1.  $OPT$  does not select item  $i$ .

- $OPT$  selects best of  $\{1, 2, \dots, i-1\}$  using weight limit  $w$ .

Case 2.  $OPT$  selects item  $i$ .

- New weight limit =  $w - w_i$ .
- $OPT$  selects best of  $\{1, 2, \dots, i-1\}$  using this new weight limit.

optimal substructure property  
(proof via exchange argument)

$$OPT(i, w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max \{ OPT(i-1, w), v_i + OPT(i-1, w - w_i) \} & \text{otherwise} \end{cases}$$

use two dimensional table

## Knapsack problem: bottom-up

items      capacity  
                ↑  
weights      values

KNAPSACK ( $n, W, w_1, \dots, w_n, v_1, \dots, v_n$ )

FOR  $w = 0$  TO  $W$

$M[0, w] \leftarrow 0.$  assign first row for value  $\Rightarrow$  equal to 0 for each capacity

FOR  $i = 1$  TO  $n$

FOR  $w = 1$  TO  $W$

IF  $(w_i > w)$   $M[i, w] \leftarrow M[i-1, w].$

ELSE  $M[i, w] \leftarrow \max \{ M[i-1, w], v_i + M[i-1, w - w_i]\}.$

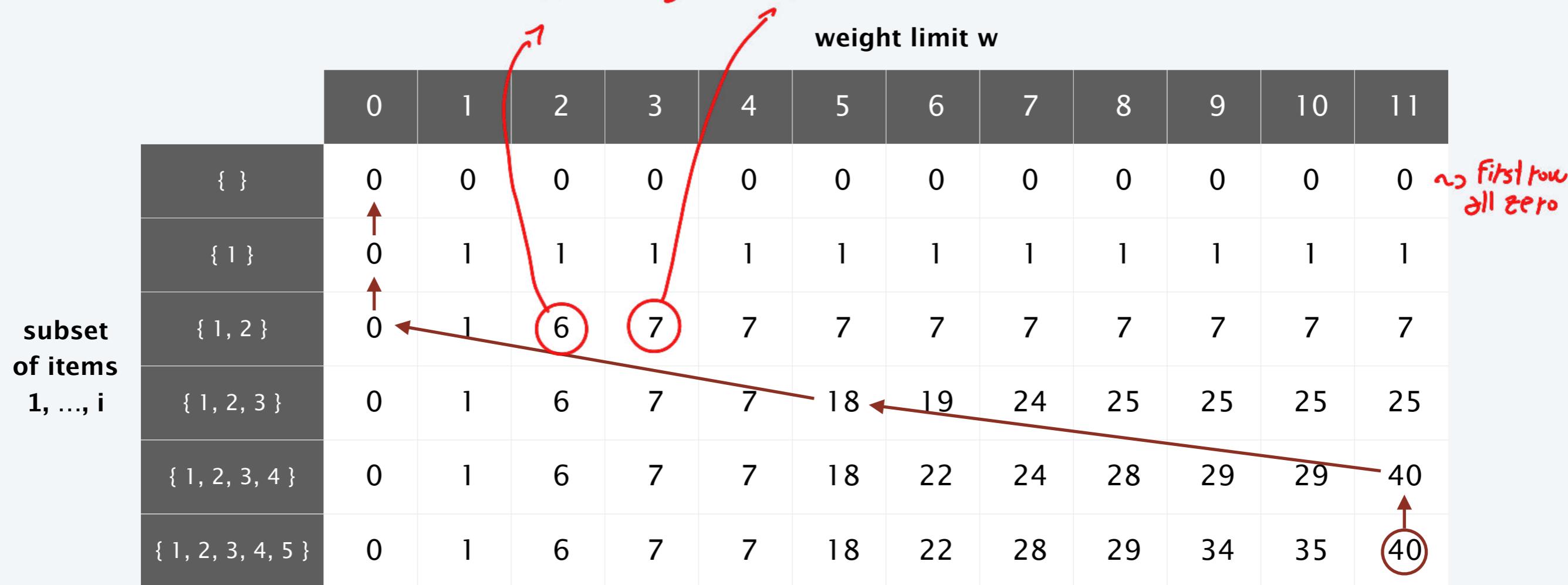
RETURN  $M[n, W].$

# Knapsack problem: bottom-up demo

$i$	$v_i$	$w_i$
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

$$OPT(i, w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max\{ OPT(i-1, w), v_i + OPT(i-1, w - w_i) \} & \text{otherwise} \end{cases}$$

$6+0$   
 $6+1+0$   
 $w=2 \text{ only } i=2$     $w=3 \quad \{1, 2\}$



$OPT(i, w) = \text{max profit subset of items } 1, \dots, i \text{ with weight limit } w.$

## Knapsack problem: running time

**Theorem.** There exists an algorithm to solve the knapsack problem with  $n$  items and maximum weight  $W$  in  $\Theta(n W)$  time and  $\Theta(n W)$  space.

Pf.

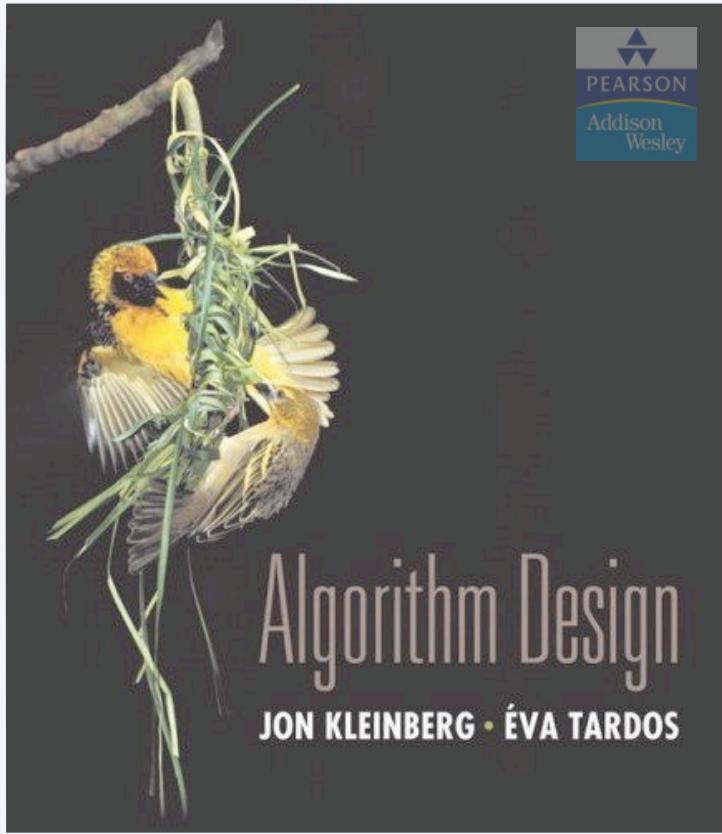
- Takes  $O(1)$  time per table entry.
- There are  $\Theta(n W)$  table entries.
- After computing optimal values, can trace back to find solution:  
take item  $i$  in  $OPT(i, w)$  iff  $M[i, w] > M[i - 1, w]$ . ▀

weights are integers  
between 1 and  $W$

### Remarks.

- Not polynomial in input size! ← "pseudo-polynomial"
- Decision version of knapsack problem is NP-COMPLETE. [ CHAPTER 8 ]
- There exists a poly-time algorithm that produces a feasible solution that has value within 1% of optimum. [ SECTION 11.8 ]

for represent  $w$  we need log<sub>2</sub> bits



## SECTION 6.6

# 6. DYNAMIC PROGRAMMING II

---

- ▶ **sequence alignment**      match the word that are you writing with that on a dictionary...
- ▶ *Hirschberg's algorithm*
- ▶ *Bellman-Ford algorithm*
- ▶ *distance vector protocols*
- ▶ *negative cycles in a digraph*

# String similarity

Q. How similar are two strings?

Ex. occurranc and occurrence.

o	c	u	r	r	a	n	c	e	-
---	---	---	---	---	---	---	---	---	---

o	c	c	u	r	r	e	n	c	e
---	---	---	---	---	---	---	---	---	---

6 mismatches, 1 gap

↳ is a mismatch when there is no character on one string and there is a character on the other string

o	c	-	u	r	r	a	n	c	e
---	---	---	---	---	---	---	---	---	---

o	c	c	u	r	r	e	n	c	e
---	---	---	---	---	---	---	---	---	---

1 mismatch, 1 gap

↳ position where two strings have different character.

o	c	-	u	r	r	-	a	n	c	e
---	---	---	---	---	---	---	---	---	---	---

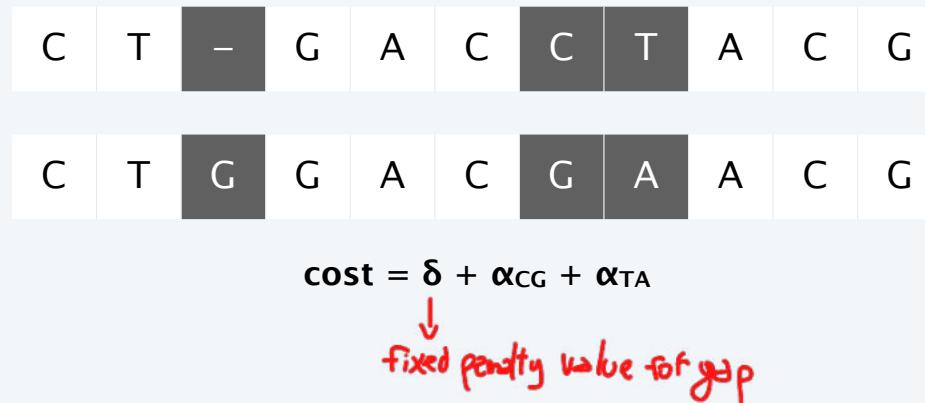
o	c	c	u	r	r	e	-	n	c	e
---	---	---	---	---	---	---	---	---	---	---

0 mismatches, 3 gaps

**Edit distance** associate a value of penalty to gap and mismatch

Edit distance. [Levenshtein 1966, Needleman-Wunsch 1970]

- Gap penalty  $\delta$ ; mismatch penalty  $\alpha_{pq}$ .
- Cost = sum of gap and mismatch penalties.



Applications. Unix diff, speech recognition, computational biology, ...

## Sequence alignment

~> find an alignment for the 2 strings  
to minimize cost of edit distance

Goal. Given two strings  $x_1 x_2 \dots x_m$  and  $y_1 y_2 \dots y_n$  find min cost alignment.

Def. An alignment  $M$  is a set of ordered pairs  $x_i - y_j$  such that each item occurs in at most one pair and no crossings.

$x_i - y_j$  and  $x_{i'} - y_{j'}$  cross if  $i < i'$ , but  $j > j'$

Def. The cost of an alignment  $M$  is:

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i : x_i \text{ unmatched}} \delta}_{\text{gap}} + \sum_{j : y_j \text{ unmatched}} \delta$$

$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	
C	T	A	C	C	-	G
$y_1$	$y_2$	$y_3$	$y_4$	$y_5$	$y_6$	
-	T	A	C	A	T	G

an alignment of CTACCG and TACATG:

$$M = \{x_2 - y_1, x_3 - y_2, x_4 - y_3, x_5 - y_4, x_6 - y_6\}$$

## Sequence alignment: problem structure

Def.  $OPT(i, j) = \min \text{ cost of aligning prefix strings } x_1 x_2 \dots x_i \text{ and } y_1 y_2 \dots y_j.$

Case 1.  $OPT$  matches  $x_i - y_j$ .

Pay mismatch for  $x_i - y_j$  + min cost of aligning  $x_1 x_2 \dots x_{i-1}$  and  $y_1 y_2 \dots y_{j-1}.$

Case 2a.  $OPT$  leaves  $x_i$  unmatched.

Pay gap for  $x_i$  + min cost of aligning  $x_1 x_2 \dots x_{i-1}$  and  $y_1 y_2 \dots y_j.$

Case 2b.  $OPT$  leaves  $y_j$  unmatched.

Pay gap for  $y_j$  + min cost of aligning  $x_1 x_2 \dots x_i$  and  $y_1 y_2 \dots y_{j-1}.$

optimal substructure property  
(proof via exchange argument)

X	Y
A	A
T	B
C	D
D	D
E	F

$$OPT(i, j) = \begin{cases} j\delta & \text{if } i = 0 \\ \min \left\{ \begin{array}{l} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{array} \right. & \text{otherwise} \\ i\delta & \text{if } j = 0 \end{cases}$$

↑ Pay the size mismatch with gaps

te recursive function

three possibilities:

- i)  $\overline{E} \overline{F}$  mismatch
- ii)  $\overline{E} \underline{F}$  gap on x
- iii)  $\underline{E} \underline{F}$  gap on y

# Sequence alignment: algorithm using bottom-up, filling a table

SEQUENCE-ALIGNMENT ( $m, n, x_1, \dots, x_m, y_1, \dots, y_n, \delta, \alpha$ )

FOR  $i = 0$  TO  $m$

$M[i, 0] \leftarrow i \delta.$

FOR  $j = 0$  TO  $n$

$M[0, j] \leftarrow j \delta.$

FOR  $i = 1$  TO  $m$       *complexity  $\propto m \cdot n$*

FOR  $j = 1$  TO  $n$

$$M[i, j] \leftarrow \min \{ \alpha[x_i, y_j] + M[i-1, j-1], \\ \delta + M[i-1, j], \\ \delta + M[i, j-1] \}.$$

RETURN  $M[m, n]$ . *→ backtrace for find the optimal solution*

## Sequence alignment: analysis

---

Theorem. The dynamic programming algorithm computes the edit distance (and optimal alignment) of two strings of length  $m$  and  $n$  in  $\Theta(mn)$  time and  $\Theta(mn)$  space.

Pf.

- Algorithm computes edit distance.
- Can trace back to extract optimal alignment itself. ▀

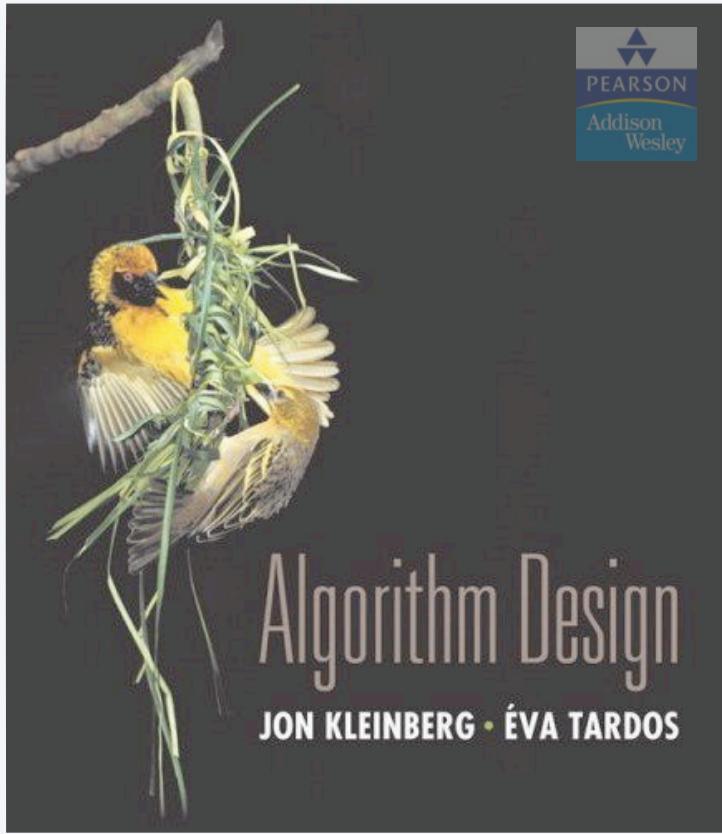
Q. Can we avoid using quadratic space?

A. Easy to compute optimal value in  $O(mn)$  time and  $O(m + n)$  space.

almost linear space  
↑

- Compute  $\text{OPT}(i, \bullet)$  from  $\text{OPT}(i - 1, \bullet)$ .
- But, no longer easy to recover optimal alignment itself.

only need precedent value, no need to track all values, but later difficult to recover optimal solution



## SECTION 6.7

# 6. DYNAMIC PROGRAMMING II

---

- ▶ *sequence alignment*
- ▶ *Hirschberg's algorithm*
- ▶ *Bellman-Ford algorithm*
- ▶ *distance vector protocols*
- ▶ *negative cycles in a digraph*

## Sequence alignment in linear space

**Theorem.** There exist an algorithm to find an optimal alignment in  $O(mn)$  time and  $O(m + n)$  space.

- Clever combination of divide-and-conquer and dynamic programming.
- Inspired by idea of Savitch from complexity theory.

Programming  
Techniques      G. Manacher  
Editor

# A Linear Space Algorithm for Computing Maximal Common Subsequences

D.S. Hirschberg  
Princeton University

The problem of finding a longest common subsequence of two strings has been solved in quadratic time and space. An algorithm is presented which will solve this problem in quadratic time and in linear space.

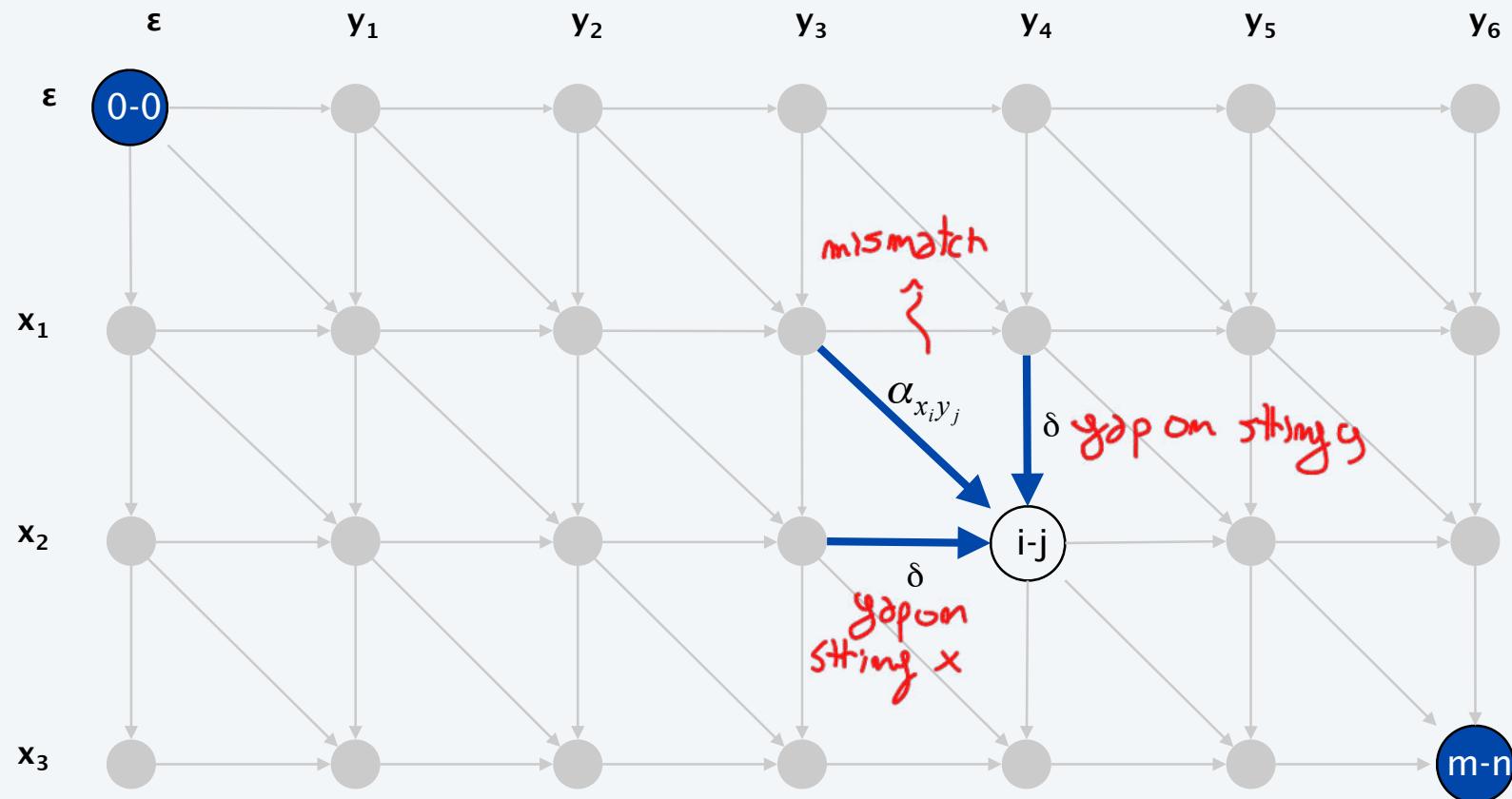
Key Words and Phrases: subsequence, longest common subsequence, string correction, editing

CR Categories: 3.63, 3.73, 3.79, 4.22, 5.25

## Hirschberg's algorithm

Edit distance graph.

- Let  $f(i, j)$  be shortest path from  $(0, 0)$  to  $(i, j)$ .
- Lemma:  $f(i, j) = OPT(i, j)$  for all  $i$  and  $j$ .



# Hirschberg's algorithm

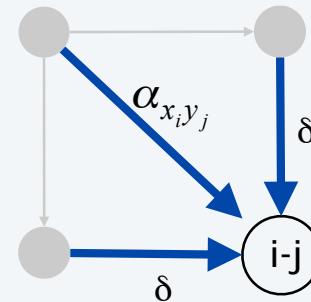
Edit distance graph.

- Let  $f(i, j)$  be shortest path from  $(0, 0)$  to  $(i, j)$ .
- Lemma:  $f(i, j) = OPT(i, j)$  for all  $i$  and  $j$ .

Pf of Lemma. [ by strong induction on  $i + j$  ]

- Base case:  $f(0, 0) = OPT(0, 0) = 0$ .
- Inductive hypothesis: assume true for all  $(i', j')$  with  $i' + j' < i + j$ .
- Last edge on shortest path to  $(i, j)$  is from  $(i - 1, j - 1)$ ,  $(i - 1, j)$ , or  $(i, j - 1)$ .
- Thus,

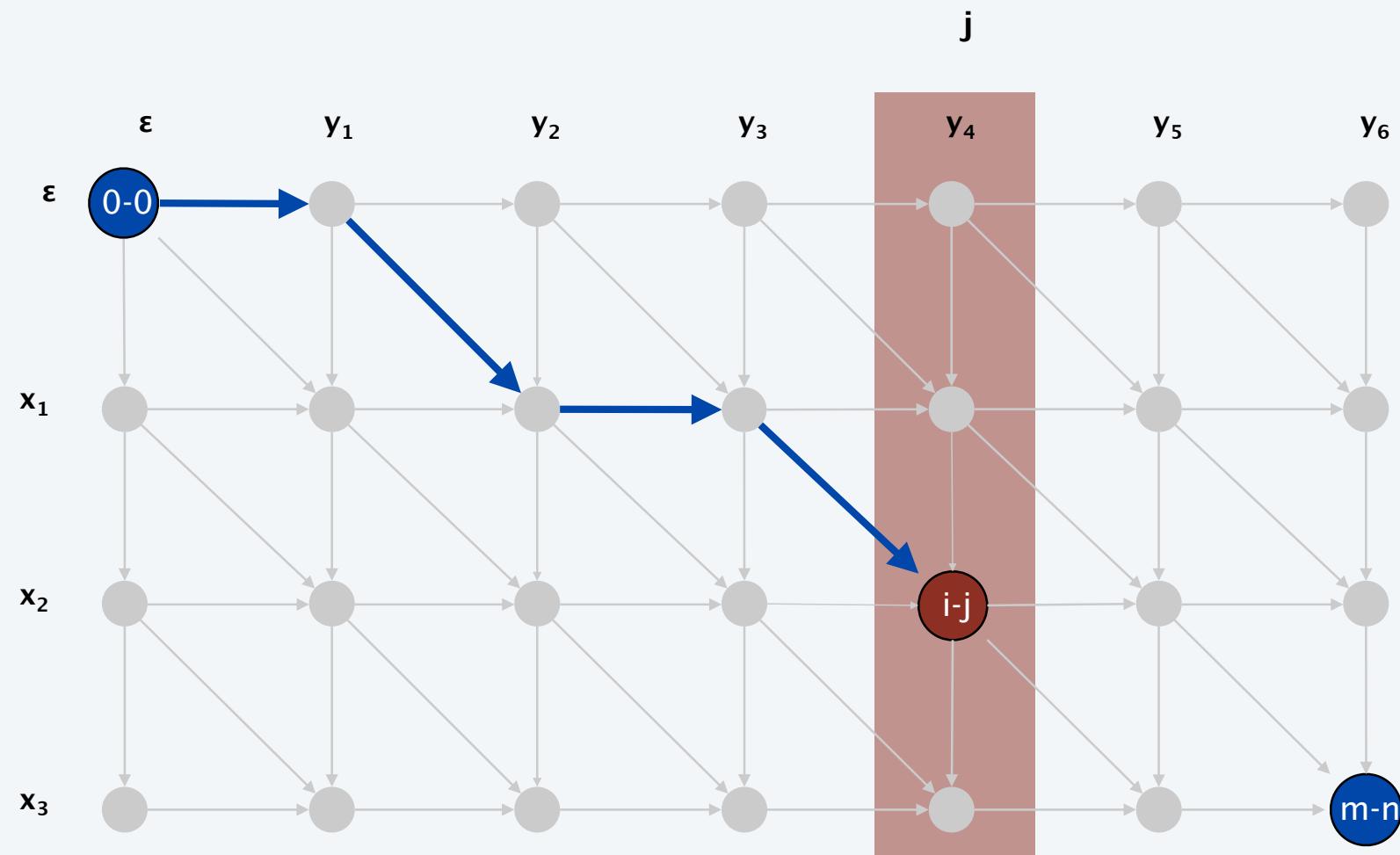
$$\begin{aligned} f(i, j) &= \min\{\alpha_{x_i y_j} + f(i - 1, j - 1), \delta + f(i - 1, j), \delta + f(i, j - 1)\} \\ &= \min\{\alpha_{x_i y_j} + OPT(i - 1, j - 1), \delta + OPT(i - 1, j), \delta + OPT(i, j - 1)\} \\ &= OPT(i, j) \blacksquare \end{aligned}$$



# Hirschberg's algorithm

Edit distance graph.

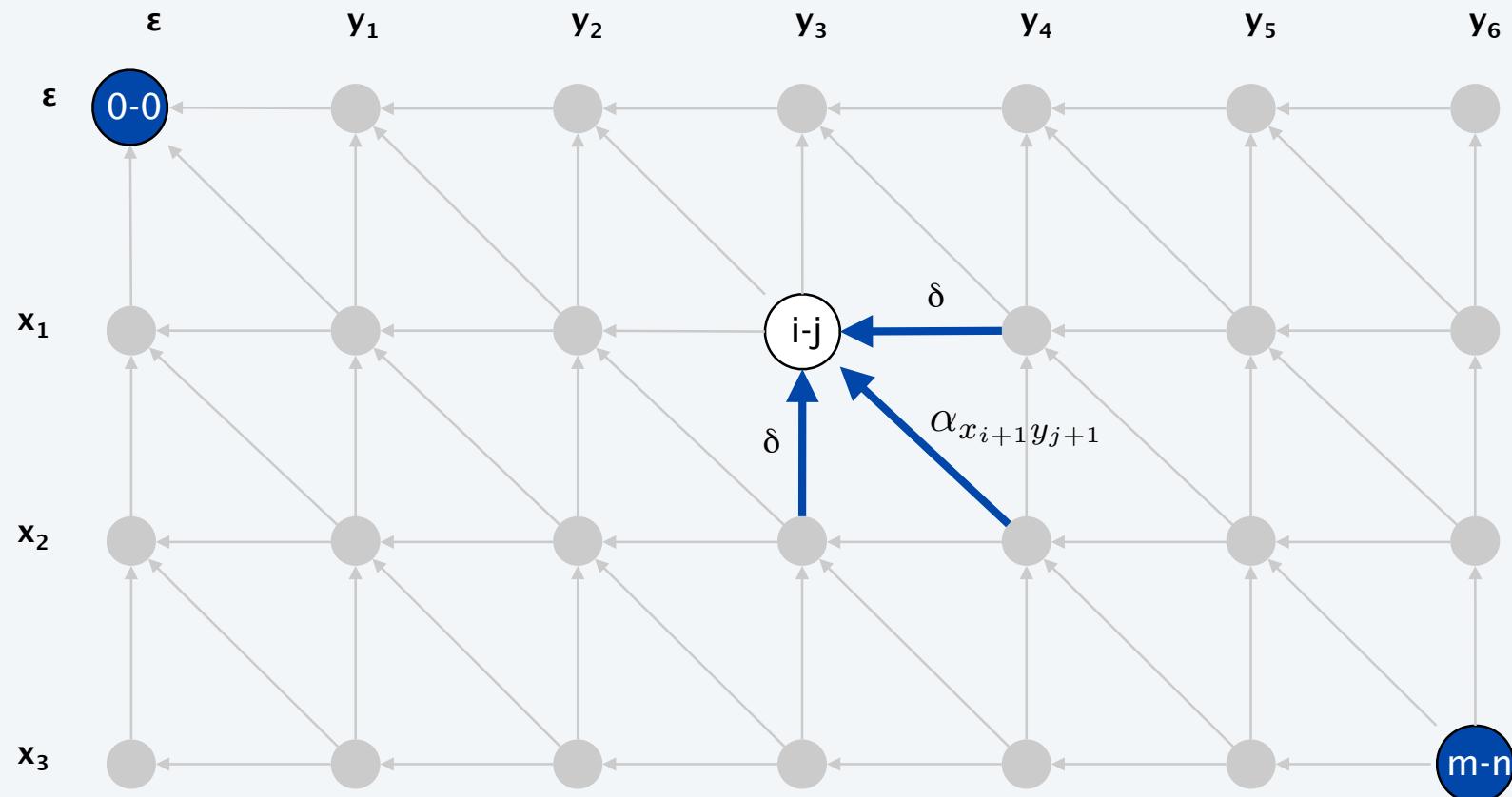
- Let  $f(i, j)$  be shortest path from  $(0, 0)$  to  $(i, j)$ .
- Lemma:  $f(i, j) = OPT(i, j)$  for all  $i$  and  $j$ .
- Can compute  $f(\cdot, j)$  for any  $j$  in  $O(mn)$  time and  $O(m + n)$  space.



# Hirschberg's algorithm

Edit distance graph.

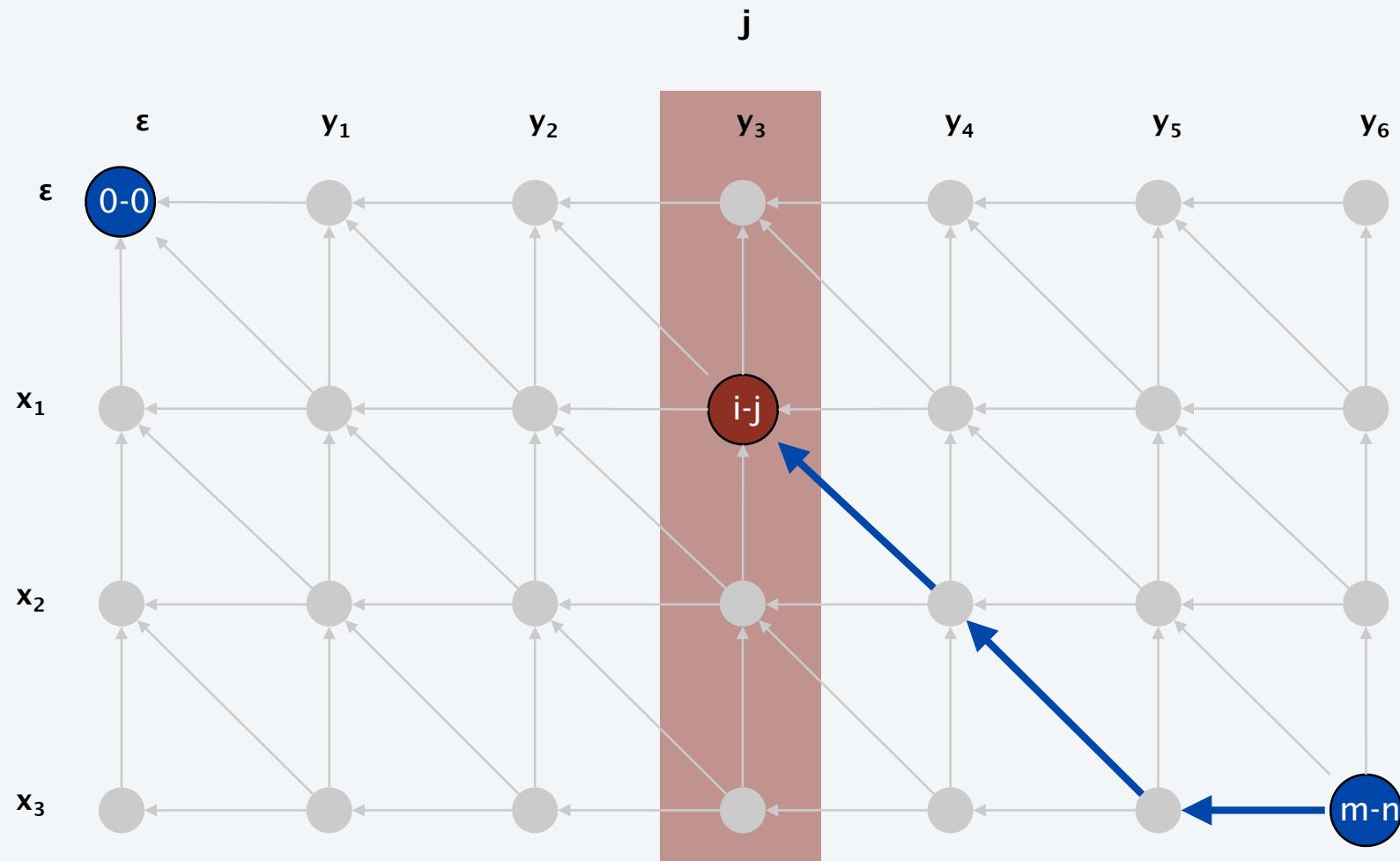
- Let  $g(i, j)$  be shortest path from  $(i, j)$  to  $(m, n)$ .
- Can compute by reversing the edge orientations and inverting the roles of  $(0, 0)$  and  $(m, n)$ .



# Hirschberg's algorithm

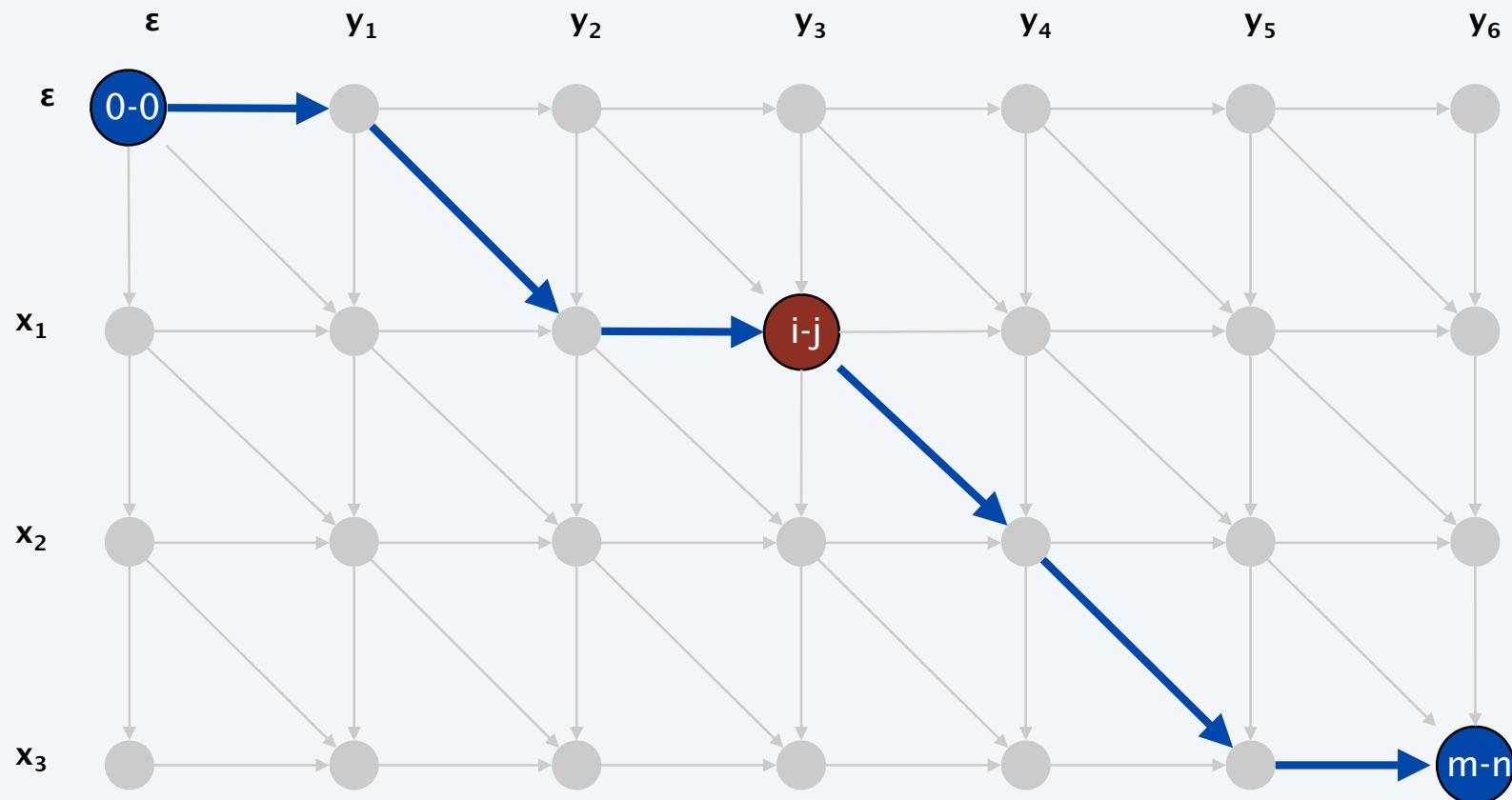
Edit distance graph.

- Let  $g(i, j)$  be shortest path from  $(i, j)$  to  $(m, n)$ .
- Can compute  $g(\cdot, j)$  for any  $j$  in  $O(mn)$  time and  $O(m + n)$  space.



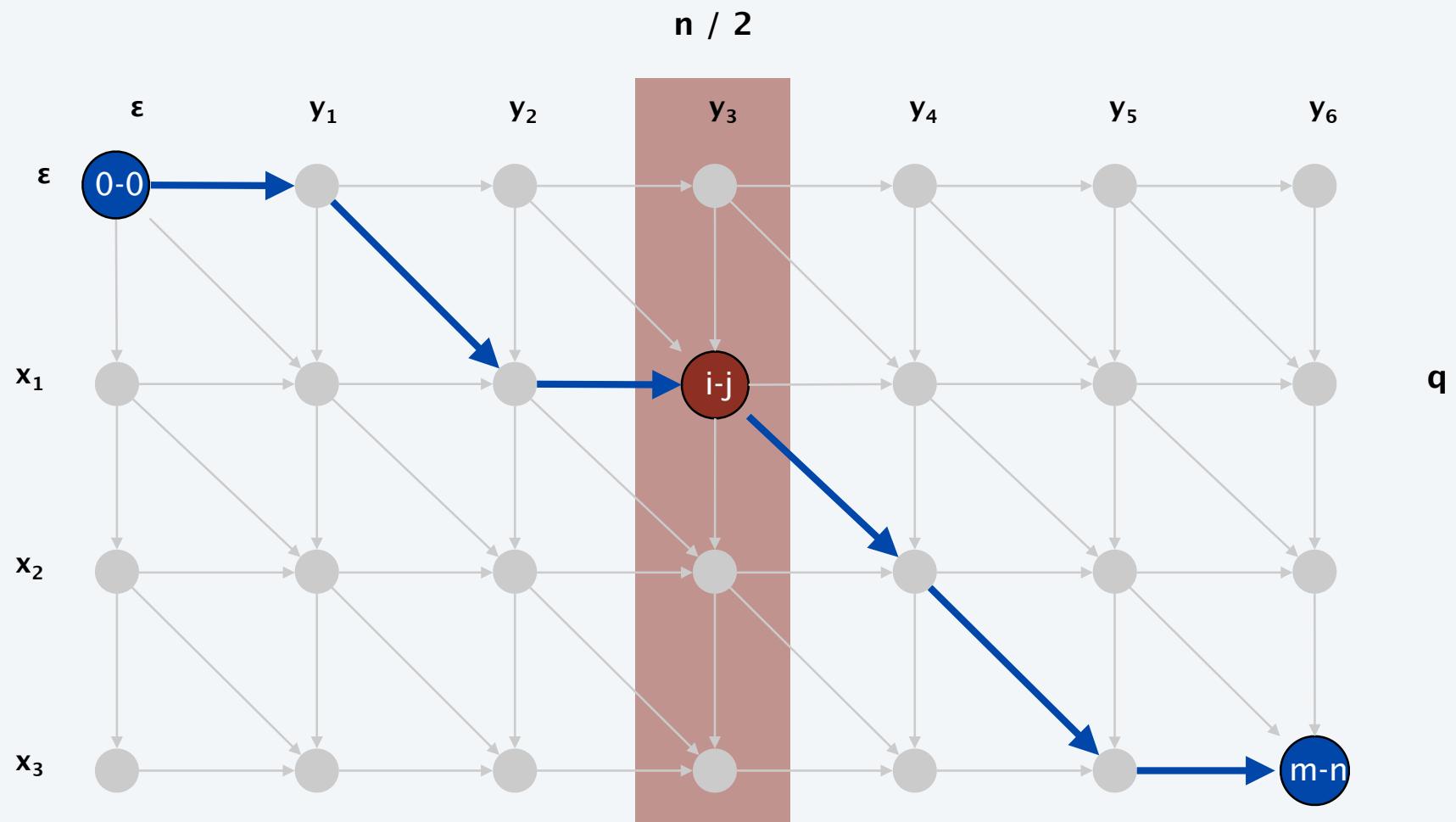
# Hirschberg's algorithm

Observation 1. The cost of the shortest path that uses  $(i, j)$  is  $f(i, j) + g(i, j)$ .



# Hirschberg's algorithm

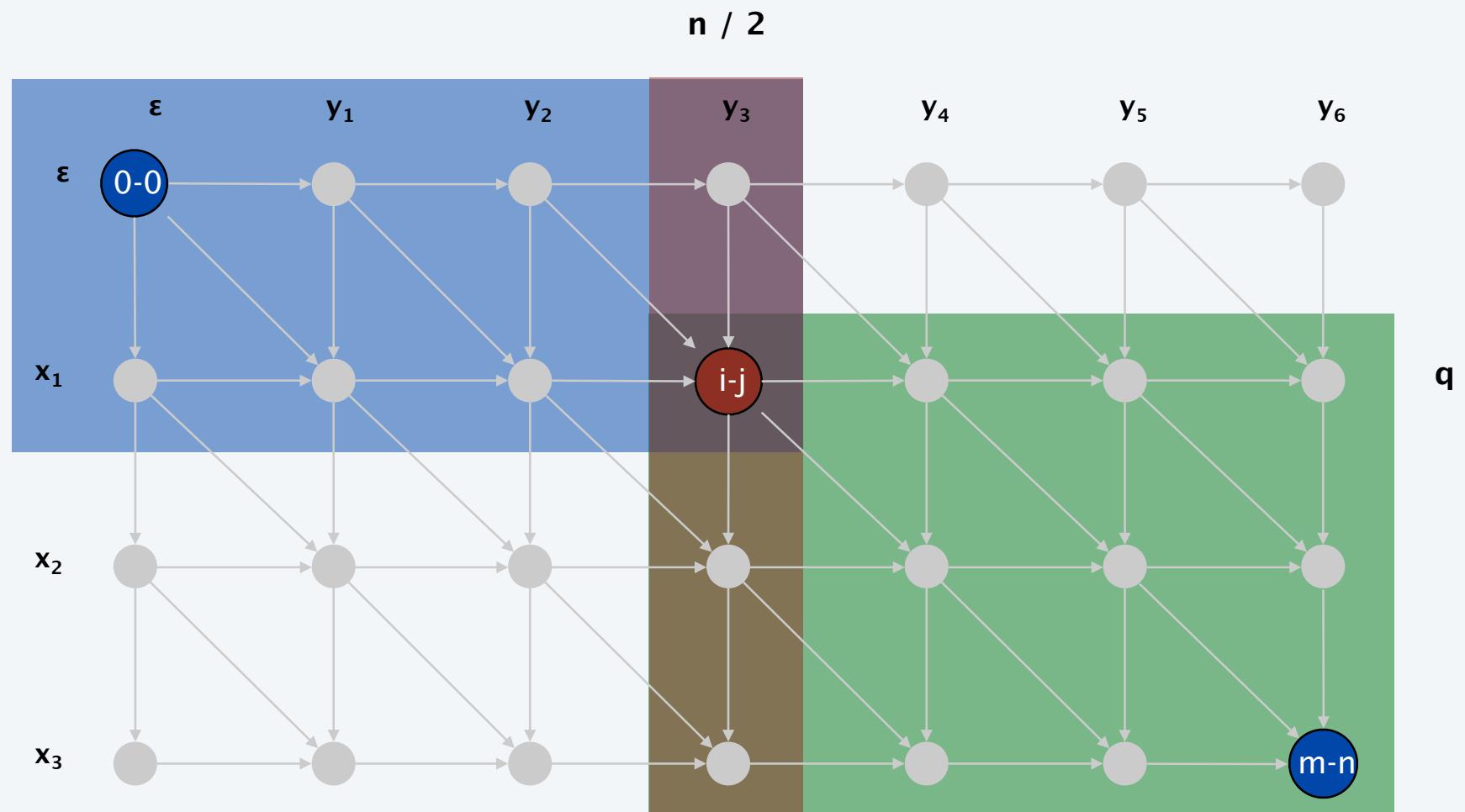
**Observation 2.** let  $q$  be an index that minimizes  $f(q, n/2) + g(q, n/2)$ .  
Then, there exists a shortest path from  $(0, 0)$  to  $(m, n)$  uses  $(q, n/2)$ .



# Hirschberg's algorithm

Divide. Find index  $q$  that minimizes  $f(q, n/2) + g(q, n/2)$ ; align  $x_q$  and  $y_{n/2}$ .

Conquer. Recursively compute optimal alignment in each piece.



## Hirschberg's algorithm: running time analysis warmup

---

**Theorem.** Let  $T(m, n) = \max$  running time of Hirschberg's algorithm on strings of length at most  $m$  and  $n$ . Then,  $T(m, n) = O(m n \log n)$ .

---

Pf.  $T(m, n) \leq 2 T(m, n/2) + O(m n)$   
 $\Rightarrow T(m, n) = O(m n \log n)$ .

**Remark.** Analysis is not tight because two subproblems are of size  $(q, n/2)$  and  $(m - q, n/2)$ . In next slide, we save  $\log n$  factor.

## Hirschberg's algorithm: running time analysis

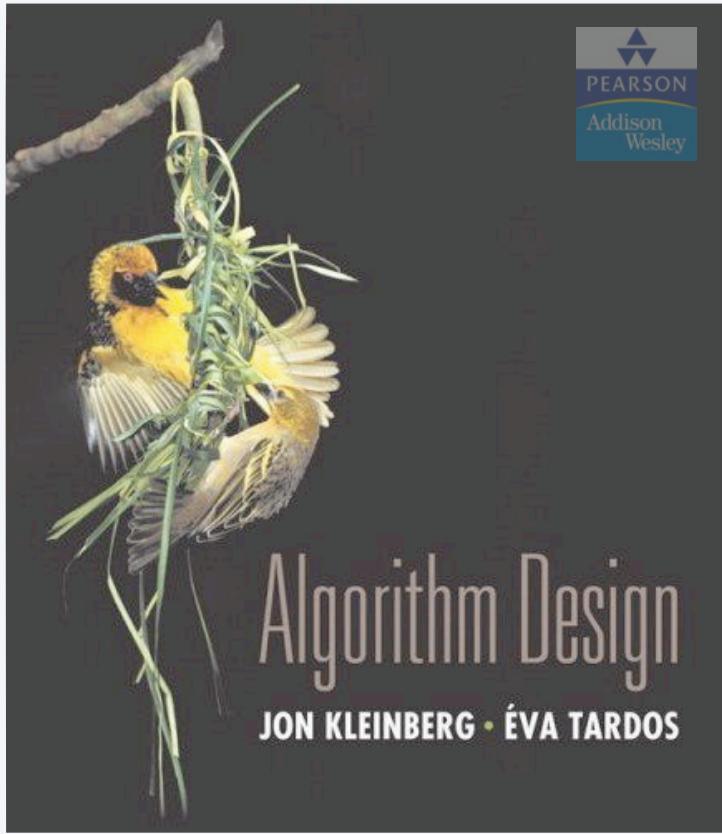
---

**Theorem.** Let  $T(m, n) = \max$  running time of Hirschberg's algorithm on strings of length at most  $m$  and  $n$ . Then,  $T(m, n) = O(m n)$ .

Pf. [ by induction on  $n$  ]

- $O(m n)$  time to compute  $f(\bullet, n/2)$  and  $g(\bullet, n/2)$  and find index  $q$ .
- $T(q, n/2) + T(m - q, n/2)$  time for two recursive calls.
- Choose constant  $c$  so that:  $T(m, 2) \leq c m$   
 $T(2, n) \leq c n$   
 $T(m, n) \leq c m n + T(q, n/2) + T(m - q, n/2)$
- Claim.  $T(m, n) \leq 2 c m n$ .
- Base cases:  $m = 2$  or  $n = 2$ .
- Inductive hypothesis:  $T(m, n) \leq 2 c m n$  for all  $(m', n')$  with  $m' + n' < m + n$ .

$$\begin{aligned} T(m, n) &\leq T(q, n/2) + T(m - q, n/2) + c m n \\ &\leq 2 c q n/2 + 2 c (m - q) n/2 + c m n \\ &= c q n + c m n - c q n + c m n \\ &= 2 c m n \blacksquare \end{aligned}$$



## SECTION 6.8

# 6. DYNAMIC PROGRAMMING II

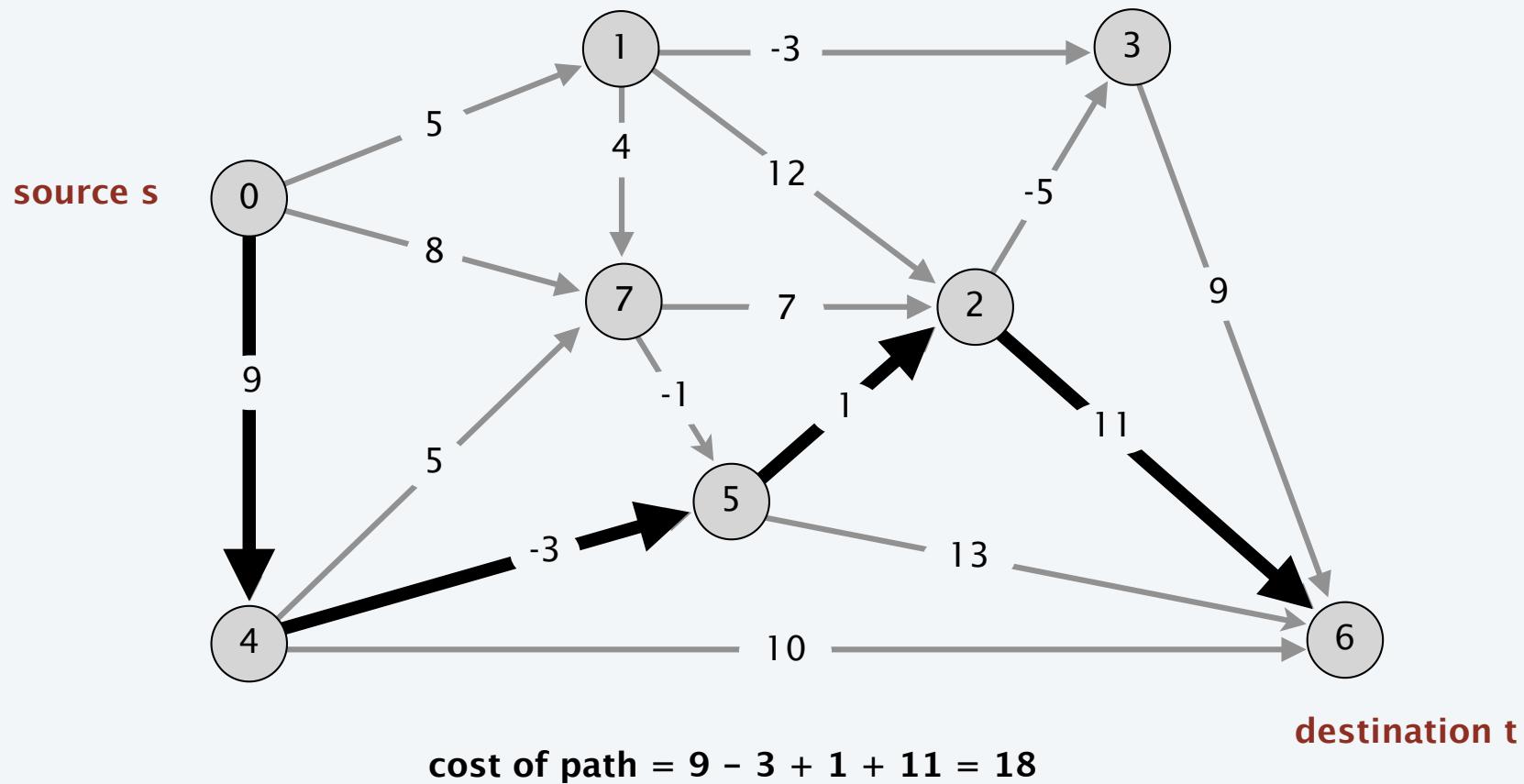
---

- ▶ *sequence alignment*
- ▶ *Hirschberg's algorithm*
- ▶ **Bellman-Ford**
- ▶ *distance vector protocols*
- ▶ *negative cycles in a digraph*

## Shortest paths

**Shortest path problem.** Given a digraph  $G = (V, E)$ , with arbitrary edge weights or costs  $c_{vw}$ , find cheapest path from node  $s$  to node  $t$ .

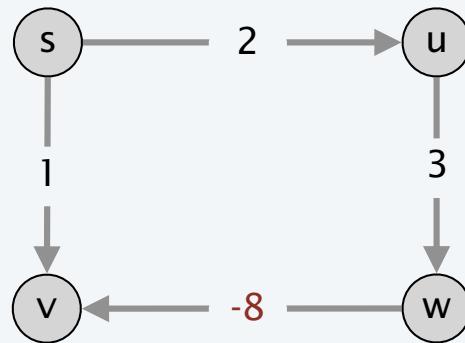
↳ also negative  $c_{vw}$



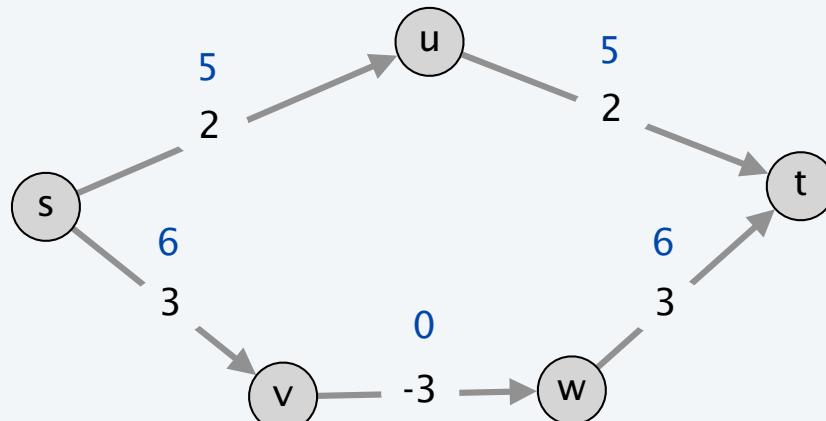
## Shortest paths: failed attempts

---

Dijkstra. Can fail if negative edge weights.

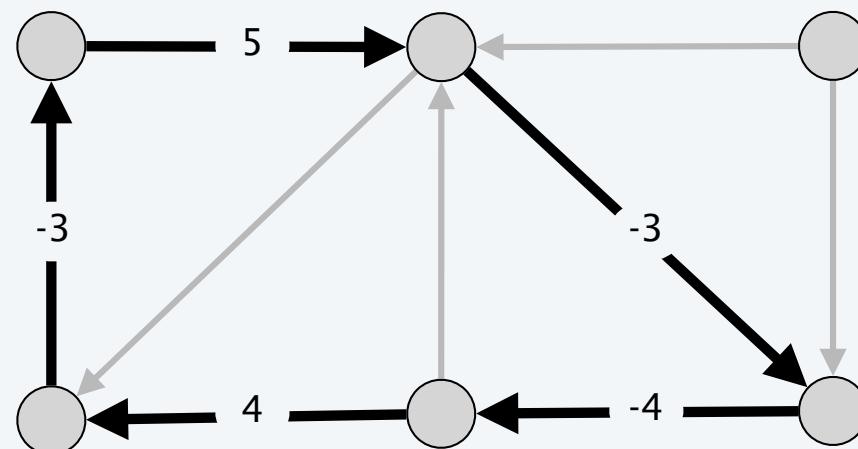


Reweighting. Adding a constant to every edge weight can fail. *Moo!*



## Negative cycles

**Def.** A **negative cycle** is a directed cycle such that the sum of its edge weights is negative.



a negative cycle  $W$  :  $c(W) = \sum_{e \in W} c_e < 0$

## Shortest paths and negative cycles

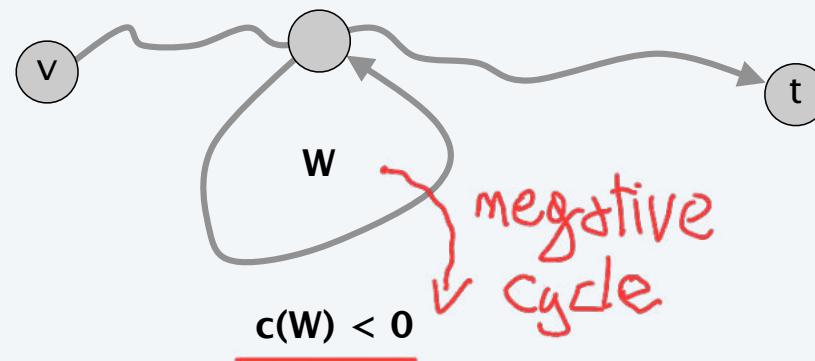
---

**Lemma 1.** If some path from  $v$  to  $t$  contains a negative cycle, then there does not exist a cheapest path from  $v$  to  $t$ .

---

**Pf.** If there exists such a cycle  $W$ , then can build a  $v \rightarrow t$  path of arbitrarily negative weight by detouring around cycle as many times as desired. ▀

---

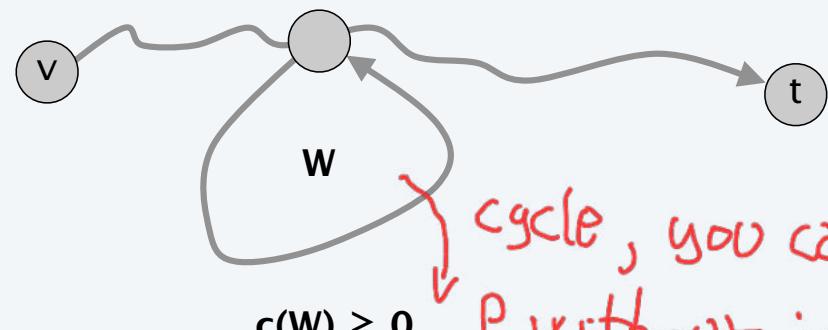


## Shortest paths and negative cycles

**Lemma 2.** If  $G$  has no negative cycles, then there exists a cheapest path from  $v$  to  $t$  that is simple (and has  $\leq n - 1$  edges).

Pf.

- Consider a cheapest  $v \rightarrow t$  path  $P$  that uses the fewest number of edges.
- If  $P$  contains a cycle  $W$ , can remove portion of  $P$  corresponding to  $W$  without increasing the cost. ■



$$c(W) \geq 0$$

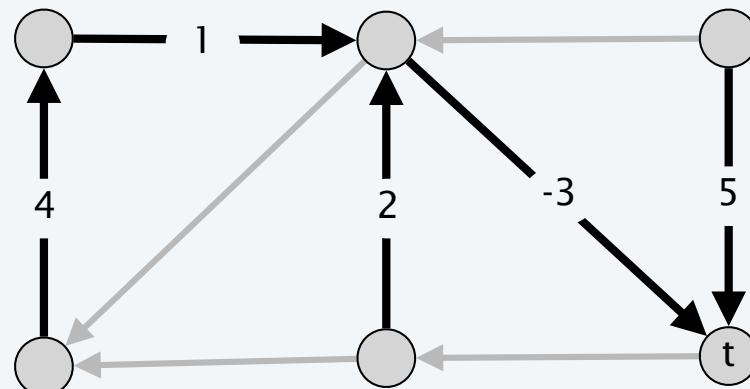
cycle, you can delete from path  
P without increase cost!

# Shortest path and negative cycle problems

**Shortest path problem.** Given a digraph  $G = (V, E)$  with edge weights  $c_{vw}$  and no negative cycles, find cheapest  $v \rightarrow t$  path for each node  $v$ .

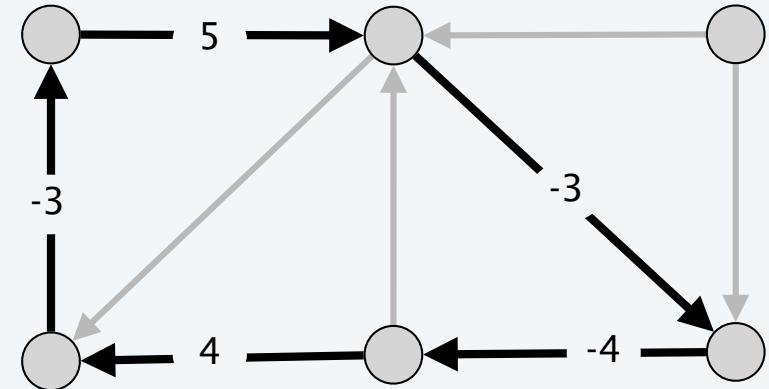
**Negative cycle problem.** Given a digraph  $G = (V, E)$  with edge weights  $c_{vw}$ , find a negative cycle (if one exists).

→ important!



shortest-paths tree

For each node these are the  
shortest paths



negative cycle

## Shortest paths: dynamic programming

up to  $m$

Def.  $OPT(i, v) = \text{cost of shortest } v \rightarrow t \text{ path that uses } \leq i \text{ edges.}$

- Case 1: Cheapest  $v \rightarrow t$  path uses  $\leq i - 1$  edges.

-  $OPT(i, v) = OPT(i - 1, v)$

optimal substructure property  
(proof via exchange argument)

- Case 2: Cheapest  $v \rightarrow t$  path uses exactly  $i$  edges.

- if  $(v, w)$  is first edge, then  $OPT$  uses  $(v, w)$ , and then selects best  $w \rightarrow t$  path using  $\leq i - 1$  edges

we can go anywhere  
↓

$$OPT(i, v) = \begin{cases} \infty & \text{if } i = 0 \\ \min \left\{ OPT(i-1, v), \min_{(v,w) \in E} \{ OPT(i-1, w) + c_{vw} \} \right\} & \text{otherwise} \end{cases}$$

Observation. If no negative cycles,  $OPT(n - 1, v) = \text{cost of cheapest } v \rightarrow t \text{ path.}$

Pf. By Lemma 2, cheapest  $v \rightarrow t$  path is simple. ■

## Shortest paths: implementation

---

$O(m^2)$  space

SHORTEST-PATHS ( $V, E, c, t$ )

FOREACH node  $v \in V$

$M[0, v] \leftarrow \infty.$

$M[0, t] \leftarrow 0.$  *cost of starting + itself*

FOR  $i = 1$  TO  $n - 1$

FOREACH node  $v \in V$

$M[i, v] \leftarrow M[i - 1, v].$

FOREACH edge  $(v, w) \in E$

$M[i, v] \leftarrow \min \{ M[i, v], M[i - 1, w] + c_{vw} \}.$

## Shortest paths: **implementation**

---

Theorem 1. Given a digraph  $G = (V, E)$  with no negative cycles, the dynamic programming algorithm computes the cost of the cheapest  $v \rightarrow t$  path for each node  $v$  in  $\Theta(mn)$  time and  $\Theta(n^2)$  space.

---

Pf.

- Table requires  $\Theta(n^2)$  space.
- Each iteration  $i$  takes  $\Theta(m)$  time since we examine each edge once. ▀

Finding the shortest paths.

- Approach 1: Maintain a  $\text{successor}(i, v)$  that points to next node on cheapest  $v \rightarrow t$  path using at most  $i$  edges.
- Approach 2: Compute optimal costs  $M[i, v]$  and consider only edges with  $M[i, v] = M[i - 1, w] + c_{vw}$ .

## Shortest paths: practical improvements

---

Space optimization. Maintain two 1d arrays (instead of 2d array).

- $d(v)$  = cost of cheapest  $v \rightarrow t$  path that we have found so far.
- $\text{successor}(v)$  = next node on a  $v \rightarrow t$  path.

Performance optimization. If  $d(w)$  was not updated in iteration  $i - 1$ , then no reason to consider edges entering  $w$  in iteration  $i$ .

# Bellman-Ford: efficient implementation

**BELLMAN-FORD** ( $V, E, c, t$ )

FOREACH node  $v \in V$

$d(v) \leftarrow \infty.$

$successor(v) \leftarrow null.$

$d(t) \leftarrow 0.$

FOR  $i = 1$  TO  $n - 1$

    FOREACH node  $w \in V$

        IF ( $d(w)$  was updated in previous iteration)

            FOREACH edge  $(v, w) \in E$

                IF ( $d(v) > d(w) + c_{vw}$ )

$d(v) \leftarrow d(w) + c_{vw}.$

$successor(v) \leftarrow w.$

        IF no  $d(w)$  value changed in iteration  $i$ , STOP.

↑

1 pass

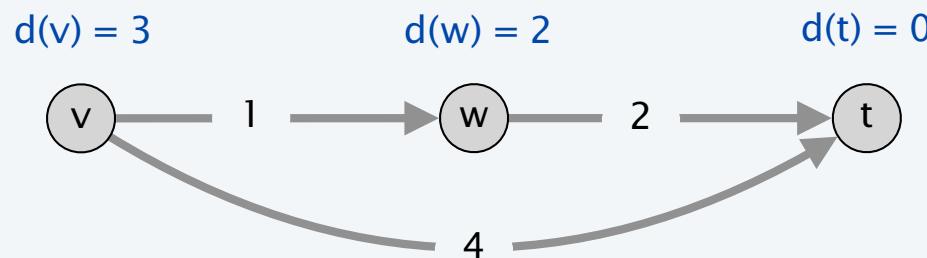
## Bellman-Ford: analysis

*is atmost*



**Claim.** After the  $i^{th}$  pass of Bellman Ford,  $d(v)$  equals the cost of the cheapest  $v \rightarrow t$  path using at most  $i$  edges.

**Counterexample.** Claim is false!



**if nodes w considered before node v,  
then  $d(v) = 3$  after 1 pass**

## Bellman-Ford: analysis

---

**Lemma 3.** Throughout Bellman-Ford algorithm,  $d(v)$  is the cost of some  $v \rightarrow t$  path; after the  $i^{th}$  pass,  $d(v)$  is no larger than the cost of the cheapest  $v \rightarrow t$  path using  $\leq i$  edges.

Pf. [by induction on  $i$ ]

- Assume true after  $i^{th}$  pass.
- Let  $P$  be any  $v \rightarrow t$  path with  $i + 1$  edges.
- Let  $(v, w)$  be first edge on path and let  $P'$  be subpath from  $w$  to  $t$ .
- By inductive hypothesis,  $d(w) \leq c(P')$  since  $P'$  is a  $w \rightarrow t$  path with  $i$  edges.
- After considering  $v$  in pass  $i+1$ :  
$$\begin{aligned} d(v) &\leq c_{vw} + d(w) \\ &\leq c_{vw} + c(P') \\ &= c(P) \quad \blacksquare \end{aligned}$$

**Theorem 2.** Given a digraph with no negative cycles, Bellman-Ford computes the costs of the cheapest  $v \rightarrow t$  paths in  $O(mn)$  time and  $\Theta(n)$  extra space.

Pf. Lemmas 2 + 3. ■

can be substantially  
faster in practice

## Bellman-Ford: analysis

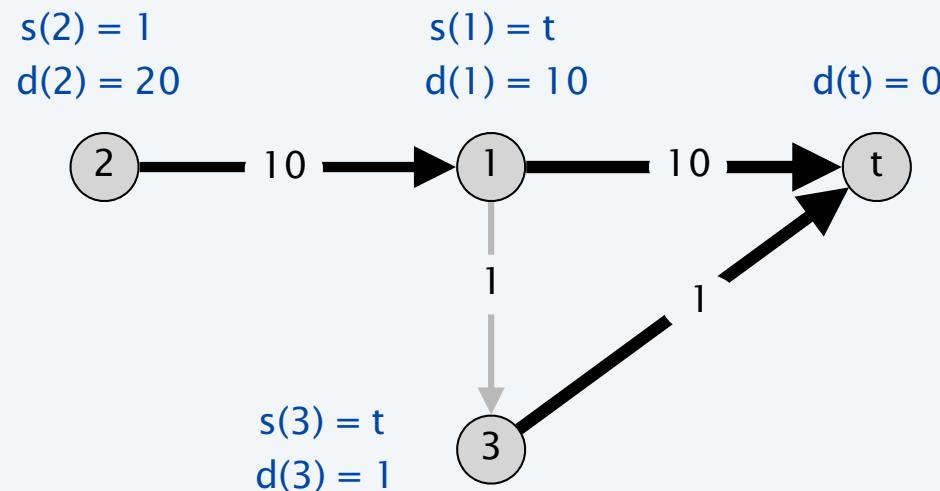
---

**Claim.** ~~Throughout the Bellman Ford algorithm, following  $\text{successor}(v)$  pointers gives a directed path from  $v$  to  $t$  of cost  $d(v)$ .~~

**Counterexample.** Claim is false!

- Cost of successor  $v \rightarrow t$  path may have strictly lower cost than  $d(v)$ .

consider nodes in order:  $t, 1, 2, 3$



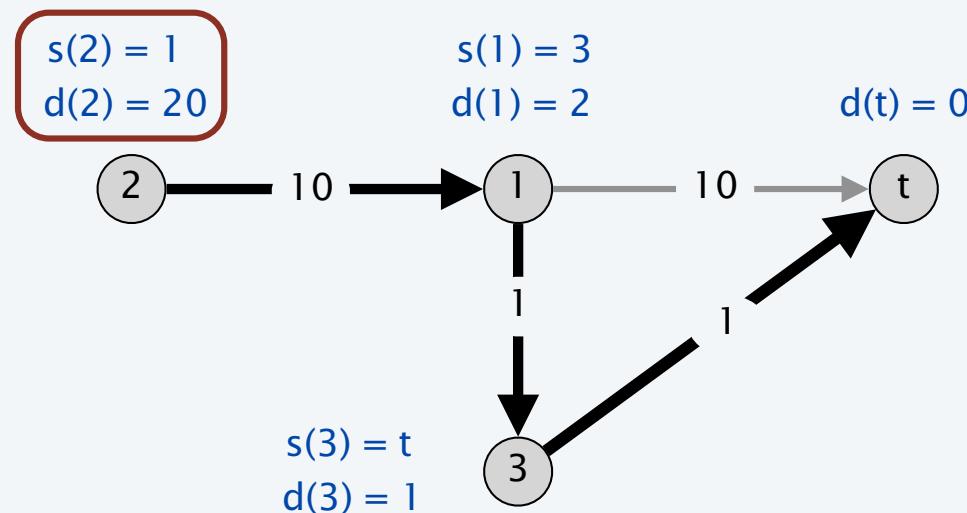
## Bellman-Ford: analysis

**Claim.** ~~Throughout the Bellman Ford algorithm, following  $\text{successor}(v)$  pointers gives a directed path from  $v$  to  $t$  of cost  $d(v)$ .~~

**Counterexample.** Claim is false!

- Cost of successor  $v \rightarrow t$  path may have strictly lower cost than  $d(v)$ .

consider nodes in order:  $t, 1, 2, 3$



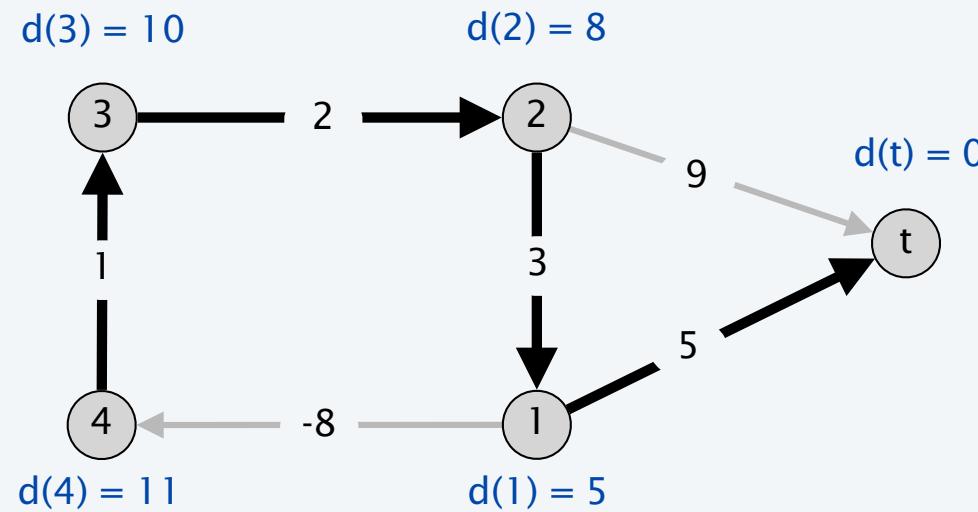
## Bellman-Ford: analysis

**Claim.** ~~Throughout the Bellman Ford algorithm, following  $\text{successor}(v)$  pointers gives a directed path from  $v$  to  $t$  of cost  $d(v)$ .~~

**Counterexample.** Claim is false!

- Cost of successor  $v \rightarrow t$  path may have strictly lower cost than  $d(v)$ .
- Successor graph may have cycles.

consider nodes in order:  $t, 1, 2, 3, 4$



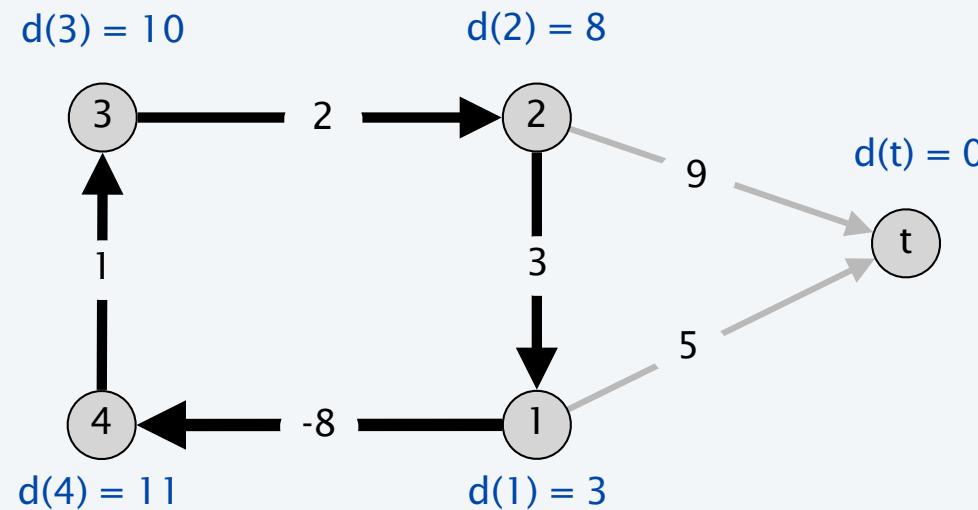
## Bellman-Ford: analysis

**Claim.** ~~Throughout the Bellman Ford algorithm, following  $\text{successor}(v)$  pointers gives a directed path from  $v$  to  $t$  of cost  $d(v)$ .~~

**Counterexample.** Claim is false!

- Cost of successor  $v \rightarrow t$  path may have strictly lower cost than  $d(v)$ .
- Successor graph may have cycles.

consider nodes in order:  $t, 1, 2, 3, 4$



## Bellman-Ford: finding the shortest path

---

**Lemma 4.** If the successor graph contains a directed cycle  $W$ , then  $W$  is a negative cycle.

Pf.

- If  $\text{successor}(v) = w$ , we must have  $d(v) \geq d(w) + c_{vw}$ .  
(LHS and RHS are equal when  $\text{successor}(v)$  is set;  $d(w)$  can only decrease;  $d(v)$  decreases only when  $\text{successor}(v)$  is reset)
- Let  $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$  be the nodes along the cycle  $W$ .
- Assume that  $(v_k, v_1)$  is the last edge added to the successor graph.
- Just prior to that:  
$$\begin{aligned} d(v_1) &\geq d(v_2) + c(v_1, v_2) \\ d(v_2) &\geq d(v_3) + c(v_2, v_3) \\ &\vdots &&\vdots \\ d(v_{k-1}) &\geq d(v_k) + c(v_{k-1}, v_k) \\ d(v_k) &> d(v_1) + c(v_k, v_1) \end{aligned}$$
← holds with strict inequality since we are updating  $d(v_k)$

- Adding inequalities yields  $c(v_1, v_2) + c(v_2, v_3) + \dots + c(v_{k-1}, v_k) + c(v_k, v_1) < 0$ . ■



W is a negative cycle

## Bellman-Ford: finding the shortest path

**Theorem 3.** Given a digraph with no negative cycles, Bellman-Ford finds the cheapest  $s \rightarrow t$  paths in  $O(mn)$  time and  $\Theta(n)$  extra space.

Pf.

- The successor graph cannot have a negative cycle. [Lemma 4]
- Thus, following the successor pointers from  $s$  yields a directed path to  $t$ .
- Let  $s = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k = t$  be the nodes along this path  $P$ .
- Upon termination, if  $\text{successor}(v) = w$ , we must have  $d(v) = d(w) + c_{vw}$ .  
(LHS and RHS are equal when  $\text{successor}(v)$  is set;  $d(\cdot)$  did not change)
- Thus,  
$$\begin{aligned} d(v_1) &= d(v_2) + c(v_1, v_2) \\ d(v_2) &= d(v_3) + c(v_2, v_3) \\ &\vdots && \vdots \\ d(v_{k-1}) &= d(v_k) + c(v_{k-1}, v_k) \end{aligned}$$
- since algorithm terminated

Adding equations yields  $d(s) = d(t) + c(v_1, v_2) + c(v_2, v_3) + \dots + c(v_{k-1}, v_k)$ . ▀



## LONGEST INCREASING SUBSEQUENCES

$\text{length} = 3$

5 2 8 6 3 6 9 7      v      5 2 8 6 9 6 9 7       $\Rightarrow \text{length} = 4$

$\Rightarrow \text{SOLUZIONE}$

Trovare una sottosequenza che è incrementale ed è la più lunga

$m = \# \text{ numeri}$      $i = \# \text{ indice di un numero}$      $i \leq m$

$\text{ALG}(i) = 1 + \max \{ \text{ALG}(j) \}$

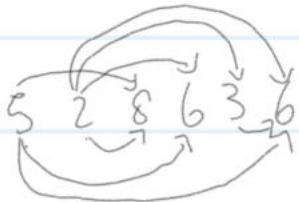
$\downarrow$        $\downarrow$        $\downarrow$        $\downarrow$        $\downarrow$

SOLUTION FROM POSITION 1 to i       $\text{ALG}(i)$  mi dice dove le lunghezze delle sequenze

$\downarrow$

perché settemo 1 perché settemo il LARGEST senza più lunghezze

ogni iterazione, se l'elemento viene incluso nella soluzione, la lunghezza aumenta di 1



$$\text{ALG}(1) = 1$$

$$\text{ALG}(2) = 1$$

$$\text{ALG}(3) = 2$$

$$\text{ALG}(4) = 2$$

$$\text{ALG}(5) = 2$$

$$\text{ALG}(6) = 3$$

$$\text{ALG}(7) = 4$$

$$\text{ALG}(8) = 4$$

Part 1 dynamic programming: paradigm where you try to build up the solution slowly, by remembering solution by earlier of smaller problems.

## The Chinese Sticks Problem

# The Chinese Sticks Problem

(Based on the description at this website.)

Assume you have a long stick that needs to be cut into shorter pieces. The cuts must be done in certain parts of the stick. You must cut the stick in all marked places. For example, consider a stick of size 10 and 3 places marked in it (positions 2, 4 and 7) where the cuts must be made.

The cost to make a cut is the same as the size of the stick. For instance, if you have a stick of size 10 and you need to make one cut (anywhere) in it, that will cost you 10.

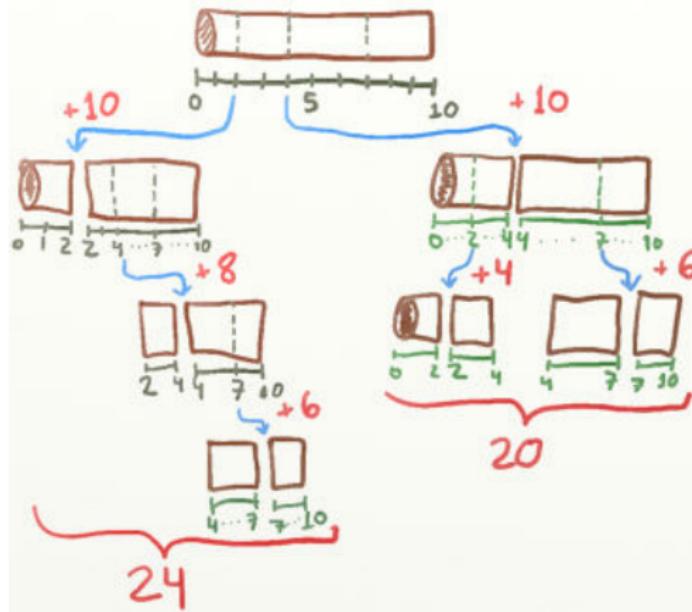
**Goal:** Find a cheapest way of making all necessary cuts.

## Example

Cutting on positions 2, 4 and 7 (one after the other), the total cost will be  $10+8+6=24$ .

A better way to cut that stick would be:

Start on position 4, then 2 then 7. Now the cost is  $10+4+6=20$ .



## More Formal Problem Definition

Given a stick of length  $N$ , and cutting places  $k_1, k_2, \dots, k_n$ , find an ordering  $S = (k_{i_1}, k_{i_2}, \dots, k_{i_n})$  of all  $k_i$  such that the total cost

$$\text{cost}(S) = \sum_{j=1}^n \text{Current length of the stick when cutting at } k_{i_j}$$

is minimized.

## Key Observation

Whenever we cut the sticks, we are left with *two* shorter ones that might have to be cut again. The optimal cost for cutting in all the marked places of the original stick equals the optimal costs from the two smaller sticks, plus the length of the original stick.

Recursive function idea

```
optimalCost(initialStickPosition, endStickPosition) =  
    optimalCost(initialStickPosition, k) +  
    optimalCost(k, endStickPosition) +  
    sizeOfTheCurrentStick
```

## Recursive Formula

We can replace the stick size by the distance between beginning and end of the stick.

```
optimalCost(initialStickPosition, endStickPosition) =  
optimalCost(initialStickPosition, k) +  
optimalCost(k, endStickPosition) +  
(endStickPosition - initialStickPosition)
```

# Recursive Algorithm

```
1 int bestCut = INF
2 for(int k=indexOfFirstCut, k<=indexOfLastCut, k=nextCutIndex()){
3     int subCutCost = optimalCost(0, N, k);
4     if(subCutCost < bestCut){
5         bestCut = subCutCost;
6     }
7 }
8
9 int optimalCost(int initialStickPosition,
10                 int endStickPosition,
11                 int k){
12     return optimalCost(initialStickPosition, k) +
13             optimalCost(k, endStickPosition) +
14             (endStickPosition - initialStickPosition)
15 }
```

## Problem: Computation Time

Sadly, a lot of possibilities are calculated over and over again.

But, we are only interested in the *best-possible* solution for each sub-stick!

→ Store this optimum value for every sub-stick so we can just look it up!

## Making the Algorithm more Efficient

```
1 int optimalCost(int initialStickPosition,
2                 int endStickPosition,
3                 int k){
4
5     int memorizedResult = m[initialStickPosition][endStickPosition];
6     if(memorizedResult != INF){
7         return memorizedResult;
8     }
9     int bestCost = optimalCost(initialStickPosition, k) +
10                optimalCost(k, endStickPosition) +
11                (endStickPosition - initialStickPosition)
12
13    m[initialStickPosition][endStickPosition] = bestCost;
14    return bestCost;
15 }
```

## Running Time / Space

At worst, we compute an optimum cutting for every combination of initial and end- stick position.

Those are chosen from the  $n$  cutting positions, plus  $\{0, N\}$ .

→ we have  $O(n^2)$  values to compute and store.

Each actual computation of a value in the matrix  $m$  requires to compare all up to  $n$  possibilities of cutting this stick in two (except that for shorter sticks, it will of course be less than  $n$ ).

## Part 2

# Maximum Weighted Independent Set on a Path

# Maximum Weighted Independent Set on a Path

Given are a set  $V$  of vertices on a path, and weights  $w(v) \geq 0$  for each vertex  $v \in V$ .

We define a subset  $I \subseteq V$  to be an *Independent Set* if for any two vertices  $v_1, v_2 \in I$ , there exists *no edge* between  $v_1$  and  $v_2$ .

**Goal:** Find a maximum-weight independent set in  $V$ .

## Problem Structure

An optimal solution can have one of two properties:

- ▶ Either, the last element in the path is not part of the maximum weighted independent set  
(then, we know that the solution is equally valid for the subgraph  $V'$  formed by deleting the last vertex from the path)
- ▶ Or the last element is part of the set  
(then, we know that the predecessor cannot be part of the set, and the solution minus the last vertex is equally valid for the subgraph  $V'$  formed by deleting the last two vertices from the path)

## Time Considerations

If we knew which of the above options is true (last vertex in or out?), we could walk backwards in linear number of steps!

As previously for Chinese Sticks, doing this the naive way is costly.  
Trying out both options recursively results in exponential running time.

**Solution:** Again, remember the result of past computations and look it up instead of doing it all over again!

# Algorithm Structure

- ▶ Walk through the path from left to right, and use calculated weights to decide whether or not the last vertex is in the set.
- ▶ **Base Case:**
  - ▶ For empty sets, their weight is zero.
  - ▶ For any set  $S = \{v\}$ ,  $v \in V$ , the weight of the set is  $w(v)$ .  
Store  $w(v)$  in matrix entry  $A[0]$ .
- ▶ **General Case** (for the  $i$ -th vertex  $v_i$ ):
  - ▶ We can either choose the old solution up to the last point, i.e.  $A[i - 1]$ , or the  $A[i - 2]$  solution (that can never pick the vertex next to the  $i$ -th), together with the  $i$ -th vertex.
  - ▶ Store in  $A[i]$  the maximum over  $A[i - 1]$  and  $A[i - 2] + w(v_i)$ .

## Algorithm Properties

The algorithm runs in linear time: it picks a maximum between two values for every vertex on the path.

Although this gives us only the weight of a maximum weight independent set, not the set itself, we can easily adjust the algorithm:

- ▶ For every step from left to right along the path, we could also store the selected vertices.
- ▶ Or, we go backwards along the steps of the original algorithm to *reconstruct* the solution.

The time complexity is preserved also when the goal is to find the actual independent set.

# Algorithm for Reconstruction of the Independent Set

In pseudocode, we could get the set like this:

$S = \emptyset$

$v_i = \text{last element in } A$

**While**  $i \geq 1$ :

**If**  $A[i] = A[i - 1]$  // Last vertex not in set  $S$

$i --$

**Else** // Last vertex is in set  $S$

$S = S \cup v_i$

$i = i - 2$

**End**

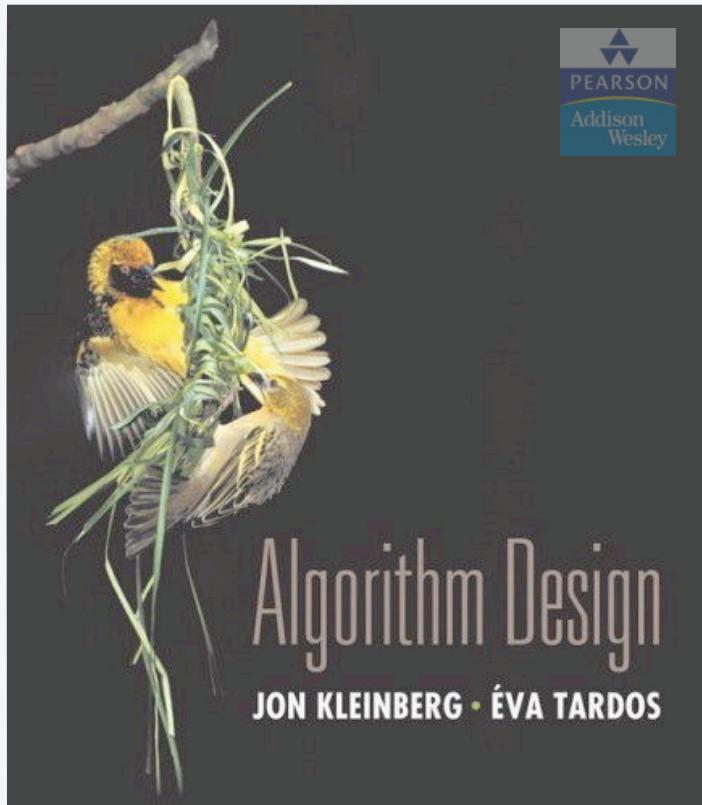
**End**

Return  $S$ .

## Segmented Least Squares

### Segmented Least Squares Problem

Slides are the ones belonging to the book, and can be found [here](#).



## SECTION 7.1

# 7. NETWORK FLOW I

---

- ▶ *max-flow and min-cut problems*
- ▶ *Ford-Fulkerson algorithm*
- ▶ *max-flow min-cut theorem*
- ▶ *capacity-scaling algorithm*
- ▶ *shortest augmenting paths*
- ▶ *blocking-flow algorithm*
- ▶ *unit-capacity simple networks*

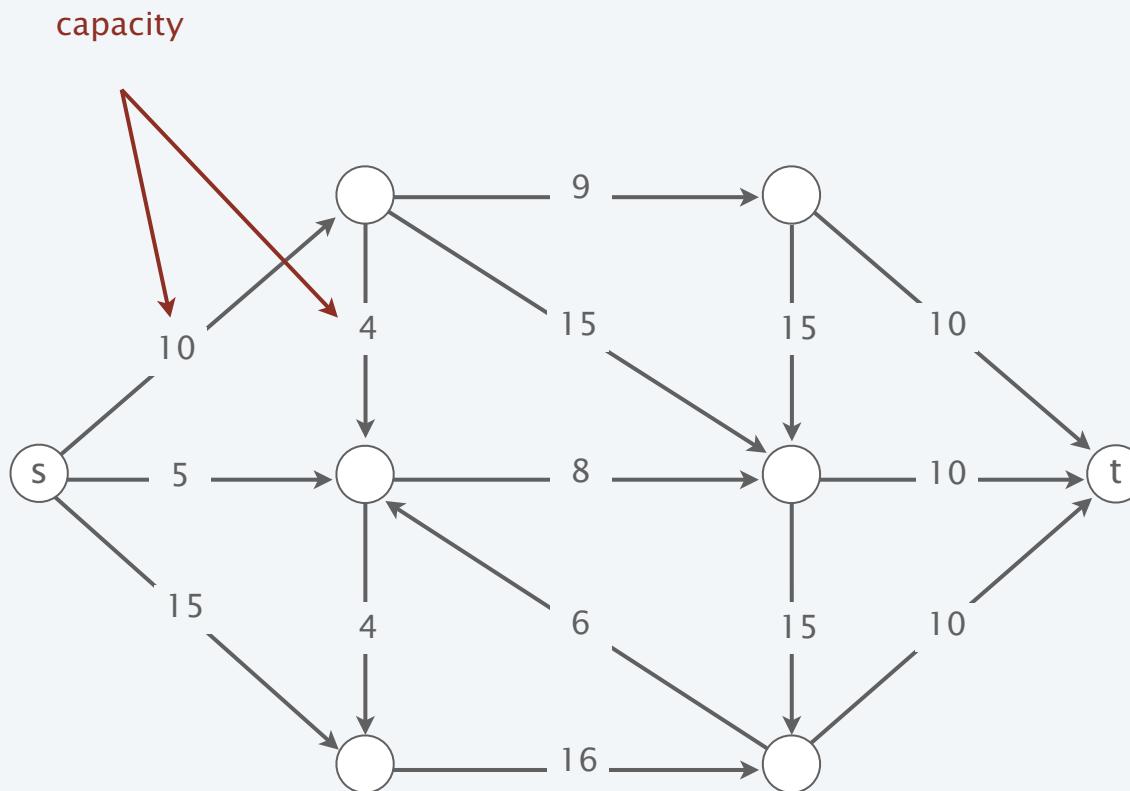
longest increasing subsequences

## Flow network

like water, gas...

- Abstraction for material **flowing** through the edges.
- Digraph  $G = (V, E)$  with source  $s \in V$  and sink  $t \in V$ .
- Nonnegative integer capacity  $c(e)$  for each  $e \in E$ .

no parallel edges  
no edge enters **source**  
no edge leaves **target**



Minimum cut problem ↵ max flow problem is related to cut problem

Def. A  $s-t$ -cut (cut) is a partition  $(A, B)$  of the vertices with  $s \in A$  and  $t \in B$ .

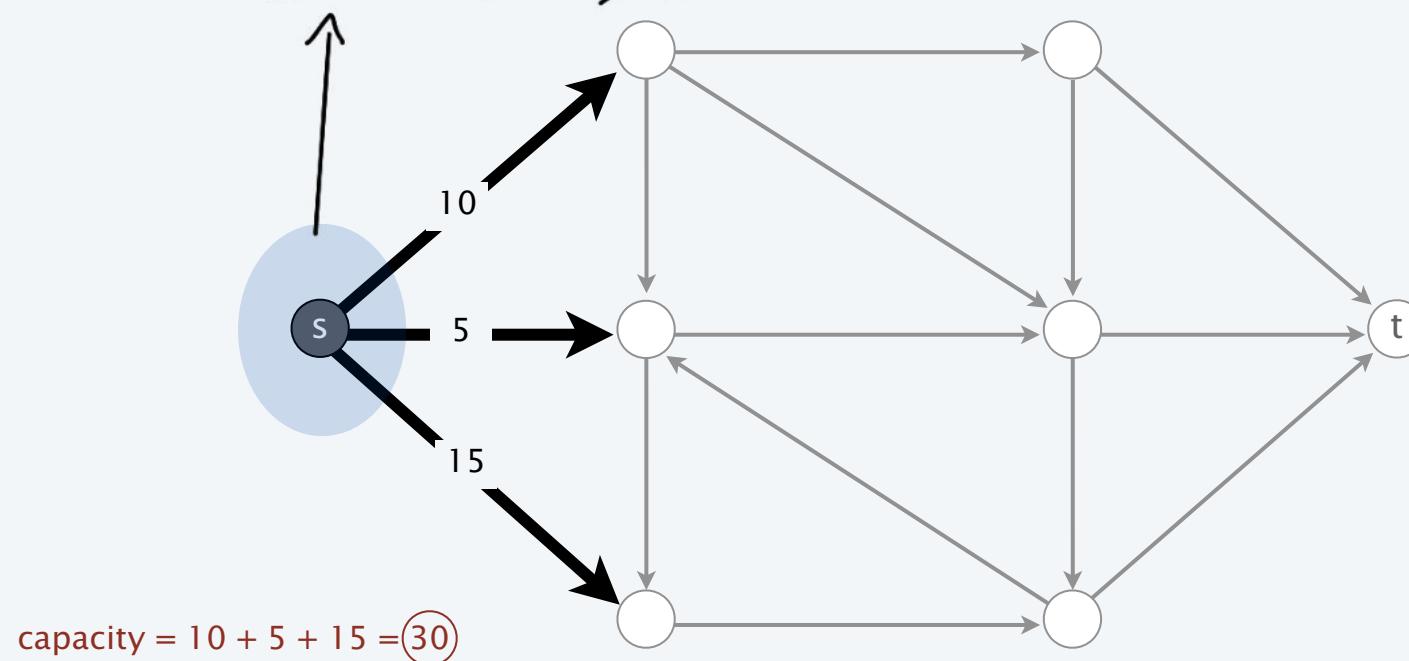
$\hookrightarrow B$  contain at least target, but not  $s$

$\hookrightarrow A$  contain at least source, but not  $t$

Def. Its capacity is the sum of the capacities of the edges from  $A$  to  $B$ .

$$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$

cut with  $A = S$ ,  $B = \text{Vertices} - s$

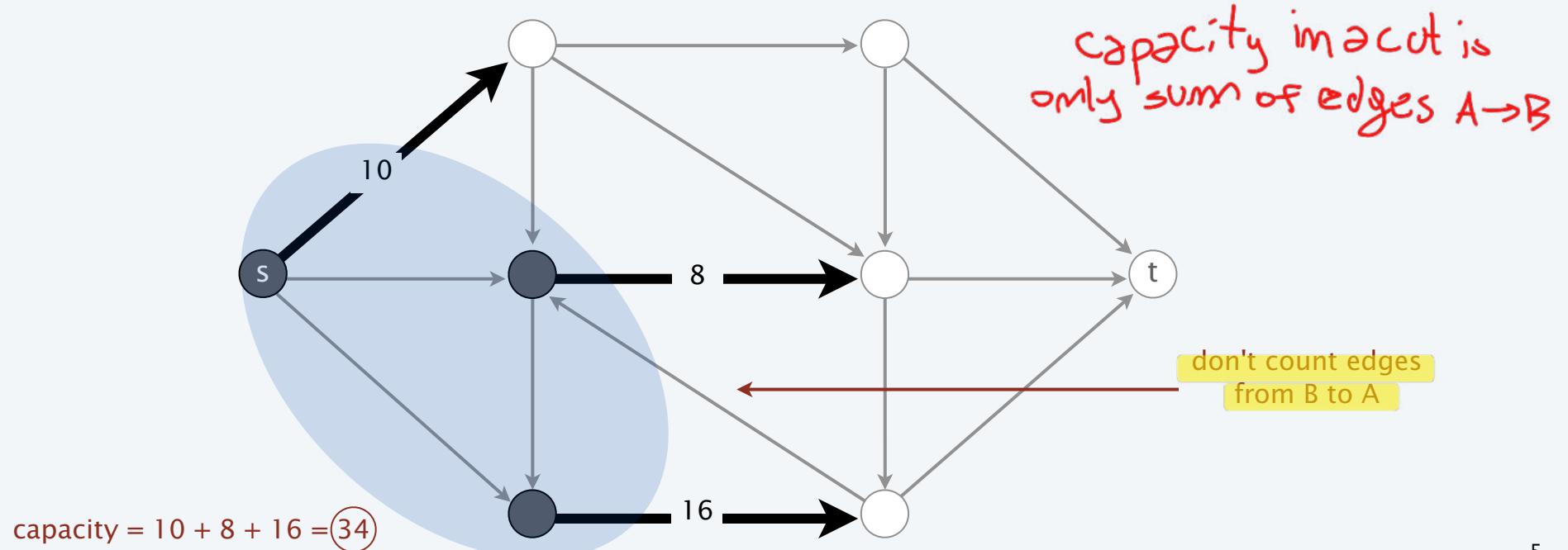


# Minimum cut problem

Def. A *st-cut (cut)* is a partition  $(A, B)$  of the vertices with  $s \in A$  and  $t \in B$ .

Def. Its **capacity** is the sum of the capacities of the edges from  $A$  to  $B$ .

$$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$



# Minimum cut problem

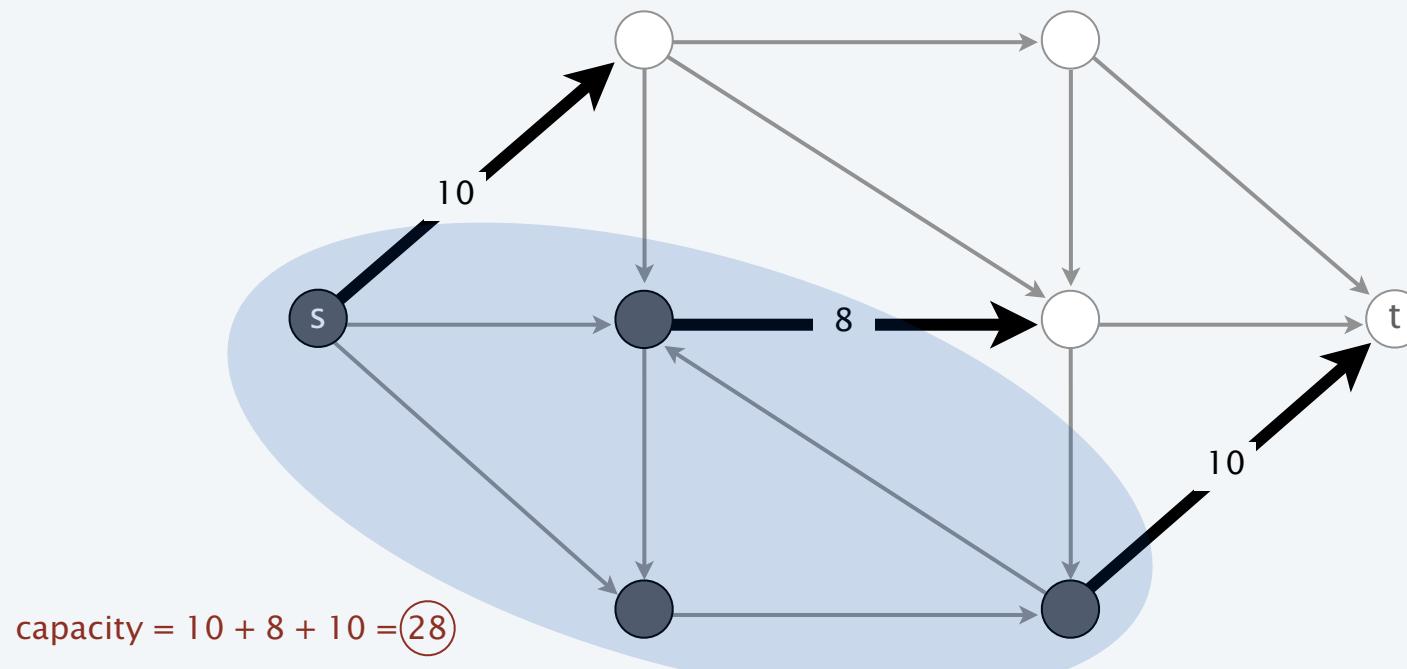
Def. A *st-cut (cut)* is a partition  $(A, B)$  of the vertices with  $s \in A$  and  $t \in B$ .  
↪ a collection of vertices, a set

Def. Its capacity is the sum of the capacities of the edges from  $A$  to  $B$ .

$$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$

↪ partition of vertex, A grey vertex with  $s$ , B is white with  $t$

Min-cut problem. Find a cut of minimum capacity.



## Maximum flow problem

Def. An  $st$ -flow (flow)  $f$  is a function that satisfies:

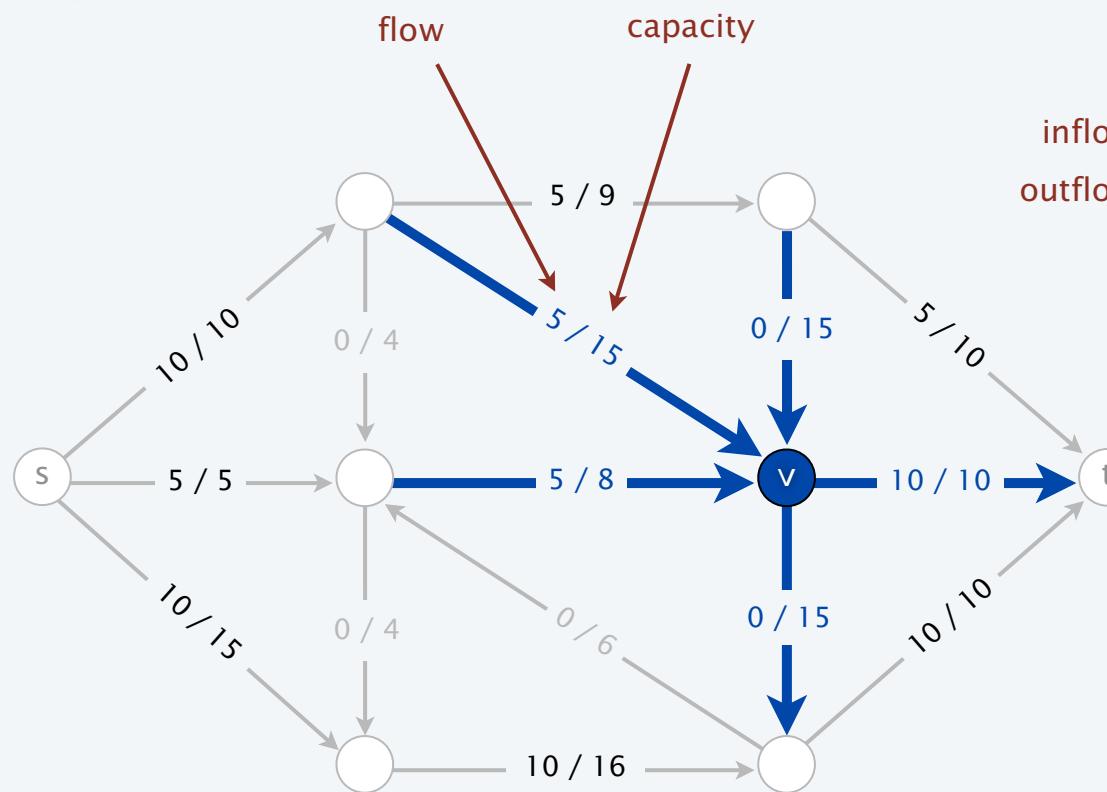
- For each  $e \in E$ :  $0 \leq f(e) \leq c(e)$  [capacity]

- For each  $v \in V - \{s, t\}$ :  $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$  [flow conservation]

→ can't transport negative flow

• flow exit to  $s$  is equal to flow enter to  $t$   
for flow conservation

↳ incoming flow of  $v$  is equal to outgoing



$$\begin{aligned} \text{inflow at } v &= 5 + 5 + 0 = 10 \\ \text{outflow at } v &= 10 + 0 = 10 \end{aligned}$$

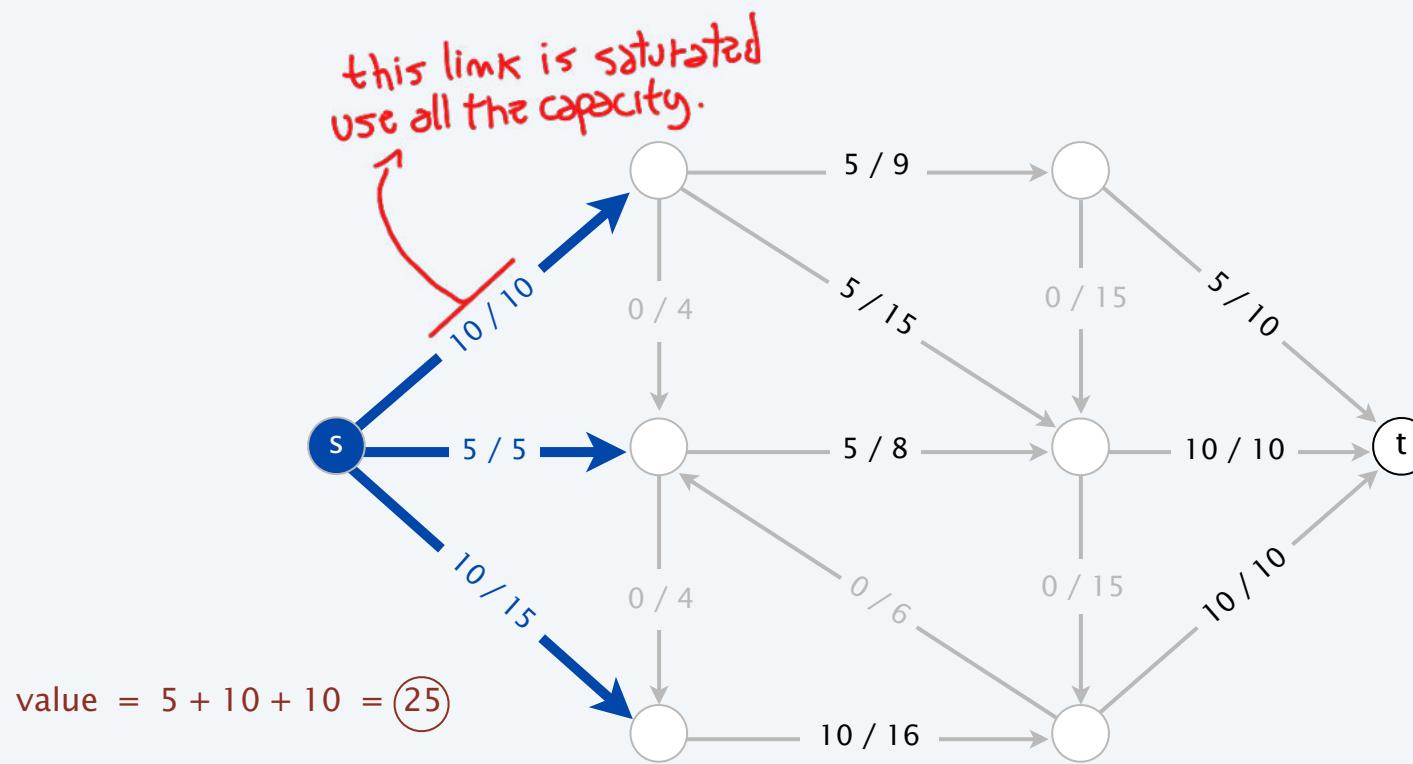
$$f(s) = f(t)$$

# Maximum flow problem

Def. An  $st$ -flow (flow)  $f$  is a function that satisfies:

- For each  $e \in E$ :  $0 \leq f(e) \leq c(e)$  [capacity]
- For each  $v \in V - \{s, t\}$ :  $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$  [flow conservation]

Def. The value of a flow  $f$  is:  $val(f) = \sum_{e \text{ out of } s} f(e)$ . *outcoming flow from source  
of incoming flow in target*



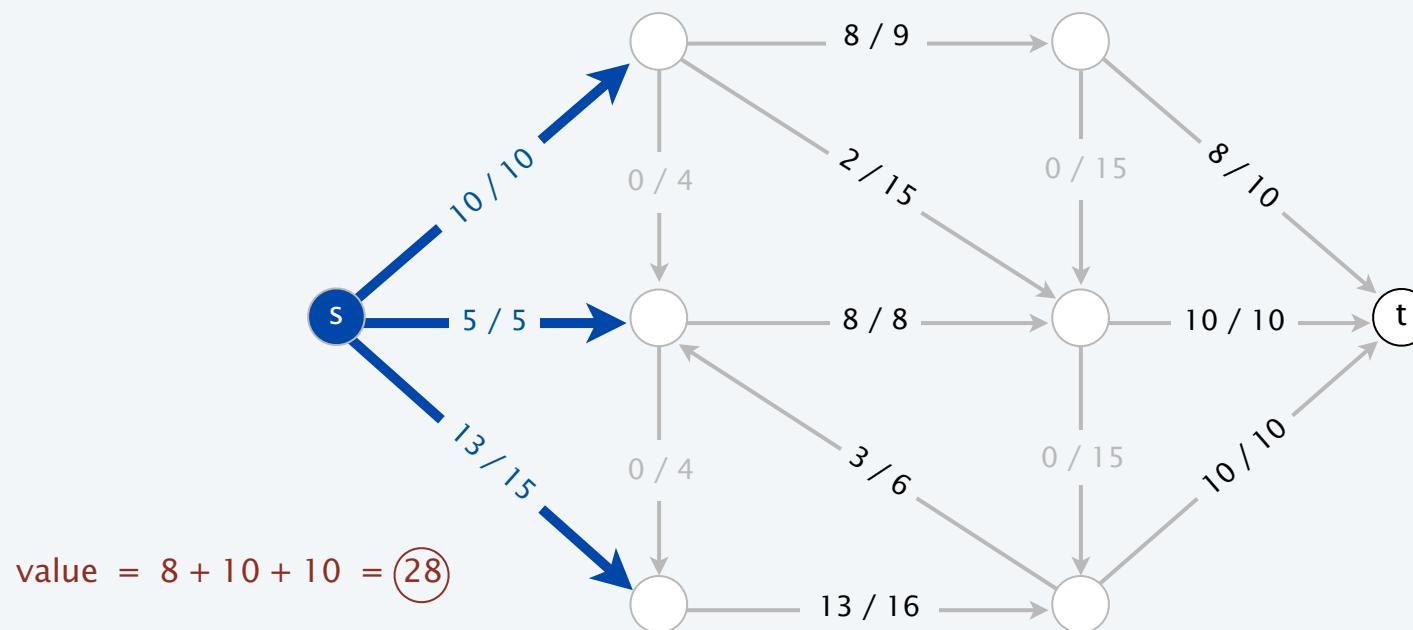
# Maximum flow problem

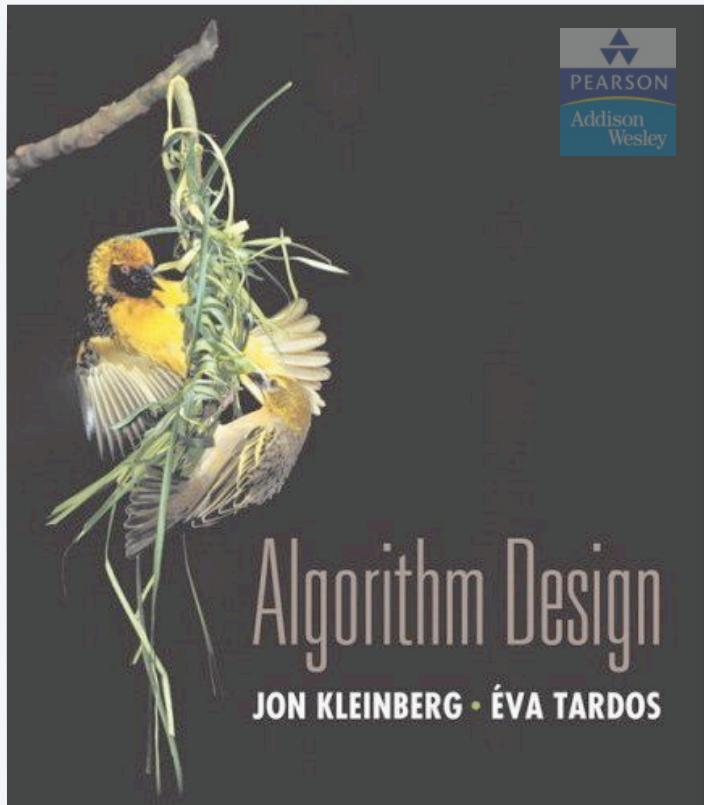
Def. An *st*-flow (flow)  $f$  is a function that satisfies:

- For each  $e \in E$ :  $0 \leq f(e) \leq c(e)$  [capacity]
- For each  $v \in V - \{s, t\}$ :  $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$  [flow conservation]

Def. The **value** of a flow  $f$  is:  $val(f) = \sum_{e \text{ out of } s} f(e)$ .

Max-flow problem. Find a flow of maximum value.





## SECTION 7.1

# 7. NETWORK FLOW I

---

- ▶ *max-flow and min-cut problems*
- ▶ ***Ford-Fulkerson algorithm***
- ▶ *max-flow min-cut theorem*
- ▶ *capacity-scaling algorithm*
- ▶ *shortest augmenting paths*
- ▶ *blocking-flow algorithm*
- ▶ *unit-capacity simple networks*

## Towards a max-flow algorithm

iteratively pushing flow from source to destination on paths that still accepts at least some minimum amount of flow.

Greedy algorithm. *not optimal*

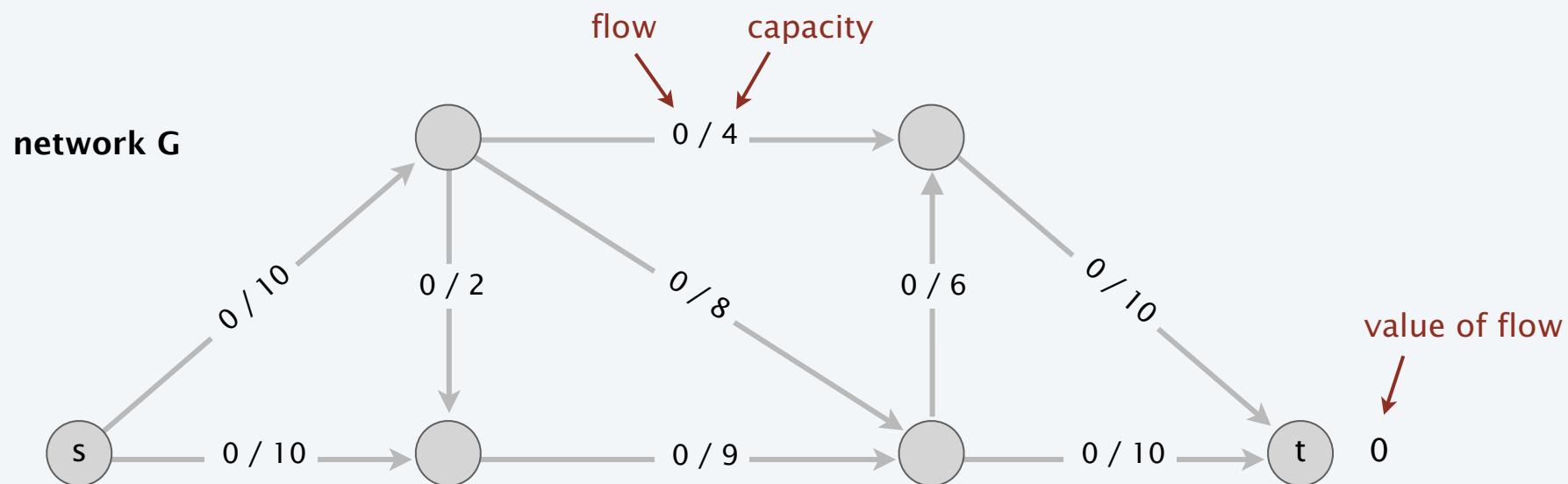
- Start with  $f(e) = 0$  for all edge  $e \in E$ .

- Find an  $s \rightarrow t$  path  $P$  where each edge has  $f(e) < c(e)$ .

- Augment flow along path  $P$ .

- Repeat until you get stuck.

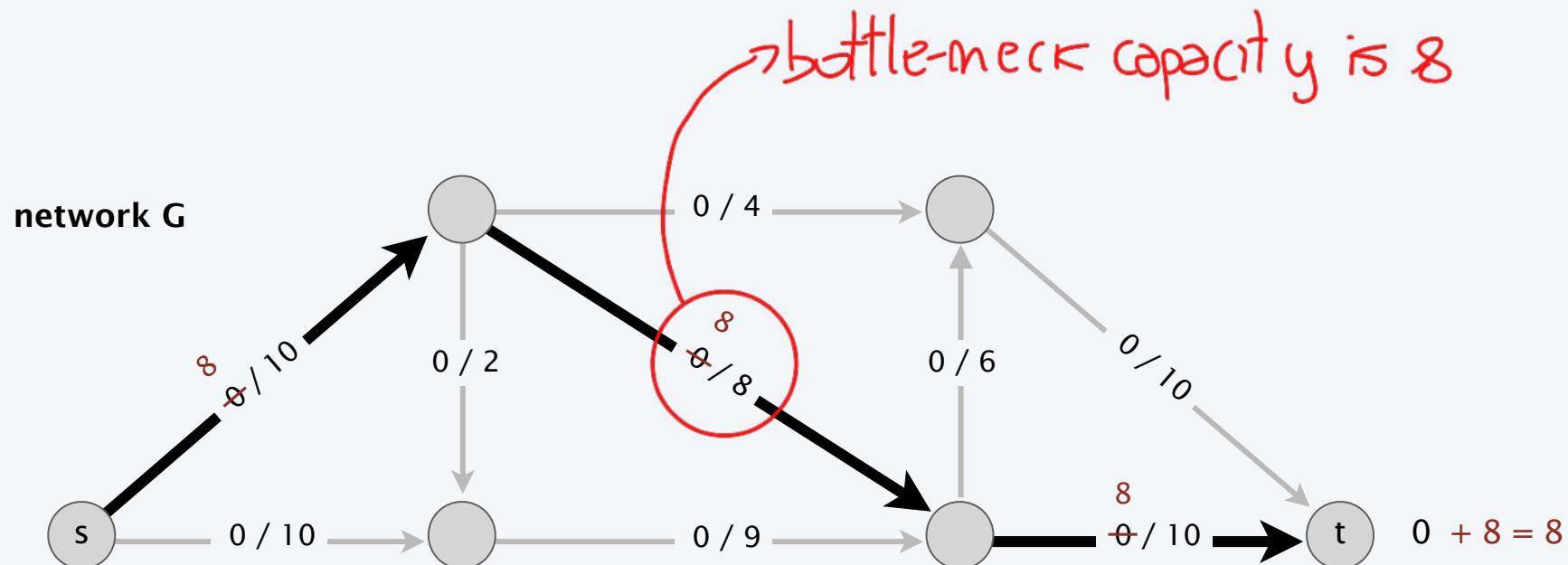
↳ find a path from source to destination where there is some remaining unused capacity, and push amount of flow that is equal to minimum unused capacity of the link on the path. (bottle-neck capacity)



# Towards a max-flow algorithm

## Greedy algorithm.

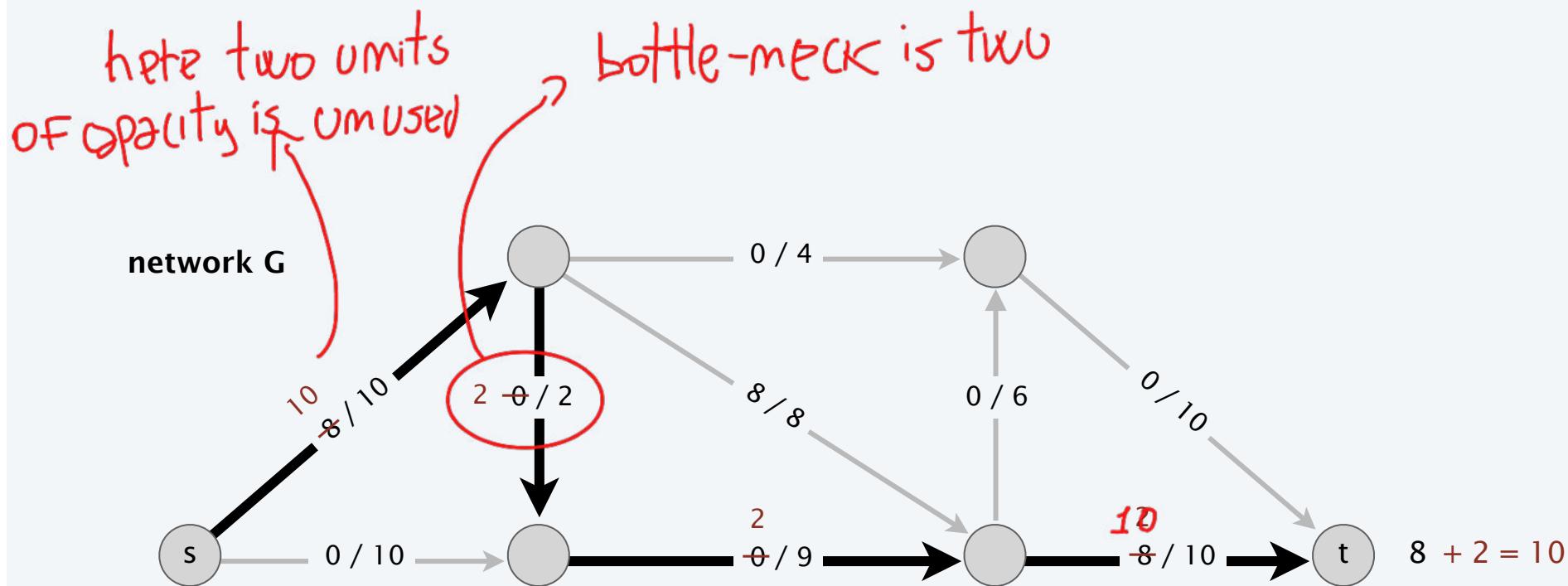
- Start with  $f(e) = 0$  for all edge  $e \in E$ .
- Find an  $s \rightarrow t$  path  $P$  where each edge has  $f(e) < c(e)$ .
- Augment flow along path  $P$ .
- Repeat until you get stuck.



# Towards a max-flow algorithm

## Greedy algorithm.

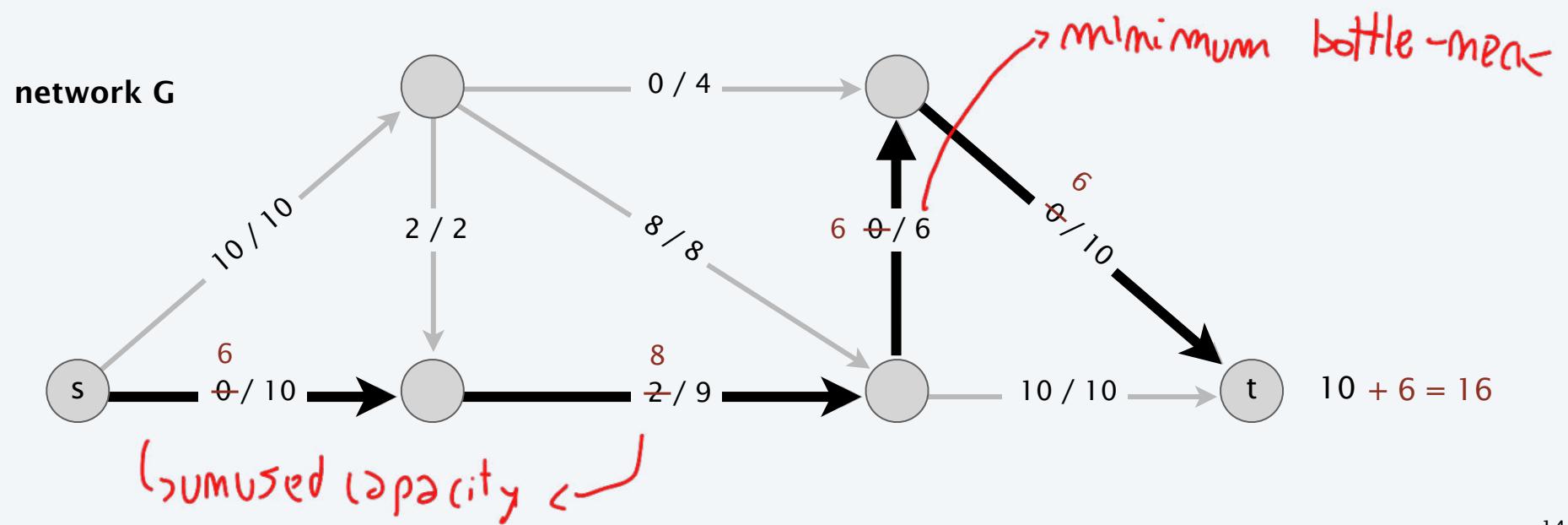
- Start with  $f(e) = 0$  for all edge  $e \in E$ .
- Find an  $s \rightarrow t$  path  $P$  where each edge has  $f(e) < c(e)$ .
- Augment flow along path  $P$ .
- Repeat until you get stuck.



# Towards a max-flow algorithm

## Greedy algorithm.

- Start with  $f(e) = 0$  for all edge  $e \in E$ .
- Find an  $s \rightarrow t$  path  $P$  where each edge has  $f(e) < c(e)$ .
- Augment flow along path  $P$ .
- Repeat until you get stuck.



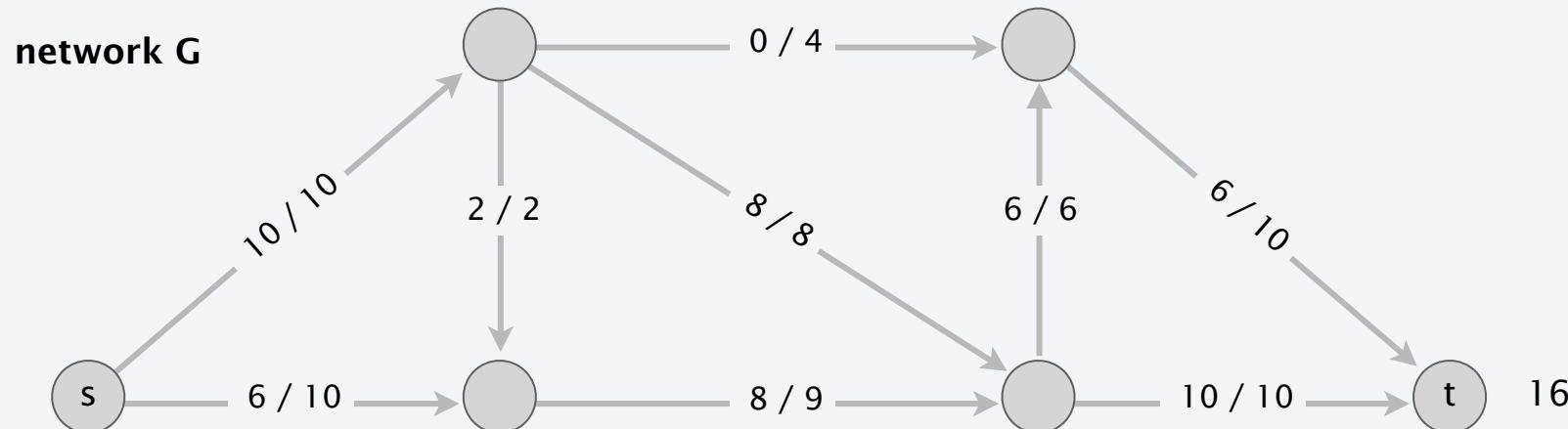
# Towards a max-flow algorithm

Greedy algorithm.

- Start with  $f(e) = 0$  for all edge  $e \in E$ .
- Find an  $s \rightarrow t$  path  $P$  where each edge has  $f(e) < c(e)$ .
- Augment flow along path  $P$ .
- Repeat until you get stuck.

*we want to correct the suboptimal decision made at early stage of the algorithm by reversing direction of the flow on a link. Find augmenting path not only bringing flow to destination through link with unused capacity, but also correct eventually previous wrong augmenting path by reducing the amount of flow on a link.*

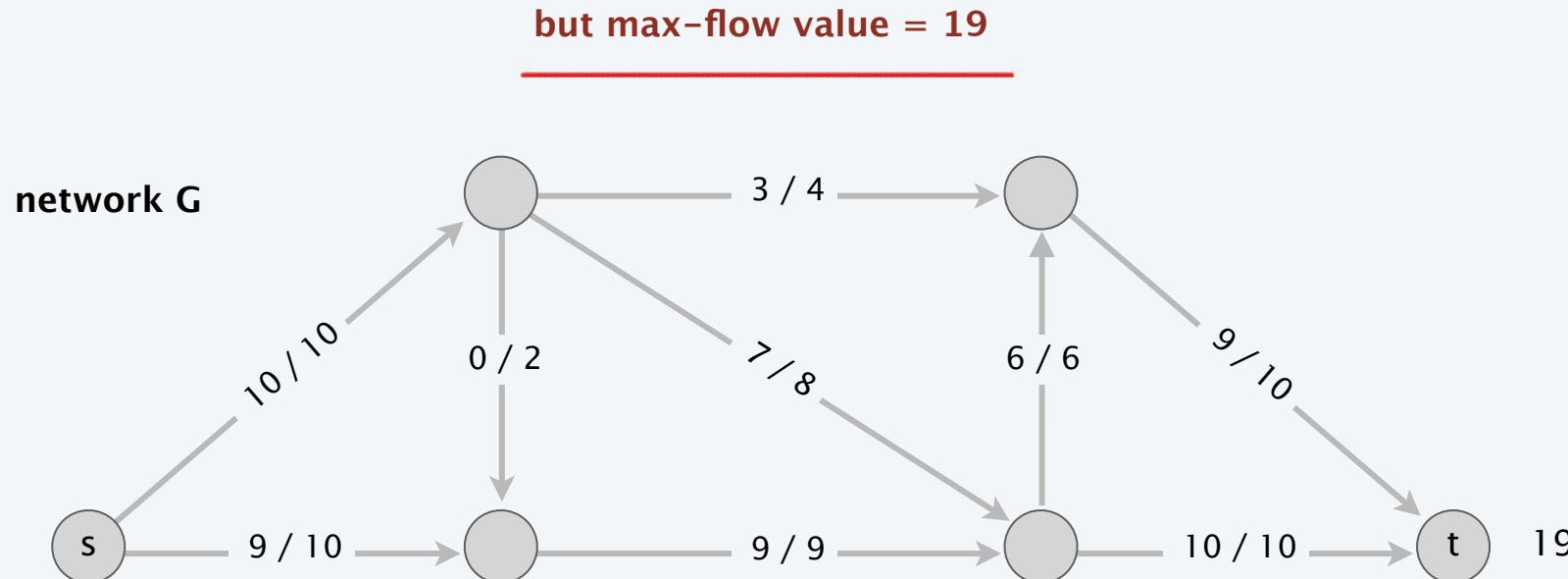
*ending flow value = 16 ↗ saturated but not optimal ↑*



# Towards a max-flow algorithm

## Greedy algorithm.

- Start with  $f(e) = 0$  for all edge  $e \in E$ .
- Find an  $s \rightarrow t$  path  $P$  where each edge has  $f(e) < c(e)$ .
- Augment flow along path  $P$ .
- Repeat until you get stuck.

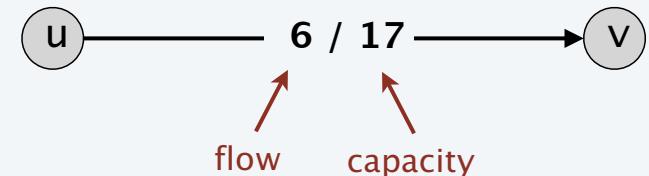


## Residual graph

**Original edge:**  $e = (u, v) \in E$ .

- Flow  $f(e)$ .
- Capacity  $c(e)$ .

original graph  $G$

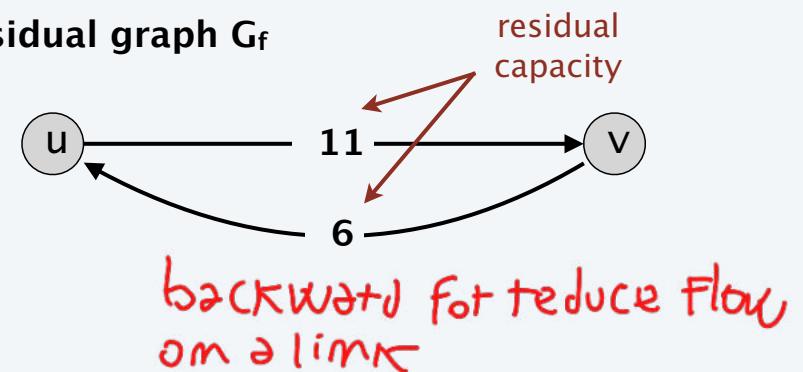


**Residual edge.**

- "Undo" flow sent.
- $e = (u, v)$  and  $e^R = (v, u)$ .
- Residual capacity:

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$$

residual graph  $G_f$



**Residual graph:**  $G_f = (V, E_f)$ .

- Residual edges with positive residual capacity.

where flow on a reverse edge negates flow on a forward edge

- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$ .

- Key property:  $f'$  is a flow in  $G_f$  iff  $f + f'$  is a flow in  $G$ .

Find an augmenting path on original network  $f$ , to  
Find an augmenting path in residual graph.

set of edges is  $e + e^T$

## Augmenting path

Def. An augmenting path is a simple  $s \rightarrow t$  path  $P$  in the residual graph  $G_f$ .

Def. The bottleneck capacity of an augmenting  $P$  is the minimum residual capacity of any edge in  $P$ .

Key property. Let  $f$  be a flow and let  $P$  be an augmenting path in  $G_f$ .

Then  $f'$  is a flow and  $\text{val}(f') = \text{val}(f) + \text{bottleneck}(G_f, P)$ .

AUGMENT( $f, c, P$ )

$b \leftarrow$  bottleneck capacity of path  $P$ . *→ minimum capacity of a link in the path*

FOREACH edge  $e \in P$  *→ of residual graph*

*edge is an a  
forward edge*      ← IF ( $e \in E$ )  $f(e) \leftarrow f(e) + b$ . *increase flow by bottleneck capacity*  
*is a backward edge*      ELSE  $f(e^R) \leftarrow f(e^R) - b$ . *reduce the flow*

RETURN  $f$ .

## Ford-Fulkerson algorithm

### Ford-Fulkerson augmenting path algorithm.

- Start with  $f(e) = 0$  for all edge  $e \in E$ .
- Find an augmenting path  $P$  in the residual graph  $G_f$ .
- Augment flow along path  $P$ .
- Repeat until you get stuck.  $\rightsquigarrow$  the source is disconnected to destination, vertex set  $P$  is split in two, a set of vertex reached by  $S$  and a set of vertex reached by  $t$ . this is a cut in graph.

FORD-FULKERSON ( $G, s, t, c$ )

FOREACH edge  $e \in E : f(e) \leftarrow 0$ .

$G_f \leftarrow$  residual graph.

WHILE (there exists an augmenting path  $P$  in  $G_f$ )

$f \leftarrow$  AUGMENT ( $f, c, P$ ).

Update  $G_f$ .

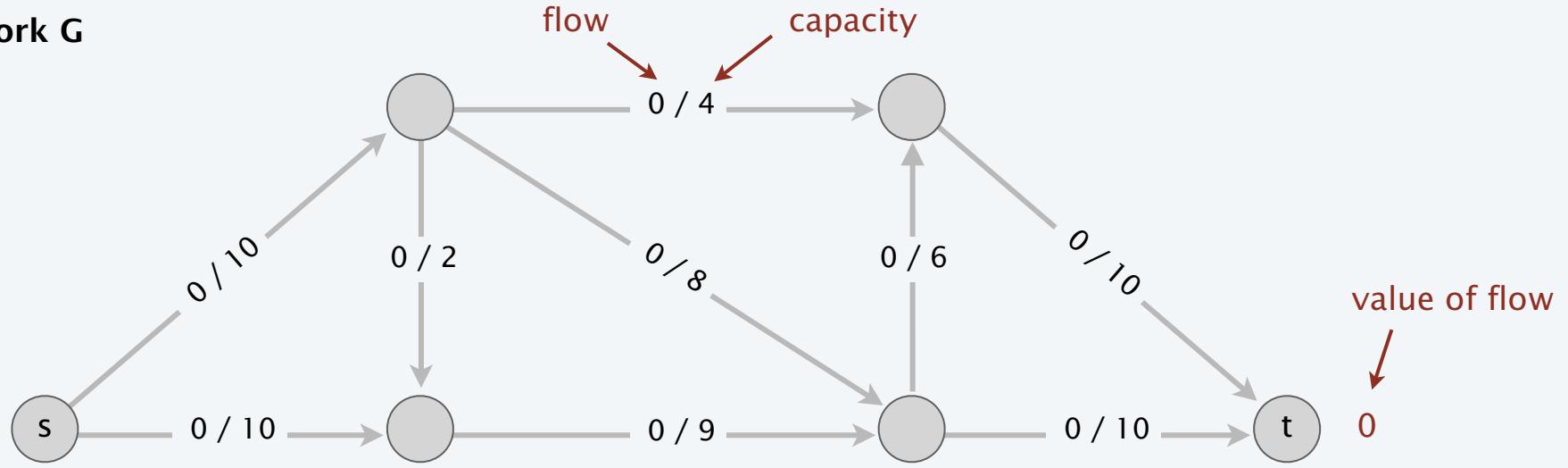
RETURN  $f$ .

}

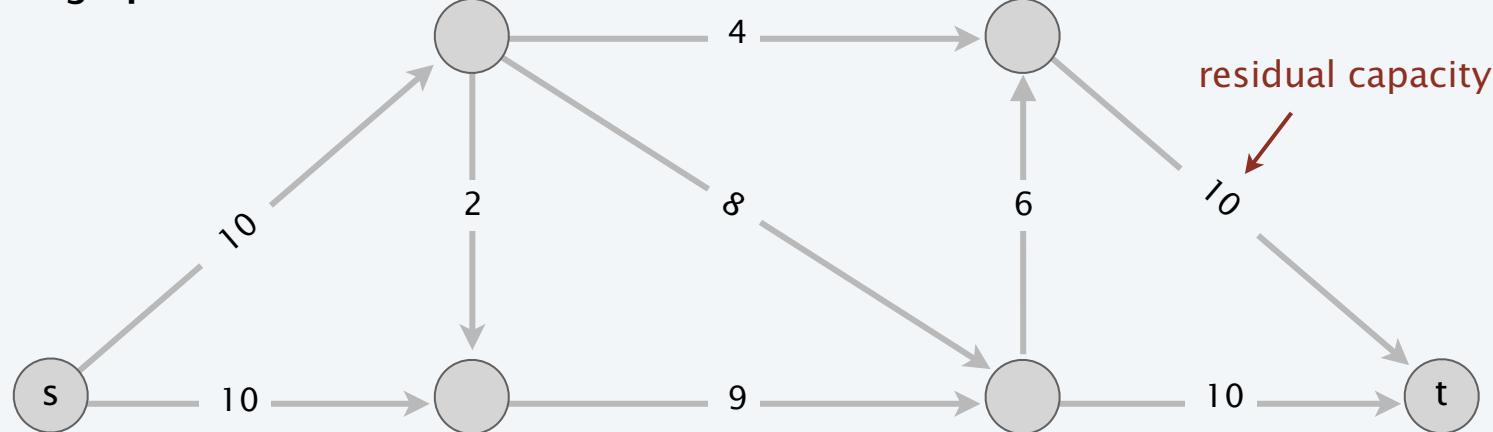
↳ we find a max-flow in the graph and a cut of minimum value

# Ford-Fulkerson algorithm demo

network G

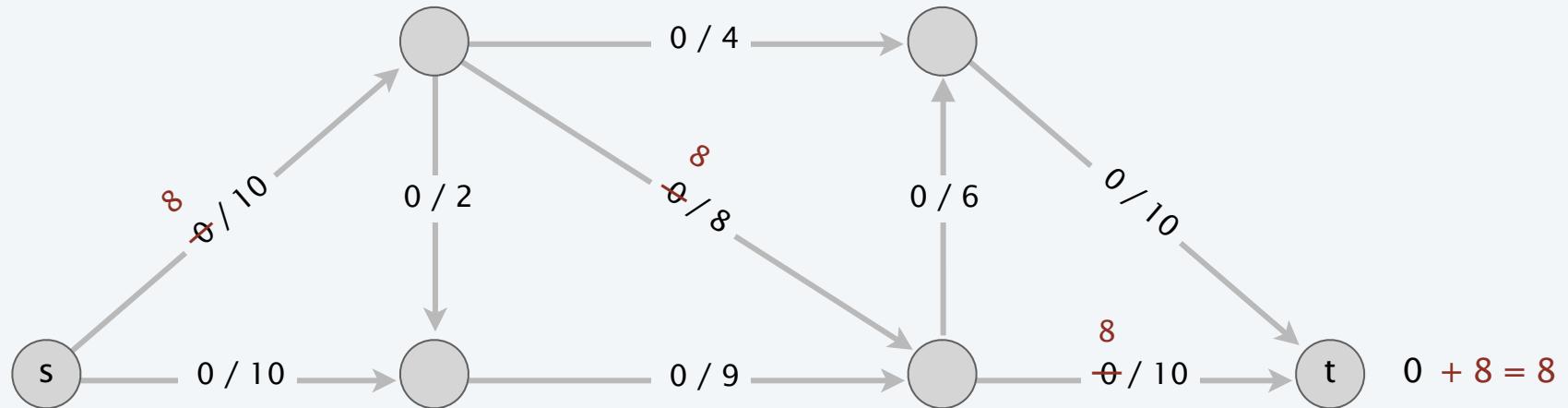


residual graph  $G_f$

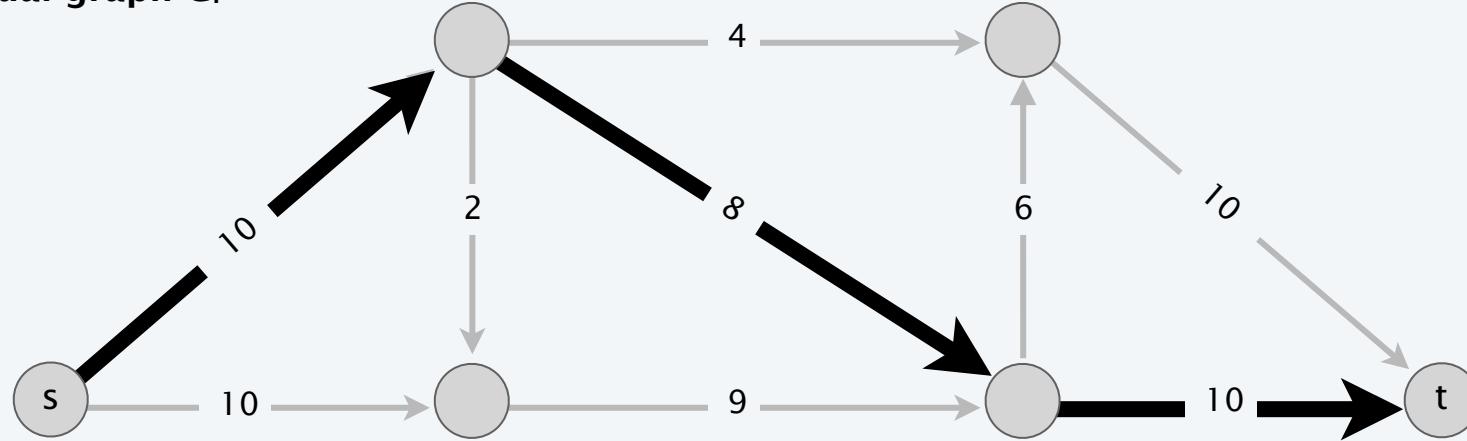


# Ford-Fulkerson algorithm demo

network G

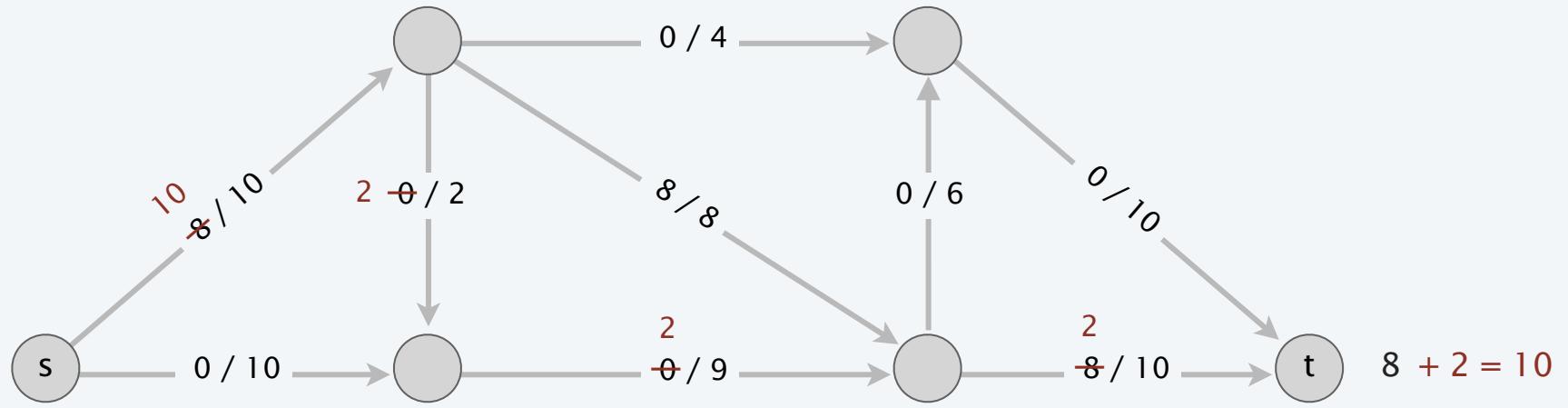


residual graph  $G_f$

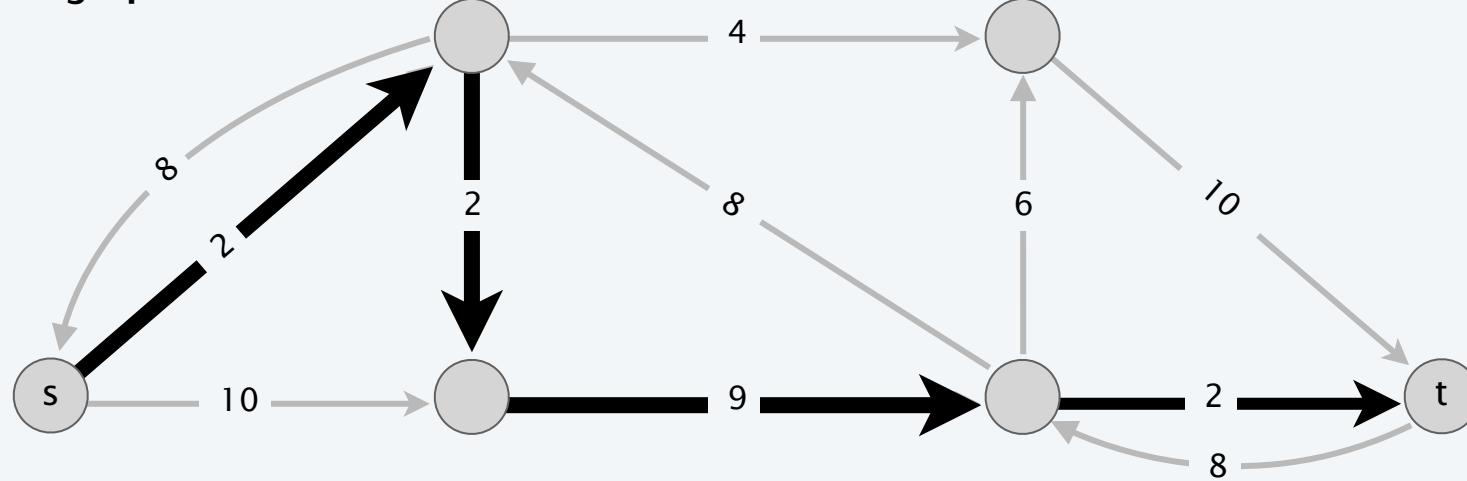


# Ford-Fulkerson algorithm demo

network G

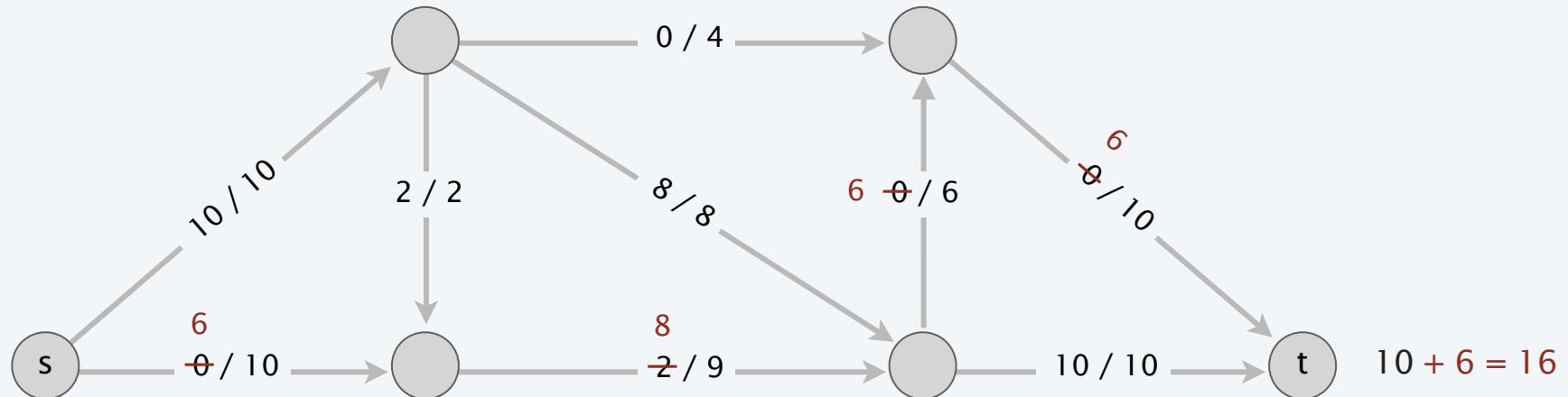


residual graph  $G_f$

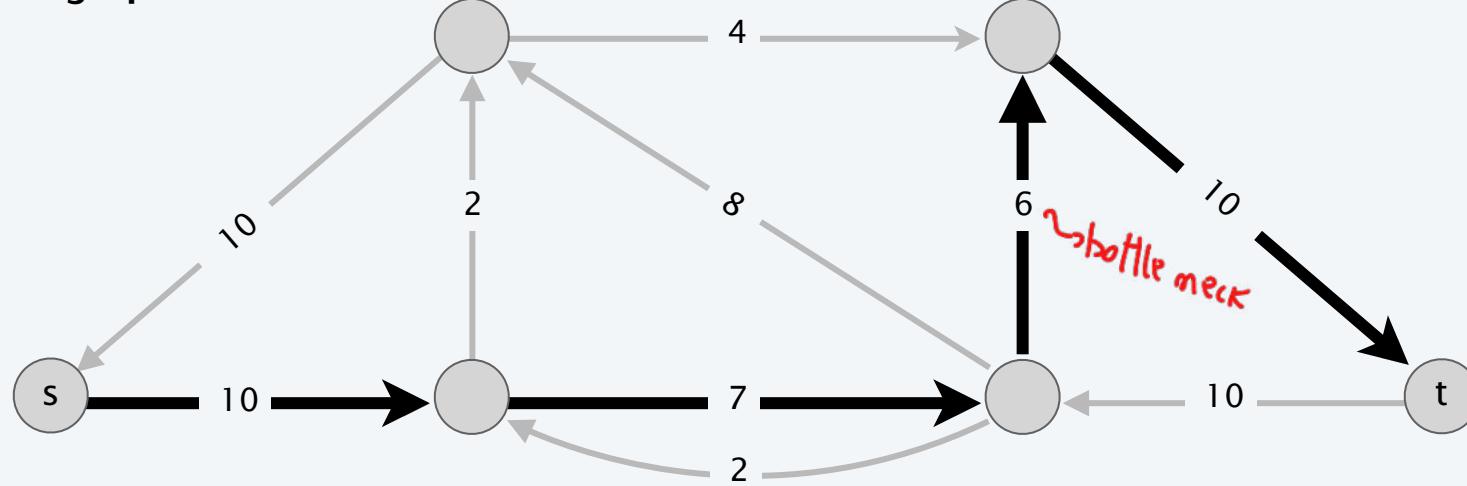


# Ford-Fulkerson algorithm demo

network G

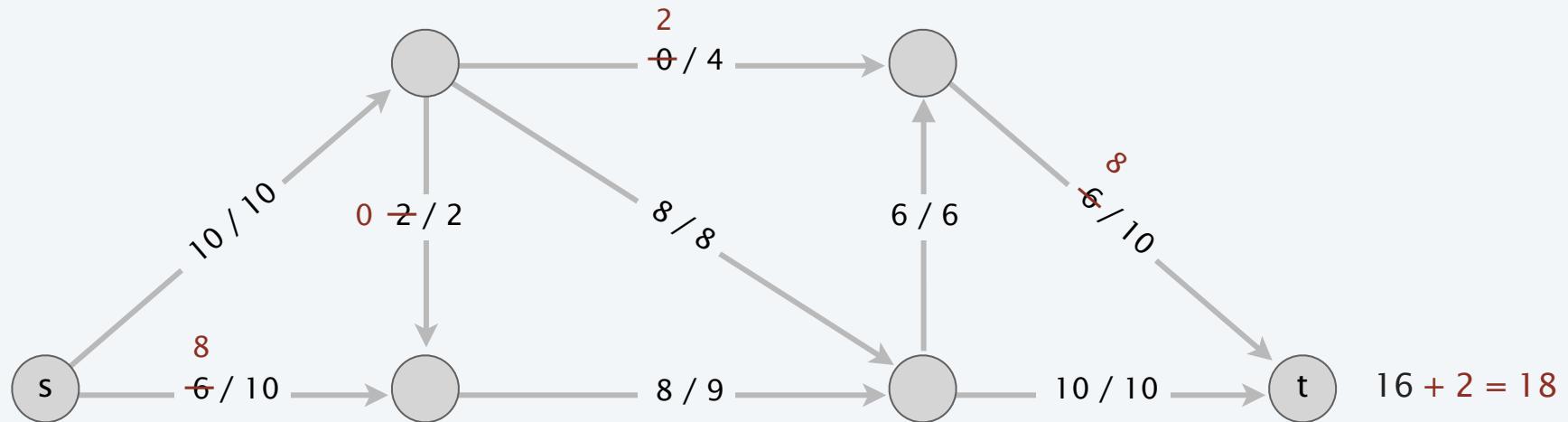


residual graph  $G_f$

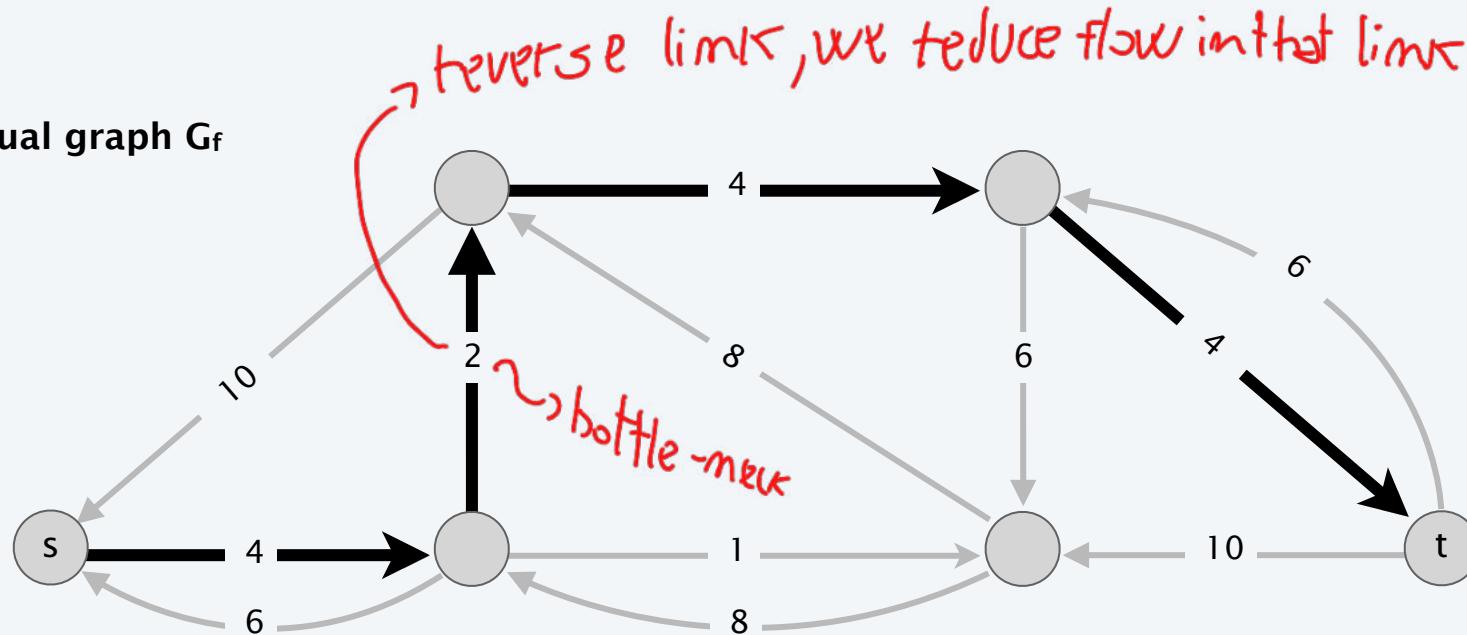


# Ford-Fulkerson algorithm demo

network G



residual graph  $G_f$

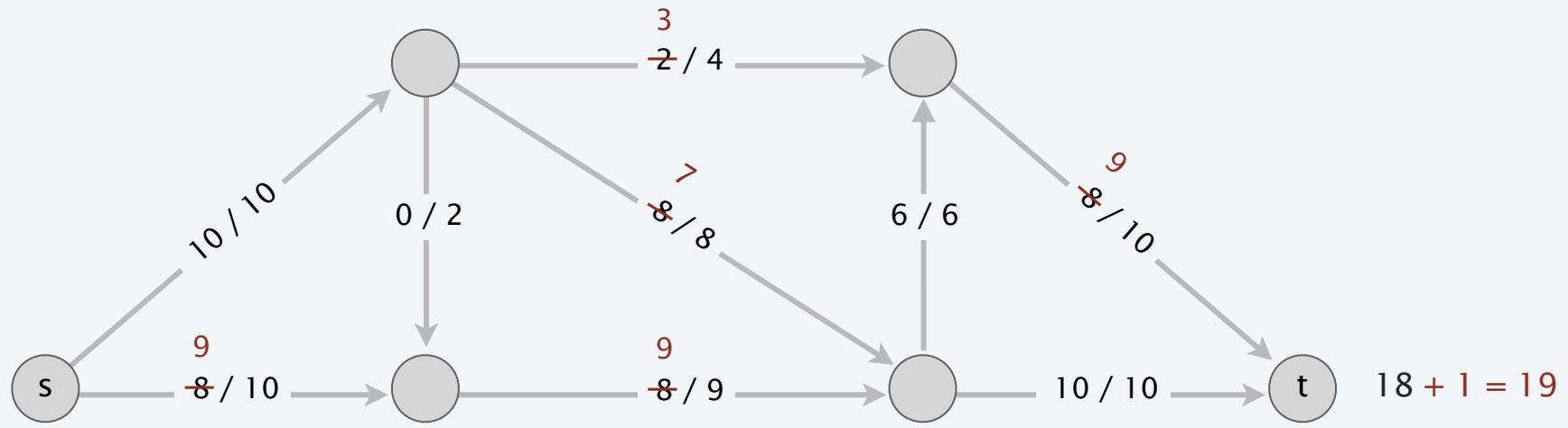


## Ford-Fulkerson algorithm demo

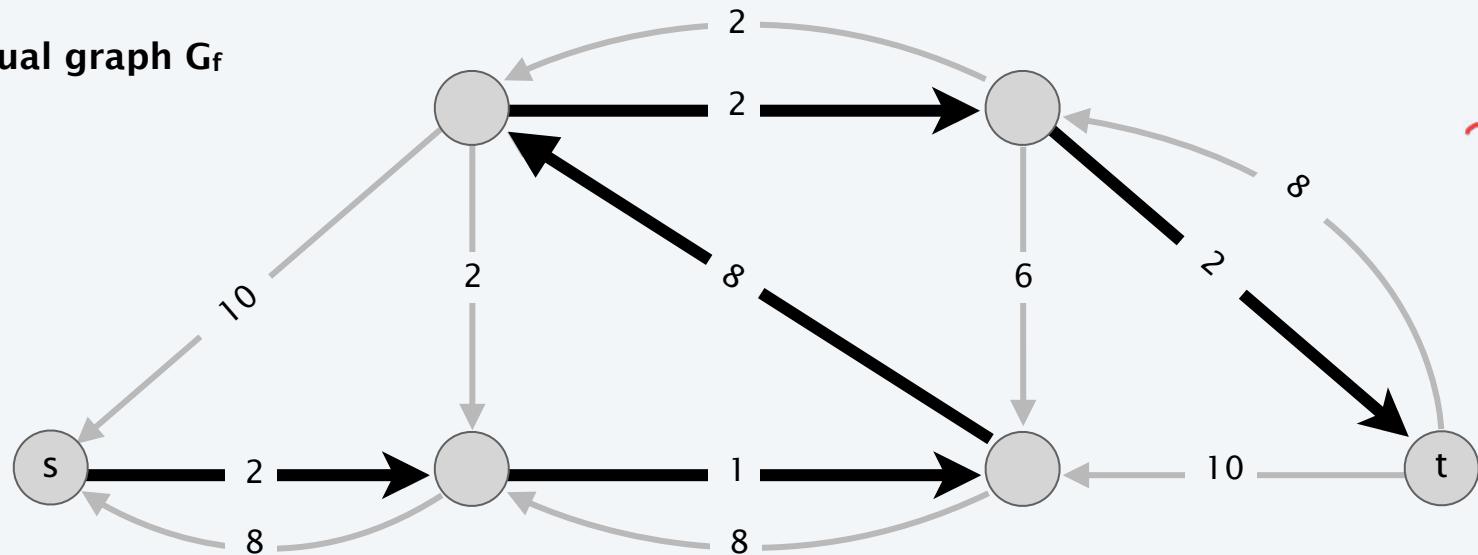
$$U = \{ \dots \dots \}$$

$$S_i = \{\dots, \dots\}$$

network G



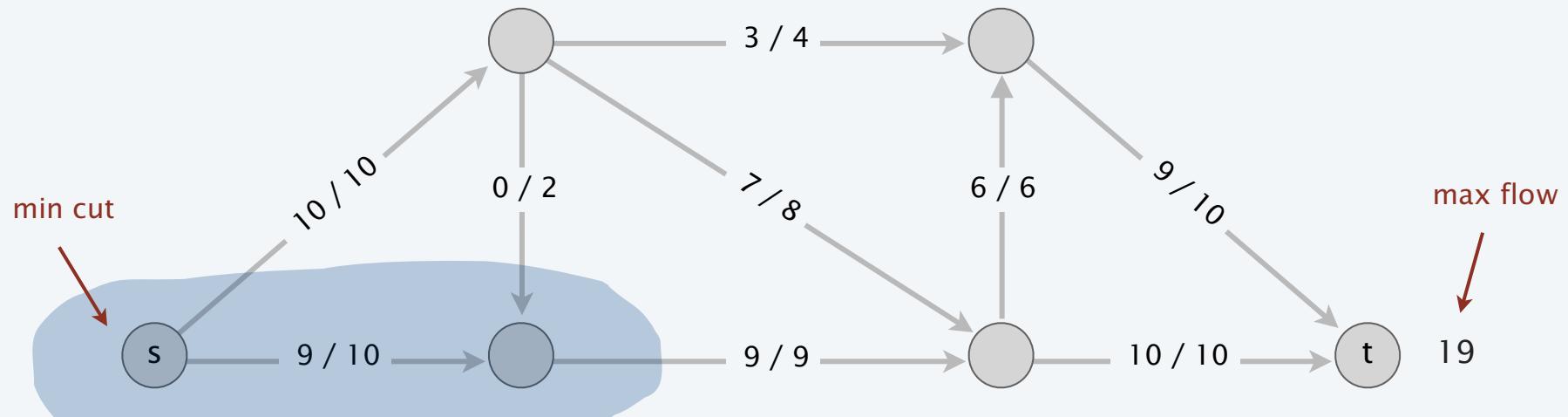
residual graph  $G_f$



# Ford-Fulkerson algorithm demo

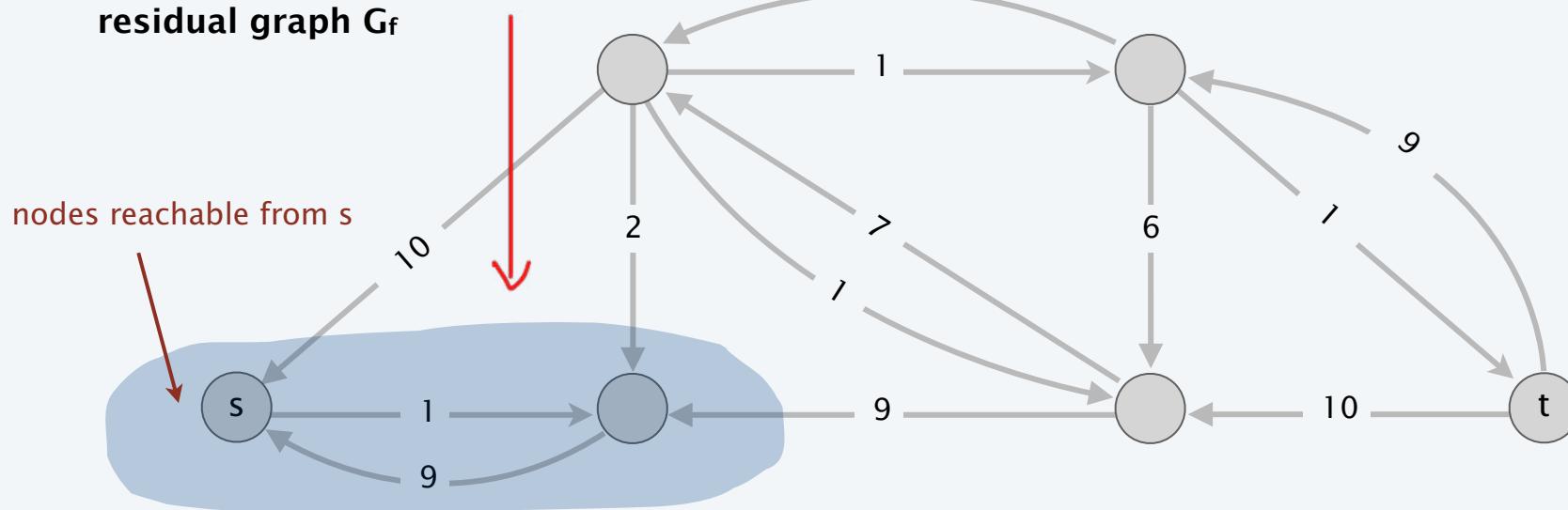
we are stucked, find the optimal max flow

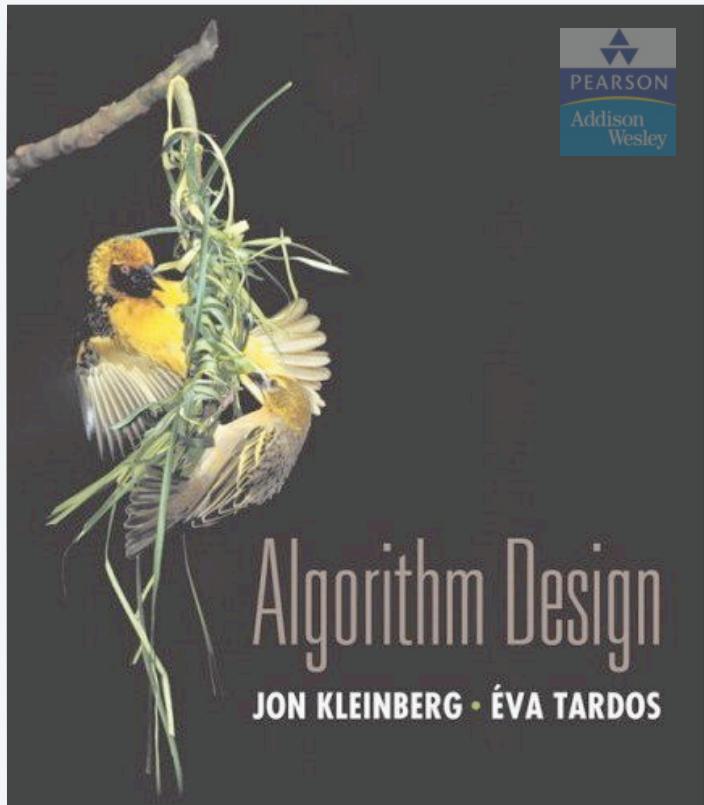
network G



↳ only edges reachable  
by s

residual graph  $G_f$





## SECTION 7.2

# 7. NETWORK FLOW I

---

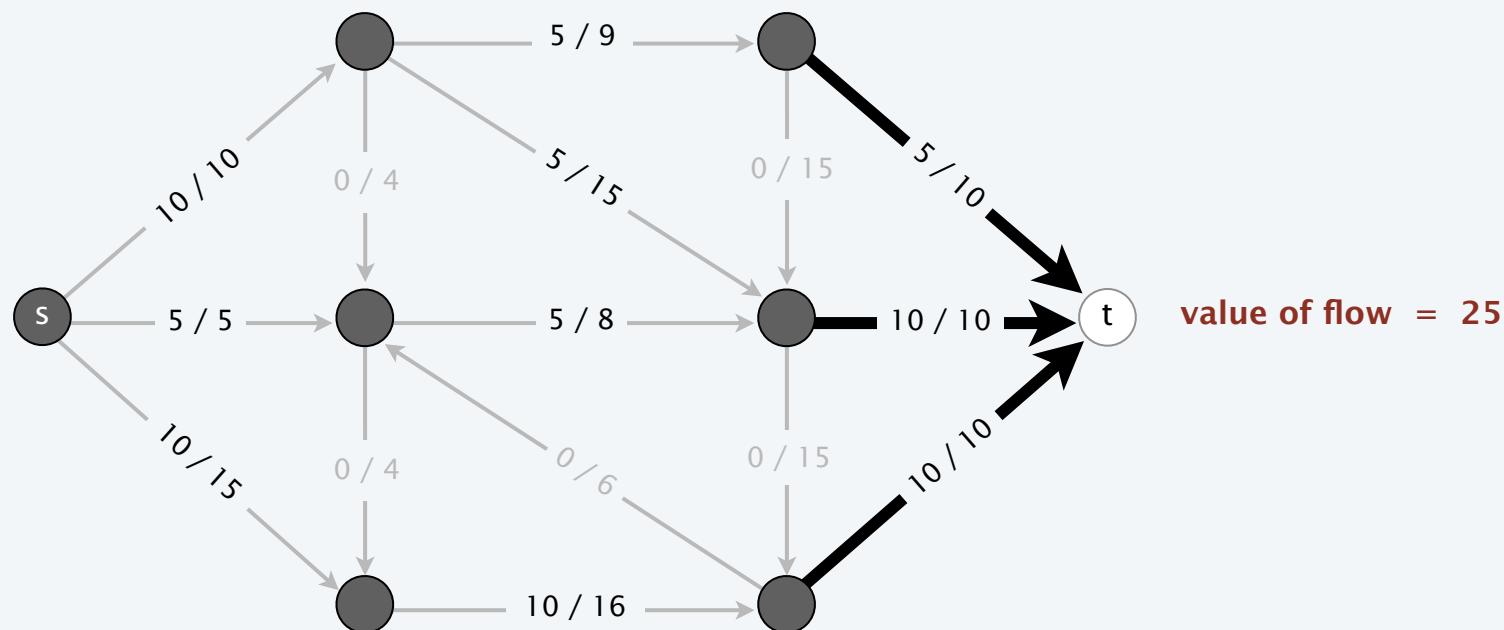
- ▶ *max-flow and min-cut problems*
- ▶ *Ford-Fulkerson algorithm*
- ▶ *max-flow min-cut theorem* ↗ *Proving that  
is optimal*
- ▶ *capacity-scaling algorithm*
- ▶ *shortest augmenting paths*
- ▶ *blocking-flow algorithm*
- ▶ *unit-capacity simple networks*

## Relationship between flows and cuts

**Flow value lemma.** Let  $f$  be any flow and let  $(A, B)$  be any cut. Then, the net flow across  $(A, B)$  equals the value of  $f$ .

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)$$

net flow across cut =  $5 + 10 + 10 = 25$



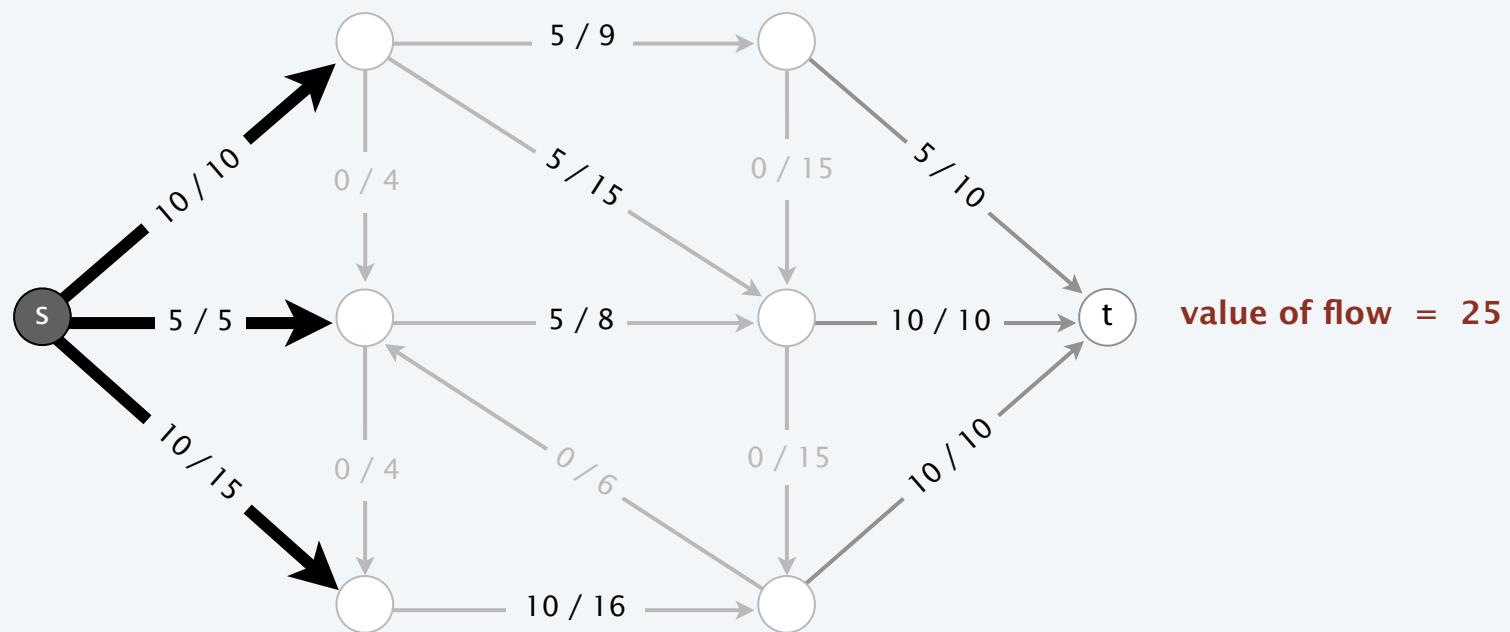
## Relationship between flows and cuts

**Flow value lemma.** Let  $f$  be any flow and let  $(A, B)$  be any cut. Then, the net flow across  $(A, B)$  equals the value of  $f$ .

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)$$

flow out source is equal flow in destination

net flow across cut =  $10 + 5 + 10 = 25$



## Relationship between flows and cuts

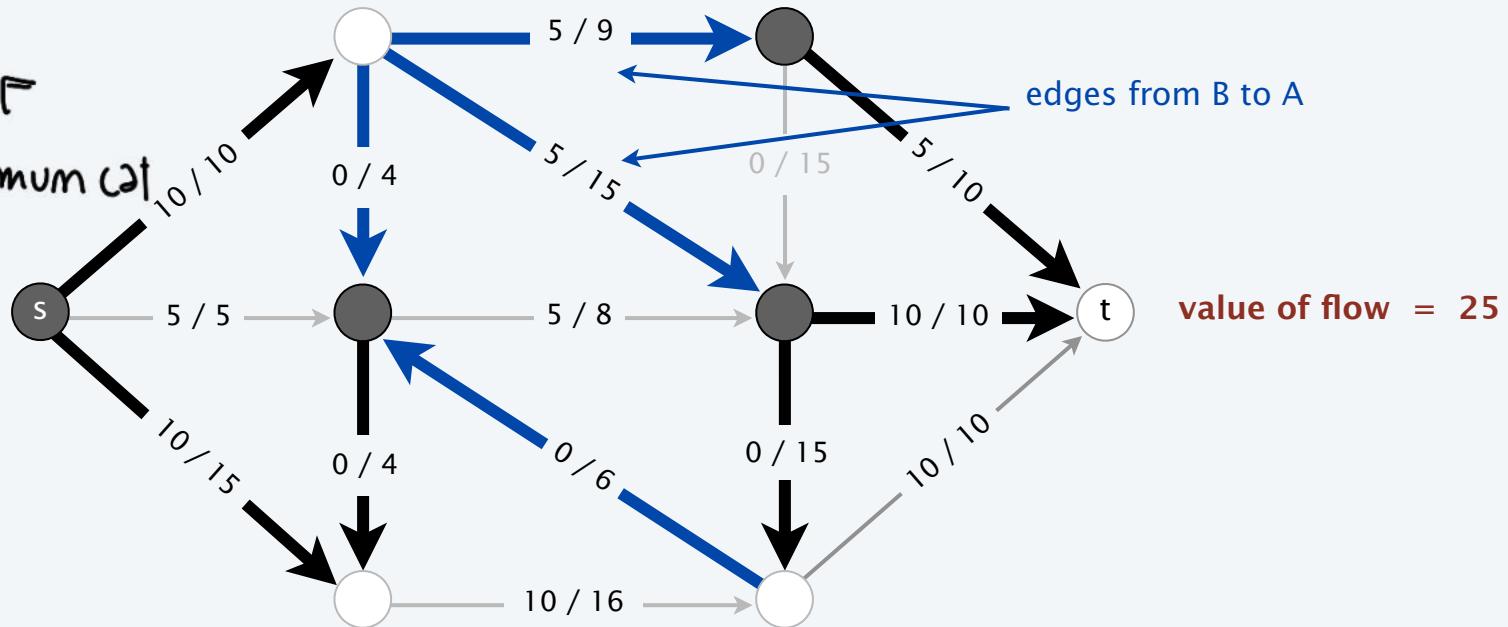
**Flow value lemma.** Let  $f$  be any flow and let  $(A, B)$  be any cut. Then, the net flow across  $(A, B)$  equals the value of  $f$ .

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)$$

$f(e), e \text{ out of } A$        $f(e), e \text{ in to } A$

$$\text{net flow across cut} = (10 + 10 + 5 + 10 + 0 + 0) - (5 + 5 + 0 + 0) = 25$$

example of  
2 cut, minimum cut



## Relationship between flows and cuts

**Flow value lemma.** Let  $f$  be any flow and let  $(A, B)$  be any cut. Then, the net flow across  $(A, B)$  equals the value of  $f$ .

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)$$

value of the flow is equal to the  
net flow across the cut

Pf.

$$v(f) = \sum_{e \text{ out of } s} f(e) \quad \rightarrow \text{ zero if } v \neq s$$

by flow conservation, all terms  
except  $v = s$  are 0

$$\rightarrow = \sum_{v \in A} \left( \sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) \right)$$

$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e). \blacksquare$$

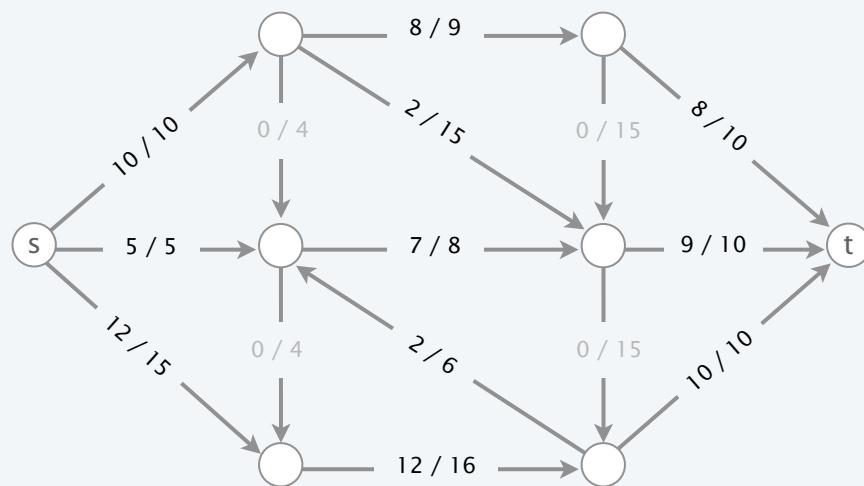
# Relationship between flows and cuts

**Weak duality.** Let  $f$  be any flow and  $(A, B)$  be any cut. Then,  $v(f) \leq \text{cap}(A, B)$ .

Pf.

$$\begin{aligned}
 v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
 &\leq \sum_{e \text{ out of } A} f(e) \\
 &\leq \sum_{e \text{ out of } A} c(e) \\
 &= \text{cap}(A, B) \quad \blacksquare
 \end{aligned}$$

(Capacity of a cut  
is  $\sum_{e \text{ out of } A} c(e)$ )



value of flow = 27

<

capacity of cut = 30

## Max-flow min-cut theorem

Augmenting path theorem. A flow  $f$  is a max-flow iff no augmenting paths.

Max-flow min-cut theorem. Value of the max-flow = capacity of min-cut.

Pf. The following three conditions are equivalent for any flow  $f$ :

- i. There exists a cut  $(A, B)$  such that  $\text{cap}(A, B) = \text{val}(f)$ .
- ii.  $f$  is a max-flow.
- iii. There is no augmenting path with respect to  $f$ .

we must demonstrate  
that  $i \Rightarrow ii \Rightarrow iii \Rightarrow i$

[  $i \Rightarrow ii$  ]

- Suppose that  $(A, B)$  is a cut such that  $\text{cap}(A, B) = \text{val}(f)$ .
- Then, for any flow  $f'$ ,  $\text{val}(f') \leq \text{cap}(A, B) = \text{val}(f)$ .
- Thus,  $f$  is a max-flow.

■      ↑      ↑  
weak duality      by assumption

→ flow must be biggest  
of  $\text{cap}(A, B)$  for weak  
duality

## Max-flow min-cut theorem

---

Augmenting path theorem. A flow  $f$  is a max-flow iff no augmenting paths.

Max-flow min-cut theorem. Value of the max-flow = capacity of min-cut.

Pf. The following three conditions are equivalent for any flow  $f$ :

- i. There exists a cut  $(A, B)$  such that  $\text{cap}(A, B) = \text{val}(f)$ .
- ii.  $f$  is a max-flow.
- iii. There is no augmenting path with respect to  $f$ .

[ ii  $\Rightarrow$  iii ] We prove contrapositive:  $\sim$ iii  $\Rightarrow$   $\sim$ ii.

- Suppose that there is an augmenting path with respect to  $f$ .  $\sim$ ii
- Can improve flow  $f$  by sending flow along this path.
- Thus,  $f$  is not a max-flow. ■

↳ very easy

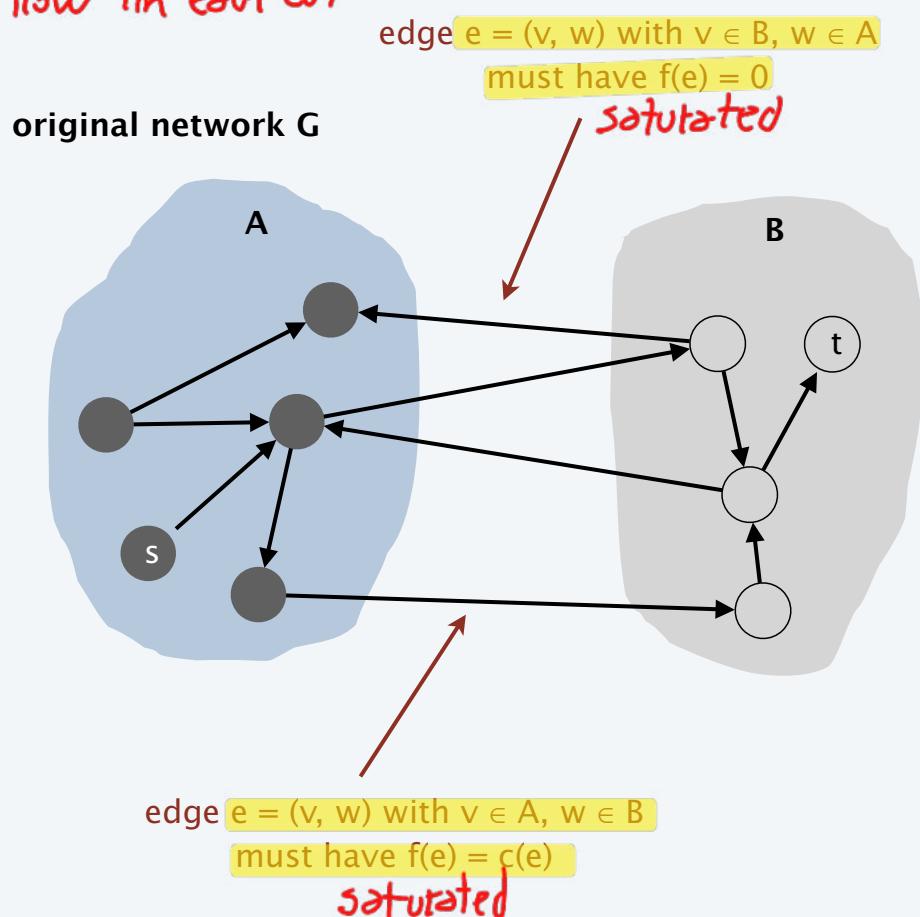
# Max-flow min-cut theorem

[ iii  $\Rightarrow$  i ]

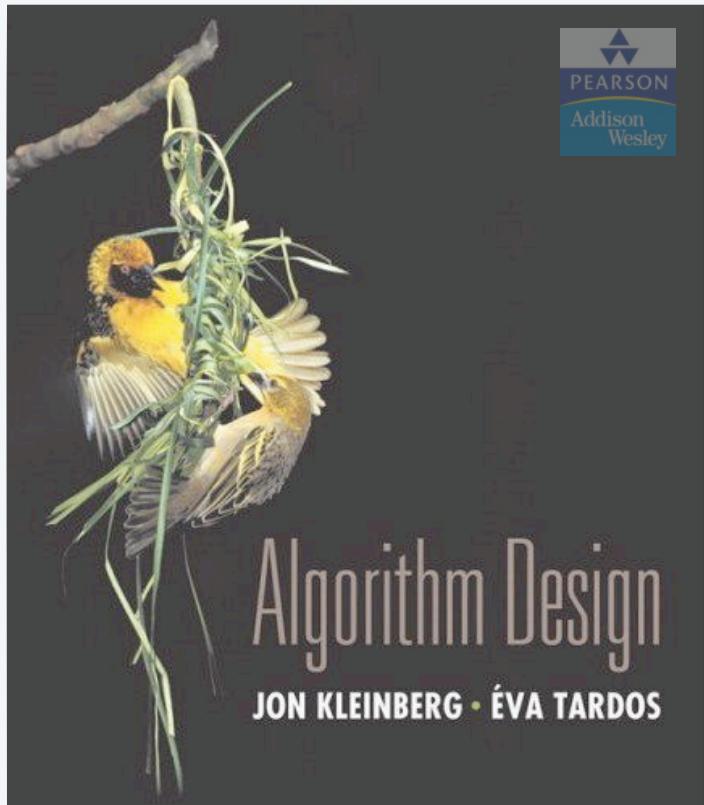
- Let  $f$  be a flow with no augmenting paths. ( $f$  is a max flow)
- Let  $A$  be set of nodes reachable from  $s$  in residual graph  $G_f$ .
- By definition of cut  $A$ ,  $s \in A$ . (all links that  $A \rightarrow B$  are saturated)
- By definition of flow  $f$ ,  $t \notin A$ . ~net flow in each cut

$$\begin{aligned}
 v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
 &= \sum_{e \text{ out of } A} c(e) \\
 &= \text{cap}(A, B) \quad \blacksquare
 \end{aligned}$$

flow-value lemma



we demonstrate the theorem



## SECTION 7.3

# 7. NETWORK FLOW I

---

- ▶ *max-flow and min-cut problems*
- ▶ *Ford-Fulkerson algorithm* →  
number of aug. meeting  
path is very large,  
convergence is very slow
- ▶ *max-flow min-cut theorem*
- ▶ ***capacity-scaling algorithm***
- ▶ *shortest augmenting paths*
- ▶ *blocking-flow algorithm*
- ▶ *unit-capacity simple networks*

## Running time

---

Assumption. Capacities are integers between 1 and  $C$ .

**Integrality invariant.** Throughout the algorithm, the flow values  $f(e)$  and the residual capacities  $c_f(e)$  are integers. *always, Flow each time change of an integer's value*

Theorem. The algorithm terminates in at most  $\text{val}(f^*) \leq nC$  iterations.

Pf. Each augmentation increases the value by at least 1. • ↗ m vertices,  $C$  max capacity of an edge  
↗ worst case all time augment only by 1

Corollary. The running time of Ford-Fulkerson is  $O(mnC)$ .

Corollary. If  $C = 1$ , the running time of Ford-Fulkerson is  $O(mn)$ .

**Integrality theorem.** Then exists a max-flow  $f^*$  for which every flow value  $f^*(e)$  is an integer.

Pf. Since algorithm terminates, theorem follows from invariant. •

## Bad case for Ford-Fulkerson

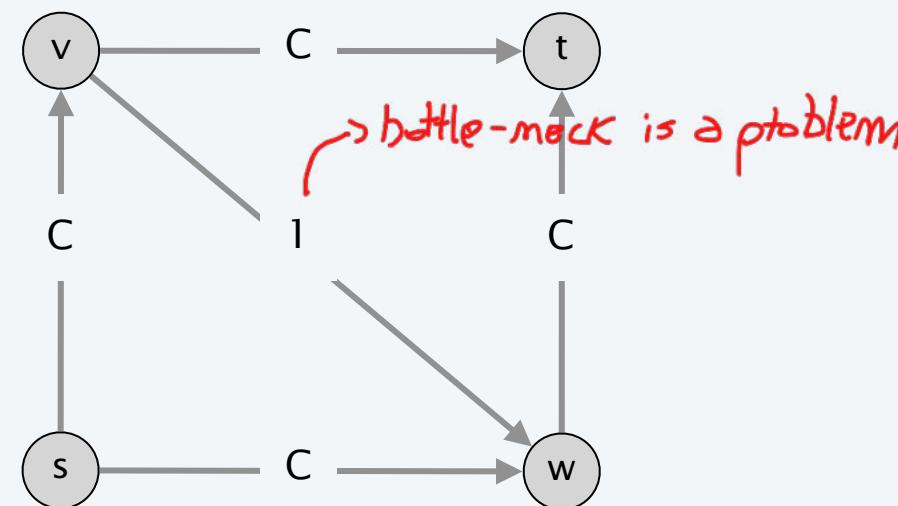
Q. Is generic Ford-Fulkerson algorithm poly-time in input size?

m, n, and  $\log C$

A. No. If max capacity is  $C$ , then algorithm can take  $\geq C$  iterations.

- $s \rightarrow v \rightarrow w \rightarrow t$
- $s \rightarrow w \rightarrow v \rightarrow t$
- $s \rightarrow v \rightarrow w \rightarrow t$
- $s \rightarrow w \rightarrow v \rightarrow t$
- ...
- $s \rightarrow v \rightarrow w \rightarrow t$
- $s \rightarrow w \rightarrow v \rightarrow t$

each augmenting path  
sends only 1 unit of flow  
(# augmenting paths =  $2C$ )



## Choosing good augmenting paths

---

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms. *on size of the input size that is*  
*m+m+log c*
- Clever choices lead to polynomial algorithms.
- If capacities are irrational, algorithm not guaranteed to terminate!

*need to be integers*

**Goal.** Choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

# Choosing good augmenting paths

Choose augmenting paths with:

- Max bottleneck capacity.
- Sufficiently large bottleneck capacity.
- Fewest number of edges.

→ all work

shortest augmenting path

## Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems

JACK EDMONDS

*University of Waterloo, Waterloo, Ontario, Canada*

AND

RICHARD M. KARP

*University of California, Berkeley, California*

**ABSTRACT.** This paper presents new algorithms for the maximum flow problem, the Hitchcock transportation problem, and the general minimum-cost flow problem. Upper bounds on the numbers of steps in these algorithms are derived, and are shown to compare favorably with upper bounds on the numbers of steps required by earlier algorithms.

Edmonds-Karp 1972 (USA)

Dokl. Akad. Nauk SSSR  
Tom 194 (1970), No. 4

Soviet Math. Dokl.  
Vol. 11 (1970), No. 5

## ALGORITHM FOR SOLUTION OF A PROBLEM OF MAXIMUM FLOW IN A NETWORK WITH POWER ESTIMATION

UDC 518.5

E. A. DINIC

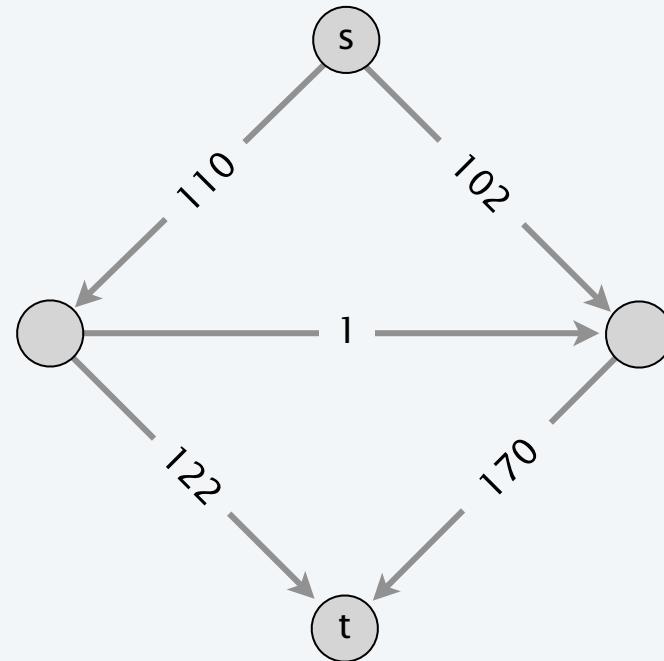
Different variants of the formulation of the problem of maximal stationary flow in a network and its many applications are given in [1]. There also is given an algorithm solving the problem in the case where the initial data are integers (or, what is equivalent, commensurable). In the general case this algorithm requires preliminary rounding off of the initial data, i.e. only an approximate solution of the problem is possible. In this connection the rapidity of convergence of the algorithm is inversely proportional to the relative precision.

Dinic 1970 (Soviet Union)

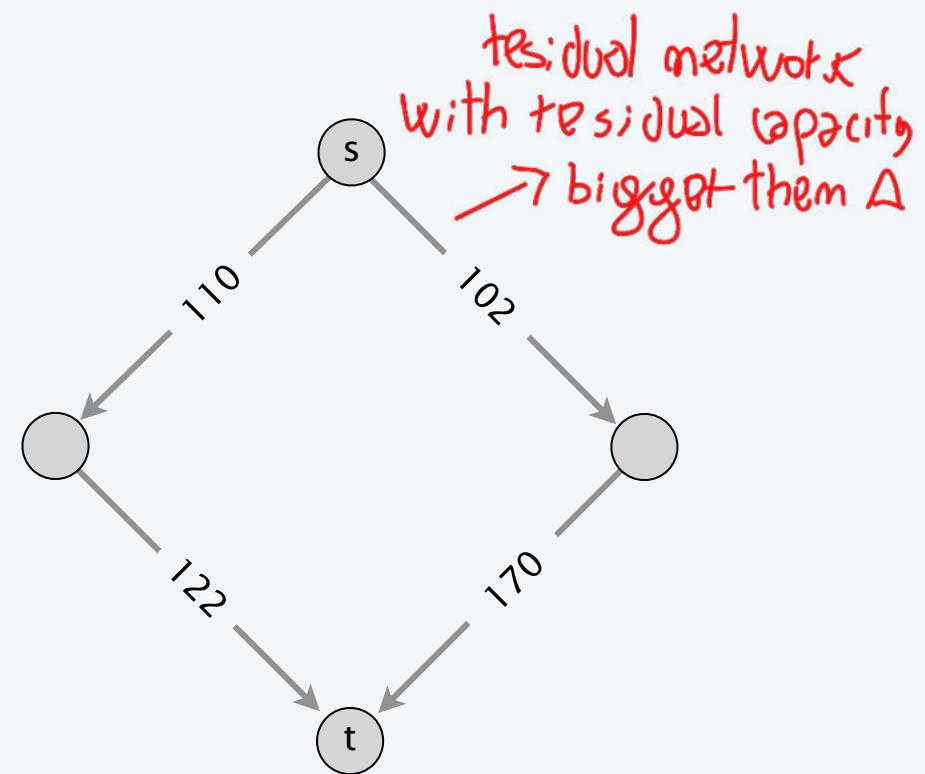
## Capacity-scaling algorithm

Intuition. Choose augmenting path with highest bottleneck capacity:  
it increases flow by max possible amount in given iteration.

- Don't worry about finding exact highest bottleneck path.
- Maintain scaling parameter  $\Delta$ .
- Let  $G_f(\Delta)$  be the subgraph of the residual graph consisting only of arcs with capacity  $\geq \Delta$ .



$G_f$



$G_f(\Delta), \Delta = 100$  initial scale parameter

# Capacity-scaling algorithm

---

CAPACITY-SCALING( $G, s, t, c$ )

FOREACH edge  $e \in E : f(e) \leftarrow 0.$

$\Delta \leftarrow$  largest power of  $2 \leq C.$  *→ initial  $\Delta$ , in use before  
 $\Delta = 128 \leq 170$*

WHILE ( $\Delta \geq 1$ )

$G_f(\Delta) \leftarrow \Delta$ -residual graph. *~only edges with residual capacity  $\geq \Delta$*

WHILE (there exists an augmenting path  $P$  in  $G_f(\Delta)$ )

$f \leftarrow$  AUGMENT ( $f, c, P).$

Update  $G_f(\Delta).$

$\Delta \leftarrow \Delta / 2.$

RETURN  $f.$

## Capacity-scaling algorithm: proof of correctness

---

Assumption. All edge capacities are integers between 1 and  $C$ .

Integrality invariant. All flow and residual capacity values are integral.

Theorem. If capacity-scaling algorithm terminates, then  $f$  is a max-flow.

Pf.

- By integrality invariant, when  $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$ .
- Upon termination of  $\Delta = 1$  phase, there are no augmenting paths. ▀

## Capacity-scaling algorithm: analysis of running time

because divide by 2

Lemma 1. The outer while loop repeats  $1 + \lceil \log_2 C \rceil$  times.

Pf. Initially  $C/2 < \Delta \leq C$ ;  $\Delta$  decreases by a factor of 2 in each iteration. ▀

Lemma 2. Let  $f$  be the flow at the end of a  $\Delta$ -scaling phase. Then,  
the value of the max-flow  $\leq \text{val}(f) + m \Delta$ . ← proof on next slide



Lemma 3. There are at most  $2m$  augmentations per scaling phase.

Pf. each augmentation  $\Delta/2$

- Let  $f$  be the flow at the end of the previous scaling phase.
- LEMMA 2  $\Rightarrow \text{val}(f^*) \leq \text{val}(f) + 2m\Delta$ .
- Each augmentation in a  $\Delta$ -phase increases  $\text{val}(f)$  by at least  $\Delta$ . ▀

Theorem. The scaling max-flow algorithm finds a max flow in  $O(m \log C)$  augmentations. It can be implemented to run in  $O(m^2 \log C)$  time.

Pf. Follows from LEMMA 1 and LEMMA 3. ▀

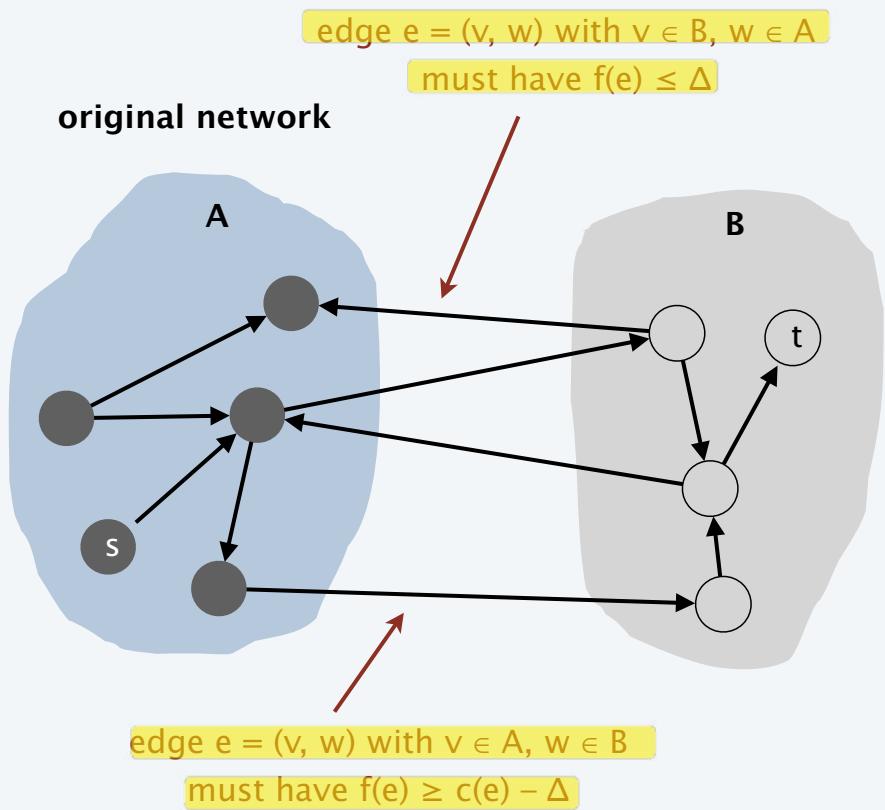
# Capacity-scaling algorithm: analysis of running time

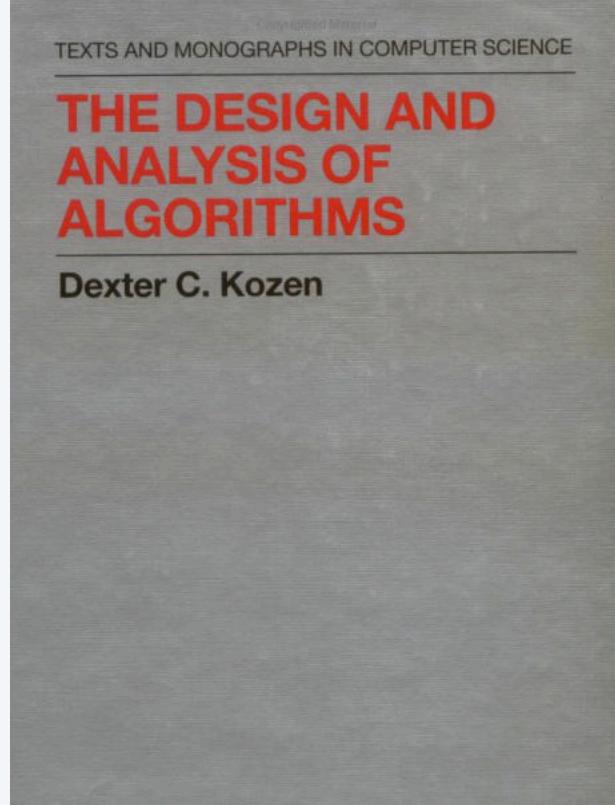
**Lemma 2.** Let  $f$  be the flow at the end of a  $\Delta$ -scaling phase. Then, the value of the max-flow  $\leq \text{val}(f) + m \Delta$ .

Pf.

- We show there exists a cut  $(A, B)$  such that  $\text{cap}(A, B) \leq \text{val}(f) + m \Delta$ .
- Choose  $A$  to be the set of nodes reachable from  $s$  in  $G_f(\Delta)$ .
- By definition of cut  $A$ ,  $s \in A$ .
- By definition of flow  $f$ ,  $t \notin A$ .

$$\begin{aligned}
 \text{val}(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
 &\geq \sum_{e \text{ out of } A} (c(e) - \Delta) - \sum_{e \text{ in to } A} \Delta \\
 &= \sum_{e \text{ out of } A} c(e) - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ in to } A} \Delta \\
 &\geq \text{cap}(A, B) - m\Delta \blacksquare
 \end{aligned}$$





## SECTION 17.2

# 7. NETWORK FLOW I

---

- ▶ *max-flow and min-cut problems*
- ▶ *Ford-Fulkerson algorithm*
- ▶ *max-flow min-cut theorem*
- ▶ *capacity-scaling algorithm*
- ▶ ***shortest augmenting paths***
- ▶ *blocking-flow algorithm*
- ▶ *unit-capacity simple networks*

## Shortest augmenting path

Q. Which augmenting path?

A. The one with the fewest number of edges.

can find via BFS

the shortest path in number of edges

**SHORTEST-AUGMENTING-PATH**( $G, s, t, c$ )

FOREACH  $e \in E : f(e) \leftarrow 0$ .

$G_f \leftarrow$  residual graph.

WHILE (there exists an augmenting path in  $G_f$ )

$P \leftarrow$  BREADTH-FIRST-SEARCH ( $G_f, s, t$ ).

$f \leftarrow$  AUGMENT ( $f, c, P$ ).

Update  $G_f$ .

RETURN  $f$ .

## Shortest augmenting path: overview of analysis

- L1. Throughout the algorithm, length of the shortest path never decreases.  
only go to saturate edges, that are replaced with opposite edges, not decrease length
- L2. After at most  $m$  shortest path augmentations, the length of the shortest augmenting path strictly increases. , at least by one, because are going to remove edges and add in other direction, that not are part of shortest augmenting path. After m removals, must be longer than the path

Theorem. The shortest augmenting path algorithm runs in  $O(m^2 n)$  time.

Pf.

- $O(m + n)$  time to find shortest augmenting path via BFS.
- $O(m)$  augmentations for paths of length  $k$ .
- If there is an augmenting path, there is a simple one. of length  $k$ 
  - ⇒  $1 \leq k < n$
  - ⇒  $O(m n)$  augmentations. ▀

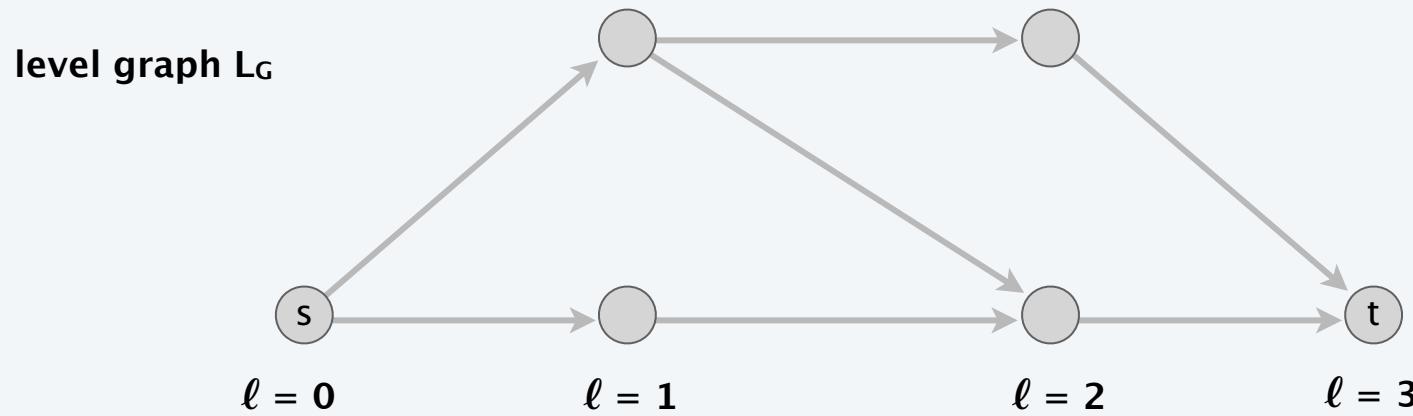
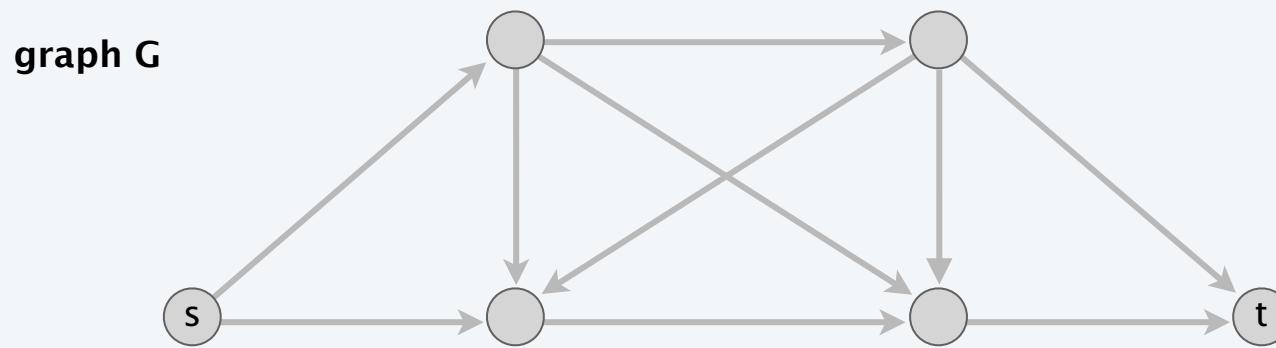
$$O(m m (m+n)) \approx O(m^2 n)$$

## Shortest augmenting path: **analysis**

---

**Def.** Given a digraph  $G = (V, E)$  with source  $s$ , its **level graph** is defined by:

- $\ell(v) = \text{number of edges in shortest path from } s \text{ to } v$ .
- $L_G = (V, E_G)$  is the subgraph of  $G$  that contains only those edges  $(v, w) \in E$  with  $\ell(w) = \ell(v) + 1$ . , interested only in edges that take part in shortest path



## Shortest augmenting path: analysis

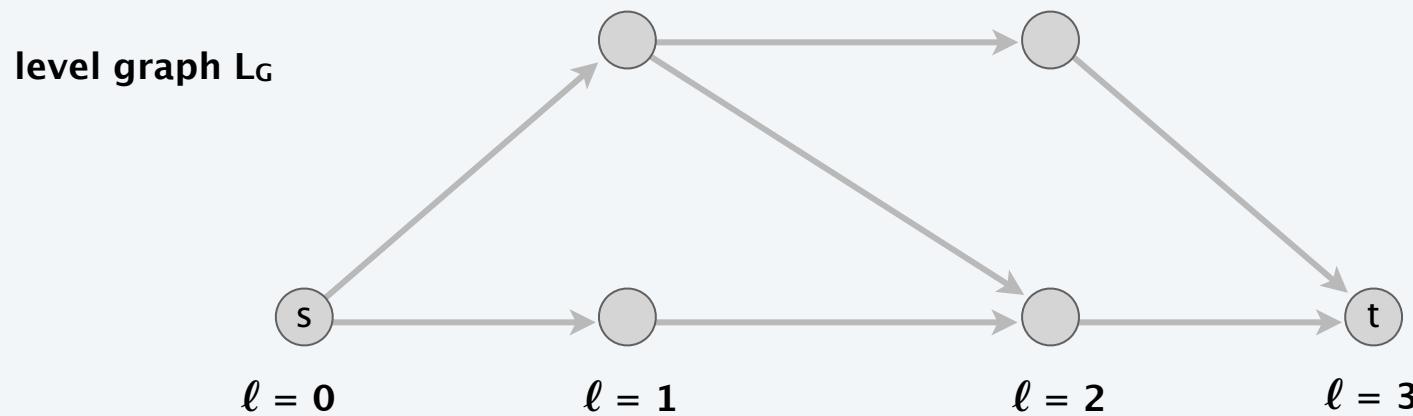
**Def.** Given a digraph  $G = (V, E)$  with source  $s$ , its **level graph** is defined by:

- $\ell(v) = \text{number of edges in shortest path from } s \text{ to } v$ .
- $L_G = (V, E_G)$  is the subgraph of  $G$  that contains only those edges  $(v, w) \in E$  with  $\ell(w) = \ell(v) + 1$ .

**Property.** Can compute level graph in  $O(m + n)$  time.

**Pf.** Run BFS; delete back and side edges.

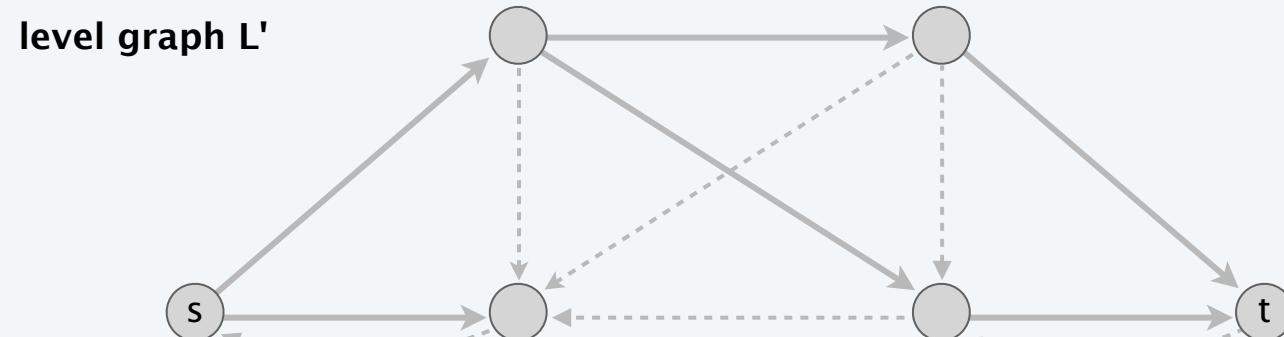
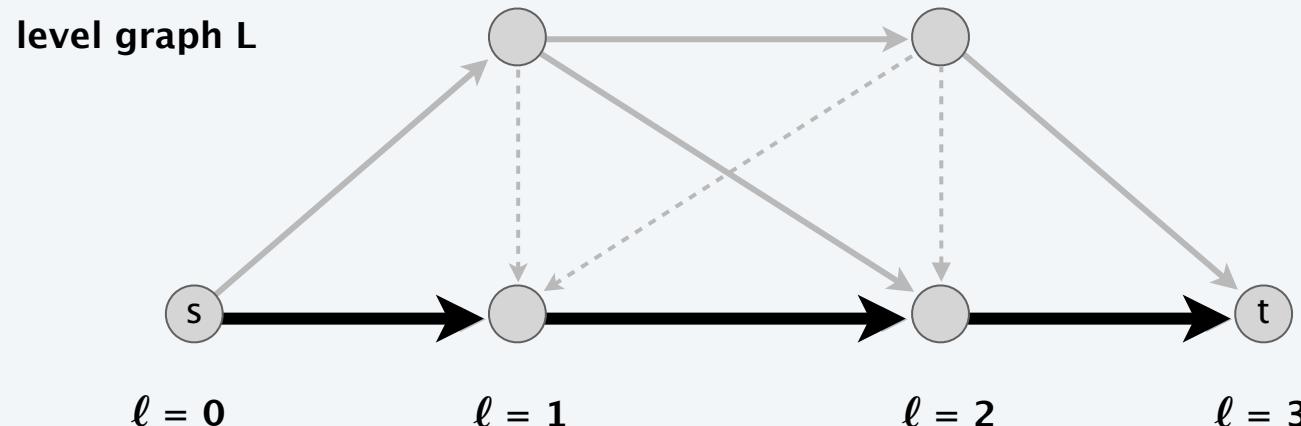
**Key property.**  $P$  is a shortest  $s \rightarrow v$  path in  $G$  iff  $P$  is an  $s \rightarrow v$  path  $L_G$ .



# Shortest augmenting path: analysis

L1. Throughout the algorithm, length of the shortest path never decreases.

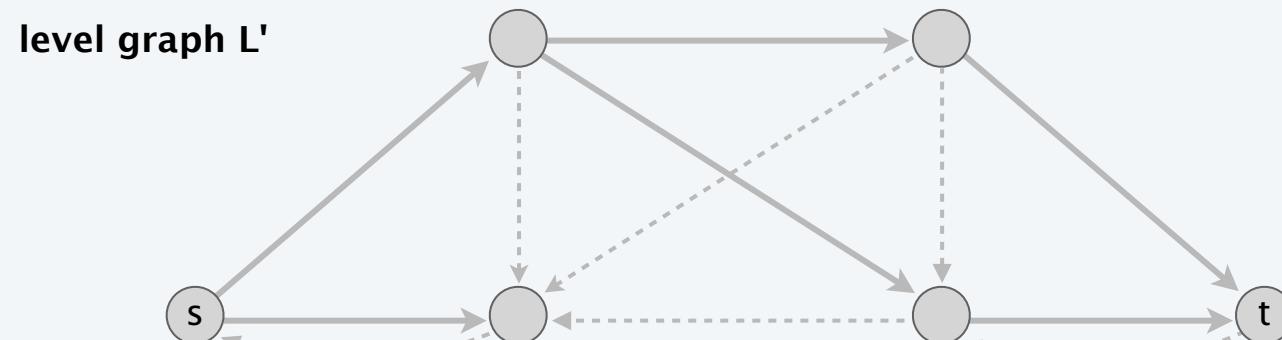
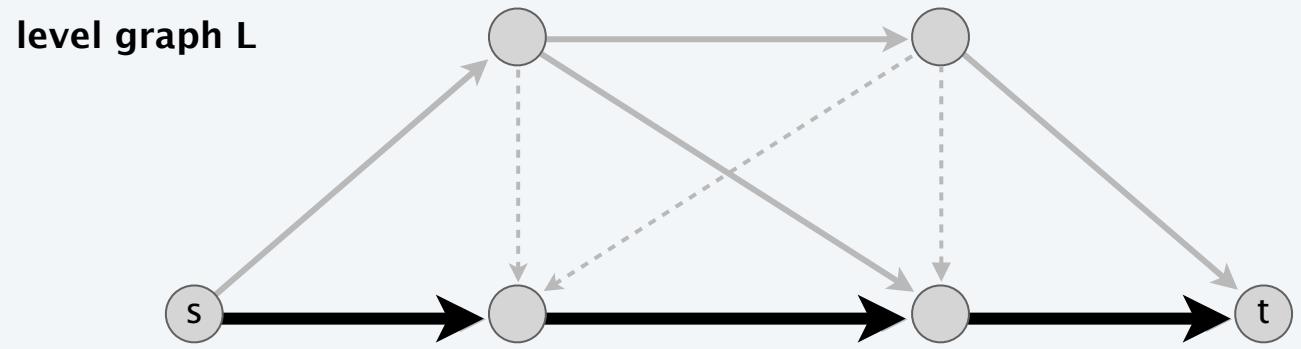
- Let  $f$  and  $f'$  be flow before and after a shortest path augmentation.
- Let  $L$  and  $L'$  be level graphs of  $G_f$  and  $G_{f'}$ .
- Only back edges added to  $G_{f'}$   
(any path with a back edge is longer than previous length) ▀



## Shortest augmenting path: analysis

L2. After at most  $m$  shortest path augmentations, the length of the shortest augmenting path strictly increases.

- The bottleneck edge(s) is deleted from  $L$  after each augmentation.
- No new edge added to  $L$  until length of shortest path strictly increases. ▀



## Shortest augmenting path: **review of analysis**

---

- L1. Throughout the algorithm, length of the shortest path never decreases.
- L2. After at most  $m$  shortest path augmentations, the length of the shortest augmenting path strictly increases.

**Theorem.** The shortest augmenting path algorithm runs in  $O(m^2 n)$  time.

Pf.

- $O(m + n)$  time to find shortest augmenting path via BFS.
- $O(m)$  augmentations for paths of exactly  $k$  edges.
- $O(m n)$  augmentations. ▀

# Shortest augmenting path: **improving the running time**

---

**Note.**  $\Theta(m n)$  augmentations necessary on some networks.

- Try to decrease time per augmentation instead.
- Simple idea  $\Rightarrow O(m n^2)$  [Dinic 1970]
- Dynamic trees  $\Rightarrow O(m n \log n)$  [Sleator-Tarjan 1983]

**A Data Structure for Dynamic Trees**

DANIEL D. SLEATOR AND ROBERT ENDRE TARJAN

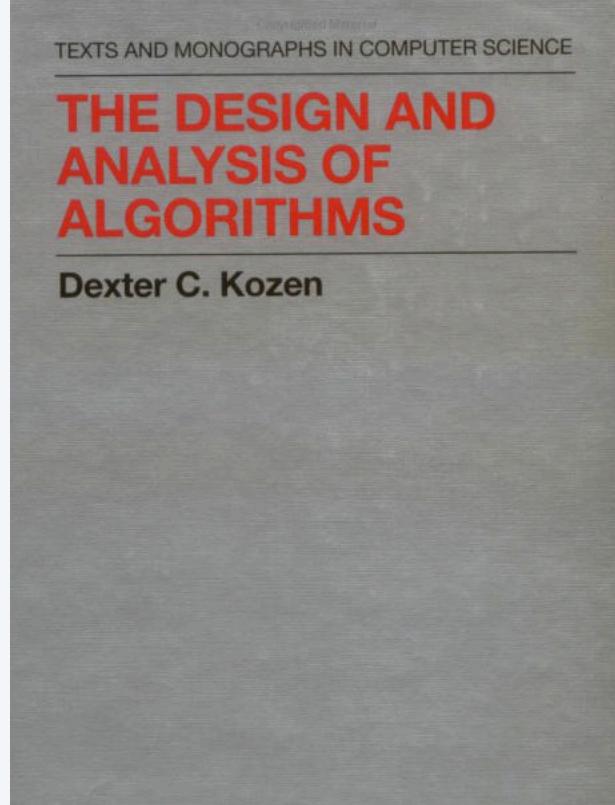
*Bell Laboratories, Murray Hill, New Jersey 07974*

Received May 8, 1982; revised October 18, 1982

A data structure is proposed to maintain a collection of vertex-disjoint trees under a sequence of two kinds of operations: a *link* operation that combines two trees into one by adding an edge, and a *cut* operation that divides one tree into two by deleting an edge. Each operation requires  $O(\log n)$  time. Using this data structure, new fast algorithms are obtained for the following problems:

- (1) Computing nearest common ancestors.
- (2) Solving various network flow problems including finding maximum flows, blocking flows, and acyclic flows.
- (3) Computing certain kinds of constrained minimum spanning trees.
- (4) Implementing the network simplex algorithm for minimum-cost flows.

The most significant application is (2); an  $O(mn \log n)$ -time algorithm is obtained to find a maximum flow in a network of  $n$  vertices and  $m$  edges, beating by a factor of  $\log n$  the fastest algorithm previously known for sparse graphs.



## SECTION 18.1

# 7. NETWORK FLOW I

---

- ▶ *max-flow and min-cut problems*
- ▶ *Ford-Fulkerson algorithm*
- ▶ *max-flow min-cut theorem*
- ▶ *capacity-scaling algorithm*
- ▶ *shortest augmenting paths*
- ▶ *blocking-flow algorithm*
- ▶ *unit-capacity simple networks*

# Blocking-flow algorithm

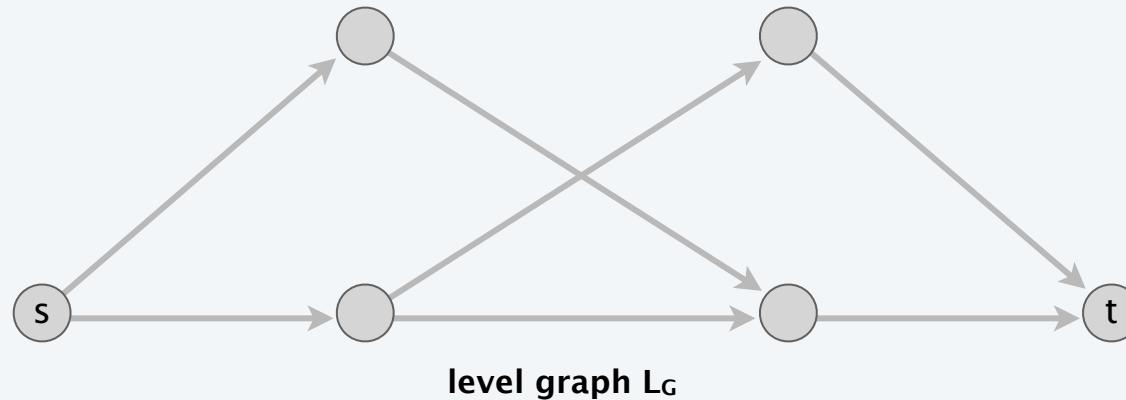
---

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph  $L_G$ .
- Start at  $s$ , advance along an edge in  $L_G$  until reach  $t$  or get stuck.
- If reach  $t$ , augment and update  $L_G$ .
- If get stuck, delete node from  $L_G$  and go to previous node.



# Blocking-flow algorithm

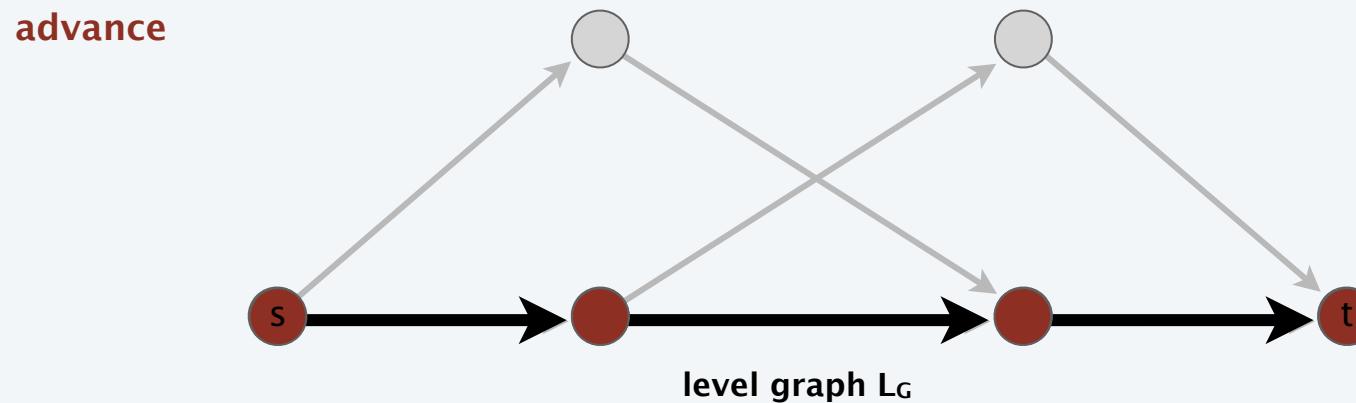
---

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph  $L_G$ .
- Start at  $s$ , advance along an edge in  $L_G$  until reach  $t$  or get stuck.
- If reach  $t$ , augment and update  $L_G$ .
- If get stuck, delete node from  $L_G$  and go to previous node.



# Blocking-flow algorithm

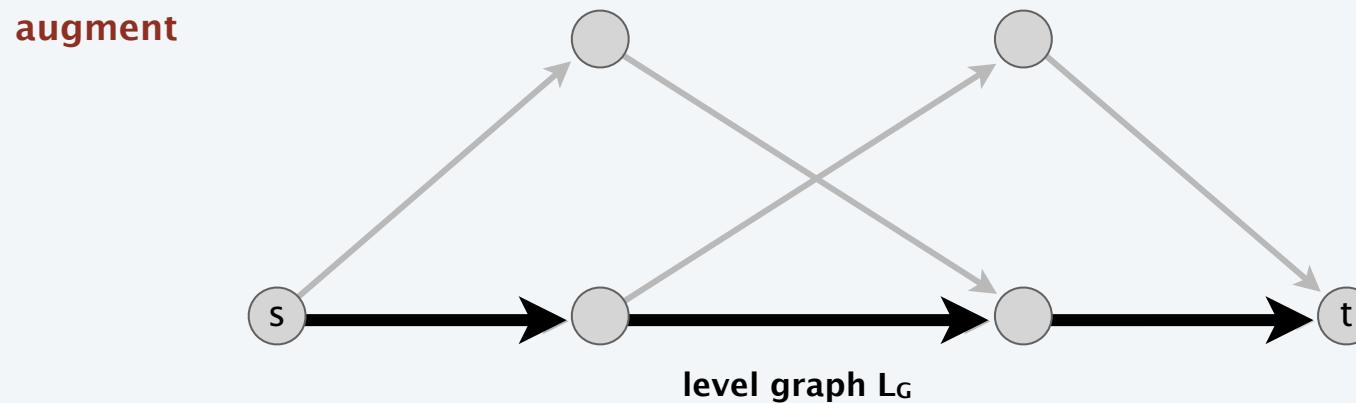
---

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph  $L_G$ .
- Start at  $s$ , advance along an edge in  $L_G$  until reach  $t$  or get stuck.
- If reach  $t$ , augment and update  $L_G$ .
- If get stuck, delete node from  $L_G$  and go to previous node.



# Blocking-flow algorithm

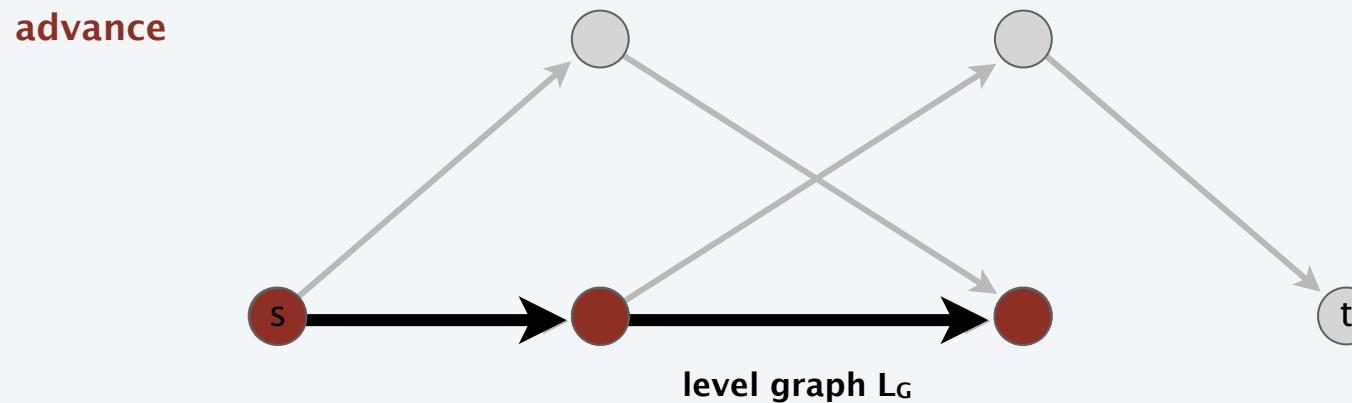
---

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph  $L_G$ .
- Start at  $s$ , advance along an edge in  $L_G$  until reach  $t$  or get stuck.
- If reach  $t$ , augment and update  $L_G$ .
- If get stuck, delete node from  $L_G$  and go to previous node.



# Blocking-flow algorithm

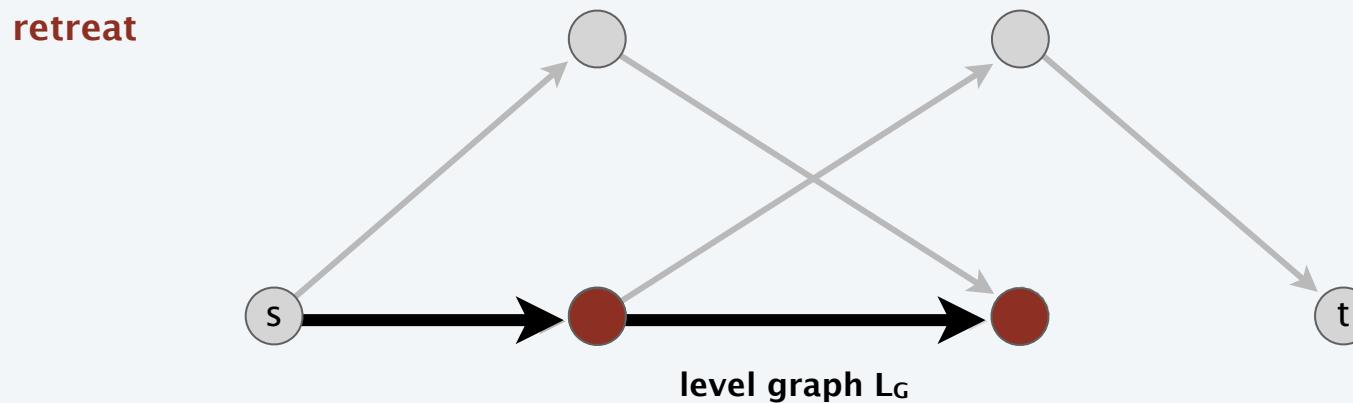
---

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph  $L_G$ .
- Start at  $s$ , advance along an edge in  $L_G$  until reach  $t$  or get stuck.
- If reach  $t$ , augment and update  $L_G$ .
- If get stuck, delete node from  $L_G$  and go to previous node.



# Blocking-flow algorithm

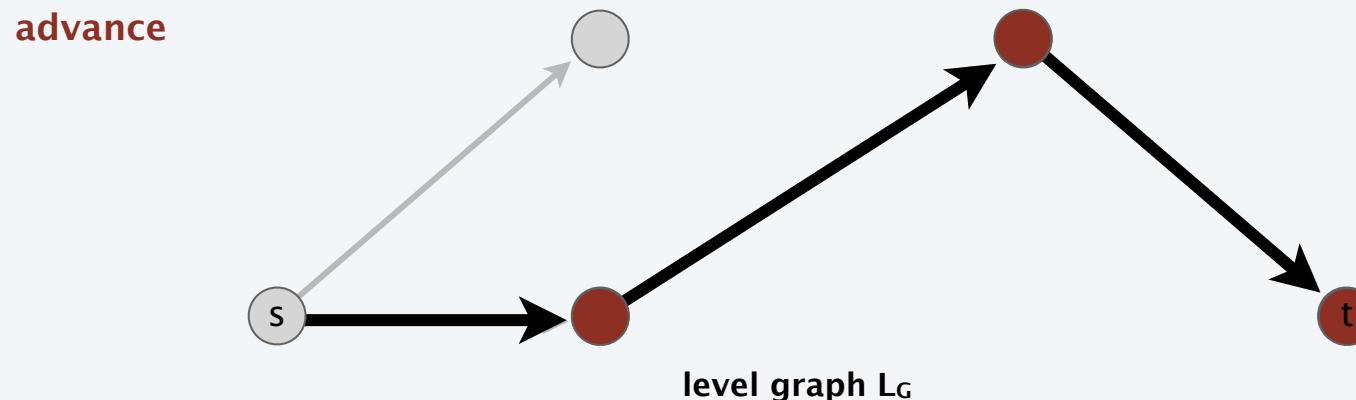
---

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph  $L_G$ .
- Start at  $s$ , advance along an edge in  $L_G$  until reach  $t$  or get stuck.
- If reach  $t$ , augment and update  $L_G$ .
- If get stuck, delete node from  $L_G$  and go to previous node.



# Blocking-flow algorithm

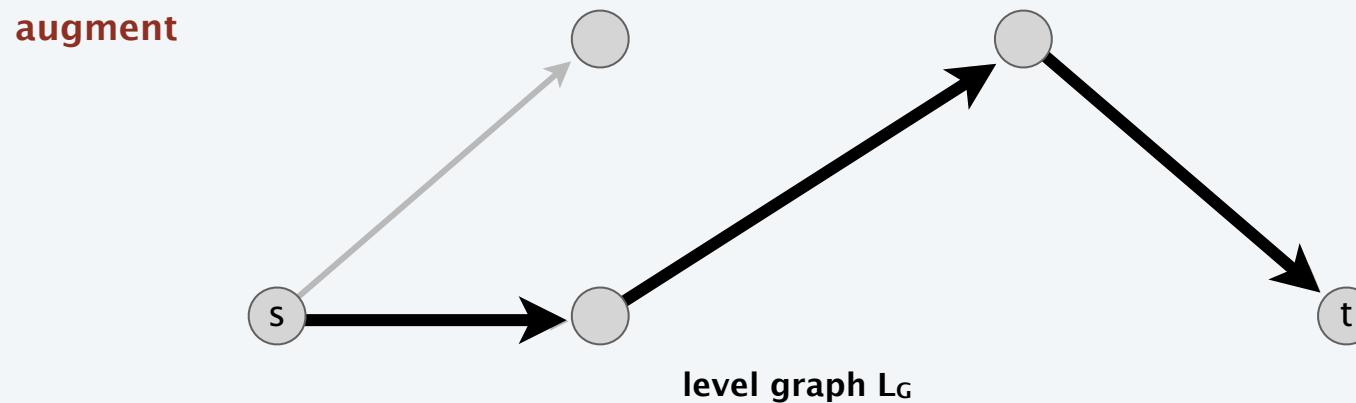
---

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph  $L_G$ .
- Start at  $s$ , advance along an edge in  $L_G$  until reach  $t$  or get stuck.
- If reach  $t$ , augment and update  $L_G$ .
- If get stuck, delete node from  $L_G$  and go to previous node.



# Blocking-flow algorithm

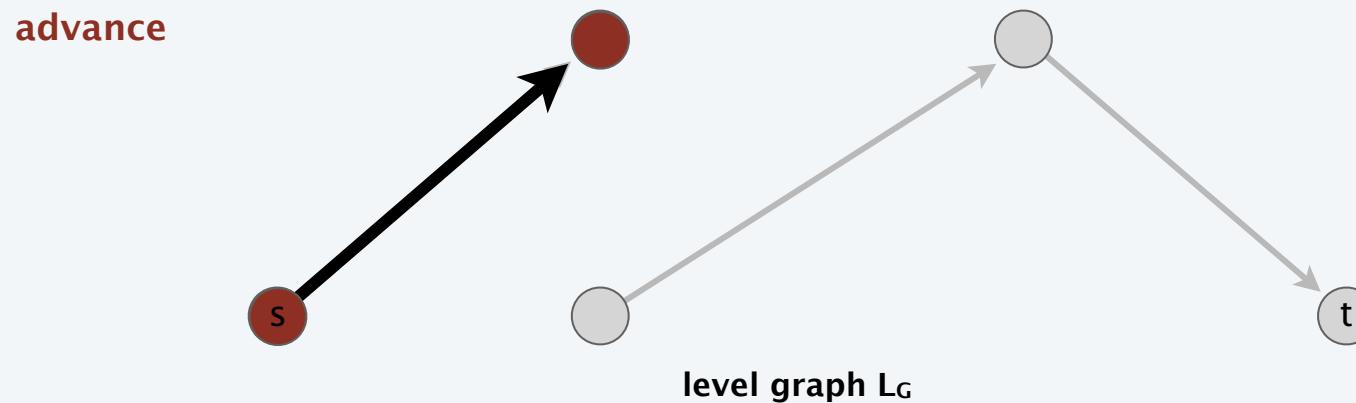
---

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph  $L_G$ .
- Start at  $s$ , advance along an edge in  $L_G$  until reach  $t$  or get stuck.
- If reach  $t$ , augment and update  $L_G$ .
- If get stuck, delete node from  $L_G$  and go to previous node.



# Blocking-flow algorithm

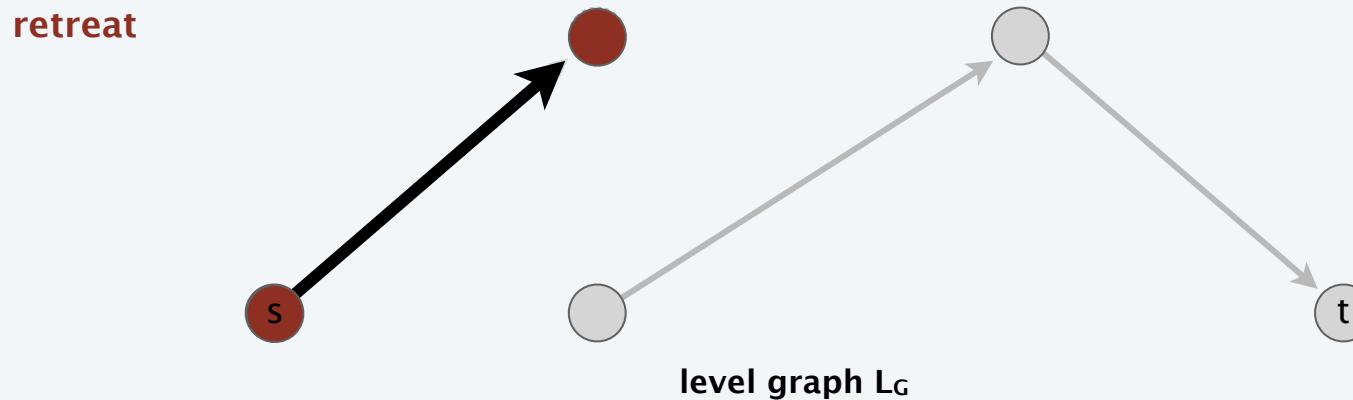
---

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph  $L_G$ .
- Start at  $s$ , advance along an edge in  $L_G$  until reach  $t$  or get stuck.
- If reach  $t$ , augment and update  $L_G$ .
- If get stuck, delete node from  $L_G$  and go to previous node.



# Blocking-flow algorithm

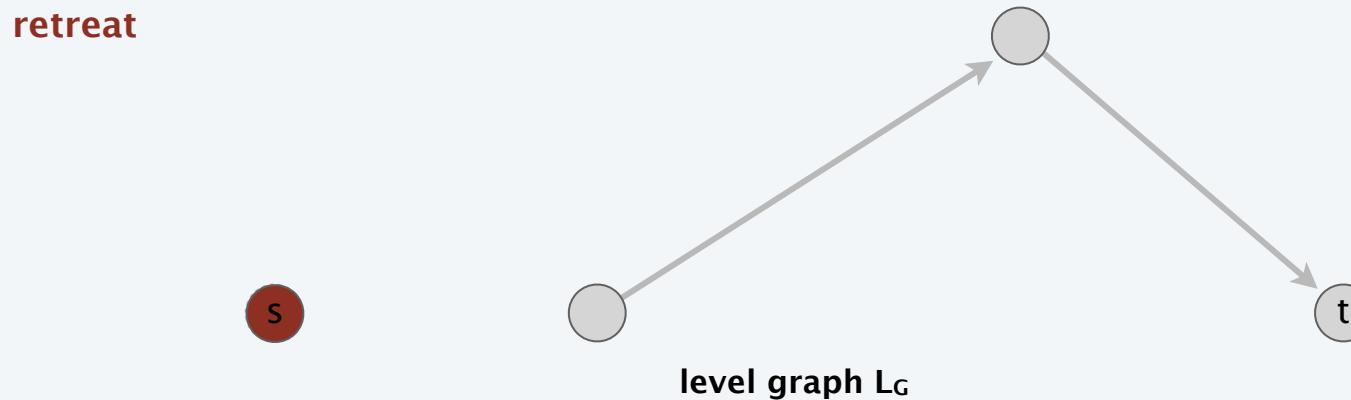
---

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph  $L_G$ .
- Start at  $s$ , advance along an edge in  $L_G$  until reach  $t$  or get stuck.
- If reach  $t$ , augment and update  $L_G$ .
- If get stuck, delete node from  $L_G$  and go to previous node.



# Blocking-flow algorithm

---

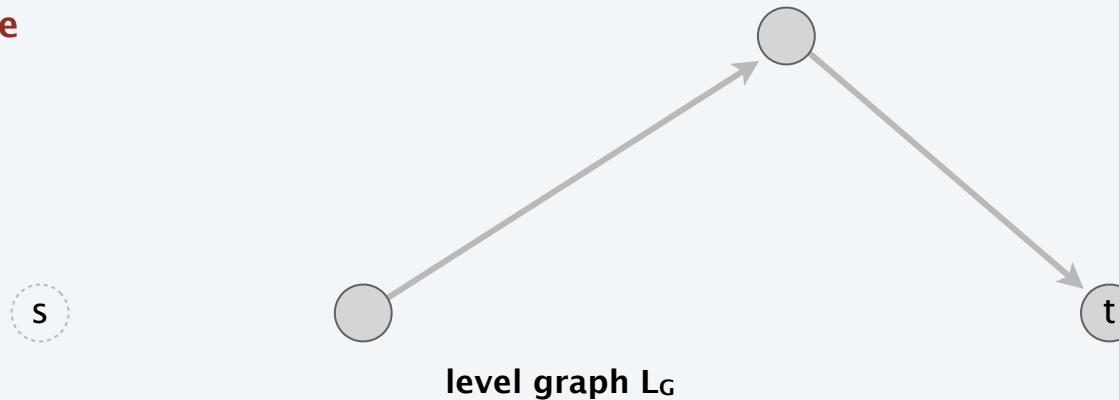
Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Explicitly maintain level graph  $L_G$ .
- Start at  $s$ , advance along an edge in  $L_G$  until reach  $t$  or get stuck.
- If reach  $t$ , augment and update  $L_G$ .
- If get stuck, delete node from  $L_G$  and go to previous node.

end of phase



# Blocking-flow algorithm

---

**INITIALIZE**( $G, s, t, f, c$ )

$L_G \leftarrow$  level-graph of  $G_f$ .

$P \leftarrow \emptyset$ .

**GOTO ADVANCE**( $s$ ).

**RETREAT**( $v$ )

**IF** ( $v = s$ ) **STOP**.

**ELSE**

Delete  $v$  (and all incident edges) from  $L_G$ .

Remove last edge  $(u, v)$  from  $P$ .

**GOTO ADVANCE**( $u$ ).

**ADVANCE**( $v$ )

**IF** ( $v = t$ )

**AUGMENT**( $P$ ).

Remove saturated edges from  $L_G$ .

$P \leftarrow \emptyset$ .

**GOTO ADVANCE**( $s$ ).

**IF** (there exists edge  $(v, w) \in L_G$ )

Add edge  $(v, w)$  to  $P$ .

**GOTO ADVANCE**( $w$ ).

**ELSE GOTO RETREAT**( $v$ ).

## Blocking-flow algorithm: analysis

---

**Lemma.** A phase can be implemented in  $O(mn)$  time.

Pf.

- Initialization happens once per phase.  $\leftarrow O(m)$  using BFS
- At most  $m$  augmentations per phase.  $\leftarrow O(mn)$  per phase  
(because an augmentation deletes at least one edge from  $L_G$ )
- At most  $n$  retreats per phase.  $\leftarrow O(m + n)$  per phase  
(because a retreat deletes one node from  $L_G$ )
- At most  $mn$  advances per phase.  $\leftarrow O(mn)$  per phase  
(because at most  $n$  advances before retreat or augmentation) ▀

**Theorem.** [Dinic 1970] The blocking-flow algorithm runs in  $O(mn^2)$  time.

Pf.

- By lemma,  $O(mn)$  time per phase.
- At most  $n$  phases (as in shortest augment path analysis). ▀

# Choosing good augmenting paths: summary

---

Assumption. Integer capacities between 1 and  $C$ .

method	# augmentations	running time
augmenting path	$n C$	$O(m n C)$
fattest augmenting path	$m \log (mC)$	$O(m^2 \log n \log (mC))$
capacity scaling	$m \log C$	$O(m^2 \log C)$
improved capacity scaling	$m \log C$	$O(m n \log C)$
shortest augmenting path	$m n$	$O(m^2 n)$
improved shortest augmenting path	$m n$	$O(m n^2)$
dynamic trees	$m n$	$O(m n \log n)$

# Maximum flow algorithms: theory

---

year	method	worst case	discovered by
1951	simplex	$O(m^3 C)$	Dantzig
1955	augmenting path	$O(m^2 C)$	Ford-Fulkerson
1970	shortest augmenting path	$O(m^3)$	Dinic, Edmonds-Karp
1970	fattest augmenting path	$O(m^2 \log m \log(mC))$	Dinic, Edmonds-Karp
1977	blocking flow	$O(m^{5/2})$	Cherkasky
1978	blocking flow	$O(m^{7/3})$	Galil
1983	dynamic trees	$O(m^2 \log m)$	Sleator-Tarjan
1985	capacity scaling	$O(m^2 \log C)$	Gabow
1997	length function	$O(m^{3/2} \log m \log C)$	Goldberg-Rao
2012	compact network	$O(m^2 / \log m)$	Orlin
?	?	$O(m)$	?

max-flow algorithms for sparse digraphs with  $m$  edges, integer capacities between 1 and  $C$

# Maximum flow algorithms: practice

---

Push-relabel algorithm (SECTION 7.4). [Goldberg-Tarjan 1988]

Increases flow one edge at a time instead of one augmenting path at a time.

## A New Approach to the Maximum-Flow Problem

ANDREW V. GOLDBERG

*Massachusetts Institute of Technology, Cambridge, Massachusetts*

AND

ROBERT E. TARJAN

*Princeton University, Princeton, New Jersey, and AT&T Bell Laboratories, Murray Hill, New Jersey*

**Abstract.** All previously known efficient maximum-flow algorithms work by finding augmenting paths, either one path at a time (as in the original Ford and Fulkerson algorithm) or all shortest-length augmenting paths at once (using the layered network approach of Dinic). An alternative method based on the *preflow* concept of Karzanov is introduced. A preflow is like a flow, except that the total amount flowing into a vertex is allowed to exceed the total amount flowing out. The method maintains a preflow in the original network and pushes local flow excess toward the sink along what are estimated to be shortest paths. The algorithm and its analysis are simple and intuitive, yet the algorithm runs as fast as any other known method on dense graphs, achieving an  $O(n^3)$  time bound on an  $n$ -vertex graph. By incorporating the dynamic tree data structure of Sleator and Tarjan, we obtain a version of the algorithm running in  $O(nm \log(n^2/m))$  time on an  $n$ -vertex,  $m$ -edge graph. This is as fast as any known method for any graph density and faster on graphs of moderate density. The algorithm also admits efficient distributed and parallel implementations. A parallel implementation running in  $O(n^2 \log n)$  time using  $n$  processors and  $O(m)$  space is obtained. This time bound matches that of the Shiloach-Vishkin algorithm, which also uses  $n$  processors but requires  $O(n^2)$  space.

# Maximum flow algorithms: practice

**Warning.** Worst-case running time is generally not useful for predicting or comparing max-flow algorithm performance in practice.

**Best in practice.** Push-relabel method with gap relabeling:  $O(m^{3/2})$ .

## On Implementing Push-Relabel Method for the Maximum Flow Problem

Boris V. Cherkassky<sup>1</sup> and Andrew V. Goldberg<sup>2</sup>

<sup>1</sup> Central Institute for Economics and Mathematics,  
Krasikova St. 32, 117418, Moscow, Russia  
*cher@cemii.msk.su*

<sup>2</sup> Computer Science Department, Stanford University  
Stanford, CA 94305, USA  
*goldberg@cs.stanford.edu*

**Abstract.** We study efficient implementations of the push-relabel method for the maximum flow problem. The resulting codes are faster than the previous codes, and much faster on some problem families. The speedup is due to the combination of heuristics used in our implementations. We also exhibit a family of problems for which the running time of all known methods seem to have a roughly quadratic growth rate.



European Journal of Operational Research 97 (1997) 509–542

EUROPEAN  
JOURNAL  
OF OPERATIONAL  
RESEARCH

## Theory and Methodology Computational investigations of maximum flow algorithms

Ravindra K. Ahuja <sup>a</sup>, Murali Kodialam <sup>b</sup>, Ajay K. Mishra <sup>c</sup>, James B. Orlin <sup>d,\*</sup>

<sup>a</sup> Department of Industrial and Management Engineering, Indian Institute of Technology, Kanpur, 208 016, India

<sup>b</sup> AT & T Bell Laboratories, Holmdel, NJ 07733, USA

<sup>c</sup> Katz Graduate School of Business, University of Pittsburgh, Pittsburgh, PA 15260, USA

<sup>d</sup> Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

Received 30 August 1995; accepted 27 June 1996

# Maximum flow algorithms: practice

Computer vision. Different algorithms work better for some dense problems that arise in applications to computer vision.

An Experimental Comparison of  
Min-Cut/Max-Flow Algorithms for  
Energy Minimization in Vision

Yuri Boykov and Vladimir Kolmogorov\*

## Abstract

After [15, 31, 19, 8, 25, 5] minimum cut/maximum flow algorithms on graphs emerged as an increasingly useful tool for exact or approximate energy minimization in low-level vision. The combinatorial optimization literature provides many min-cut/max-flow algorithms with different polynomial time complexity. Their practical efficiency, however, has to date been studied mainly outside the scope of computer vision. The goal of this paper is to provide an experimental comparison of the efficiency of min-cut/max flow algorithms for applications in vision. We compare the running times of several standard algorithms, as well as a new algorithm that we have recently developed. The algorithms we study include both Goldberg-Tarjan style “push-relabel” methods and algorithms based on Ford-Fulkerson style “augmenting paths”. We benchmark these algorithms on a number of typical graphs in the contexts of image restoration, stereo, and segmentation. In many cases our new algorithm works several times faster than any of the other methods making near real-time performance possible. An implementation of our max-flow/min-cut algorithm is available upon request for research purposes.

VERMA, BATRA: MAXFLOW REVISITED

1

## MaxFlow Revisited: An Empirical Comparison of Maxflow Algorithms for Dense Vision Problems

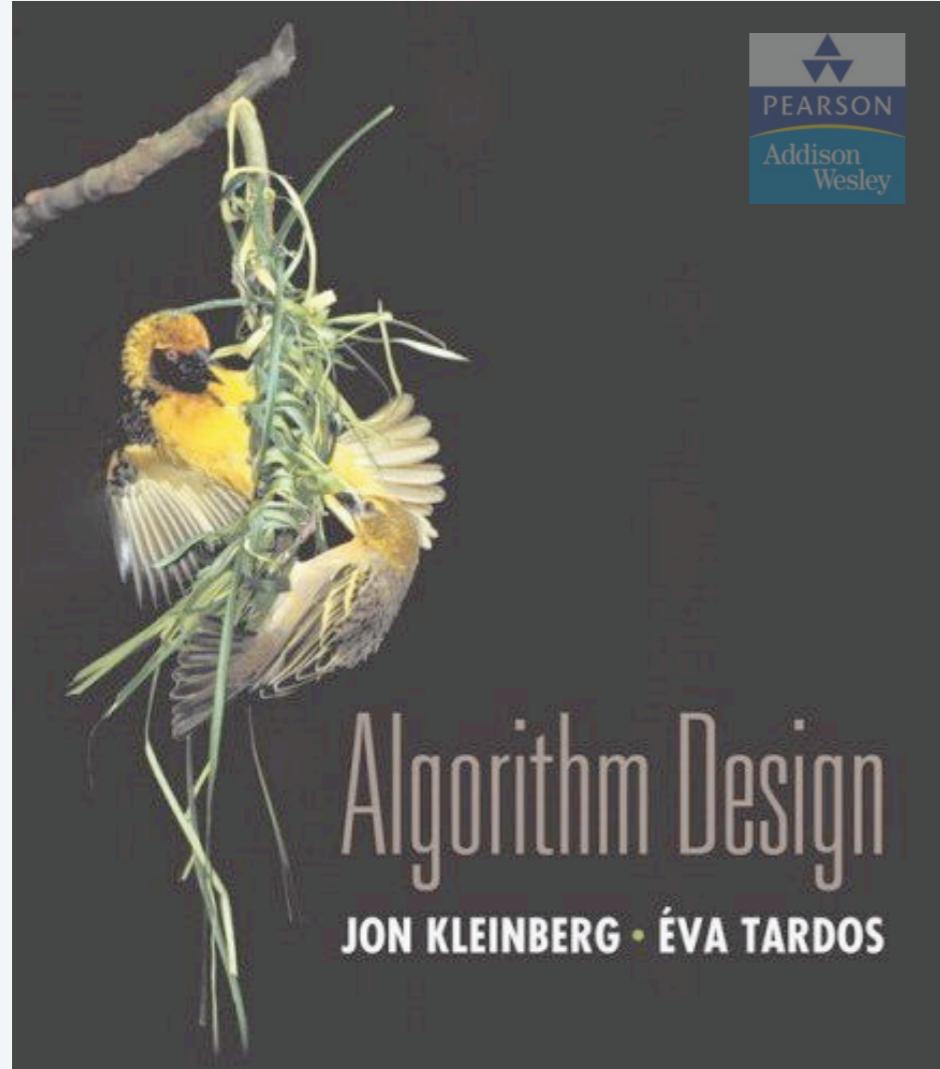
Tanmay Verma  
[tanmay08054@iiitd.ac.in](mailto:tanmay08054@iiitd.ac.in)

Dhruv Batra  
[dbatra@ttic.edu](mailto:dbatra@ttic.edu)

IIIT-Delhi  
Delhi, India  
TTI-Chicago  
Chicago, USA

## Abstract

Algorithms for finding the maximum amount of flow possible in a network (or max-flow) play a central role in computer vision problems. We present an empirical comparison of different max-flow algorithms on modern problems. Our problem instances arise from energy minimization problems in Object Category Segmentation, Image Deconvolution, Super Resolution, Texture Restoration, Character Completion and 3D Segmentation. We compare 14 different implementations and find that the most popularly used implementation of Kolmogorov [5] is no longer the fastest algorithm available, especially for dense graphs.



## SECTION

# 7. NETWORK FLOW II

---

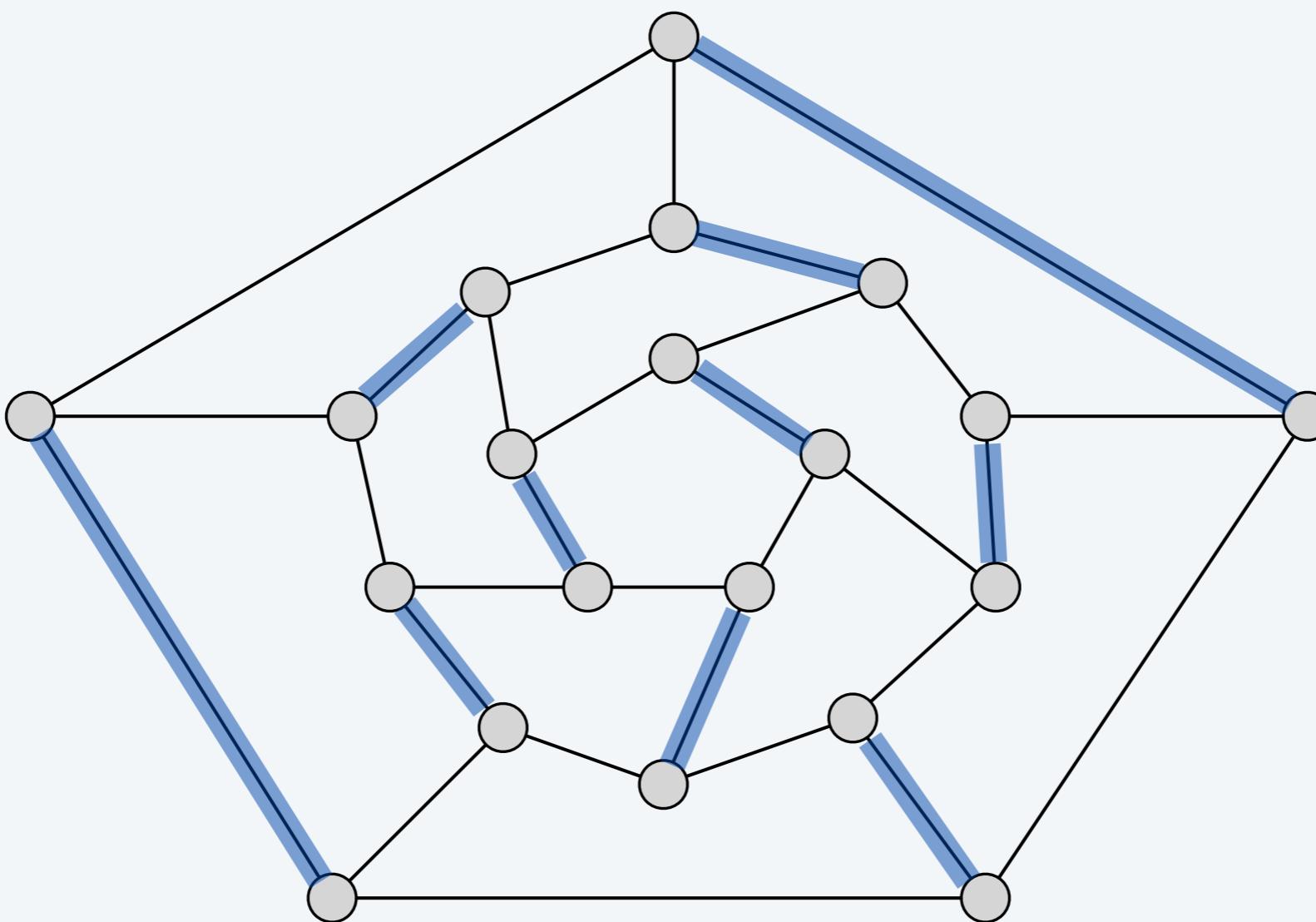
- ▶ *bipartite matching*
- ▶ *disjoint paths*
- ▶ *extensions to max flow*
- ▶ *survey design*
- ▶ *airline scheduling*
- ▶ *image segmentation*
- ▶ *project selection*
- ▶ *baseball elimination*

## Matching

---

**Def.** Given an undirected graph  $G = (V, E)$  a subset of edges  $M \subseteq E$  is a **matching** if each node appears in at most one edge in  $M$ .

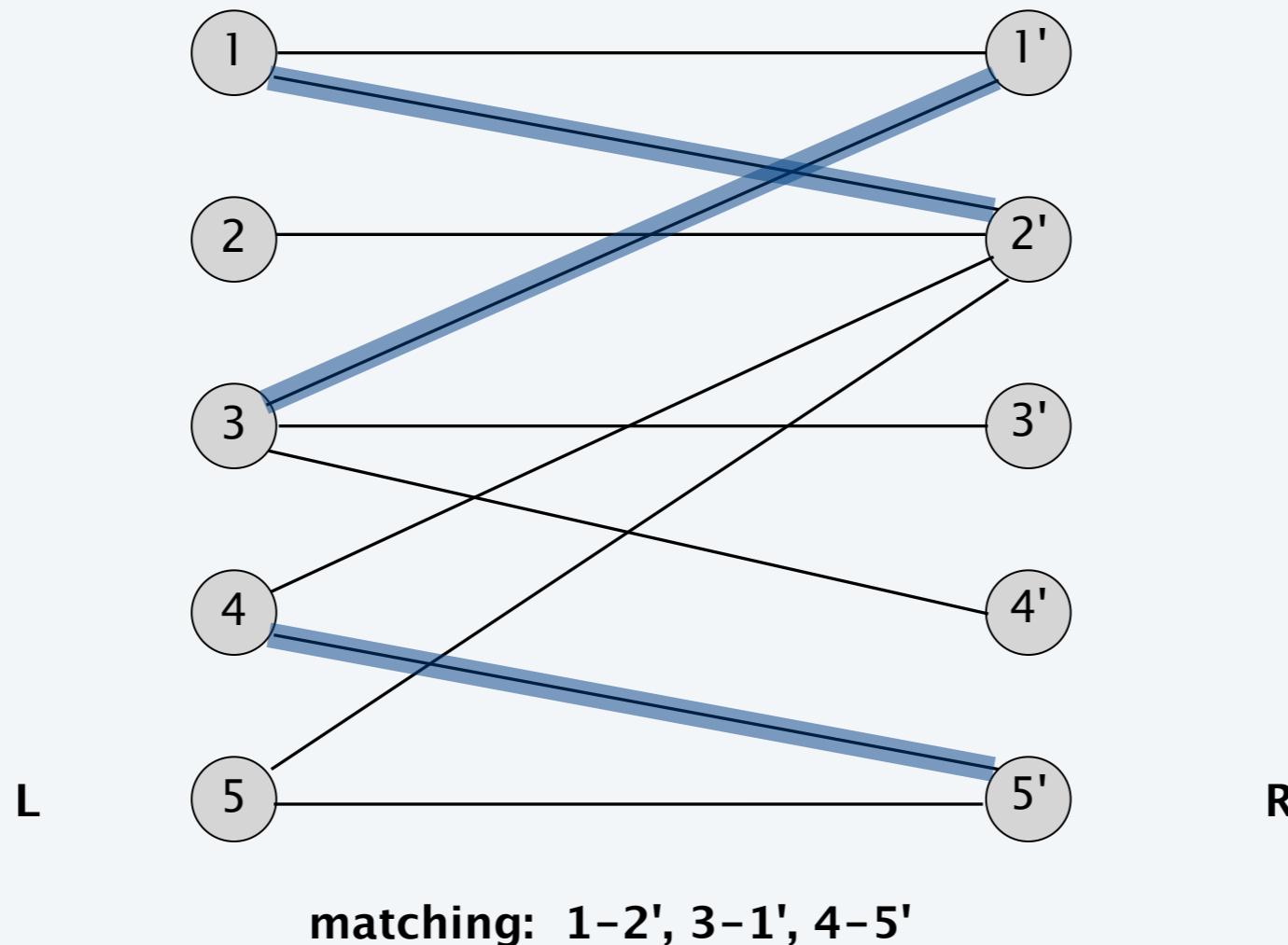
**Max matching.** Given a graph, find a max cardinality matching.



## Bipartite matching

**Def.** A graph  $G$  is bipartite if the nodes can be partitioned into two subsets  $L$  and  $R$  such that every edge connects a node in  $L$  to one in  $R$ .

**Bipartite matching.** Given a bipartite graph  $G = (L \cup R, E)$ , find a max cardinality matching.

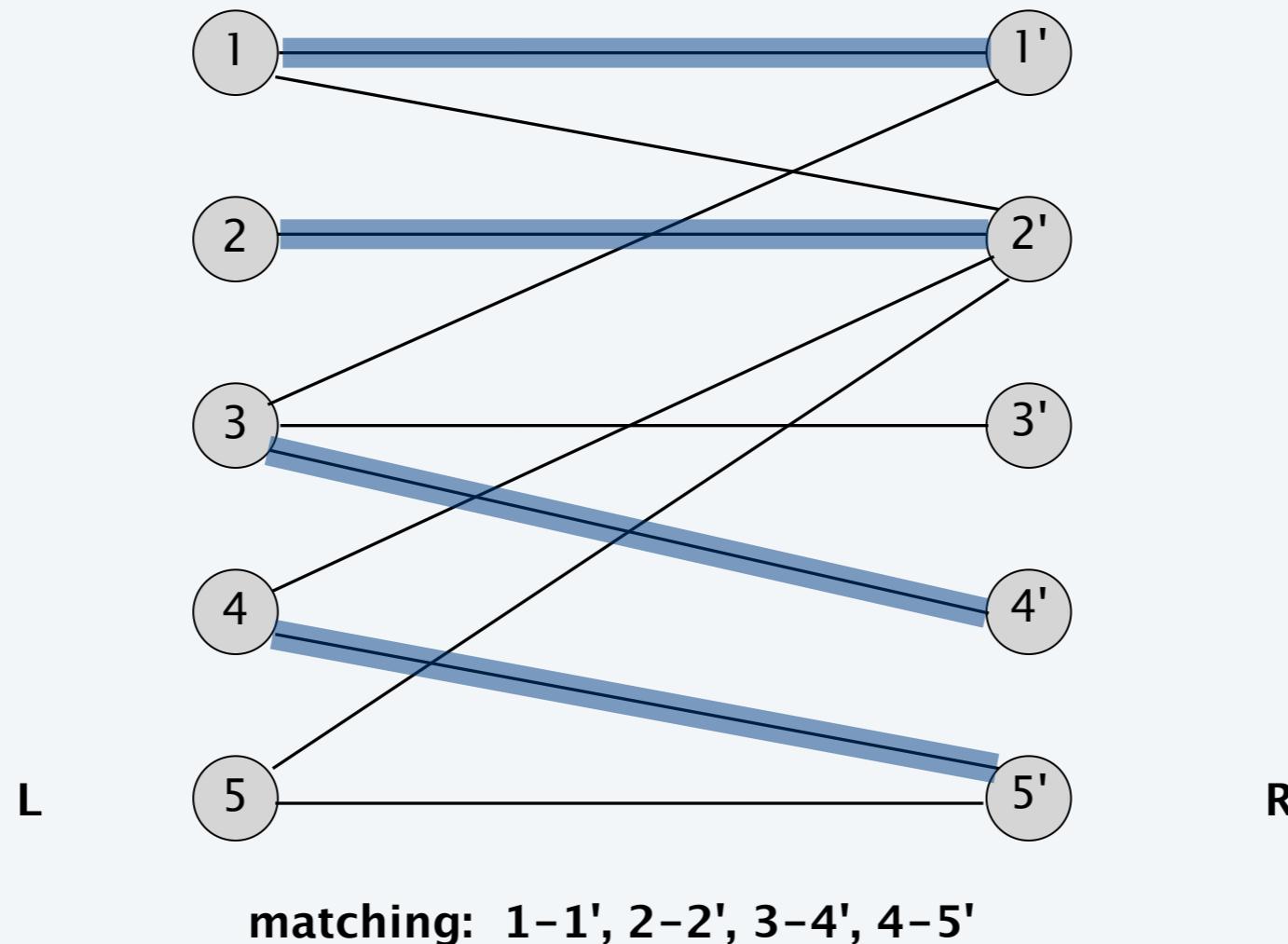


# Bipartite matching

---

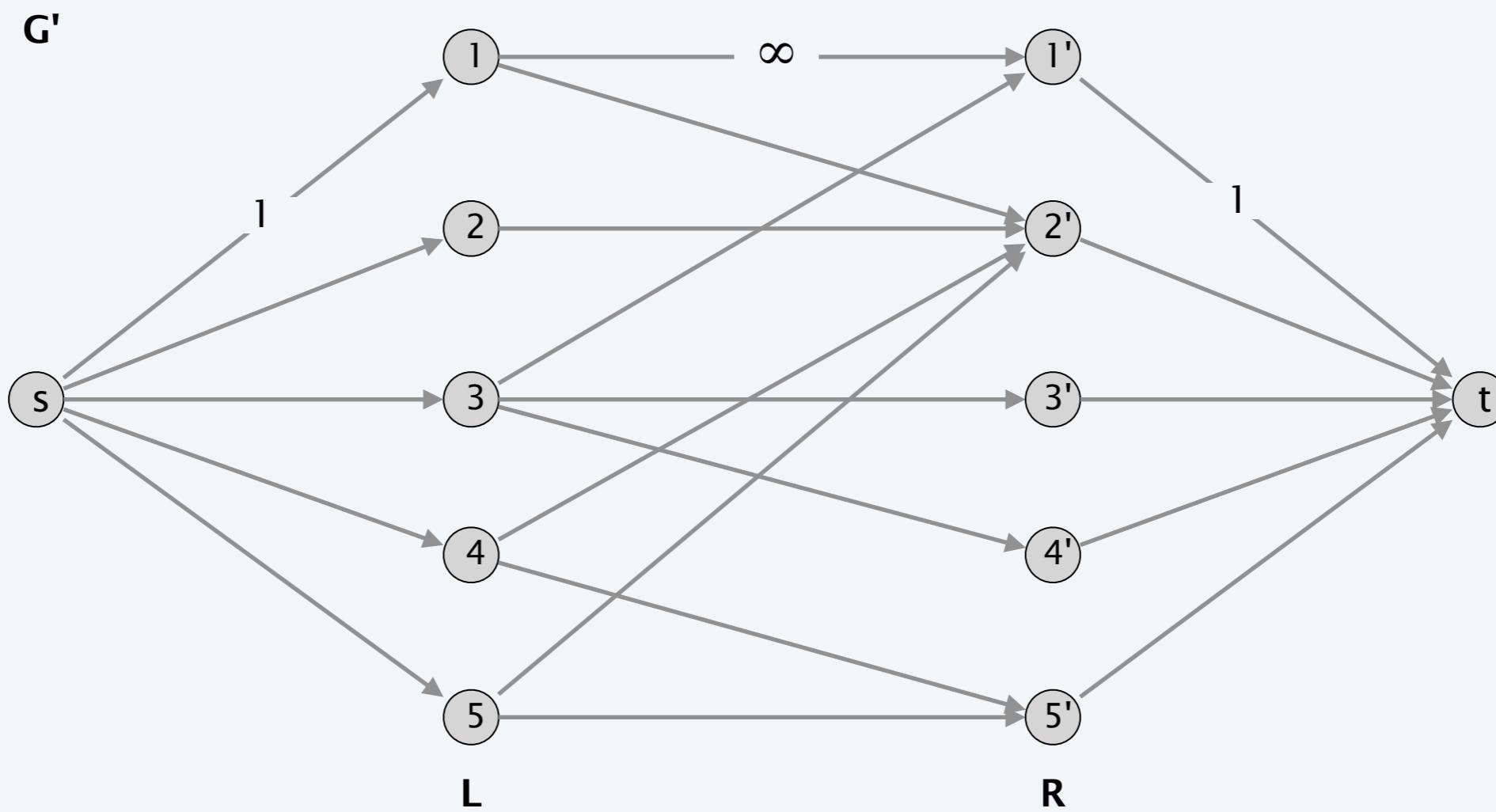
**Def.** A graph  $G$  is **bipartite** if the nodes can be partitioned into two subsets  $L$  and  $R$  such that every edge connects a node in  $L$  to one in  $R$ .

**Bipartite matching.** Given a bipartite graph  $G = (L \cup R, E)$ , find a max cardinality matching.



## Bipartite matching: max flow formulation

- Create digraph  $G' = (L \cup R \cup \{s, t\}, E')$ .
- Direct all edges from  $L$  to  $R$ , and assign infinite (or unit) capacity.
- Add source  $s$ , and unit capacity edges from  $s$  to each node in  $L$ .
- Add sink  $t$ , and unit capacity edges from each node in  $R$  to  $t$ .

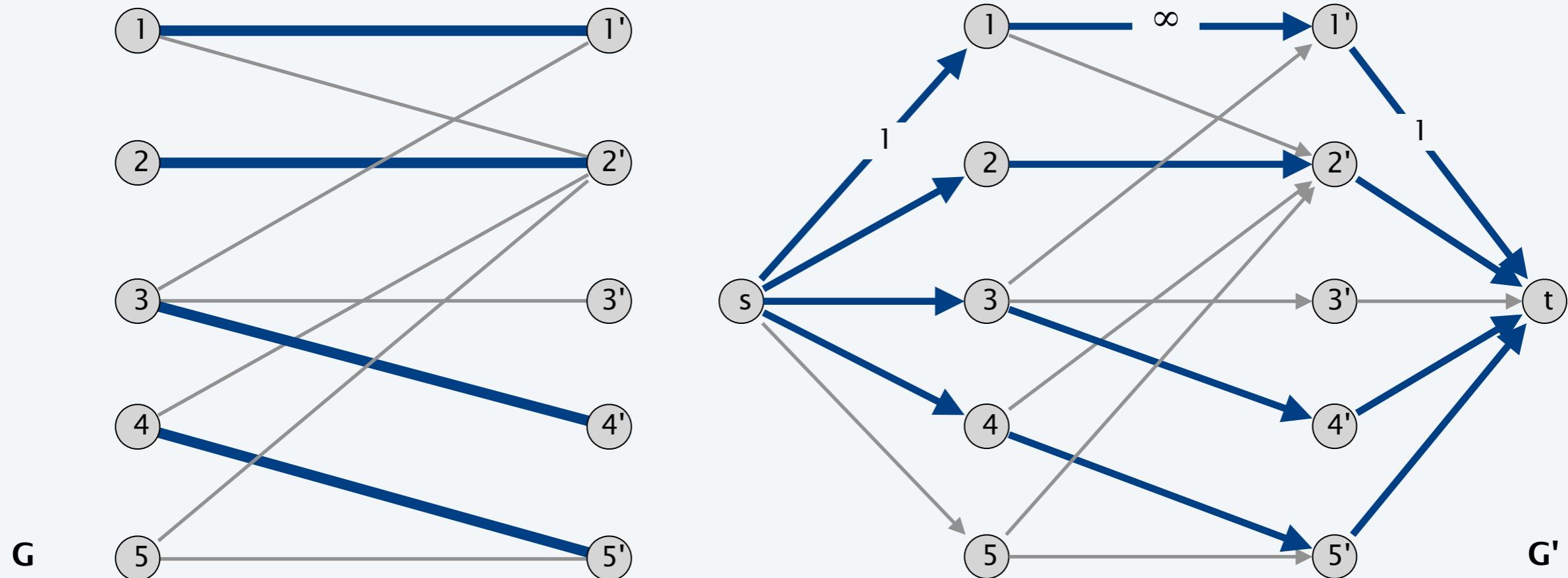


## Max flow formulation: proof of correctness

**Theorem.** Max cardinality of a matching in  $G$  = value of max flow in  $G'$ .

Pf.  $\leq$

- Given a max matching  $M$  of cardinality  $k$ .
- Consider flow  $f$  that sends 1 unit along each of  $k$  paths.
- $f$  is a flow, and has value  $k$ . ■

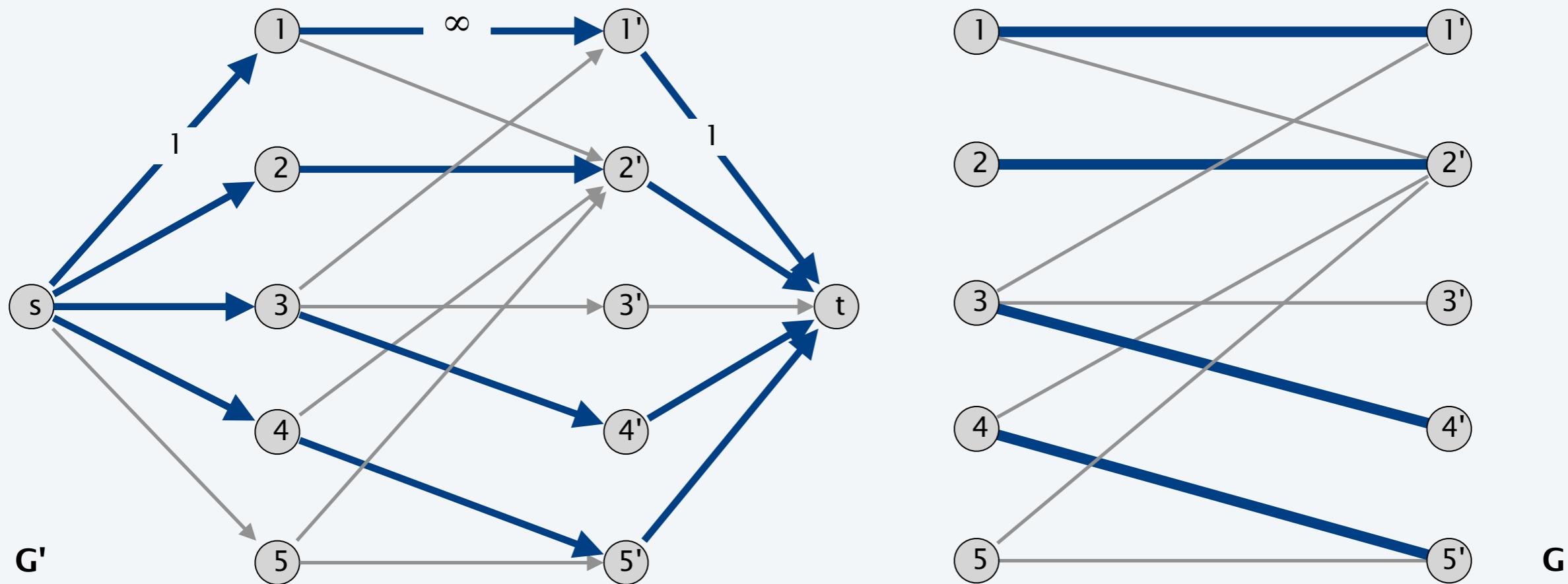


# Max flow formulation: proof of correctness

Theorem. Max cardinality of a matching in  $G$  = value of max flow in  $G'$ .

Pf.  $\geq$

- Let  $f$  be a max flow in  $G'$  of value  $k$ .
- Integrality theorem  $\Rightarrow k$  is integral and can assume  $f$  is 0-1.
- Consider  $M$  = set of edges from  $L$  to  $R$  with  $f(e) = 1$ .
  - each node in  $L$  and  $R$  participates in at most one edge in  $M$
  - $|M| = k$ : consider cut  $(L \cup s, R \cup t)$  ■



## Perfect matching in a bipartite graph

---

**Def.** Given a graph  $G = (V, E)$  a subset of edges  $M \subseteq E$  is a **perfect matching** if each node appears in exactly one edge in  $M$ .

**Q.** When does a bipartite graph have a perfect matching?

**Structure of bipartite graphs with perfect matchings.**

- Clearly we must have  $|L| = |R|$ .
- What other conditions are necessary?
- What conditions are sufficient?

# Perfect matching in a bipartite graph

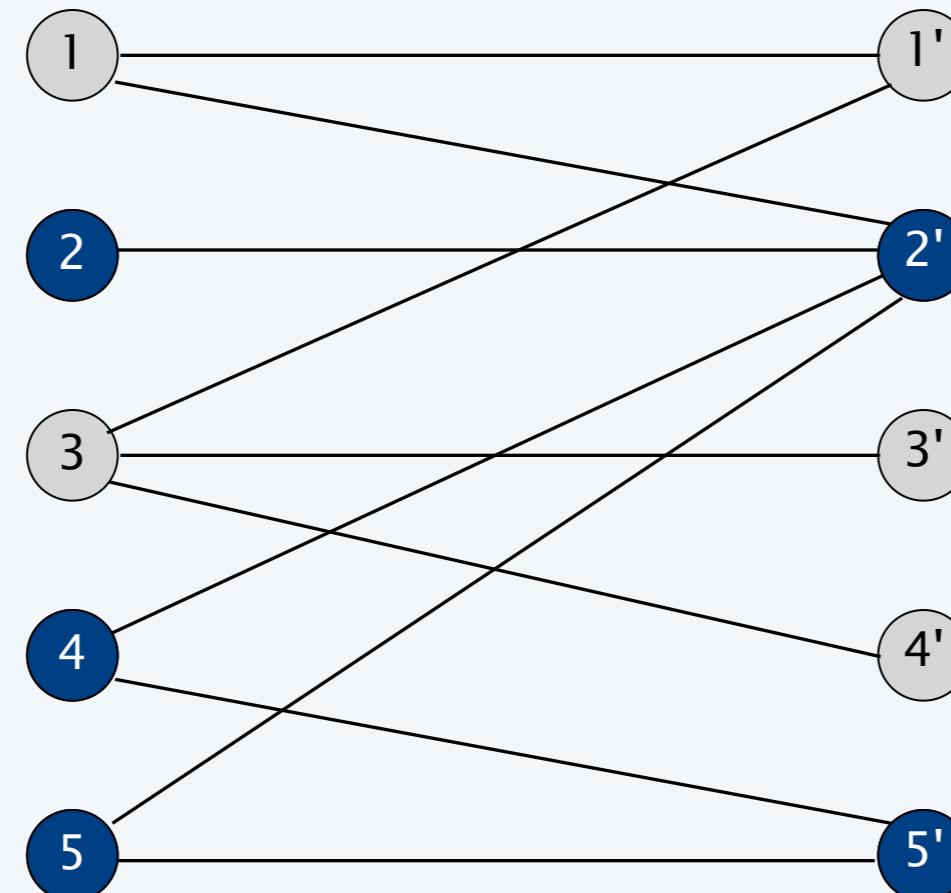
**Notation.** Let  $S$  be a subset of nodes, and let  $N(S)$  be the set of nodes adjacent to nodes in  $S$ .

**Observation.** If a bipartite graph  $G = (L \cup R, E)$  has a perfect matching,

then  $|N(S)| \geq |S|$  for all subsets  $S \subseteq L$ ., is clearly necessary and sufficient

**Pf.** Each node in  $S$  has to be matched to a different node in  $N(S)$ . ▀

subset of nodes  
are always on the left  
 $\uparrow$   
 $S = \{ 2, 4, 5 \}$   
 $\downarrow$   
all the nodes on the right  
that are adjacent to  $s \in S$



no perfect matching

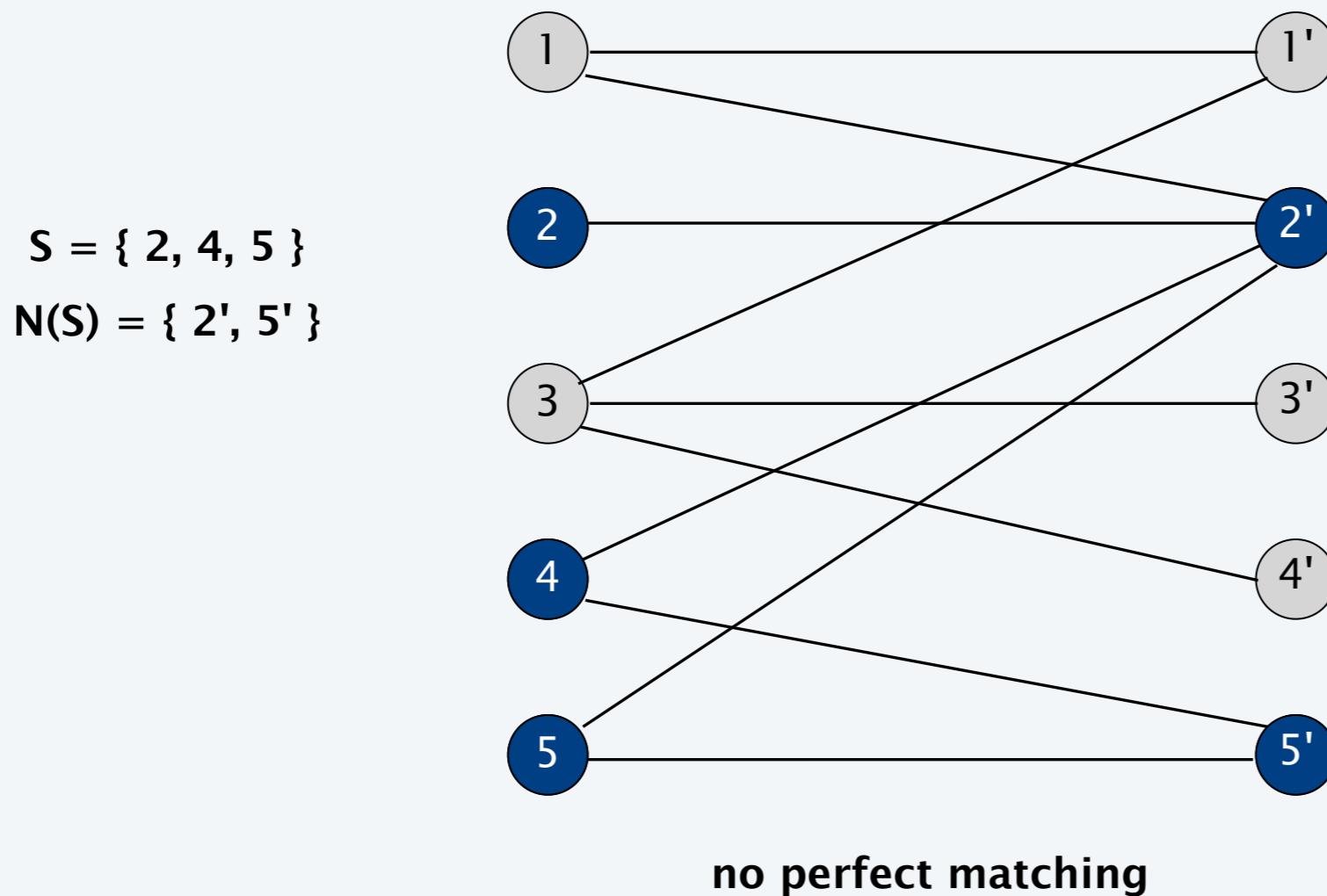
## Hall's theorem

**Theorem.** Let  $G = (L \cup R, E)$  be a bipartite graph with  $|L| = |R|$ .

$G$  has a perfect matching iff  $|N(S)| \geq |S|$  for all subsets  $S \subseteq L$ .

*necessary and sufficient ↴*

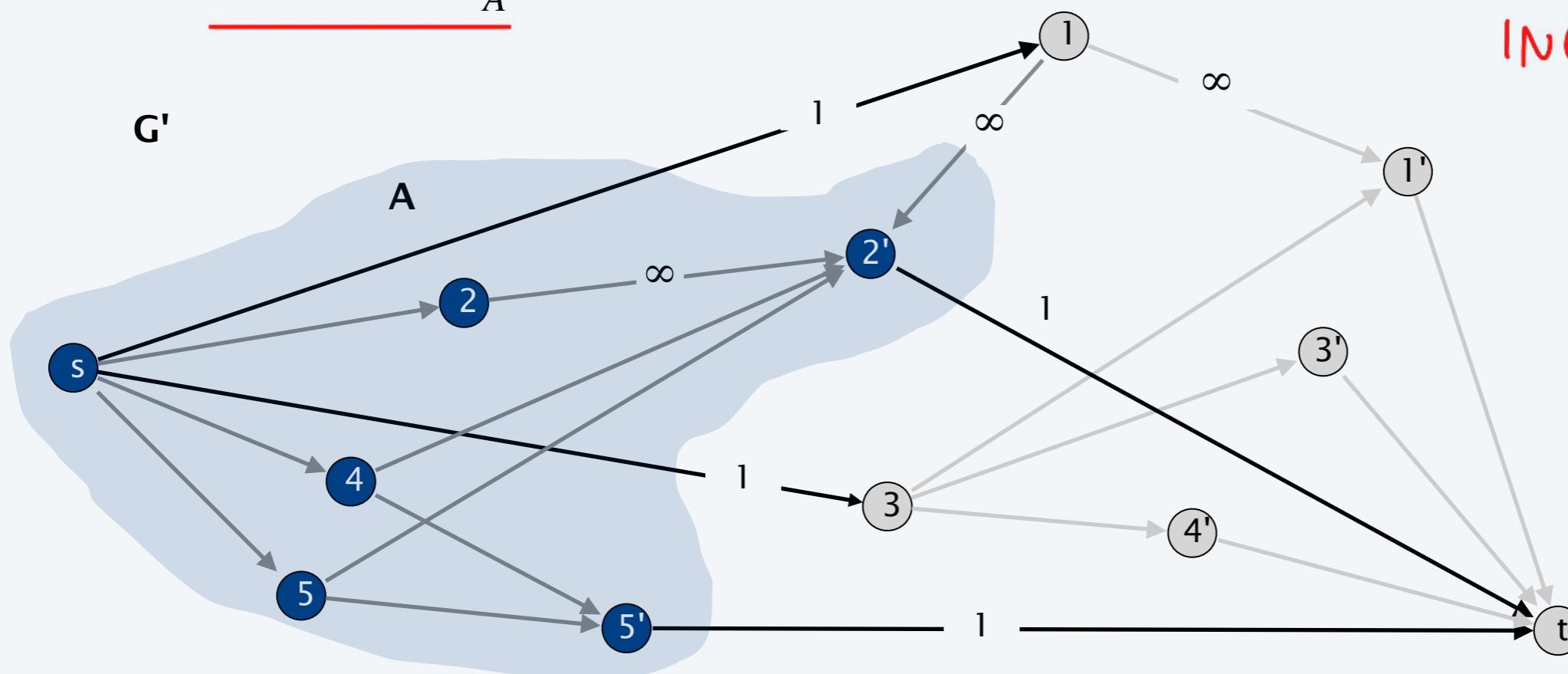
**Pf.**  $\Rightarrow$  This was the previous observation.



## Proof of Hall's theorem

by contradiction, assume no perfect matching

- Pf.  $\Leftarrow$  Suppose  $G$  does not have a perfect matching.
- Formulate as a max flow problem and let  $(A, B)$  be min cut in  $G'$ .
  - By max-flow min-cut theorem,  $cap(A, B) < |L|$ . Capacity of min-cut < cardinality of left vertices  
(no perfect matching)
  - Define  $L_A = L \cap A$ ,  $L_B = L \cap B$ ,  $R_A = R \cap A$ .
  - $cap(A, B) = |L_B| + |R_A|$ .
  - Since min cut can't use  $\infty$  edges:  $N(L_A) \subseteq R_A$ .
  - $|N(L_A)| \leq |R_A| = cap(A, B) - |L_B| < |L| - |L_B| = |L_A|$ . contradiction of  
precedent theorem !  
 $|N(L_A)| \leq |L_A|$
  - Choose  $S = L_A$ . ■



$$\begin{aligned} L_A &= \{2, 4, 5\} \\ L_B &= \{1, 3\} \\ R_A &= \{2', 5'\} \\ N(L_A) &= \{2', 5'\} \end{aligned}$$

## Bipartite matching **running time**

---

**Theorem.** The Ford-Fulkerson algorithm solves the bipartite matching problem in  $O(m n)$  time.

---

**Theorem.** [Hopcroft-Karp 1973] The bipartite matching problem can be solved in  $O(m n^{1/2})$  time.

---

SIAM J. COMPUT.  
Vol. 2, No. 4, December 1973

### AN $n^{5/2}$ ALGORITHM FOR MAXIMUM MATCHINGS IN BIPARTITE GRAPHS\*

JOHN E. HOPCROFT† AND RICHARD M. KARP‡

**Abstract.** The present paper shows how to construct a maximum matching in a bipartite graph with  $n$  vertices and  $m$  edges in a number of computation steps proportional to  $(m + n)\sqrt{n}$ .

**Key words.** algorithm, algorithmic analysis, bipartite graphs, computational complexity, graphs, matching

# Nonbipartite matching

**Nonbipartite matching.** Given an undirected graph (not necessarily bipartite), find a matching of maximum cardinality.

- Structure of nonbipartite graphs is more complicated.
- But well-understood. [Tutte-Berge, Edmonds-Galai]
- Blossom algorithm:  $O(n^4)$ . [Edmonds 1965]
- Best known:  $O(m n^{1/2})$ . [Micali-Vazirani 1980, Vazirani 1994]

## PATHS, TREES, AND FLOWERS

JACK EDMONDS

**1. Introduction.** A *graph*  $G$  for purposes here is a finite set of elements called *vertices* and a finite set of elements called *edges* such that each edge *meets* exactly two vertices, called the *end-points* of the edge. An edge is said to *join* its end-points.

A *matching* in  $G$  is a subset of its edges such that no two meet the same vertex. We describe an efficient algorithm for finding in a given graph a matching of maximum cardinality. This problem was posed and partly solved by C. Berge; see Sections 3.7 and 3.8.

**COMBINATORICA**  
Akadémiai Kiadó – Springer-Verlag

COMBINATORICA 14 (1) (1994) 71–109

A THEORY OF ALTERNATING PATHS AND BLOSSOMS FOR  
PROVING CORRECTNESS OF THE  $O(\sqrt{VE})$  GENERAL GRAPH  
MAXIMUM MATCHING ALGORITHM

VIJAY V. VAZIRANI<sup>1</sup>

Received December 30, 1989

Revised June 15, 1993

# **k-regular bipartite graphs**

## **Dancing problem.**

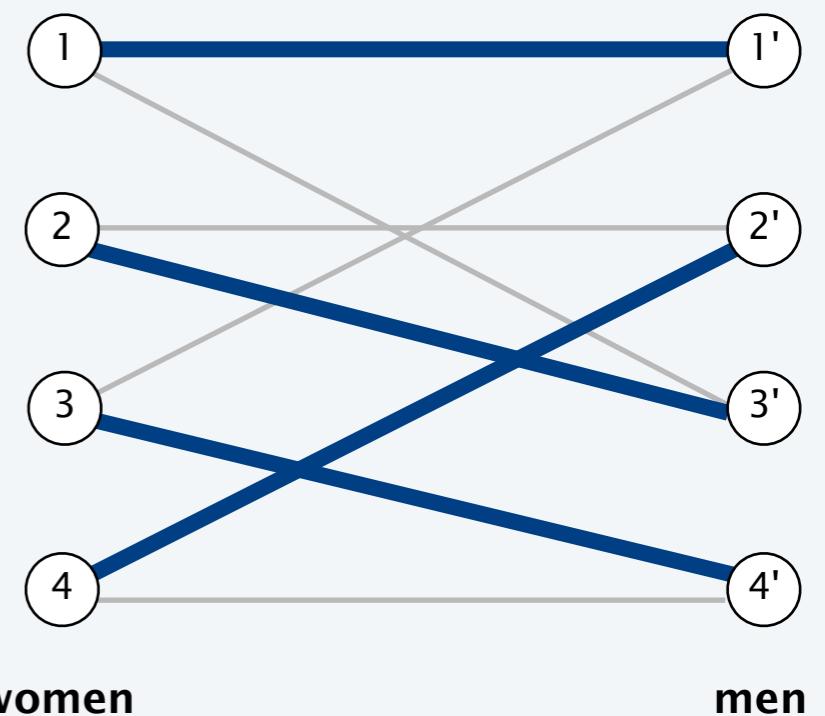
- Exclusive Ivy league party attended by  $n$  men and  $n$  women.
- Each man knows exactly  $k$  women; each woman knows exactly  $k$  men.
- Acquaintances are mutual.
- Is it possible to arrange a dance so that each woman dances with a different man that she knows?

*Perfect matching*

**Mathematical reformulation.** Does every  $k$ -regular bipartite graph have a perfect matching?

**Ex.** Boolean hypercube.

$|L|=|R|$  and left vertices  
and right vertices have same  
2-regular bipartite graph degree



# **k-regular bipartite graphs have perfect matchings**

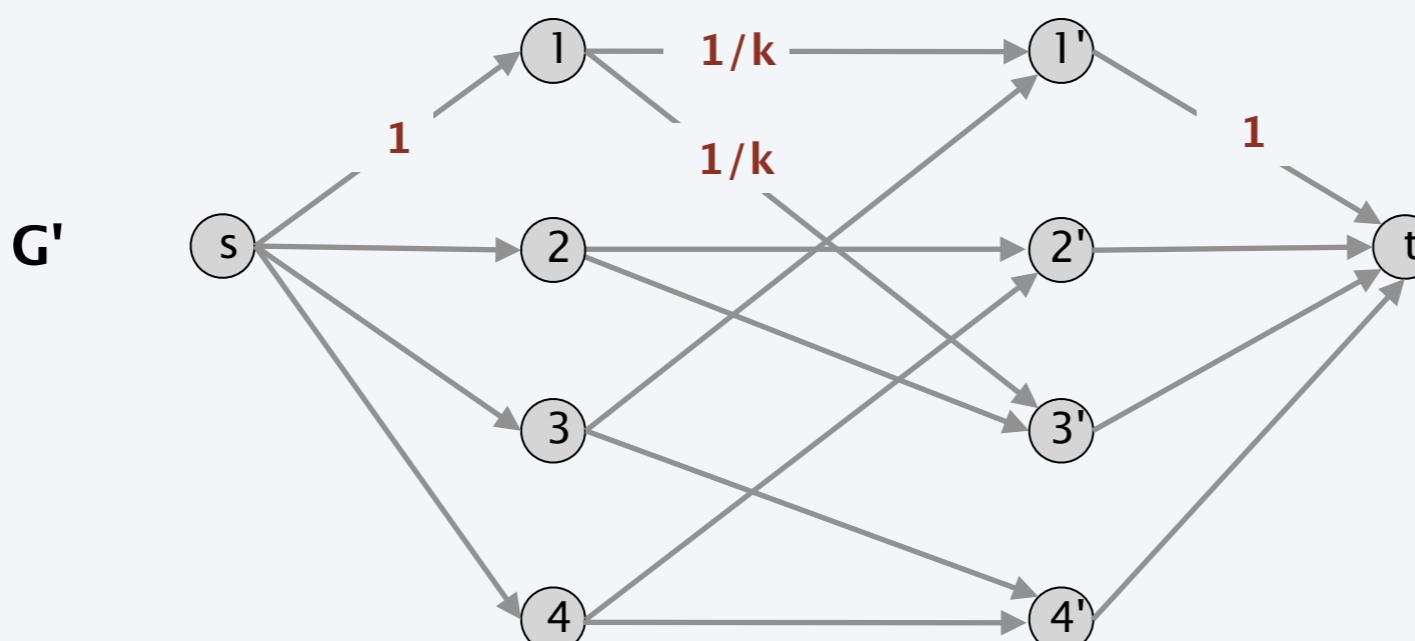
**Theorem.** Every  $k$ -regular bipartite graph  $G$  has a perfect matching.

Pf. *all*

- Size of max matching = value of max flow in  $G'$ .
- Consider flow

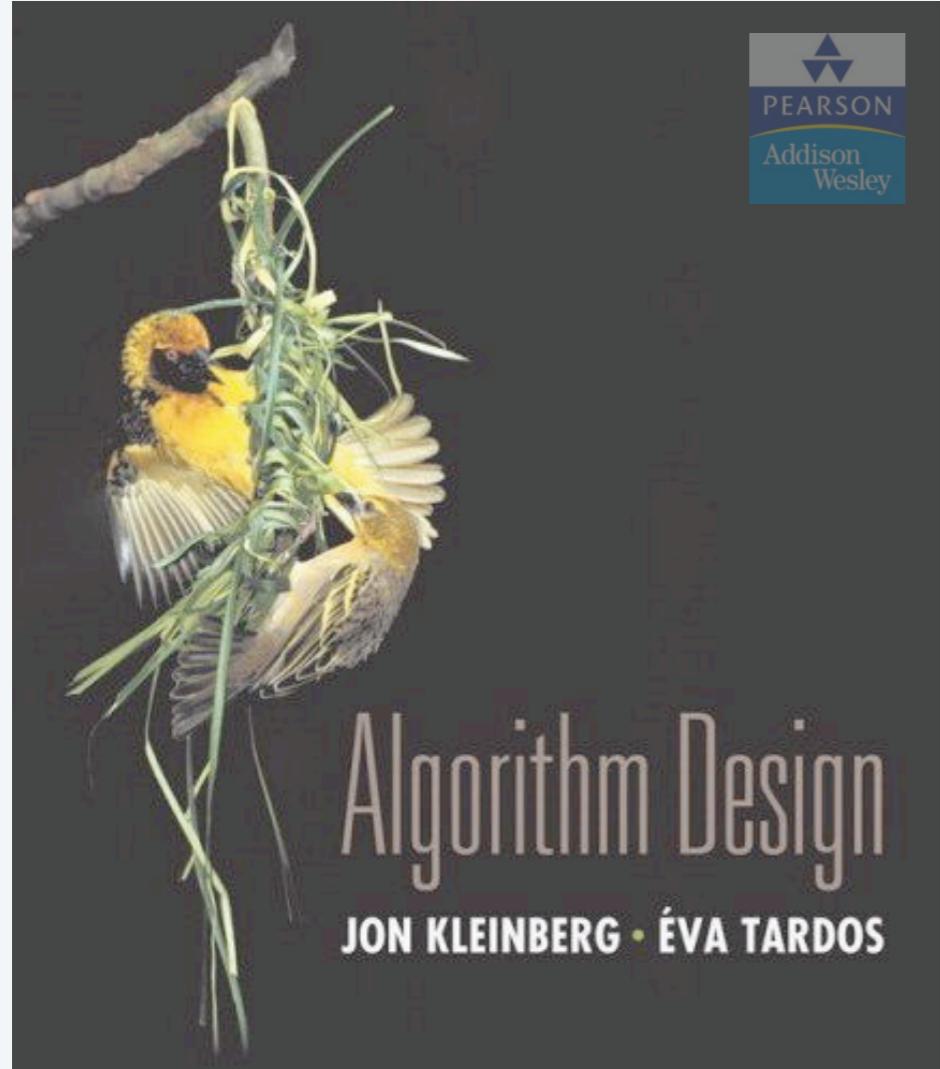
$$\text{Flow rule } \rightarrow f(u, v) = \begin{cases} 1/k & \text{if } (u, v) \in E \\ 1 & \text{if } u = s \text{ or } v = t \\ 0 & \text{otherwise} \end{cases}$$

- $f$  is a flow in  $G'$  and its value =  $n$   $\Rightarrow$  perfect matching. ■



a feasible flow  $f$  of value  $n$

incoming flow is integer, must be also outgoing flow integer, it is not important that therefore discrete value



## SECTION

# 7. NETWORK FLOW II

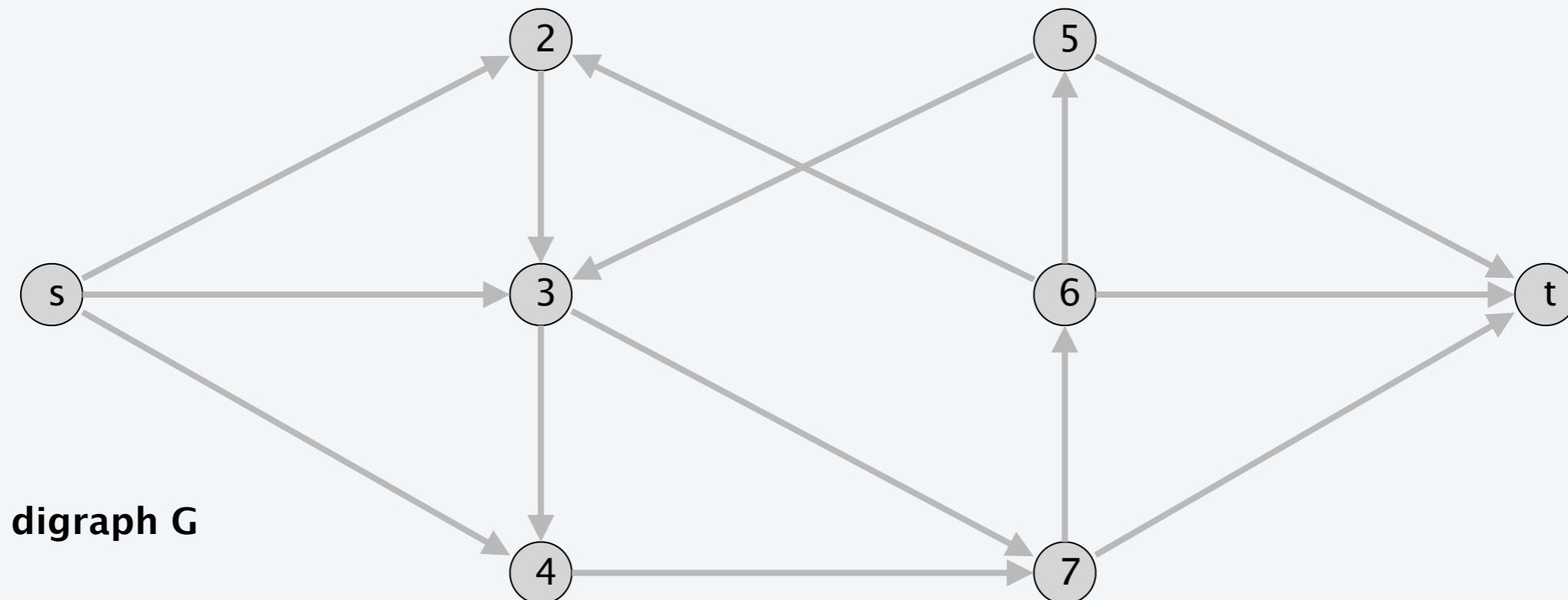
---

- ▶ *bipartite matching*
- ▶ ***disjoint paths***
- ▶ *extensions to max flow*
- ▶ *survey design*
- ▶ *airline scheduling*
- ▶ *image segmentation*
- ▶ *project selection*
- ▶ *baseball elimination*

## Edge-disjoint paths

**Def.** Two paths are **edge-disjoint** if they have no edge in common. , never share two edges

**Disjoint path problem.** Given a digraph  $G = (V, E)$  and two nodes  $s$  and  $t$ , find the max number of edge-disjoint  $s \rightarrow t$  paths.



# Edge-disjoint paths

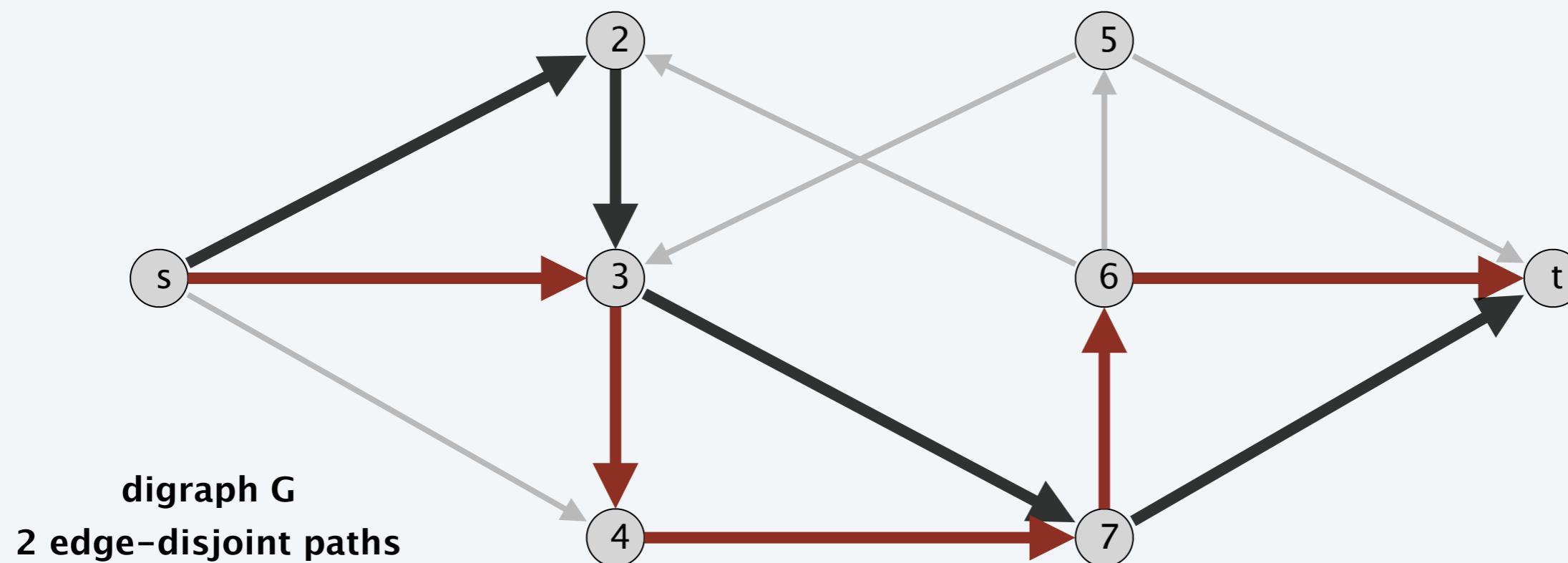
---

Def. Two paths are **edge-disjoint** if they have no edge in common.

**Disjoint path problem.** Given a digraph  $G = (V, E)$  and two nodes  $s$  and  $t$ , find the max number of edge-disjoint  $s \rightarrow t$  paths.

Ex. Communication networks.

---



# Edge-disjoint paths

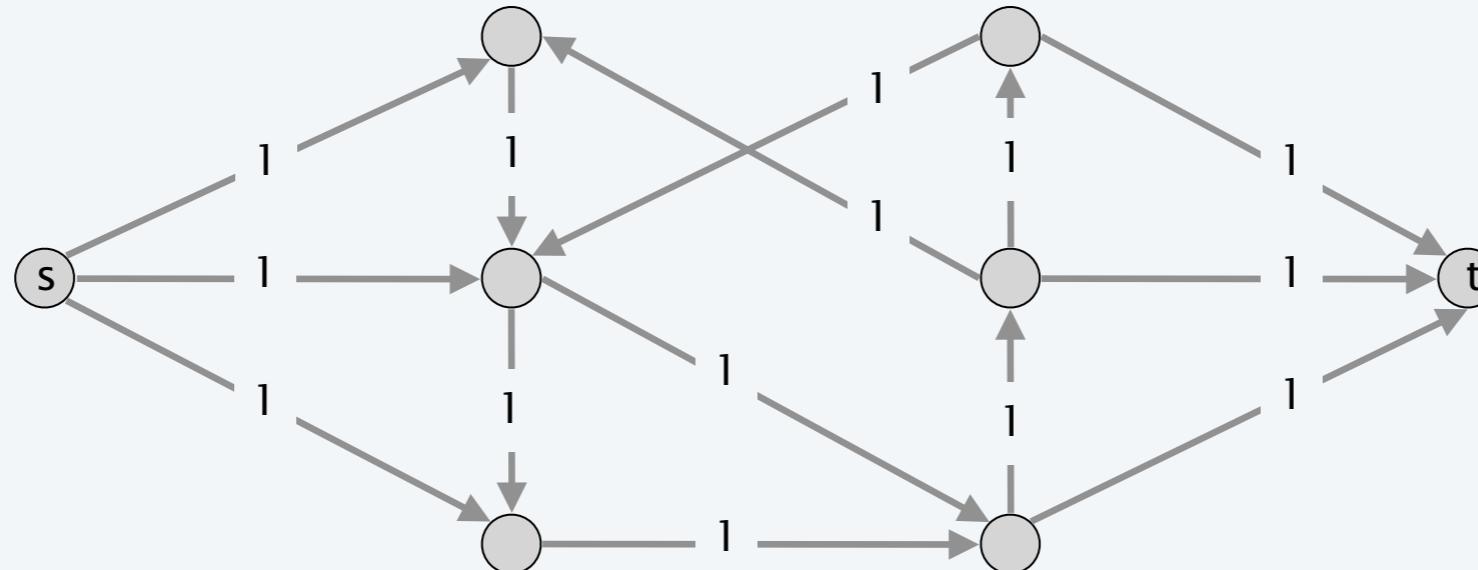
**Max flow formulation.** Assign unit capacity to every edge.

**Theorem.** Max number edge-disjoint  $s \rightarrow t$  paths equals value of max flow.

Pf. ≤

- Suppose there are  $k$  edge-disjoint  $s \rightarrow t$  paths  $P_1, \dots, P_k$ .
  - Set  $f(e) = 1$  if  $e$  participates in some path  $P_j$ ; else set  $f(e) = 0$ .
  - Since paths are edge-disjoint,  $f$  is a flow of value  $k$ . ■

↳ solve in polytime  
if only one source  
and one destination,  
otherwise NP hard.



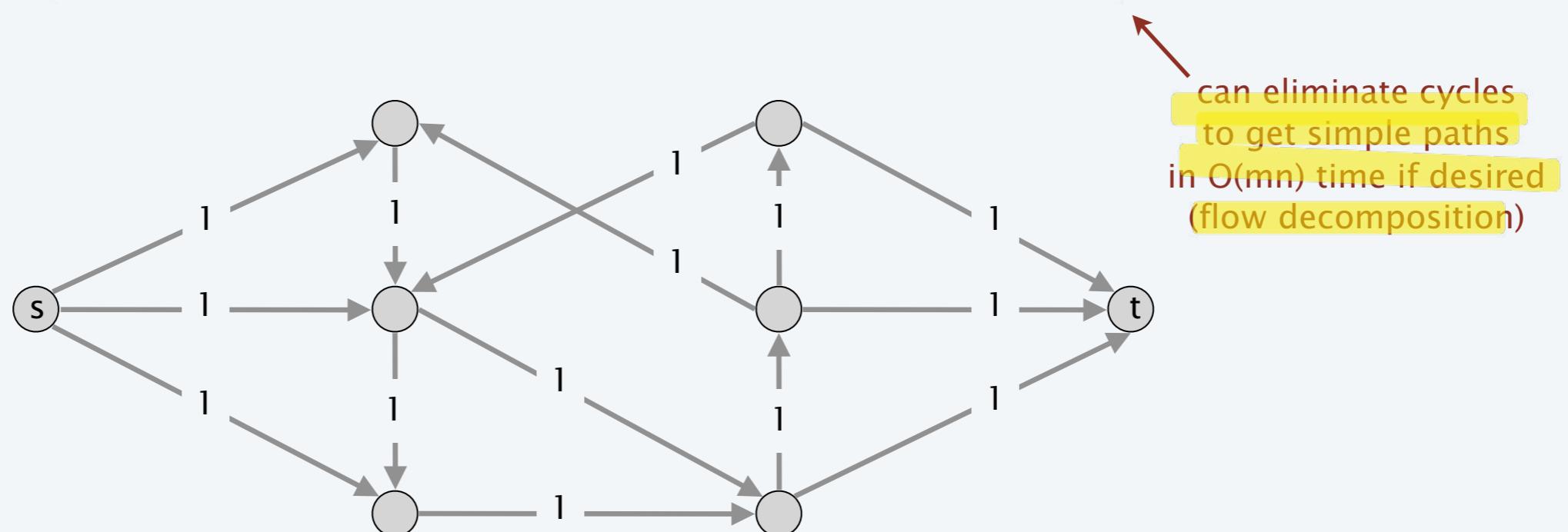
# Edge-disjoint paths

Max flow formulation. Assign unit capacity to every edge.

Theorem. Max number edge-disjoint  $s \rightarrow t$  paths equals value of max flow.

Pf.  $\geq$

- Suppose max flow value is  $k$ .
- Integrality theorem  $\Rightarrow$  there exists 0-1 flow  $f$  of value  $k$ .
- Consider edge  $(s, u)$  with  $f(s, u) = 1$ .
  - by conservation, there exists an edge  $(u, v)$  with  $f(u, v) = 1$
  - continue until reach  $t$ , always choosing a new edge
- Produces  $k$  (not necessarily simple) edge-disjoint paths. ■

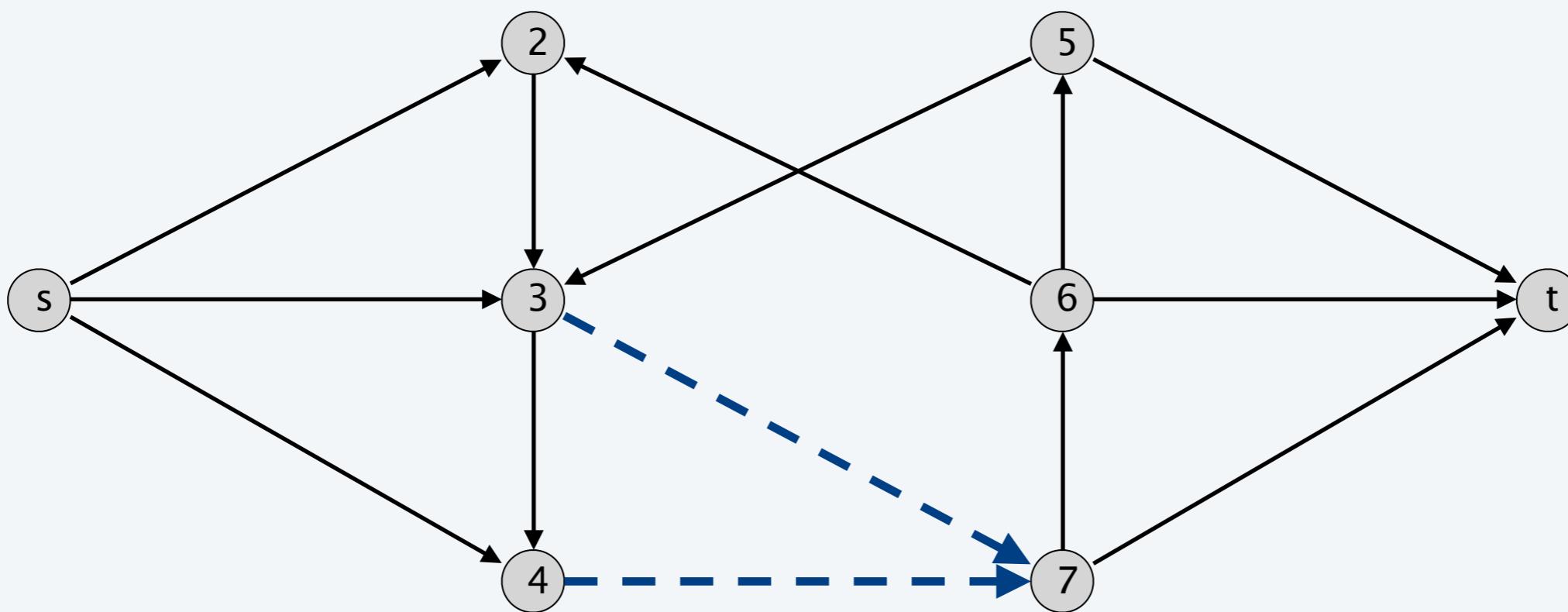


# Network connectivity

---

**Def.** A set of edges  $F \subseteq E$  **disconnects**  $t$  from  $s$  if every  $s \rightarrow t$  path uses at least one edge in  $F$ .

**Network connectivity.** Given a digraph  $G = (V, E)$  and two nodes  $s$  and  $t$ , find min number of edges whose removal disconnects  $t$  from  $s$ .

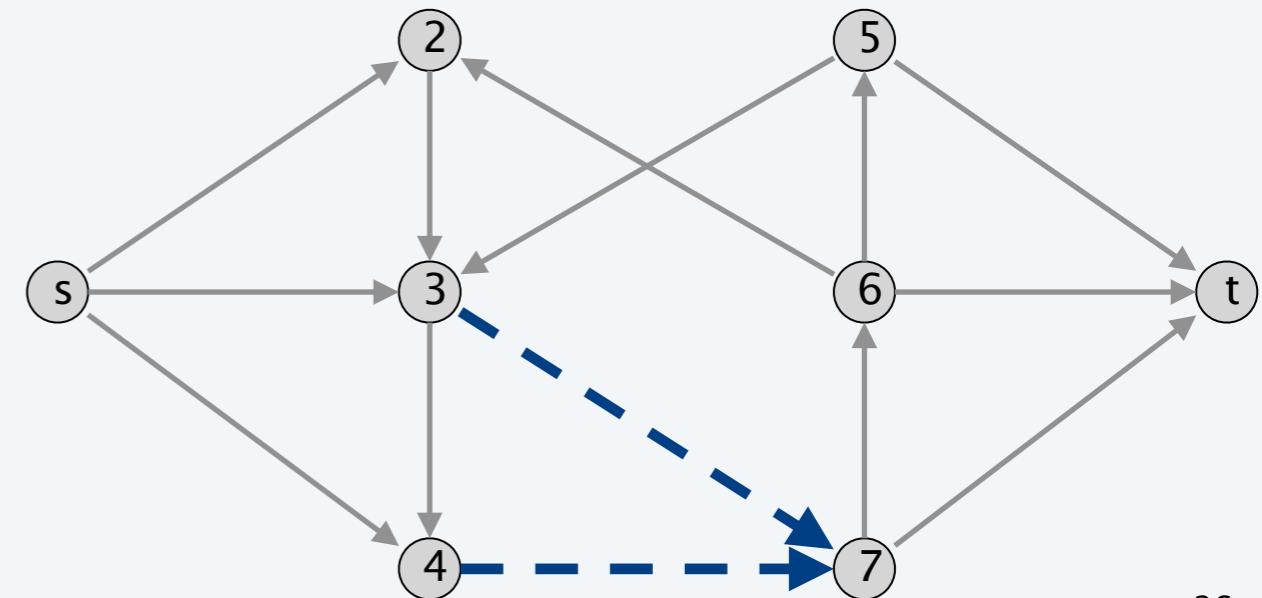
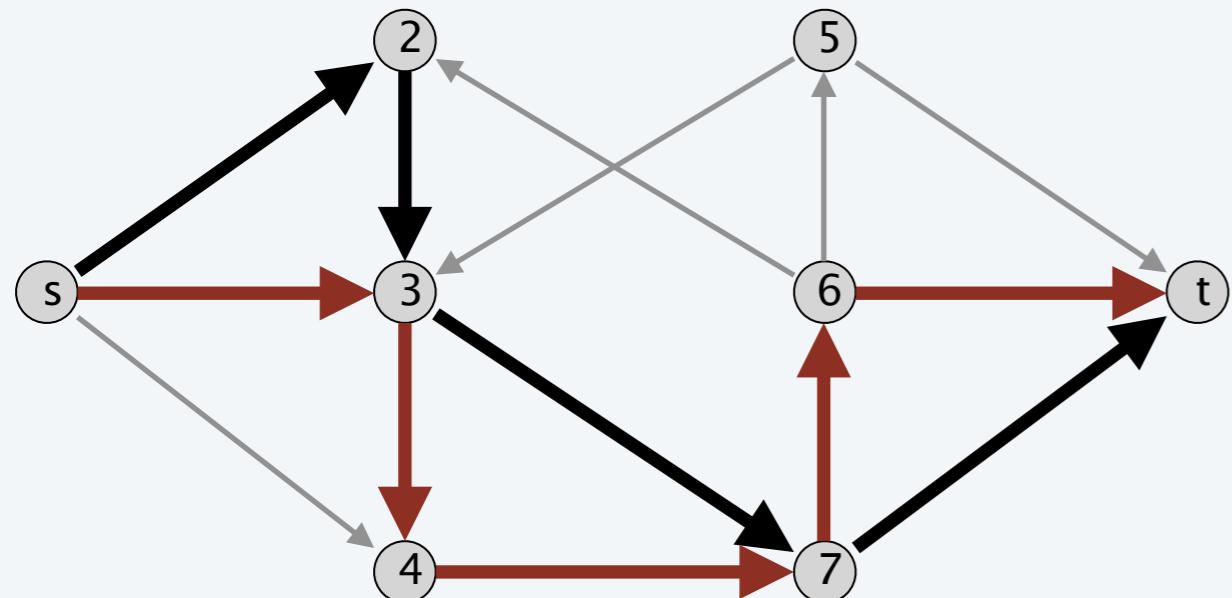


## Menger's theorem

**Theorem.** [Menger 1927] The max number of edge-disjoint  $s \rightarrow t$  paths is equal to the min number of edges whose removal disconnects  $t$  from  $s$ .

Pf.  $\leq$

- Suppose the removal of  $F \subseteq E$  disconnects  $t$  from  $s$ , and  $|F| = k$ .
- Every  $s \rightarrow t$  path uses at least one edge in  $F$ .
- Hence, the number of edge-disjoint paths is  $\leq k$ . ■

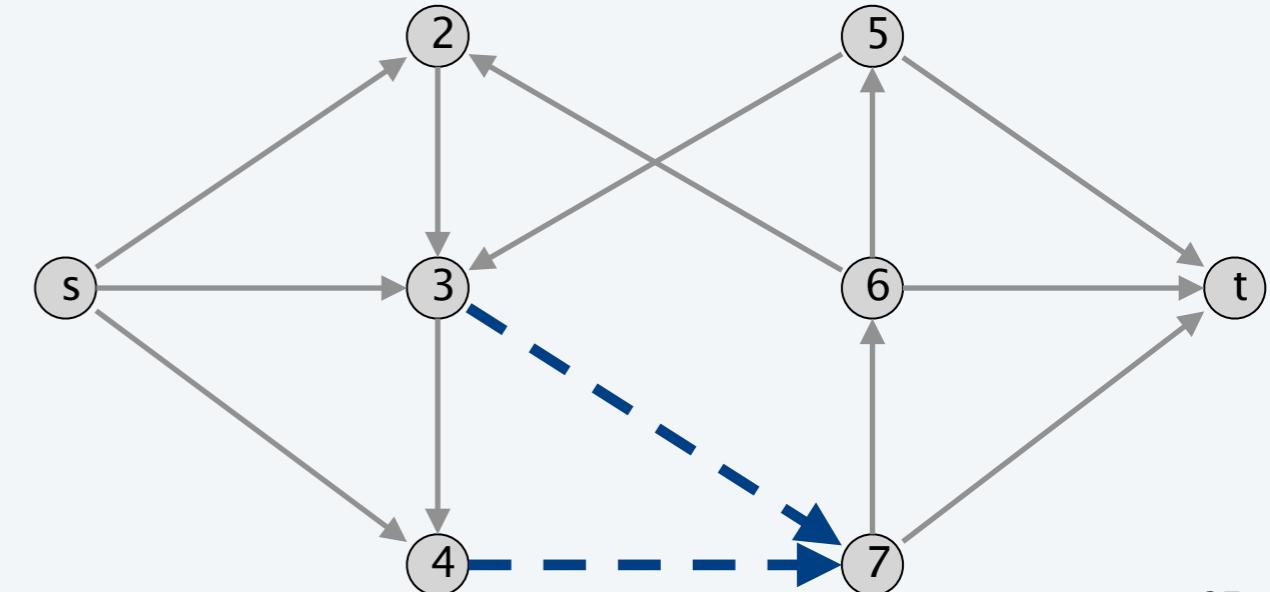
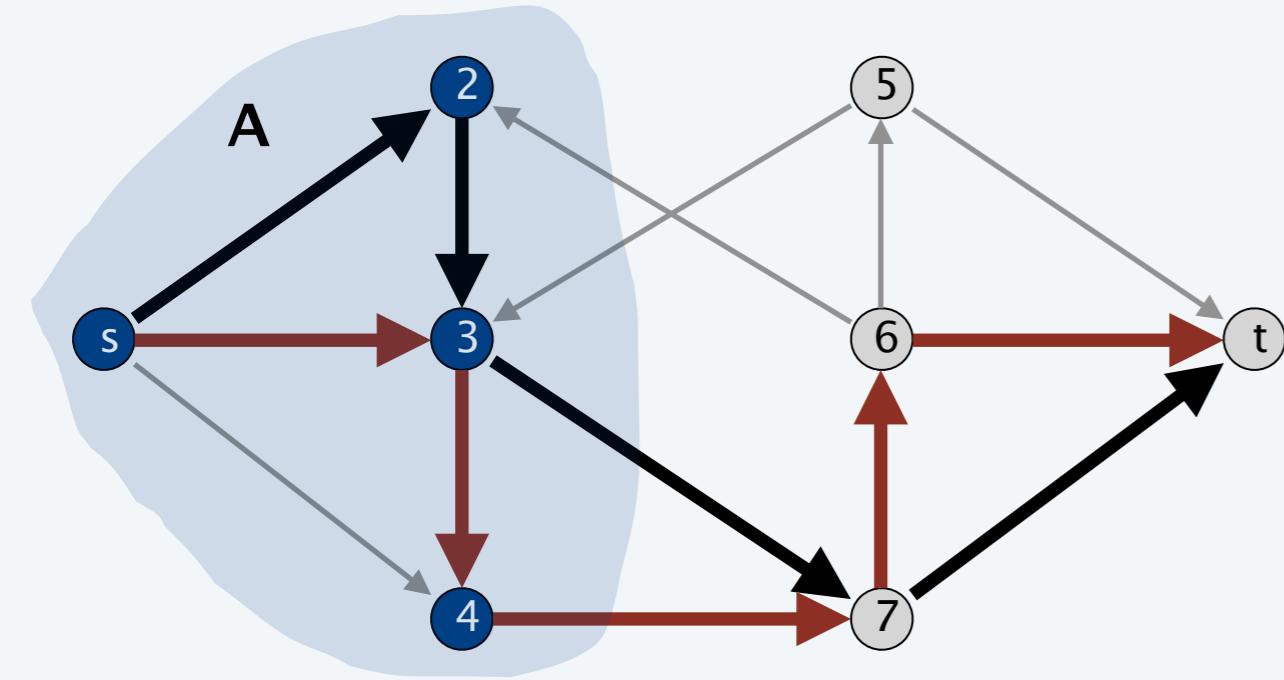


# Menger's theorem

**Theorem.** [Menger 1927] The max number of edge-disjoint  $s \rightarrow t$  paths equals the min number of edges whose removal disconnects  $t$  from  $s$ .

Pf.  $\geq$

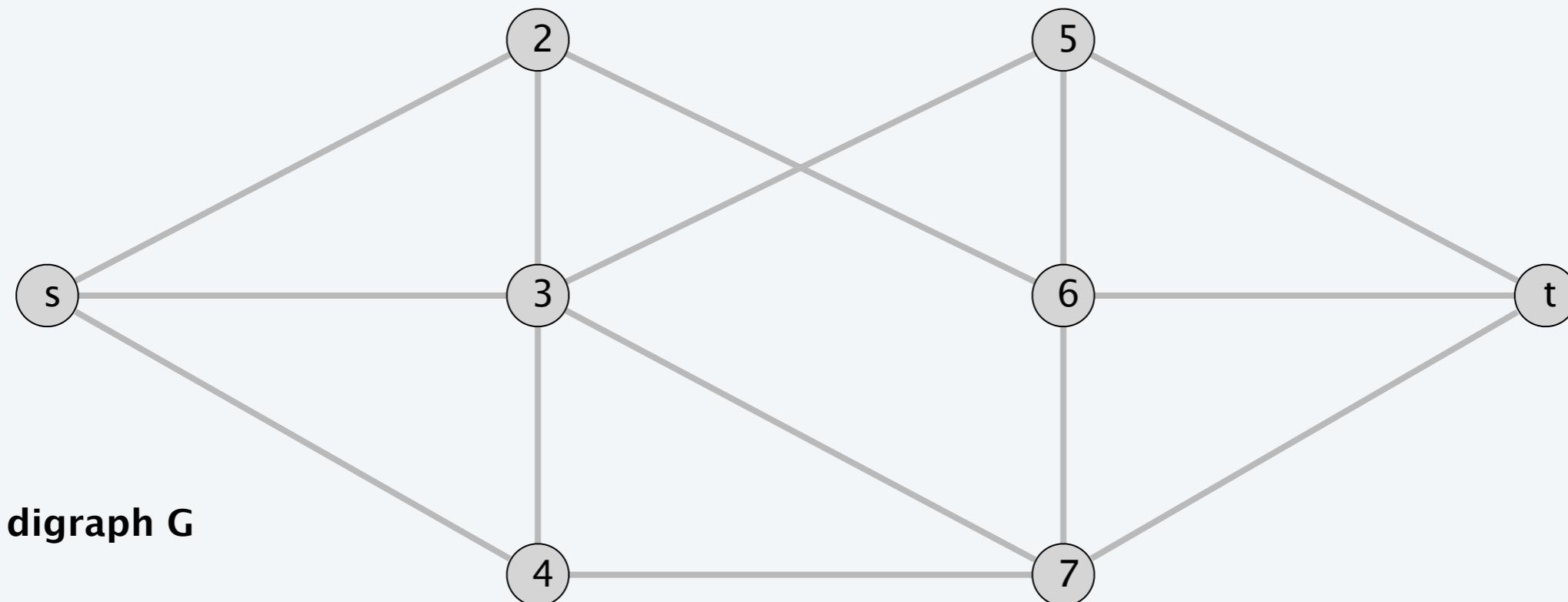
- Suppose max number of edge-disjoint paths is  $k$ .
- Then value of max flow =  $k$ .
- Max-flow min-cut theorem  $\Rightarrow$  there exists a cut  $(A, B)$  of capacity  $k$ .
- Let  $F$  be set of edges going from  $A$  to  $B$ .
- $|F| = k$  and disconnects  $t$  from  $s$ . ■



## Edge-disjoint paths in undirected graphs

**Def.** Two paths are **edge-disjoint** if they have no edge in common.

**Disjoint path problem in undirected graphs.** Given a graph  $G = (V, E)$  and two nodes  $s$  and  $t$ , find the max number of edge-disjoint  $s-t$  paths.

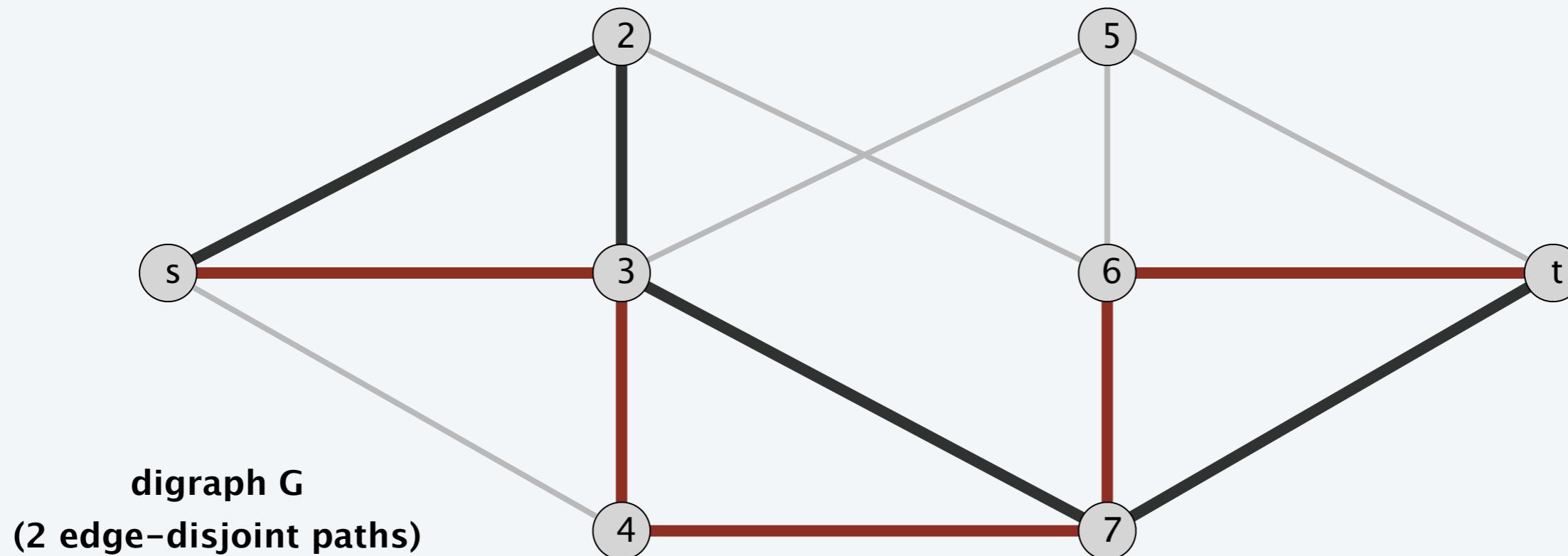


# Edge-disjoint paths in undirected graphs

---

Def. Two paths are **edge-disjoint** if they have no edge in common.

**Disjoint path problem in undirected graphs.** Given a graph  $G = (V, E)$  and two nodes  $s$  and  $t$ , find the max number of edge-disjoint  $s-t$  paths.

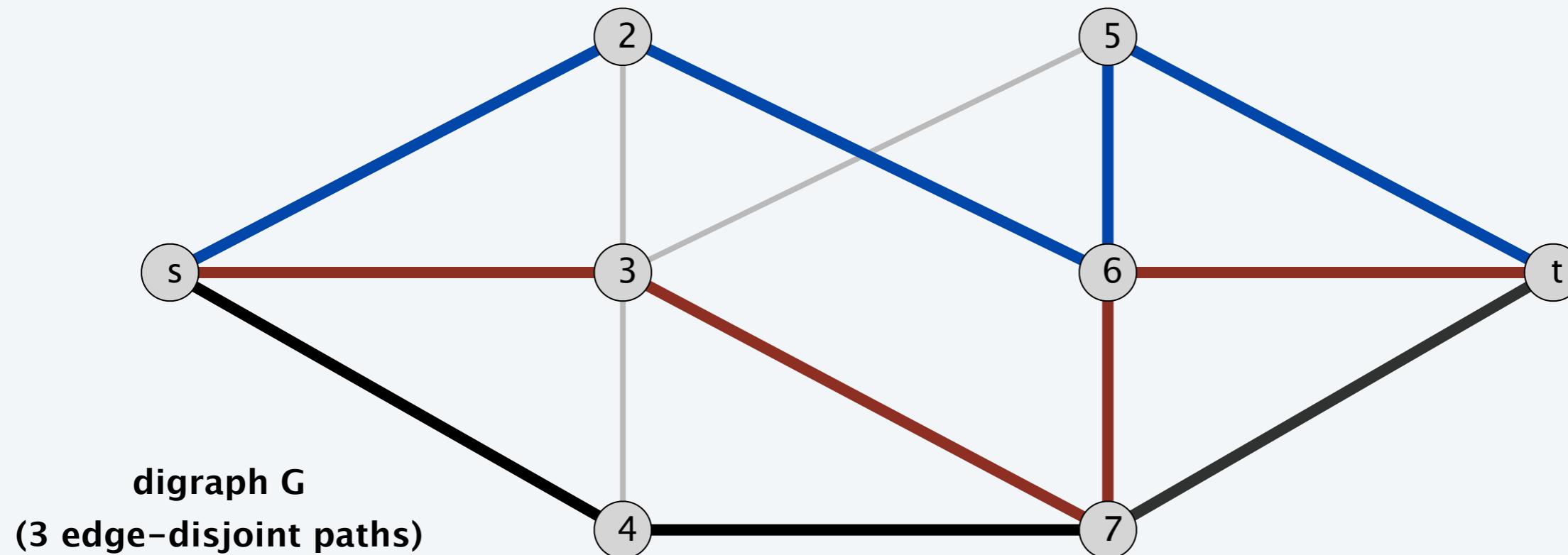


# Edge-disjoint paths in undirected graphs

---

Def. Two paths are **edge-disjoint** if they have no edge in common.

**Disjoint path problem in undirected graphs.** Given a graph  $G = (V, E)$  and two nodes  $s$  and  $t$ , find the max number of edge-disjoint  $s-t$  paths.

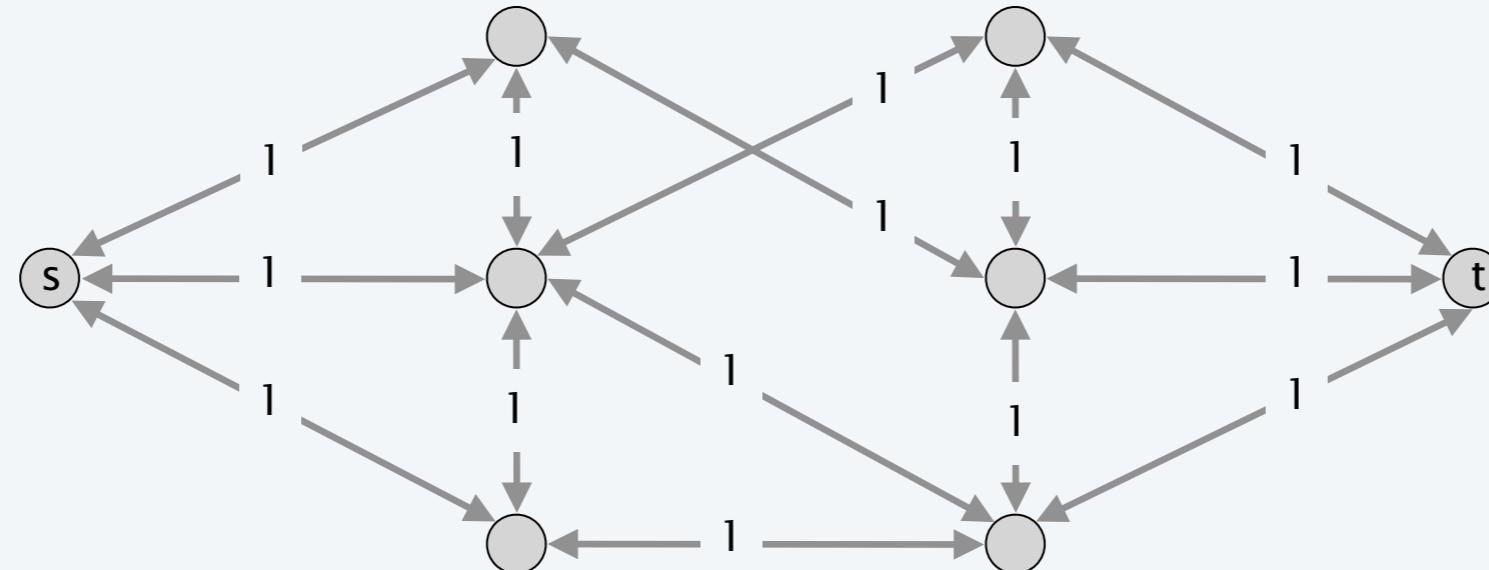


# Edge-disjoint paths in undirected graphs

**Max flow formulation.** Replace edge  $e$  with two antiparallel edges and assign unit capacity to every edge.

**Observation.** Two paths  $P_1$  and  $P_2$  may be edge-disjoint in the digraph but not edge-disjoint in the undirected graph.

if  $P_1$  uses edge  $(u, v)$   
and  $P_2$  uses its antiparallel edge  $(v, u)$



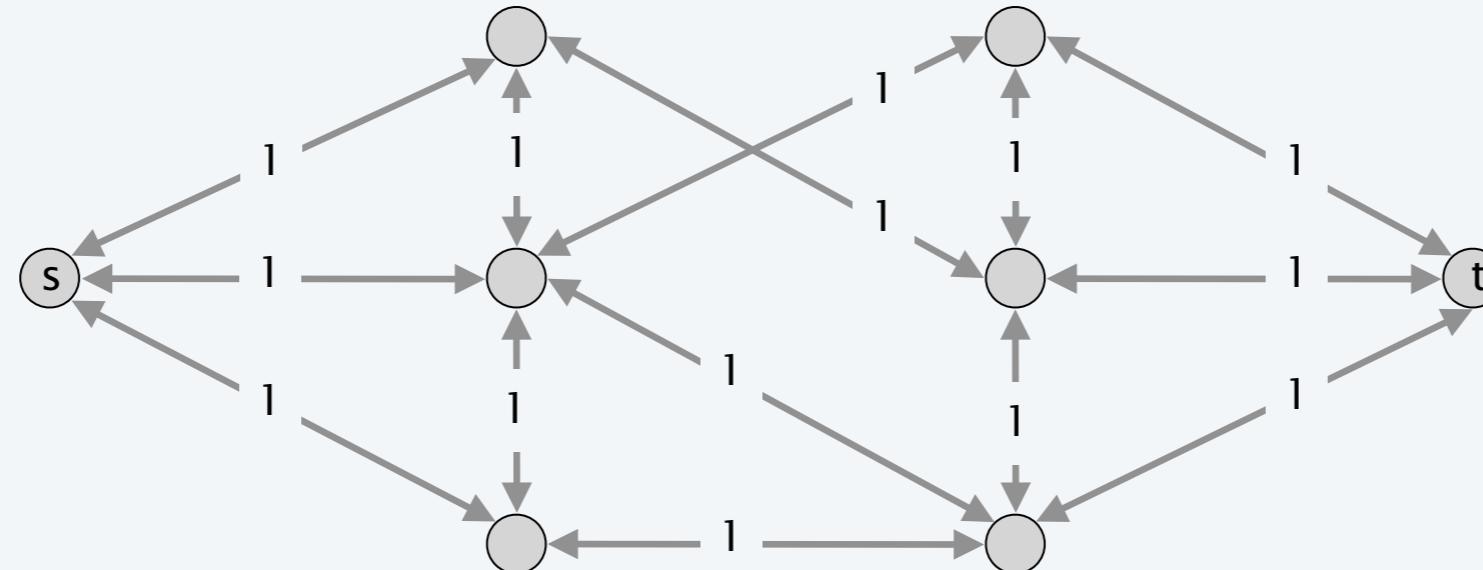
# Edge-disjoint paths in undirected graphs

**Max flow formulation.** Replace edge  $e$  with two antiparallel edges and assign unit capacity to every edge.

**Lemma.** In any flow network, there exists a maximum flow  $f$  in which for each pair of antiparallel edges  $e$  and  $e'$ , either  $f(e) = 0$  or  $f(e') = 0$  or both. Moreover, integrality theorem still holds.

**Pf.** [ by induction on number of such pairs of antiparallel edges ]

- Suppose  $f(e) > 0$  and  $f(e') > 0$  for a pair of antiparallel edges  $e$  and  $e'$ .
- Set  $f(e) = f(e) - \delta$  and  $f(e') = f(e') - \delta$ , where  $\delta = \min \{ f(e), f(e') \}$ .
- $f$  is still a flow of the same value but has one fewer such pair. ■



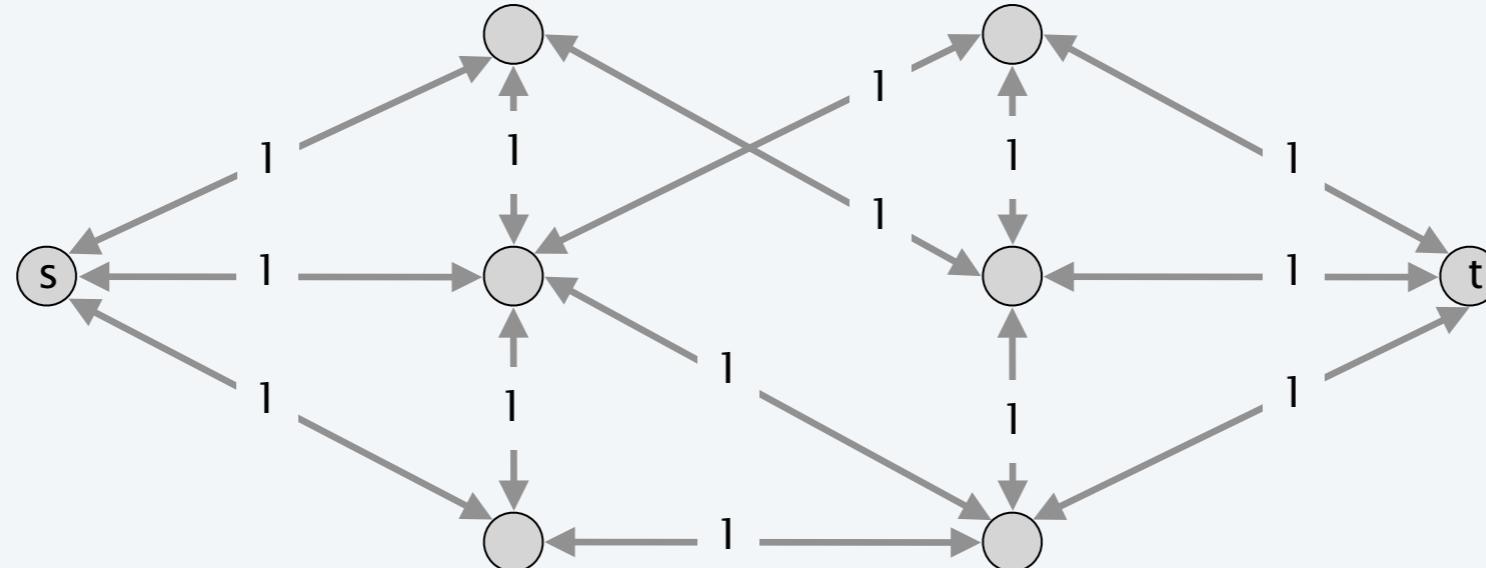
# Edge-disjoint paths in undirected graphs

**Max flow formulation.** Replace edge  $e$  with two antiparallel edges and assign unit capacity to every edge.

**Lemma.** In any flow network, there exists a maximum flow  $f$  in which for each pair of antiparallel edges  $e$  and  $e'$ , either  $f(e) = 0$  or  $f(e') = 0$  or both. Moreover, integrality theorem still holds.

**Theorem.** Max number edge-disjoint  $s \rightarrow t$  paths equals value of max flow.

**Pf.** Similar to proof in digraphs; use lemma.



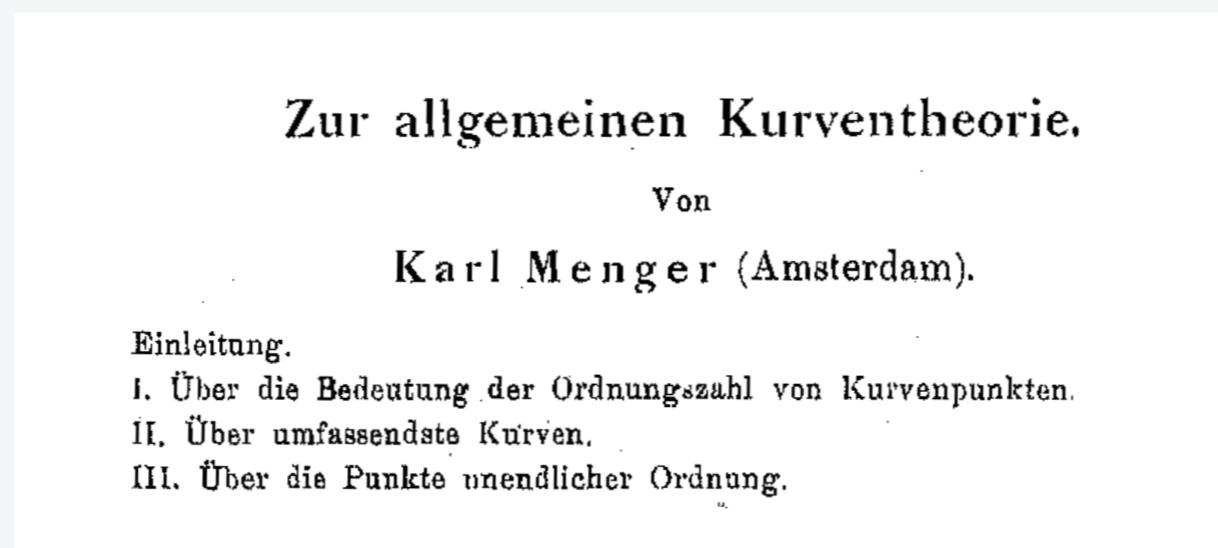
# Menger's theorems

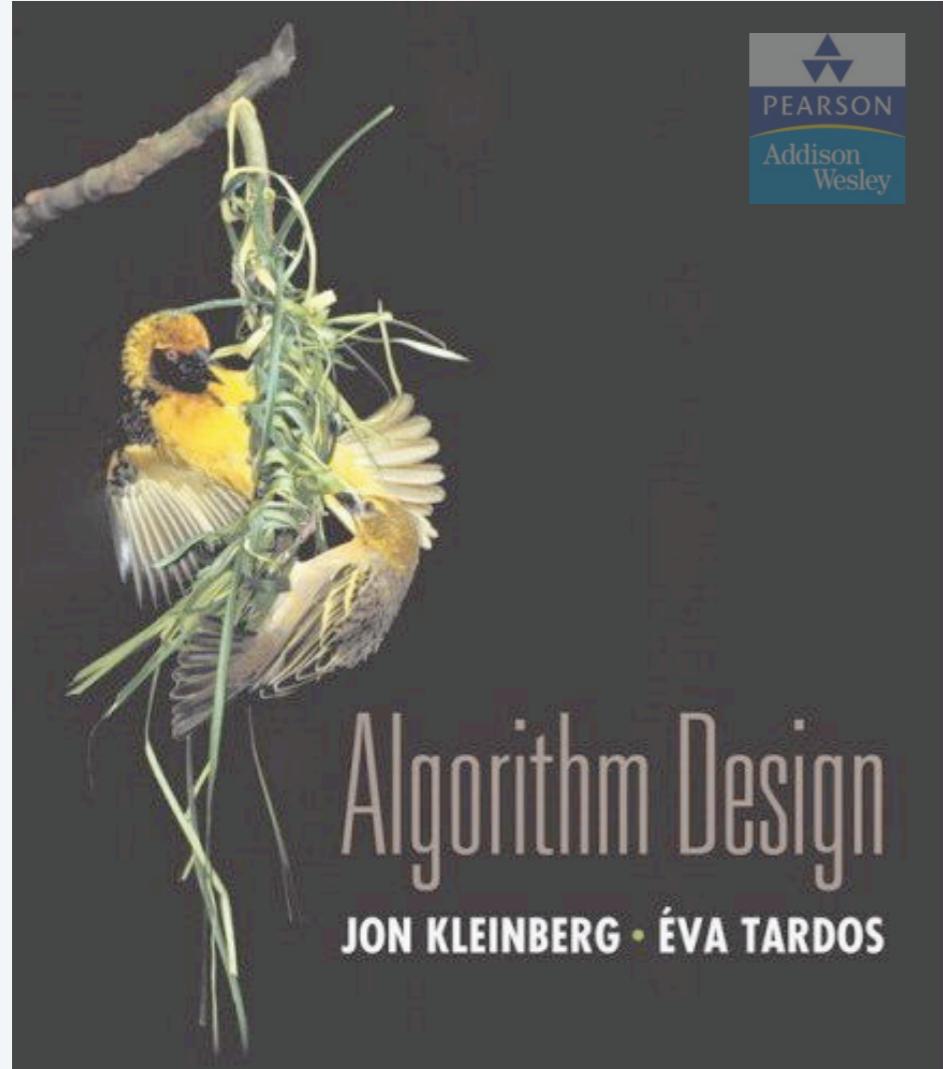
---

**Theorem.** Given an **undirected** graph with two nodes  $s$  and  $t$ , the max number of **edge-disjoint**  $s$ - $t$  paths equals the min number of edges whose removal disconnects  $s$  and  $t$ .

**Theorem.** Given a **undirected** graph with two nonadjacent nodes  $s$  and  $t$ , the max number of internally **node-disjoint**  $s$ - $t$  paths equals the min number of internal nodes whose removal disconnects  $s$  and  $t$ .

**Theorem.** Given an **directed** graph with two nonadjacent nodes  $s$  and  $t$ , the max number of internally **node-disjoint**  $s \rightarrow t$  paths equals the min number of internal nodes whose removal disconnects  $t$  from  $s$ .





**SECTION**

## 7. NETWORK FLOW II

---

- ▶ *bipartite matching*
- ▶ *disjoint paths*
- ▶ ***extensions to max flow***
- ▶ *survey design*
- ▶ *airline scheduling*
- ▶ *image segmentation*
- ▶ *project selection*
- ▶ *baseball elimination*

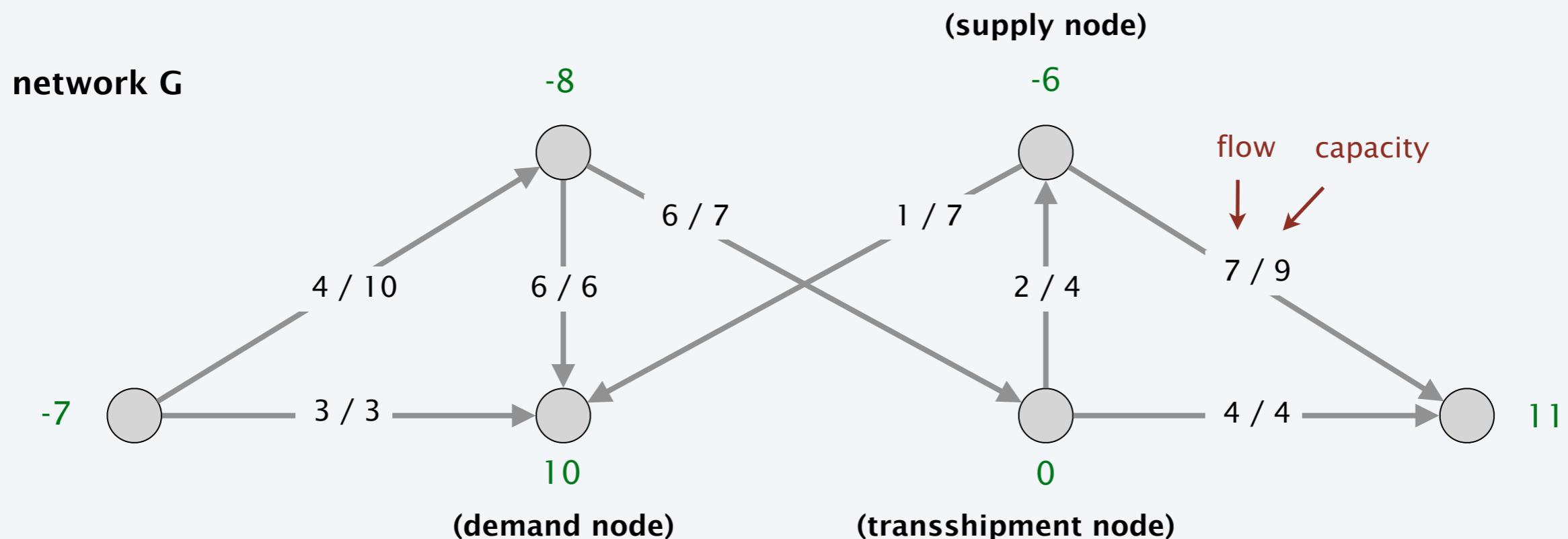
## Circulation with demands

**Def.** Given a digraph  $G = (V, E)$  with nonnegative edge capacities  $c(e)$  and node supply and demands  $d(v)$ , a **circulation** is a function that satisfies:

- For each  $e \in E$ :  $0 \leq f(e) \leq c(e)$  (capacity)
- For each  $v \in V$ :  $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$  (conservation)

↪ no source and destination

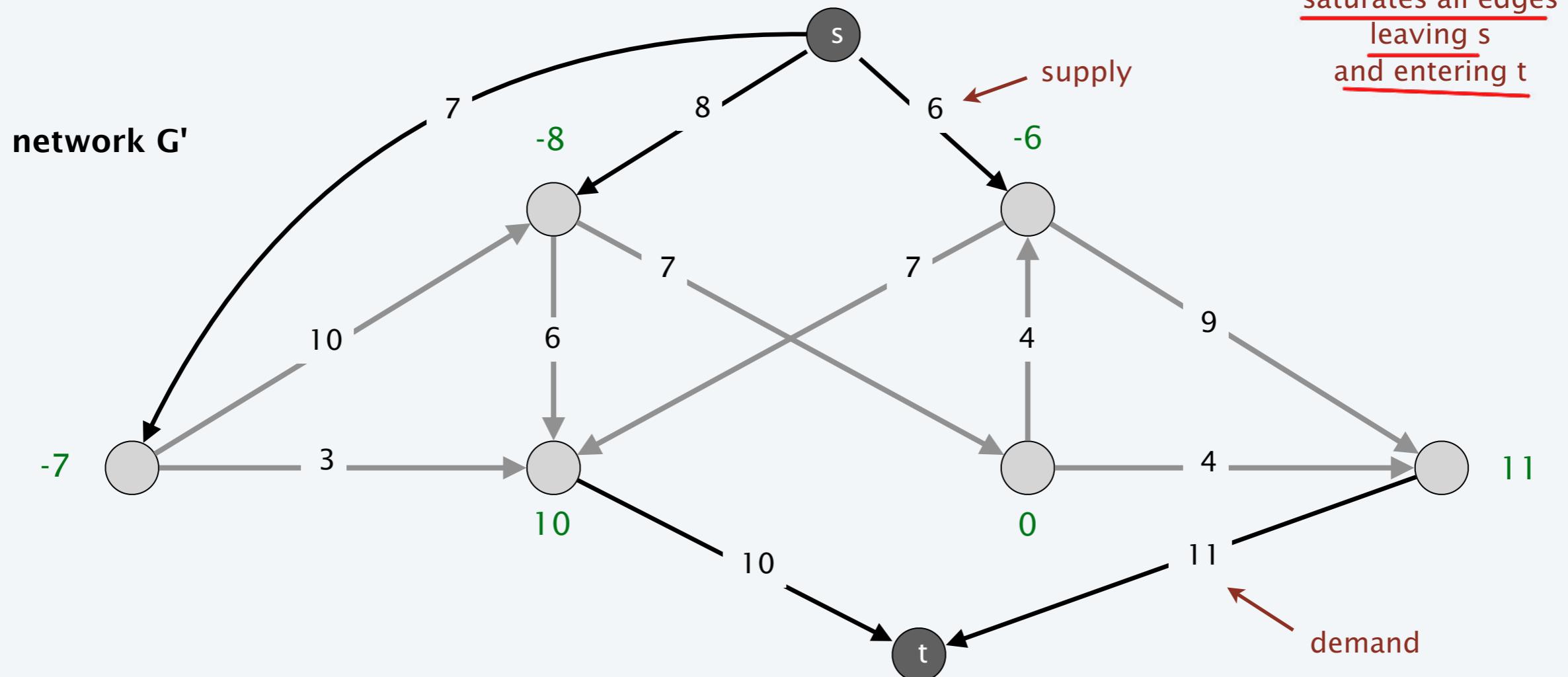
↪ demand node need some flow incoming



## Circulation with demands: max-flow formulation

- Add new source  $s$  and sink  $t$ .
- For each  $v$  with  $d(v) < 0$ , add edge  $(s, v)$  with capacity  $-d(v)$ .
- For each  $v$  with  $d(v) > 0$ , add edge  $(v, t)$  with capacity  $d(v)$ .

**Claim.**  $G$  has circulation iff  $G'$  has max flow of value  $D = \sum_{v : d(v) > 0} d(v) = \sum_{v : d(v) < 0} -d(v)$



## Circulation with demands

---

**Integrality theorem.** If all capacities and demands are integers, and there exists a circulation, then there exists one that is integer-valued.

Pf. Follows from max-flow formulation + integrality theorem for max flow.

---

**Theorem.** Given  $(V, E, c, d)$ , there does **not** exist a circulation iff there exists a node partition  $(A, B)$  such that  $\sum_{v \in B} d(v) > \text{cap}(A, B)$ .

---

Pf sketch. Look at min cut in  $G'$ .

demand by nodes in B exceeds  
supply of nodes in B plus  
max capacity of edges going from A to B



## Circulation with demands and lower bounds

---

Feasible circulation.

- Directed graph  $G = (V, E)$ .
- Edge capacities  $c(e)$  and lower bounds  $\ell(e)$  for each edge  $e \in E$ .
- Node supply and demands  $d(v)$  for each node  $v \in V$ .

Def. A **circulation** is a function that satisfies:

- For each  $e \in E$ :  $\ell(e) \leq f(e) \leq c(e)$  (capacity)
- For each  $v \in V$ :  $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$  (conservation)

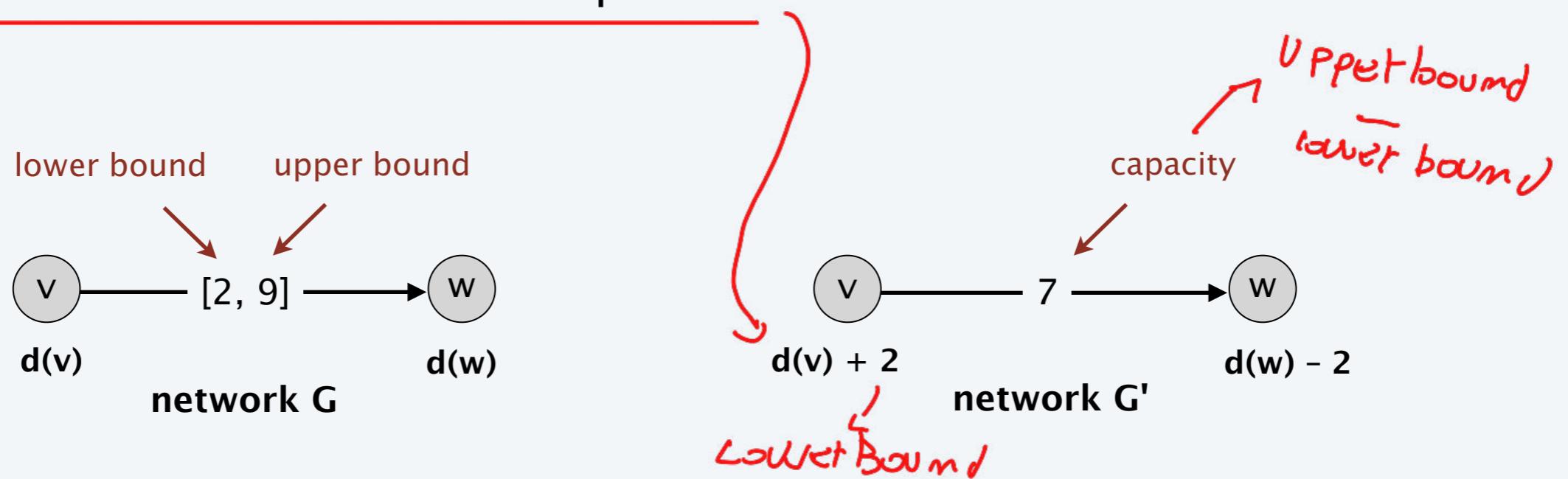
**Circulation problem with lower bounds.** Given  $(V, E, \ell, c, d)$ , does there exist a feasible circulation?

---

# Circulation with demands and lower bounds

**Max flow formulation.** Model lower bounds as circulation with demands.

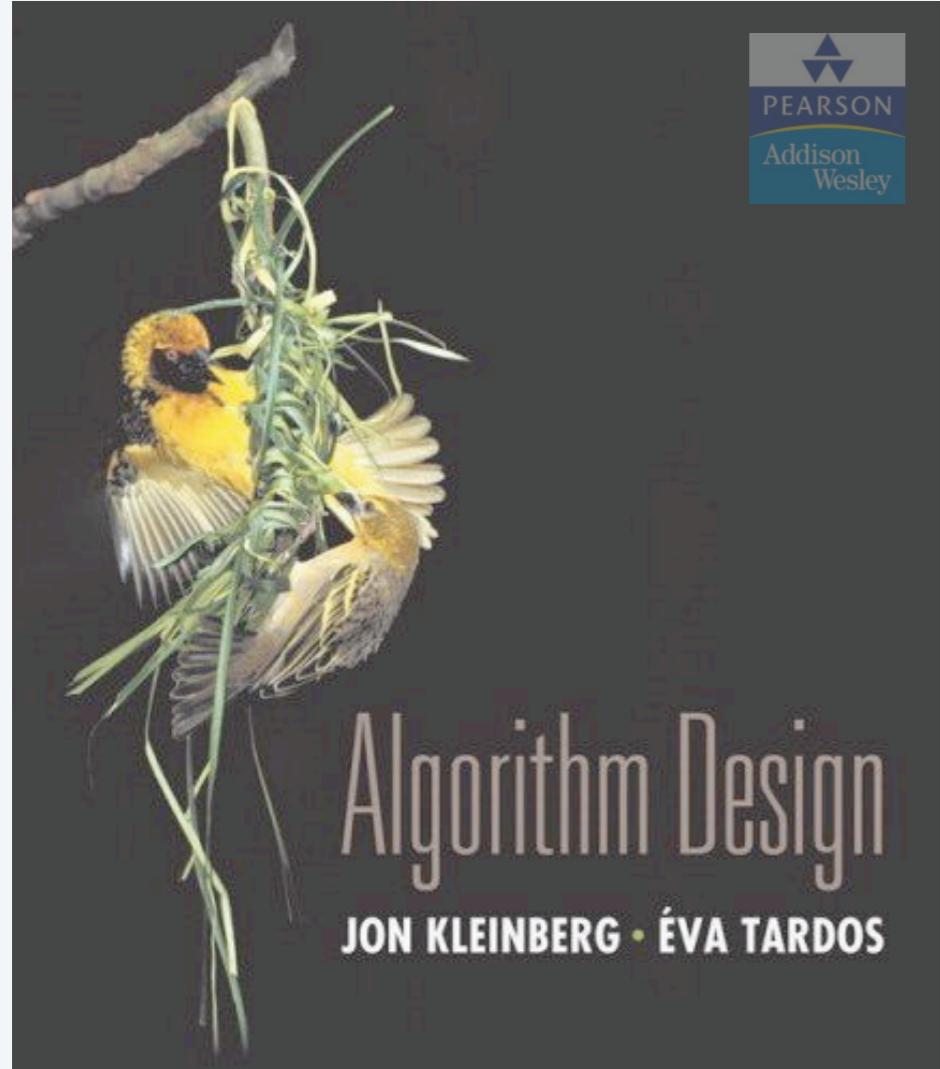
- Send  $\ell(e)$  units of flow along edge  $e$ .
- Update demands of both endpoints.



**Theorem.** There exists a circulation in  $G$  iff there exists a circulation in  $G'$ .

Moreover, if all demands, capacities, and lower bounds in  $G$  are integers,  
then there is a circulation in  $G$  that is integer-valued.

**Pf sketch.**  $f(e)$  is a circulation in  $G$  iff  $f'(e) = f(e) - \ell(e)$  is a circulation in  $G'$ .



SECTION

## 7. NETWORK FLOW II

---

- ▶ *bipartite matching*
- ▶ *disjoint paths*
- ▶ *extensions to max flow*
- ▶ ***survey design***
- ▶ *airline scheduling*
- ▶ *image segmentation*
- ▶ *project selection*
- ▶ *baseball elimination*

## Survey design

You have consumer and products, each consumer own set of products

- Design survey asking  $n_1$  consumers about  $n_2$  products. ← one survey question per product
- Can only survey consumer  $i$  about product  $j$  if they own it.
- Ask consumer  $i$  between  $c_i$  and  $c_i'$  questions.
- Ask between  $p_j$  and  $p_j'$  consumers about product  $j$ .

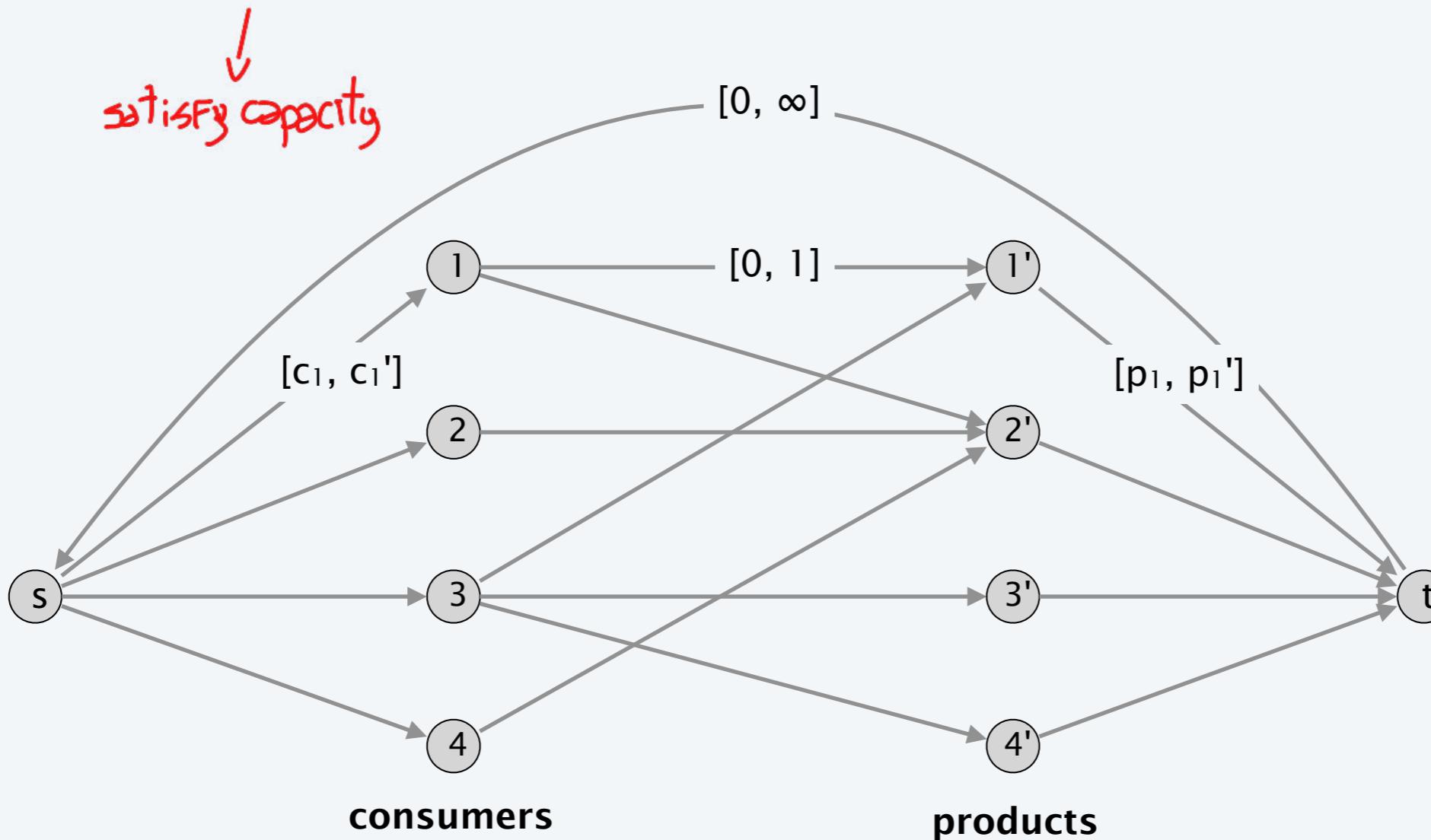
Goal. Design a survey that meets these specs, if possible.

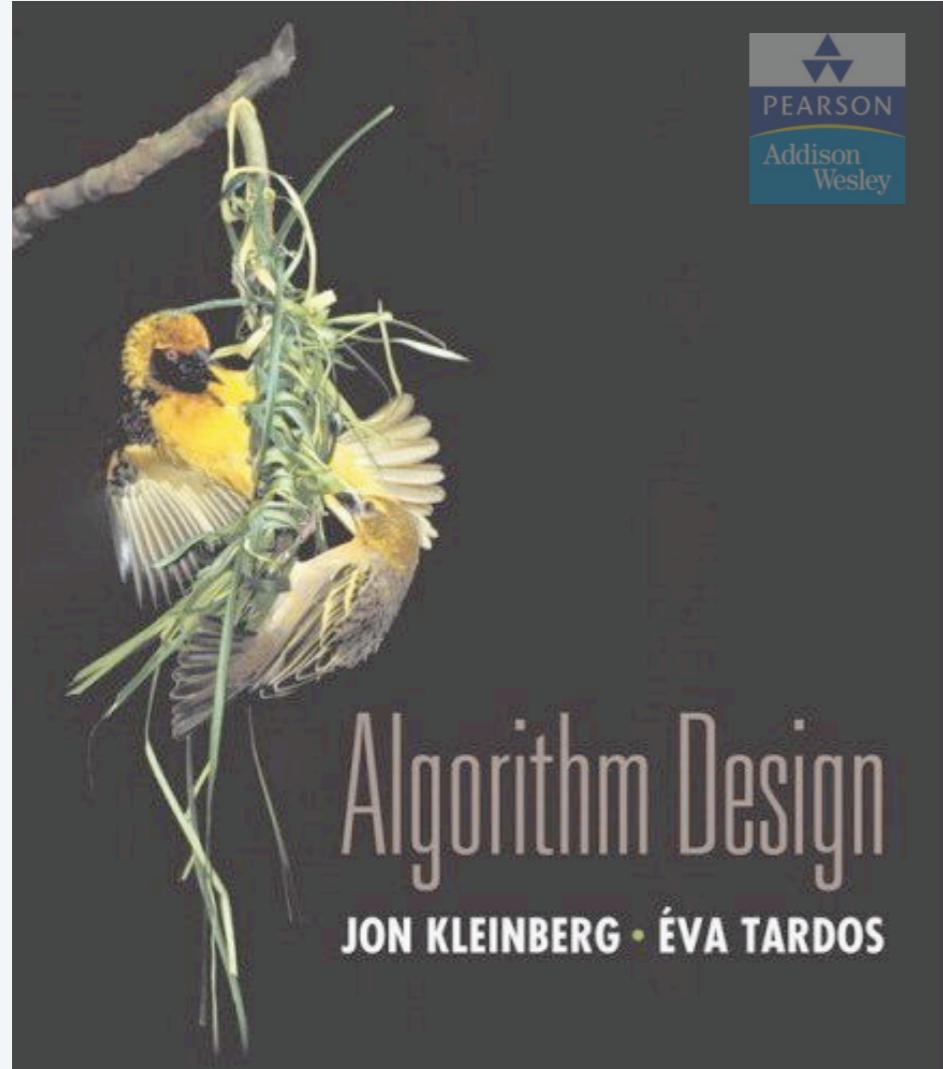
Bipartite perfect matching. Special case when  $c_i = c_i' = p_j = p_j' = 1$ .

# Survey design

**Max-flow formulation.** Model as circulation problem with lower bounds.

- Add edge  $(i, j)$  if consumer  $j$  owns product  $i$ .
  - Add edge from  $s$  to consumer  $j$ .
  - Add edge from product  $i$  to  $t$ .
  - Add edge from  $t$  to  $s$ .
- 2m up) Integer circulation  $\Leftrightarrow$  feasible survey design.





SECTION

## 7. NETWORK FLOW II

---

- ▶ *bipartite matching*
- ▶ *disjoint paths*
- ▶ *extensions to max flow*
- ▶ *survey design*
- ▶ ***airline scheduling***
- ▶ *image segmentation*
- ▶ *project selection*
- ▶ *baseball elimination*

## Airline scheduling

---

### Airline scheduling.

- Complex computational problem faced by nation's airline carriers.
- Produces schedules that are efficient in terms of:
  - equipment usage, crew allocation, customer satisfaction
  - in presence of unpredictable issues like weather, breakdowns
- One of largest consumers of high-powered algorithmic techniques.

### "Toy problem."

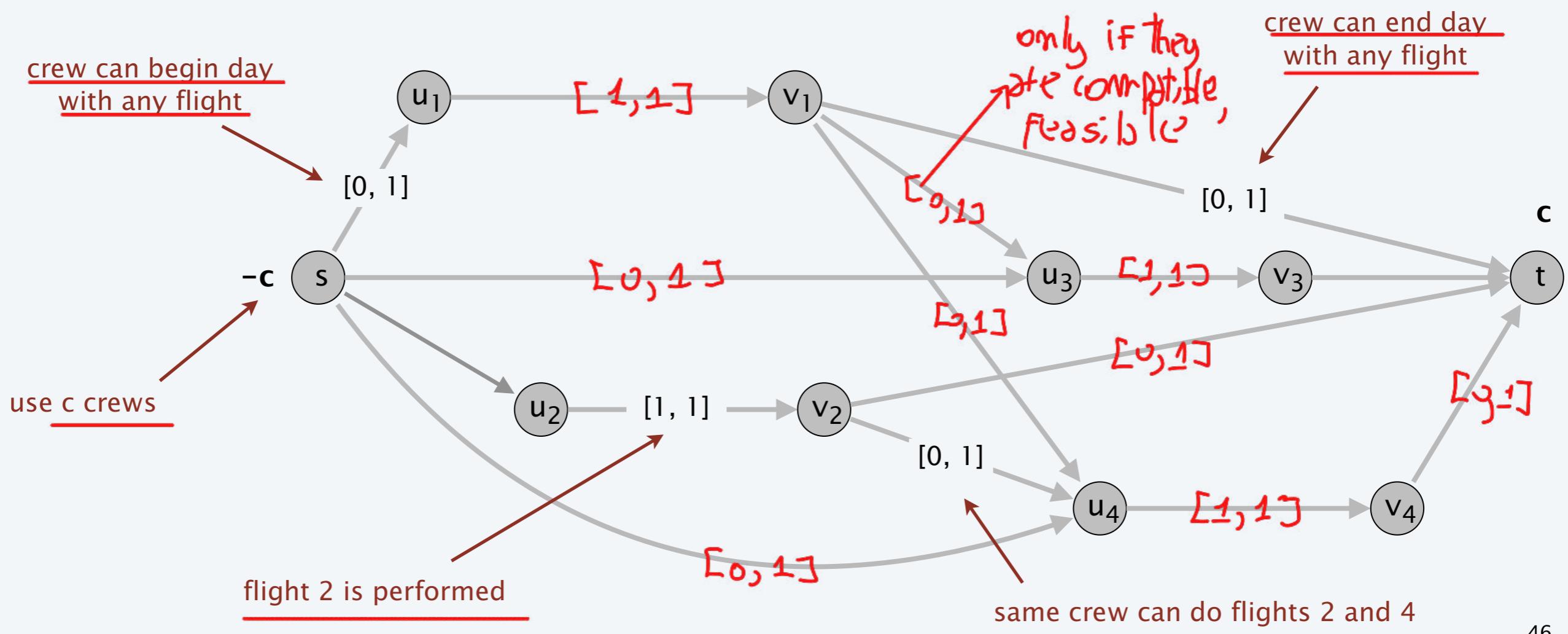
- Manage flight crews by reusing them over multiple flights.
- Input: set of  $k$  flights for a given day.  $\sim$  each have 4 parameters  
 $o_i, s_i, d_i, f_i$
- Flight  $i$  leaves origin  $o_i$  at time  $s_i$  and arrives at destination  $d_i$  destination at time  $f_i$ .
- Minimize number of flight crews.

# Airline scheduling

*We need one member for each flight*

**Circulation formulation.** [to see if  $c$  crews suffice]

- For each flight  $i$ , include two nodes  $u_i$  and  $v_i$ .
- Add source  $s$  with demand  $-c$ , and edges  $(s, u_i)$  with capacity 1.
- Add sink  $t$  with demand  $c$ , and edges  $(v_i, t)$  with capacity 1.
- For each  $i$ , add edge  $(u_i, v_i)$  with lower bound and capacity 1.
- if flight  $j$  reachable from  $i$ , add edge  $(v_i, u_j)$  with capacity 1.



## Airline scheduling: running time

---

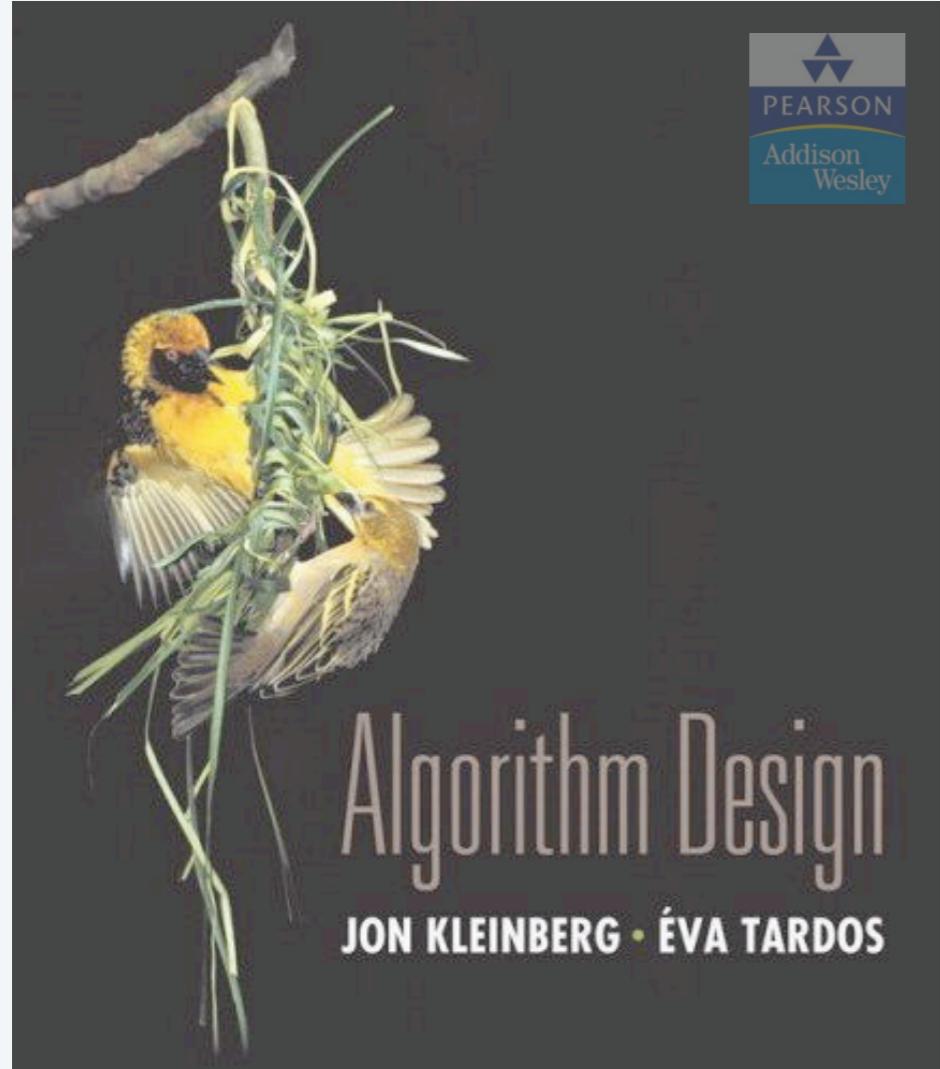
**Theorem.** The airline scheduling problem can be solved in  $O(k^3 \log k)$  time.

Pf.

- $k$  = number of flights.
- $c$  = number of crews (unknown).
- $O(k)$  nodes,  $O(k^2)$  edges.
- At most  $k$  crews needed.  
     $\Rightarrow$  solve  $\lg k$  circulation problems.  $\leftarrow$  **binary search for optimal value  $c^*$**
- Value of the flow is between 0 and  $k$ .  
     $\Rightarrow$  at most  $k$  augmentations per circulation problem.
- Overall time =  $O(k^3 \log k)$ .

**Remark.** Can solve in  $O(k^3)$  time by formulating as minimum flow problem.

---



## SECTION

# 7. NETWORK FLOW II

---

- ▶ *bipartite matching*
- ▶ *disjoint paths*
- ▶ *extensions to max flow*
- ▶ *survey design*
- ▶ *airline scheduling*
- ▶ *image segmentation*
- ▶ ***project selection***
- ▶ *baseball elimination*

## Project selection (maximum weight closure problem)

Projects with prerequisites.

if all positive revenue, you take everything  
is easy problem

can be positive  
or negative

- Set of possible projects  $P$ : project  $v$  has associated revenue  $p_v$ .
- Set of prerequisites  $E$ : if  $(v, w) \in E$ , cannot do project  $v$  unless also do project  $w$ .
- A subset of projects  $A \subseteq P$  is feasible if the prerequisite of every project in  $A$  also belongs to  $A$ .

if  $E = \emptyset$ , any prerequisites, take only positive projects

Project selection problem. Given a set of projects  $P$  and prerequisites  $E$ , choose a feasible subset of projects to maximize revenue.

MANAGEMENT SCIENCE  
Vol. 22, No. 11, July, 1976  
Printed in U.S.A.

### MAXIMAL CLOSURE OF A GRAPH AND APPLICATIONS TO COMBINATORIAL PROBLEMS\*†

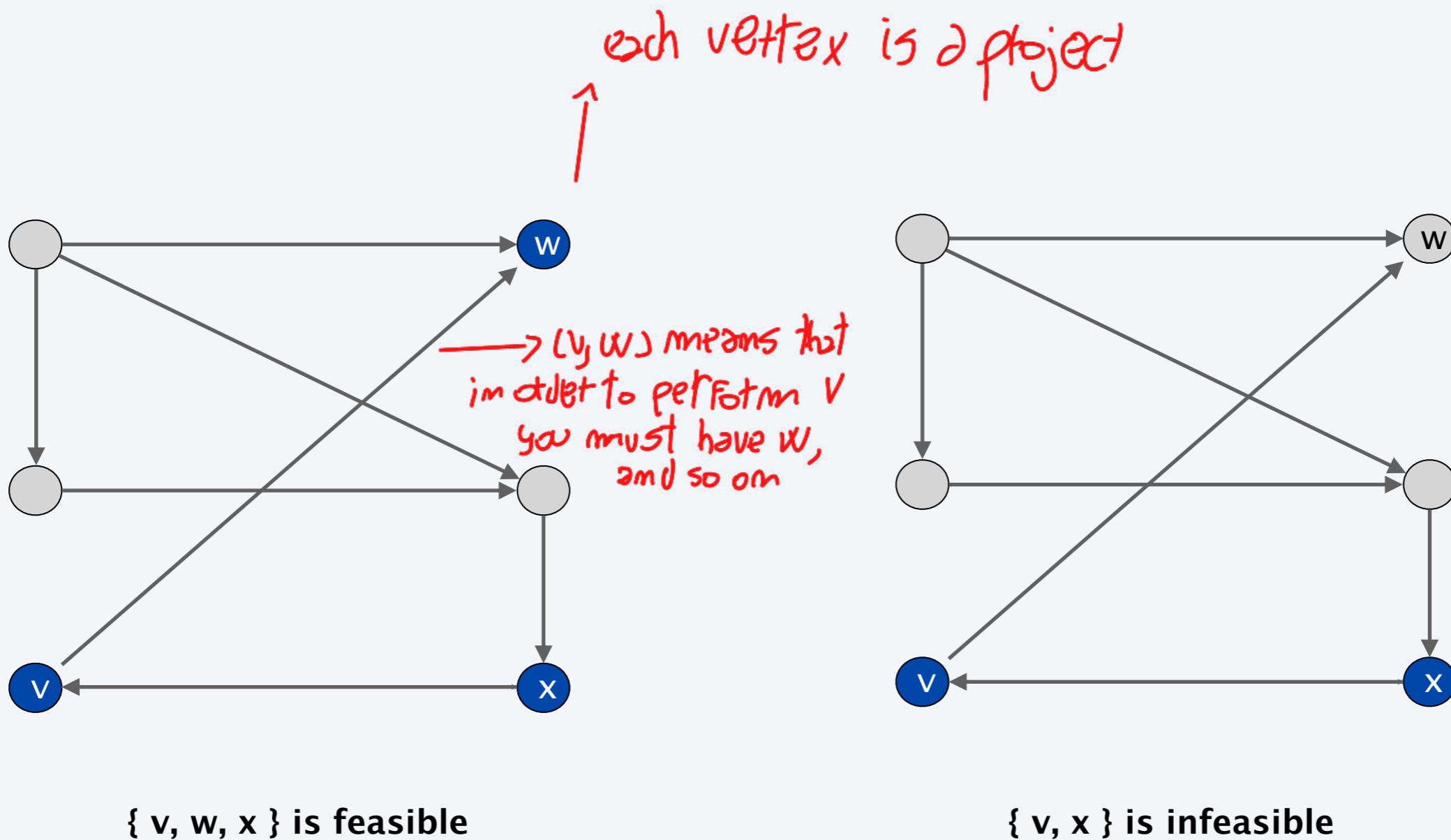
JEAN-CLAUDE PICARD

*Ecole Polytechnique, Montreal*

This paper generalizes the selection problem discussed by J. M. Rhys [12], J. D. Murchland [9], M. L. Balinski [1] and P. Hansen [4]. Given a directed graph  $G$ , a closure of  $G$  is defined as a subset of nodes such that if a node belongs to the closure all its successors also belong to the set. If a real number is associated to each node of  $G$  a maximal closure is defined as a closure of maximal value.

## Project selection: **prerequisite graph**

**Prerequisite graph.** Add edge  $(v, w)$  if can't do  $v$  without also doing  $w$ .

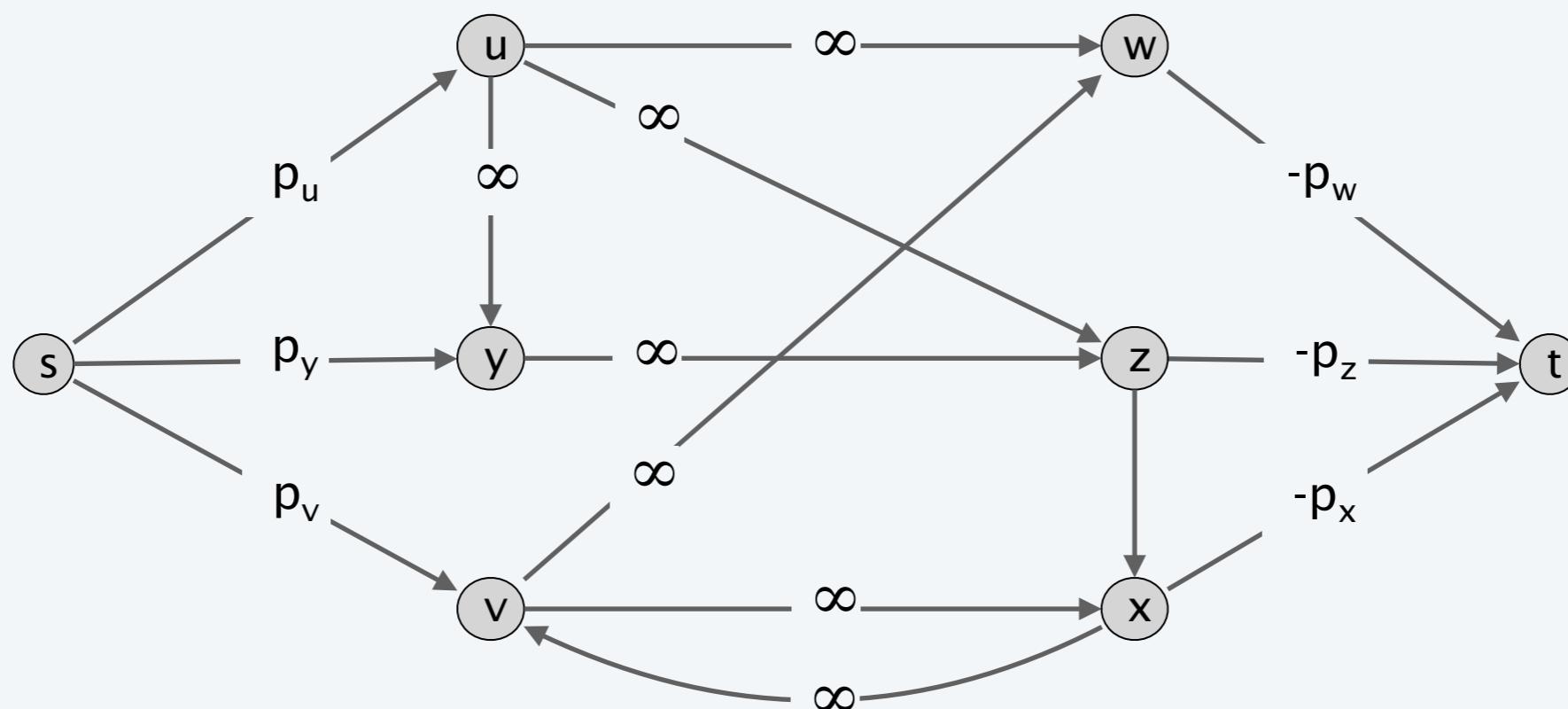


↳ if only incoming edges in blue set, no outgoing edges from blue

# Project selection: min-cut formulation

## Min-cut formulation.

- Assign capacity  $\infty$  to all prerequisite edge.
- Add edge  $(s, v)$  with capacity  $p_v$  if  $p_v > 0$ .
- Add edge  $(v, t)$  with capacity  $-p_v$  if  $p_v < 0$ .
- For notational convenience, define  $p_s = p_t = 0$ .



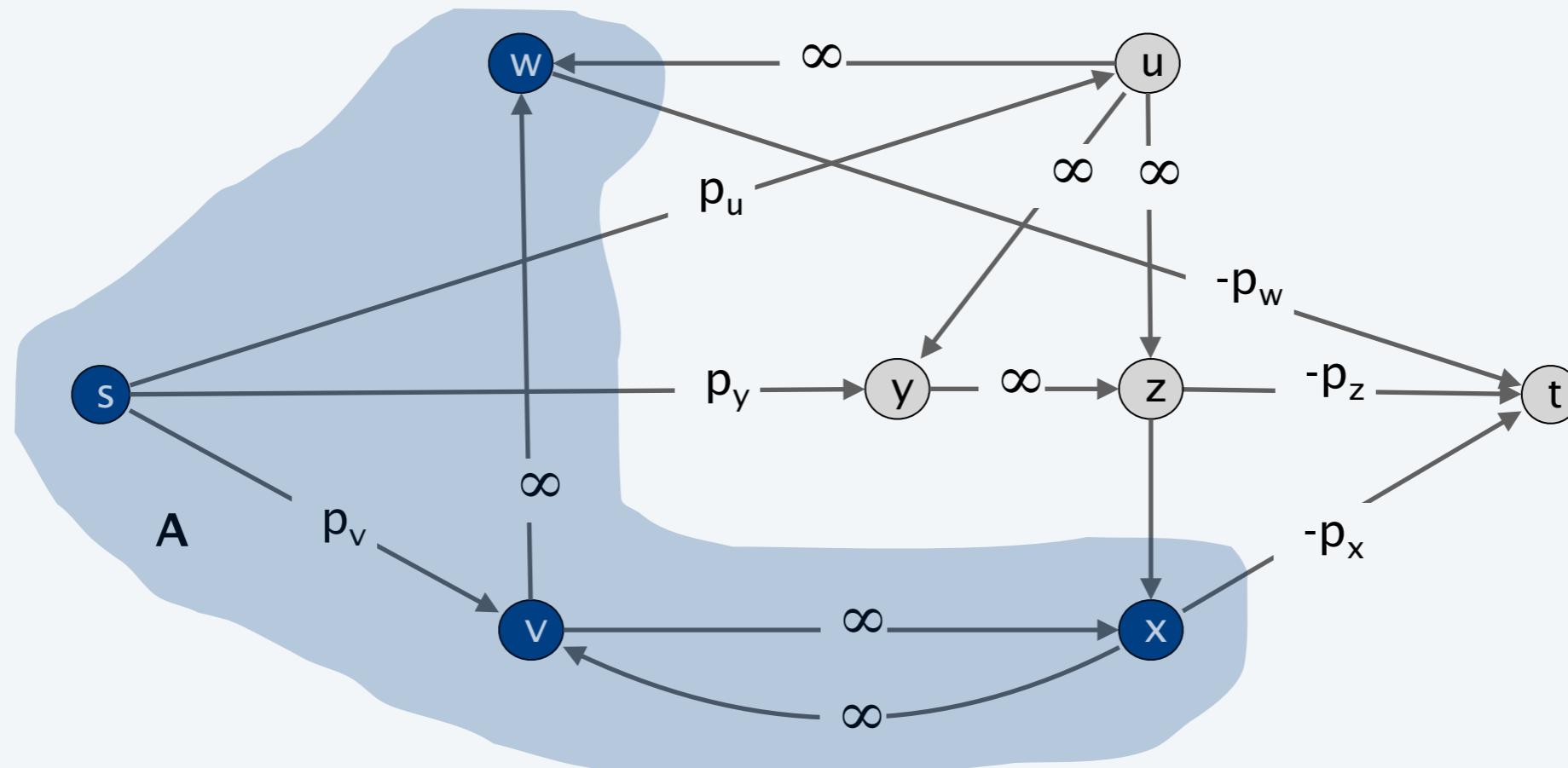
# Project selection: min-cut formulation

**Claim.**  $(A, B)$  is min cut iff  $A - \{s\}$  is optimal set of projects.

- Infinite capacity edges ensure  $A - \{s\}$  is feasible.

Max revenue because: 
$$\begin{aligned} cap(A, B) &= \sum_{v \in B: p_v > 0} p_v + \sum_{v \in A: p_v < 0} (-p_v) \\ &= \underbrace{\sum_{v: p_v > 0} p_v}_{\text{constant}} - \sum_{v \in A} p_v \end{aligned}$$

$\sum_{v \in A: p_v > 0} p_v - \sum_{v \in A: p_v < 0} + \sum_{v \in A: p_v = 0} 0$



2018/2020

**Exercise 3.** Federico now deals in exquisite chocolates, which he manufactures at home. He has a complicated distribution network set up in the university to sell his merchandise, but every one of his friends  $f \in F$  is only willing to handle a fixed amount  $n_f$  every week. He knows for every pair  $(f_1, f_2) \in F$  if these two people see each other regularly, and of course, he also knows if he himself will meet them in person or not. Finally, some of his friends are able to sell a fixed amount  $s_f \leq n_f$  to customers every week, but can only hand the rest of it over to other people.

- Model the problem as a flow problem in a graph  $G$ , only consisting of regular vertices, one source, one sink, and capacitated edges to find out how many chocolates Federico should actually make every week. Give a formal definition of your network, and also draw a small example, e.g. for  $|F| = 4$ .
- In the new semester, each friend in Federico's network will be attending courses in a certain building, according to their respective field of study, i.e. every  $f \in F$  will have an associated building  $b_f \in B$ , where  $B$  are all the university buildings. Federico is worried this will affect his sales, since in every building  $b \in B$ , there will be only a certain regular number of customers available, limiting the weekly amount of chocolate sold there to  $c_b$ . Adjust your network with the added restrictions in order to find out if and how this impacts Federico's business.

$F$  is the set of Federico's friends.

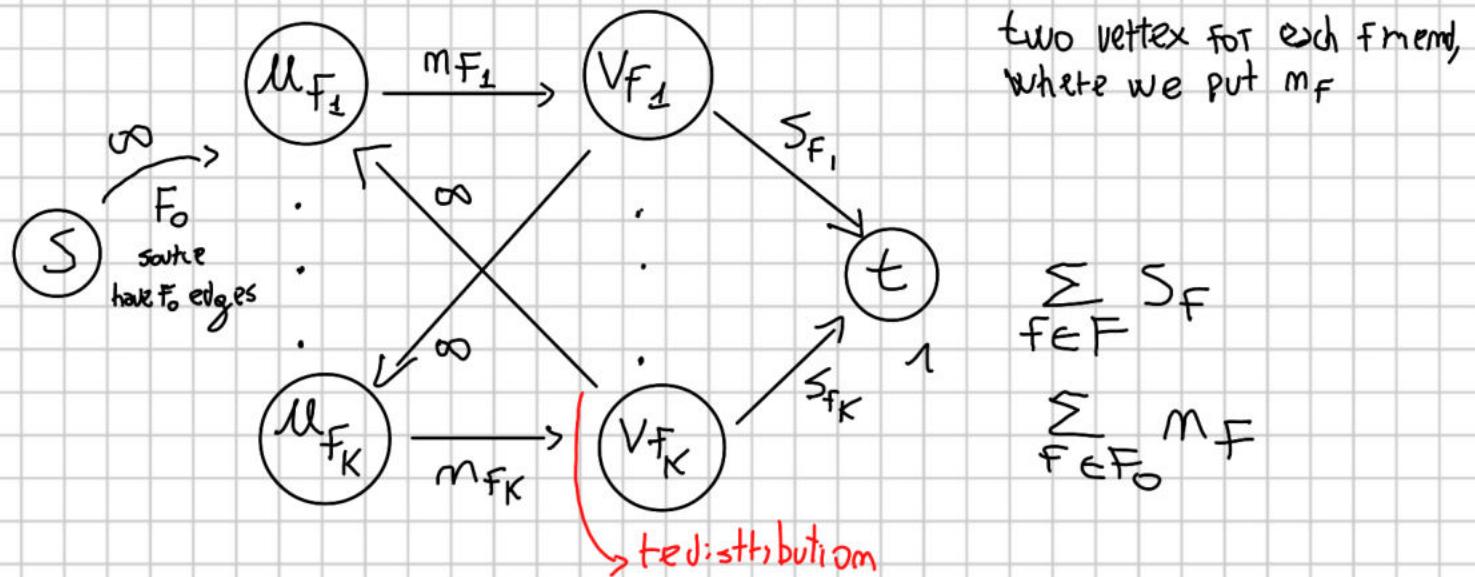
for each  $f \in F$  we define:

$m_f$ : number of chocolates that  $f$  can handle

$s_f$ : number of chocolates that  $f$  can sell.

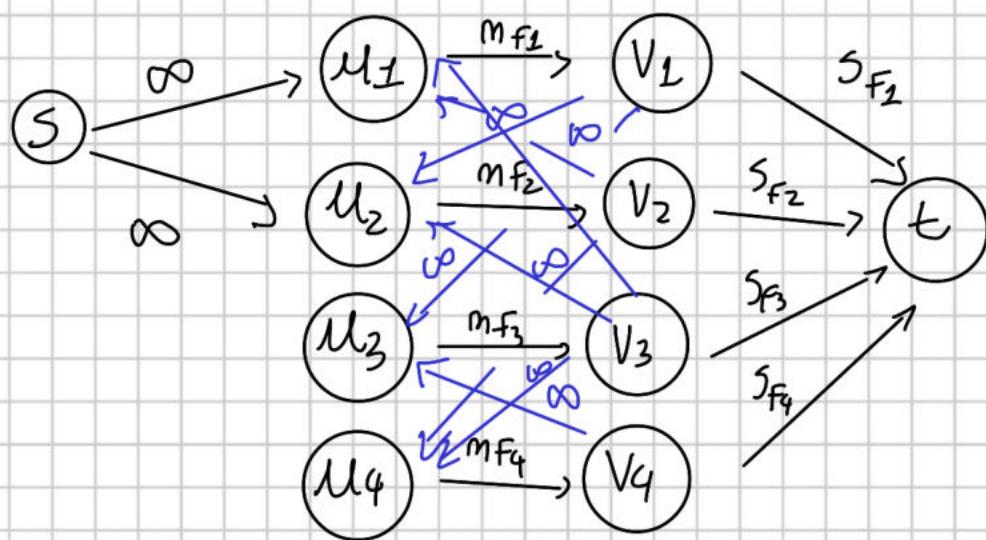
$F_0$ : number of friends that Federico will meet

$E \subseteq F \times F$  subset of all pairs of friend whenever they meet



complexity for calculate max-flow depend on algorithm

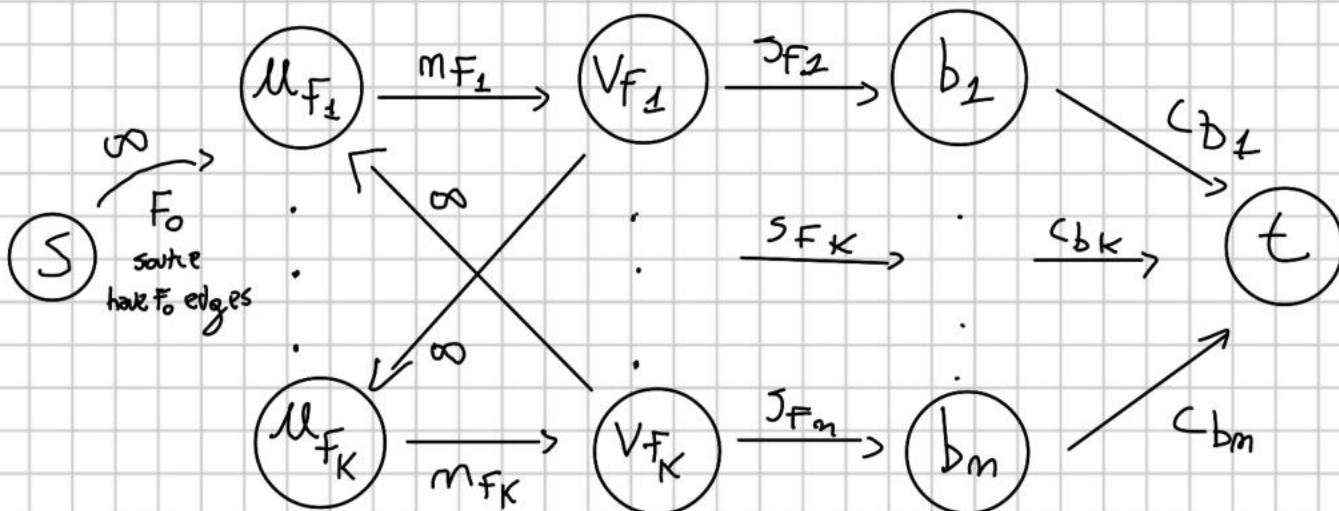
Ex.:  $|F| = 4$ ,  $F_0 = 2$



b)  $B$ : set of building

$b_F$ : associated building of  $F$

$c_b$ : capacity of building  $b$



Formalization:

$$V = \{S, t\} \cup \{(U_F, V_F) \mid F \in F\} \cup B \text{ Vertices}$$

$$E = \{S \times F_0 \cup \{(U_F, V_F) \mid F \in F\} \cup \{(V_F, b_F) \mid F \in F\}$$

$$\text{edges} \cup \{(b, t) \mid b \in B\} \cup \{(V_F, U_{F'}) \mid (F, F') \in E\}$$

$$C(S, \underline{\quad}) = \infty \quad C(V_F, U_{F'}) = \infty \quad C(b, t) = c_b$$

$$C(U_F, V_F) = m_F \quad C(V_F, b_F) = s_F \text{ capacities}$$

## Exercise 2

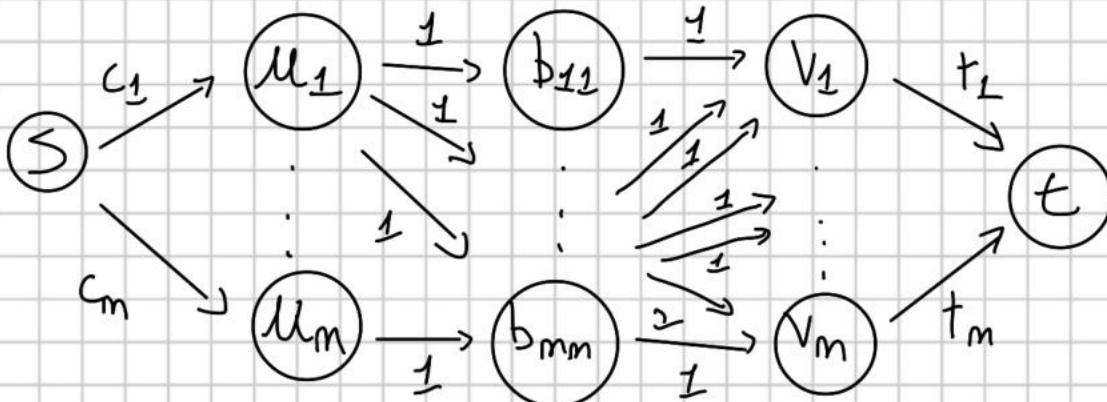
You are the owner of a large chain of franchise shops, and you would like to expand to a new city. The blocks in your city make an  $n \times n$  grid. However, although your products are awesome and in high demand, the city will not allow you to open a shop in each block. Instead, for every row of blocks  $i$ , you are given a number  $r_i$  that limits the maximum number of shops opened there - and for every column  $j$ , there also is a maximum number  $c_j$ .

- a) Find the maximum number of franchise shops you can legally open in the city. To do so, model the problem as a flow network. Then, describe how to get the right answer using Ford-Fulkerson, and prove the correctness of your construction.
- b) To really become known, you would like to make sure that people driving through the city have a good chance at seeing at least one of your shops. Therefore, at least one shop should be placed every few streets. More exactly, for row  $\{1, \dots, 5\}$  of the grid, there should be at least one block in these rows that has a shop, and the same should hold for row  $\{6, \dots, 10\}$ , et cetera. Conversely, for every five columns of the grid, you want to ensure also at least one shop. Adjust your network such that it can be used to determine whether this requirement can be fulfilled. Here, you are allowed to not also pose upper bounds on the capacity on each edge, but also state a lower bound on the minimum amount of flow you want on the edge. Give a variation of the Ford-Fulkerson algorithm that solves the problem for this richer type of network. Prove that together, your network and the algorithm solve the decision problem described above, while also maximizing the number of shops in case of a 'yes'-instance.

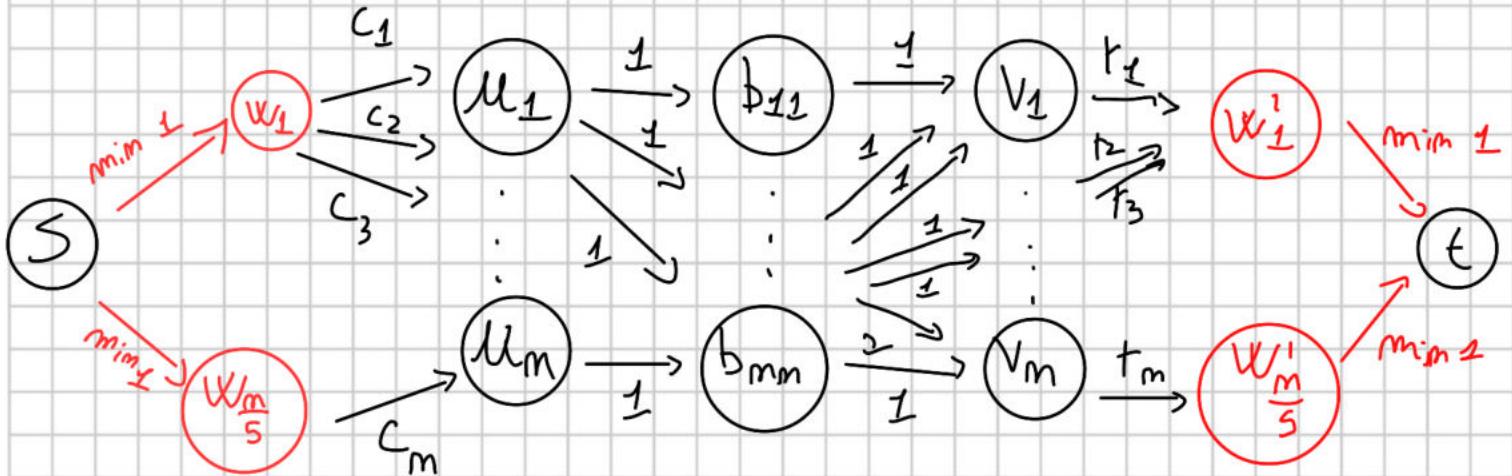
$m \times m$  size of the grid

$t_i$ : capacity on row  $i$

$c_j$ : capacity on column  $j$



b)

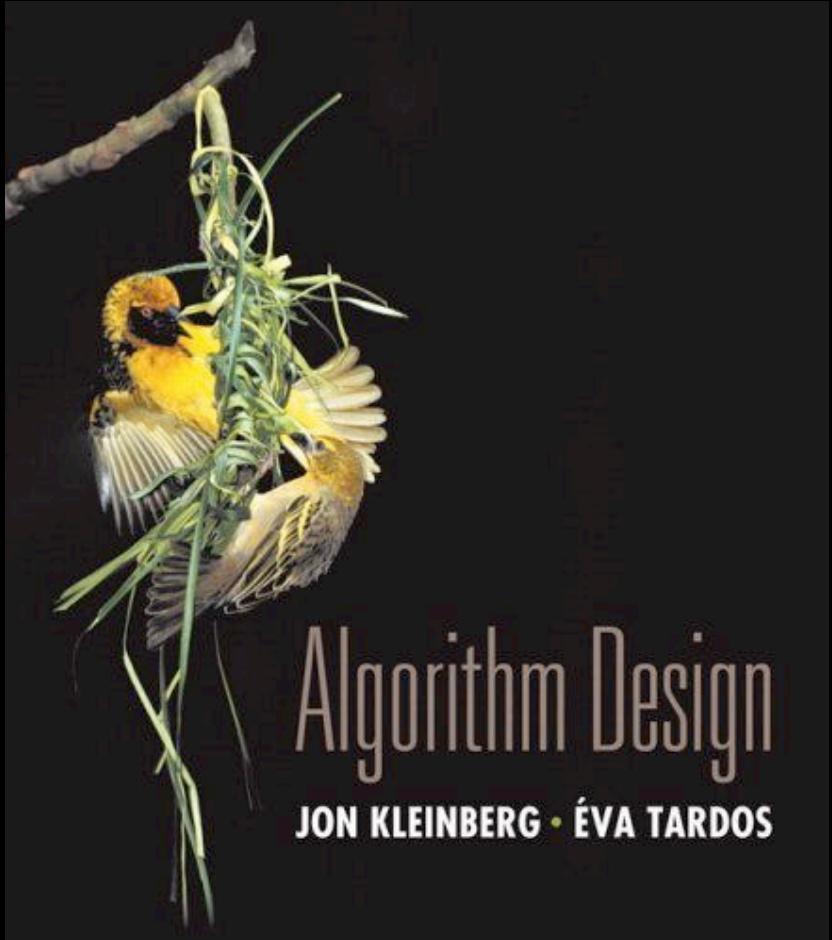


formalized:

$$V = \{S, t\} \cup \{w_1, \dots, w_{m/5}\} \cup \{u_1, \dots, u_m\} \cup \{b_{1L}, \dots, b_{mL}\}$$
$$\cup \{v_1, \dots, v_m\} \cup \{w_1', \dots, w_{m/5}'\}$$

$$E = (S, w_1) \cup \{(w_i, u_{5i-j}) \mid i=1, \dots, m/5, j=0, \dots, 4\}$$
$$\cup \{v_{5i-j}, w_i' \mid i=1, \dots, m/5, j=0, \dots, 4\} \cup \{u_i, b_{ij}\} \cup \{b_{ij}, v_j\}$$
$$\cup \{w_i', t\}$$

Capacities are immediate by graph.



# Chapter 8

## NP and Computational Intractability



Slides by Kevin Wayne.  
Copyright © 2005 Pearson-Addison Wesley.  
All rights reserved.

# Algorithm Design Patterns and Anti-Patterns

## Algorithm design patterns.

- Greedy.
- Divide-and-conquer.
- Dynamic programming.
- Duality.
- **Reductions.**
- Local search.
- Randomization.

## Ex.

- $O(n \log n)$  interval scheduling.
- $O(n \log n)$  FFT.
- $O(n^2)$  edit distance.
- $O(n^3)$  bipartite matching.

## Algorithm design anti-patterns.

- **NP-completeness.**
  - **PSPACE-completeness.**
  - Undecidability.
- $O(n^k)$  algorithm unlikely.
  - $O(n^k)$  certification algorithm unlikely.
  - No algorithm possible.

## 8.1 Polynomial-Time Reductions

---

# Classify Problems According to Computational Requirements

Q. Which problems will we be able to solve in practice?

A working definition. [von Neumann 1953, Godel 1956, Cobham 1964, Edmonds 1965, Rabin 1966]  
Those with polynomial-time algorithms.

know how to solve in polytime		don't know polytime algorithm
Yes	Probably no	
Shortest path		Longest path
Matching		3D-matching
Min cut		Max cut
2-SAT		3-SAT
Planar 4-color		Planar 3-color
Bipartite vertex cover		Vertex cover
Primality testing		Factoring

## Classify Problems

**Desiderata.** Classify problems according to those that can be solved in polynomial-time and those that cannot.

Provably requires exponential-time.

→ halting problem

- Given a Turing machine, does it halt in at most k steps?
- Given a board position in an n-by-n generalization of chess, can black guarantee a win?

Frustrating news. Huge number of fundamental problems have defied classification for decades.

This chapter. Show that these fundamental problems are "computationally equivalent" and appear to be different manifestations of one really hard problem.

## Polynomial-Time Reduction

Desiderata'. Suppose we could solve  $X$  in polynomial-time. What else could we solve in polynomial time?

don't confuse with reduces from

Reduction. Problem  $X$  polynomial reduces to problem  $Y$  if arbitrary instances of problem  $X$  can be solved using:

- Polynomial number of standard computational steps, plus instances of problem  $Y$ .
- Polynomial number of calls to oracle that solves problem  $Y$ .

Notation.  $X \leq_p Y$ .

computational model supplemented by special piece of hardware that solves instances of  $Y$  in a single step

Remarks.

- We pay for time to write down instances sent to black box  $\Rightarrow$  instances of  $Y$  must be of polynomial size.
- Note: Cook reducibility.

in contrast to Karp reductions

## Polynomial-Time Reduction

Purpose. Classify problems according to **relative** difficulty.

**Design algorithms.** If  $X \leq_p Y$  and  $Y$  can be solved in polynomial-time,  
then  $X$  can also be solved in polynomial time.

**Establish intractability.** If  $X \leq_p Y$  and  $X$  cannot be solved in  
polynomial-time, then  $Y$  cannot be solved in polynomial time.

**Establish equivalence.** If  $X \leq_p Y$  and  $Y \leq_p X$ , we use notation  $X \equiv_p Y$ .



up to cost of reduction

# Reduction By Simple Equivalence

---

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

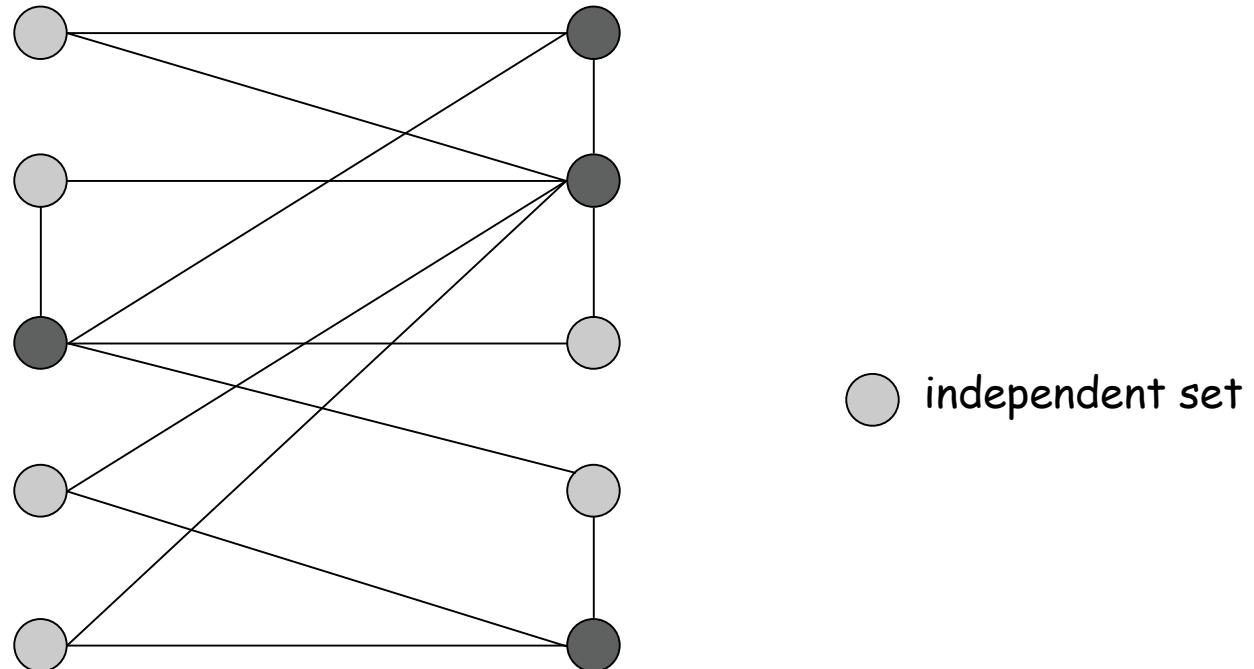
## Independent Set

**INDEPENDENT SET:** Given a graph  $G = (V, E)$  and an integer  $k$ , is there a subset of vertices  $S \subseteq V$  such that  $|S| \geq k$ , and for each edge at most one of its endpoints is in  $S$ ?

→ every edge at most one edge of  $S$

Ex. Is there an independent set of size  $\geq 6$ ? Yes.

Ex. Is there an independent set of size  $\geq 7$ ? No.



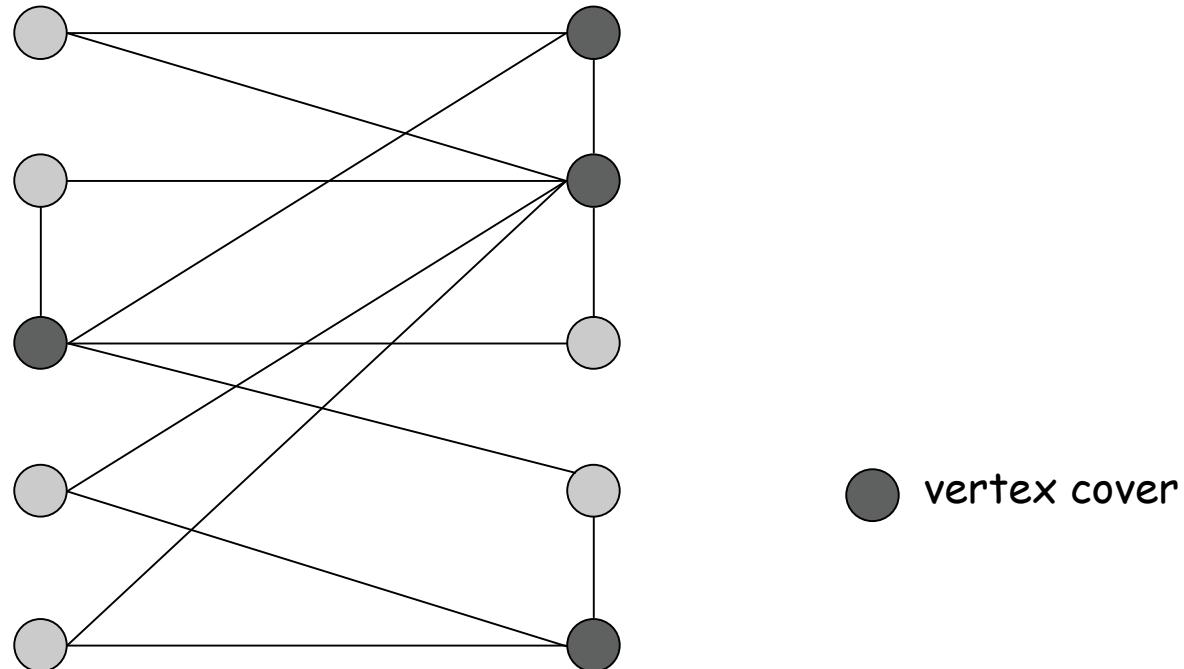
## Vertex Cover

**VERTEX COVER:** Given a graph  $G = (V, E)$  and an integer  $k$ , is there a subset of vertices  $S \subseteq V$  such that  $|S| \leq k$ , and for each edge, at least one of its endpoints is in  $S$ ?

Ex. Is there a vertex cover of size  $\leq 4$ ? Yes.

Ex. Is there a vertex cover of size  $\leq 3$ ? No.

↳ every edge have at least one of the two endpoint of  $S$

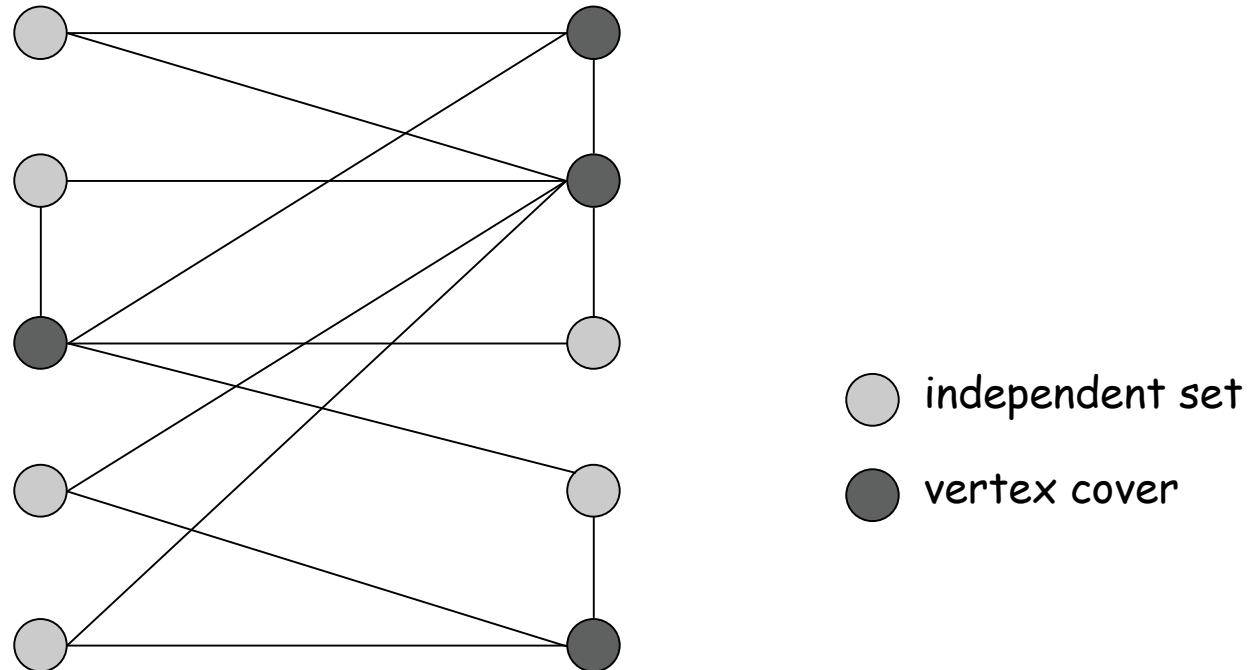


# Vertex Cover and Independent Set

**Claim.** VERTEX-COVER  $\equiv_p$  INDEPENDENT-SET.

Pf. We show  $S$  is an independent set iff  $V - S$  is a vertex cover.

---



## Vertex Cover and Independent Set

**Claim.** VERTEX-COVER  $\equiv_p$  INDEPENDENT-SET.

Pf. We show  $S$  is an independent set iff  $V - S$  is a vertex cover.



- Let  $S$  be any independent set.
- Consider an arbitrary edge  $(u, v)$ . ↗ either/or
- $S$  independent  $\Rightarrow u \notin S$  or  $v \notin S \Rightarrow u \in V - S$  or  $v \in V - S$ .
- Thus,  $V - S$  covers  $(u, v)$ .



- Let  $V - S$  be any vertex cover.
- Consider two nodes  $u \in S$  and  $v \in S$ . ↗ neither  $u$  and  $v$  are in vertex cover, not possible that exist edge  $(u, v)$
- Observe that  $(u, v) \notin E$  since  $V - S$  is a vertex cover.
- Thus, no two nodes in  $S$  are joined by an edge  $\Rightarrow S$  independent set. ■

# Reduction from Special Case to General Case

---

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

, not optimal solution, they  
are not equivalent

problem  $X$  is a subclass of problem  $Y$

## Set Cover

→ need to cover

**SET COVER:** Given a set  $U$  of elements, a collection  $S_1, S_2, \dots, S_m$  of subsets of  $U$ , and an integer  $k$ , does there exist a collection of  $\leq k$  of these sets whose union is equal to  $U$ ?

↳ univers

$S_i$  may overlap with  $S_j$

### Sample application.

- m available pieces of software.
- Set  $U$  of n capabilities that we would like our system to have.
- The  $i$ th piece of software provides the set  $S_i \subseteq U$  of capabilities.
- Goal: achieve all  $n$  capabilities using fewest pieces of software.

Ex:

$$U = \{1, 2, 3, 4, 5, 6, 7\} \text{ Univers}$$

$k = 2$  number of sets to choose for cover  $U$

$$S_1 = \{3, 7\} \quad S_4 = \{2, 4\}$$

$$S_2 = \{3, 4, 5, 6\} \quad S_5 = \{5\}$$

$$S_3 = \{1\} \quad S_6 = \{1, 2, 6, 7\}$$

# Vertex Cover Reduces to Set Cover

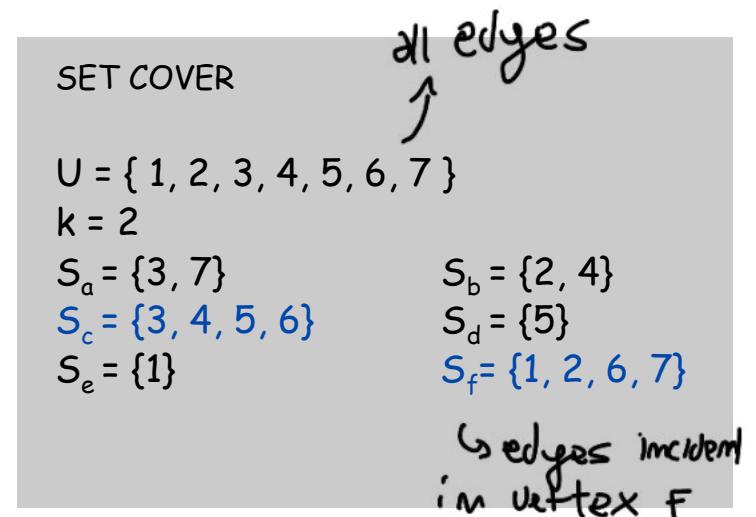
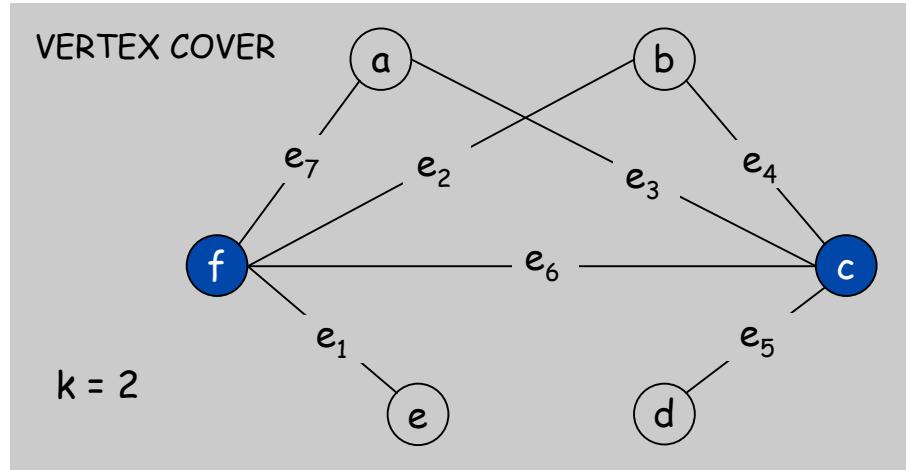
**Claim.** VERTEX-COVER  $\leq_p$  SET-COVER.

Pf. Given a VERTEX-COVER instance  $G = (V, E)$ ,  $k$ , we construct a set cover instance whose size equals the size of the vertex cover instance.

↳ exist a solution with key set and key vertex.

Construction.

- Create SET-COVER instance:  
-  $k = k$ ,  $U = E$ ,  $S_v = \{e \in E : e \text{ incident to } v\}$
- Set-cover of size  $\leq k$  iff vertex cover of size  $\leq k$ . ■



# Polynomial-Time Reduction

## Basic strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

→ can appear in clause

## Satisfiability

**Literal:** A Boolean variable or its negation.

→ negation of  $x$ :  
 $x_i$  or  $\overline{x_i}$

**Clause:** A disjunction of literals.

$$C_j = x_1 \vee \overline{x_2} \vee x_3$$

**Conjunctive normal form:** A propositional formula  $\Phi$  that is the conjunction of clauses.

$$\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$$

**SAT:** Given CNF formula  $\Phi$ , does it have a satisfying truth assignment?

**3-SAT:** SAT where each clause contains exactly 3 literals.

↳ exist a truth assignment?

each corresponds to a different variable

↳ combination of literal assignment that satisfies 3-SAT

Ex:  $(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$

Yes:  $x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{false}$ .

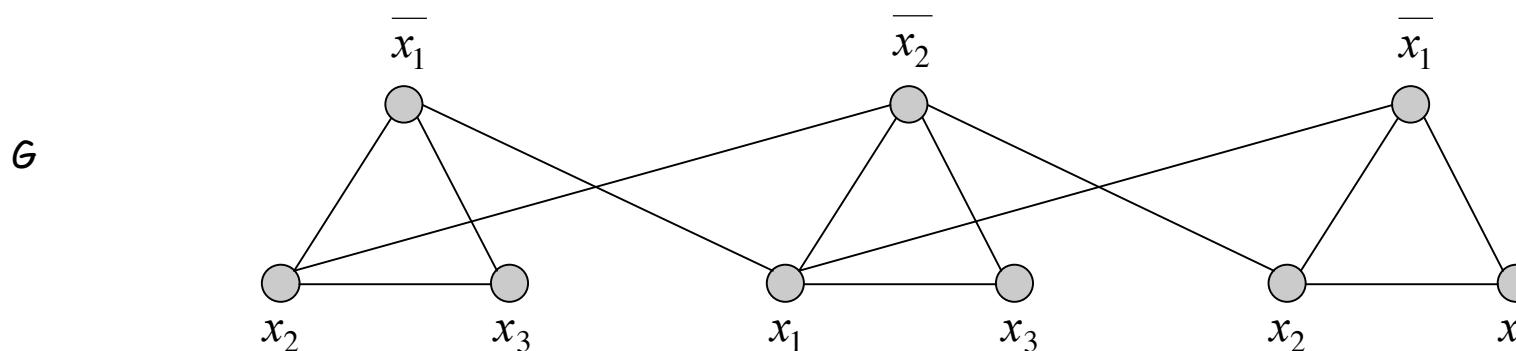
### 3 Satisfiability Reduces to Independent Set

**Claim.**  $\text{3-SAT} \leq_p \text{INDEPENDENT-SET}$ .

**Pf.** Given an instance  $\Phi$  of 3-SAT, we construct an instance  $(G, k)$  of INDEPENDENT-SET that has an independent set of size  $k$  iff  $\Phi$  is satisfiable.

**Construction.**

- $G$  contains 3 vertices for each clause, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.



$$k = 3$$

$$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

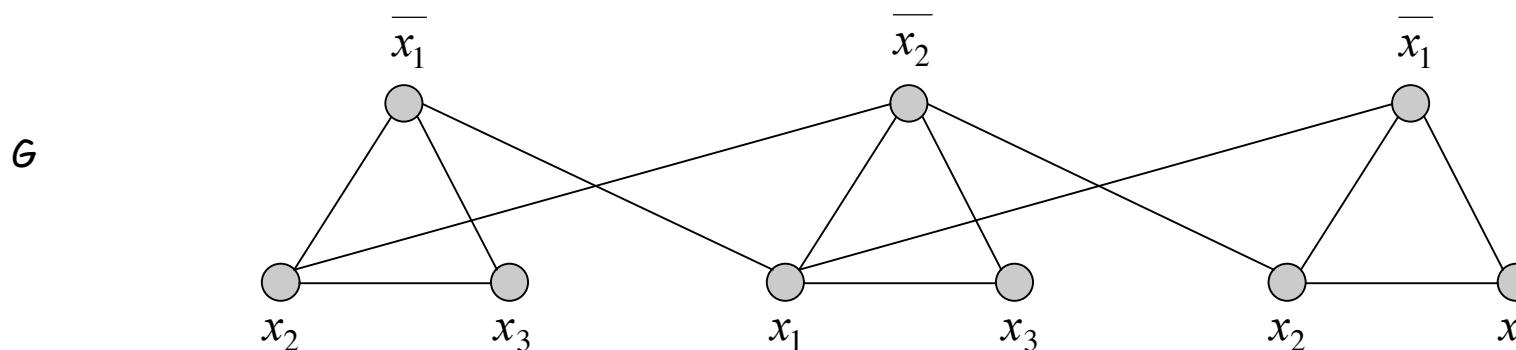
### 3 Satisfiability Reduces to Independent Set

**Claim.**  $G$  contains independent set of size  $k = |\Phi|$  iff  $\Phi$  is satisfiable.

**Pf.**  $\Rightarrow$  Let  $S$  be independent set of size  $k$ .

- $S$  must contain exactly one vertex in each triangle.
- Set these literals to true.  $\leftarrow$  and any other variables in a consistent way
- Truth assignment is consistent and all clauses are satisfied.

**Pf**  $\Leftarrow$  Given satisfying assignment, select one true literal from each triangle. This is an independent set of size  $k$ . ■



$$k = 3$$

$$\Phi = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4)$$

## Review

### Basic reduction strategies.

- Simple equivalence:  $\text{INDEPENDENT-SET} =_P \text{VERTEX-COVER}$ .
- Special case to general case:  $\text{VERTEX-COVER} \leq_P \text{SET-COVER}$ .
- Encoding with gadgets:  $\text{3-SAT} \leq_P \text{INDEPENDENT-SET}$ .

Transitivity. If  $X \leq_P Y$  and  $Y \leq_P Z$ , then  $X \leq_P Z$ .

Pf idea. Compose the two algorithms.

Ex:  $\text{3-SAT} \leq_P \text{INDEPENDENT-SET} \leq_P \text{VERTEX-COVER} \leq_P \text{SET-COVER}$ .

## Self-Reducibility

Decision problem. Does there exist a vertex cover of size  $\leq k$ ?

Search problem. Find vertex cover of minimum cardinality.

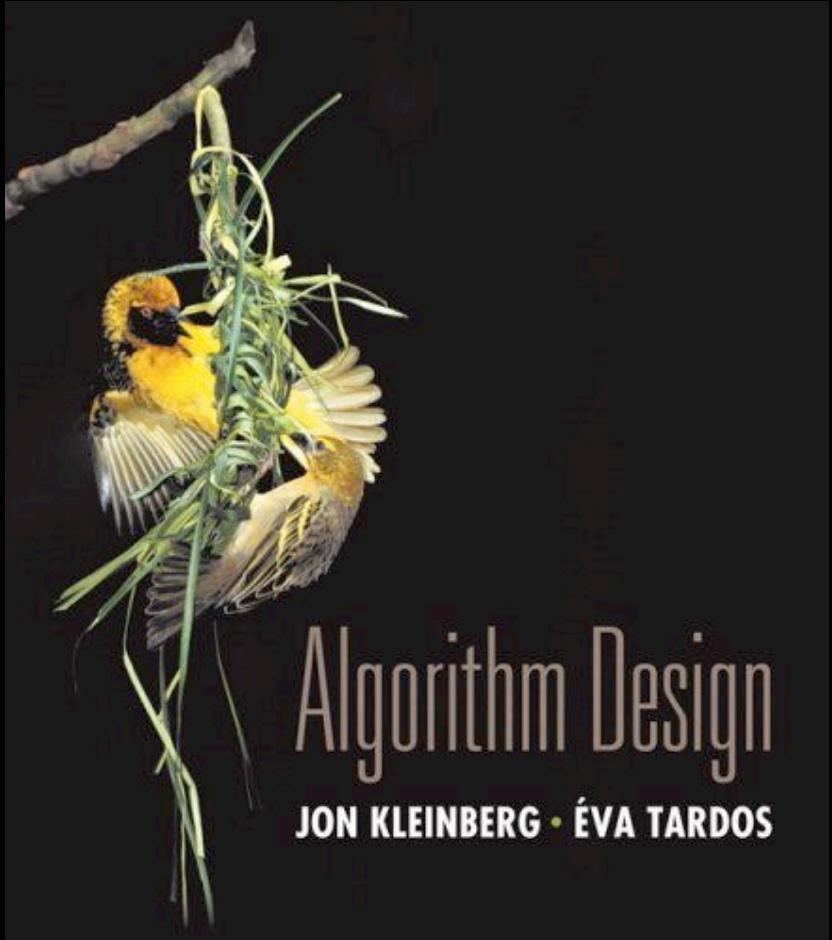
Self-reducibility. Search problem  $\leq_p$  decision version.

- Applies to all (NP-complete) problems in this chapter.
- Justifies our focus on decision problems.

Ex: to find min cardinality vertex cover.

- (Binary) search for cardinality  $k^*$  of min vertex cover.
- Find a vertex  $v$  such that  $G - \{v\}$  has a vertex cover of size  $\leq k^* - 1$ .
  - any vertex in any min vertex cover will have this property
- Include  $v$  in the vertex cover.
- Recursively find a min vertex cover in  $G - \{v\}$ .

delete  $v$  and all incident edges



# Chapter 8

## NP and Computational Intractability



Slides by Kevin Wayne.  
Copyright © 2005 Pearson-Addison Wesley.  
All rights reserved.

## 8.3 Definition of NP

---

## Decision Problems

### Decision problem.

- $X$  is a set of strings.
- Instance: string  $s$ .
- Algorithm  $A$  solves problem  $X$ :  $A(s) = \text{yes}$  iff  $s \in X$ .

input instance

all strings that are instances of the problem define the language  
language of set of strings

Polynomial time. Algorithm  $A$  runs in poly-time if for every string  $s$ ,  
 $A(s)$  terminates in at most  $p(|s|)$  "steps", where  $p(\cdot)$  is some polynomial.

↑  
length of  $s$

PRIMES:  $X = \{ 2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, \dots \}$   $\rightsquigarrow$  decide if  $s$  is prime is done with a poly-time algorithm  
Algorithm. [Agrawal-Kayal-Saxena, 2002]  $p(|s|) = |s|^8$ .

## Definition of P

### P. Decision problems for which there is a poly-time algorithm.

---

Problem	Description	Algorithm	Yes	No
MULTIPLE	Is $x$ a multiple of $y$ ?	Grade school division	51, 17	51, 16
RELPRIME	Are $x$ and $y$ relatively prime?	Euclid (300 BCE)	34, 39	34, 51
PRIMES	Is $x$ prime?	AKS (2002)	53	51
EDIT-DISTANCE	Is the edit distance between $x$ and $y$ less than 5?	Dynamic programming	neither neither	acgggt ttttta
LSOLVE	Is there a vector $x$ that satisfies $Ax = b$ ?	Gauss-Edmonds elimination	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

Problem can be solved in polytime if there exist an algorithm that runs a number of steps which is polynomial in input size

NP

## Certification algorithm intuition.

- Certifier views things from "managerial" viewpoint.
- Certifier doesn't determine whether  $s \in X$  on its own; (yes or no instances) rather, it checks a proposed proof  $t$  that  $s \in X$ .

Def. Algorithm  $C(s, t)$  is a **certifier** for problem  $X$  if for every string  $s$ ,  $s \in X$  iff there exists a string  $t$  such that  $C(s, t) = \text{yes}$ .

"certificate" or "witness"  
Proof  $t$

↳ no, don't mean  $s \notin X$ , only  
mean that  $t$  is not a proof.

NP. Decision problems for which there exists a **poly-time certifier**.

$C(s, t)$  is a poly-time algorithm and  
 $|t| \leq p(|s|)$  for some polynomial  $p(\cdot)$ .  
witness of  $t$

Remark. NP stands for nondeterministic polynomial-time.

↳ problem that are solved in polynomial-time in nondeterministic computational model

## Certifiers and Certificates: Composite

COMPOSITES. Given an integer  $s$ , is  $s$  composite?

nontrivial factor

↳ complementary to primality

**Certificate.** A nontrivial factor  $t$  of  $s$ . Note that such a certificate exists iff  $s$  is composite. Moreover  $|t| \leq |s|$ .

Certifier.

```
boolean C(s, t) {  
    if (t ≤ 1 or t ≥ s)  
        return false  
    else if (s is a multiple of t)  
        return true  
    else  
        return false  
}
```

Instance.  $s = 437,669$ .

Certificate.  $t = 541$  or  $809$ .  $\leftarrow 437,669 = 541 \times 809$

**Conclusion.** COMPOSITES is in NP.

To check a solution in polytime

## Certifiers and Certificates: 3-Satisfiability

**SAT.** Given a CNF formula  $\Phi$ , is there a satisfying assignment?

**Certificate.** An assignment of truth values to the  $n$  boolean variables.

**Certifier.** Check that each clause in  $\Phi$  has at least one true literal.

**Ex.**

$$(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_3} \vee \overline{x_4})$$

instance  $s$

$$x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1$$

certificate  $t$

**Conclusion.** SAT is in NP.

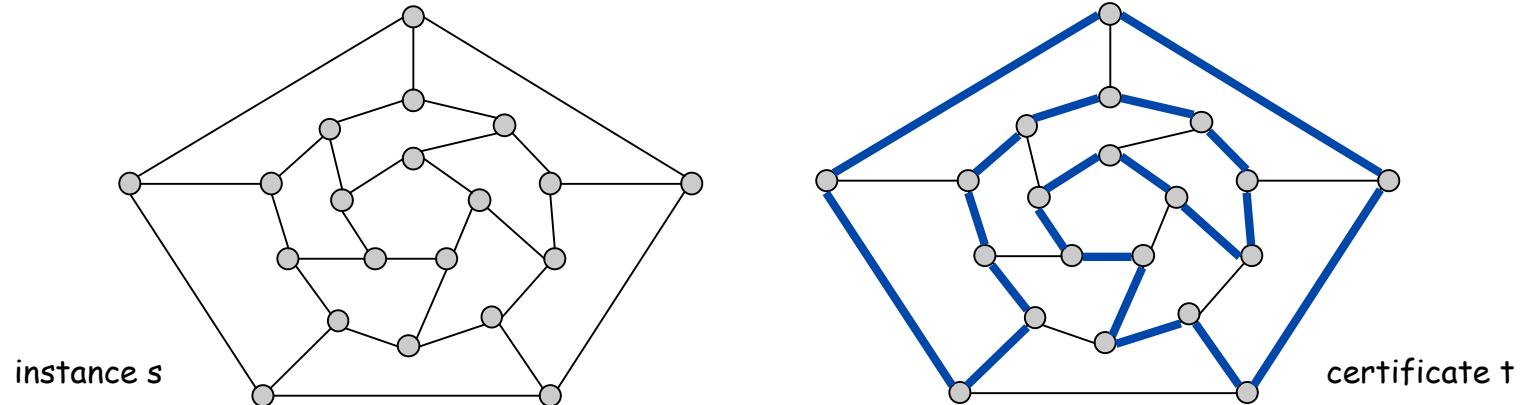
## Certifiers and Certificates: Hamiltonian Cycle

HAM-CYCLE. Given an undirected graph  $G = (V, E)$ , does there exist a simple cycle  $C$  that visits every node? , not a complete graph

**Certificate.** A permutation of the  $n$  nodes.

**Certifier.** Check that the permutation contains each node in  $V$  exactly once, and that there is an edge between each pair of adjacent nodes in the permutation.

**Conclusion.** HAM-CYCLE is in NP.



## P, NP, EXP

P. Decision problems for which there is a poly-time algorithm.

EXP. Decision problems for which there is an exponential-time algorithm.

NP. Decision problems for which there is a poly-time certifier.

Claim.  $P \subseteq NP$ .

Pf. Consider any problem  $X$  in  $P$ .

- By definition, there exists a poly-time algorithm  $A(s)$  that solves  $X$ .
- Certificate:  $t = \epsilon$ , certifier  $C(s, t) = A(s)$ . ■

↳ empty string

↳ polytime computation decide  $s \in X$

Claim.  $NP \subseteq EXP$ .

Pf. Consider any problem  $X$  in  $NP$ .

- By definition, there exists a poly-time certifier  $C(s, t)$  for  $X$ .
- To solve input  $s$ , run  $C(s, t)$  on all strings  $t$  with  $|t| \leq p(|s|)$ .  
↳ all possible proofs is checked
- Return yes, if  $C(s, t)$  returns yes for any of these. ■

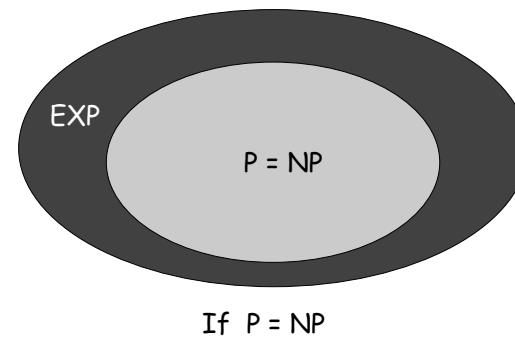
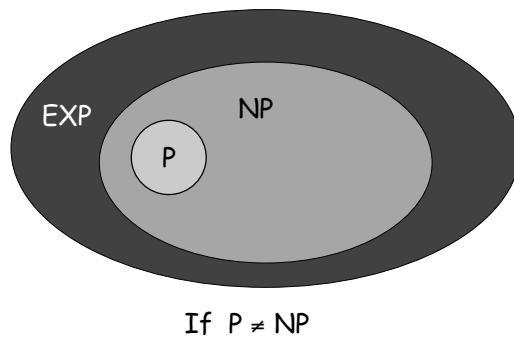
}

all proofs is checked in  
polytime

## The Main Question: P Versus NP

Does  $P = NP$ ? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]

- Is the decision problem as easy as the certification problem?
- Clay \$1 million prize.



would break RSA cryptography  
(and potentially collapse  
economy)

If yes: Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, ...

If no: No efficient algorithms possible for 3-COLOR, TSP, SAT, ...

Consensus opinion on  $P = NP$ ? Probably no.

## 8.4 NP-Completeness

---

## Polynomial Transformation

Def. Problem X polynomial reduces (Cook) to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.

Def. Problem X polynomial transforms (Karp) to problem Y if given any input  $x$  to X, we can construct an input  $y$  such that  $x$  is a  $\text{yes}$  instance of X iff  $y$  is a  $\text{yes}$  instance of Y.



we require  $|y|$  to be of size polynomial in  $|x|$

*is believed that  $Karp \Rightarrow Cook$*

Note. Polynomial transformation is polynomial reduction with just one call to oracle for Y, exactly at the end of the algorithm for X. Almost all previous reductions were of this form.

Open question. Are these two concepts the same?



we abuse notation  $\leq_p$  and blur distinction

## NP-Complete

NP-complete. A problem  $Y$  in NP with the property that for every problem  $X$  in NP,  $X \leq_p Y$ .

Theorem. Suppose  $Y$  is an NP-complete problem. Then  $Y$  is solvable in poly-time iff  $P = NP$ .

Pf.  $\Leftarrow$  If  $P = NP$  then  $Y$  can be solved in poly-time since  $Y$  is in NP.

Pf.  $\Rightarrow$  Suppose  $Y$  can be solved in poly-time.

- Let  $X$  be any problem in NP. Since  $X \leq_p Y$ , we can solve  $X$  in poly-time. This implies  $NP \subseteq P$ .
- We already know  $P \subseteq NP$ . Thus  $P = NP$ . ■

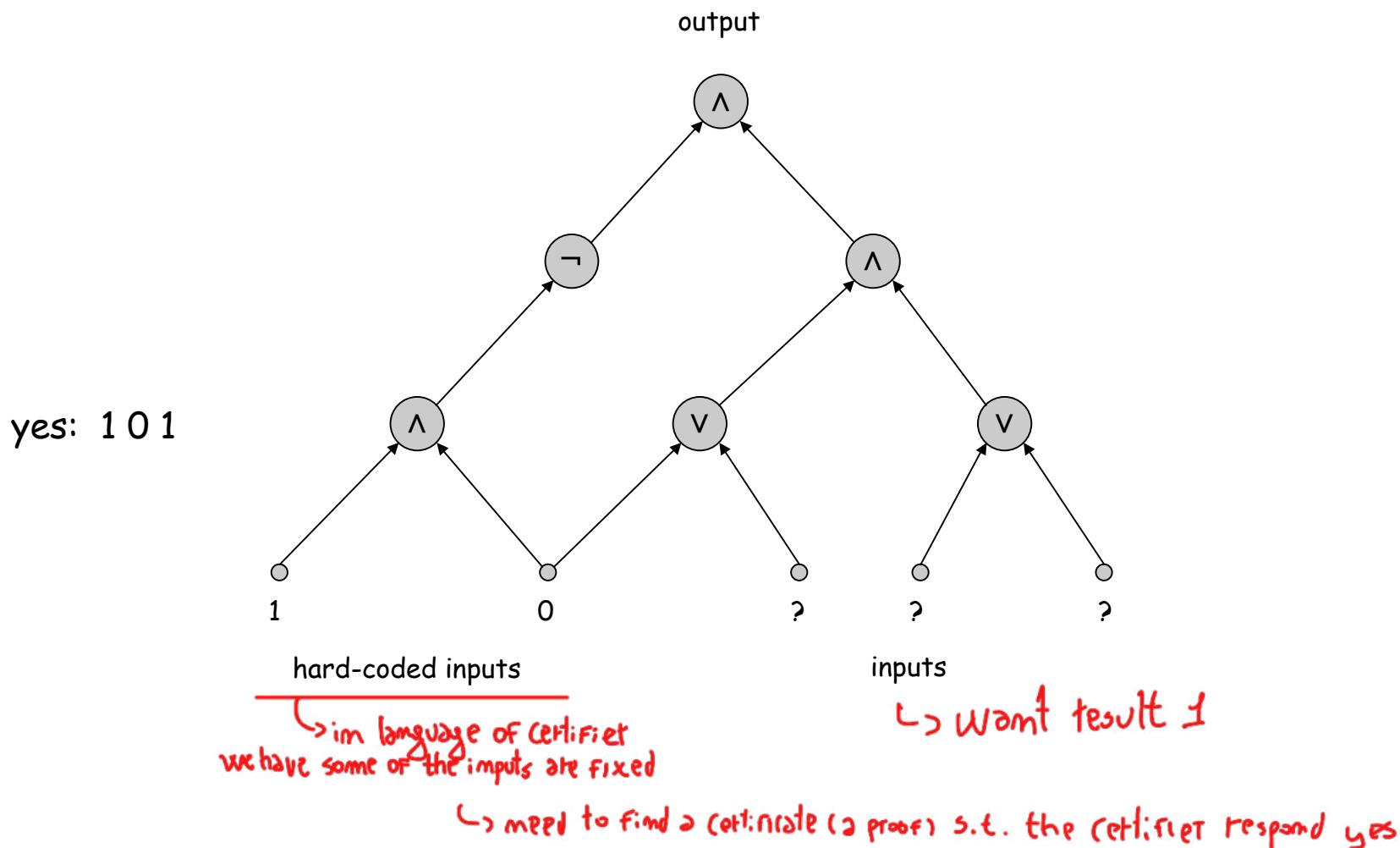
Fundamental question. Do there exist "natural" NP-complete problems?

have a polynomial size

## Circuit Satisfiability

type of logical gate

CIRCUIT-SAT. Given a combinational circuit built out of AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1?



## The "First" NP-Complete Problem

**Theorem.** CIRCUIT-SAT is NP-complete. [Cook 1971, Levin 1973]

Pf. (sketch)

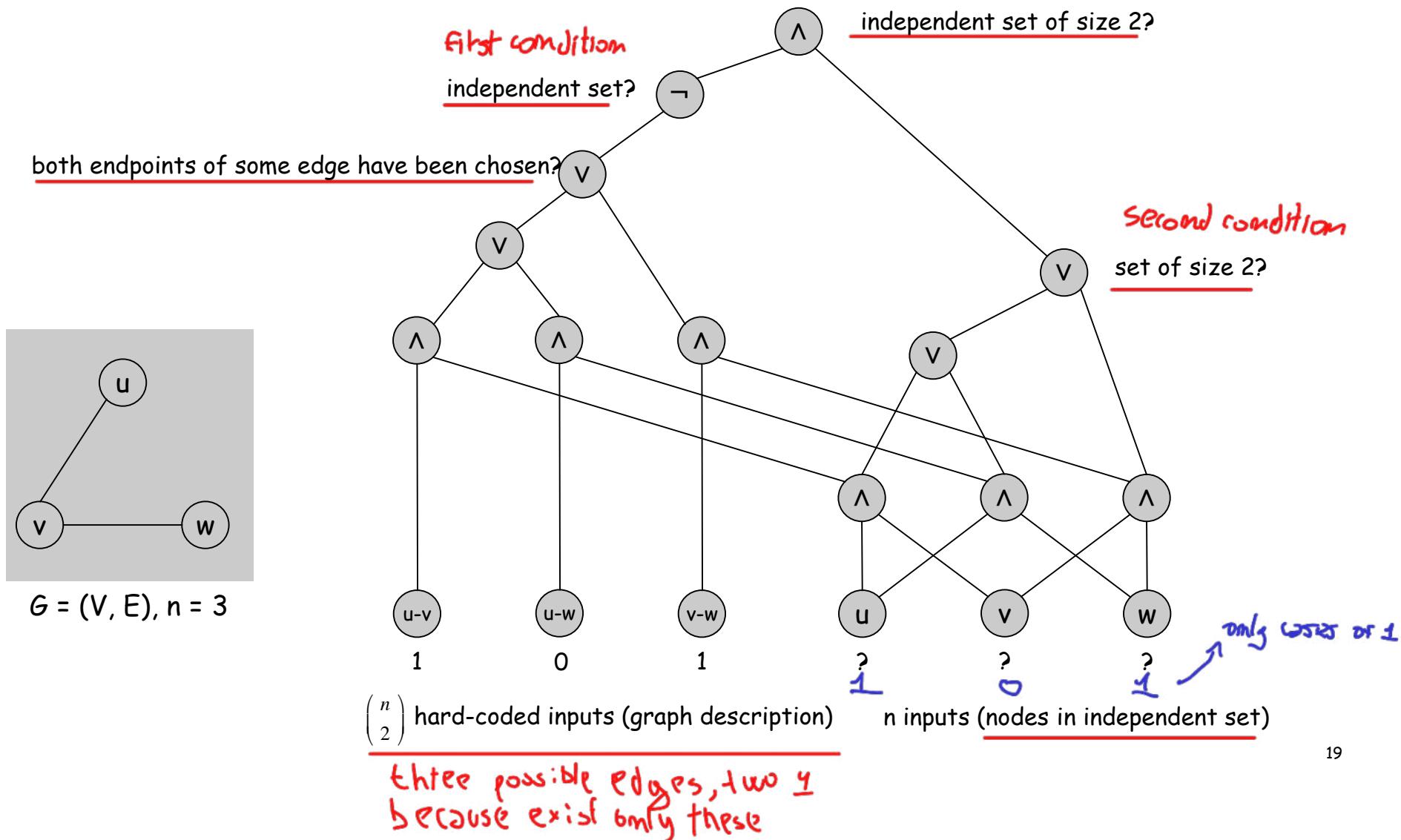
- Any algorithm that takes a fixed number of bits  $n$  as input and produces a yes/no answer can be represented by such a circuit.  
Moreover, if algorithm takes poly-time, then circuit is of poly-size.

sketchy part of proof; fixing the number of bits is important,  
and reflects basic distinction between algorithms and circuits

- Consider some problem  $X$  in NP. It has a poly-time certifier  $C(s, t)$ .  
To determine whether  $s$  is in  $X$ , need to know if there exists a certificate  $t$  of length  $p(|s|)$  such that  $C(s, t) = \text{yes}$ .
- View  $C(s, t)$  as an algorithm on  $|s| + p(|s|)$  bits (input  $s$ , certificate  $t$ ) and convert it into a poly-size circuit  $K$ .
  - first  $|s|$  bits are hard-coded with  $s$
  - remaining  $p(|s|)$  bits represent bits of  $t$
- Circuit  $K$  is satisfiable iff  $C(s, t) = \text{yes}$ .

→ deciding if exist an independent set of size 2 in a graph of Example three nodes

Ex. Construction below creates a circuit K whose inputs can be set so that K outputs true iff graph G has an independent set of size 2.



## Establishing NP-Completeness

**Remark.** Once we establish first "natural" NP-complete problem, others fall like dominoes.

**Recipe to establish NP-completeness of problem Y.**

- Step 1. Show that Y is in NP.
- Step 2. Choose an NP-complete problem X.
- Step 3. Prove that  $X \leq_p Y$ .

**Justification.** If X is an NP-complete problem, and Y is a problem in NP with the property that  $X \leq_p Y$  then Y is NP-complete.

Pf. Let W be any problem in NP. Then  $W \leq_p X \leq_p Y$ .

- By transitivity,  $W \leq_p Y$ .
- Hence Y is NP-complete. ▀

$\uparrow$                        $\uparrow$   
by definition of        by assumption  
NP-complete

## 3-SAT is NP-Complete

Theorem. 3-SAT is NP-complete.

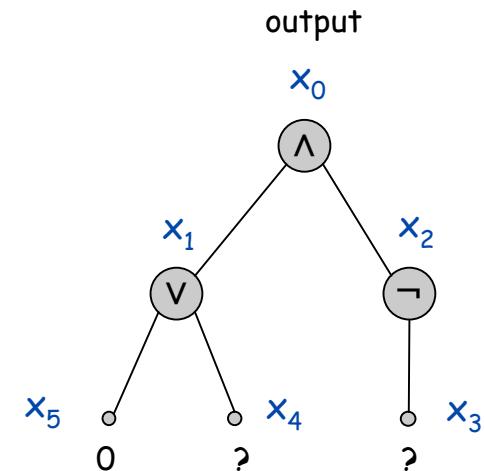
Pf. Suffices to show that CIRCUIT-SAT  $\leq_p$  3-SAT since 3-SAT is in NP.

- Let K be any circuit.
- Create a 3-SAT variable  $x_i$  for each circuit element i.
- Make circuit compute correct values at each node:
  - $x_2 = \neg x_3 \Rightarrow$  add 2 clauses:  $x_2 \vee x_3, \overline{x}_2 \vee \overline{x}_3$
  - $x_1 = x_4 \vee x_5 \Rightarrow$  add 3 clauses:  $x_1 \vee \overline{x}_4, x_1 \vee \overline{x}_5, \overline{x}_1 \vee x_4 \vee x_5$
  - $x_0 = x_1 \wedge x_2 \Rightarrow$  add 3 clauses:  $\overline{x}_0 \vee x_1, \overline{x}_0 \vee x_2, x_0 \vee \overline{x}_1 \vee \overline{x}_2$

- Hard-coded input values and output value.

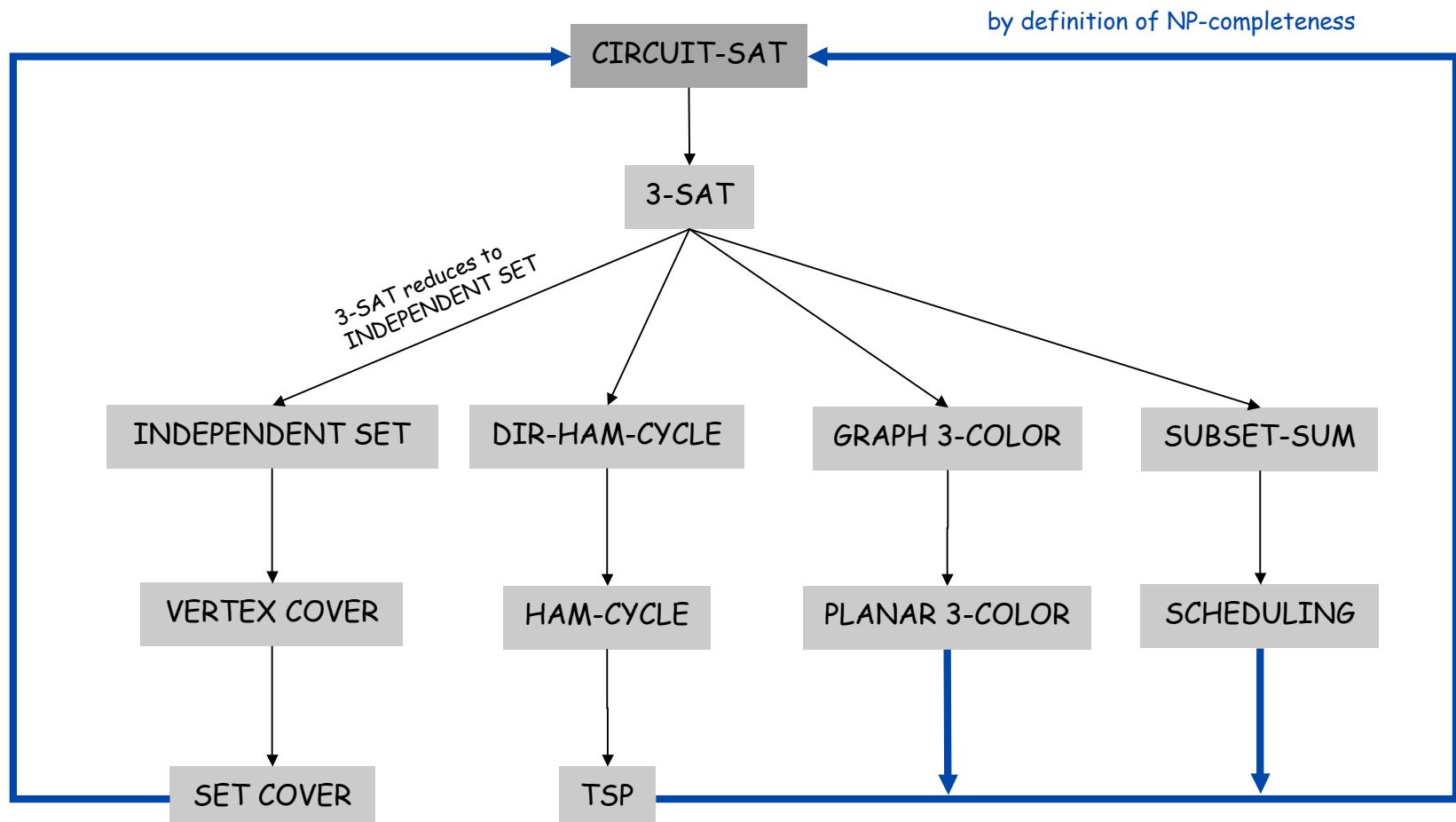
- $x_5 = 0 \Rightarrow$  add 1 clause:  $\overline{x}_5$
- $x_0 = 1 \Rightarrow$  add 1 clause:  $x_0$

- Final step: turn clauses of length < 3 into clauses of length exactly 3. ■



# NP-Completeness

**Observation.** All problems below are NP-complete and polynomial reduce to one another!



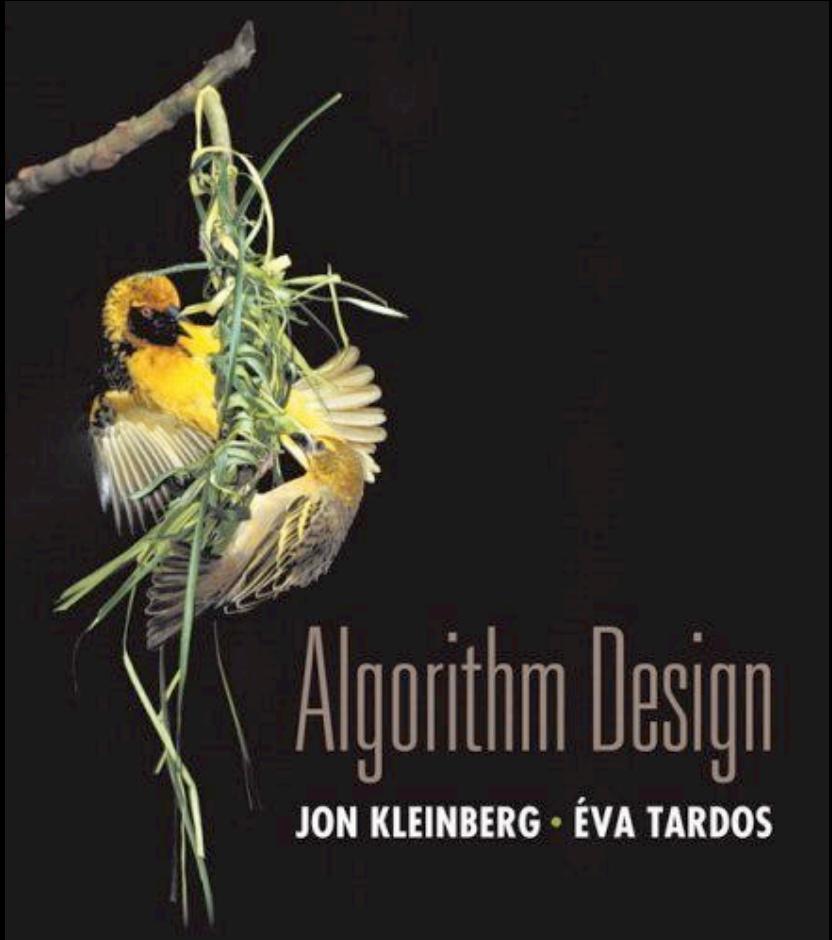
## Extent and Impact of NP-Completeness

### Extent of NP-completeness. [Papadimitriou 1995]

- Prime intellectual export of CS to other disciplines.
- 6,000 citations per year (title, abstract, keywords).
  - more than "compiler", "operating system", "database"
- Broad applicability and classification power.
- "Captures vast domains of computational, scientific, mathematical endeavors, and seems to roughly delimit what mathematicians and scientists had been aspiring to compute feasibly."

### NP-completeness can guide scientific inquiry.

- 1926: Ising introduces simple model for phase transitions.
- 1944: Onsager solves 2D case in tour de force.
- 19xx: Feynman and other top minds seek 3D solution.
- 2000: Istrail proves 3D problem NP-complete.



# Chapter 8

## NP and Computational Intractability



Slides by Kevin Wayne.  
Copyright © 2005 Pearson-Addison Wesley.  
All rights reserved.

## 8.5 Sequencing Problems

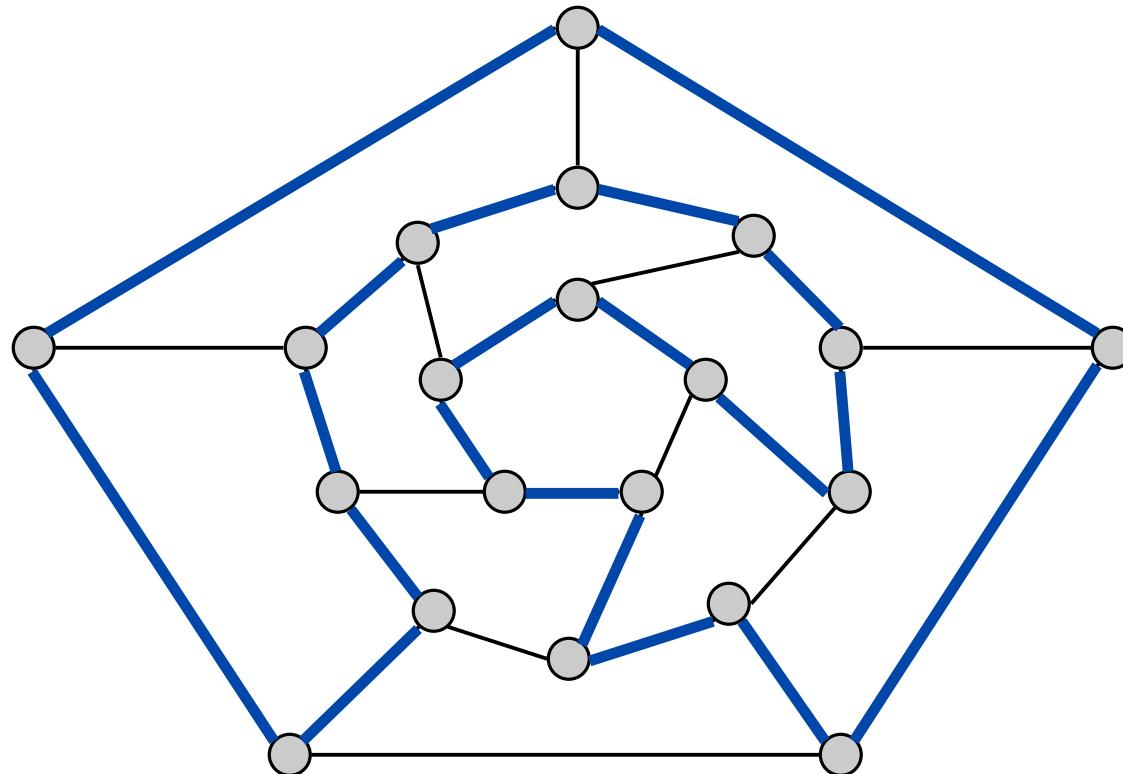
---

Basic genres.

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- **Sequencing problems:** HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

## Hamiltonian Cycle

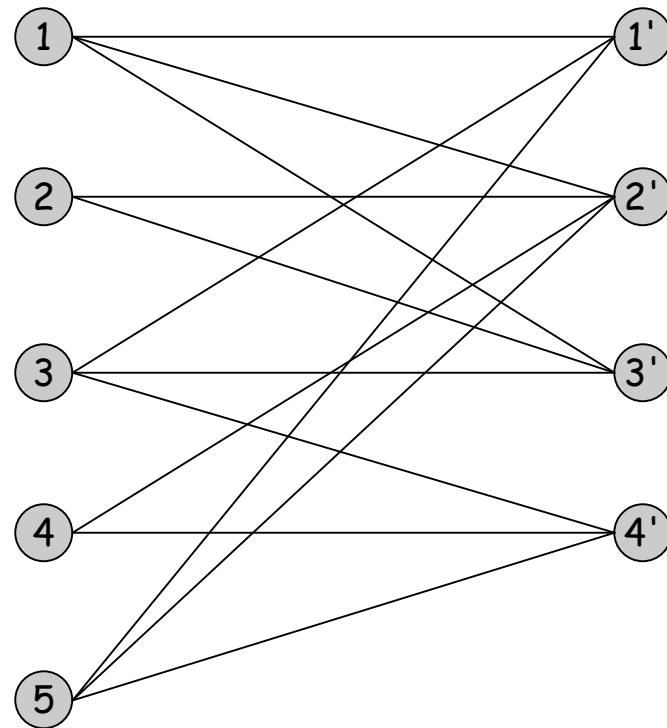
**HAM-CYCLE:** given an undirected graph  $G = (V, E)$ , does there exist a simple cycle  $\Gamma$  that contains every node in  $V$ .



YES: vertices and faces of a dodecahedron.

## Hamiltonian Cycle

**HAM-CYCLE:** given an undirected graph  $G = (V, E)$ , does there exist a simple cycle  $\Gamma$  that contains every node in  $V$ .



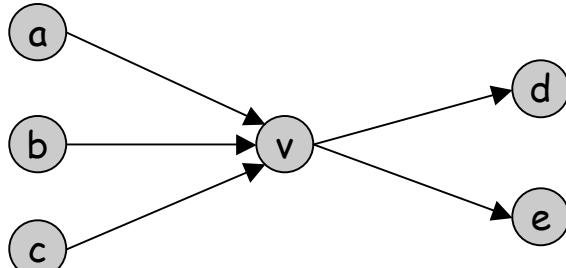
NO: bipartite graph with odd number of nodes.

## Directed Hamiltonian Cycle

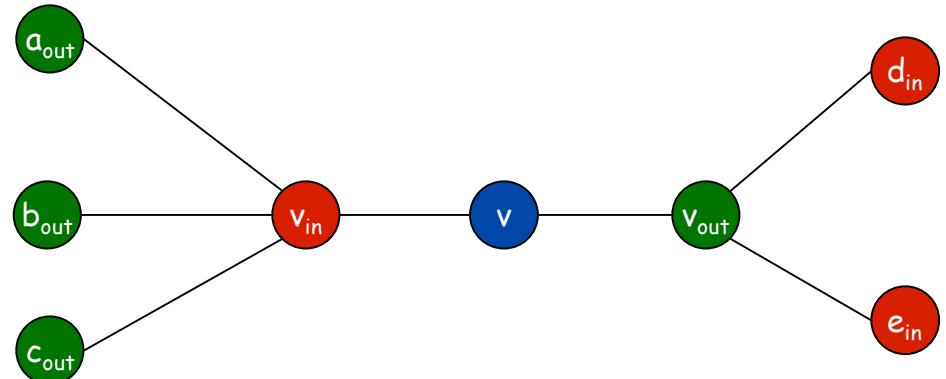
**DIR-HAM-CYCLE:** given a digraph  $G = (V, E)$ , does there exist a simple directed cycle  $\Gamma$  that contains every node in  $V$ ?

**Claim.** DIR-HAM-CYCLE  $\leq_p$  HAM-CYCLE.

**Pf.** Given a directed graph  $G = (V, E)$ , construct an undirected graph  $G'$  with  $3n$  nodes.



$G$



$G'$

## Directed Hamiltonian Cycle

**Claim.**  $G$  has a Hamiltonian cycle iff  $G'$  does.

**Pf.  $\Rightarrow$**

- Suppose  $G$  has a directed Hamiltonian cycle  $\Gamma$ .
- Then  $G'$  has an undirected Hamiltonian cycle (same order).

**Pf.  $\Leftarrow$**

- Suppose  $G'$  has an undirected Hamiltonian cycle  $\Gamma'$ .
- $\Gamma'$  must visit nodes in  $G'$  using one of following two orders:
  - ..., B, G, R, B, G, R, B, G, R, B, ...
  - ..., B, R, G, B, R, G, B, R, G, B, ...
- Blue nodes in  $\Gamma'$  make up directed Hamiltonian cycle  $\Gamma$  in  $G$ , or reverse of one. ▪

## 3-SAT Reduces to Directed Hamiltonian Cycle

**Claim.**  $\text{3-SAT} \leq_p \text{DIR-HAM-CYCLE}$ .

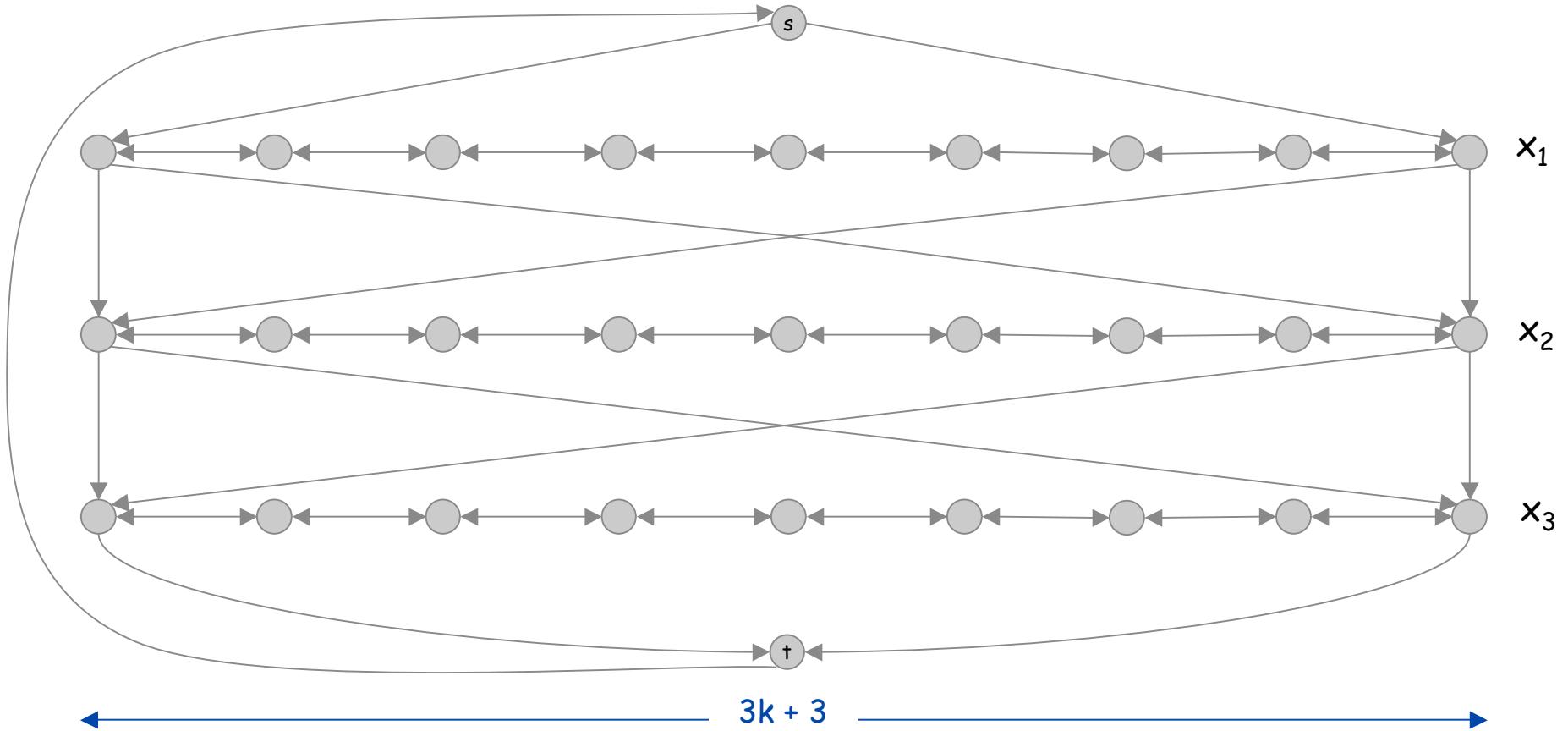
**Pf.** Given an instance  $\Phi$  of 3-SAT, we construct an instance of DIR-HAM-CYCLE that has a Hamiltonian cycle iff  $\Phi$  is satisfiable.

**Construction.** First, create graph that has  $2^n$  Hamiltonian cycles which correspond in a natural way to  $2^n$  possible truth assignments.

## 3-SAT Reduces to Directed Hamiltonian Cycle

**Construction.** Given 3-SAT instance  $\Phi$  with  $n$  variables  $x_i$  and  $k$  clauses.

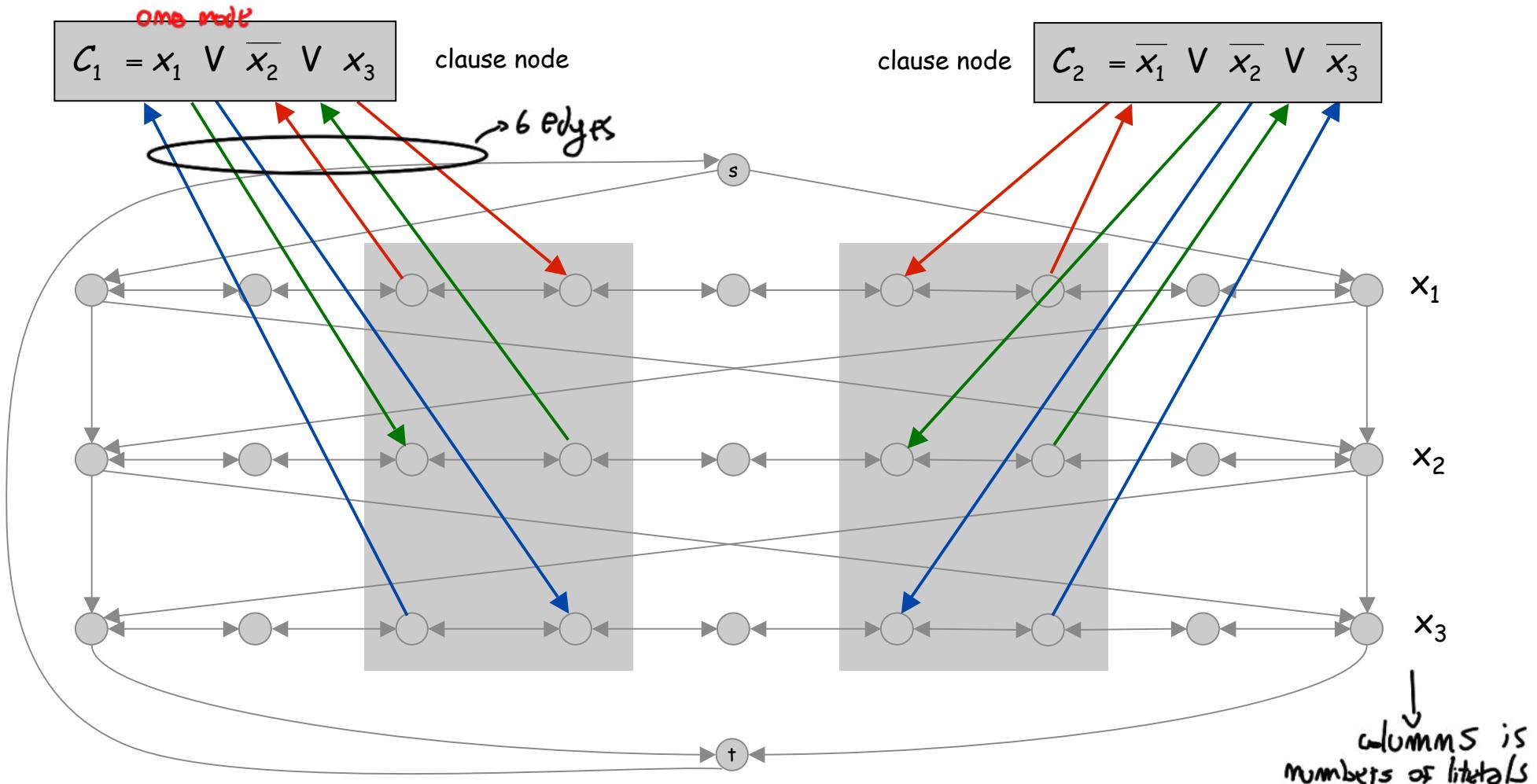
- Construct  $G$  to have  $2^n$  Hamiltonian cycles.
- Intuition: traverse path  $i$  from left to right  $\Leftrightarrow$  set variable  $x_i = 1$ .



# 3-SAT Reduces to Directed Hamiltonian Cycle

**Construction.** Given 3-SAT instance  $\Phi$  with  $n$  variables  $x_i$  and  $k$  clauses.

- For each clause: add a node and 6 edges.



## 3-SAT Reduces to Directed Hamiltonian Cycle

**Claim.**  $\Phi$  is satisfiable iff  $G$  has a Hamiltonian cycle.

Pf.  $\Rightarrow$

- Suppose 3-SAT instance has satisfying assignment  $x^*$ .
- Then, define Hamiltonian cycle in  $G$  as follows:
  - if  $x^*_i = 1$ , traverse row  $i$  from left to right
  - if  $x^*_i = 0$ , traverse row  $i$  from right to left
  - for each clause  $C_j$ , there will be at least one row  $i$  in which we are going in "correct" direction to splice node  $C_j$  into tour

$\rightarrow$  is NP-complete

## 3-SAT Reduces to Directed Hamiltonian Cycle

**Claim.**  $\Phi$  is satisfiable iff  $G$  has a Hamiltonian cycle.

Pf.  $\Leftarrow$

- Suppose  $G$  has a Hamiltonian cycle  $\Gamma$ .
- If  $\Gamma$  enters clause node  $C_j$ , it must depart on mate edge.
  - thus, nodes immediately before and after  $C_j$  are connected by an edge  $e$  in  $G$
  - removing  $C_j$  from cycle, and replacing it with edge  $e$  yields Hamiltonian cycle on  $G - \{C_j\}$
- Continuing in this way, we are left with Hamiltonian cycle  $\Gamma'$  in  $G - \{C_1, C_2, \dots, C_k\}$ .
- Set  $x^*_i = 1$  iff  $\Gamma'$  traverses row  $i$  left to right.
- Since  $\Gamma$  visits each clause node  $C_j$ , at least one of the paths is traversed in "correct" direction, and each clause is satisfied. ■

## Longest Path

**SHORTEST-PATH.** Given a digraph  $G = (V, E)$ , does there exists a simple path of length **at most**  $k$  edges?

**LONGEST-PATH.** Given a digraph  $G = (V, E)$ , does there exists a simple path of length **at least**  $k$  edges?

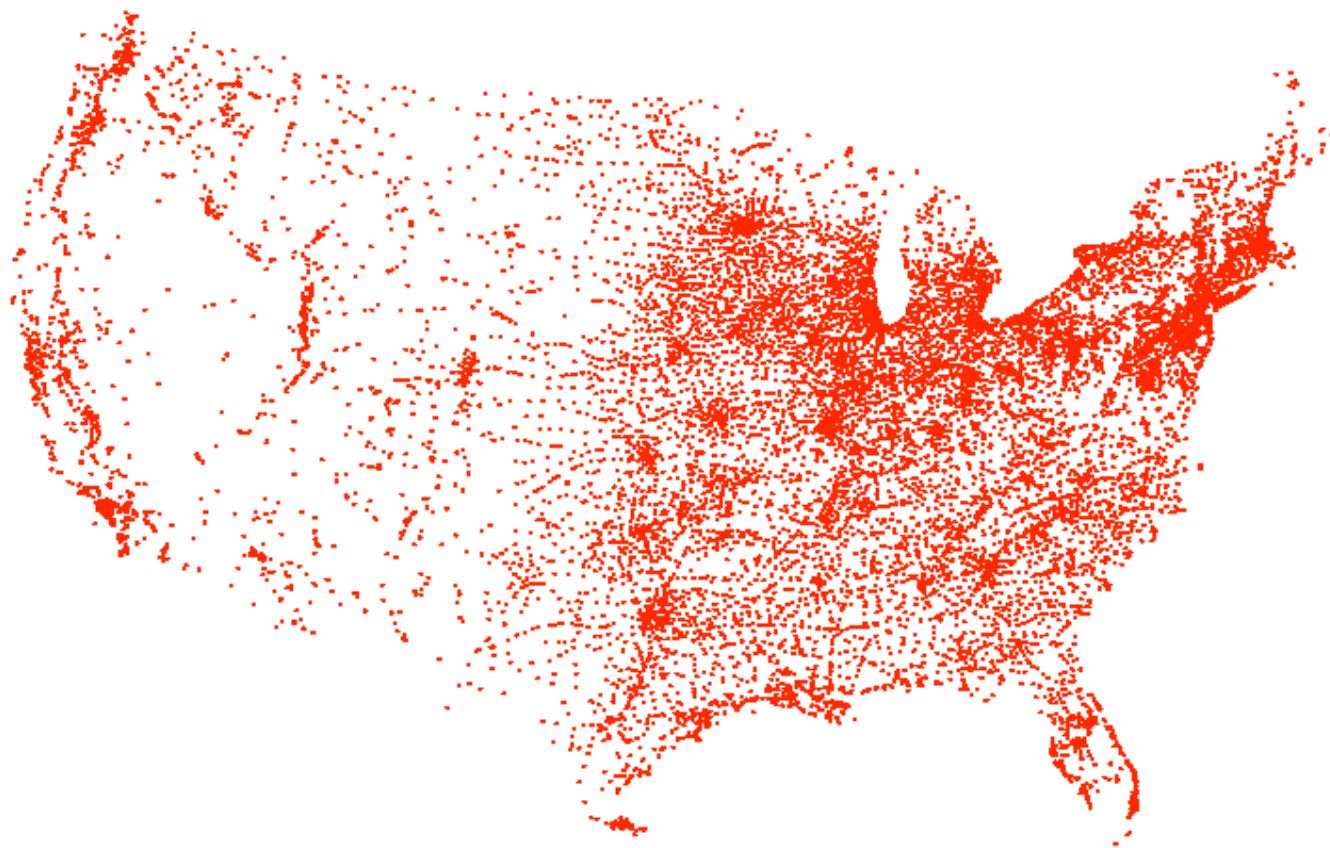
**Claim.**  $3\text{-SAT} \leq_p \text{LONGEST-PATH}$ .

Pf 1. Redo proof for **DIR-HAM-CYCLE**, ignoring back-edge from  $t$  to  $s$ .

Pf 2. Show **HAM-CYCLE**  $\leq_p \text{LONGEST-PATH}$ .

## Traveling Salesperson Problem

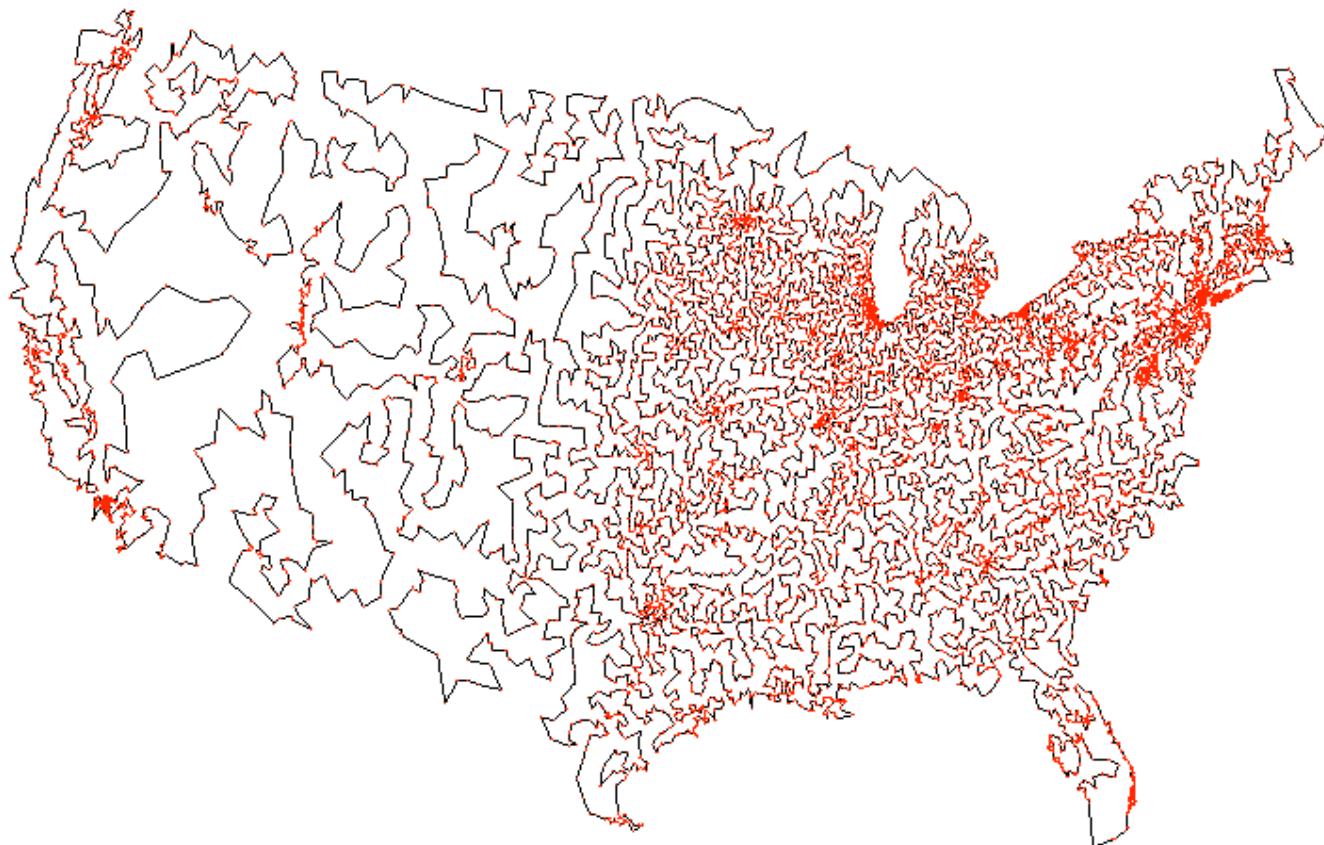
TSP. Given a set of  $n$  cities and a pairwise distance function  $d(u, v)$ , is there a tour of length  $\leq D$ ?



All 13,509 cities in US with a population of at least 500  
Reference: <http://www.tsp.gatech.edu>

## Traveling Salesperson Problem

TSP. Given a set of  $n$  cities and a pairwise distance function  $d(u, v)$ , is there a tour of length  $\leq D$ ?



Optimal TSP tour  
Reference: <http://www.tsp.gatech.edu>

## Traveling Salesperson Problem

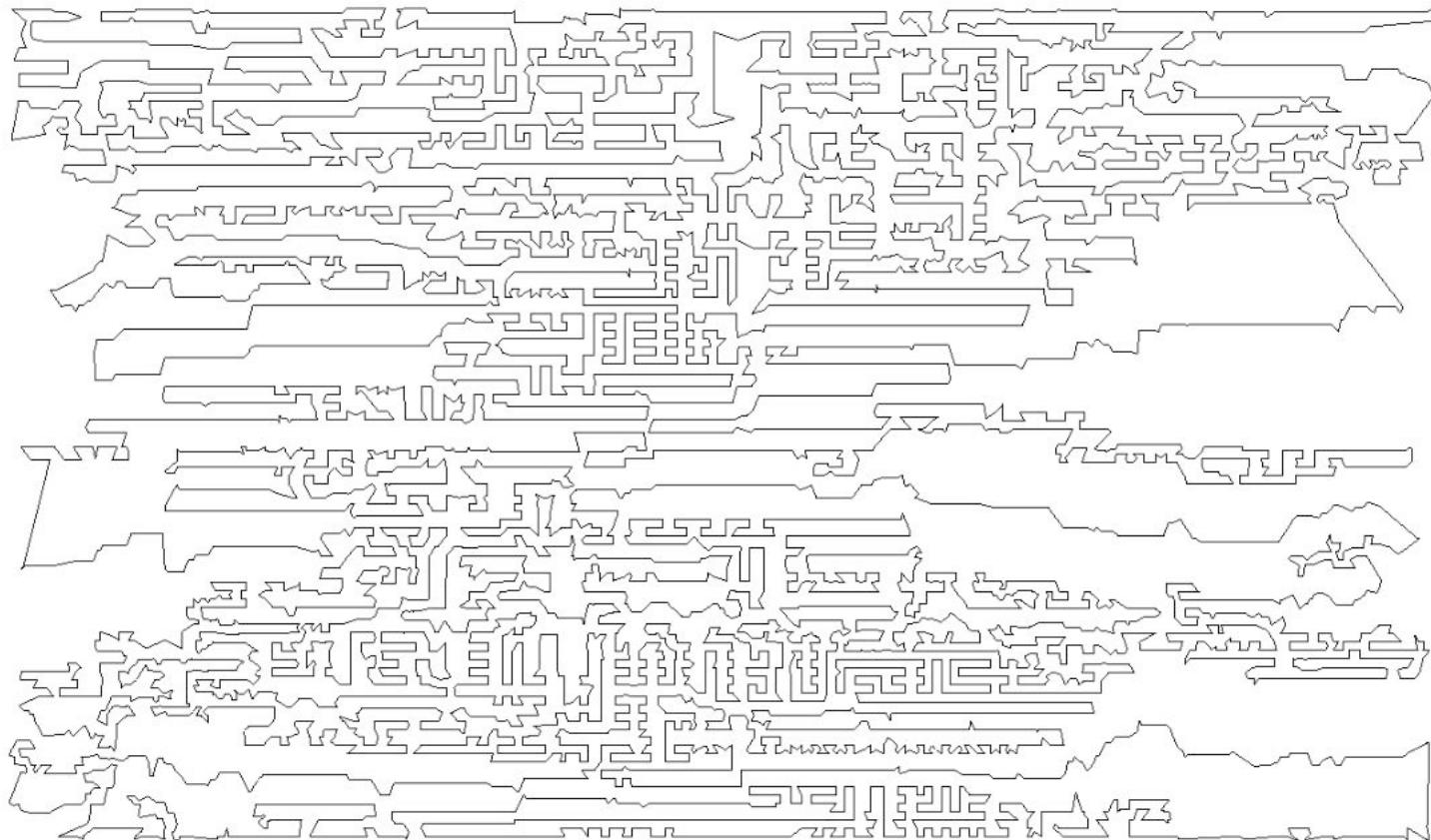
TSP. Given a set of  $n$  cities and a pairwise distance function  $d(u, v)$ , is there a tour of length  $\leq D$ ?



11,849 holes to drill in a programmed logic array  
Reference: <http://www.tsp.gatech.edu>

## Traveling Salesperson Problem

TSP. Given a set of  $n$  cities and a pairwise distance function  $d(u, v)$ , is there a tour of length  $\leq D$ ?



Optimal TSP tour  
Reference: <http://www.tsp.gatech.edu>

## Traveling Salesperson Problem

TSP. Given a set of  $n$  cities and a pairwise distance function  $d(u, v)$ , is there a tour of length  $\leq D$ ?

HAM-CYCLE: given a graph  $G = (V, E)$ , does there exists a simple cycle that contains every node in  $V$ ?

**Claim.** HAM-CYCLE  $\leq_p$  TSP.

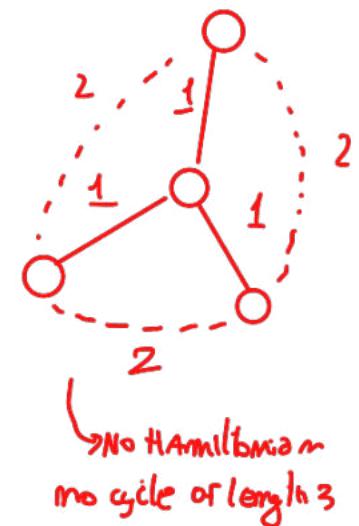
**Pf.**

- Given instance  $G = (V, E)$  of HAM-CYCLE, create  $n$  cities with distance function

$$d(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 2 & \text{if } (u, v) \notin E \end{cases}$$

- TSP instance has tour of length  $\leq n$  iff  $G$  is Hamiltonian. ■

We have HC if no vertex of  $d(u, v) = 2$  in TSP and vice versa



**Remark.** TSP instance in reduction satisfies  $\Delta$ -inequality.

$$d(v, \sqrt{v}) + d(\sqrt{v}, z) \geq d(v, z)$$

## 8.7 Graph Coloring

---

Basic genres.

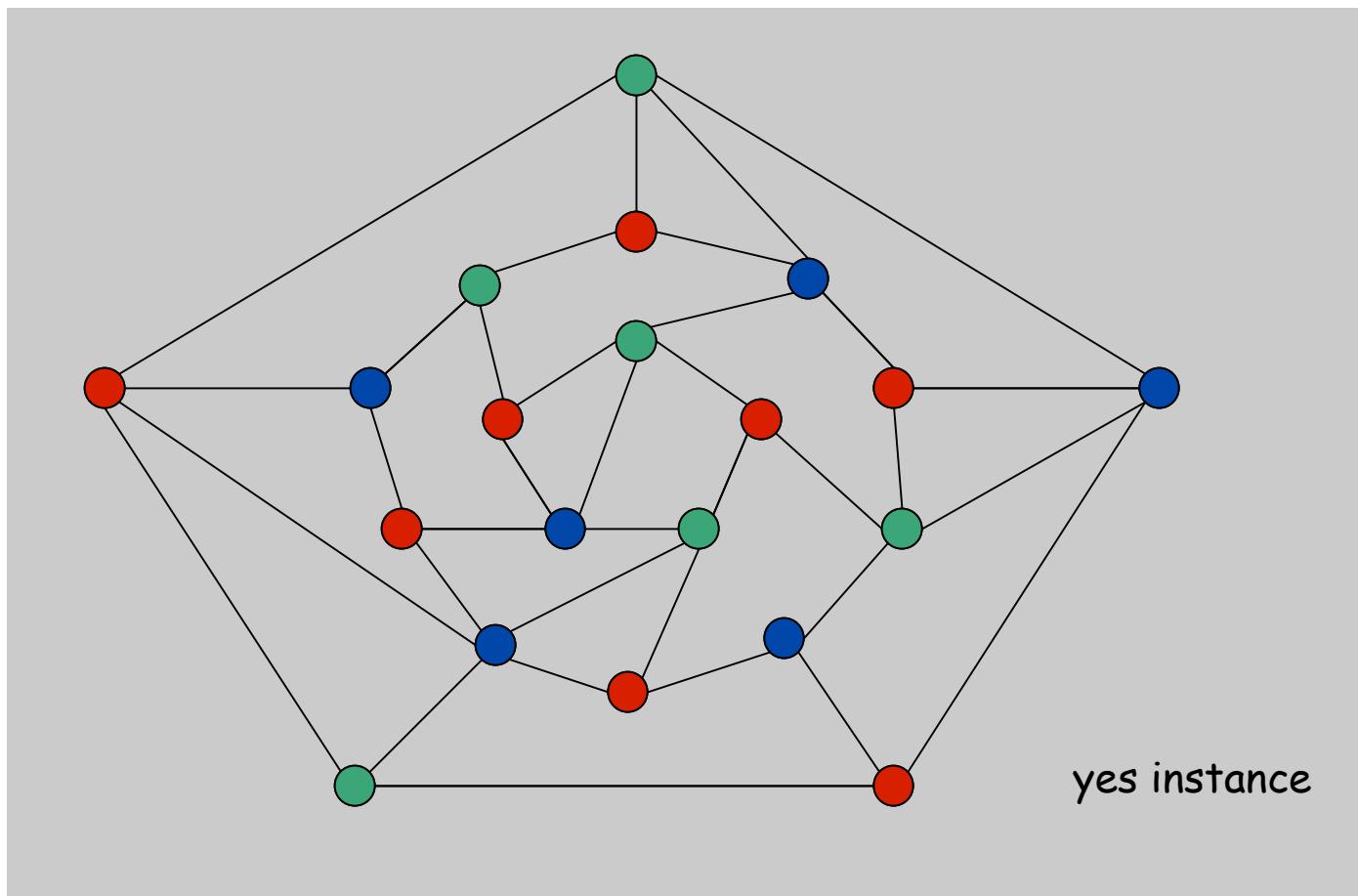
- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

↳ NP-HARD PROBLEM



## 3-Colorability

**3-COLOR:** Given an undirected graph  $G$  does there exist a way to color the nodes red, green, and blue so that no adjacent nodes have the same color?



## Register Allocation

**Register allocation.** Assign program variables to machine register so that no more than  $k$  registers are used and no two program variables that are needed at the same time are assigned to the same register.

**Interference graph.** Nodes are program variables names, edge between  $u$  and  $v$  if there exists an operation where both  $u$  and  $v$  are "live" at the same time.

**Observation.** [Chaitin 1982] Can solve register allocation problem iff interference graph is  $k$ -colorable.

**Fact.**  $\text{3-COLOR} \leq_p \text{k-REGISTER-ALLOCATION}$  for any constant  $k \geq 3$ .

## 3-Colorability

Claim.  $3\text{-SAT} \leq_p 3\text{-COLOR}$ .

Pf. Given 3-SAT instance  $\Phi$ , we construct an instance of 3-COLOR that  
is 3-colorable iff  $\Phi$  is satisfiable.

Construction.

- i. For each literal, create a node.
- ii. Create 3 new nodes T, F, B; connect them in a triangle, and connect each literal to B.
- iii. Connect each literal to its negation.
- iv. For each clause, add gadget of 6 nodes and 13 edges.

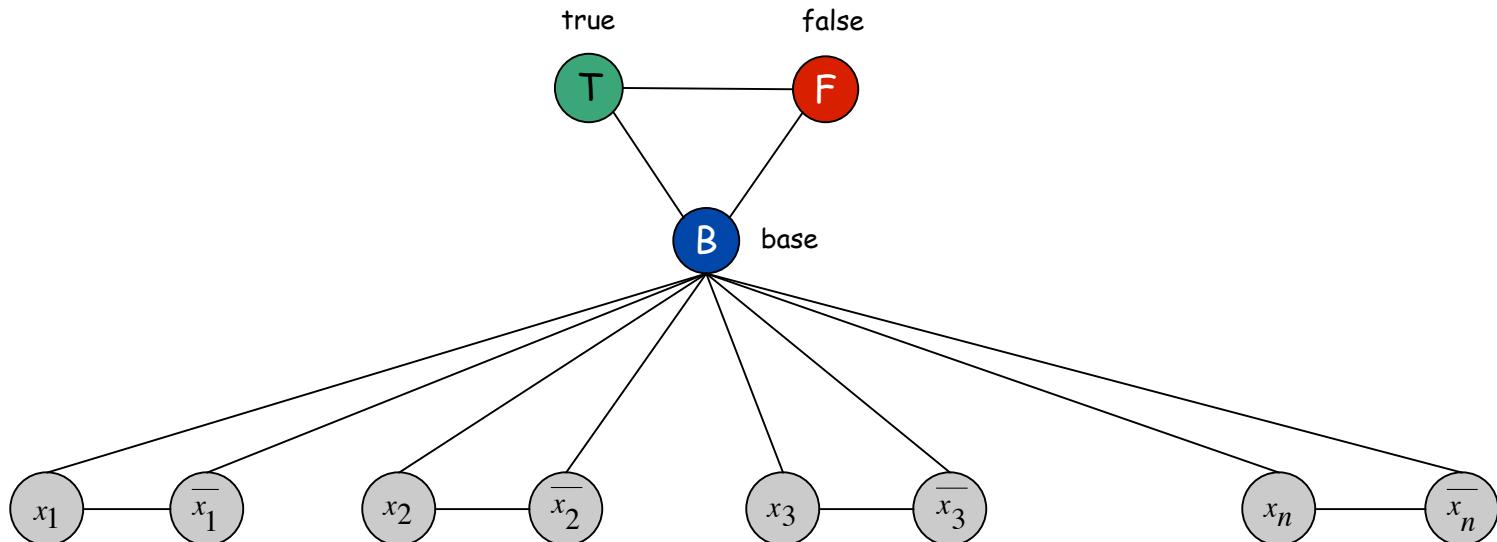
↑  
to be described next

## 3-Colorability

**Claim.** Graph is 3-colorable iff  $\Phi$  is satisfiable.

**Pf.**  $\Rightarrow$  Suppose graph is 3-colorable.

- Consider assignment that sets all T literals to true.
- (ii) ensures each literal is T or F.
- (iii) ensures a literal and its negation are opposites.

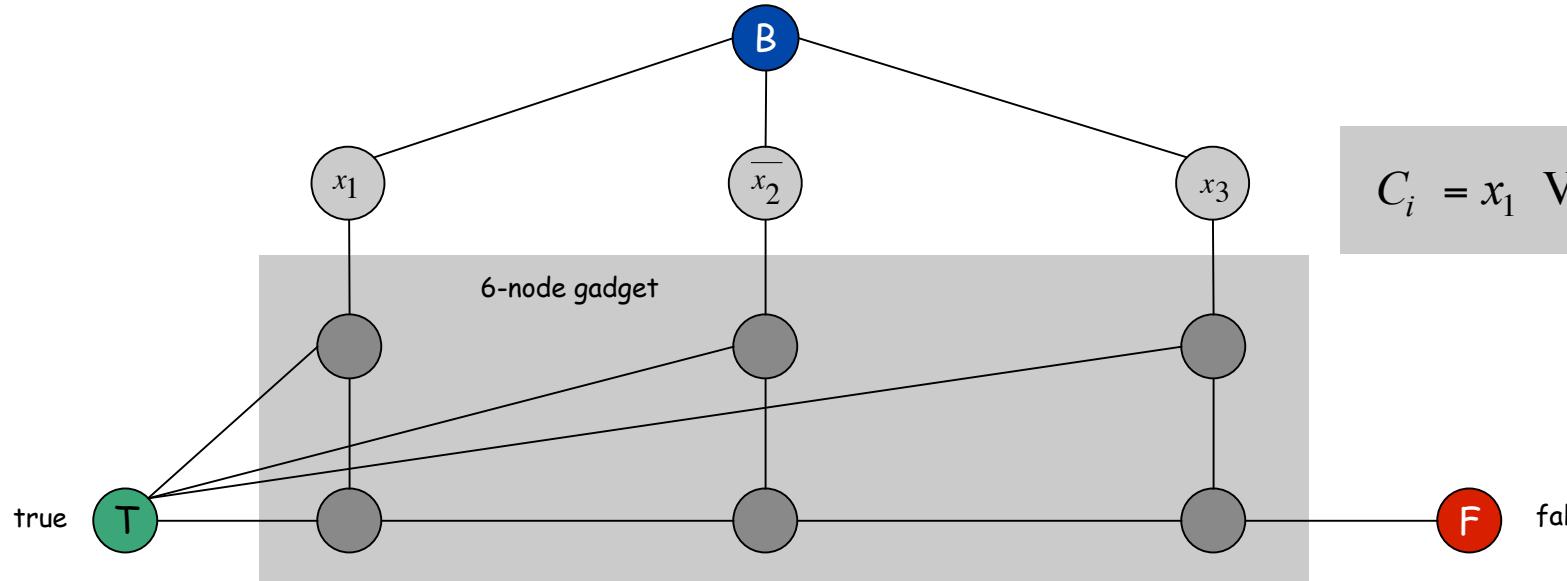


## 3-Colorability

**Claim.** Graph is 3-colorable iff  $\Phi$  is satisfiable.

**Pf.**  $\Rightarrow$  Suppose graph is 3-colorable.

- Consider assignment that sets all T literals to true.
- (ii) ensures each literal is T or F.
- (iii) ensures a literal and its negation are opposites.
- (iv) ensures at least one literal in each clause is T.



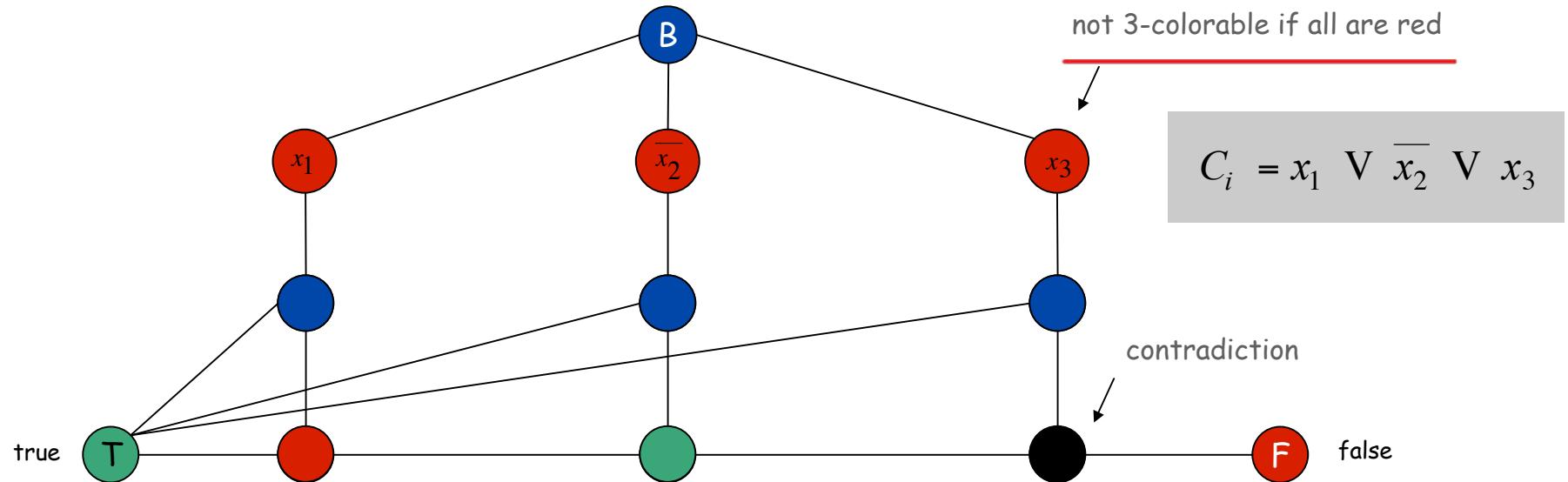
## 3-Colorability

**Claim.** Graph is 3-colorable iff  $\Phi$  is satisfiable.

Pf.  $\Rightarrow$  Suppose graph is 3-colorable.

- Consider assignment that sets all T literals to true.
- (ii) ensures each literal is T or F.
- (iii) ensures a literal and its negation are opposites.
- (iv) ensures at least one literal in each clause is T.

must have a true literal in clause

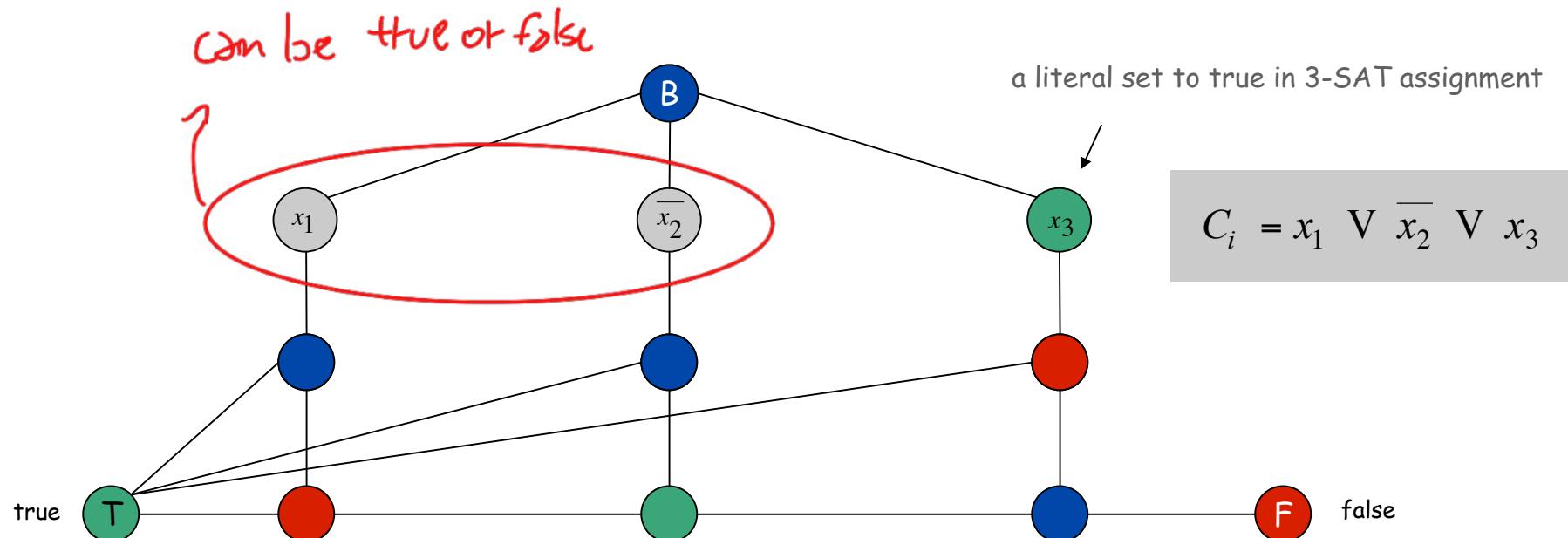


## 3-Colorability

**Claim.** Graph is 3-colorable iff  $\Phi$  is satisfiable.

**Pf.**  $\Leftarrow$  Suppose 3-SAT formula  $\Phi$  is satisfiable.

- Color all true literals T.
- Color node below green node F, and node below that B.
- Color remaining middle row nodes B.
- Color remaining bottom nodes T or F as forced. ▪



## 8.8 Numerical Problems

---

Basic genres.

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3-COLOR, 3D-MATCHING.
- Numerical problems: SUBSET-SUM, KNAPSACK.

## Subset Sum

**SUBSET-SUM.** Given natural numbers  $w_1, \dots, w_n$  and an integer  $W$ , is there a subset that adds up to exactly  $W$ ?

Ex:  $\{1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344\}$ ,  $W = 3754$ .  
Yes.  $1 + 16 + 64 + 256 + 1040 + 1093 + 1284 = 3754$ .

Remark. With arithmetic problems, input integers are encoded in binary. Polynomial reduction must be polynomial in binary encoding.

**Claim.**  $3\text{-SAT} \leq_p \text{SUBSET-SUM}$ .

Pf. Given an instance  $\Phi$  of 3-SAT, we construct an instance of SUBSET-SUM that has solution iff  $\Phi$  is satisfiable.

## Subset Sum

**Construction.** Given 3-SAT instance  $\Phi$  with  $n$  variables and  $k$  clauses, form  $2n + 2k$  decimal integers, each of  $n+k$  digits, as illustrated below.

**Claim.**  $\Phi$  is satisfiable iff there exists a subset that sums to  $W$ .

**Pf.** No carries possible.

$$C_1 = \bar{x} \vee y \vee z$$

$$C_2 = x \vee \bar{y} \vee z$$

$$C_3 = \bar{x} \vee \bar{y} \vee \bar{z}$$

dummies to get clause  
columns to sum to 4

	x	y	z	$C_1$	$C_2$	$C_3$	
x	1	0	0	0	1	0	100,010
$\neg x$	1	0	0	1	0	1	100,101
y	0	1	0	1	0	0	10,100
$\neg y$	0	1	0	0	1	1	10,011
z	0	0	1	1	1	0	1,110
$\neg z$	0	0	1	0	0	1	1,001
	0	0	0	1	0	0	100
	0	0	0	2	0	0	200
	0	0	0	0	1	0	10
	0	0	0	0	2	0	20
	0	0	0	0	0	1	1
	0	0	0	0	0	2	2
W	1	1	1	4	4	4	111,444

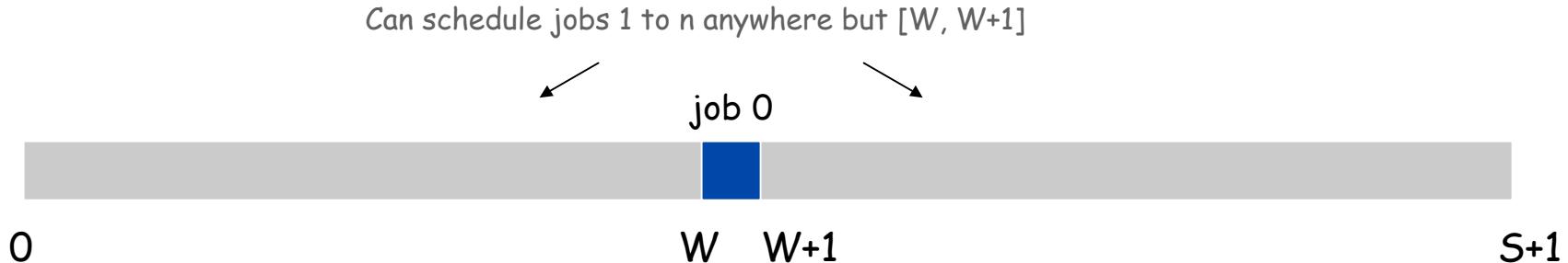
## Scheduling With Release Times

**SCHEDULE-RELEASE-TIMES.** Given a set of  $n$  jobs with processing time  $t_i$ , release time  $r_i$ , and deadline  $d_i$ , is it possible to schedule all jobs on a single machine such that job  $i$  is processed with a contiguous slot of  $t_i$  time units in the interval  $[r_i, d_i]$ ?

**Claim.**  $\text{SUBSET-SUM} \leq_p \text{SCHEDULE-RELEASE-TIMES}$ .

Pf. Given an instance of  $\text{SUBSET-SUM } w_1, \dots, w_n$ , and target  $W$ ,

- Create  $n$  jobs with processing time  $t_i = w_i$ , release time  $r_i = 0$ , and no deadline ( $d_i = 1 + \sum_j w_j$ ).
- Create job 0 with  $t_0 = 1$ , release time  $r_0 = W$ , and deadline  $d_0 = W+1$ .

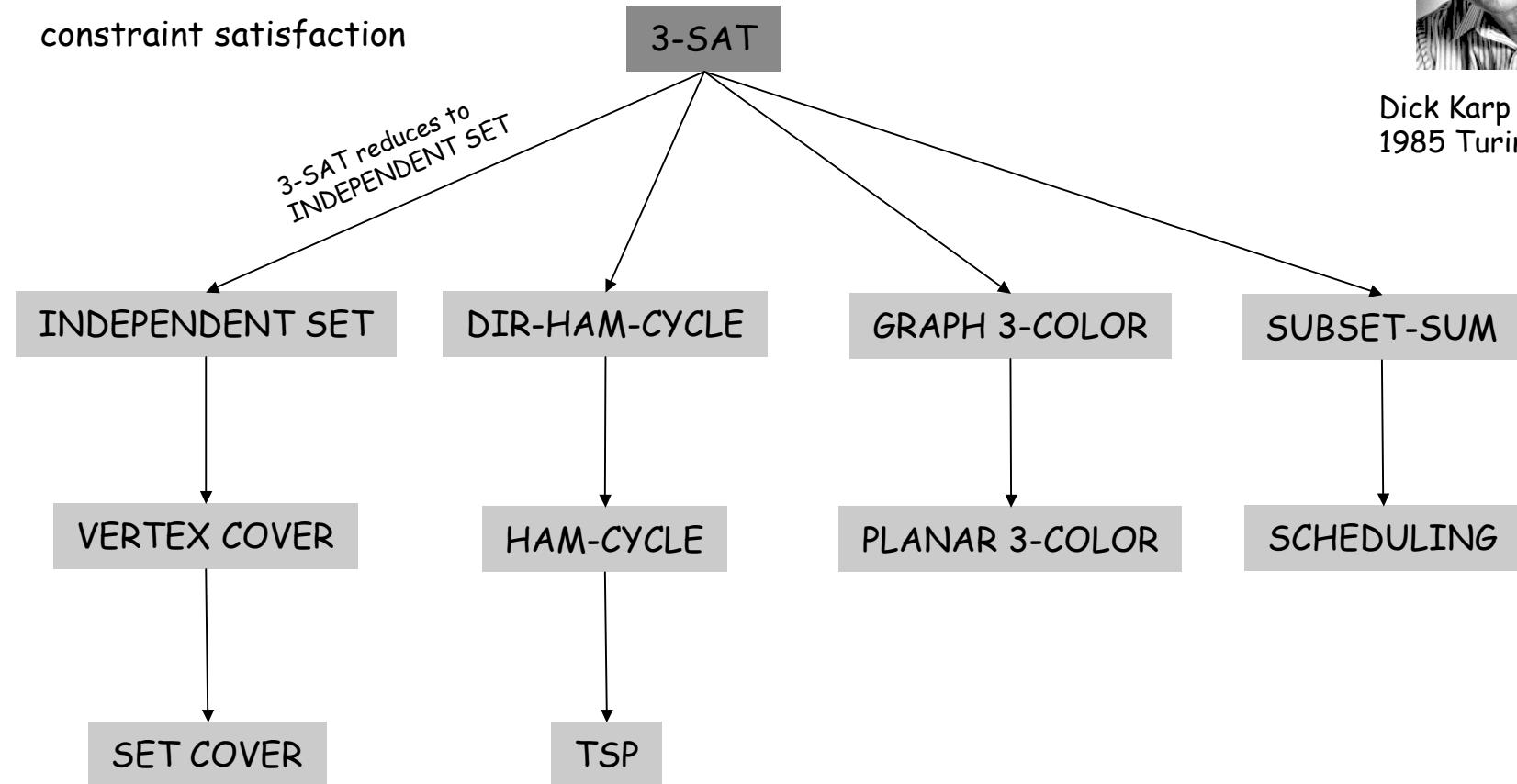


## 8.10 A Partial Taxonomy of Hard Problems

---

# Polynomial-Time Reductions

constraint satisfaction



Dick Karp (1972)  
1985 Turing Award

packing and covering

sequencing

partitioning

numerical

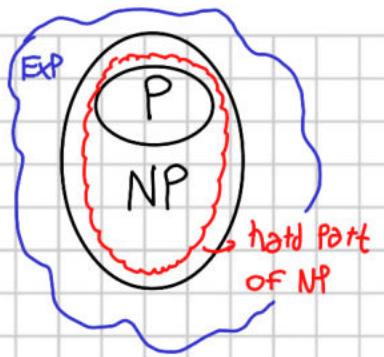
## Reductions

P      NP

NP-Completeness

Problem  $\in$  NP

Problem is NP-Hard



Given problem B prove that is NPC:

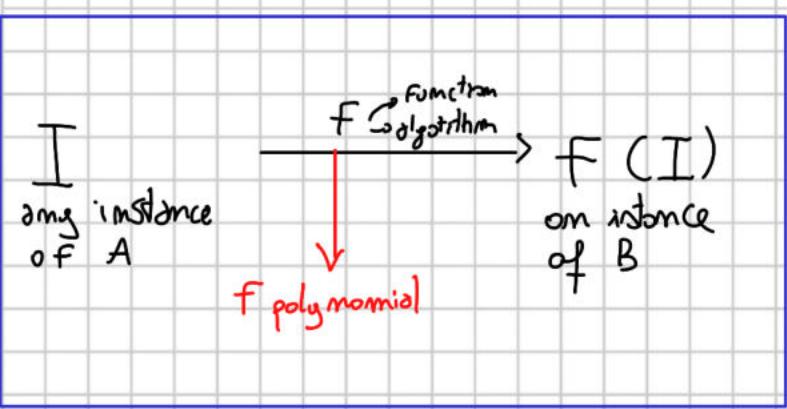
Consider A that is NPC, want to

$A \rightarrow B$

reduce A to B

↓  
out problem

difficult



STEP 2

- S solution of  $f(I) \xrightarrow{h}$  Solution  $h(s)$  of I
- No solution to  $f(I) \Rightarrow \nexists$  solution for I

# Exercise 1 HW-E4 - 2021/2022

The grinch is planning to make all the city's kids sick the day before christmas, by overfeeding them with sweets. Each kid has a set of sweets they like, and the grinch would like to give them all of those. He needs to give each kid some bags of sweets from Santa's supply that cover their needs, but also wants to minimize the number  $k$  of times he has to steal the most popular bag: if more than  $k$  of the same kind are missing, Santa might notice... More formally, from a given set  $S$  of sweets, bag types  $B_1, B_2, \dots, B_n \subseteq S$  and preferred sets of sweets  $K_1, K_2, \dots, K_m$ , you need to find a set of bags  $\hat{B}(j) \subseteq \{B_1, B_2, \dots, B_n\}$  for each kid  $i$  such that:

$$K_i \subseteq \bigcup_{B \in \hat{B}(j)} B.$$

Moreover, for all bags  $B_i$ , you need to make sure that

$$|\{\hat{B}(j) | B_i \in \hat{B}(j), j \in \{1, \dots, m\}\}| \leq k.$$

Planning this is taking the grinch a long time. Prove that indeed, the problem is NP-complete.

**Hint:** For your reduction, use a version of SAT where every clause has either all positive or all negative literals, which is an NP-Complete problem (i.e., all clauses are of the form  $(x_1 \vee x_2 \vee \dots)$ , or  $(\bar{x}_1 \vee \bar{x}_2 \vee \dots)$ ).

- Each kid has a set of sweets they like
- $S = \{S_1, S_2, \dots, S_k\}$  set of sweets
- Each Kid  $i \in \{1, \dots, m\}$  has the set of sweets they want  $K_i$
- Santa has a set of bags  $\{B_1, \dots, B_m\} \subseteq S$

The goal is to cover the demand of every Kid.  
We need to find a set of Bags:  $\hat{B}(i) \subseteq \{B_1, \dots, B_m\}$   
s.t.  $K_i \subseteq \bigcup_{B \in \hat{B}(i)} B$  (union is a superset of the original demand of the kid  $i$ )

**constraint:** Each Bag  $B_i$  cannot be used more than  $K$  times!! ( $K$  is given)

Demonstrate that this problem is NP-complete, using the SAT:  $(x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4)$  that is NPC.

$\downarrow$        $\downarrow$   
**all positive**      **all negative**

$K = \{1, \dots, m\}$  Kids

$S = \{s_1, \dots, s_k\}$  Sweets

Bags  $B_1, \dots, B_m \subseteq S$

$K_i$  demands set of kid  $i$

$\hat{B}(i)$  set of bags cover  $K_i$

$K_i \subseteq \bigcup_{B \in \hat{B}(i)} B$  + constraint  $K$

## Transformation:

1. For each clauses  $C_i$  we create a sweet  $s_i$ .

2. For each  $X_i$  we create a bag  $B_i$

↗ set of bags they want

instance: We have 2 kids  $1, 2 \rightarrow K_1, K_2$

$K_1$  contain the sweets that correspond to the positive clauses.

$K_2$  contain the sweets that correspond to the negative clauses.

$K = 1$  constraint!

Example:  $(\overbrace{x_1 \vee x_2}^{s_1}) \wedge (\overbrace{x_1 \vee x_3}^{s_2}) \wedge (\overbrace{\neg x_1 \vee \neg x_4}^{s_3})$

$K_1 = \{s_1, s_2\}$   $K_2 = \{s_3\}$

truthfull assignment  
 $x_1=T, x_4=F$   
↗ that sat:isfy these SAT

Bag  $i$  what contains?  $B_i$  contains  $s_j$  iff  $X_i \in C_j$

$B_1 = \{s_1, s_2, s_3\}$ ,  $B_2 = \{s_1\}$ ,  $B_3 = \{s_2\}$ ,  $B_4 = \{s_3\}$

if SAT have no solution  $\rightarrow$  our problem have no solution:  $\Rightarrow x_1=T, x_4=F$

$\overbrace{x_1}^{s_1} \wedge \overbrace{\neg x_1}^{s_2}$   $B_1 = \{s_1, s_2\}$

$K_1 = s_1$  and  $K_2 = s_2$

We can give the bag  $B_1$  only to one of the kid for the constraint,  
we have no solution

Final part: how do we match the assignment between SAT and Grinch?

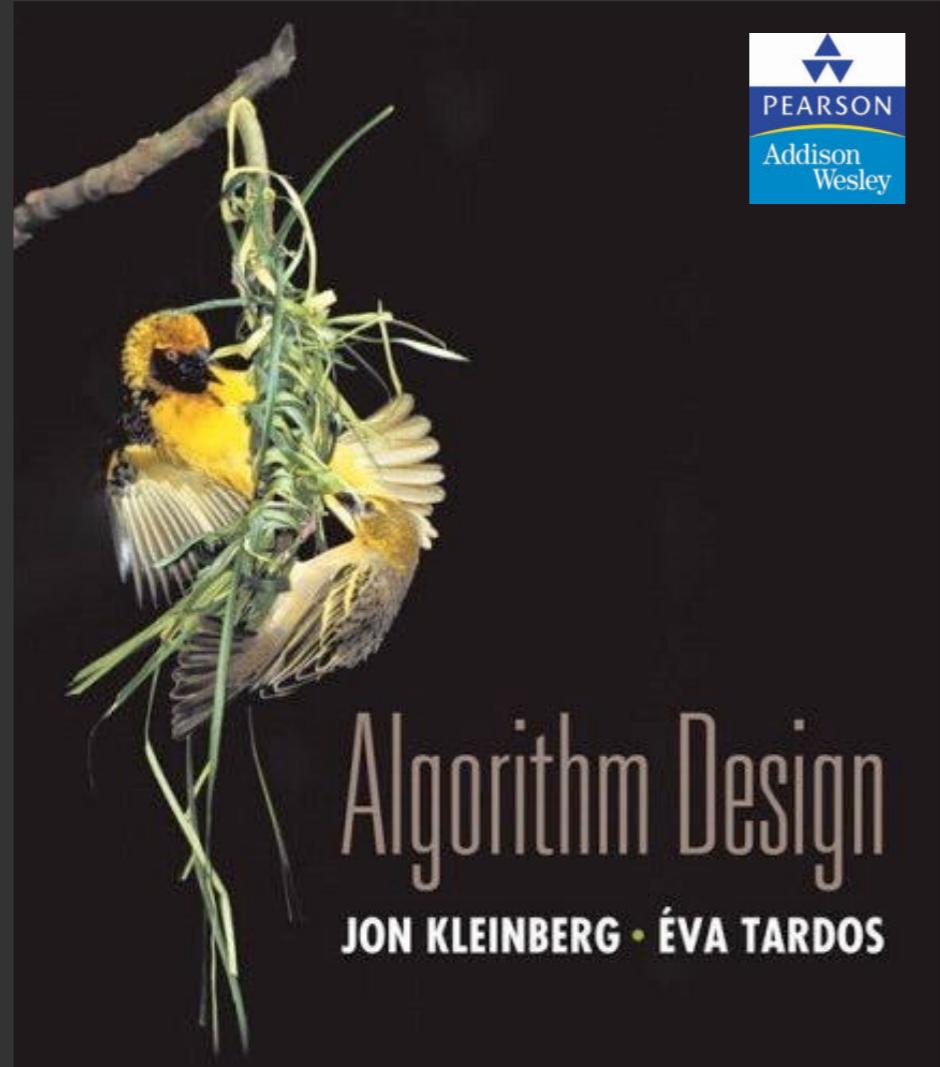
if  $B_1 \rightsquigarrow K_1 \Rightarrow X_1 \rightsquigarrow T$

if  $B_1 \rightsquigarrow K_2 \Rightarrow X_1 \rightsquigarrow F$

if G has a solution  $\Rightarrow$  SAT has a solution

$\exists$  assignment of bags that cover  $K_1, K_2$  and respect the constraint  $K=1$ .

$\Rightarrow$  We can set  $X_i$  to T or F and it is easy to check that this solution of the instance.



## 11. APPROXIMATION ALGORITHMS

---

- ▶ *load balancing*
- ▶ *center selection*
- ▶ *pricing method: vertex cover*
- ▶ *LP rounding: vertex cover*
- ▶ *generalized load balancing*
- ▶ *knapsack problem*

Lecture slides by Kevin Wayne

Copyright © 2005 Pearson–Addison Wesley

<http://www.cs.princeton.edu/~wayne/kleinberg-tardos>

# Coping with NP-completeness

Fot an optimization problem

Q. Suppose I need to solve an NP-hard problem. What should I do?

A. Sacrifice one of three desired features.

i. Solve arbitrary instances of the problem.

ii. Solve problem to optimality. → will not guarantee  $\rho$  time the optimal solution

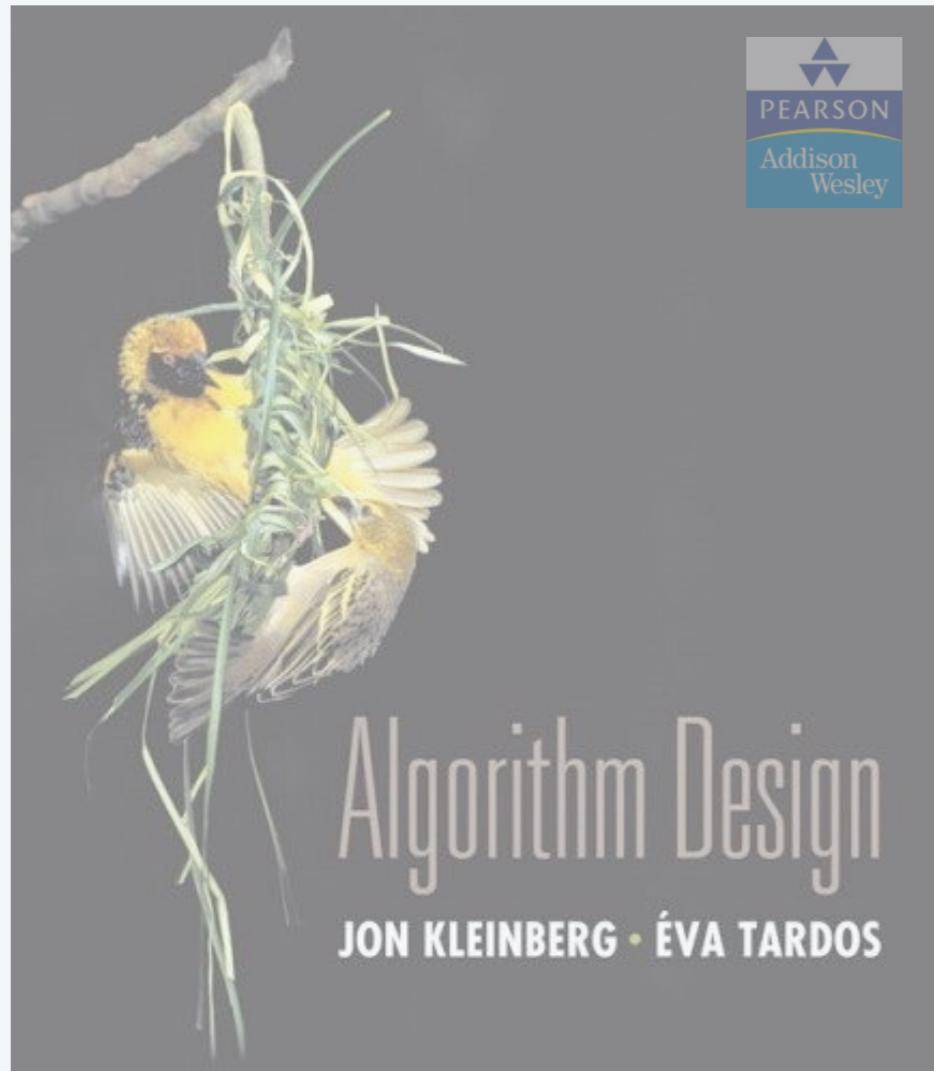
iii. Solve problem in polynomial time.

$\rho$ -approximation algorithm. ↗ not depend from the instance

- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instance of the problem
- Guaranteed to find solution within ratio  $\rho$  of true optimum.

↪ never worst ratio  $\rho$ , maybe better

Challenge. Need to prove a solution's value is close to optimum,  
without even knowing what optimum value is



## 11. APPROXIMATION ALGORITHMS

---

- ▶ *load balancing*
- ▶ *center selection*
- ▶ *pricing method: vertex cover*
- ▶ *LP rounding: vertex cover*
- ▶ *generalized load balancing*
- ▶ *knapsack problem*

## Load balancing

want finish earliest possible

**Input.**  $m$  identical machines;  $n$  jobs, job  $j$  has processing time  $t_j$ .

- Job  $j$  must run contiguously on one machine.
- A machine can process at most one job at a time.

**Def.** Let  $J(i)$  be the subset of jobs assigned to machine  $i$ .

The load of machine  $i$  is  $L_i = \sum_{j \in J(i)} t_j$ .

**Def.** The makespan is the maximum load on any machine  $L = \max_i L_i$ .

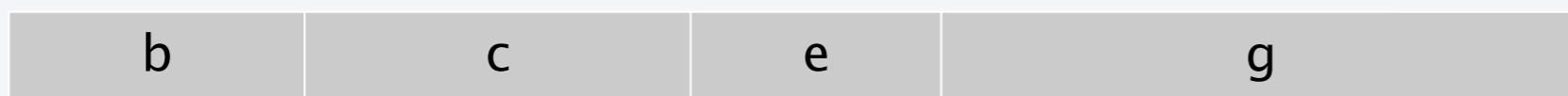
**Load balancing.** Assign each job to a machine to minimize makespan.

optimization problem

machine 1



machine 2



optimal  
solution



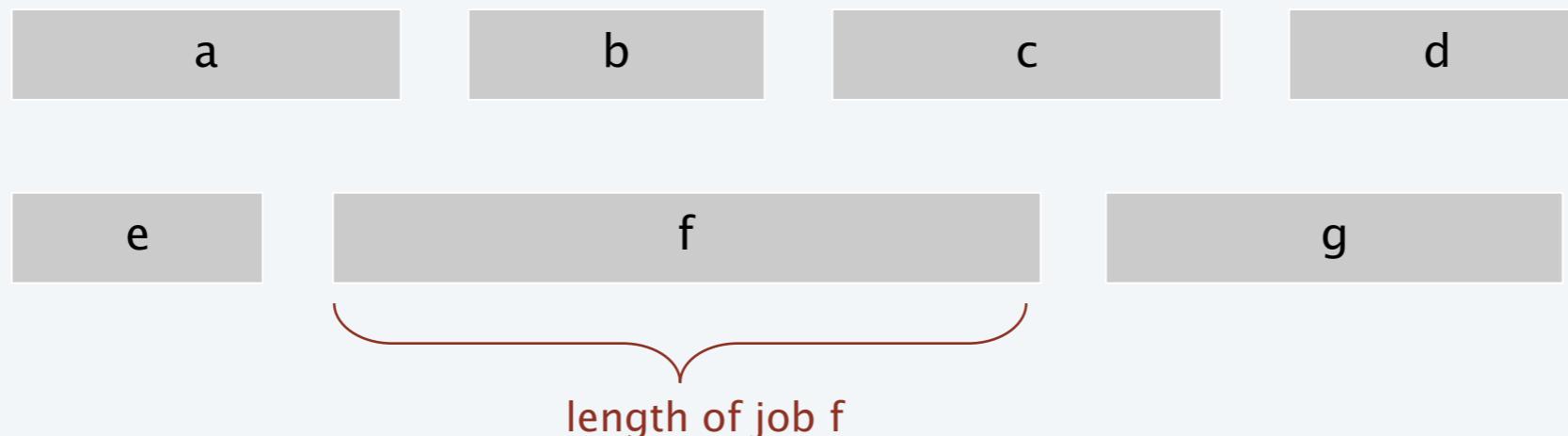
# Load balancing on 2 machines is NP-hard

**Claim.** Load balancing is hard even if only 2 machines.

**Pf.**  $\text{NUMBER-PARTITIONING} \leq_p \text{LOAD-BALANCE}$ .

divide set of number  
in two subset s.t.  
they have exact same value

NP-complete by Exercise 8.26



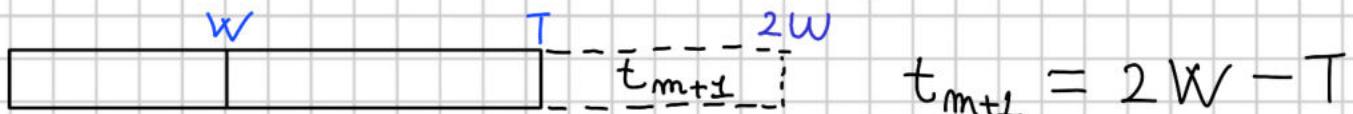
yes



$t_1, \dots, t_m$  want to find subset of the value that sum up to  $W$

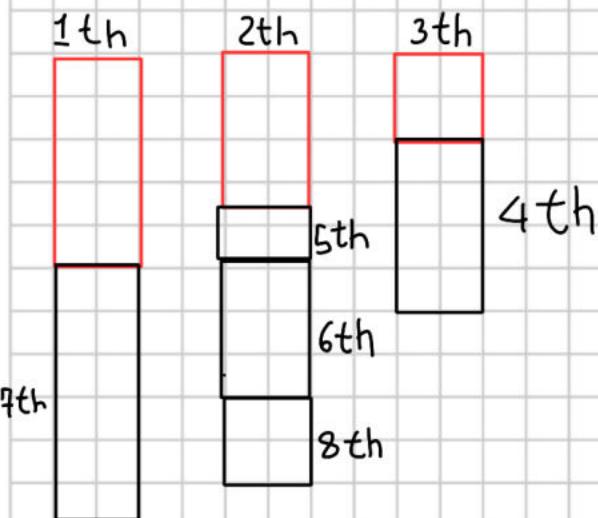
$T = \sum_j t_j$  is the sum of this subset values

If  $W \geq T/2$  we study a different problem, finding a subset of values that sum up to  $T - W$



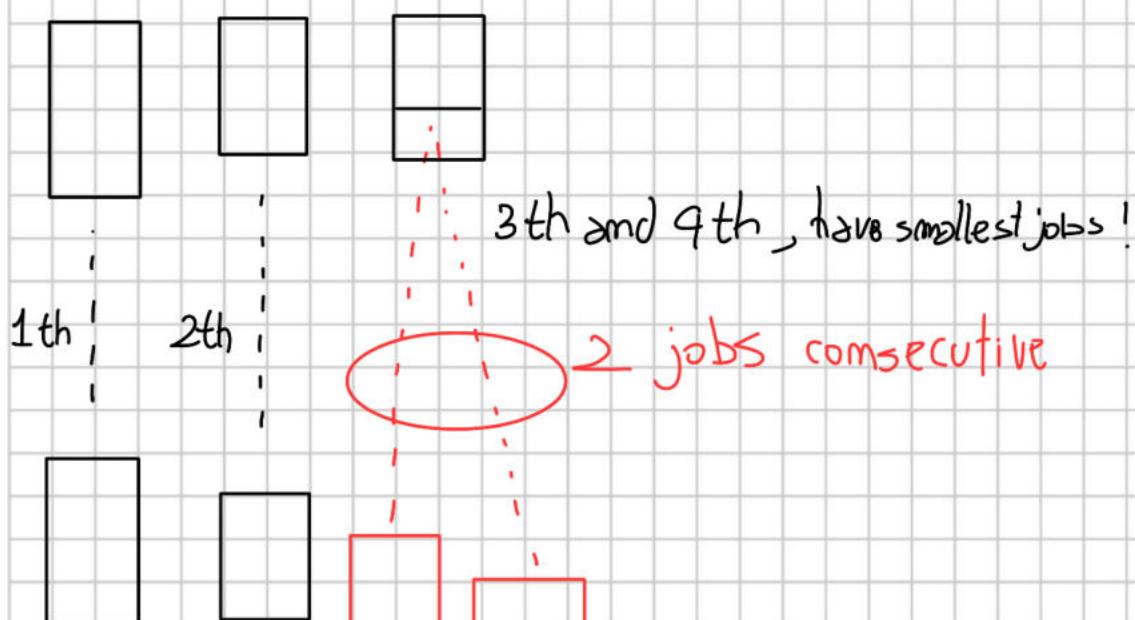
this is a reduction to subset sum to number partitioning

Example: 3 machines (algorithm in next slide →)



- Order to insert the jobs in the machines

Example of  $\frac{3}{2}L^*$  algorithm



# Load balancing: list scheduling

## List-scheduling algorithm.

- Consider  $n$  jobs in some fixed order.
- Assign job  $j$  to machine whose load is smallest so far.



```
List-Scheduling(m, n, t1, t2, ..., tn) {
    for i = 1 to m {
        Li ← 0           ← load on machine i
        J(i) ← ∅          ← jobs assigned to machine i
    }

    for j = 1 to n {
        i = argmink Lk      ← machine i has smallest load
        J(i) ← J(i) ∪ {j}       ← assign job j to machine i
        Li ← Li + tj       ← update load of machine i
    }
    return J(1), ..., J(m)
}
```

initialize load and jobs lists

Implementation.  $O(n \log m)$  using a priority queue.

## Load balancing: list scheduling analysis

→ never worst of 2 times  
of optimal solution

**Theorem.** [Graham 1966] Greedy algorithm is a 2-approximation.

- First worst-case analysis of an approximation algorithm.
- Need to compare resulting solution with optimal makespan  $L^*$ .

two lower bound:

**Lemma 1.** The optimal makespan  $L^* \geq \max_j t_j$ .

$L^* \geq$  biggest processing time of  
> job.  
(must be scheduled)

**Pf.** Some machine must process the most time-consuming job. ▀

**Lemma 2.** The optimal makespan  $L^* \geq \frac{1}{m} \sum_j t_j$ . average load at  $t_j$

**Pf.**

- The total processing time is  $\sum_j t_j$ .  
↳ never be smaller of average processing time.
- One of  $m$  machines must do at least a  $1/m$  fraction of total work. ▀

# Load balancing: list scheduling analysis

**Theorem.** Greedy algorithm is a 2-approximation.

Pf. Consider load  $L_i$  of bottleneck machine  $i$ .

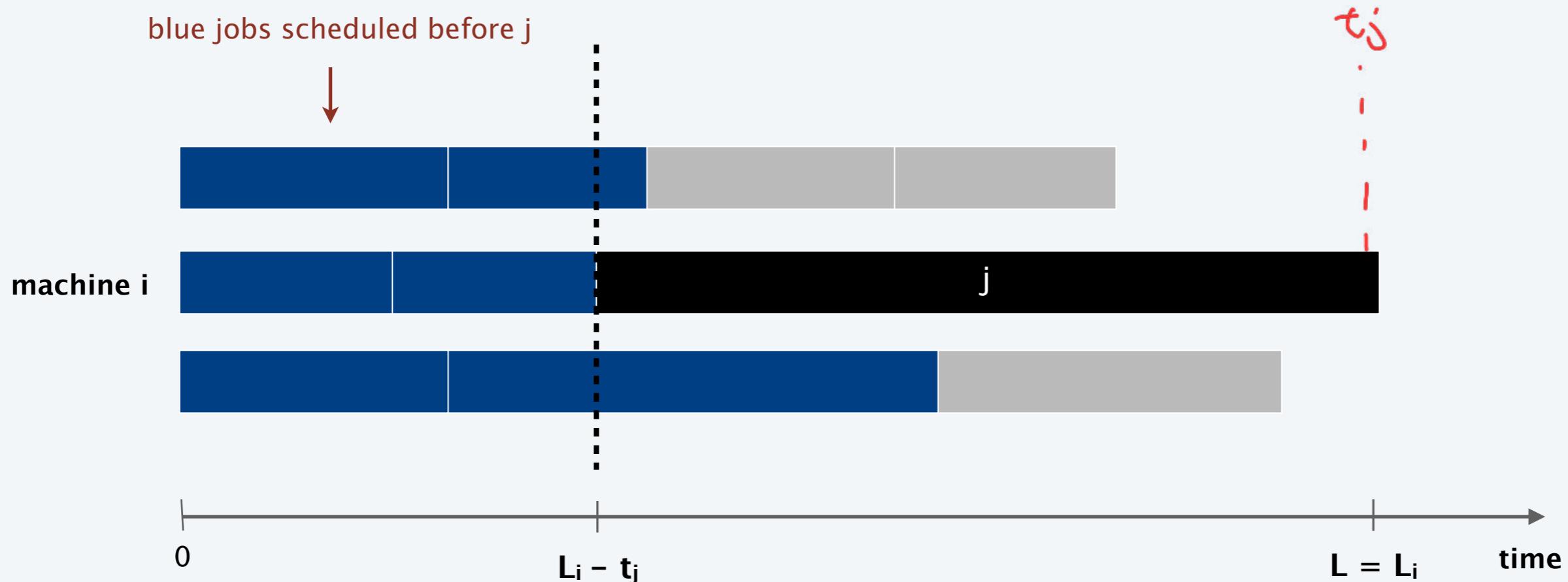
- Let  $j$  be last job scheduled on machine  $i$ .

- When job  $j$  assigned to machine  $i$ ,  $i$  had smallest load.

Its load before assignment is  $L_i - t_j \Rightarrow L_i - t_j \leq L_k$  for all  $1 \leq k \leq m$ .

given by algorithm

↳ smallest load at this time,  
smallest of average load !



# Load balancing: list scheduling analysis

**Theorem.** Greedy algorithm is a 2-approximation.

Pf. Consider load  $L_i$  of bottleneck machine  $i$ .

- Let  $j$  be last job scheduled on machine  $i$ .
- When job  $j$  assigned to machine  $i$ ,  $i$  had smallest load.  
Its load before assignment is  $L_i - t_j \Rightarrow L_i - t_j \leq L_k$  for all  $1 \leq k \leq m$ .
- Sum inequalities over all  $k$  and divide by  $m$ :

because  $i$  was the last overload machine  $\leftarrow$

$$\begin{aligned} L_i - t_j &\leq \frac{1}{m} \sum_k L_k \text{ (the average load)} \\ &= \frac{1}{m} \sum_k t_k \\ \text{Lemma 2} \rightarrow &\leq L^* \text{ (optimal load)} \end{aligned}$$

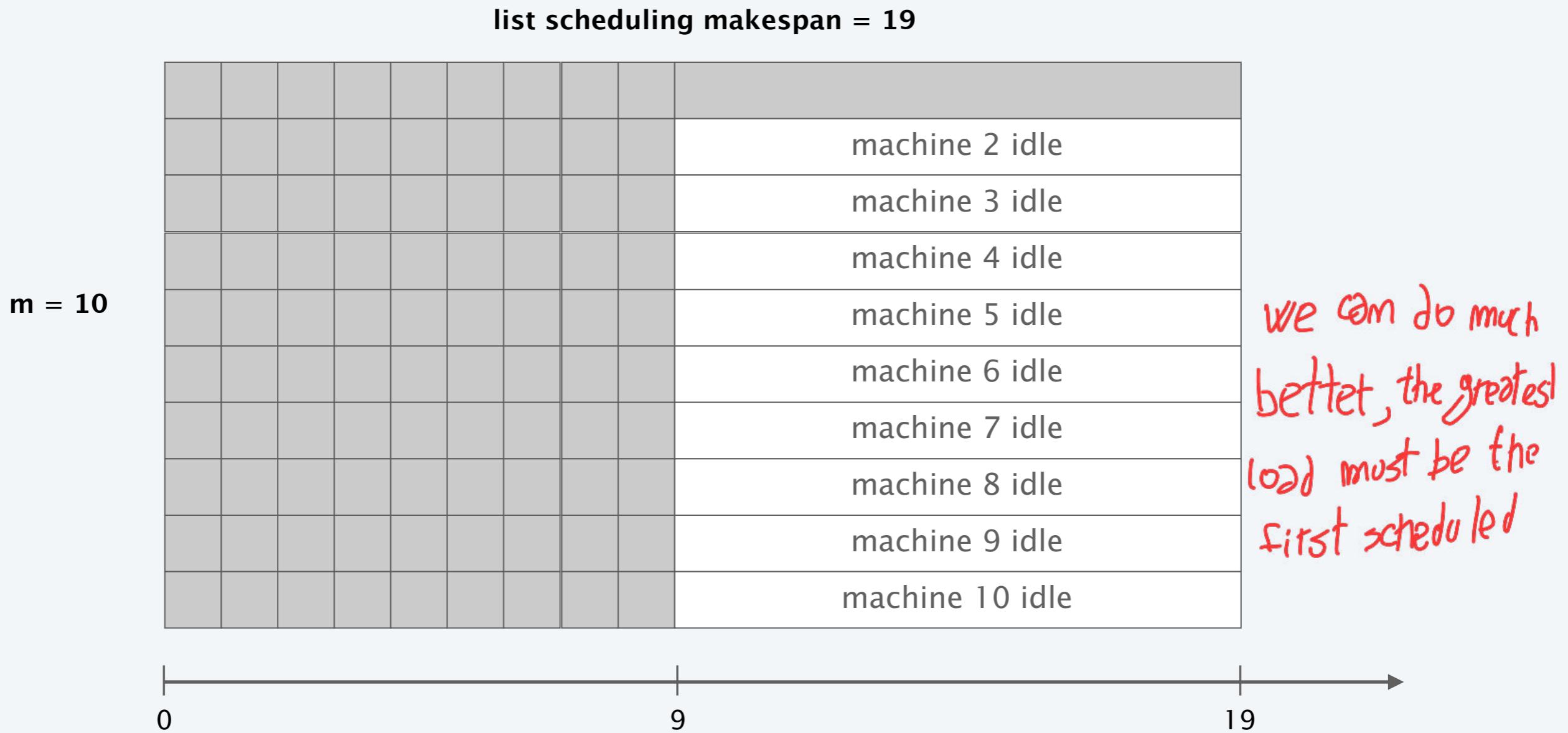
- Now  $L_i = \underbrace{(L_i - t_j)}_{\leq L^*} + \underbrace{t_j}_{\leq L^*} \leq 2L^*$ . ■  
 $\uparrow$   
Lemma 1
- $\hookrightarrow$  2 times optimal load

# Load balancing: list scheduling analysis

Q. Is our analysis tight?

A. Essentially yes.

Ex:  $m$  machines,  $m(m - 1)$  jobs length 1 jobs, one job of length  $m$ .



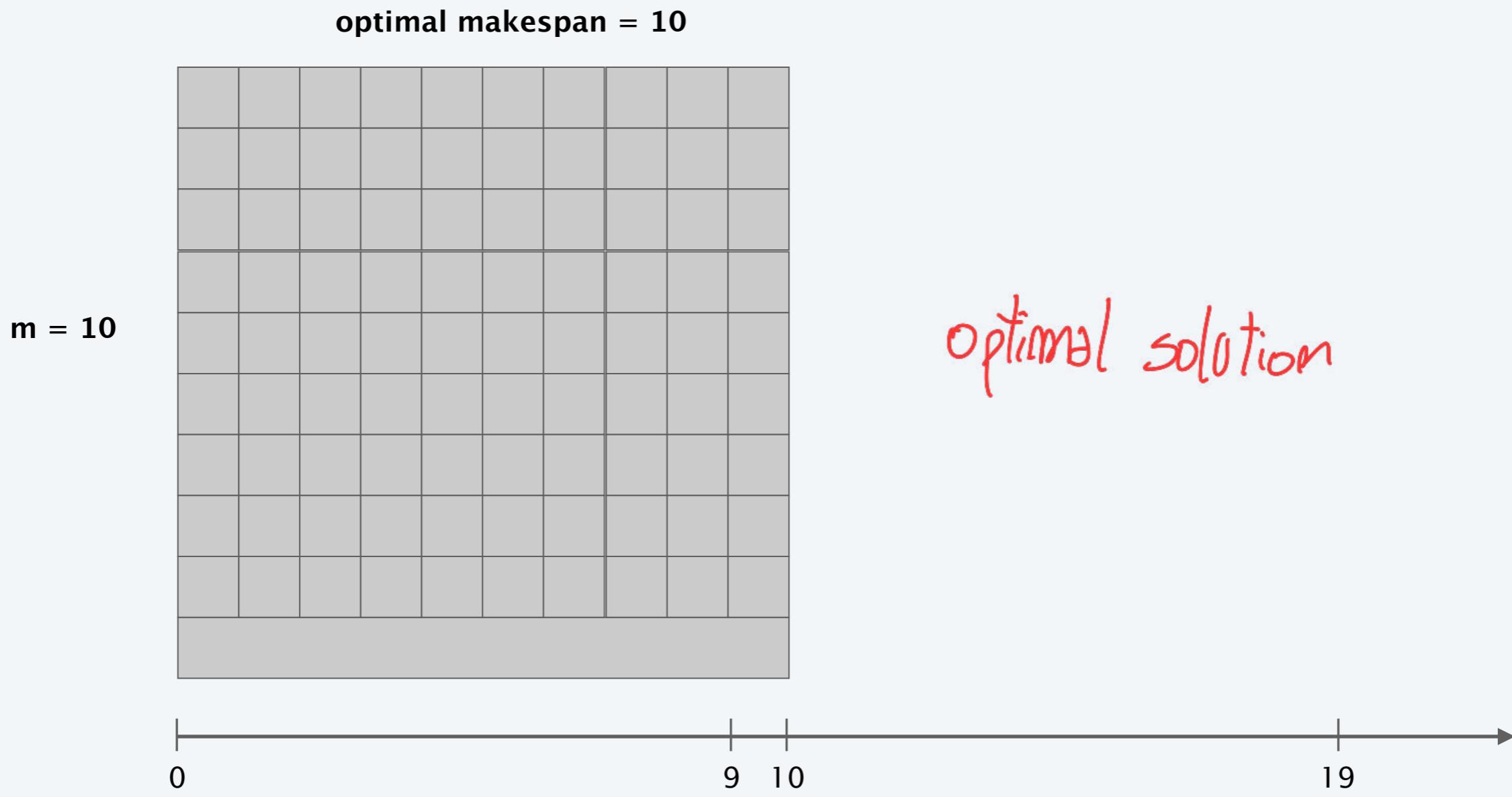
# Load balancing: list scheduling analysis

---

Q. Is our analysis tight?

A. Essentially yes.

Ex:  $m$  machines,  $m(m - 1)$  jobs length 1 jobs, one job of length  $m$ .



## Load balancing: LPT rule *want long job at start.*

Longest processing time (LPT). Sort  $n$  jobs in descending order of processing time, and then run list scheduling algorithm.

```
LPT-List-Scheduling(m, n, t1, t2, ..., tn) {  
    Sort jobs so that t1 ≥ t2 ≥ ... ≥ tn  
    ↪ Only difference from before, assign jobs in increasing processing time  
    for i = 1 to m {  
        Li ← 0           ← load on machine i  
        J(i) ← ∅          ← jobs assigned to machine i  
    }  
  
    for j = 1 to n {  
        i = argmink Lk      ← machine i has smallest load  
        J(i) ← J(i) ∪ {j}       ← assign job j to machine i  
        Li ← Li + tj       ← update load of machine i  
    }  
    return J(1), ..., J(m)  
}
```

## Load balancing: LPT rule

**Observation.** If at most  $m$  jobs, then list-scheduling is optimal.

Pf. Each job put on its own machine. ■  $m$  jobs -  $m$  machines we have optimum

**Lemma 3.** If there are more than  $m$  jobs,  $L^* \geq 2t_{m+1}$ .

Pf.

- Consider first  $m+1$  jobs  $t_1, \dots, t_{m+1}$ .
- Since the  $t_i$ 's are in descending order, each takes at least  $t_{m+1}$  time.
- There are  $m+1$  jobs and  $m$  machines, so by pigeonhole principle, at least one machine gets two jobs. ■

↳ if we have assigned  $m$  jobs, the  $(m+1)$ th job is assigned to last machine in queue!

↓  
at least one  
have ≥ jobs

**Theorem.** LPT rule is a  $3/2$ -approximation algorithm.

Pf. Same basic approach as for list scheduling.

$$L_i = \underbrace{(L_i - t_j)}_{\leq L^*} + \underbrace{t_j}_{\leq \frac{1}{2}L^*} \leq \frac{3}{2}L^*. \blacksquare$$



Lemma 3

(by observation, can assume number of jobs > m)

## Load Balancing: LPT rule

---

Q. Is our  $3/2$  analysis tight?

A. No.

best until now

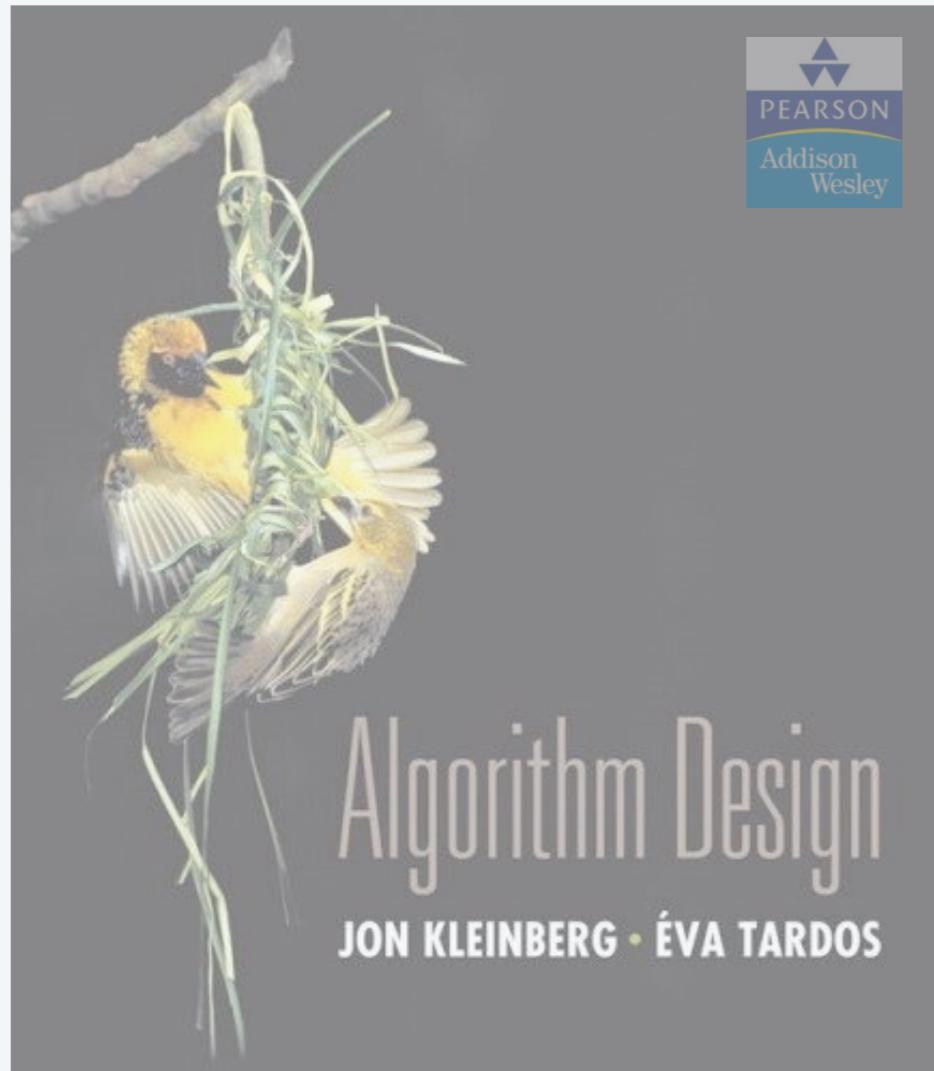
Theorem. [Graham 1969] LPT rule is a  $4/3$ -approximation.

Pf. More sophisticated analysis of same algorithm.

Q. Is Graham's  $4/3$  analysis tight?

A. Essentially yes.

Ex:  $m$  machines,  $n = 2m + 1$  jobs, 2 jobs of length  $m, m+1, \dots, 2m-1$  and one more job of length  $m$ .



## 11. APPROXIMATION ALGORITHMS

---

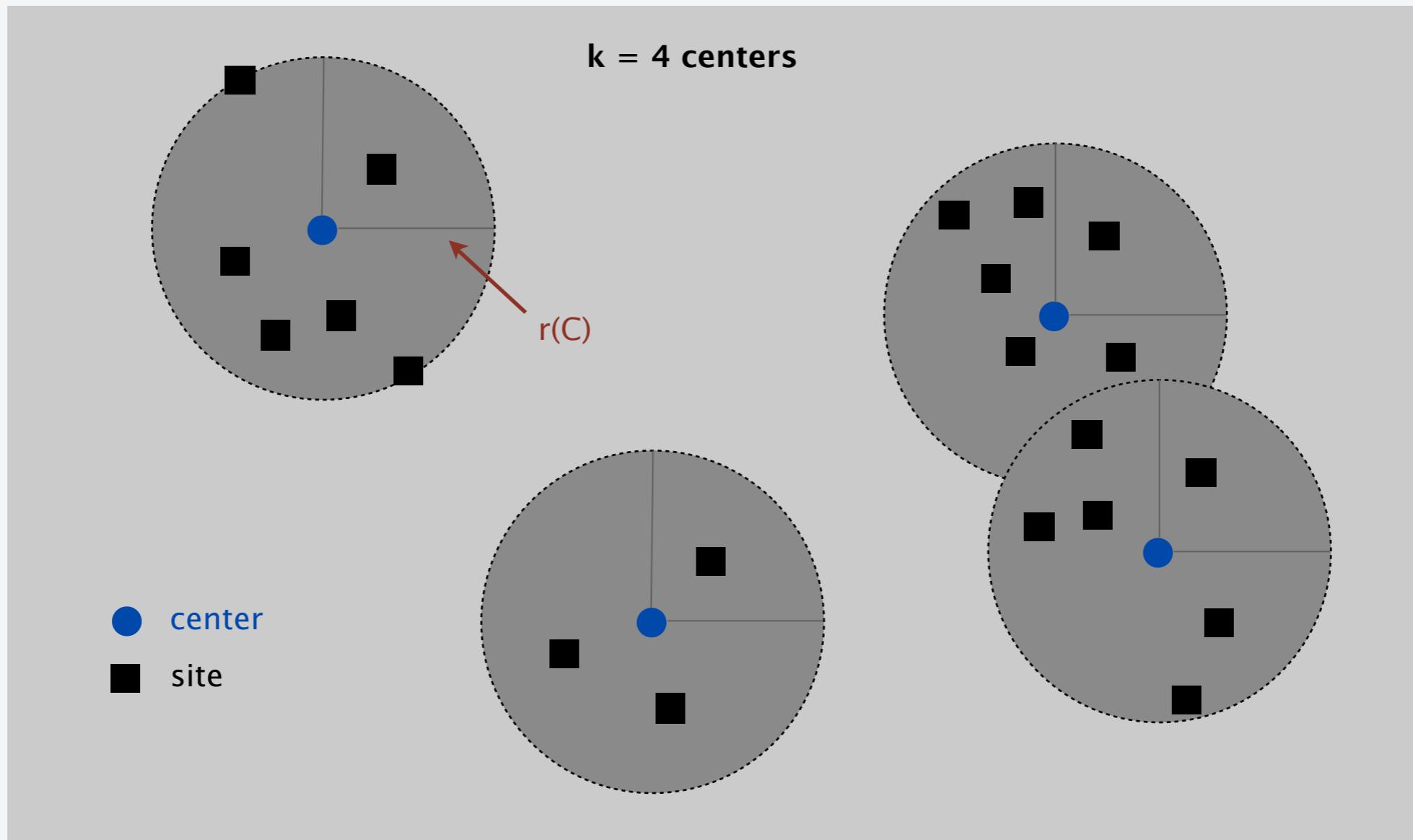
- ▶ *load balancing*
- ▶ ***center selection***
- ▶ *pricing method: vertex cover*
- ▶ *LP rounding: vertex cover*
- ▶ *generalized load balancing*
- ▶ *knapsack problem*

## Center selection problem

Problem of clustering point in a metric space. Want  $k$  centers, and each point is associated to closest one

Input. Set of  $n$  sites  $s_1, \dots, s_n$  and an integer  $k > 0$ .

Center selection problem. Select set of  $k$  centers  $C$  so that maximum distance  $r(C)$  from a site to nearest center is minimized.



# Center selection problem

---

**Input.** Set of  $n$  sites  $s_1, \dots, s_n$  and an integer  $k > 0$ .

**Center selection problem.** Select set of  $k$  centers  $C$  so that maximum distance  $r(C)$  from a site to nearest center is minimized.

## Notation.

- $dist(x, y) = \text{distance between sites } x \text{ and } y$
- $dist(s_i, C) = \min_{c \in C} dist(s_i, c) = \text{distance from } s_i \text{ to closest center}$
- $r(C) = \max_i dist(s_i, C) = \text{smallest covering radius}$

**Goal.** Find set of centers  $C$  that minimizes  $r(C)$ , subject to  $|C| = k$ .

## Distance function properties.

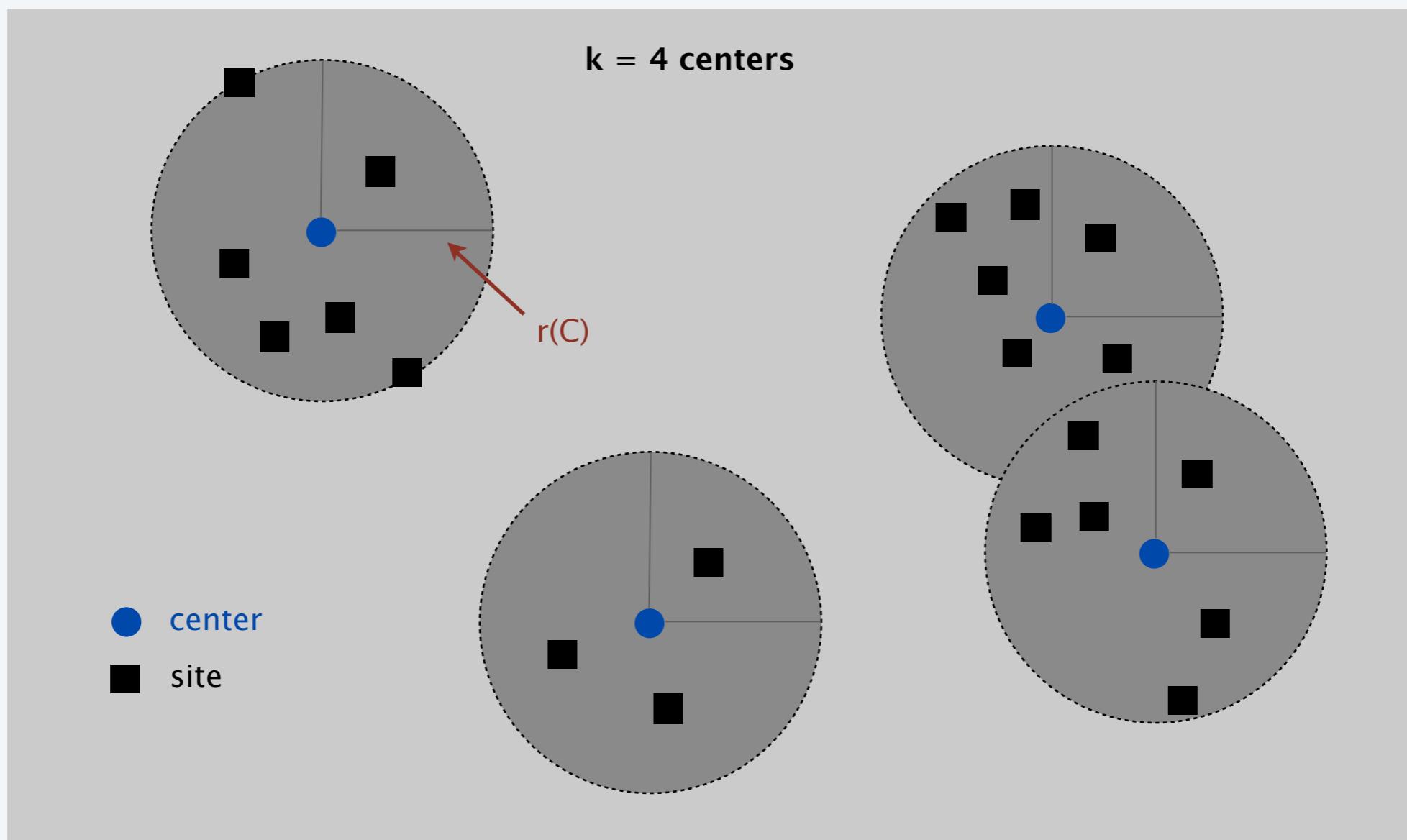
- $dist(x, x) = 0$  [ identity ]
- $dist(x, y) = dist(y, x)$  [ symmetry ]
- $dist(x, y) \leq dist(x, z) + dist(z, y)$  [ triangle inequality ]

for every 3 point  
↗

## Center selection example

Ex: each site is a point in the plane, a center can be any point in the plane,  
 $dist(x, y)$  = Euclidean distance.

Remark: search can be infinite!



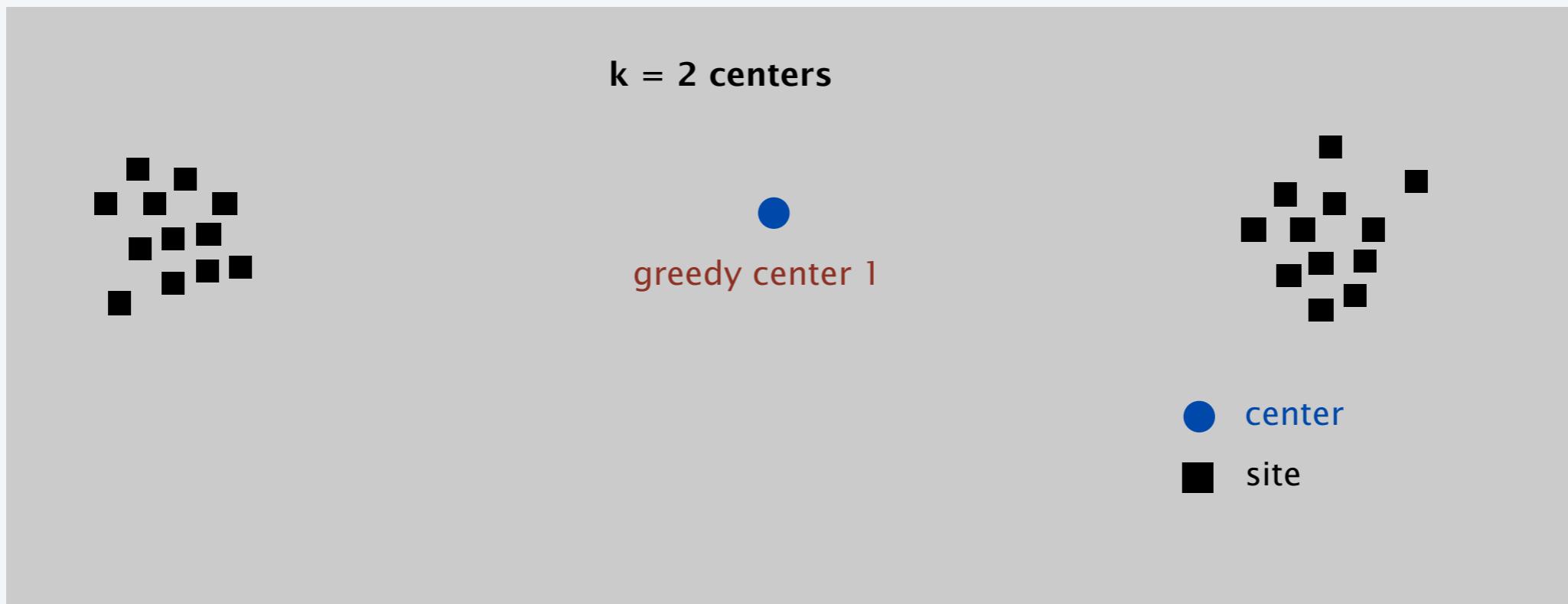
## Greedy algorithm: a **false start**

---

**Greedy algorithm.** Put the first center at the best possible location for a single center, and then keep adding centers so as to reduce the covering radius each time by as much as possible.

---

Remark: arbitrarily bad!



## Center selection: greedy algorithm

Repeatedly choose next center to be site **farthest** from any existing center.

Very simple, but also optimal  $\rho$

GREEDY-CENTER-SELECTION ( $k, n, s_1, s_2, \dots, s_n$ )

$C \leftarrow \emptyset$ .

REPEAT  $k$  times    *first arbitrarily in the metric space*

Select a site  $s_i$  with maximum distance  $dist(s_i, C)$ .

$C \leftarrow C \cup s_i$ .

RETURN  $C$ .

↑  
site farthest  
from any center

**Property.** Upon termination, all centers in  $C$  are pairwise at least  $r(C)$  apart.

**Pf.** By construction of algorithm.

## Center selection: analysis of greedy algorithm

radius of  $K$  optimal centers  
↑

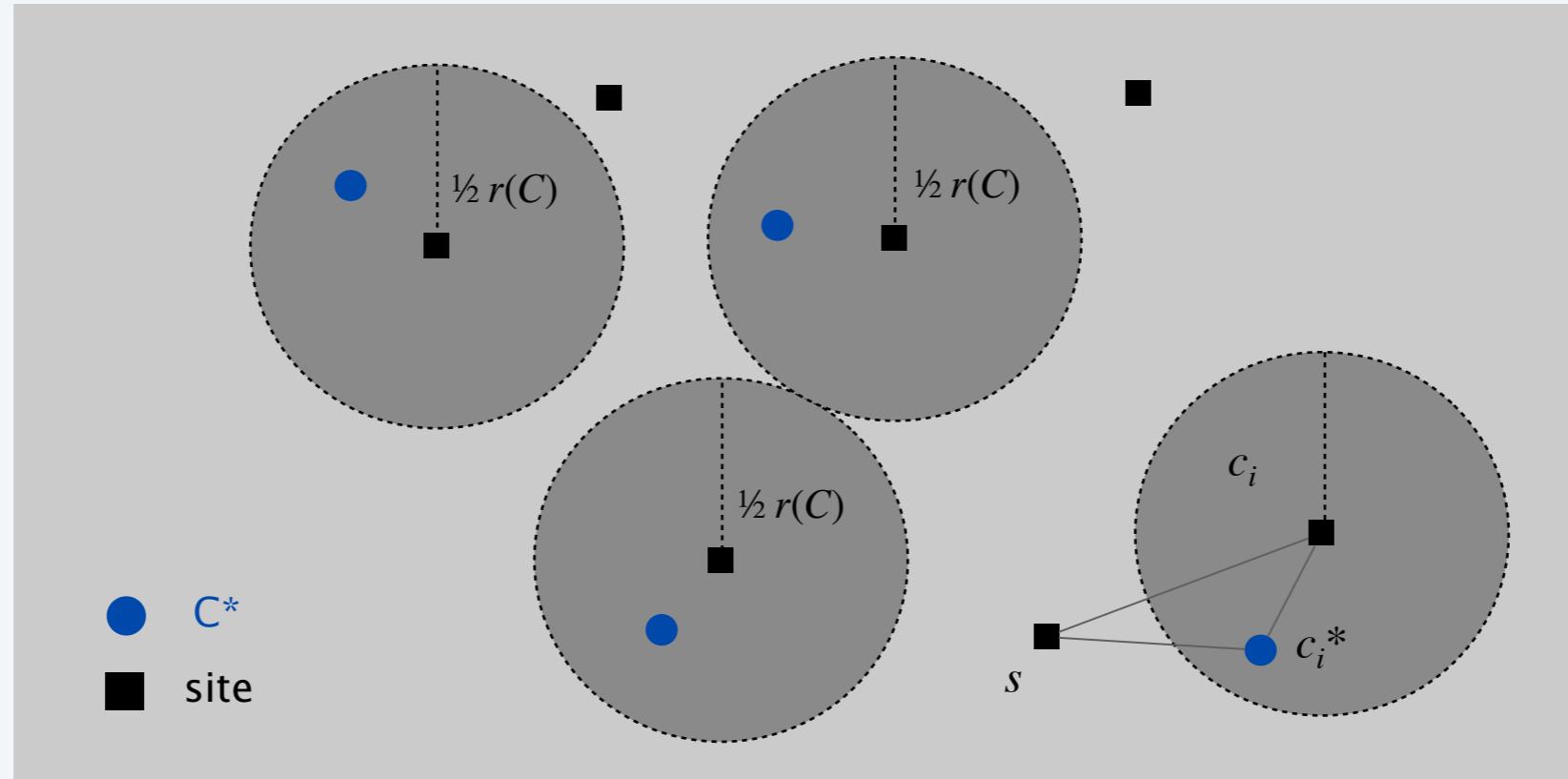
**Lemma.** Let  $C^*$  be an optimal set of centers. Then  $r(C) \leq 2r(C^*)$ .

**Pf.** [by contradiction] Assume  $r(C^*) < \frac{1}{2} r(C)$ .

- For each site  $c_i \in C$ , consider ball of radius  $\frac{1}{2} r(C)$  around it.
- Exactly one  $c_i^*$  in each ball; let  $c_i$  be the site paired with  $c_i^*$ .
- Consider any site  $s$  and its closest center  $c_i^* \in C^*$ .
- $dist(s, C) \leq dist(s, c_i) \leq dist(s, c_i^*) + dist(c_i^*, c_i) \leq 2r(C^*)$ .
- Thus,  $r(C) \leq 2r(C^*)$ .  
■

Δ-inequality

$\leq r(C^*)$  since  $c_i^*$  is closest center



## Center selection

---

**Lemma.** Let  $C^*$  be an optimal set of centers. Then  $r(C) \leq 2r(C^*)$ .

**Theorem.** Greedy algorithm is a 2-approximation for center selection problem.

**Remark.** Greedy algorithm always places centers at sites, but is still within a factor of 2 of best solution that is allowed to place centers anywhere.

e.g., points in the plane

**Question.** Is there hope of a  $3/2$ -approximation?  $4/3$ ?

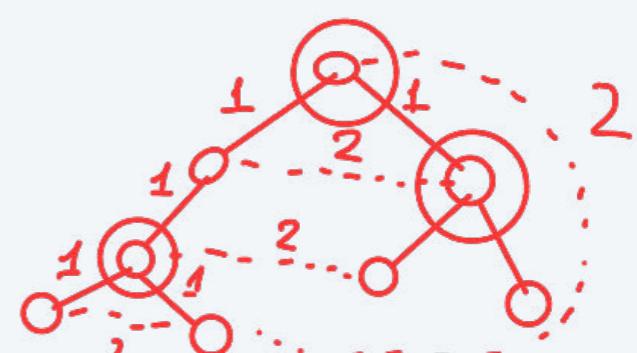
## Dominating set reduces to center selection

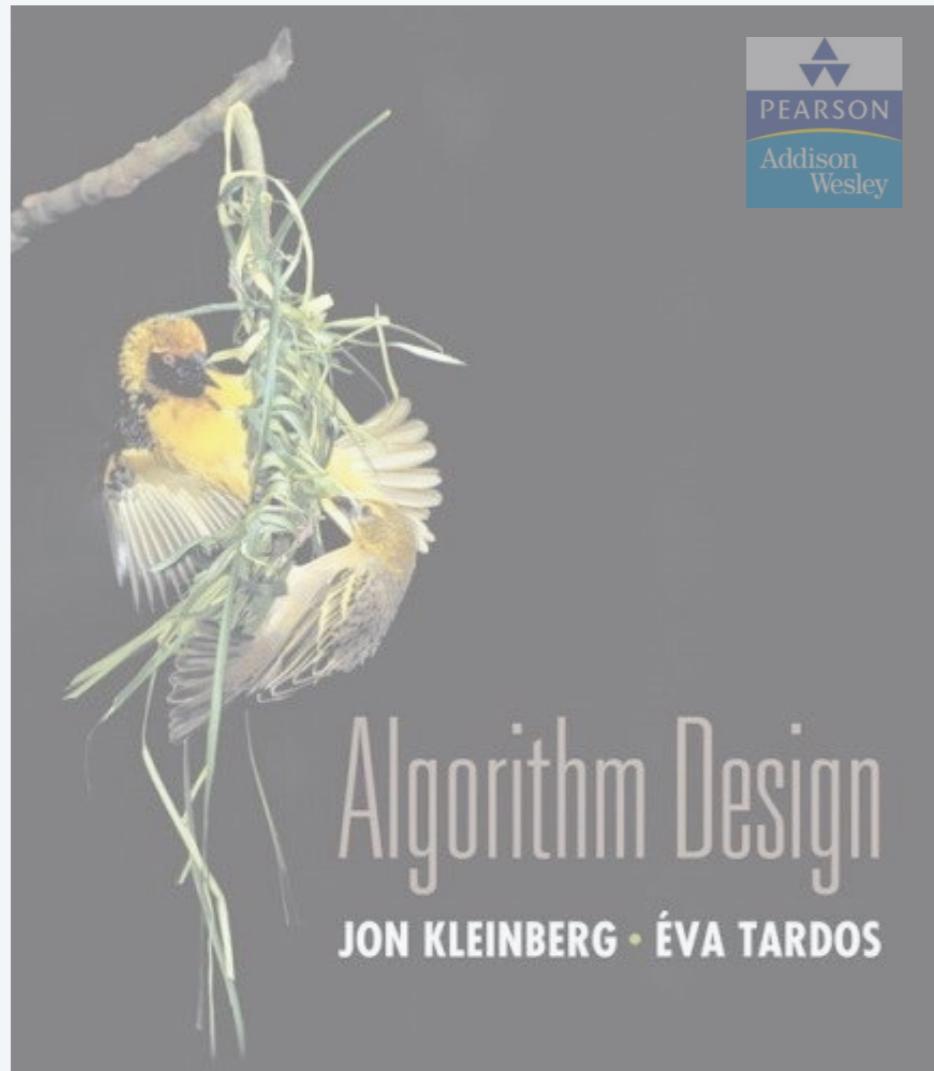
DOMINATING SET is problem of finding  
a dominating set of size  $\leq k$ .

Theorem. Unless  $P = NP$ , there no  $\rho$ -approximation for center selection problem for any  $\rho < 2$ .

Pf. We show how we could use a  $(2 - \varepsilon)$  approximation algorithm for CENTER-SELECTION selection to solve DOMINATING-SET in poly-time.

- Let  $G = (V, E)$ ,  $k$  be an instance of DOMINATING-SET.
- Construct instance  $G'$  of CENTER-SELECTION with sites  $V$  and distances
  - $dist(u, v) = 1$  if  $(u, v) \in E$
  - $dist(u, v) = 2$  if  $(u, v) \notin E$
- Note that  $G'$  satisfies the triangle inequality.
- $G$  has dominating set of size  $k$  iff there exists  $k$  centers  $C^*$  with  $r(C^*) = 1$ .
- Thus, if  $G$  has a dominating set of size  $k$ , a  $(2 - \varepsilon)$ -approximation algorithm for CENTER-SELECTION would find a solution  $C^*$  with  $r(C^*) = 1$  since it cannot use any edge of distance 2. ▀





## 11. APPROXIMATION ALGORITHMS

---

- ▶ *load balancing*
- ▶ *center selection*
- ▶ ***pricing method: vertex cover***
- ▶ *LP rounding: vertex cover*
- ▶ *generalized load balancing*
- ▶ *knapsack problem*

## Weighted vertex cover

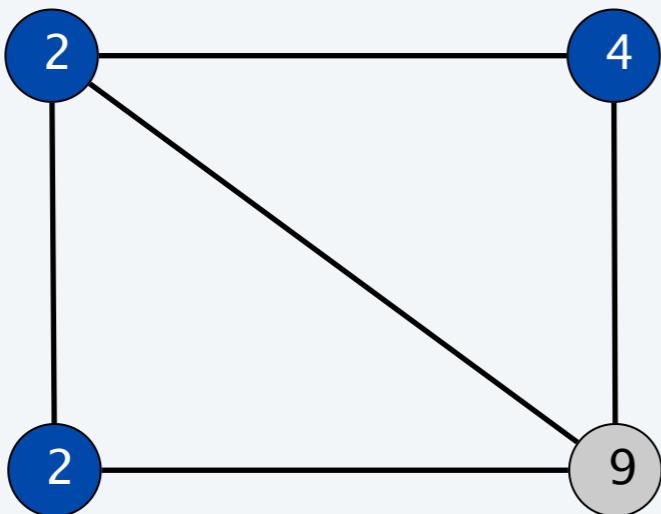
---

**Definition.** Given a graph  $G = (V, E)$ , a vertex cover is a set  $S \subseteq V$  such that each edge in  $E$  has at least one end in  $S$ .

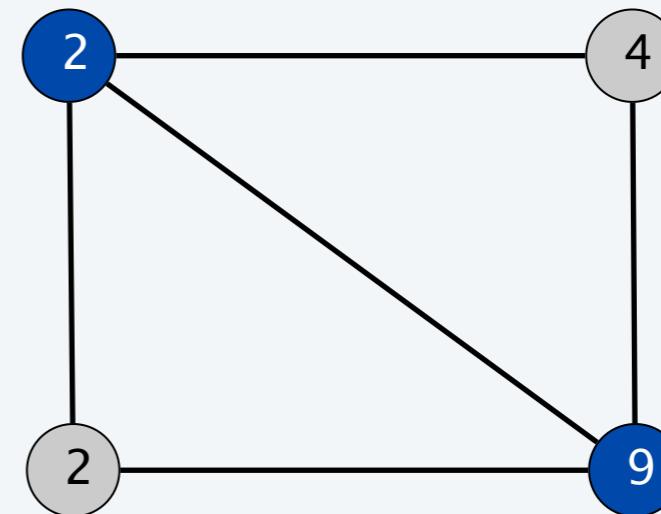
---

**Weighted vertex cover.** Given a graph  $G$  with vertex weights, find a vertex cover of minimum weight.

---



**weight = 2 + 2 + 4**



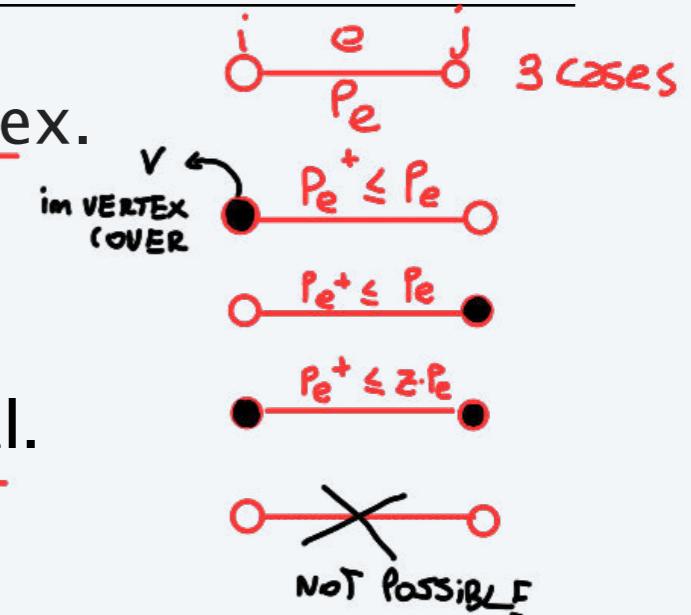
**weight = 11**

## Pricing method

$$\sum_{e \in E} p_e \leq \sum_{i \in S} \sum_{e=(i,j)} p_e$$

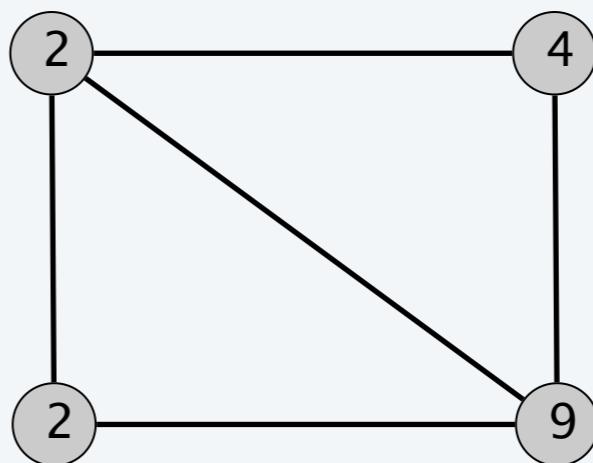
**Pricing method.** Each edge must be covered by some vertex.

Edge  $e = (i, j)$  pays price  $p_e \geq 0$  to use both vertex  $i$  and  $j$ .



**Fairness.** Edges incident to vertex  $i$  should pay  $\leq w_i$  in total.

for each vertex  $i$ :  $\sum_{e=(i,j)} p_e \leq w_i$



IF WE HAVE LOCAL FAIR  
WE HAVE ALSO GLOBAL FAIR

**Fairness lemma.** For any vertex cover  $S$  and any fair prices  $p_e$ :  $\sum_e p_e \leq w(S)$ .

Pf.

$$\sum_{e \in E} p_e \leq \sum_{i \in S} \sum_{e=(i,j)} p_e \leq \sum_{i \in S} w_i = w(S). \blacksquare$$

each edge  $e$  covered by  
at least one node in  $S$

sum fairness inequalities  
for each node in  $S$

# Pricing method

Set prices and find vertex cover simultaneously.

WEIGHTED-VERTEX-COVER ( $G, w$ )

$S \leftarrow \emptyset$ .

FOREACH  $e \in E$

$p_e \leftarrow 0$ .

$$\sum_{e=(i,j)} p_e = w_i$$

selected at random  
+—————  
WHILE (there exists an edge  $(i, j)$  such that neither  $i$  nor  $j$  is tight)



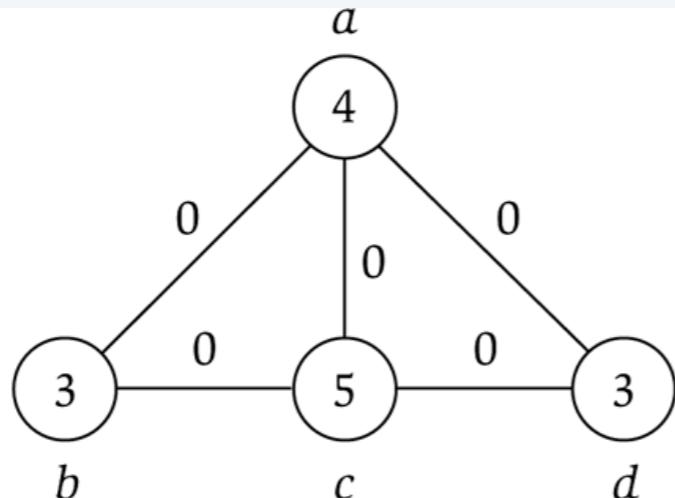
Select such an edge  $e = (i, j)$ .

Increase  $p_e$  as much as possible until  $i$  or  $j$  tight.

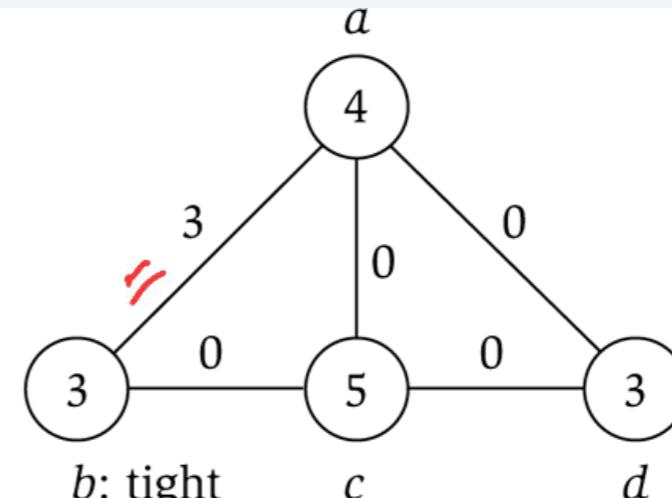
$S \leftarrow$  set of all tight nodes.

RETURN  $S$ .

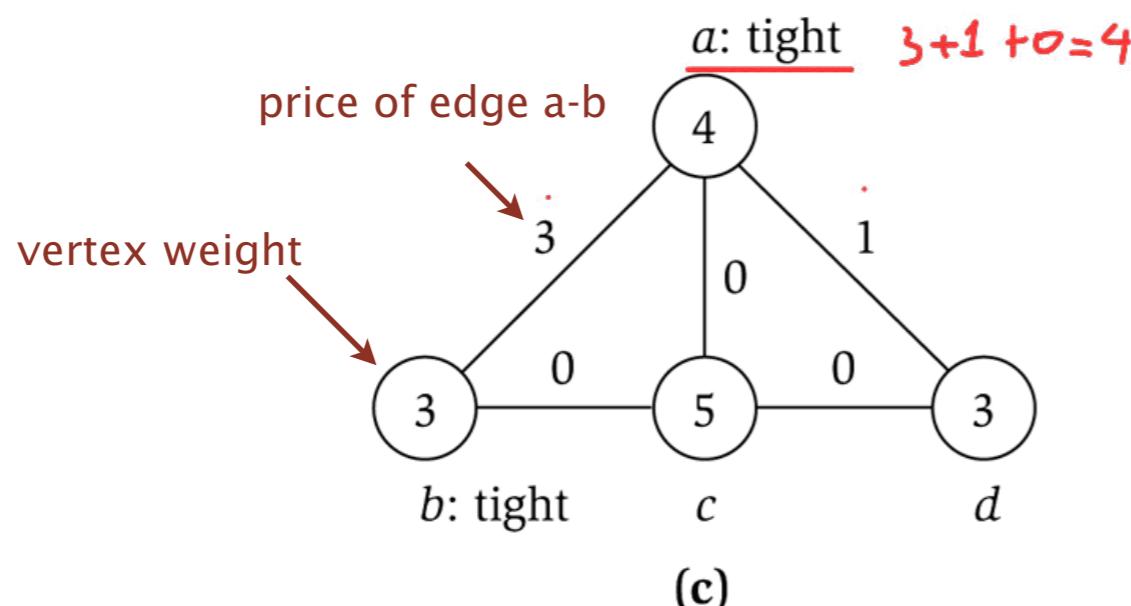
# Pricing method example



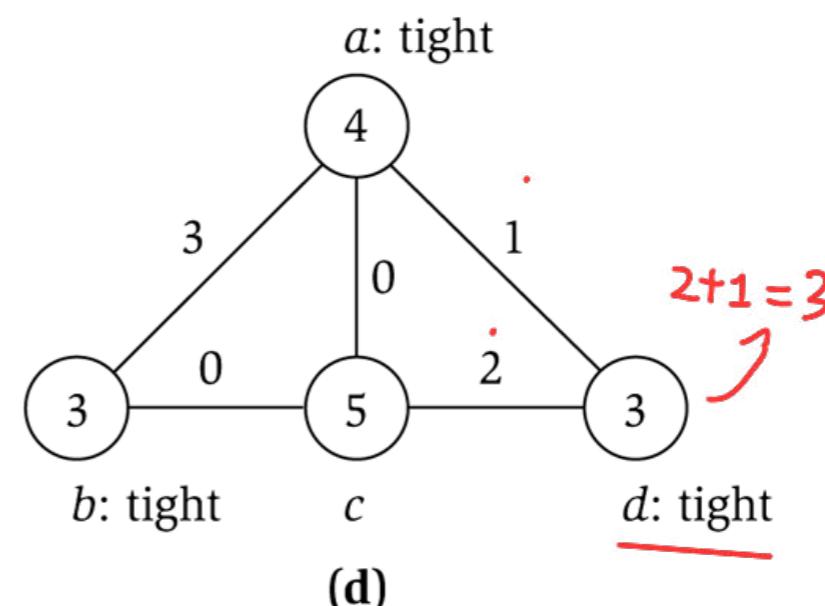
(a)



(b)



(c)



(d)

## Pricing method: analysis

**Theorem.** Pricing method is a 2-approximation for WEIGHTED-VERTEX-COVER.

Pf.

- Algorithm terminates since at least one new node becomes tight after each iteration of while loop.
- Let  $S$  = set of all tight nodes upon termination of algorithm.  
 $S$  is a vertex cover: if some edge  $(i, j)$  is uncovered, then neither  $i$  nor  $j$  is tight. But then while loop would not terminate.
- Let  $S^*$  be optimal vertex cover. We show  $w(S) \leq 2 w(S^*)$ .

$$w(S) = \sum_{i \in S} w_i = \sum_{i \in S} \sum_{e=(i,j)} p_e \leq \sum_{i \in V} \sum_{e=(i,j)} p_e = 2 \sum_{e \in E} p_e \leq 2w(S^*)$$

■

First argument

all nodes in  $S$  are tight

$S \subseteq V$ ,  
prices  $\geq 0$

each edge counted twice

fairness lemma

Second argument

# *Approximation Algorithms: Basic Concepts*

*Stefano Leonardi*

*Sapienza University of Rome*

## **NP-completeness**

There **many polynomial time solvable optimization problems**.  
e.g. maximum matching, min-cost flow, minimum cut, etc.

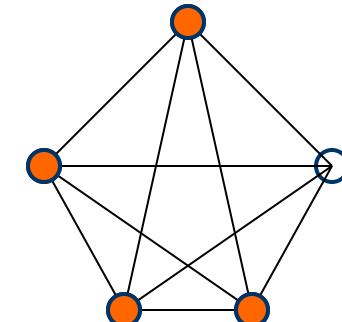
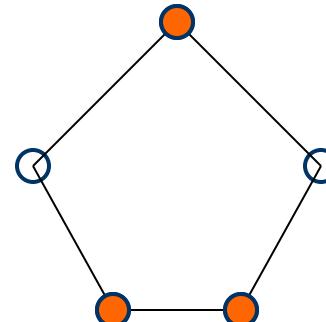
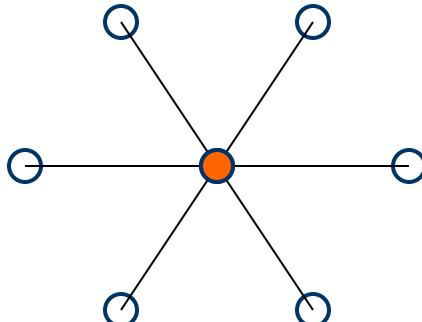
However, **there are much more optimization problems that are NP-complete**: we **do not know how to solve in polynomial time**.

e.g. traveling salesman, graph colorings, maximum independent set, set cover, maximum clique, maximum cut, minimum Steiner tree, satisfiability, etc.

# **Vertex Cover**

**Vertex cover**: a subset of vertices which “covers” every edge.  
An edge is covered if one of its endpoint is chosen.

**The Minimum Vertex Cover Problem**: Given a graph  
find a vertex cover with minimum number of vertices.  
(optimization problem)



## *Some Alternatives*

- ❖ Special graph classes

e.g. vertex cover in bipartite graphs, perfect graphs.

- ❖ Fixed parameter algorithms

find a vertex cover of size  $k$  efficiently for small  $k$ .

- ❖ Average case analysis

find an algorithm which works well on average.

- ❖ Approximation algorithms

find an algorithm which return solutions that are guaranteed to be close to an optimal solution.

# **Approximation Algorithms**

Key: provably close to optimal.

Let  $\text{OPT}$  be the value of an optimal solution,  
and let  $\text{SOL}$  be the value of the solution that our algorithm returned.

**Additive approximation algorithms:**  $\text{SOL} \leq \text{OPT} + c$  for some constant  $c$ .

Very few examples known:

edge coloring, minimum maximum-degree spanning tree, bin packing

**Constant factor approximation algorithms:**

$\text{SOL} \leq c \text{OPT}$  for some constant  $c$ .

Many more examples known.

# **Different approximation factors**

**Constant approximation algorithms:**  $SOL \leq c \text{ OPT}$

e.g.: Vertex Cover, TSP, Max SAT, Steiner Tree, Facility Location,  
Max Cut

**Logarithmic approximation algorithms:**  $SOL = O(\log n)^c \text{ OPT}$

e.g.: Set Cover, Multi-Cut, Dominating Set,..

**Polynomial approximation:**  $SOL = O(n^c) \text{ OPT}$ ,  $c \leq 1$

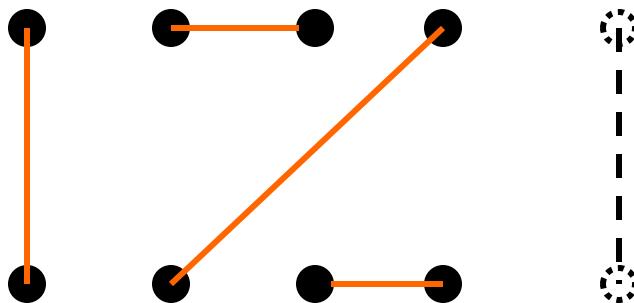
e.g: Max Clique, Independent Set, Coloring

**Polynomial Approximation Schemes:**  $SOL \leq (1+\epsilon) \text{ OPT}$ , for each  $\epsilon > 0$

Fully Polynomial Approximation schemes if running time  $O(\text{poly}(1/\epsilon))$

e.g.: Knapsack, Scheduling, Budgeted MST, Euclidean TSP, Bin packing

## *Vertex Cover: 2 apx algorithm*



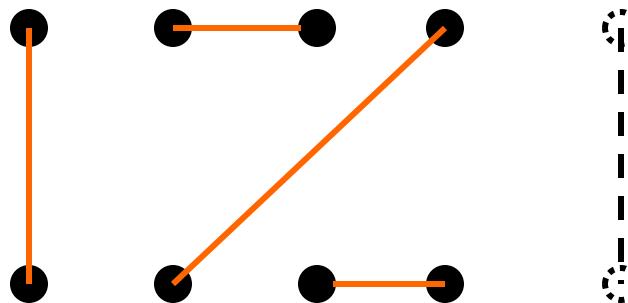
Fix a maximum matching. Call the vertices involved **black**.

Since the matching is maximum, every edge must have a black endpoint.

So, by choosing all the black vertices, we have a vertex cover.

**SOL  $\leq 2 * \text{size of a maximum matching}$**

# *Vertex Cover: 2-apx algorithm*



What about an optimal solution?

Each edge in the matching has to be covered by a **different vertex!**

**Lower bound to the optimal solution:**

OPT  $\geq$  size of a maximum matching

So,  $SOL \leq 2 \text{ OPT}$ , and we have a 2-approximation algor.!

# *Vertex Cover*

**Approximate min-max theorem:**

Maximum matching  $\leq$  minimum vertex cover  $\leq 2 \cdot$  maximum matching

Is the analysis tight? Yes. Consider for instance a bipartite graph.

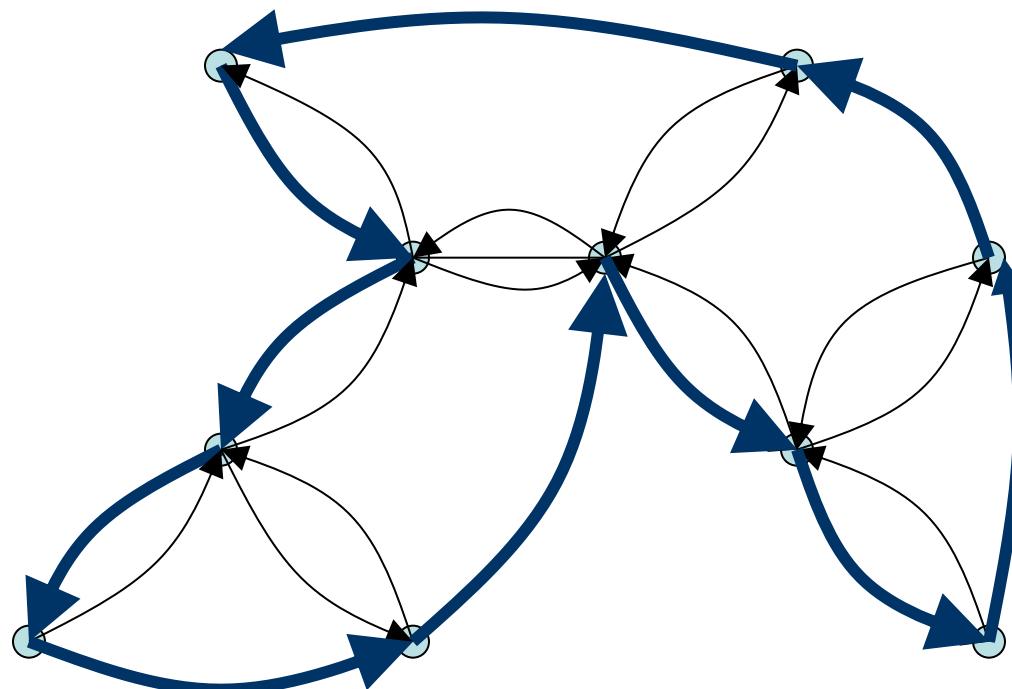
**Major open question:**

Can we obtain a  $c$ -approximation algorithm,  $c < 2$ ?

**Hardness result:**

It is NP-complete even to *approximate* within a factor of 1.36!!

# Approximation Algorithms for (Min) Traveling Salesman Problem (TSP)



# The Hamiltonian Path Problem

## Hamiltonian Path Problem:

Given an undirected graph, find a cycle visiting every vertex exactly once.

## Eulerian Path Problem:

Given an undirected graph, find a walk visiting every edge exactly once.

Notice that in a walk some vertices may have been visited more than once.

The Eulerian Path problem is polynomial time solvable.

A graph has an Eulerian path if and only if every vertex has an even degree.

The Hamiltonian Path problem is NP-complete.

# The Traveling Salesman Problem

**Traveling Salesman Problem (TSP):**

Given a complete graph with nonnegative edge costs,

Find a minimum cost cycle visiting every vertex exactly once.

Given a number of cities and the costs of traveling from any city to any other city, what is the cheapest round-trip route that visits each city exactly once and then returns to the starting city?

One of the most well-studied problem in combinatorial optimization.

# Inapproximability of Traveling Salesman Problem

**Theorem:** There is no constant factor approximation algorithm for TSP, unless P=NP.

Idea: Use the Hamiltonian path problem.

- For each edge, we add an edge of cost 1.
- For each non-edge, we add an edge of cost nk.

- If there is a Hamiltonian path, then there is a cycle of cost n.
- If there is no Hamiltonian path, then every cycle has cost greater than nk.

So, if you have a k-approximation algorithm for TSP, one just needs to check if the returned solution is at most nk.

- If yes, then the original graph has a Hamiltonian path.
- Otherwise, the original graph has no Hamiltonian path.

# Inapproximability of Traveling Salesman Problem

**Theorem:** There is no constant factor approximation algorithm for TSP, unless P=NP.

This type of theorem is called “hardness result” in the literature.  
Just like their names, usually they are very hard to obtain.

- If there is a Hamiltonian path, then there is a cycle of cost n.
- If there is no Hamiltonian path, then every cycle has cost greater than nk.

The strategy is usually like this.

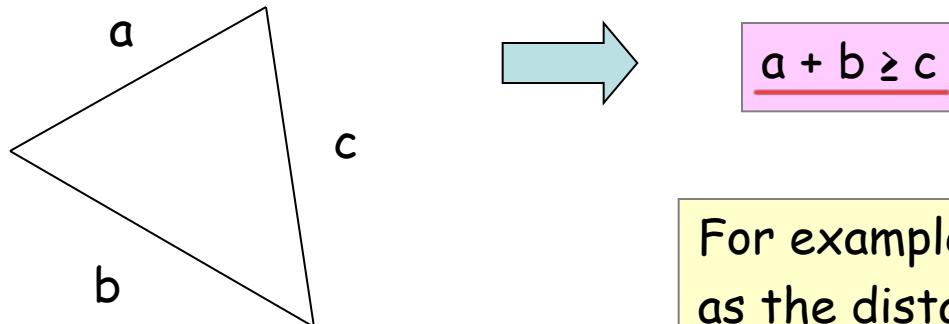
This creates a gap between yes and no instances.

The bigger the gap, the problem is harder to approximate.

# Approximation Algorithm for Metric TSP

Metric Traveling Salesman Problem (metric TSP):

Given a complete graph with edge costs satisfying triangle inequalities,  
Find a minimum cost cycle visiting every vertex exactly once.



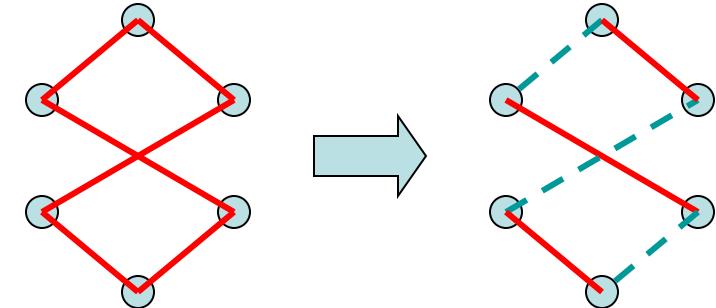
For example, think of cost of an edge  
as the distance between two points.

How could triangle inequalities help in finding approximation algorithm?

# Lower Bounds for TSP

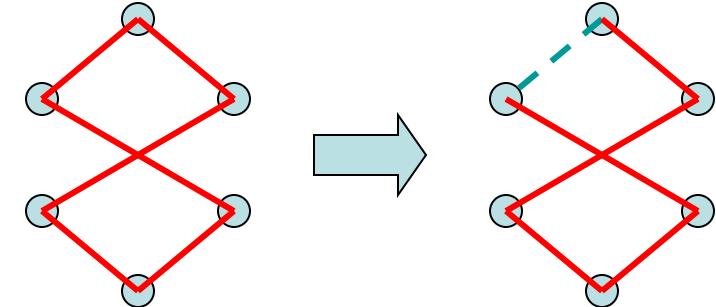
What can be a good lower bound to the cost of TSP?

A tour contains a matching.



Let OPT be the cost of an optimal tour, since a tour contains two matchings, the cost of a minimum weight perfect matching is at most  $OPT/2$ .

A tour contains a spanning tree.



So, the cost of a minimum spanning tree is at most  $OPT$ .

# Spanning Tree and TSP

Let the thick edges have cost 1,  
And all other edges have cost greater than 1.

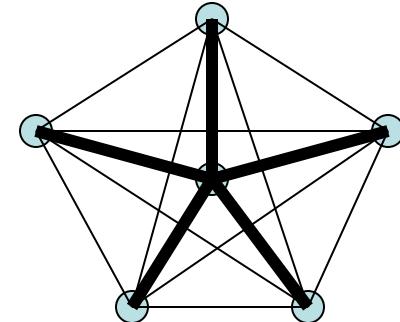
So the thick edges form a minimum spanning tree.

But it doesn't look like a Hamiltonian cycle at all!

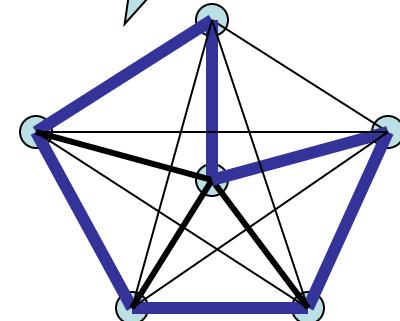
Consider a Hamiltonian cycle.

The costs of the edges which are not in the  
minimum spanning tree might have very high costs.

Not really! Each such edge has cost at most 2  
because of the triangle inequality.

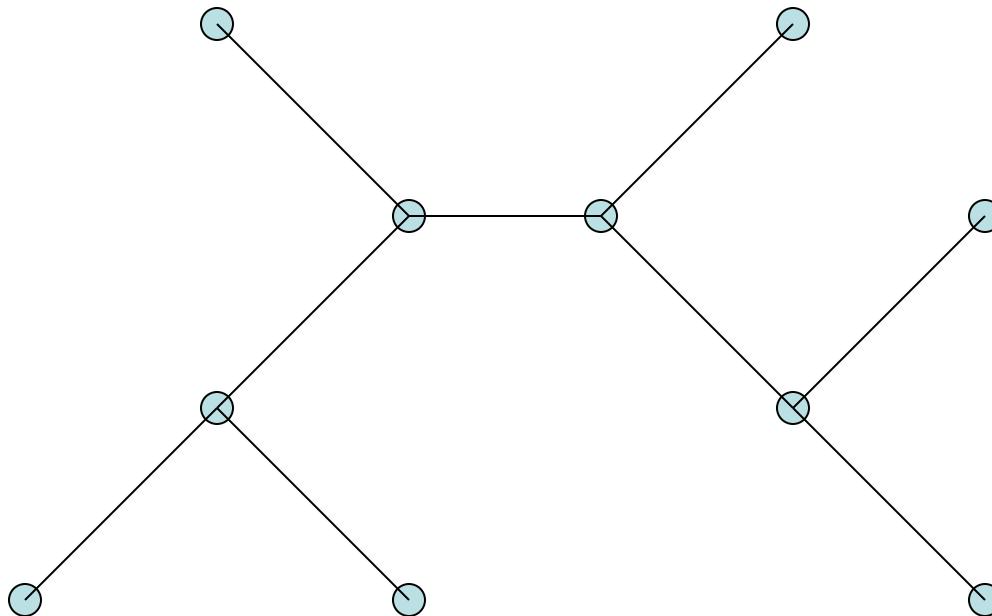


Edge cost  
at most 2



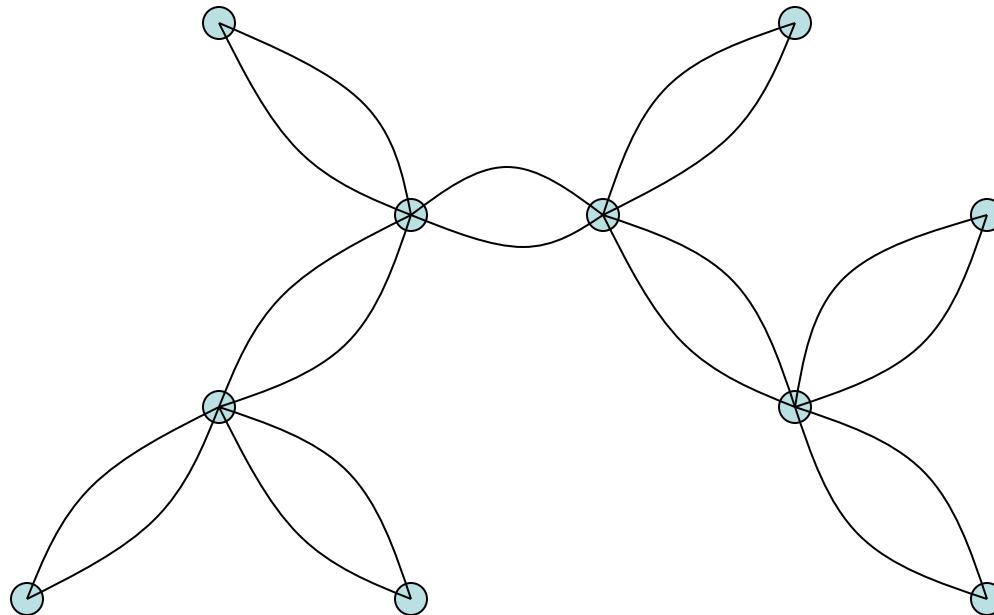
# Spanning Tree and TSP

How to formalize the idea of “following” a minimum spanning tree?



# Spanning Tree and TSP

How to formalize the idea of "following" a minimum spanning tree?

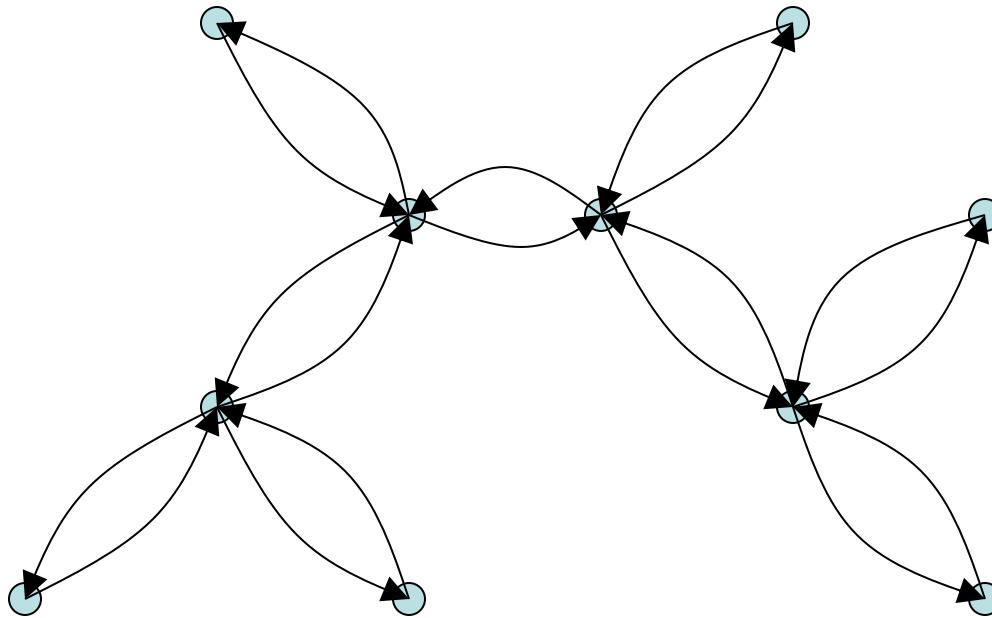


Key idea: double all the edges  
and find an Eulerian tour.

This graph has cost 2MST.

# Spanning Tree and TSP

How to formalize the idea of "following" a minimum spanning tree?

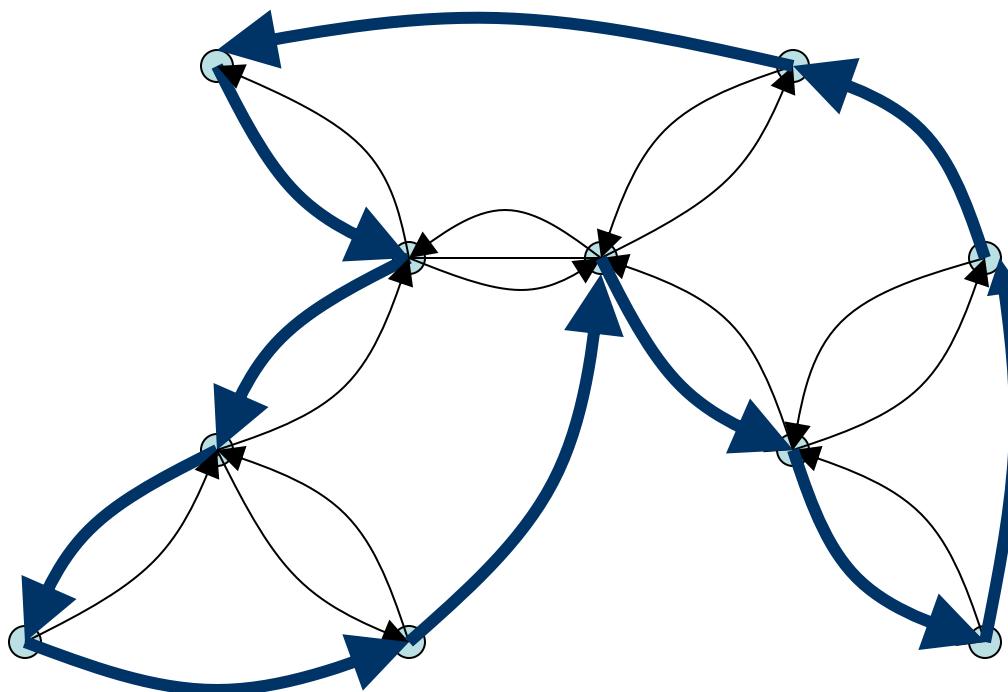


Key idea: double all the edges  
and find an Eulerian tour.

This graph has cost  $2MST$ .

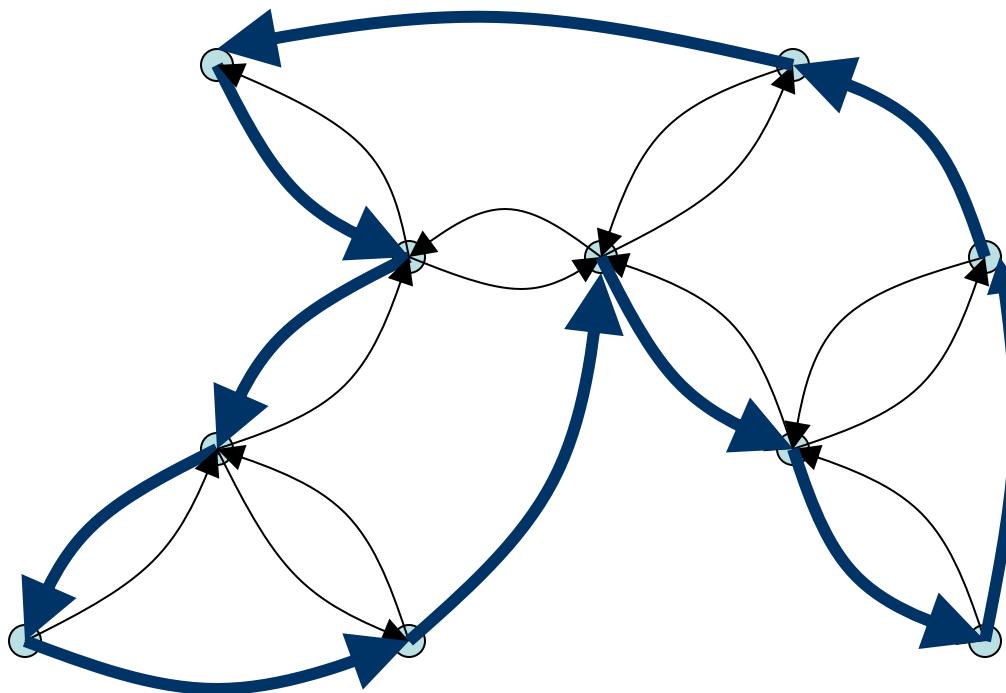
# Spanning Tree and TSP

**Strategy:** shortcut this Eulerian tour.



# Spanning Tree and TSP

By triangle inequalities, the shortcut tour is not longer than the Eulerian tour.



Each directed edge is used exactly once in the shortcut tour.

# A 2-Approximation Algorithm for Metric TSP

## (Metric TSP - Factor 2)

1. Find an MST,  $T$ , of  $G$ .
2. Double every edge of the MST to obtain an Eulerian graph.
3. Find an Eulerian tour,  $T^*$ , on this graph.
4. Output the tour that visits vertices of  $G$  in the order of their first appearance in  $T^*$ . Let  $C$  be this tour.  
(That is, shortcut  $T^*$ )

### Analysis:

1.  $\text{cost}(T) \leq \text{OPT}$  (because MST is a lower bound of TSP)
2.  $\text{cost}(T^*) = 2\text{cost}(T)$  (because every edge appears twice)
3.  $\text{cost}(C) \leq \text{cost}(T^*)$  (because of triangle inequalities, shortcutting)
4. So,  $\text{cost}(C) \leq 2\text{OPT}$

# Better approximation?

There is a 1.5 approximation algorithm for metric TSP.

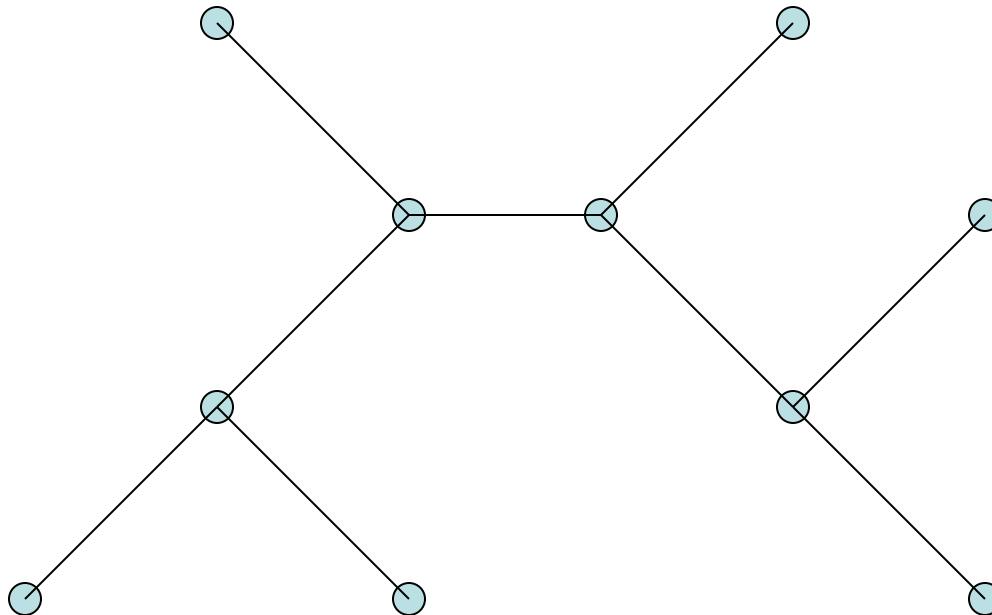
Hint: use a minimum spanning tree and a maximum matching  
(instead of double a minimum spanning tree). See textbook

Major open problem: Improve this to  $4/3$ ?

An aside: hardness result is not an excuse to stop working,  
but to guide us to identify interesting cases.

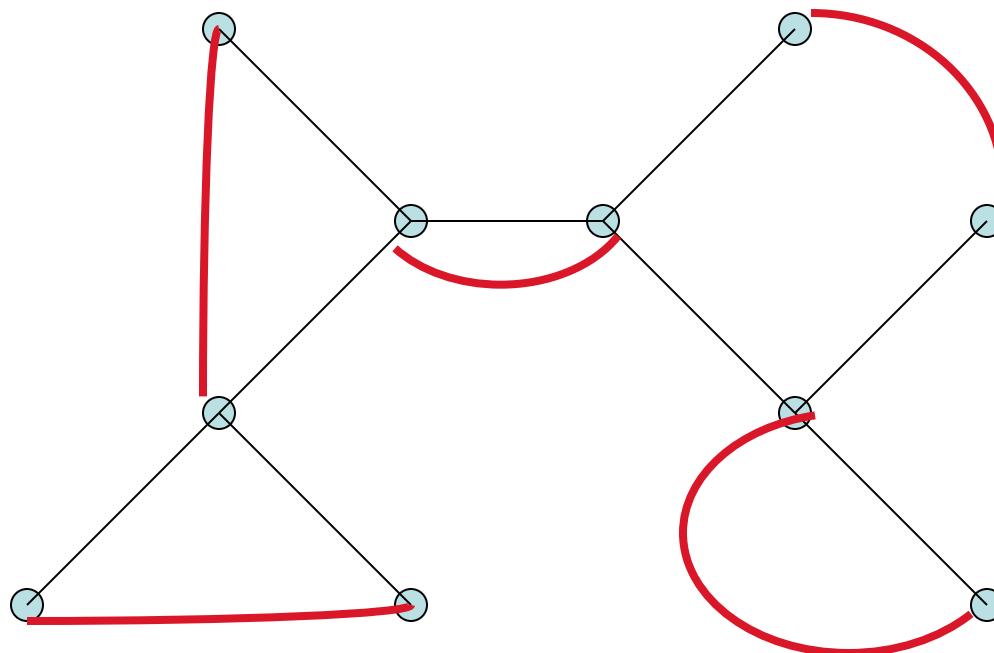
# Spanning Tree matching and TSP

How to generalize the idea of "following" a minimum spanning tree?



# Spanning Tree matching and TSP

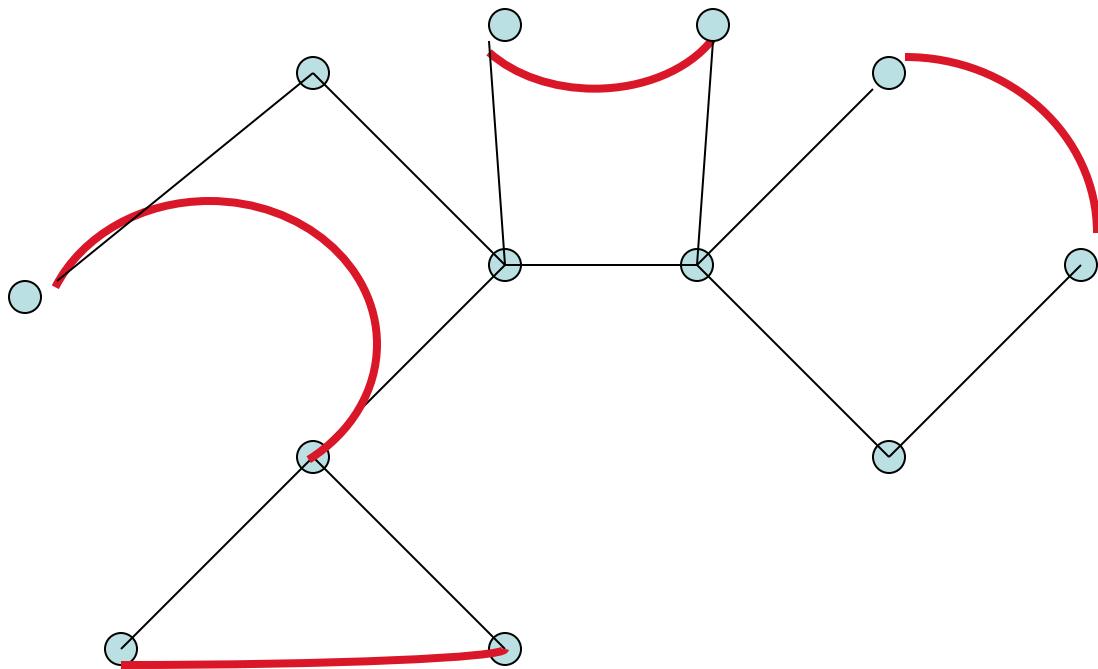
How to generalize the idea of "following" a matching and a minimum spanning tree?



IDEA: obtain an Eulerian graph using a spanning tree and a matching

# Spanning Tree matching and TSP

Eulerian graph: each vertex has even degree



Christofides: apply matching to vertices with odd degree in the minimum spanning tree!

# A Christofides' Algorithm for Metric TSP

(Metric TSP - Factor 1.5)

1. Find an MST,  $T$ , of  $G$ .
2. Find a matching  $M$  among odd degree vertices.
3. Find an Eulerian tour  $E$ , on the graph with edges from  $T$  and  $M$ .
4. Output the tour  $C$  that visits vertices of  $G$  in the order of their first appearance. Let  $C$  be this tour.  
  
(That is, shortcut  $T^*$ )

Analysis:

1.  $\text{cost}(T) \leq \text{OPT}$  (because MST is a lower bound of TSP)
2.  $\text{cost}(M) \leq 0.5 \text{ OPT}$  (because every edge appears twice)
3.  $\text{cost}(C) \leq \text{cost}(T) + \text{cost}(M)$  (because of triangle inequalities,  
shortcutting)
4. So,  $\text{cost}(C) \leq 1.5 \text{ OPT}$

# Approximation criteria

An algorithm is  $c$  approximation algorithm if

- $SOL \leq c \text{ OPT}$  for a minimization problem
- $SOL \geq c \text{ OPT}$  for a maximization problem

An algorithm **A** is an approximation scheme if for every  $\epsilon > 0$ ,

**A** runs in polynomial time (which may depend on  $\epsilon$ ) and return a solution:

- $SOL \leq (1+\epsilon)\text{OPT}$  for a minimization problem
- $SOL \geq (1-\epsilon)\text{OPT}$  for a maximization problem

# Knapsack

Given  $n$  objects with  
size( $a_i$ ) and profit( $a_i$ ),  $i=1,2,\dots,n$

Knapsack can only hold a  
• total weight of  $B$

Goal: to pick a subset which can fit into the knapsack  
and maximize the value of this subset.



# Knapsack

Given a set  $S = \{a_1, \dots, a_n\}$  of objects,  
with specified sizes and profits,  $\text{size}(a_i)$  and  $\text{profit}(a_i)$ ,  
and a knapsack capacity  $B$ , find a subset of objects whose  
total size is bounded by  $B$  and total profit is maximized.

Assume  $\text{size}(a_i)$ ,  $\text{profit}(a_i)$ , and  $B$  are all integers.

Integer programming formulation

$$\text{Max } \sum_i \text{profit}(a_i)x_i$$

$$\text{s.t. } \sum_i \text{size}(a_i)x_i \leq B,$$

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

# Greedy methods

## General greedy method:

Sort the objects by some rule,

and then put the objects into the knapsack according to this order

---

### Sort

- G1: Sort by object size in non-decreasing order
- G2: Sort by profit in non-increasing order
- G3: Sort by profit/object size in non-increasing order

Fact: All previous ordering do not work to find a good approximate solution

---

Theorem: For all instances G3 returns a solution SOL-G3 such that

---

$$\max(\text{profit}(a_i), \text{SOL-G3}) \geq 1/2 \text{ OPT}$$

Exercise: prove the Fact and the Theorem

# Dynamic Programming

Dynamic programming is just exhaustive search  
with polynomial number of subproblems.

We only need to compute each subproblem once,  
and each subproblem is looked up at most a polynomial number of times,  
and so the total running time is at most a polynomial.

# Dynamic Programming for Knapsack

Suppose we have considered object 1 to object i.

We want to remember what profits are achievable.

For each achievable profit, we want to minimize the size.

Let  $S(i,p)$  denote a subset of  $\{a_1, \dots, a_i\}$  whose total profit is exactly  $p$  and total size is minimized.

Let  $A(i,p)$  denote the size of the set  $S(i,p)$

$(A(i,p) = \infty \text{ if no such set exists}).$

For example,  $A(1,p) = \text{size}(a_1)$  if  $p = \text{profit}(a_1)$ ,

Otherwise  $A(1,p) = \infty$  (if  $p \neq \text{profit}(a_1)$ ).

# Recurrence

**Remember:** A(i,p) denote the minimum size to achieve profit p using objects from 1 to i.

**Goal:** we know A(i,q) for all q and we want to compute A(i+1,p)

**Two possibilities:**

If we do not choose object i+1:  
then  $A(i+1,p) = A(i,p)$ .

If we choose object i+1:  
then  $A(i+1,p) = \text{size}(a_{i+1}) + A(i, p - \text{profit}(a_{i+1}))$  if  $p > \text{profit}(a_{i+1})$

$A(i+1,p) = \text{minimum of these two values.}$

# An Example

Remember:  $A(i,p)$  denote the minimize size to achieve profit  $p$  using objects from 1 to  $i$ .

**Optimal Solution:**  $\max\{ p \mid A(n,p) \leq B \}$  where  $B$  is the size of the knapsack.

$\text{size}(a1)=2, \text{profit}(a1)=4; \text{size}(a2)=3, \text{profit}(a2)=5;$   
 $\text{size}(a3)=2, \text{profit}(a3)=3; \text{size}(a4)=1, \text{profit}(a4)=2$

B

# An Example

$$A(i+1, p) = \min\{A(i, p), \text{size}(a_{i+1}) + A(i, p - \text{profit}(a_{i+1}))\}.$$

$$A(2,p) = \min\{A(1,p), A(1,p-5)+3\}.$$

$\text{size}(a1)=2, \text{profit}(a1)=4; \text{size}(a2)=3, \text{profit}(a2)=5;$   
 $\text{size}(a3)=2, \text{profit}(a3)=3; \text{size}(a4)=1, \text{profit}(a4)=2$

# An Example

$$A(i+1, p) = \min\{A(i, p), \text{size}(a_{i+1}) + A(i, p - \text{profit}(a_{i+1}))\}.$$

$$A(3,p) = \min\{A(2,p), A(2,p-3)+2\}.$$

$\text{size}(a1)=2, \text{profit}(a1)=4; \text{size}(a2)=3, \text{profit}(a2)=5;$   
 $\text{size}(a3)=2, \text{profit}(a3)=3; \text{size}(a4)=1, \text{profit}(a4)=2$

# An Example

$$A(i+1, p) = \min\{A(i, p), \text{size}(a_{i+1}) + A(i, p - \text{profit}(a_{i+1}))\}.$$

$$A(4,p) = \min\{A(3,p), A(3,p-2)+1\}.$$

$\text{size}(a1)=2, \text{profit}(a1)=4; \text{size}(a2)=3, \text{profit}(a2)=5;$   
 $\text{size}(a3)=2, \text{profit}(a3)=3; \text{size}(a4)=1, \text{profit}(a4)=2$

# An Example

$$A(i+1, p) = \min\{A(i, p), \text{size}(a_{i+1}) + A(i, p - \text{profit}(a_{i+1}))\}.$$

$$A(4, p) = \min\{A(3, p), A(3, p-2)+1\}.$$

$\text{size}(a_1)=2, \text{profit}(a_1)=4; \text{ size}(a_2)=3, \text{profit}(a_2)=5;$   
 $\text{size}(a_3)=2, \text{profit}(a_3)=3; \text{ size}(a_4)=1, \text{profit}(a_4)=2$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	$\infty$	$\infty$	$\infty$	2	$\infty$									
2	0	$\infty$	$\infty$	$\infty$	2	3	$\infty$	$\infty$	$\infty$	5	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
3	0	$\infty$	$\infty$	2	2	3	$\infty$	4	5	5	$\infty$	$\infty$	7	$\infty$	$\infty$
4	0	$\infty$	1	2	2	3	3	4	5	5	6	7	$\infty$	8	

# An Example

Remember:  $A(i,p)$  denote the minimize size to achieve profit  $p$  using objects from 1 to  $i$ .

Optimal Solution:  $\max\{ p \mid A(n,p) \leq B \}$  where  $B$  is the size of the knapsack.

For example, if  $B=8$ ,  $OPT=14$ , if  $B=7$ ,  $OPT=12$ , if  $B=6$ ,  $OPT=11$ .

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	$\infty$	$\infty$	$\infty$	2	$\infty$									
2	0	$\infty$	$\infty$	$\infty$	2	3	$\infty$	$\infty$	$\infty$	5	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
3	0	$\infty$	$\infty$	2	2	3	$\infty$	4	5	5	$\infty$	7	$\infty$	$\infty$	
4	0	$\infty$	1	2	2	3	3	4	5	5	6	7	$\infty$	8	

## Running Time

The input has  $2n$  numbers, so the input total length is  $2n$

Assume  $P = \max_{i=1,2,\dots,n} \text{profit}(a_i)$  and  $S = \max_{i=1,2,\dots,n} \text{size}(a_i)$

So the input has total length  $2n\log(\max(P,S))$ .

For the dynamic programming algorithm,

there are  $n$  rows and at most  $nP$  columns.

Each entry can be computed in constant time (look up two entries).

So the total time complexity is  $O(n^2P)$ .

The running time is not polynomial if  $P$  is very large compared to  $n$

In fact the Knapsack problem is NP-complete and we do not expect

That there exists a polynomial time algorithm

# Scaling Down

Idea: use dynamic programming to scale down the numbers and compute the optimal solution in this modified instance

- Suppose  $P \geq 1000n$ , ( $P$  max profit).
- Then  $OPT \geq 1000n$  (each single object must enter the knapsack).
- Now scale down each element by 100 times (profit\*:=profit/100).
- Compute the optimal solution using this new profit.
- Can't distinguish between element of size, say 2199 and 2100.
- Each element contributes at most an error of 100.
- So total error is at most 100n.
- This is at most  $1/10$  of the optimal solution.
- However, the running time is 100 times faster.

# Approximation Scheme

**Goal:** to find a solution which is at least  $(1 - \epsilon)OPT$  for any  $\epsilon > 0$ .

## Approximation Scheme for Knapsack

1. Given  $\epsilon > 0$ , let  $K = \epsilon P/n$ , where  $P$  is the largest profit of an object.
2. For each object  $a_i$ , define  $\text{profit}^*(a_i) = \lfloor \text{profit}(a_i)/K \rfloor$ .
3. With these as profits of objects, using the dynamic programming algorithm, find the most profitable set, say  $S'$ .
4. Output  $S'$  as the approximate solution.

# Quality of Solution

**Theorem.** Let  $S$  denote the set returned by the algorithm. Then,  
 $\text{profit}(S) \geq (1 - \epsilon)\text{OPT}$ .

**Proof.** Let  $O$  denote the optimal set.

For each object  $a$ , because of rounding down,

$K \cdot \text{profit}^*(a)$  can be smaller than  $\text{profit}(a)$ , but by not more than  $K$ .

Since there are at most  $n$  objects in  $O$ ,

$$\text{profit}(O) - K \cdot \text{profit}^*(O) \leq nK.$$

Since the algorithm return an optimal solution under the new profits,

$$\begin{aligned}\text{profit}(S) &\geq K \cdot \text{profit}^*(S) \geq K \cdot \text{profit}^*(O) \geq \text{profit}(O) - nK \\ &= \text{OPT} - \epsilon P \geq (1 - \epsilon)\text{OPT}\end{aligned}$$

because  $\text{OPT} \geq P$ .

# Running Time

For the dynamic programming algorithm,  
there are  $n$  rows and at most  $n \lfloor P/K \rfloor$  columns.  
Each entry can be computed in constant time (look up two entries).  
So the total time complexity is  $O(n^2 \lfloor P/K \rfloor) = O(n^3/\epsilon)$ .

Therefore, we have an approximation scheme for Knapsack.

**Claim: Strong NP-hard problems do not admit FPAS  
assuming  $P \neq NP$**

**Knapsack is not Strong NP-hard**

# Approximation Scheme

## Quick Summary

1. Modify the instance by rounding the numbers.
2. Use dynamic programming to compute an optimal solution  $S$  in the modified instance.
3. Output  $S$  as the approximate solution.

Other examples: bin packing, Euclidean TSP.

logarithmic approximation

# Set covering

Given  $n$  subsets  $S_1, S_2, \dots, S_m$  of  $U$

with  $U = \{1, 2, \dots, n\}$  (so  $|U| = n$ )

Find a minimum subset  $C$  of  $\{1, 2, \dots, m\}$

Such that  $\bigcup_{i \in C} S_i = U$

NP-hard;

Goal: to pick a subset which cover all items  
and **minimize the cardinality** of this subset.

---

# Greedy for set covering

**General greedy method:**

Sol = emptyset

While not finished

choose the set that covers most elements not yet covered

Example

$$U=\{1,2,3,4,5,6\}$$

$$\text{Sets: } -S^1=\{1,2\} -S^2=\{3,4\} -S^3=\{5,6\} -S^4=\{1,3,5\}$$

Algorithm picks  $\{4,1,2,3\}$

Not optimal!  $\rightarrow \{1,2,3\}$  4 is not useful here

# Greedy for set covering

Notation:  $C^{OPT}$  = optimal cover let  $k=|C^{OPT}|$

$C^{OPT}$  is a collection of subsets

↳ defining lowerbound for characterized optimum

Fact: At any iteration of the algorithm, there exists  $S_j$  which contains at  $\geq 1/k$  fraction of yet-not-covered elements

↳  $U=15$  elements,  $k=5$

↳ must exist at least one set that cover 3 elements of the univers at any iteration

Proof: by contradiction.

If all sets cover  $<1/k$  fraction of yet-not-covered elements, there is no way to cover them using  $k$  sets

But  $C^{OPT}$  does that !

Therefore, at each iteration greedy covers  $\geq 1/k$  fraction of yet-not-covered elements

# Greedy for set covering

Fact: At any iteration of the algorithm, there exists  $S_j$  which contains at  $\geq 1/k$  fraction of yet-not-covered elements

Let  $C_i$  be the number of yet-not-covered elements at the beginning of step  $i=1,2,\dots$

We have  $C_{i+1} \leq C^i(1-1/k)$   $C_1=n$

Therefore, after  $t=k \ln n$  steps, we have

$C_t \leq C_0 (1-1/k)^t \leq n (1-1/k)^{k \ln n} < n 1/e^{\ln n} = 1$

I.e., all elements are covered by the  $k \ln n$  sets chosen by greedy algorithm

Opt size is  $k \Rightarrow$  greedy is  $\ln(n)$ -approximate

# The weighted set cover problem

- Assume set  $S$  has cost  $c(S)$ .
- Find a min cost collection of sets that cover all elements  $U = \{e_1, e_2, \dots, e_n\}$
- $C_i$  be the set of not yet covered elements at the beginning of step  $i$
- The greedy algorithm repeatedly selects the most cost-effective set, i.e. the set that maximizes  $c(S_i)/|S_i \cap C_i|$  ↗ cardinality
- Repeat till all elements are covered
- For an element  $e_j$  covered by set  $S_i$  define price( $e_j$ ) =  $c(S_i)/|S_i \cap C_i|$

↙ root of the algorithm

Claim:  $C_{ALG} = \sum_j \text{price}(e_j)$

EXAMPLE (same before)

$$c(S_i) = 1 \quad i=1,2 \quad c(S_3) = 3 \quad c(S_4) = 2$$

$$\text{Price}(S_1) = \frac{1}{2} = \frac{c(S_1)}{|S_1 \cap C_1|} \quad i=1,2$$

$$\text{Price}(S_3) = \frac{3}{2}$$

$$\text{Price}(S_4) = \frac{2}{3}$$

↙ for example:  $S_1 = \{e_1, e_2\} \rightarrow \text{Price}(e_1) + \text{Price}(e_2) = 1 = c(S_1)$

→ we select

$S_1$  (or  $S_2$ )  
minimum cost

→  $\text{Price}(S_2) = \frac{2}{2}$

$\text{Price}(S_3) = \frac{3}{2}$   
 $\text{Price}(S_4) = \frac{2}{2}$

$\text{Price}(S_2) = \frac{2}{1}$   
 $\text{Price}(S_3) = \frac{3}{1}$   
 $\text{Price}(S_4) = \frac{2}{1}$

$S_1, S_4 \rightarrow$   
 $S_2, S_3$

The price always increase at each step,

# Analysis of Greedy Set Cover

- **Claim:** When set  $S_i$  is selected there exists a set of OPT with cost effectiveness  $\leq \frac{C^{OPT}}{C_i}$   $\sim$  number of element uncovered at  $t=i$   
 $\hookrightarrow$  OPTIMAL COST
- For  $e_j$  covered by set  $S_i$  it holds  $price(e_j) \leq \frac{C^{OPT}}{C_i} \leq \frac{C^{OPT}}{n-j+1}$   
 $\sim$  order elements that are covered to the greedy

- We conclude  $C^{ALG} = \sum_j price(e_j) \leq \sum_j \frac{C^{OPT}}{n-j+1} \leq H_n C^{OPT}$   
 $\hookrightarrow$  Harmonic number

EXAMPLE: optimum cost = 15, 10 elements to be cover  
must be at least one set with price for element less than 3  
 $\hookrightarrow$  logarithmic

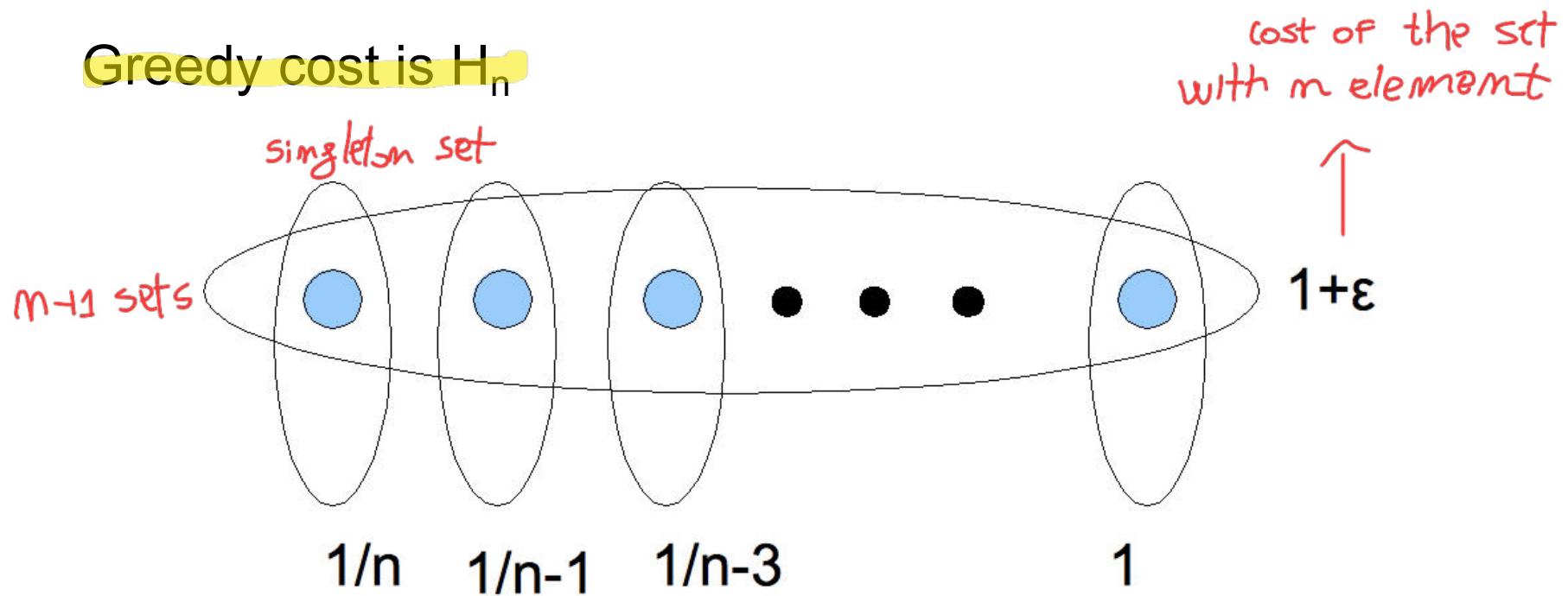
$$\frac{C^{OPT}}{C_i} \leq \frac{15}{5} = 3$$

$\hookrightarrow$  5 element need to be covered

# Lower bound for Greedy

Optimal cost is  $1 + \varepsilon$

Greedy cost is  $H_n$



FIRST SET PRICE  $\frac{1}{m}$  is strict less than  $\frac{1+\varepsilon}{m}$ , take  $\frac{1}{m}$

# Better algorithm for Set covering?

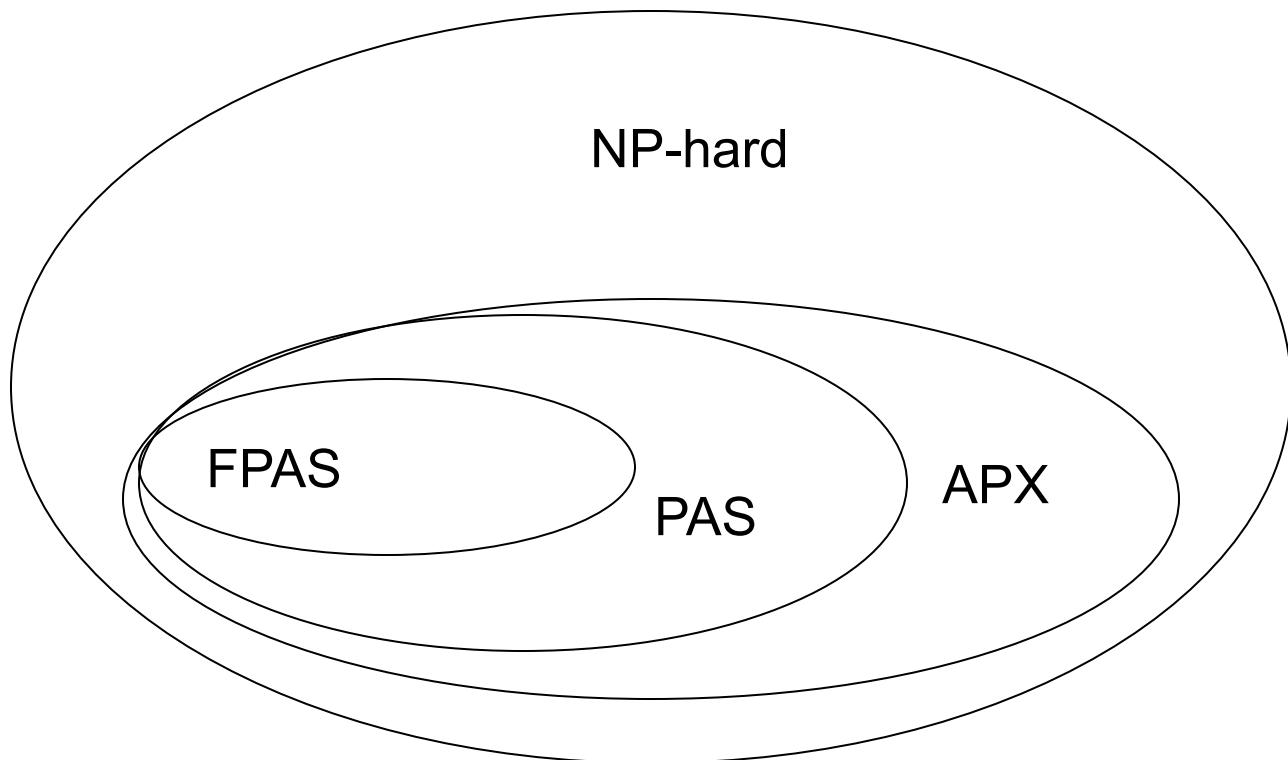
It is possible to show that approximating Set covering better than  $0.999.. \ln(m)$  is NP-hard!!

APX-hard problem: a problem for which there is a constant  $c$  such that it is NP-hard to find an approximation algorithm with approximation ratio better than  $c$

Equivalently: there exists constant  $c$  such that finding an approximation better than  $c$  is as hard as finding the optimal solution

**Claim: APX hard problems do not admit PAS (Polynomial Approximation Schemes)**

# APX, PAS and FPAS



# PAS and FPAS

**Class of APX problems: problems that have a Polynomial Approximation Algorithm:** for some constant  $c$  running time polynomial in input lenght

**Class of PAS problems: problem that have a Polynomial Approximation Schemes:** for any given  $\epsilon$  running time polynomial in input lenght

**Class of FPAS problems: problems that have a Fully Polynomial Approximation Schemes:** for any given  $\epsilon$  running time polynomial in input lenght and  $1/\epsilon$

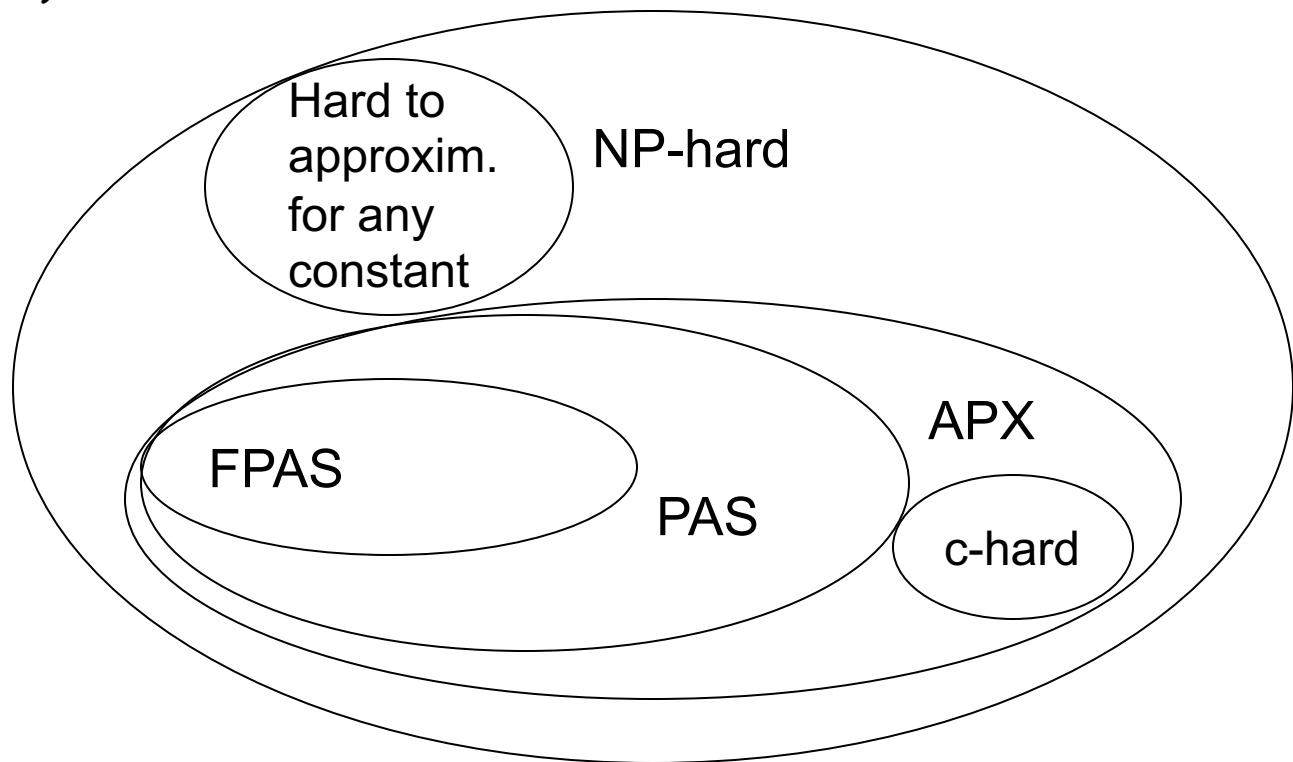
# APX-hard problems

There are problems that are hard to approximate:

For any constant: TSP in the general case, Set covering

For some constant  $c$ : Vertex cover, Max Sat (maximize no. of satisfied clauses)

APX hard problems  
do not admit PAS or  
FPAS



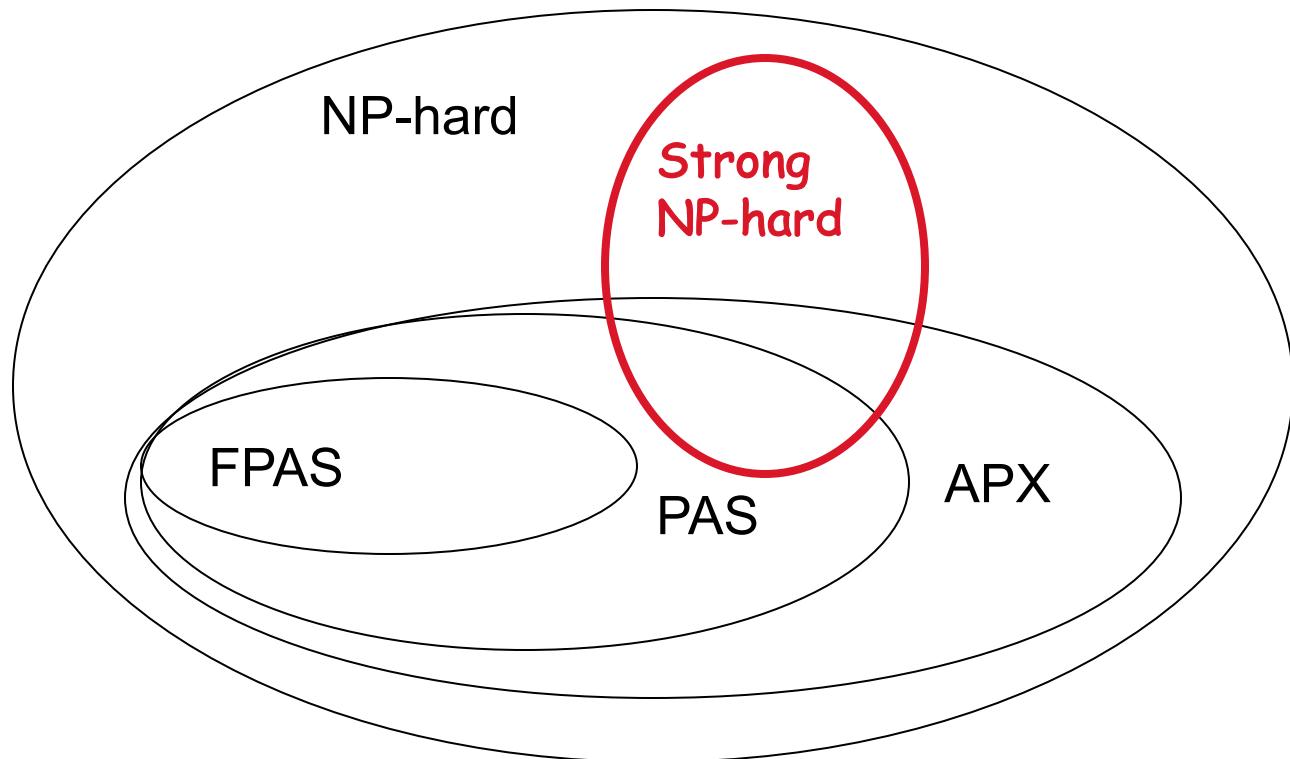
# Strong NP-hard problems

There are Strong NP-hard problems that admit a PAS

But finding a FPAS is as hard as finding the optimum:

Generalization of Knapsack with two constraints

Note: not all  
Strong NP-hard  
admit a good  
approximation



Linear programming is either a maximization or  
an optimization problem, we have linear objective, variables  $x(v)$   
we have coefficient for variable  $w(v \dots)$ , constraint on variable  
want to max/min a linear function subject to linear disequality  
 $\hookrightarrow$  exist polynomial time algorithm for solve it

## LP-based Approximation Algorithms

Stefano Leonardi

Sapienza Università di Roma

Theoretical Computer Science, Academic Year 2010/2011



# LP formulation for Vertex Cover

## Vertex Cover

$x(v)$  variable that define whatever we select  $v$  to be in  $V$ .

- $G = (V, E)$ ,  $w(u) \in \mathbb{R}^+$ ,  $u \in V$
- Find a set  $U \subseteq V$  of min total cost  $\sum_{u \in U} w(u)$  such that:
- $\forall e = (u, v) \in E$ , either  $u \in U$  or  $v \in U$

$$\begin{aligned} & \min \sum_{v \in V} x(v)w(v) \\ \text{s.t. } & x(u) + x(v) \geq 1 \quad \forall (u, v) \in E \quad \text{constraint that we want cover all edges} \\ & x(u) \in \{0, 1\} \quad u \in V \end{aligned}$$

objective function  
where we sum  
the weights of variable

INTEGER PROGRAMMING  
FORMULATION

→ we use LP  
for residue H

### Lecture Outline

#### LP-based Relaxation and Rounding

- Optimization Problems
- Relate to the Optimum
- Use the solution of the relaxed problem
- LP formulation for Vertex Cover
- LP-relaxation for Vertex Cover

- Rounding
- Approximation
- Integrality Gap
- Integrality Gap for Vertex Cover
- LP rounding for Set Cover
- LP formulation for Set Cover
- $f$ -approximation for Set Cover
- More basic techniques

#### The Primal-Dual Method

#### Primal-dual Method for Vertex Cover

#### Set Cover via Dual Lifting

#### Randomized Rounding



# LP-relaxation for Vertex Cover

how LP help us? → original Problem was an ILP, we have relaxed

$$\begin{aligned} \min \quad & \sum_{v \in V} x(v)w(v) \\ \text{s.t.} \quad & x(u) + x(v) \geq 1 \quad \forall (u, v) \in E \\ & x(u) \in [0, 1] \quad u \in V \end{aligned}$$

- The fractional LP program can be computed in polynomial time
- All vertex covers are still feasible solution to the LP relaxation
- The optimum to the LP relaxation is a lower bound to the optimum vertex cover

$x(u) \in \{0, 1\}$  we relax integer programming in  $x(u) \in [0, 1]$  in linear programming



# Rounding

→ LP give an lower bound to original problem

We know that optimal solution LP is better than the opt solution of ILP, because is an easier problem

■ Find an optimal solution  $x^* \in R$  to the relaxed problem in polynomial time

■ Round  $x^* \in R$  to a  $\bar{x} \in S$

- ◆  $\bar{x}(u) = 1$  if  $x^*(u) \geq \frac{1}{2}$
- ◆  $\bar{x}(u) = 0$  if  $x^*(u) < \frac{1}{2}$

ROUNDING : is taking the function solution (integer) and translate it. Take opt function solution and say that if  $x^*(u)$  is greater than select it, otherwise no.

■ The solution  $\bar{x}$  is feasible:

$$\forall e = (u, v), \bar{x}(u) + \bar{x}(v) \geq 1$$

since either  $x^*(u) \geq \frac{1}{2}$  or  $x^*(v) \geq \frac{1}{2}$

$\bar{x}(u) + \bar{x}(v)$  is at least 1 for each edge, for this is a valid vertex cover the solution

● Lecture Outline

LP-based Relaxation and Rounding

- Optimization Problems
- Relate to the Optimum
- Use the solution of the relaxed problem
- LP formulation for Vertex Cover
- LP-relaxation for Vertex Cover
- Rounding
- Approximation
- Integrality Gap
- Integrality Gap for Vertex Cover
- LP rounding for Set Cover
- LP formulation for Set Cover
- $f$ -approximation for Set Cover
- More basic techniques

The Primal-Dual Method

Primal-dual Method for Vertex Cover

Set Cover via Dual Lifting

Randomized Rounding



# Approximation

- Lecture Outline

- LP-based Relaxation and Rounding

- Optimization Problems
- Relate to the Optimum
- Use the solution of the relaxed problem
- LP formulation for Vertex Cover
- LP-relaxation for Vertex Cover
- Rounding

- Approximation

- Integrality Gap
- Integrality Gap for Vertex Cover
- LP rounding for Set Cover
- LP formulation for Set Cover
- $f$ -approximation for Set Cover
- More basic techniques

The Primal-Dual Method

Primal-dual Method for Vertex Cover

Set Cover via Dual Lifting

Randomized Rounding

## ■ The solution is 2-approximated:

$$\sum_{u \in U} w(u) \bar{x}(u) \leq 2 \times \sum_{u \in V} w(u) x^*(u) \leq 2 \times OPT,$$

since  $\bar{x}(u) \leq 2 \times x^*(u)$ .

original solution  
 $\rightarrow x^*$  is a relaxation of the optimum

↳ with round  $\geq \frac{1}{2} \rightarrow 1$  we at most multiple all  $x$   
 $< \frac{1}{2} \rightarrow 0$  by 2.



# Integrality Gap

: ratio between an integer solution to the fractional solution

● Lecture Outline

LP-based Relaxation and Rounding

● Optimization Problems

● Relate to the Optimum

● Use the solution of the relaxed problem

● LP formulation for Vertex Cover

● LP-relaxation for Vertex Cover

● Rounding

● Approximation

● Integrality Gap

● Integrality Gap for Vertex Cover

● LP rounding for Set Cover

● LP formulation for Set Cover

●  $f$ -approximation for Set Cover

● More basic techniques

The Primal-Dual Method

Primal-dual Method for Vertex Cover

Set Cover via Dual Lifting

Randomized Rounding

**Definition:** largest ratio on all instances between the optimum integral solution and the optimum relaxed solution.

One cannot hope to achieve an approximation ratio better than the integrality gap of the relaxation

The rounding step should pay a factor at least equal to the integrality gap of the relaxation



# Integrality Gap for Vertex Cover

EXAMPLE

- Lecture Outline

LP-based Relaxation and  
Rounding

- Optimization Problems
- Relate to the Optimum
- Use the solution of the relaxed problem
- LP formulation for Vertex Cover
- LP-relaxation for Vertex Cover
- Rounding

- Approximation

- Integrality Gap

- Integrality Gap for Vertex Cover

- LP rounding for Set Cover
- LP formulation for Set Cover
- $f$ -approximation for Set Cover
- More basic techniques

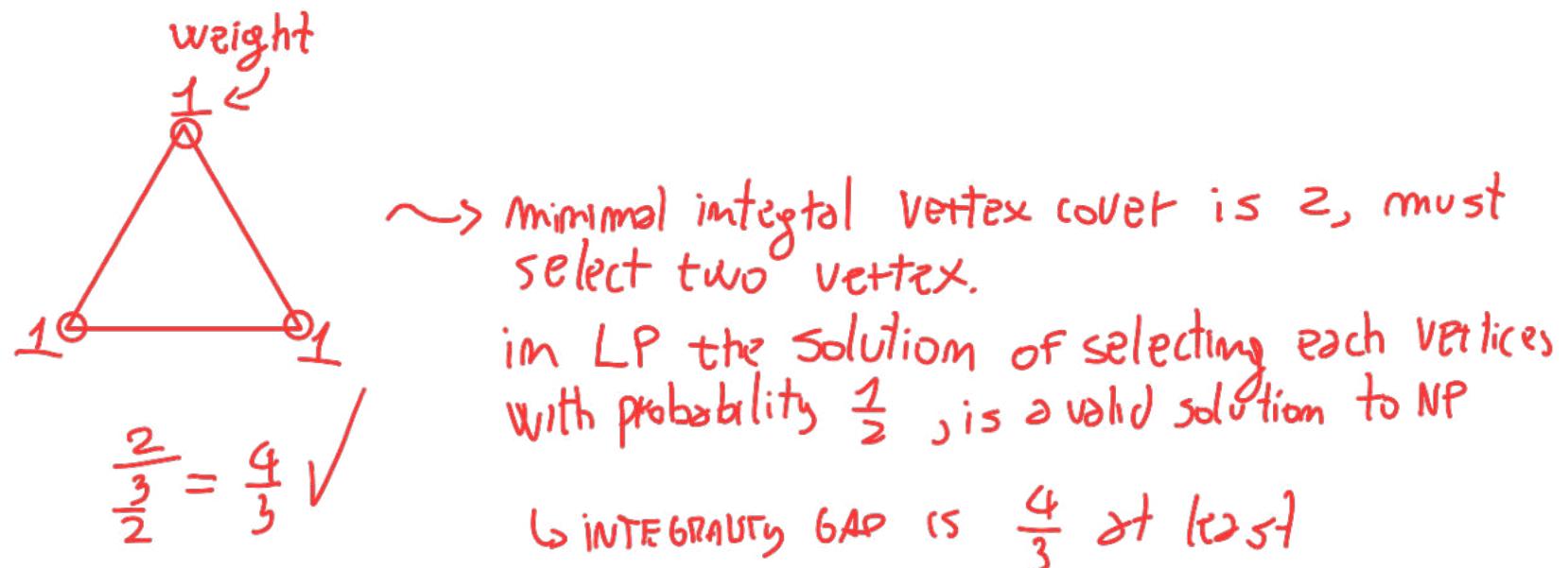
The Primal-Dual Method

Primal-dual Method for Vertex Cover

Set Cover via Dual Lifting

Randomized Rounding

- On a clique graph the optimal vertex cover is of size  $n - 1$
- $x^*(u) = \frac{1}{2}$  is a feasible fractional solution of value  $n/2$
- The integrality gap is equal to  $2 \left(1 - \frac{1}{n}\right) \rightarrow \frac{n-1}{n/2}$
- It is not possible to prove better than 2 approximation for Vertex Cover with this LP





# LP rounding for Set Cover

- Lecture Outline

- LP-based Relaxation and Rounding

- Optimization Problems
- Relate to the Optimum
- Use the solution of the relaxed problem
- LP formulation for Vertex Cover
- LP-relaxation for Vertex Cover
- Rounding
- Approximation
- Integrality Gap
- Integrality Gap for Vertex Cover
- LP rounding for Set Cover
- LP formulation for Set Cover
- $f$ -approximation for Set Cover
- More basic techniques

- The Primal-Dual Method

- Primal-dual Method for Vertex Cover

- Set Cover via Dual Lifting

- Randomized Rounding

## Given:

- $\underline{U}$ : universe of  $n$  elements  $\{e_1, \dots, e_n\}$
- $\underline{\mathcal{S}} = \{S_1, \dots, S_m\}$ : collection of  $m$  subsets of  $U$
- $\underline{c} : S_i \rightarrow \mathbb{R}^+$ : cost function for sets  $\rightsquigarrow c(s_i) \in \mathbb{R}^+$

## Goal:

Find a subcollection of minimum cost that covers  $U$



# LP formulation for Set Cover

cost of the cover we select

$$\begin{aligned} \min \quad & \sum_{S \in \mathcal{S}} c(S)x(S) \\ \text{s.t.} \quad & \sum_{S: e \in S} x(S) \geq 1 \quad \forall e \in U \\ & x(S) \in \{0, 1\} \quad S \in \mathcal{S} \end{aligned}$$

constraint each element must  
be covered

LP relaxation:

$$\begin{aligned} \min \quad & \sum_{S \in \mathcal{S}} c(S)x(S) \\ \text{s.t.} \quad & \sum_{S: e \in S} x(S) \geq 1 \quad \forall e \in U \\ & x(S) \in [0, 1] \quad S \in \mathcal{S} \end{aligned}$$

$x(S)$  define if we take or not  $S_i$

relax problem

● Lecture Outline

LP-based Relaxation and Rounding

● Optimization Problems

● Relate to the Optimum

● Use the solution of the relaxed problem

● LP formulation for Vertex Cover

● LP-relaxation for Vertex Cover

● Rounding

● Approximation

● Integrality Gap

● Integrality Gap for Vertex Cover

● LP rounding for Set Cover

● LP formulation for Set Cover

●  $f$ -approximation for Set Cover

● More basic techniques

The Primal-Dual Method

Primal-dual Method for Vertex Cover

Set Cover via Dual Lifting

Randomized Rounding



# $f$ -approximation for Set Cover

maximal element of the number of set that contain it

Let  $f = \max_{e \in U} |\{S \in \mathcal{S} | e \in S\}|$  ~ element contained in most sets

1. Round to 1 all variables  $x(S) \geq \frac{1}{f}$  sum each set is must be 1,  $\notin$  element in
2. The solution is feasible since every elements appears in at least one set with  $x(S) \geq \frac{1}{f}$
3.  $ALG \leq f \times OPT^{LP}$

For Vertex cover we have  $f = 2$ .

An  $O(\log n)$  approximation algorithm for Weighted Set Cover will follow in this lecture

## Lecture Outline

### LP-based Relaxation and Rounding

- Optimization Problems
- Relate to the Optimum
- Use the solution of the relaxed problem
- LP formulation for Vertex Cover
- LP-relaxation for Vertex Cover
- Rounding
- Approximation
- Integrality Gap
- Integrality Gap for Vertex Cover
- LP rounding for Set Cover
- LP formulation for Set Cover
- $f$ -approximation for Set Cover
- More basic techniques

### The Primal-Dual Method

#### Primal-dual Method for Vertex Cover

#### Set Cover via Dual Lifting

#### Randomized Rounding

# Exercise 1 HW2 - 2021/2022

Santa is worried about his employee relations, since Christmas preparations have led to a lot of overtime. To make sure all the elves are happy, he wants to recruit some of them as *complaint officers*, with weekly meetings to report any complaints or worries to him. His *worker elves*  $W$  are pretty busy already, so Santa wants to task no more than  $k$  elves with this additional workload. Still, Santa wants to make sure that for as many elves  $e \in W$  as possible, at least one of his friends (whose identities he knows) is a complaint officer.

1. Give an intuitive greedy algorithm that outputs  $k$  elves that will serve as compliant officers.
2. Prove that for large numbers of  $k$  the algorithm approximates a solution with ratio no more than  $(1 - \frac{1}{e})$ .

1. We have a graph, vertices are the elves  $W$ , and we have the network we can choose  $K$  elves, want to maximize cover (SET COVER VERTICES)  
A possible greedy algorithm is that: each timestamp we select the next elf that covers the most uncoveted elves.

2. Now analyze performance of the greedy algorithm, starting  
analyze the performance after first step:

- denote OPT the maximal number of elves we can cover with  $K$  elves =  $m$

- denote  $A_1, \dots, A_K$  is the cover that is covered by the  $K$  elves.

We know that  $|A_1 \cup \dots \cup A_K| = m$

After first step we guarantee that: we select in first round  $B_1$

$|B_1| \geq |A_i|$  because is the set that covers the most

$|B_1| \leq \frac{m}{K}$  we cover this at least

At second step:

$$\begin{aligned} |B_2| &\geq \left| \frac{A}{B_1} \right| && \text{depends on first choice, } A \text{ is the union of all } A_i \\ &\geq \frac{|A \setminus B_1|}{K} && \underbrace{|A \setminus B_1 + C \setminus B_1|}_{|A \cup C \setminus B_1|} \geq |A \cup C| - |B_1| \\ &\geq \frac{m - |B_1|}{K} \end{aligned}$$

Let  $d_i$  denote the number of coverage we have:

$$d_1 \geq \frac{m}{K}, \quad d_2 = d_1 + \left(\frac{m-d_1}{K}\right)$$

each time we get at least  $1/K$   $c_i$  complement of  $d_i$

$$c_i = m - d_i$$

$$c_i \leq (c_{i-1} - 1) \left(1 - \frac{1}{K}\right) \text{ geometric series}$$

$$c_K \leq c_0 \left(1 - \frac{1}{K}\right)^K \leq \frac{1}{e} m$$



what remains to cover after 0 to  $m$

Possible solution:

$$m - d_2 = c_1 - d_1 + \beta_2 \leq c_1 - d_1 - \frac{m-d_1}{K} \leq c_1 = m - d_1$$

$$c_1 \leq m - \frac{m}{K} = m \left(1 - \frac{1}{K}\right)$$

$$c_2 = m - d_2 \leq c_1 - \frac{c_1}{K} = c_1 \left(1 - \frac{1}{K}\right)$$

$$d_2 \geq \frac{c_1}{K} = \frac{m-d_1}{K}, \quad c_0 = m \left(1 - \frac{1}{K}\right)^K \leq \frac{1}{e} m$$

## Exercise? - HW2 2021/2022

You are tasked with shipping a number  $n$  of goods  $g_i \in \{1, \dots, n\}$  to a target location  $t_i$ . In your very rural area, the roads are in bad shape and often blocked by trees and the like, and also, there are only three cargo companies you can use for the transport of any good  $g_i$ . You have to order one truck for each good, and know the companies' routes  $P_{i,1}, P_{i,2}$  and  $P_{i,3}$  they would take to all specific locations, where each route consists of a sequence of road segments that it uses. Now since you are aware of the frequent road blockages, and you want not too many of your transports to be obstructed, your aim is to pick the paths/companies for the goods in such a way that a single blocked road can intercept as few shipments as possible. I.e. choose a path  $P_{i,j}$  for every good  $g_i$  such that  $\max_{e \in E} \{|\{P_{i,j} | e \in P_{i,j}\}| \}$  is as small as possible, where  $E$  is the set of all road segments.

- (a) Formulate the problem as an ILP, and relax to an according LP.

trick: additional variable  $Z$   
for describe maximal load

- (b) Give a rounding algorithm to compute a feasible solution starting from the LP optimum that guarantees a 3-approximation to the optimal solution, and prove its approximation guarantee.

(2) set of goods  $G = \{g_1, \dots, g_m\}$ ,  $g_i \in \{1, \dots, m\}$

$P_{i,1}, P_{i,2}, P_{i,3}$  for each good  $i$ , path use multiple edges on graph  
we want don't put too much traffic on the same edge. Want  
the maximal edge load lower of possible.

Formulation of ILP: (we have set of edges  $E$ )

$X_{i,1}, X_{i,2}, X_{i,3}$  are the variables that represent each good  
with the shipment company.  $i = 1, \dots, m$

want define objective and constraints.

OBJECTIVE: min  $Z$  where  $Z$  is the maximal load

CONSTRAINTS:  $\sum_{j,j} X_{ij} \cdot [e \in P_{ij}] \leq Z \quad \forall e \in E$

↳ load on edge  $e$  under solution  $X_{ij}$

if I choose  $X_{ij}$  indicate  
if i use edge  $e$  or not

$$X_{ij} \in \{0, 1\}$$

$$\sum_j X_{ij} = 1 \quad \forall i$$

For relax to LP we change range of  $X_{ij} \in [0, 1]$

We can translate any solution to this LP with value  $Z$  to assignment  
to a company with maximal cost  $Z$  and vice versa

(b) A rounding algorithm to compute a feasible solution is: Want to round  $X_{i,j} \rightarrow$  we found a solution to  $X_{i,j}^*$  that minimize this LP, we found it around  $1/3$ , if it is at least  $1/3$  ( $\geq 1/3$ ) we round to 1, if it is less than  $1/3$  we round to 0 ( $< 1/3 \rightarrow 0$ ). For not violating the fact that we can choose only one company. We take one that is known is at least  $1/3$  and other reduce to 0. This is a feasible solution

The constraint how much can change, i could multiple each  $X_{ij}$  at most to a factor of 3. It is at most 3-time the optimization problem of LP that is a lower bound to ILP 3-time-optimum

## Exercise 4 HW2 – 2020/2021

Suppose that there is a graph  $G = (V, E)$  where the vertices can be of two types: there is a set  $A$  of possible antennae locations and a set  $S$  of cities, with  $V = A \cup S$ . The graph is complete, i.e. for any pair  $\{x, y\} \subseteq V$  we also have that  $(x, y) \in E$ . Moreover, the weight of each edge  $(x, y)$  is given by the distance function  $d : V^2 \rightarrow \mathbb{R}$ ; it satisfies the properties  $d(x, y) = d(y, x) \geq 0$ ,  $d(x, x) = 0$  and  $d(x, z) + d(z, y) \geq d(x, y)$ , for all  $x, y, z \in V$ . Our goal, is to select  $k$  antennae such that the maximum distance from a city to any chosen antenna is minimized. Specifically, we need to find a  $U \subset A$  such that  $|U| = k$  and

$$\max_{x \in S} \min_{y \in U} d(x, y)$$

is minimized.

1. Show that minimizing this objective is NP-hard.
2. Find a 3-approximation algorithm.
3. Show that finding an  $a$ -approximation with  $a < 3$  is also NP-hard. If you do this, there is no need to include an answer to the first sub-question.

We have a graph containing two types of vertices  $A$  (antennae) and  $S$  (cities). We have a metric satisfying a symmetry and triangle inequality. Goal: find location for antennae that minimize maximal distance.  $U \subseteq A$

1.  $A \subseteq E$  antennae,  $S \subseteq E$  cities,  $d : (A \cup S)^2 \rightarrow \mathbb{R}$

Find set  $U$  of size  $k$  that minimize:  $\max_{s \in S} \min_{a \in U} d(s, a)$

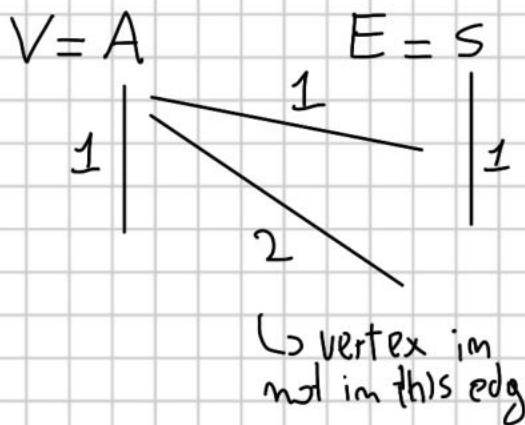
To show that minimizing this objective is NP-hard we use a reduction from Vertex Cover:

In vertex cover we have  $G = (V, E)$   $U \subseteq V$  containing all edges

Edges  $\rightarrow$  Cities

Vertices  $\rightarrow$  Antennae

We define distances in this way

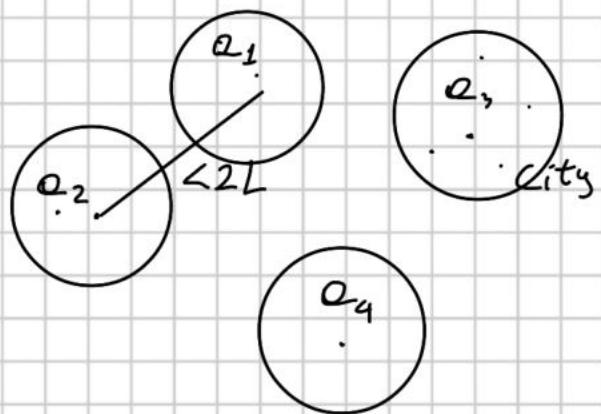


- distances between antennae = 1
- distances between cities = 1
- distances between antennae and edges that contain each other = 1
- $d = 2$  for antennae that doesn't correspond to an edge he touches

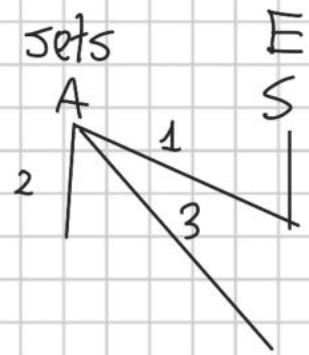
If we have a vertex cover of size  $K$ , then there is a set of antennas of size  $K$  that the minimal distance is 1.

## 2. 3-approximation algorithm:

We have  $A$  antennas and  $S$  cities,  $L$  optimal value



$E$  <sup>set cover</sup>  
 $s_1, \dots, s_k$



greedy algorithm that at each point we a city not covered, we find an antennae within a distance of  $L$  ???



● Lecture Outline

LP-based Relaxation and  
Rounding

---

The Primal-Dual Method

- LP Duality
- LP Duality
- Primal-Dual Formulation
- LP Duality
- LP Duality
- Complementary Slackness  
Conditions
- The Primal-Dual Schema
- The Primal-Dual Schema
- Application of the Primal-Dual  
schema

Primal-dual Method for Vertex  
Cover

---

Set Cover via Dual Lifting

---

Randomized Rounding

---

# The Primal-Dual Method



# LP Duality

min-max  
good for find  
lower bound

linear function objective and constraint,  
we choose value for variables

$$\min \quad 7x_1 + x_2 + 5x_3 \quad \text{objective function}$$

$$\begin{aligned} \text{s.t.} \quad & x_1 - x_2 + 3x_3 \geq 10 \\ & 5x_1 + 2x_2 - x_3 \geq 6 \quad \text{constraint} \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

**Lower bounds on OPT:**

$$7x_1 \geq 1 \wedge x_2 \geq -x_2 \wedge 5x_3 \geq 3x_3$$

$$7x_1 + x_2 + 5x_3 \geq x_1 - x_2 + 3x_3 \geq 10$$

$$7x_1 + x_2 + 5x_3 \geq x_1 - x_2 + 3x_3$$

$$+ 5x_1 + 2x_2 - x_3 \geq 16$$

a better L.B.  
For the OPT

● Lecture Outline

LP-based Relaxation and Rounding

The Primal-Dual Method

- LP Duality
- LP Duality
- Primal-Dual Formulation
- LP Duality
- LP Duality
- Complementary Slackness Conditions
- The Primal-Dual Schema
- The Primal-Dual Schema
- Application of the Primal-Dual schema

Primal-dual Method for Vertex Cover

Set Cover via Dual Lifting

Randomized Rounding



# LP Duality

number of constraint of linear program  
is the number of variables of the  
dual problem, it is the transposition  
of the metrics

## Best lower bound on OPT

**Primal Program**

$$\begin{array}{ll} \min & 7x_1 + x_2 + 5x_3 \\ \text{s.t.} & x_1 - x_2 + 3x_3 \geq 10 \\ & 5x_1 + 2x_2 - x_3 \geq 6 \\ & x_1, x_2, x_3 \geq 0 \end{array}$$

dual problem

**maximize tight part**

$$\max \quad 10y_1 + 6y_2$$

s.t.

$$\begin{array}{lll} y_1 + 5y_2 & \leq & 7 \\ -y_1 + 2y_2 & \leq & 1 \\ 3y_1 - y_2 & \leq & 5 \\ y_1, y_2 & \geq & 0 \end{array}$$

because we want  
keep high at possible

### Lecture Outline

LP-based Relaxation and Rounding

The Primal-Dual Method

● LP Duality

● LP Duality

● Primal-Dual Formulation

● LP Duality

● LP Duality

● Complementary Slackness Conditions

● The Primal-Dual Schema

● The Primal-Dual Schema

● Application of the Primal-Dual schema

Primal-dual Method for Vertex Cover

Set Cover via Dual Lifting

Randomized Rounding



# Primal-Dual Formulation

- Lecture Outline

LP-based Relaxation and  
Rounding

---

The Primal-Dual Method

---

- LP Duality
- LP Duality
- Primal-Dual Formulation
- LP Duality
- LP Duality
- Complementary Slackness  
Conditions
- The Primal-Dual Schema
- The Primal-Dual Schema
- Application of the Primal-Dual  
schema

Primal-dual Method for Vertex  
Cover

---

Set Cover via Dual Lifting

---

Randomized Rounding

---

$$\begin{aligned} \min \quad & \sum_{j=1}^n c_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^n a_{ij} x_j \geq b_i \quad i = 1, \dots, m \\ & x_j \geq 0 \quad j = 1, \dots, n \\ \max \quad & \sum_{i=1}^m b_i y_i \\ \text{s.t.} \quad & \sum_{i=1}^m a_{ij} y_i \leq c_j \quad j = 1, \dots, n \\ & y_i \geq 0 \quad i = 1, \dots, m \end{aligned}$$



# LP Duality

● Lecture Outline

LP-based Relaxation and  
Rounding

The Primal-Dual Method

- LP Duality
- LP Duality
- Primal-Dual Formulation
- LP Duality**
- LP Duality
- Complementary Slackness  
Conditions
- The Primal-Dual Schema
- The Primal-Dual Schema
- Application of the Primal-Dual  
schema

Primal-dual Method for Vertex  
Cover

Set Cover via Dual Lifting

Randomized Rounding

## Strong Duality Theorem

**Theorem 1** If the Primal has finite optimum then the Dual has finite optimum. Let  $x^*$  and  $y^*$  be the primal and the dual optimum solutions. Then

$$\sum_{j=1}^n c_j x_j^* = \sum_{i=1}^m b_i y_i^*$$

primal optimum  
is equal to dual optimum



# LP Duality

dual of the dual is the primal

ILP relax to FLP

## Weak Duality Theorem

**Theorem 2** If  $x$  is feasible for the Primal and  $y$  is feasible for the Dual, then

every dual is a value  
feasible less them  
equal to primal

min-cut      flow

$$\sum_{j=1}^n c_j x_j \geq \sum_{i=1}^m b_i y_i$$

$$\sum_{j=1}^n c_j x_j \geq \sum_{j=1}^n \left( \sum_{i=1}^m a_{ij} y_i \right) x_j = \quad (1)$$

$$= \sum_{i=1}^m \left( \sum_{j=1}^n a_{ij} x_j \right) y_i \geq \sum_{i=1}^m b_i y_i \quad (2)$$

**Corollary 3**  $x$  and  $y$  are optimal for the Primal and the Dual if and only if (1) and (2) hold with equality.

● Lecture Outline

LP-based Relaxation and Rounding

The Primal-Dual Method

- LP Duality
- LP Duality
- Primal-Dual Formulation
- LP Duality
- LP Duality
- Complementary Slackness Conditions
- The Primal-Dual Schema
- The Primal-Dual Schema
- Application of the Primal-Dual schema

Primal-dual Method for Vertex Cover

Set Cover via Dual Lifting

Randomized Rounding



# Complementary Slackness Conditions

● Lecture Outline

LP-based Relaxation and  
Rounding

The Primal-Dual Method

- LP Duality
- LP Duality
- Primal-Dual Formulation
- LP Duality
- LP Duality

● Complementary Slackness  
Conditions

- The Primal-Dual Schema
- The Primal-Dual Schema
- Application of the Primal-Dual  
schema

Primal-dual Method for Vertex  
Cover

Set Cover via Dual Lifting

Randomized Rounding

$x$  and  $y$  are optimal solutions if and only if:

## (1) Primal Complementary Slackness Condition

$$\forall 1 \leq j \leq n \quad \text{either} \quad x_j = 0$$

$$\text{or} \quad \sum_{i=1}^m a_{ij} y_i = c_j$$

Dual constraint is tight

## (2) Dual Complementary Slackness Condition

$$\forall 1 \leq i \leq m \quad \text{either} \quad y_i = 0$$

$$\text{or} \quad \sum_{j=1}^n a_{ij} x_j = b_i$$



# The Primal-Dual Schema

● Lecture Outline

LP-based Relaxation and  
Rounding

The Primal-Dual Method

- LP Duality
- LP Duality
- Primal-Dual Formulation
- LP Duality
- LP Duality
- Complementary Slackness  
Conditions

● The Primal-Dual Schema

- The Primal-Dual Schema
- Application of the Primal-Dual  
schema

Primal-dual Method for Vertex  
Cover

Set Cover via Dual Lifting

Randomized Rounding

## Ensure Primal Complementary Slackness Condition:

$$\forall 1 \leq j \leq n \quad \text{either} \quad x_j = 0$$

$$\text{or} \quad \sum_{i=1}^m a_{ij} y_i = c_j$$

Possible only on  
a problem solvable  
in polytime

## Relax Dual Complementary Slackness Condition

$$\forall 1 \leq i \leq m \quad \text{either} \quad y_i = 0$$

$$\text{or} \quad \sum_{j=1}^n a_{ij} x_j \leq r \times b_i$$

→  
for our problem this  
is not solvable in  
polytime.

•  $t$  will be the approximation  
ratio!

↳ not get the best possible  
value, but not far from  
that  $t$ -distance



# The Primal-Dual Schema

● Lecture Outline

LP-based Relaxation and  
Rounding

The Primal-Dual Method

- LP Duality
- LP Duality
- Primal-Dual Formulation
- LP Duality
- LP Duality
- Complementary Slackness  
Conditions
- The Primal-Dual Schema
- The Primal-Dual Schema**
- Application of the Primal-Dual  
schema

Primal-dual Method for Vertex  
Cover

Set Cover via Dual Lifting

Randomized Rounding

**Theorem 4** *If  $x$  and  $y$  satisfy the conditions of the Primal-Dual schema then*

$$\sum_{j=1}^n c_j x_j \leq r \times \sum_{i=1}^m b_i y_i$$

**Proof:**

$$\begin{aligned} \sum_{j=1}^n c_j x_j &= \sum_{j=1}^n \left( \sum_{i=1}^m a_{ij} y_i \right) x_j = \\ &= \sum_{i=1}^m \left( \sum_{j=1}^n a_{ij} x_j \right) y_i \leq r \times \sum_{i=1}^m b_i y_i \end{aligned}$$

□



# Application of the Primal-Dual schema

■ Primal is a relaxation of a problem  $P$ .

■  $x$  is integral feasible for  $P$

■ It follows:

$$\begin{aligned} \sum_{j=1}^n c_j x_j &\leq r \times \sum_{i=1}^m b_i y_i \leq r \times \sum_{i=1}^m b_i y_i^* \\ &= r \times \sum_{j=1}^n c_j x_j^* \leq r \times OPT \end{aligned}$$

**Primal-dual schema gives  $r$ -approximation algorithm**

● Lecture Outline

LP-based Relaxation and Rounding

The Primal-Dual Method

- LP Duality
- LP Duality
- Primal-Dual Formulation
- LP Duality
- LP Duality
- Complementary Slackness Conditions
- The Primal-Dual Schema
- The Primal-Dual Schema

● Application of the Primal-Dual schema

Primal-dual Method for Vertex Cover

Set Cover via Dual Lifting

Randomized Rounding



● Lecture Outline

LP-based Relaxation and  
Rounding

---

The Primal-Dual Method

---

Primal-dual Method for Vertex  
Cover

- LP formulation for Vertex  
Cover
- The Primal-Dual Algorithm for  
Vertex Cover
- Proof of 2-approximation

Set Cover via Dual Lifting

---

Randomized Rounding

---

# Primal-dual Method for Vertex Cover



# LP formulation for Vertex Cover

neighbour of  $v$

Given a graph  $G = (V, E)$ ,  $\delta(v) = \{e = (v, u) \in E\}$

**Primal:**

$$\begin{aligned} & \min \quad \sum_{v \in V} x(v)w(v) \\ \text{s.t.} \quad & x(u) + x(v) \geq \frac{1}{0} \quad \forall e = (u, v) \in E \\ & x(u) \geq 0 \quad u \in V \end{aligned}$$

$x(u) \leq 1$  in the fractional relaxation can be omitted

**Dual:**

$$\begin{aligned} & \max \quad \sum_{e \in E} y(e) \cdot \underline{1} \\ \text{s.t.} \quad & \sum_{e \in \delta(v)} y(e) \leq w(v) \quad \forall v \in V \\ & y(e) \geq 0 \quad \forall e \in E \end{aligned}$$



# The Primal-Dual Algorithm for Vertex Cover

● Lecture Outline

LP-based Relaxation and  
Rounding

The Primal-Dual Method

Primal-dual Method for Vertex  
Cover

● LP formulation for Vertex  
Cover

● The Primal-Dual Algorithm for  
Vertex Cover

● Proof of 2-approximation

Set Cover via Dual Lifting

Randomized Rounding

1. Increase variable  $y(e)$  for an edge  $e = (u, v)$  until

$$\sum_{e \in \delta(u)} y(e) = w(u) \quad \left( \text{or} \quad \sum_{e \in \delta(v)} y(e) = w(v) \right)$$

2. Set  $x(u) = 1$  ( or  $x(v) = 1$  )

3. Remove all edges adjacent to  $u$  ( or  $v$  )

Repeat until all edges are removed



# Proof of 2-approximation

● Lecture Outline

LP-based Relaxation and  
Rounding

The Primal-Dual Method

Primal-dual Method for Vertex  
Cover

● LP formulation for Vertex  
Cover

● The Primal-Dual Algorithm for  
Vertex Cover

● Proof of 2-approximation

Set Cover via Dual Lifting

Randomized Rounding

Primal complementary slackness condition holds

$$\begin{aligned} \forall u \in V \quad & \text{either} \quad x(u) = 0 \\ & \text{or} \quad \sum_{e \in \delta(u)} y(e) = w(u) \end{aligned}$$

Dual complementary slackness condition is 2-relaxed

$$\begin{aligned} \forall e \in E \quad & \text{either} \quad y(e) = 0 \\ & \text{or} \quad x(u) + x(v) \leq 2 \end{aligned}$$

max is equal to 2,  
not possible if vertex  
is in set

worst case that each  
endpoint is in vertex cover



● Lecture Outline

LP-based Relaxation and  
Rounding

---

The Primal-Dual Method

---

Primal-dual Method for Vertex  
Cover

---

Set Cover via Dual Lifting

- Set Cover
- Greedy algorithm for Set  
Cover
- Analysis of Greedy
- A tight example for Greedy
- Dual-fitting analysis of Greedy  
Set Cover
- LP formulation for Set Cover
- Analysis of Greedy

Randomized Rounding

---

# Set Cover via Dual Lifting



# Set Cover

- Lecture Outline

- [LP-based Relaxation and Rounding](#)

---

- [The Primal-Dual Method](#)

---

- [Primal-dual Method for Vertex Cover](#)

---

- [Set Cover via Dual Lifting](#)

---

- Set Cover

- Greedy algorithm for Set Cover

- Analysis of Greedy

- A tight example for Greedy

- Dual-fitting analysis of Greedy Set Cover

- LP formulation for Set Cover

- Analysis of Greedy

- [Randomized Rounding](#)

---

## Given:

- $U$ : universe of  $n$  elements  $\{e_1, \dots, e_n\}$
- $\mathcal{S} = \{S_1, \dots, S_m\}$ : collection of  $m$  subsets of  $U$
- $c : S_i \rightarrow \mathbb{R}^+$ : cost function for sets

## Goal:

Find a subcollection of minimum cost that covers  $U$

The greedy algorithm achieves an  $O(\log n)$  approximation.



# Greedy algorithm for Set Cover

- Lecture Outline

- LP-based Relaxation and Rounding

- The Primal-Dual Method

- Primal-dual Method for Vertex Cover

- Set Cover via Dual Lifting

- Set Cover
- Greedy algorithm for Set Cover
- Analysis of Greedy
- A tight example for Greedy
- Dual-fitting analysis of Greedy Set Cover
- LP formulation for Set Cover
- Analysis of Greedy

- Randomized Rounding

- Pick at any iteration the most cost-effective set
- $C_i$ : set of elements yet not covered before set  $S_i$  is selected by Greedy
- $c(S)/(C_i \cap S)$ : cost-effectiveness of set  $S$

1.  $C_0 = U$
2. While  $C_i \neq \emptyset$  do

Find the set  $S$  with  $\min \alpha = c(S)/(C_i \cap S)$

Pick set  $S$  and  $\forall e \in S \cap C_i, \text{price}(e) = \alpha$

$C_{i+1} = C_i / S$

3. Output the picked sets



# Analysis of Greedy

● Lecture Outline

LP-based Relaxation and Rounding

The Primal-Dual Method

Primal-dual Method for Vertex Cover

Set Cover via Dual Lifting

● Set Cover

● Greedy algorithm for Set Cover

● Analysis of Greedy

● A tight example for Greedy

● Dual-fitting analysis of Greedy Set Cover

● LP formulation for Set Cover

● Analysis of Greedy

Randomized Rounding

Assume  $U$  covered by Greedy in order  $\{e_1, \dots, e_n\}$

**Lemma 5**  $price(e_j) \leq \frac{OPT}{n-j+1}$

**Proof:**

- At any iteration the optimal solution covers  $C_i$  at cost at most  $OPT$
- There exists a set of  $OPT$  with  $\alpha \leq \frac{OPT}{C_i}$
- When  $e_j$  is covered at iteration  $i$ ,  $C_i \geq n - j + 1$
- Since  $e_j$  is covered by the most cost-effective set:

$$price(e_j) \leq \frac{OPT}{C_i} \leq \frac{OPT}{n - j + 1}$$

□

## Theorem 6

$$ALG = \sum_{j=1}^n price(e_j) \leq OPT \times \sum_{j=1}^n \frac{1}{n - j + 1} = OPT \times H_n$$



# A tight example for Greedy

- Lecture Outline

- [LP-based Relaxation and Rounding](#)

---

- [The Primal-Dual Method](#)

---

- [Primal-dual Method for Vertex Cover](#)

---

- [Set Cover via Dual Lifting](#)

---

- Set Cover
- Greedy algorithm for Set Cover
- Analysis of Greedy
- A tight example for Greedy
- Dual-fitting analysis of Greedy Set Cover
- LP formulation for Set Cover
- Analysis of Greedy

- [Randomized Rounding](#)

---

Greedy outputs all singleton sets with cost

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

Optimum cost is  $1 + \epsilon$ .



# Dual-fitting analysis of Greedy Set Cover

- Lecture Outline

- [LP-based Relaxation and Rounding](#)

- [The Primal-Dual Method](#)

- [Primal-dual Method for Vertex Cover](#)

- [Set Cover via Dual Lifting](#)

- Set Cover
- Greedy algorithm for Set Cover
- Analysis of Greedy
- A tight example for Greedy

- Dual-fitting analysis of Greedy Set Cover

- LP formulation for Set Cover
- Analysis of Greedy

- [Randomized Rounding](#)

## Dual-fitting method:

- Show that the integral solution is fully paid by an unfeasible dual solution
- The dual solution can be made feasible by scaling down each variable by a factor  $f$
- 

$$ALG = DUAL^{unf} = f \times DUAL^{feas} \leq f \times OPT^{LP} \leq f \times OPT$$

- Alternative to argue about complementary slackness conditions



# LP formulation for Set Cover

● Lecture Outline

LP-based Relaxation and  
Rounding

The Primal-Dual Method

Primal-dual Method for Vertex  
Cover

Set Cover via Dual Lifting

- Set Cover
- Greedy algorithm for Set Cover
- Analysis of Greedy
- A tight example for Greedy
- Dual-fitting analysis of Greedy Set Cover
- LP formulation for Set Cover
- Analysis of Greedy

Randomized Rounding

$$\begin{aligned} \text{Primal: } & \min \sum_{S \in \mathcal{S}} c(S)x(S) \\ \text{s.t. } & \sum_{S: e \in S} x(S) \geq 1 \quad \forall e \in U \\ & x(S) \geq 0 \quad S \in \mathcal{S} \end{aligned}$$

Constraint  $x(S) \leq 1$  can be omitted

$$\begin{aligned} \text{Dual: } & \max \sum_{e \in U} y(e) \\ \text{s.t. } & \sum_{e \in S} y(e) \leq c(S) \quad \forall e \in U \\ & y(e) \geq 0 \quad \forall e \in U \end{aligned}$$



# Analysis of Greedy

● Lecture Outline

LP-based Relaxation and  
Rounding

The Primal-Dual Method

Primal-dual Method for Vertex  
Cover

Set Cover via Dual Lifting

- Set Cover
- Greedy algorithm for Set Cover
- Analysis of Greedy
- A tight example for Greedy
- Dual-fitting analysis of Greedy Set Cover
- LP formulation for Set Cover
- Analysis of Greedy

Randomized Rounding

Interpret the Greedy solution as an unfeasible dual:

$$y(e) = \text{price}(e), \text{ALG} = \sum_{e \in U} y(e)$$

**Lemma 7**  $y'(e) = \frac{\text{price}(e)}{H_n}$  is dual feasible.

■ Consider any set  $S = \{e_1, \dots, e_k\}$  with elements numbered by the order they are covered by Greedy.

■  $S$  can cover  $e_i$  at price  $\leq \frac{c(S)}{k-i+1}$  when  $i$  is covered.

■ Since Greedy picks the most cost-effective set:

$$\text{price}(e_i) \leq \frac{c(S)}{k-i+1}$$

■ Dual variables  $y'(e) \leq \frac{1}{H_n} \frac{c(S)}{k-i+1}$

■ Dual constraint for set  $S$ :

$$\sum_{i=1}^k y'(e_i) \leq \frac{c(S)}{H_n} \left( \frac{1}{k} + \frac{1}{k-1} + \dots + 1 \right) = c(S) \frac{H_k}{H_n} \leq c(S)$$



● Lecture Outline

LP-based Relaxation and  
Rounding

---

The Primal-Dual Method

---

Primal-dual Method for Vertex  
Cover

---

Set Cover via Dual Lifting

---

Randomized Rounding

- Randomized Rounding
- Randomized Rounding for Set Cover
- Make the solution feasible
- Make the solution feasible

# Randomized Rounding for Set Cover



# Randomized Rounding

● Lecture Outline

LP-based Relaxation and  
Rounding

The Primal-Dual Method

Primal-dual Method for Vertex  
Cover

Set Cover via Dual Lifting

Randomized Rounding

● Randomized Rounding  
● Randomized Rounding for Set  
Cover  
● Make the solution feasible  
● Make the solution feasible

- Interpret primal variables in the fractional relaxation as probabilities
- Obtain a primal solution by setting variables to 1 independently with probability equal to the fractional value
- The expected cost of the solution is equal to the fractional optimum ..... but
- **Solution may not be feasible**
- Repeat as many times as needed to enforce feasibility



# Randomized Rounding for Set Cover

- Lecture Outline

- [LP-based Relaxation and Rounding](#)

---

- [The Primal-Dual Method](#)

- [Primal-dual Method for Vertex Cover](#)

---

- [Set Cover via Dual Lifting](#)

- [Randomized Rounding](#)

- Randomized Rounding

- Randomized Rounding for Set Cover

- Make the solution feasible

- Make the solution feasible

$$\begin{aligned} \min \quad & \sum_{S \in \mathcal{S}} c(S)x(S) \\ \text{s.t.} \quad & \sum_{S: e \in S} x(S) \geq 1 \quad \forall e \in U \\ & x(S) \in [0, 1] \quad S \in \mathcal{S} \end{aligned}$$

- Pick set  $S$  with probability  $p(S) = x^*(S)$
- $E[ALG] = \sum_{S \in \mathcal{S}} c(S)p(S) = \sum_{S \in \mathcal{S}} c(S)x^*(S) = OPT^{LP} \leq OPT$
- Is the solution feasible?



# Make the solution feasible

● Lecture Outline

LP-based Relaxation and  
Rounding

The Primal-Dual Method

Primal-dual Method for Vertex  
Cover

Set Cover via Dual Lifting

Randomized Rounding

● Randomized Rounding  
● Randomized Rounding for Set  
Cover

● Make the solution feasible

● Make the solution feasible

- For an element  $a$ :  $\{S_1, \dots, S_k\} = \{S \in \mathcal{S} : a \in S\}$

$$\begin{aligned} \Pr[a \text{ is covered}] &= 1 - (1 - p(S_1)) \times \dots \times (1 - p(S_k)) \\ &\geq 1 - \left(1 - \frac{1}{k}\right)^k \\ &\geq 1 - \frac{1}{e} \end{aligned}$$

since  $p(S_1) + \dots + p(S_k) \geq 1$

- Each element  $a \in U$  is covered with  $Pb \geq 1 - \frac{1}{e}$



# Make the solution feasible

- Lecture Outline

---

- LP-based Relaxation and Rounding

---

- The Primal-Dual Method

---

- Primal-dual Method for Vertex Cover

---

- Set Cover via Dual Lifting

---

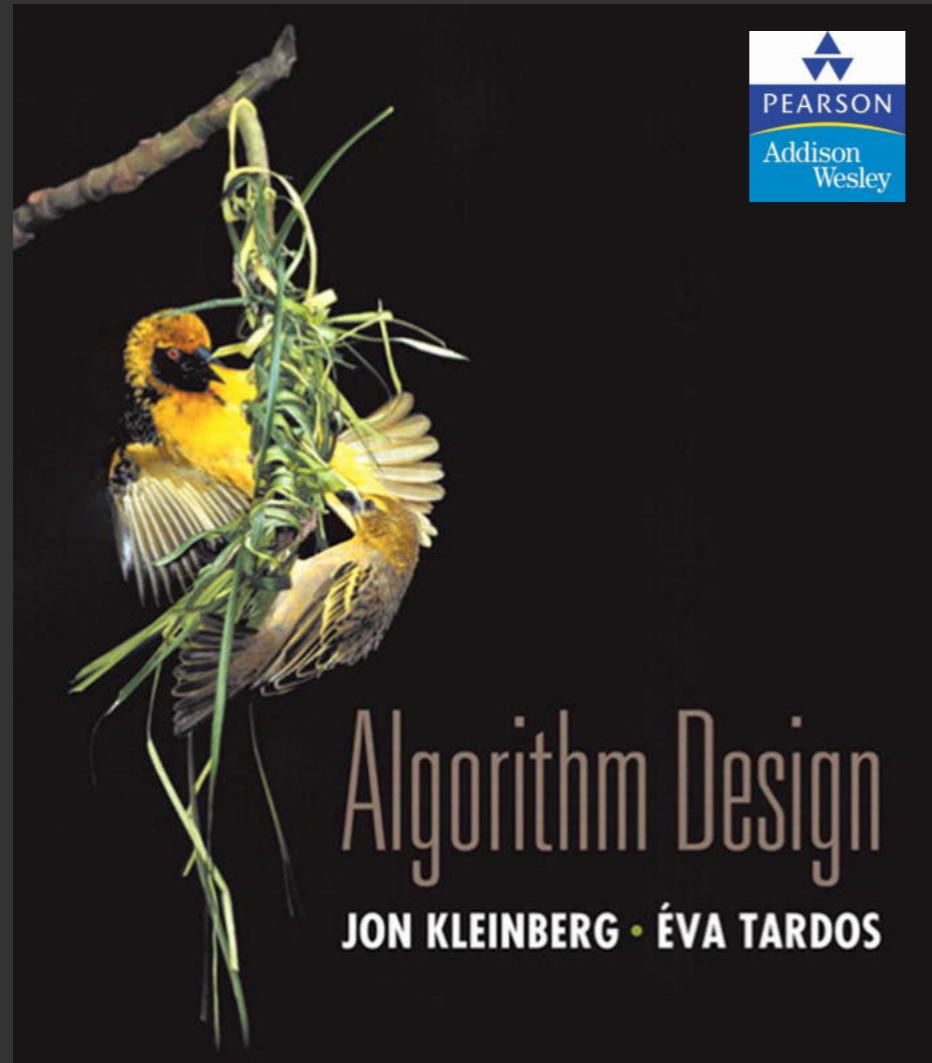
- Randomized Rounding

- Randomized Rounding
- Randomized Rounding for Set Cover
- Make the solution feasible
- Make the solution feasible

- Pick  $d \log n$  subcollections  $C' = C_1 \cup \dots \cup C_{d \log n}$  with  $d$  such that:

$$\Pr[a \text{ not covered}] \leq \left(\frac{1}{e}\right)^{d \log n} \leq \frac{1}{4n}$$

- $E[COST(C')] \leq d \times \log n OPT^{LP}$
- $\Pr[COST(C') \geq 4d \times \log n OPT^{LP}] \leq \frac{1}{4}$
- $\Pr[C' \text{ not feasible}] \leq n \times \frac{1}{4n} \leq \frac{1}{4}$
- $\Pr[COST(C') \leq 4d \times \log n OPT^{LP} \text{ AND } C' \text{ feasible}] \geq \frac{1}{2}$
- Expected[number of repetitions] = 2



## 13. RANDOMIZED ALGORITHMS

---

- ▶ *contention resolution*
- ▶ *global min cut*
- ▶ *linearity of expectation*
- ▶ *max 3-satisfiability*
- ▶ *universal hashing*
- ▶ *Chernoff bounds*
- ▶ *load balancing*

Lecture slides by Kevin Wayne

Copyright © 2005 Pearson–Addison Wesley

<http://www.cs.princeton.edu/~wayne/kleinberg-tardos>

# Randomization

---

## Algorithmic design patterns.

- Greedy.
- Divide-and-conquer.
- Dynamic programming.
- Network flow.
- Randomization.

in practice, access to a pseudo-random number generator



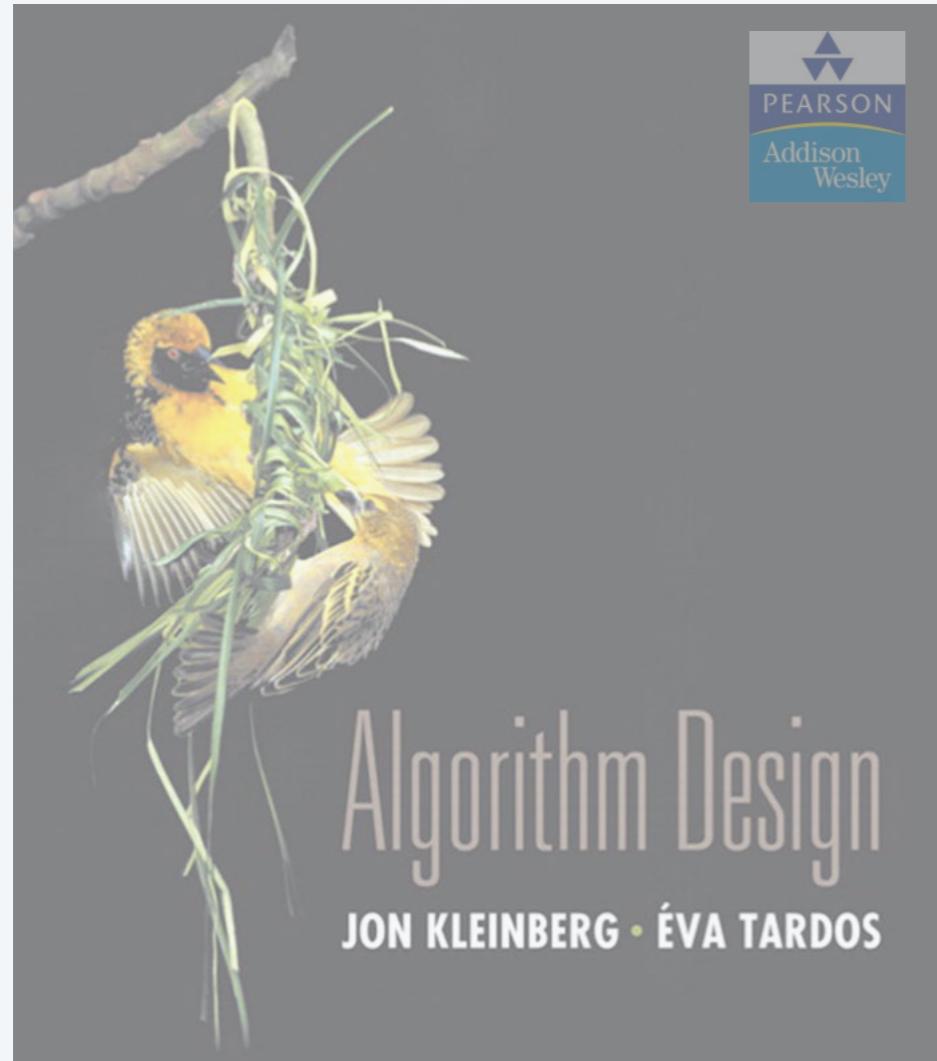
Randomization. Allow fair coin flip in unit time.

---

Why randomize? Can lead to simplest, fastest, or only known algorithm for a particular problem.

---

Ex. Symmetry-breaking protocols, graph algorithms, quicksort, hashing, load balancing, Monte Carlo integration, cryptography.



## 13. RANDOMIZED ALGORITHMS

---

- ▶ *contention resolution*
- ▶ *global min cut*
- ▶ *linearity of expectation*
- ▶ *max 3-satisfiability*
- ▶ *universal hashing*
- ▶ *Chernoff bounds*
- ▶ *load balancing*

## Contention resolution in a distributed system

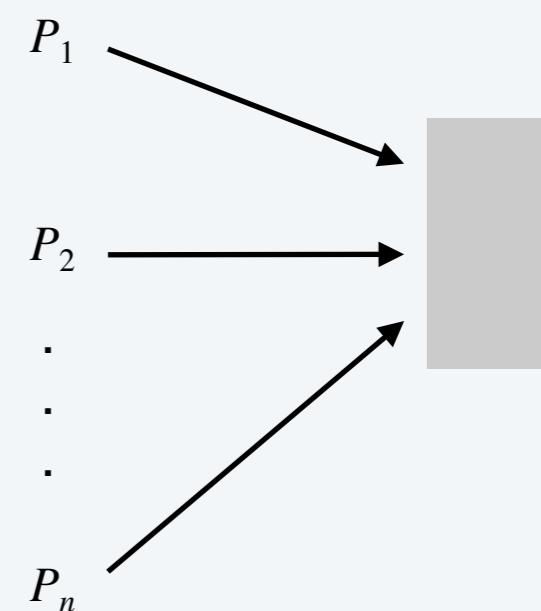
---

**Contention resolution.** Given  $n$  processes  $P_1, \dots, P_n$ , each competing for access to a shared database. If two or more processes access the database simultaneously, all processes are locked out. Devise protocol to ensure all processes get through on a regular basis.

**Restriction.** Processes can't communicate.

**Challenge.** Need symmetry-breaking paradigm.

Ethernet protocol is based  
on contention resolution



## Contention resolution: randomized protocol

processes are independent

**Protocol.** Each process requests access to the database at time  $t$  with probability  $p = 1/n$ . ,  $n$  number of processes

i success on time t

**Claim.** Let  $S[i, t]$  = event that process  $i$  succeeds in accessing the database at time  $t$ . Then  $1/(e \cdot n) \leq \Pr[S(i, t)] \leq 1/(2n)$ .

process i succeed, other  $n-1$  fails !

**Pf.** By independence,  $\Pr[S(i, t)] = p (1-p)^{n-1}$ .

$$\Pr[S(i, t)] = \underbrace{p}_{\text{process } i \text{ requests access}} \underbrace{(1-p)^{n-1}}_{\text{none of remaining } n-1 \text{ processes request access}}$$

- Setting  $p = 1/n$ , we have  $\Pr[S(i, t)] = 1/n \underbrace{(1 - 1/n)^{n-1}}_{\text{value that maximizes } \Pr[S(i, t)]}$ . ■

$$\frac{1}{e} < \left(1 - \frac{1}{n}\right)^{n-1} < \frac{1}{2}$$

**Useful facts from calculus.** As  $n$  increases from 2, the function:

- $(1 - 1/n)^{n-1}$  converges monotonically from  $1/4$  up to  $1/e$ .
- $(1 - 1/n)^{n-1}$  converges monotonically from  $1/2$  down to  $1/e$ .

## Contention resolution: randomized protocol

$$S[i,t] \geq \frac{1}{en} \quad F[i,t] \geq 1 - \frac{1}{en}$$

**Claim.** The probability that process  $i$  fails to access the database in  $en$  rounds is at most  $1/e$ . After  $e \cdot n(c \ln n)$  rounds, the probability  $\leq n^{-c}$ .

↳ probability of fail drop to a very small value

**Pf.** Let  $F[i, t] =$  event that process  $i$  fails to access database in rounds 1 through  $t$ . By independence and previous claim, we have

$$\Pr[F[i, t]] \leq (1 - 1/(en))^t.$$

- Choose  $t = \lceil e \cdot n \rceil$ :

$$\Pr[F(i, t)] \leq \left(1 - \frac{1}{en}\right)^{\lceil en \rceil} \leq \left(1 - \frac{1}{en}\right)^{en} \leq \frac{1}{e}$$

- Choose  $t = \lceil e \cdot n \rceil \lceil c \ln n \rceil$ :

$$\Pr[F(i, t)] \leq \left(\frac{1}{e}\right)^{c \ln n} = n^{-c}$$

Probability of failure drop, with  $e \cdot n(c \ln n)$ , to a polynomial

## Contention resolution: randomized protocol

**Claim.** The probability that all processes succeed within  $2e \cdot n \ln n$  rounds is  $\geq 1 - 1/n$ .

**Pf.** Let at least one of the  $n$  processes fails to access database in any of the rounds 1 through  $t$ .

$$\Pr[F[t]] = \Pr\left[\bigcup_{i=1}^n F[i,t]\right] \leq \sum_{i=1}^n \Pr[F[i,t]] \leq n\left(1 - \frac{1}{en}\right)^t$$

union bound    previous slide

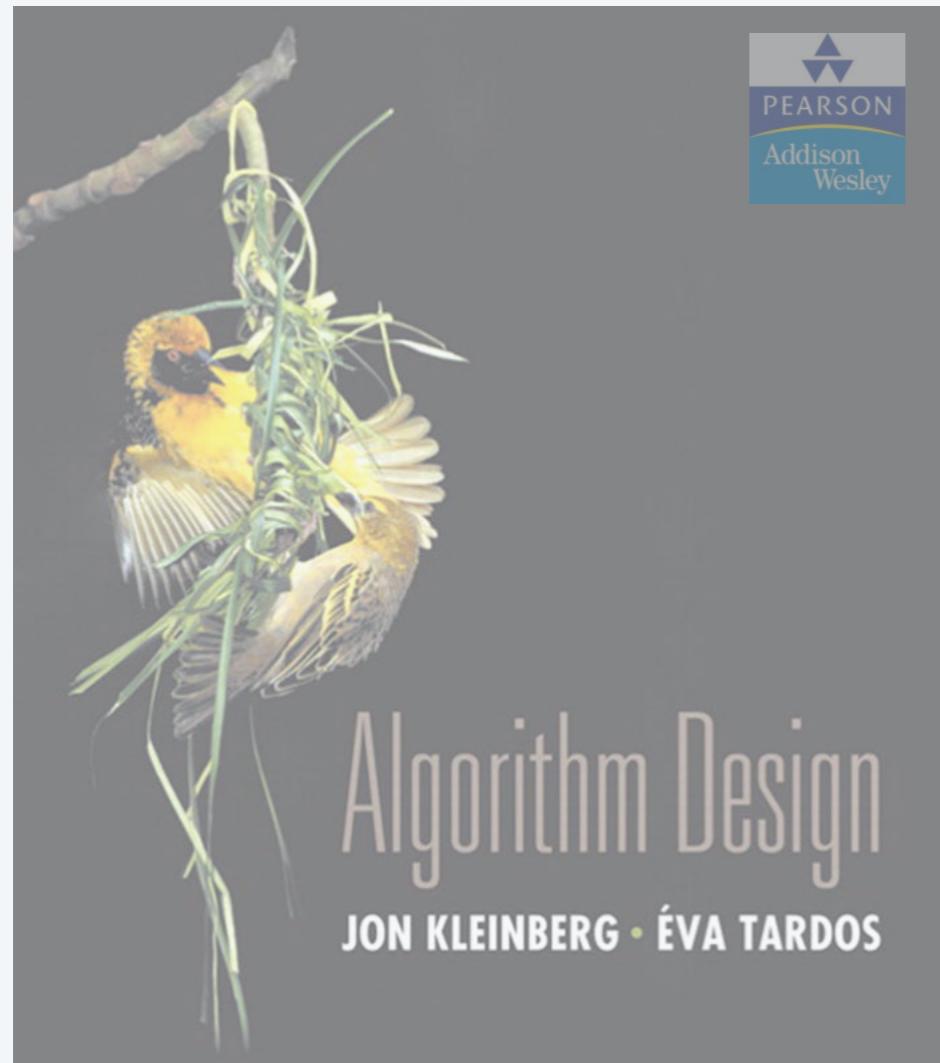


- Choosing  $t = 2 \lceil en \rceil \lceil c \ln n \rceil$  yields  $\Pr[F[t]] \leq n \cdot n^{-2} = 1/n$ . ■

$$\Pr[S(i,+)] = \frac{1}{\text{all}} - \frac{1}{n}$$

**Union bound.** Given events  $E_1, \dots, E_n$ ,

$$\Pr\left[\bigcup_{i=1}^n E_i\right] \leq \sum_{i=1}^n \Pr[E_i]$$

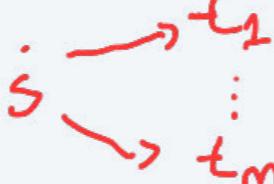


## 13. RANDOMIZED ALGORITHMS

---

- ▶ *contention resolution*
- ▶ ***global min cut***
- ▶ *linearity of expectation*
- ▶ *max 3-satisfiability*
- ▶ *universal hashing*
- ▶ *Chernoff bounds*
- ▶ *load balancing*

## Global minimum cut

 **multiple target**

**Global min cut.** Given a connected, undirected graph  $G = (V, E)$ , find a cut  $(A, B)$  of minimum cardinality.

**Applications.** Partitioning items in a database, identify clusters of related documents, network reliability, network design, circuit design, TSP solvers.

## Network flow solution.

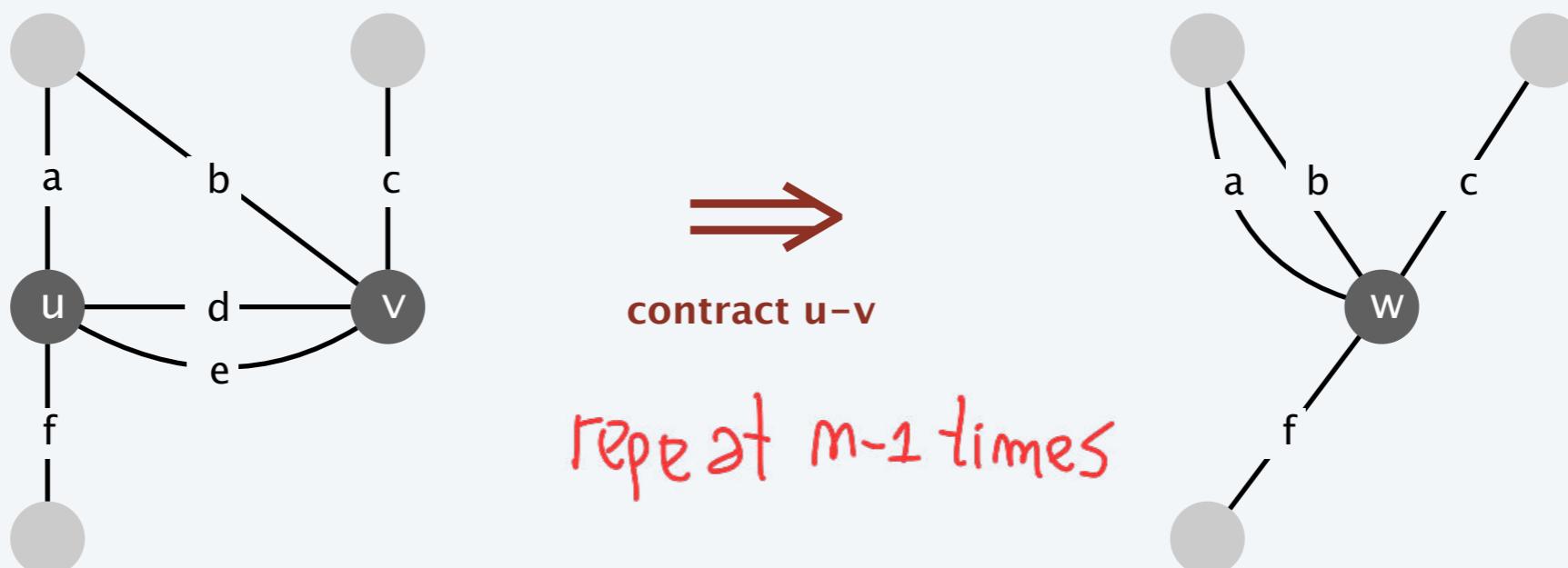
- Replace every edge  $(u, v)$  with two antiparallel edges  $(u, v)$  and  $(v, u)$ .
- Pick some vertex  $s$  and compute  $\min s-v$  cut separating  $s$  from each other vertex  $v \in V$ .

**False intuition.** Global min-cut is harder than  $\min s-t$  cut.

# Contraction algorithm

Contraction algorithm. [Karger 1995]

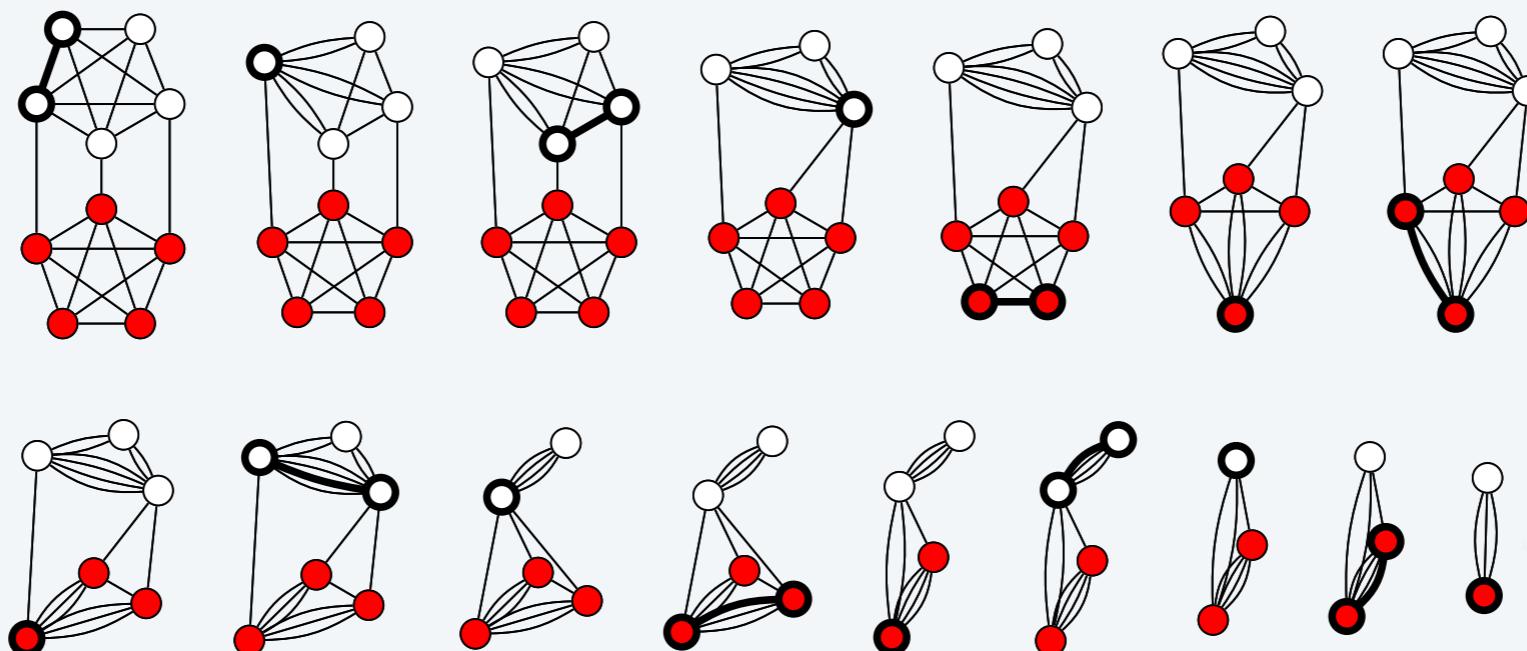
- Pick an edge  $e = (u, v)$  uniformly at random.
- Contract edge  $e$ .
  - replace  $u$  and  $v$  by single new super-node  $w$
  - preserve edges, updating endpoints of  $u$  and  $v$  to  $w$
  - keep parallel edges, but delete self-loops
- Repeat until graph has just two nodes  $u_1$  and  $v_1$ .
- Return the cut (all nodes that were contracted to form  $v_1$ ).



# Contraction algorithm

Contraction algorithm. [Karger 1995]

- Pick an edge  $e = (u, v)$  uniformly at random.
- Contract edge  $e$ .
  - replace  $u$  and  $v$  by single new super-node  $w$
  - preserve edges, updating endpoints of  $u$  and  $v$  to  $w$
  - keep parallel edges, but delete self-loops
- Repeat until graph has just two nodes  $u_1$  and  $v_1$ .
- Return the cut (all nodes that were contracted to form  $v_1$ ).



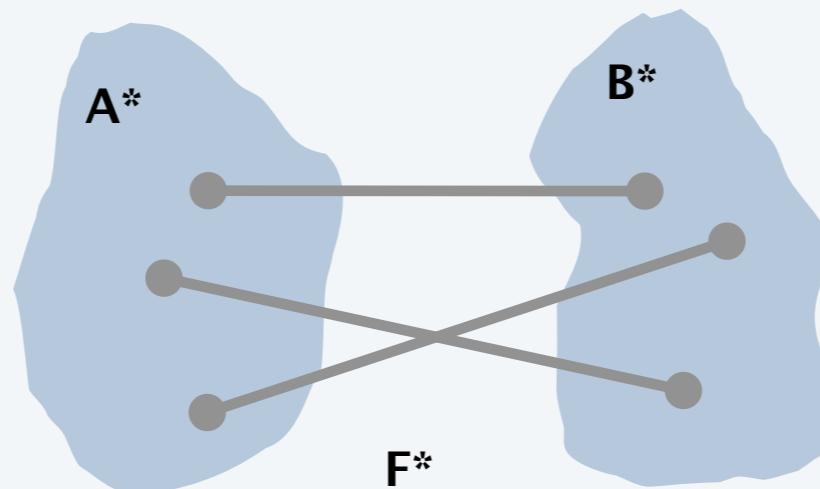
# Contraction algorithm

---

**Claim.** The contraction algorithm returns a min cut with prob  $\geq 2 / n^2$ .

**Pf.** Consider a global min-cut  $(A^*, B^*)$  of  $G$ .

- Let  $F^*$  be edges with one endpoint in  $A^*$  and the other in  $B^*$ .
- Let  $k = |F^*| = \text{size of min cut}$ .
- In first step, algorithm contracts an edge in  $F^*$  probability  $k / |E|$ .
- Every node has degree  $\geq k$  since otherwise  $(A^*, B^*)$  would not be a min-cut  $\Rightarrow |E| \geq \frac{1}{2} k n \Leftrightarrow k / |E| \leq 2 / n$ .
- Thus, algorithm contracts an edge in  $F^*$  with probability  $\leq 2 / n$ .



# Contraction algorithm

---

**Claim.** The contraction algorithm returns a min cut with prob  $\geq 2/n^2$ .

**Pf.** Consider a global min-cut  $(A^*, B^*)$  of  $G$ .

- Let  $F^*$  be edges with one endpoint in  $A^*$  and the other in  $B^*$ .
- Let  $k = |F^*| = \text{size of min cut}$ .
- Let  $G'$  be graph after  $j$  iterations. There are  $n' = n - j$  supernodes.
- Suppose no edge in  $F^*$  has been contracted. The min-cut in  $G'$  is still  $k$ .
- Since value of min-cut is  $k$ ,  $|E'| \geq \frac{1}{2}kn' \Leftrightarrow k/|E'| \leq 2/n'$ .
- Thus, algorithm contracts an edge in  $F^*$  with probability  $\leq 2/n'$ .
- Let  $E_j$  = event that an edge in  $F^*$  is not contracted in iteration  $j$ .

$$\begin{aligned}\Pr[E_1 \cap E_2 \cap \dots \cap E_{n-2}] &= \Pr[E_1] \times \Pr[E_2 | E_1] \times \dots \times \Pr[E_{n-2} | E_1 \cap E_2 \cap \dots \cap E_{n-3}] \\ &\geq \left(1 - \frac{2}{n}\right) \left(1 - \frac{2}{n-1}\right) \dots \left(1 - \frac{2}{4}\right) \left(1 - \frac{2}{3}\right) \\ &= \left(\frac{n-2}{n}\right) \left(\frac{n-3}{n-1}\right) \dots \left(\frac{2}{4}\right) \left(\frac{1}{3}\right) \\ &= \frac{2}{n(n-1)} \\ &\geq \frac{2}{n^2}\end{aligned}$$

# Contraction algorithm

---

**Amplification.** To amplify the probability of success, run the contraction algorithm many times.

with independent random choices,

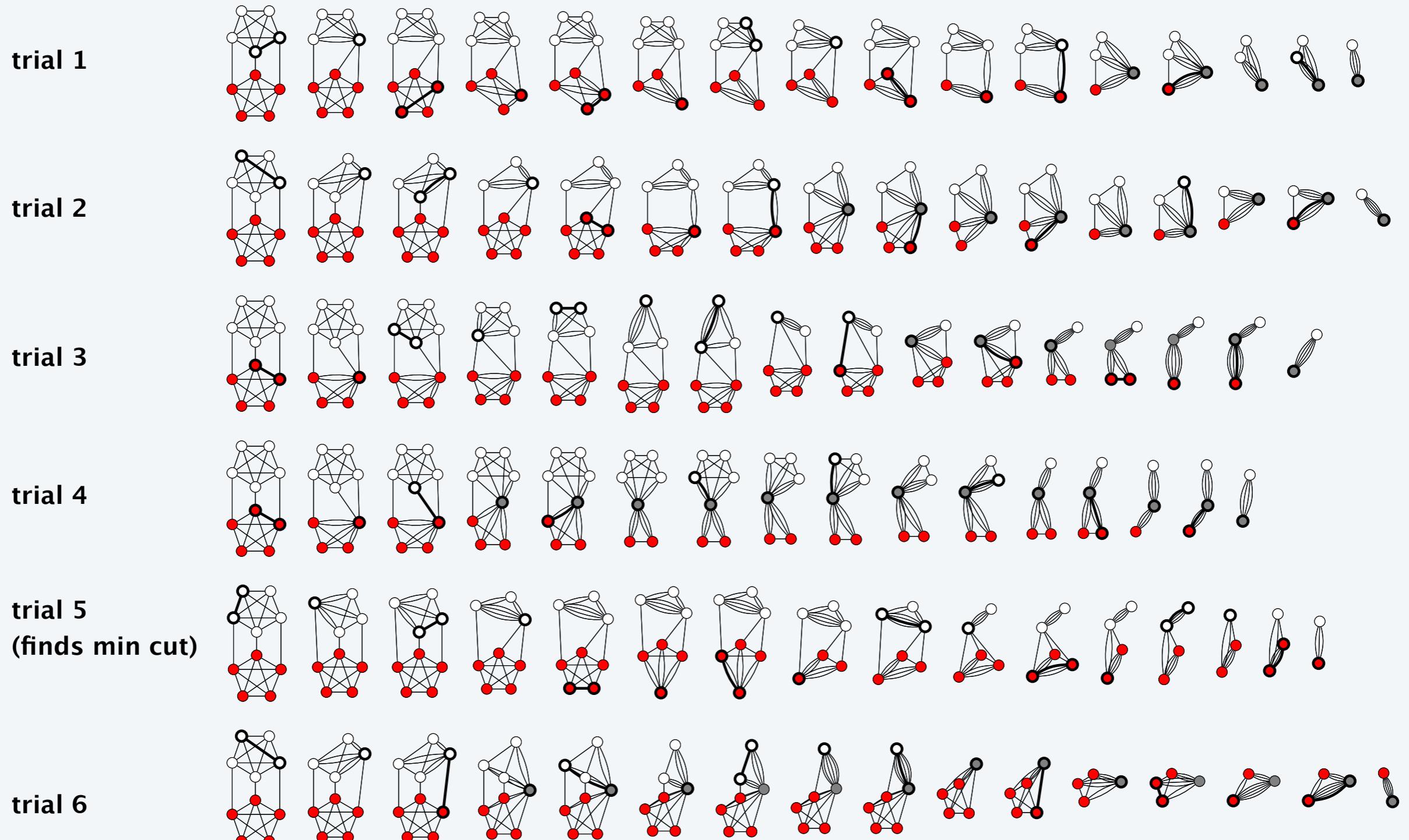
**Claim.** If we repeat the contraction algorithm  $n^2 \ln n$  times, then the probability of failing to find the global min-cut is  $\leq 1/n^2$ .

**Pf.** By independence, the probability of failure is at most

$$\left(1 - \frac{2}{n^2}\right)^{n^2 \ln n} = \left[\left(1 - \frac{2}{n^2}\right)^{\frac{1}{2}n^2}\right]^{2 \ln n} \leq \left(e^{-1}\right)^{2 \ln n} = \frac{1}{n^2}$$

$$(1 - 1/x)^x \leq 1/e$$

# Contraction algorithm: example execution



## Global min cut: context

---

**Remark.** Overall running time is slow since we perform  $\Theta(n^2 \log n)$  iterations and each takes  $\Omega(m)$  time.

---

**Improvement.** [Karger–Stein 1996]  $O(n^2 \log^3 n)$ .

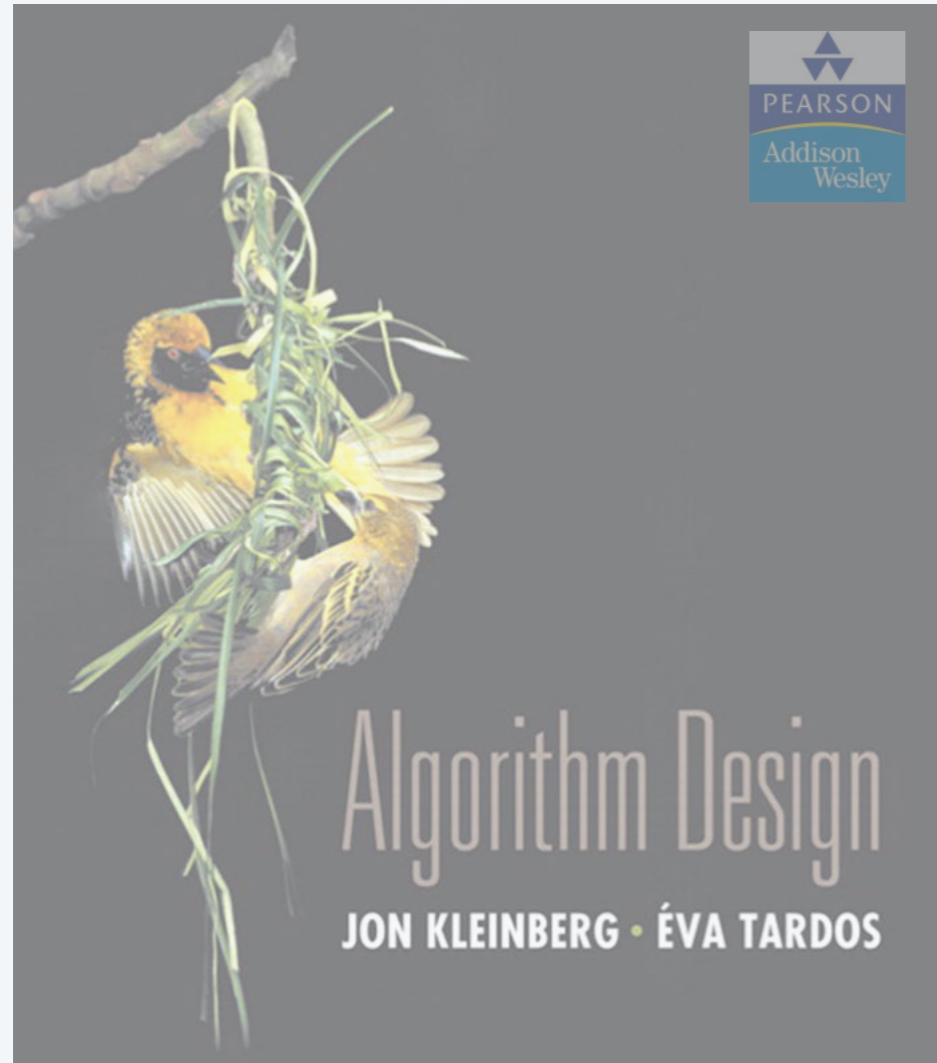
- Early iterations are less risky than later ones: probability of contracting an edge in min cut hits 50% when  $n / \sqrt{2}$  nodes remain.
  - Run contraction algorithm until  $n / \sqrt{2}$  nodes remain.
  - Run contraction algorithm twice on resulting graph and return best of two cuts.
- 

**Extensions.** Naturally generalizes to handle positive weights.

**Best known.** [Karger 2000]  $O(m \log^3 n)$ .



faster than best known max flow algorithm or  
deterministic global min cut algorithm



## 13. RANDOMIZED ALGORITHMS

---

- ▶ *contention resolution*
- ▶ *global min cut*
- ▶ ***linearity of expectation***
- ▶ *max 3-satisfiability*
- ▶ *universal hashing*
- ▶ *Chernoff bounds*
- ▶ *load balancing*

## Expectation

---

**Expectation.** Given a discrete random variable  $X$ , its expectation  $E[X]$  is defined by:

$$E[X] = \sum_{j=0}^{\infty} j \Pr[X = j]$$

---

**Waiting for a first success.** Coin is heads with probability  $p$  and tails with probability  $1-p$ . How many independent flips  $X$  until first heads?

$$E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j] = \sum_{j=0}^{\infty} j (1-p)^{j-1} p = \frac{p}{1-p} \sum_{j=0}^{\infty} j (1-p)^j = \frac{p}{1-p} \cdot \frac{1-p}{p^2} = \frac{1}{p}$$

j – 1 tails      1 head

## Expectation: two properties

Dependent or independent

Useful property. If  $X$  is a 0/1 random variable,  $E[X] = \Pr[X = 1]$ .

Pf.

$$E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j] = \sum_{j=0}^1 j \cdot \Pr[X = j] = \Pr[X = 1]$$

---

Linearity of expectation. Given two random variables  $X$  and  $Y$  defined over the same probability space,  $E[X + Y] = E[X] + E[Y]$ .

not necessarily independent



Benefit. Decouples a complex calculation into simpler pieces.

## Guessing cards

---

**Game.** Shuffle a deck of  $n$  cards; turn them over one at a time;  
try to guess each card.

---

**Memoryless guessing.** No psychic abilities; can't even remember what's  
been turned over already. Guess a card from full deck uniformly at random.

---

**Claim.** The expected number of correct guesses is 1.

**Pf.** [surprisingly effortless using linearity of expectation]

- Let  $X_i = 1$  if  $i^{th}$  prediction is correct and 0 otherwise.
  - Let  $X = \text{number of correct guesses} = X_1 + \dots + X_n$ .
  - $E[X_i] = \Pr[X_i = 1] = 1/n$ .
  - $E[X] = E[X_1] + \dots + E[X_n] = 1/n + \dots + 1/n = 1$ . ▀
- 



**linearity of expectation**

# Guessing cards

---

**Game.** Shuffle a deck of  $n$  cards; turn them over one at a time; try to guess each card.

**Guessing with memory.** Guess a card uniformly at random from cards not yet seen.

**Claim.** The expected number of correct guesses is  $\Theta(\log n)$ .

**Pf.**

- Let  $X_i = 1$  if  $i^{th}$  prediction is correct and 0 otherwise.
- Let  $X = \text{number of correct guesses} = X_1 + \dots + X_n$ .
- $E[X_i] = \Pr[X_i = 1] = 1 / (n - (i - 1))$ .
- $E[X] = E[X_1] + \dots + E[X_n] = 1/n + \dots + 1/2 + 1/1 = H(n)$ . ▀

↑  
linearity of expectation

↑  
 $\ln(n+1) < H(n) < 1 + \ln n$

## Coupon collector

**Coupon collector.** Each box of cereal contains a coupon. There are  $n$  different types of coupons. Assuming all boxes are equally likely to contain each coupon, how many boxes before you have  $\geq 1$  coupon of each type?

**Claim.** The expected number of steps is  $\Theta(n \log n)$ .

**Pf.**

- Phase  $j$  = time between  $j$  and  $j + 1$  distinct coupons.
- Let  $X_j$  = number of steps you spend in phase  $j$ .
- Let  $X$  = number of steps in total =  $X_0 + X_1 + \dots + X_{n-1}$ .

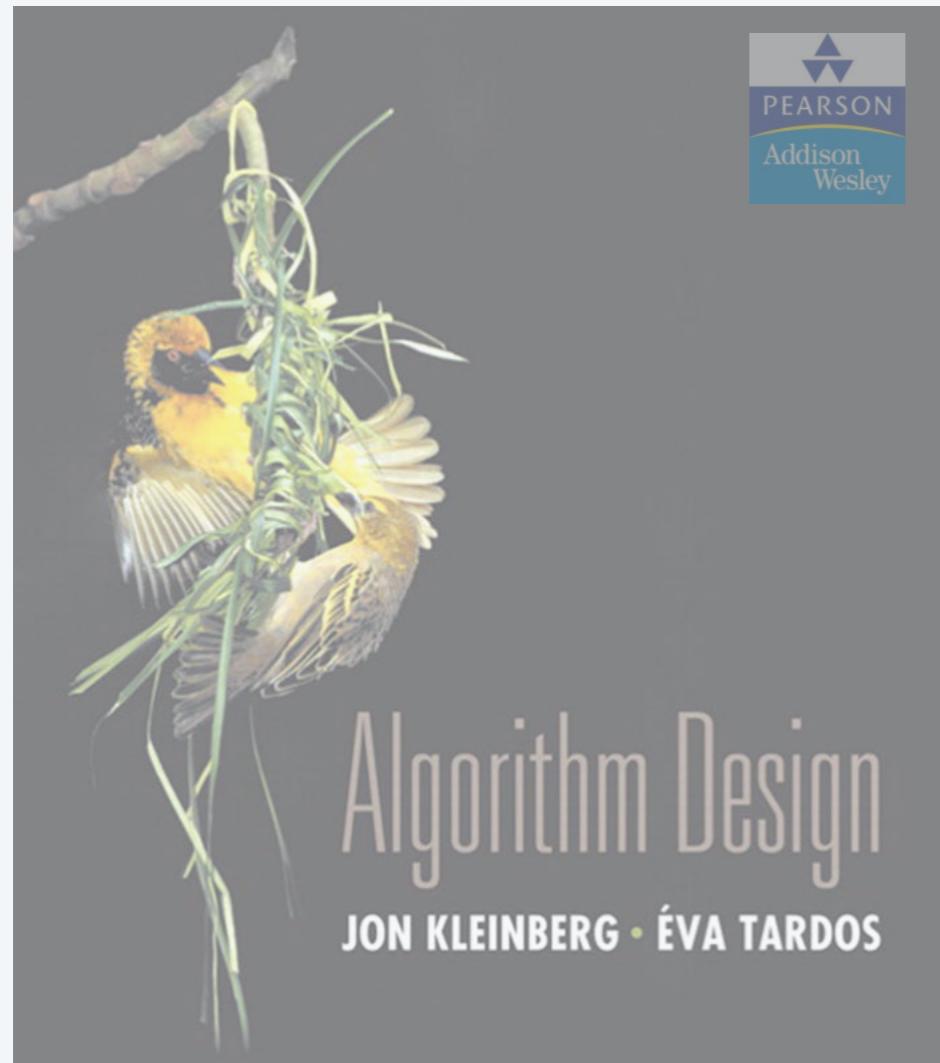
$$E[X] = \sum_{j=0}^{n-1} E[X_j] = \sum_{j=0}^{n-1} \frac{n}{n-j} = n \sum_{i=1}^n \frac{1}{i} = n H(n)$$

*↑ number of coupon i found*

*prob of success =  $(n - j) / n$*

*⇒ expected waiting time =  $n / (n - j)$*

$$\text{Ex.: } 3 \rightarrow 1 + \frac{3}{2} + 3 = 3 \left( \frac{1}{3} + \frac{1}{2} + 1 \right)$$



## 13. RANDOMIZED ALGORITHMS

---

- ▶ *contention resolution*
- ▶ *global min cut*
- ▶ *linearity of expectation*
- ▶ *max 3-satisfiability*
- ▶ *universal hashing*
- ▶ ***Chernoff bounds***
- ▶ *load balancing*

## Chernoff Bounds (above mean)

Probability of deviating from expectation by  $\delta$  percentage,  $\delta=1 \rightarrow 100\%$  deviation

**Theorem.** Suppose  $X_1, \dots, X_n$  are independent 0-1 random variables. Let  $X = X_1 + \dots + X_n$ . Then for any  $\mu \geq E[X]$  and for any  $\delta > 0$ , we have

$$P(X_i = 1) = p_i$$

$$X = \sum_{i=1}^N X_i$$

$$\begin{aligned} E[X] &= E\left[\sum_{i=1}^N X_i\right] = \sum_{i=1}^N E[X_i] = \\ &= \sum_{i=1}^N p_i = \mu \end{aligned}$$

$$\Pr[X > (1+\delta)\mu] < \left[ \frac{e^\delta}{(1+\delta)^{1+\delta}} \right]^\mu$$

because 0-1  
↑  
sum of independent 0-1 random variables  
is tightly centered on the mean

~ sum of  $p_i$ , expectation  
is on exponent

**Pf.** We apply a number of simple transformations.

- For any  $t > 0$ ,

$$\Pr[X > (1+\delta)\mu] = \Pr\left[e^{tX} > e^{t(1+\delta)\mu}\right] \leq e^{-t(1+\delta)\mu} \cdot E[e^{tX}]$$

↑  
 $f(x) = e^{tx}$  is monotone in  $x$

↑  
Markov's inequality:  $\Pr[X > a] \leq E[X] / a$

- Now  $E[e^{tX}] = E[e^{t\sum_i X_i}] = \prod_i E[e^{tX_i}]$

↑  
definition of  $X$

↑  
independence

$$\begin{aligned} E[X_1 \cdot X_2] &= \sum_{x_1} \sum_{x_2} x_1 \cdot x_2 P(X_1 \cdot X_2) = \\ &= \sum_{x_1} \sum_{x_2} P(X_1) \cdot P(X_2) = \\ &= \sum_{x_1} x_1 P(X_1) \cdot \sum_{x_2} x_2 P(X_2) = E(X_1) \cdot E(X_2) \end{aligned}$$

$x_1 \text{ and } x_2$   
independent

# Chernoff Bounds (above mean)

Pf. [ continued ]

- Let  $p_i = \Pr[X_i = 1]$ . Then,

$$E[e^{tX_i}] = p_i e^t + (1 - p_i)e^0 = 1 + p_i(e^t - 1) \leq e^{p_i(e^t - 1)}$$

for any  $\alpha \geq 0$ ,  $1 + \alpha \leq e^\alpha$

- Combining everything:

$$\Pr[X > (1 + \delta)\mu] \leq e^{-t(1+\delta)\mu} \prod_i E[e^{tX_i}] \leq e^{-t(1+\delta)\mu} \prod_i e^{p_i(e^t - 1)} \leq e^{-t(1+\delta)\mu} e^{\mu(e^t - 1)}$$

$\sum_i p_i = E[X] \leq \mu$

↑  
inequality above  
↑  
previous slide

- Finally, choose  $t = \ln(1 + \delta)$ . ■

## Chernoff Bounds (**below mean**)

---

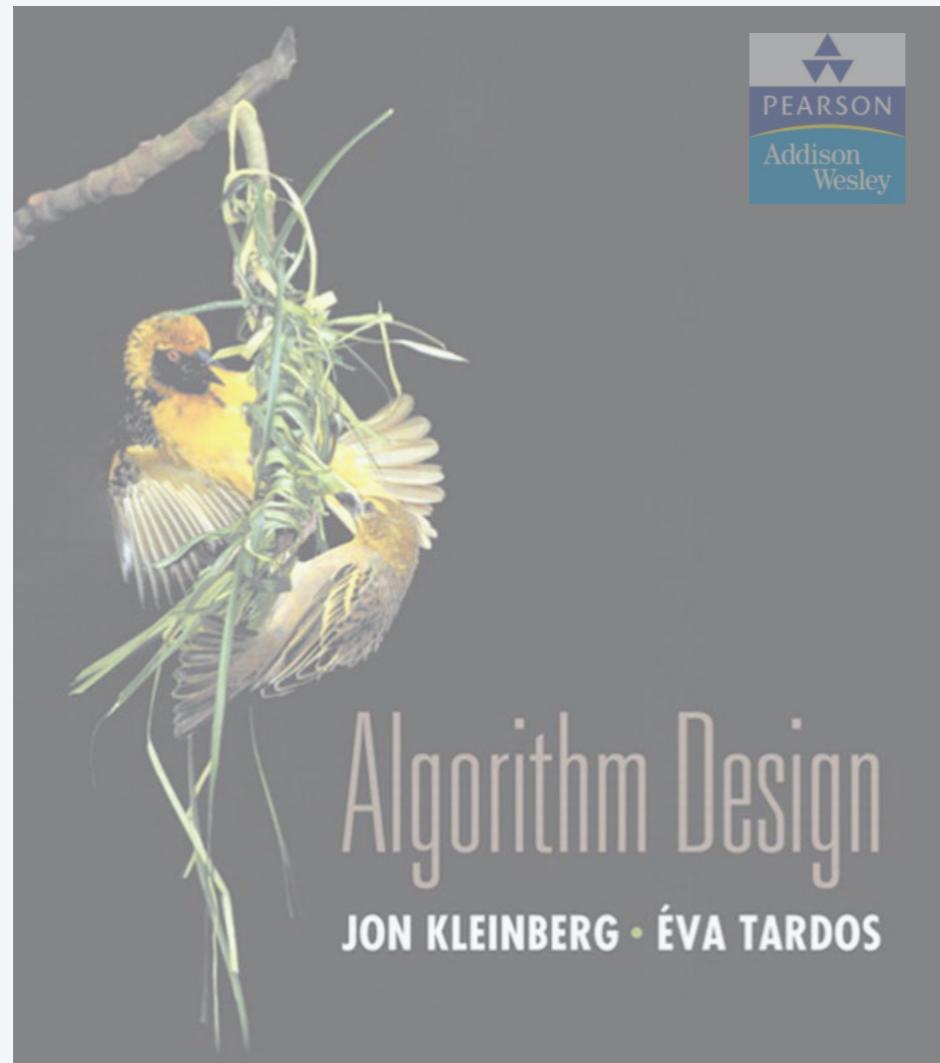
**Theorem.** Suppose  $X_1, \dots, X_n$  are independent 0-1 random variables.

Let  $X = X_1 + \dots + X_n$ . Then for any  $\mu \leq E[X]$  and for any  $0 < \delta < 1$ , we have

$$\Pr[X < (1 - \delta)\mu] < e^{-\delta^2\mu/2}$$

Pf idea. Similar.

**Remark.** Not quite symmetric since only makes sense to consider  $\delta < 1$ .



## 13. RANDOMIZED ALGORITHMS

---

- ▶ *contention resolution*
- ▶ *global min cut*
- ▶ *linearity of expectation*
- ▶ *max 3-satisfiability*
- ▶ *universal hashing*
- ▶ *Chernoff bounds*
- ▶ ***load balancing***

## Load balancing

---

**Load balancing.** System in which  $m$  jobs arrive in a stream and need to be processed immediately on  $m$  identical processors. Find an assignment that balances the workload across processors.

---

**Centralized controller.** Assign jobs in round-robin manner. Each processor receives at most  $\lceil m / n \rceil$  jobs.

**Decentralized controller.** Assign jobs to processors uniformly at random. How likely is it that some processor is assigned “too many” jobs?

---

# Load balancing

## Analysis.

- Let  $X_i$  = number of jobs assigned to processor  $i$ .
- Let  $Y_{ij} = 1$  if job  $j$  assigned to processor  $i$ , and 0 otherwise.
- We have  $E[Y_{ij}] = 1/n$ .
- Thus,  $X_i = \sum_j Y_{ij}$ , and  $\mu = E[X_i] = 1$ .
- Applying Chernoff bounds with  $\delta = c - 1$  yields  $\Pr[X_i > c] < \frac{e^{c-1}}{c^c}$
- Let  $\gamma(n)$  be number  $x$  such that  $x^x = n$ , and choose  $c = e \gamma(n)$ .  
*maximum number  
of jobs assigned  
to x*

$$\Pr[X_i > c] < \frac{e^{c-1}}{c^c} < \left(\frac{e}{c}\right)^c = \left(\frac{1}{\gamma(n)}\right)^{e\gamma(n)} < \left(\frac{1}{\gamma(n)}\right)^{2\gamma(n)} = \frac{1}{n^2}$$

- Union bound  $\Rightarrow$  with probability  $\geq 1 - 1/n$  no processor receives more than  $e \gamma(n) = \Theta(\log n / \log \log n)$  jobs.

Bonus fact: with high probability,  
some processor receives  $\Theta(\log n / \log \log n)$  jobs

$$\Pr(\cup X_i > c) < \frac{1}{m}$$

$$\Pr(X_i > c) \geq 1 - \frac{1}{m}$$

*sym*

Load balancing: many jobs

$$E[Y_{ij}] = \frac{1}{m} \rightarrow E[\sum Y_{ij}] = \frac{1}{m} \cdot 16n \ln n$$

**Theorem.** Suppose the number of jobs  $m = 16n \ln n$ . Then on average, each of the  $n$  processors handles  $\mu = 16 \ln n$  jobs. With high probability, every processor will have between half and twice the average load.

Pf.

- Let  $X_i, Y_{ij}$  be as before.
- Applying Chernoff bounds with  $\delta = 1$  yields

$$\Pr[X_i > 2\mu] < \left(\frac{e}{4}\right)^{16n \ln n} < \left(\frac{1}{e}\right)^{\ln n} = \frac{1}{n^2}$$

$$\Pr[X_i < \frac{1}{2}\mu] < e^{-\frac{1}{2}(\frac{1}{2})^2 16n \ln n} = \frac{1}{n^2}$$

$$1 - \frac{1}{n} \quad (\geq \frac{1}{2}M)$$
$$\frac{1}{n} \Rightarrow 1 - \frac{1}{n} \quad (2M)$$

- Union bound  $\Rightarrow$  every processor has load between half and twice the average with probability  $\geq 1 - 2/n$ . ■

$\hookrightarrow m$  processes

## Randomized Algorithms for Max-SAT

---

$G = (V, E)$  undirected graph

subset  $S \subseteq V$

$|S(S)| \rightarrow$  cardinality of  $S(S)$ , set of edges that have one endpoint in  $S$  and one endpoint in  $(V - S)$

$S \subseteq V - S$

↪ want maximize  $|S(S)| \Rightarrow$  max cut problem

# Max-Cut

undirected  
↑

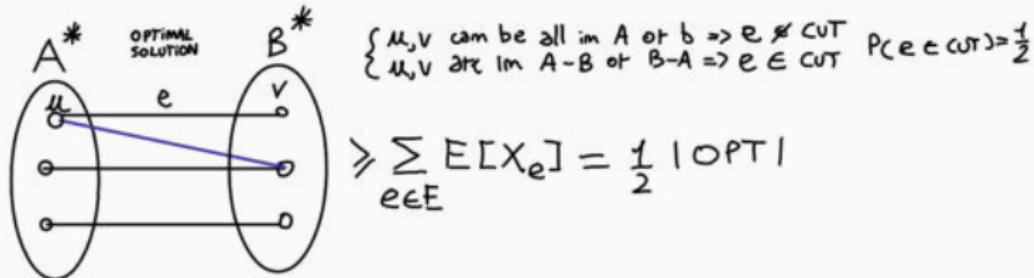
$A \cup B = V$ , want maximize edges across cut, with weight=1.

Given a graph  $G(V, E)$ , find a cut  $A, B$  with maximum weight

$w(A, B) = |(A \times B) \cap E|$ . ↳ set of pairs with one edge in B and other in A  
solution provided by algorithm

Generalizations to weighted cases straightforward and will not be considered in the following.

Min-Cut is solvable in polynomial time, Max-Cut is NP-hard.



# A Simple Randomized Algorithm

## Dumb Rounding

1. For every node  $v$  flip a fair coin.
2. If heads, place  $v$  in  $A$ .
3. If tails, place  $v$  in  $B$ .

## Theorem

Dumb rounding is a  $\frac{1}{2}$ -approximation.

## Corollary

The Max-Cut always contains at least  $\frac{|E|}{2}$  edges.

(We'll only prove the theorem.)

# Analysis of Dumb Rounding

## Theorem

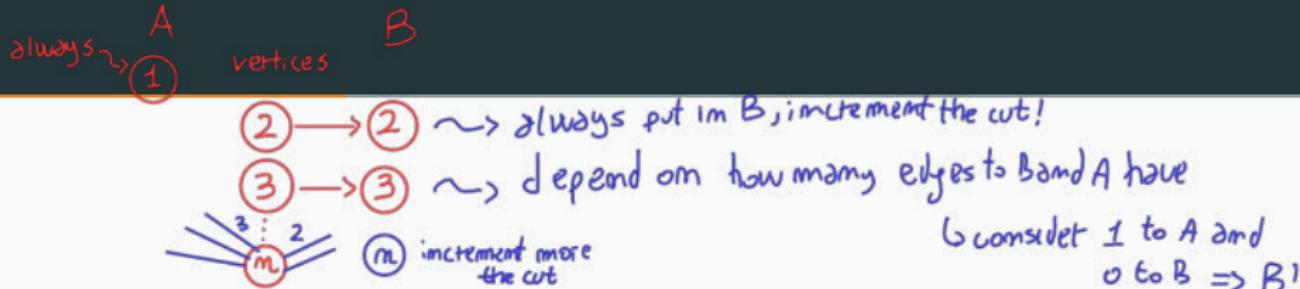
Dumb rounding is a  $\frac{1}{2}$ -approximation.

Consider an arbitrary edge  $e = (u, v)$ . We have the following four cases:

- $u \in A, v \in A \Rightarrow e \notin \text{cut}$
- $u \in A, v \in B \Rightarrow e \in \text{cut}$
- $u \in B, v \in A \Rightarrow e \in \text{cut}$
- $u \in B, v \in B \Rightarrow e \notin \text{cut}$

All cases occur with equal probability. Hence

$$\begin{aligned}\mathbb{E}[|(A \times B) \cap E|] &= \sum_{e \in E} \mathbb{E}[e \in \text{cut}] = \sum_{e \in E} \mathbb{P}[e \in \text{cut}] \\ &= \sum_{e \in E} \frac{1}{2} = \frac{|E|}{2} \geq \frac{OPT}{2}\end{aligned}$$



The output is only a 2-approximation on expectation. How can we be sure?

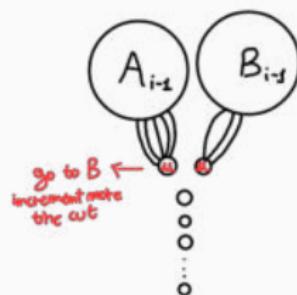
First Idea: Repeat the algorithm a few times and take the best one.  $\log n$  repetitions yield a 2 approximation with probability  $1 - 1/n$ .

Remove the randomness from the algorithm.

## Derandomization

We will use the method of conditional expectations. Let  $X$  be a function of Bernoulli random variables  $X_1, \dots, X_n$ . Then

$$\mathbb{E}[X] = \mathbb{E}[X|X_1 = 0] \cdot \mathbb{P}[X_1 = 0] + \mathbb{E}[X|X_1 = 1] \cdot \mathbb{P}[X_1 = 1].$$



This implies  $\max(\mathbb{E}[X|X_1 = 0], \mathbb{E}[X|X_1 = 1]) > \mathbb{E}[X]$ .

Suppose without loss of generality that

$\max(\mathbb{E}[X|X_1 = 0], \mathbb{E}[X|X_1 = 1]) = \mathbb{E}[X|X_1 = 0]$ . By the same argument

$$\max(\mathbb{E}[X|X_1 = 0, X_2 = 0], \mathbb{E}[X|X_1 = 0, X_2 = 1]) > \mathbb{E}[X].$$

## Evaluation of the Expectation

$K = \text{number of edges already in cut}$   
 $\ell = \text{number of edges only in } A \text{ or } B$

Denote by  $X_i$  the random coin toss of node  $v_i$ . Let  $X_1, \dots, X_{i-1}$  be the fixed coin tosses (i.e. the nodes that were already placed in  $A$  or  $B$ ). Let  $m_1 = |(A \times \{v_i\}) \cap E|$  and  $m_2 = |(B \times \{v_i\}) \cap E|$  and  $k = |(A \times B) \cap E|$  and  $\ell = |((A \times A) \cup (B \times B)) \cap E|$ .

Then

$m_1$  number of edges that go from  $v_i$  to  $A$   
 $m_2$  number of edges that go from  $v_i$  to  $B$

$$\mathbb{E}[X | X_1, \dots, X_{i-1} \text{ fixed}, X_i = 0] = k + m_1 + \frac{|E| - k - m_1 - m_2 - \ell}{2}$$

$$\mathbb{E}[X | X_1, \dots, X_{i-1} \text{ fixed}, X_i = 1] = k + m_2 + \frac{|E| - k - m_1 - m_2 - \ell}{2}$$

↪ want the one that maximizes cut

### Greedy Max-Cut (via Derandomization)

1.  $A \leftarrow \{v_1\}, B \leftarrow \emptyset$

2. For  $i = 2$  to  $n$

place  $v_i$  into the set that maximizes the current cut size.

# Maximum Satisfiability

Given a boolean formula in conjunctive normal form, satisfy as many clauses as possible.

- Formula  $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3} \vee x_4 \vee x_5) \wedge \overline{x_5} \wedge (w_3 \vee x_5) \dots$
- Clause  $(x_1 \vee \overline{x_2} \vee \overline{x_3})$
- Literal  $x_1, \overline{x_2}$

Max 2-SAT (every clause has exactly two literals) is a special case of the Max-Cut problem.

# Integer Linear Program Formulation

- For each clause  $(x_1 \vee x_2 \vee \bar{x}_3)$  add a variable  $z$
- For each literal  $x$  add a binary variable  $y$ .
- $c$  is satisfied if and only if  $y_1 + y_2 + (1 - y_3) \geq 1$ .

*weight of clause, we can think as 1.*

$$\text{maximize} \sum_{\text{clause } c} w_c \cdot z_c \quad \text{such that}$$

$$\sum_{x_i \in c} y_i + \sum_{\bar{x}_i \in c} (1 - y_i) \geq z_c \quad \text{for all clauses } c$$

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

$$0 \leq z_i \leq 1 \rightarrow z_i \in \{0, 1\}$$

$c_i$  not satisfied

$\hookrightarrow c_i$  is satisfied

# Algorithm

maximize  $\sum_{\text{clause } c} w_c \cdot z_c$  such that

$$\sum_{x_i \in c} y_i + \sum_{\bar{x}_i \in c} (1 - y_i) \geq z_c \quad \text{for all clauses } c$$

(simplify notation...)  $\sum_{x_i^* \in c} y_i^* \geq z_c$  for all clauses  $c$

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

$$0 \leq z_i \leq 1$$

$$y_i^* = \begin{cases} y_i & \text{if } x_i \in c \\ 1 - y_i & \text{if } \bar{x}_i \in c \end{cases}$$

## Randomized Rounding

1. Solve LP relaxation ( $0 \leq y_i \leq 1$ )
2. Set each  $x_i$  to 1 with probability  $y_i$

↳ fractional value

# A Useful Lemma

## AM-GM inequality

Let  $y_1, \dots, y_n$  be a list of  $n$  nonnegative numbers. Then

$$\frac{x_1 + x_2 + \dots + x_n}{n} \geq \sqrt[n]{x_1 \cdot x_2 \cdot \dots \cdot x_n}$$

AVERAGE OF  $n$  VALUE  $\geq$  SQUARE ROOT OF PRODUCT OF  $n$  VALUE

**Proof:** By induction over  $n$ . For  $n = 1$  the statement is trivial

Suppose the statement holds for any  $n$  non-negative numbers

If all  $n + 1$  numbers are equal the two means are equal

If not all numbers are equal, at least one number is greater than the arithmetic mean and one number is smaller than the arithmetic mean.

Without loss of generality assume that  $x_{n+1} < a := \frac{x_1 + x_2 + \dots + x_{n+1}}{n+1} < x_1$

## Proof continued

### AM-GM inequality

Let  $y_1, \dots, y_n$  be a list of  $n$  nonnegative numbers. Then

$$a = \frac{x_1 + x_2 + \dots + x_n}{n} \geq \sqrt[n]{x_1 \cdot x_2 \cdot \dots \cdot x_n}$$

$$\Rightarrow 0 < (a - x_{n+1})(x_1 - a)$$

$(n+1) \cdot a = x_1 + x_2 + \dots + x_{n+1} \Leftrightarrow n \cdot a = x_2 + \dots + x_1 + x_{n+1} - a$ , i.e.  $a$  is also the arithmetic mean of the numbers  $\{x_2, \dots, x_n, x_1 + x_{n+1} - a\}$

$$(x_1 + x_{n+1} - a)a - x_1 x_{n+1} = (x_1 - a)(a - x_{n+1}) > 0 \text{ which implies}$$
$$(x_1 + x_{n+1} - a)a > x_1 x_{n+1}$$

We apply the inductive hypothesis on  $\{x_2, \dots, x_n, x_1 + x_{n+1} - a\}$

$$a^{n+1} \geq x_2 \cdot x_3 \cdot \dots \cdot x_n \cdot (x_1 + x_{n+1} - a)a \geq x_2 \cdot x_3 \cdot \dots \cdot x_1 x_{n+1}$$

# Concave Functions

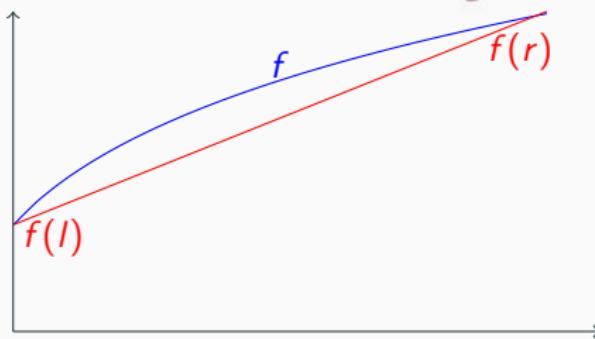
## Definition

any interval  
↑

A function  $f$  is concave on  $[l, r]$  if for any  $0 \leq \alpha \leq 1$

$$f((1 - \alpha) \cdot l + \alpha \cdot r) \geq (1 - \alpha) \cdot f(l) + \alpha \cdot f(r)$$

linear combination  
of any value on the line



We will use the special case  $l = 0$ ,  $r = 1$  and  $f(0) = 0$ :

$$\underline{f(\alpha \cdot r) \geq \alpha \cdot f(1)}$$

# Analysis of Randomized Rounding

Consider any clause  $c$  with  $k$  literals. The associated constraint is

$$\sum_{x_i \in c} y_i + \sum_{\bar{x}_i \in c} (1 - y_i) = \sum_{x_i^* \in c} y_i^* \geq z_c.$$

$$\mathbb{P}[c \text{ is satisfied}] = 1 - \prod_{i=1}^k (1 - y_i^*)$$

AM-GM-inequality  $\geq 1 - \left( \frac{k - \sum_{i=1}^k y_i^*}{k} \right)^k$

$$\left( \frac{\sum_{i=1}^k (1 - y_i^*)}{k} \right)^k \geq \sqrt{(1 - y_i^*)^k}$$

LP-constraint  $\geq 1 - \left( 1 - \frac{z_c}{k} \right)^k$

concave function ( $\alpha \equiv z_c$ )  $\geq \left( 1 - \left( 1 - \frac{1}{k} \right)^k \right) z_c$

## Analysis continued

For any clause, we have  $\mathbb{P}[c \text{ is satisfied}] \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) z_c$ .

$$\begin{aligned} \mathbb{E}\left[\sum_{\text{clause } c} w_c \cdot z_c\right] &= \sum_{\text{clause } c} w_c \cdot \mathbb{P}[c \text{ is satisfied}] \\ &\geq \min_k \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \sum_{\text{clause } c} w_c z_c \\ &\geq \min_k \left(1 - \left(1 - \frac{1}{k}\right)^k\right) OPT \\ &\geq \left(1 - \frac{1}{e}\right) OPT \end{aligned}$$

this algorithm is good  
for small value of  $k$ !  
 $k=1 \Rightarrow OPT$

## Dumb Rounding

1. Set literals to **true** with probability  $1/2$

For a clause with  $k$  literals, the probability of being satisfied is

$$\mathbb{P}[c \text{ is satisfied}] \geq \left(1 - \left(\frac{1}{2}\right)^k\right).$$

## Combined Rounding

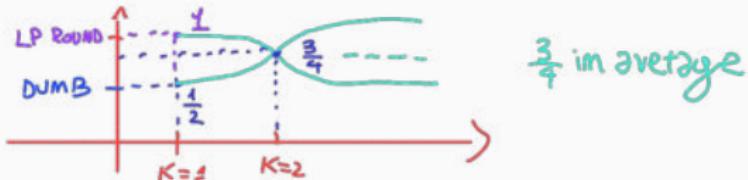
1. Run Dumb Rounding  $\rightarrow W_1$
2. Run Randomized Rounding  $\rightarrow W_2$
3. Output the better of the two

$$\left(\frac{7}{8} + \frac{15}{24}\right) \approx \frac{3}{4}$$

$$K=3 \quad \text{DUMB} = \left(1 - \frac{1}{8}\right) = \frac{7}{8}$$
$$LP = 1 - \left(\frac{2}{3}\right)^3 = \frac{19}{27}$$

$$K=1 \Rightarrow \text{DUMB} = \frac{1}{2}, LP = 1$$

$$K=2 \Rightarrow \text{DUMB} = \left(1 - \left(\frac{1}{2}\right)^2\right) = \frac{3}{4}$$
$$LP = \left(1 - \left(1 - \frac{1}{2}\right)^2\right) = \frac{3}{4}$$



$$\mathbb{E}[\max(W_1, W_2)] \geq \mathbb{E}\left[\frac{1}{2}W_1 + \frac{1}{2}W_2\right]$$

$$\begin{aligned} \text{Linearity} &\geq \sum_{\text{clause } c} w_c \cdot \left( \frac{1}{2} \mathbb{P}[c \text{ is satisfied by Dumb}] \right. \\ &\quad \left. + \frac{1}{2} \mathbb{P}[c \text{ is satisfied by Random}] \right) \\ &\geq \sum_{\text{clause } c} w_c \cdot \frac{1}{2} \left( 1 - \left(\frac{1}{2}\right)^k \right) + \frac{1}{2} \left( 1 - \left(1 - \frac{1}{k}\right)^k \right) z_c \end{aligned}$$

$$\begin{aligned} \text{boring calculations} &\geq \sum_{\text{clause } c} w_c \cdot \frac{3}{4} z_j \\ &\geq \frac{3}{4} \sum_{\text{clause } c} w_c z_j \\ &\geq \frac{3}{4} OPT \quad \text{combining the two algorithm} \end{aligned}$$

## RANDOMIZED ALGORITHM

OPT optimal solution  
ALG output of the algorithm }  
}  $\frac{\text{OPT}}{\text{ALG}}$  ~ this is approximation ratio  
↳ algorithm

In randomized we have not an exact ratio, but an expected value given that we use probability:

$\Rightarrow E[\text{OPT}/\text{ALG}]$  for Randomized algorithm  
(factors)

worst case is happen with small probability, but it's worse to deterministic, better on average

- We obtain better performances respect to deterministic algorithm. → det. algorithm is an Rand. algo with  $P_t = 1$ .

## HOMEWORK 2 2019/2020

**Exercise 2.** Chris' crazy working hours and his descent from southern Germany have given him a business idea: Since in Bavaria, the shops close very early, he will open a grocery service for working people (located in a set  $B$  of office buildings) who never get off work early enough to do their own shopping. His employees will get the ordered groceries for everyone and stash them in a special room in some of the buildings for the customers to then pick up. To save costs, Chris will not rent a room (and equip it with a refrigerator...) for each of the buildings, but wants to do so for as few buildings as possible. For every pair  $(b_1, b_2)$  from  $B$ , he knows if these two buildings allow occupants to go from one to the other in reasonably short time or not. If yes, this is represented by an edge  $(b_1, b_2)$  in the graph  $G$  on vertex set  $B$ , meaning that people from building  $b_1$  can pick up their groceries also at  $b_2$ , and vice versa. Chris has the following idea for a randomized algorithm computing a good subset  $B' \subseteq B$  of buildings to rent a room in, such that all customers can get their groceries:

Fix some order  $e_1, e_2, \dots, e_m$  of all edges in the edge set  $E$  of  $G$ , and set  $B' = \emptyset$ .

Add to  $B'$  all isolated vertices, i.e. the ones without any incident edges.

For every edge  $e_1, e_2, \dots$ , check if one of its endpoints is already contained in  $B'$ . If not, flip a fair coin deciding which of the endpoints to choose, and add this endpoint to  $B'$ .

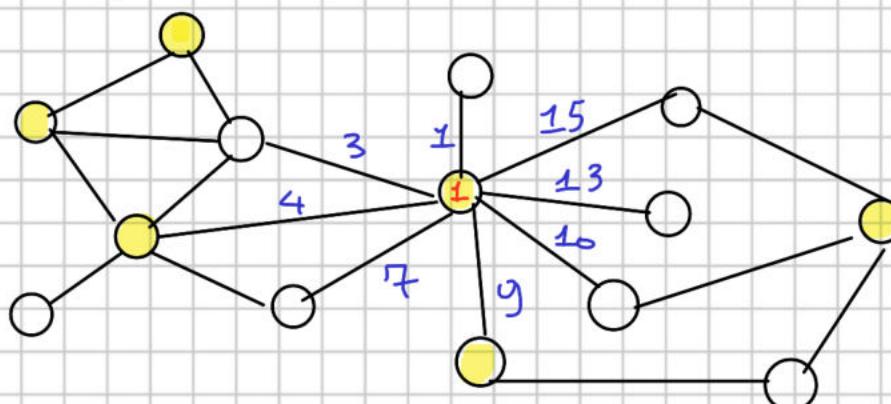
- Show that in expectation, this algorithm will output a feasible solution where Chris has to rent at most twice as many rooms as in the optimal one.  
**(Hint:** Look at the situation for one vertex, together with all of its direct neighbors.)
- Chris would like to derandomize the algorithm to prevent himself from possibly ending up with a ridiculously bad solution. Prove that indeed, for every constant  $c \geq 1$ , the algorithm might produce a  $B'$  with  $|B'| \geq c|OPT|$ .
- Give a short argument why there is not much hope of deriving an efficient, deterministic version using the method of conditional expectation as was presented in the lecture.  
**(Hint:** You can assume the problem is not efficiently approximable up to any factor better than two.)

Chris, since in Boronia shops close very early. He will open a grocery service for people that never get off work early. He has been offered to rent rooms in a set  $B$  of office buildings. To save costs he will not rent every  $b \in B$ , he wants to rent as few as possible.

For every pair  $(b_1, b_2) \in B$ , these two might or might not be connected by a street. People can choose either  $b_1$  or  $b_2$  to be served if  $b_1$  and  $b_2$  are connected. **Goal:** Find a good APX algorithm for this problem.

Each street is represented as an edge  $(b_1, b_2)$  in a graph  $G$ , on a vertex set  $B$ .

**Example:** is like a vertex cover.



- Vertices are the  $b \in B$
- Edges are the streets that connect two buildings.

■ optimal solution: cover all edges of the graphs. Max vertex count.

**Randomize algorithm:**

Fix ordering of the edges, in arbitrary way,  $e_1, e_2, \dots, e_m$  and we define a new set  $B' = \emptyset$ .

1) Add to  $B'$  vertices that are isolated. (because these offices must be rented, no other possibilities). ↳ there might be a customer to be served on the vertex.

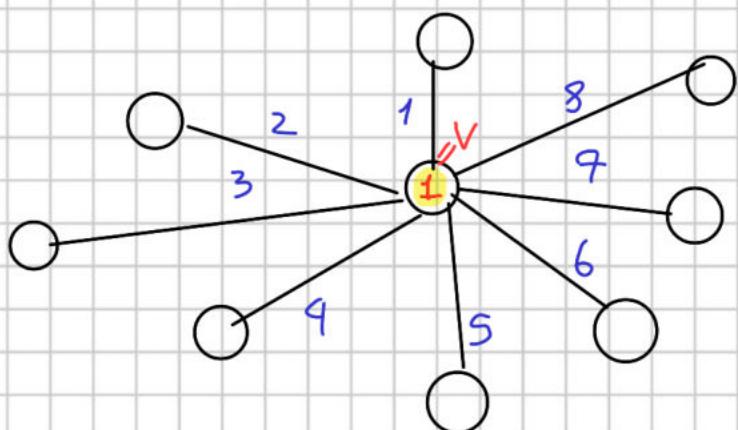
2) For edges  $e_1, e_2, \dots$  check whether one of its endpoints is already in  $B'$ , if not, flip a coin we decide which one of the endpoints will be included in  $B'$  (if yes we do nothing).

**Question (a):** this algorithm is 2-approx to the optimal solution. (Expected value)

**Question (b):**  $\forall c \geq 1$  this algorithm might produce a solution  $B'$  with  $|B'| \geq c|OPT|$

**Answer (a):** Let  $V_1, V_2, \dots, V_k$  be the vertices in the OPT solution. (without the isolated nodes). Fix one  $v \in OPT \rightarrow$  for this vertex we define the set  $E_v = \{e_1, e_2, \dots, e_l\} \rightarrow$  edges that contain  $v$  or an endpoint,  $v$  is an endpoint of these edges. • Count how many vertices that are endpoints in  $E_v$  will be in our solution.

**Example:** we start from vertex 1 of before, we take only edges that have 1 like endpoint.



we have reordered edges for simplicity.

Our algorithm will take some of these vertices with some probability.

1) Algorithm do nothing or otherwise flip a coin.  
 The do nothing if one of the endpoint is already in the solution  $B'$  (not happen in first iteration). Consider  $e_1$ , if vertex  $v$  is included then we do not consider any other vertex that is touched by  $E_v$ , this happen with probability  $1/2$ . Given that  $m_v$  is the additional number of vertices that will be included at step where we consider  $v$ .

The expected number of vertices that will be included is:

$$E(m_v) = \sum_{i \in \{1, \dots, l\}} \Pr [v \text{ is not chosen before considering } e_i]$$

$$E(m_v) = \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^i} + \dots + \frac{1}{2^{e-1}}$$

$\downarrow$   
 $e_1$  was chosen with  $\Pr 1/2$ .

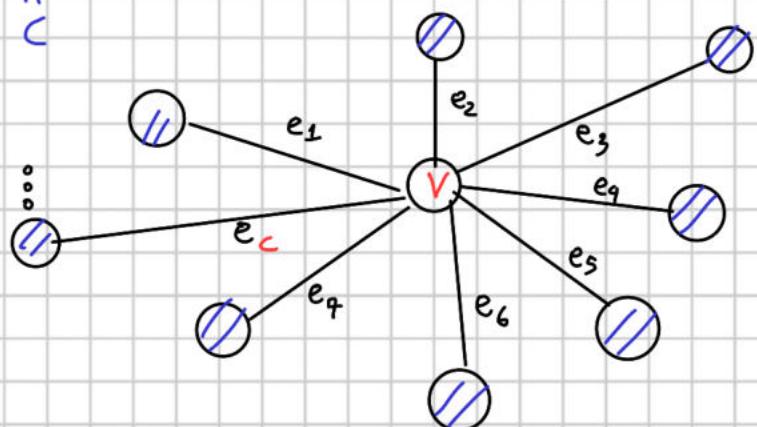
$\rightarrow v$  was not chosen in the first step,  
 $v$  was not chosen in the 2<sup>nd</sup> step as well.  $\Pr$  that other endpoint of  $e_2$  beside  $v$  is chosen.

This idea is used for every  $v \in OPT$ , and obtain:

$E(m_v) \leq 2$  for every vertex  $v$  of  $OPT$  solution  
 The expected value, number of vertices that our algorithm add is 2. Given that this hold for every vertex of the  $OPT$  solution, then  $|B'| \leq 2 |OPT|$

Answer (b): How our algorithm performs in the worst case scenario (that happen with small probability)?  
 if  $B'$  is the solution that our algorithm produce then  
 $|B'| \geq |OPT| + c \in \mathbb{N}$

$\frac{1}{c}$



im worst case algorithm choose never the central vertex

with small probability

i can have this situation with any c

Probability of this happen is  $P_r = \left(\frac{1}{2}\right)^c$

min-cut randomized algorithm RECAP:

min-cut : Contraction

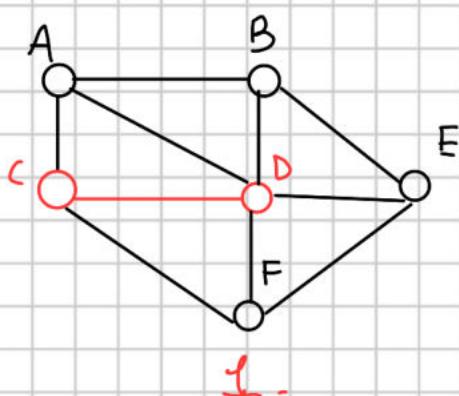
$$G = (V, E)$$

while  $|V| \geq 2$

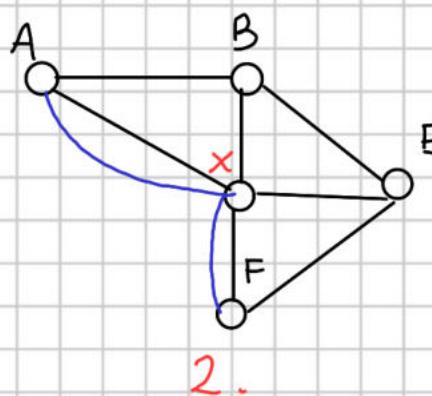
select an edge from  $E$  uniformly at random

$$G = G \setminus e$$

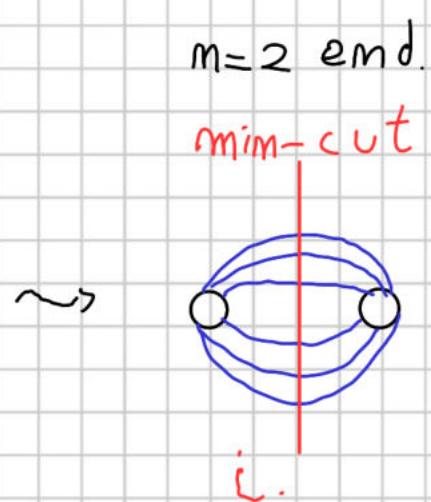
return  $|E|$



1.



2.



i.

$m=2$  end.

min-cut

# HOMEWORK 2 2021/2022 Exetcise 3

\*Alg 1  
\*Alg 2

Consider the following variation of the min-cut algorithm presented in class. We start with a graph  $G$  with  $n$  vertices, and we use the randomized min-cut algorithm to contract the graph down to a graph  $G_k$  with  $k = \sqrt{n}$  vertices. Next, we make  $l = \sqrt{n}$  copies of the graph  $G_k$ , and run the randomized algorithm independently on each copy of the reduced graph. We output the smallest min-cut set found in all the executions.

(a) What is the probability that the reduced graph  $G_k$  has the same cut-set value as the original graph  $G$ ? <sup>best</sup>

(b) What is the probability that the algorithm outputs a correct min-cut set?

Hint: For  $a << b$  you can use  $(1 - 1/b)^a \leq (1 - a/b)$ .

(c) Compare the number of contractions and the resulting error probability to that when running the original algorithm twice and taking the minimum cut-set value.

$\rightarrow$  set of edges across the cut.

Answer (c):  $F^*$  is the optimal solution for  $G$ ,  $c = |F^*|$  number of edges of min-cut  $F^*$ .

$E_j$ : the event that at iteration  $j$  of Alg 1 no edge from  $F^*$  was contracted.

$C_i = \bigcap_{j=1}^i E_j$ , in the first  $i$  iterations no edge from  $F^*$  was contracted. Intersection mean that in all  $E_j$  event no edge from  $F^*$  was not contracted.

$$\Pr [C_{m-\sqrt{m}}] = ? \quad (G_k \rightarrow \sqrt{m})$$

$\rightarrow \Pr (C_i)$  inductive way

In the original  $G$ , what is the degree of a vertex at least?  $\rightarrow$  at least  $c$ , because otherwise contradiction

$2 \cdot |E| = \text{sum of degrees}$  (because each edge is counted twice)

$$2 \cdot |E| \geq m \cdot c$$

$$\Rightarrow |E| = \frac{m \cdot c}{2}$$

$\Pr [C_1] = \text{probability of an edge of } F^* \text{ be not contracted.}$

$$\Pr[C_1] = 1 - \frac{C}{|E|} \geq 1 - \frac{2}{m} \quad (|E| \geq \frac{C \cdot m}{2})$$

Probability that we select an edge from  $F^*$

$$\Rightarrow \Pr[C_1] \geq 1 - 2/m \text{ first inductive step.}$$

$$\Pr[E_2 | C_1] \geq 1 - \frac{2}{m-1} \rightarrow \begin{array}{l} \text{in each iteration contract an edge,} \\ \text{in second iteration one less edge} \end{array}$$

$$\Pr[E_3 | C_2] = \Pr[E_3 | E_1 \cap E_2]$$

...

$$\Pr[E_j | C_{j-1}] \geq 1 - \frac{2}{m-j+1}$$

$$\begin{aligned} \Pr[C_{m-\sqrt{m}}] &= \Pr[E_1] \cdot \Pr[E_2 | C_1] \cdot \dots \cdot \Pr[E_{m-\sqrt{m}} | C_{m-\sqrt{m}-1}] \\ &\geq \prod_{j=1}^{m-\sqrt{m}} \left(1 - \frac{2}{m-j+1}\right) = \frac{1}{m+\sqrt{m}} \end{aligned}$$

# Computational Game Theory

Vincenzo Bonifaci

November 28, 2008

Each agent  $i$  play some strategies. All symmetric games with 2-players and discrete  $S_i$ .

## 1 Games: examples and definitions

Game theory deals with situations in which multiple rational, self-interested entities (individuals, firms, nations, etc.) have to interact.

A normal-form game tries to model a situation in which the entities have to take their decisions simultaneously and independently.

An example is the following Rock-Paper-Scissors game. We can represent it by a table in which the rows correspond to decisions of Player 1, and the columns to decisions of Player 2.

P1, P2	rock	paper	scissors
rock	draw	P2 wins	P1 wins
paper	P1 wins	draw	P2 wins
scissors	P2 wins	P1 wins	draw

**Definition 1.1.** A normal form game is given by:

- a set  $N$  (set of players); often we use  $N = \{1, 2, \dots, n\}$
- for each  $i \in N$ , a nonempty set  $S_i$  (strategies of player  $i$ )

→ defined by a triple, that build a matrix  $(i, S_i, u_i)$

The set  $S := S_1 \times S_2 \times \dots \times S_n$  is called the set of states of the game.

- for each  $i \in N$ , a function  $u_i : S \rightarrow \mathbb{R}$  (utility or payoff function)

**Example 1.2** (Rock-Paper-Scissors).

$u_1, u_2$	rock	paper	scissors
rock	0, 0	-1, 1	1, -1
paper	1, -1	0, 0	-1, 1
scissors	-1, 1	1, -1	0, 0

→ each cell sum is zero, it is a zero sum game

NEGATIVE ↑ POSITIVE ↑  
want the smallest want the highest

Notice that Rock-Paper-Scissors is a zero-sum game: in any state of the game, the sum of the utilities of the players is constant. The Rock-Paper-Scissors game is also finite: the set  $N$  of players has finite cardinality, as do the strategy sets  $S_1, \dots, S_n$ .

Ex.:  $S_{P_1} = \{\text{ROCK, PAPER, SCISSORS}\}$   
Each player want highest possible utility.

**Example 1.3** (Prisoner's dilemma). Two suspects are interrogated in separate rooms. Each of them can confess or not confess their crime. If both confess, they get 4 years each in prison. If one confesses and the other does not, the one that confessed gets 1 year and the other 5. If both are silent, they get 2 years each.

Like in this case, sometimes it is more natural to use cost functions  $(c_i)_{i \in N}$  instead of utility functions  $(u_i)_{i \in N}$ ; notice that it is equivalent since we can always define  $u_i := -c_i$ .

		confess	silent	
		4, 4	1, 5	<i>c<sub>1</sub> is a cost, want to minimize</i>
<i>c<sub>1</sub></i>	confess	4, 4	1, 5	<i>years of prison</i>
	silent	5, 1	2, 2	

Notice that the Prisoner's dilemma is *not* a zero-sum game; however it is a finite game.

So far we saw two-player games, but obviously there are games with more players.

**Example 1.4** (Bandwidth sharing). A group of  $n$  users has to share a common Internet connection with finite bandwidth. Each user can decide what fraction of the bandwidth to use (any amount between none and all). The payoff of each user is higher if this fraction is higher, but is lower if the remaining available bandwidth is too small (packets get delayed too much).

We can model this by defining

- $N := \{1, \dots, n\}$ ;
- $S_i := [0, 1]$  for each  $i \in N$ ;
- $u_i(s) := s_i \cdot (1 - \sum_{j \in N} s_j)$ , where  $s_i \in S_i$  is the strategy selected by player  $i$  and  $s = (s_1, s_2, \dots, s_n)$ .

Notice that this game is not finite: the set of players is finite, but the strategy sets have infinite cardinality.

**Example 1.5** ("Chicken"). Two drivers are headed against each other on a single lane road. Each of them can continue straight ahead or deviate. If both deviate, they both get low payoff. If one deviates while the other continues, he is a "Chicken" and will get low payoff, while the payoff for the other player will be high. If both continue straight ahead, however, a disaster will occur which will cost a lot to the players, as both cars will be destroyed.

		deviate	straight	
		0, 0	-1, 5	
<i>c<sub>1</sub></i>	deviate	0, 0	-1, 5	
	straight	5, -1	-100, -100	

Notice that the type of games we discussed (normal-form games) are "one-shot" in the sense that players move simultaneously and interact only once. There are also model of games in which players move one after the other (*extensive games*) or in which the same game is played many times (*repeated games*). However, in the course we will focus on normal-form games.

## 2 SOLUTION CONCEPTS

$$S = (S_1, S_2, \dots, S_m) = (S_{-i}, S_i)$$

$$\cdot S_{-i} = (S_1, \dots, S_{i-1}, S_{i+1}, \dots, S_m)$$

$\hookrightarrow$  state vector where we cut off the strategy played by i

### Representing the game computationally

When we need to process a game computationally, we have to find some means of representing the game in a concise way. In a normal-form game with a constant number of players, we can represent the whole payoff table explicitly; its size will be polynomial in the total number of strategies. If the number of players is not constant (as in the bandwidth sharing game) we need to represent the functions that compute the payoffs, by encoding them in some formalism (e.g. as C programs or Turing machines).

We also notice that in general we cannot represent real values; in most cases we will need to assume that the codomain of the payoff functions is not  $\mathbb{R}$ , but rather  $\mathbb{Z}$  or  $\mathbb{Q}$ .

## 2 Solution concepts

After we have modeled a game, we would like to know which states of the game represent outcomes that are likely to occur, assuming that players are self-interested and rational. There are different ways to do this; each of them gives rise to a different solution concept. Different solution concepts have different interpretations, advantages and drawbacks.

### 2.1 Dominant strategy solution (equilibrium)

Consider a state of a game  $\Gamma = (N, (S_i)_{i \in N}, (u_i)_{i \in N})$ . The utility of a player  $i$  in state  $s \in S$  will depend on both the action of player  $i$  himself ( $s_i$ ) as well as on the actions of the other players, which we denote conventionally by  $s_{-i}$ . So we can rewrite  $u_i(s)$  (utility of player  $i$  in state  $s$ ) as  $u_i(s_i, s_{-i})$ . Be careful when reading (or using) this notation: we are not reordering the components of the vector  $s$ , we are just writing them differently. For example, with  $(z_i, s_{-i})$  we simply mean the state vector that is obtained from  $s$  by replacing the  $i$ -th component of state  $s$  with  $z_i$ .

The idea of a dominant strategy solution is that if a player has an action that is the best among his actions independently of what the other players do, then this is certainly a possible outcome of the game. This is formalized as follows.

**Definition 2.1.** State  $s \in S$  is a dominant strategy solution if for all  $i \in N$  and for all  $s' \in S$ ,

$$u_i(s_i, s'_{-i}) \geq u_i(s'_i, s'_{-i}).$$

$\hookrightarrow$  any other possible strategy played by other players for any alternative state

(In terms of costs:  $c_i(s_i, s'_{-i}) \leq c_i(s'_i, s'_{-i})$ .)

**Example 2.2 (Dominant strategy in the Prisoner's dilemma).** Is (silent,silent) a dominant strategy in the Prisoner's dilemma game? The answer is no: if  $s = (\text{silent}, \text{silent})$ , there is a player ( $i = 1$ ) and there is an alternative state  $s' = (\text{confess}, \text{silent})$  for which  $c_1(\text{silent}, \text{silent}) > c_1(\text{confess}, \text{silent})$ . This contradicts the definition.

Is (confess,confess) a dominant strategy? We have to check 8 cases (2 players times 4 states) to be sure, but the answer is yes. The point is that no matter what the other player is doing, for each player it is cheaper to confess. So (confess,confess) is a dominant strategy.

A dominant strategy solution represents a "strong" type of equilibrium: every player can rely on his strategy independently of what the others are doing. Unfortunately, it has a big drawback: it does not always exist!

$\hookrightarrow$  S is a dominant strategy equilibrium if for all other possible states, for every agent, it is better to play strategy of state s, instead of any other s'

# Rock-Paper-Scissors has not a dominant strategy solution

2 SOLUTION CONCEPTS

$(0,0) \rightarrow 0 \leq (5,0) \rightarrow 5$  but  $(5,5) \rightarrow -100 \leq (0,5) \rightarrow 1$

if play deviate and other choose straight, is better  
for me play straight. If play straight and opponent  
play straight is better for me to deviate  $\Rightarrow$  no DOMINANT STRATEGY

Exercise 2.1. Show that the Chicken game has no dominant strategy solution.

Since it does not always exist, we cannot use the dominant strategy solution concept to predict what will happen in a game: the players will certainly do something, and this something will not in general be a dominant strategy solution, simply because the game might not admit one.

## 2.1.1 Finding dominant strategy solutions

How do we find, given a game, its dominant strategy solutions? If the game is finite and there is a constant number of players this can be done efficiently. Since there is a polynomial number of states ( $|S_1| \cdot |S_2| \cdot \dots \cdot |S_n|$ , where  $n$  is constant) we simply check for every state whether it satisfies the condition in the definition of dominant strategy solution.  $\rightsquigarrow$  in polytime, simply check all entries

## 2.2 Pure Nash equilibrium

The idea of a pure Nash equilibrium is of that of calling a state an equilibrium if for every player, assuming that other players are not changing their action, the player is selecting his "best" action. That is, no player has an incentive to deviate unilaterally from his action; no one has an interest to alter the "status quo".

Definition 2.3. A state  $s \in S$  is a *pure Nash equilibrium* (PNE) if for all  $i \in N$  and for all  $s'_i \in S_i$ ,

$$u_i(s_i, s_{-i}) \geq u_i(s'_i, s_{-i}).$$

$\xrightarrow{\text{other strategy of player } i \text{ not all other state!}}$

The definition is superficially very similar to that of dominant strategy: take your time to appreciate the difference.

However, there is a similarity and in fact every dominant strategy solution is also a pure Nash equilibrium (can you see why?).

The converse is not true: some games without dominant strategy solutions have pure Nash equilibria.

Example 2.4 (PNE in the Chicken game). Is the state  $s = (\text{straight}, \text{straight})$  a PNE in the Chicken game? The answer is no: there is a player ( $i = 1$ ) and an alternative strategy  $s'_i$  (deviate) such that  $-1 = u_1(\text{straight}, \text{straight}) < u_1(\text{deviate}, \text{straight}) = -100$ . This contradicts the definition.

Is the state  $s = (\text{deviate}, \text{straight})$  a PNE in the Chicken game? Let's see. If player 1 knows that player 2 is going straight, deviating (-1) is better than going straight (-100). On the other hand, if player 2 knows that player 1 is deviating, going straight (5) is better than deviating (0). So  $(\text{deviate}, \text{straight})$  is a PNE.

Notice that PNE need not be unique: in fact, in the Chicken game, there are two PNE (which is the other one?).

$(\text{straight}, \text{deviate})$

Let's look at a more complicated example.

Example 2.5 (PNE in the Bandwidth sharing game). Let's see what player  $i$  will do when the strategies of the other players are  $s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n$ . Let's define  $t := \sum_{j \neq i} s_j$ . From the point of view of player  $i$ , the quantity  $t$  is a constant. By definition of the payoffs we have  $u_i(s) = s_i \cdot (1-t-s_i)$ .

M, W	B	S
B	2, 1	0, 0
S	0, 0	1, 2

B = ballet  
S = soccer match

there are two pure Nash equilibrium  
(2, 1) and (1, 2), but no dominant strategy

Player  $i$  can control the one-dimensional variable  $s_i \in [0, 1]$ . If we take the derivative of  $u_i(s)$  with respect to  $s_i$  we obtain

$$\frac{\partial}{\partial s_i} u_i(s) = 1 - t - 2s_i.$$

By standard analysis we know that the maximum of  $u_i$  is achieved when  $\frac{\partial}{\partial s_i} u_i(s) = 0$  (or, possibly, when  $s_i$  is at an extreme point of  $[0, 1]$ , but this is not the case in our example because we get the worst possible payoff in that case). So the player will select  $s_i = \frac{1}{2}(1 - t) = \frac{1}{2}(1 - \sum_{j \neq i} s_j)$ . This will be true for all  $i \in N$ , so by symmetry we find out that  $s_i = 1/(n + 1)$  for all  $i$ .

Unfortunately, although the PNE solution concept applies to a larger class of games, it has basically the same problem as that of a dominant strategy solution: it does not always exist.

**Exercise 2.2.** Show that the Rock-Paper-Scissors game has no PNE.

### 2.2.1 Finding pure Nash equilibria

When the game is finite and the number of players is constant, we can find efficiently all pure Nash equilibria of the game via a simple enumeration of all states, as we did in the case of dominant strategy solutions.

## 2.3 Mixed Nash equilibrium

So far there was no way for a player to interpolate between two actions: either he selects action  $s_i$  or he performs another action  $s_j$ . We now relax this constraint by allowing the player to choose actions with certain probabilities. For example he might choose action  $s_1$  with probability 1/4, action  $s_2$  with probability 1/3, and action  $s_3$  with probability 5/12. Such strategies are called mixed, in contrast with the usual deterministic pure strategies. Pure strategies are perhaps more natural, but often the strategies arising in a game are in fact mixed strategies.

**Definition 2.6.** A mixed strategy for player  $i$  is a probability distribution on the set of  $S_i$  of pure strategies. That is, it is a function  $p_i : S_i \rightarrow [0, 1]$  such that  $\sum_{s_i \in S_i} p_i(s_i) = 1$ . A mixed state is a family  $(p_i)_{i \in N}$  consisting of one mixed strategy for each player.

Notice that every pure state  $s$  has probability  $p(s) := p_1(s_1) \cdot p_2(s_2) \cdots p_n(s_n)$  of being realized.

Thus, a mixed state  $(p_i)_{i \in N}$  induces an expected payoff for player  $i$  equal to  $\sum_{s \in S} p(s) \cdot u_i(s)$ . This is the expected payoff of a state selected probabilistically by the players according to their mixed strategies.

We can now define the notion of mixed Nash equilibrium (MNE).

$$\text{Expected payoff of } i: E[u_i] = \sum_{s \in S} p(s) \cdot u_i(s)$$

**Definition 2.7.** A mixed state is a mixed Nash equilibrium if no player can unilaterally improve his expected payoff by switching to a different mixed strategy. (at any incentive to deviate)

Since mixed strategies generalize pure strategies, it is not hard to see that every PNE is also a MNE. The opposite is not true. In fact, there are games without PNE that admit MNE. More than that: the surprising fact is that any finite game (game where  $N$  and  $S$  are finite) admits at least one mixed Nash equilibrium!

↪ Theorem: every finite game has a MNE.

**Theorem 2.1** (Nash 1950). *Every finite game admits at least one mixed Nash equilibrium.*

**Example 2.8** (MNE for the Rock-Paper-Scissors game). We saw that the Rock-Paper-Scissors game has no pure Nash equilibria. According to Nash's Theorem it should have at least one equilibrium. In fact, we claim that if we define  $p := (1/3, 1/3, 1/3)$ , then  $(p, p)$  is a MNE.

Let's verify this. Consider for example player 1. We should check that when player 2 uses probability distribution  $p$ , player 1 has no incentive to play a mixed strategy different from  $p$ . (We should also do a similar check with the roles of the players reversed, but in this case everything will be symmetric.)

If player 2 uses mixed strategy  $p$ , and player 1 uses a generic mixed strategy  $q = (a, b, c)$  where  $a + b + c = 1$ , then the expected payoff for player 1 becomes

$$\begin{aligned} a \cdot 1/3 \cdot (0) + a \cdot 1/3 \cdot (-1) + a \cdot 1/3 \cdot (+1) + \\ b \cdot 1/3 \cdot (+1) + b \cdot 1/3 \cdot (0) + b \cdot 1/3 \cdot (-1) + \\ c \cdot 1/3 \cdot (-1) + c \cdot 1/3 \cdot (+1) + c \cdot 1/3 \cdot (0) = 0. \end{aligned}$$

$P = (1/3, 1/3, 1/3)$   
 $q = (a, b, c)$   
 $(P, P)$  is a MNE

So the expected payoff is a constant (0) no matter what  $a$ ,  $b$  and  $c$  are! This means that there is no point for player 1 in changing them. Similarly, when player 1 plays  $(1/3, 1/3, 1/3)$ , player 2 has no incentive to change his strategy from  $(1/3, 1/3, 1/3)$ . The two players "lock" each other in the mixed Nash equilibrium.

At this point you might wonder why another mixed state, like  $(1/2, 1/4, 1/4)$  for both players, is not a MNE. The reason is that if e.g. player 2 plays something different from  $(1/3, 1/3, 1/3)$ , then the player 1 is no longer indifferent between his possible responses. In this case, when player 2 plays  $(1/2, 1/4, 1/4)$ , it will be more convenient for player 1 to play  $(0, 1, 0)$  than to play  $(1/2, 1/4, 1/4)$ : since player 2 is playing Rock more often than Paper or Scissors, it is best for player 1 to always play Paper (you can check this by computing the expected payoff for player 1). So  $((1/2, 1/4, 1/4), (1/2, 1/4, 1/4))$  is not a MNE.

have an incentive  
to change

### 2.3.1 Finding mixed Nash equilibria

Finding MNE is considerably harder than finding dominant strategy solutions or PNE, even when the game is finite and there are only a constant number of players, and even when there are only two players. Apparently we have to check for an infinite set of mixed states, so it is not even clear that we can do it in finite time!

Luckily, there are some notions that can help us.

**Definition 2.9.** A mixed strategy  $p_i$  is a best response to mixed strategies  $p_1, \dots, p_{i-1}, p_{i+1}, \dots, p_n$  if for all mixed strategies  $p'_i$  of player  $i$ ,

*best response to a mixed strategy*

$$\sum_{s \in S} p_1(s_1) \dots p_i(s_i) \dots p_n(s_n) \cdot u_i(s) \geq \sum_{s \in S} p_1(s_1) \dots p'_i(s_i) \dots p_n(s_n) \cdot u_i(s).$$

That is, a best response attains the maximum possible expected utility among all possible mixed strategies of this player. In fact, we can now say that a mixed state is a MNE if and only if every player is playing a best response strategy.

$p = (1, 0, 0)$  is the best response to mixed strategy  $q = (\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$

## 2 SOLUTION CONCEPTS

### SUPPORT OF MIXED STRATEGY

$$\text{Ex: } p_2 = \left( \frac{1}{2}, \frac{1}{4}, \frac{1}{4} \right) \quad \text{SUPP}(p_2) = \{1, 2, 3\}$$

$$p_1 = (1, 0, 0) \quad \text{SUPP}(p_1) = \{1\}$$

7  
Positions of non zero prob.

**Definition 2.10.** The *support* of a mixed strategy  $p_i$  is the set of all pure strategies that have nonzero probability in it:  $\text{supp}(p_i) := \{j \in S_i : p_i(j) > 0\}$ .

**Example 2.11.** If  $p_i = \left( \frac{1}{3}, 0, 0, \frac{1}{2}, \frac{1}{6} \right)$  then the support of  $p_i$  is  $\{1, 4, 5\}$ .

The following characterization will be very useful for computing mixed Nash equilibria.

**Theorem 2.2.** A mixed strategy  $p_i$  is a best response if and only if all pure strategies in  $\text{supp}(p_i)$  are best responses.  
 $\Leftrightarrow \text{supp}(p_i) = \{i\} \rightarrow \text{best response}$

*Proof.* If all strategies in  $\text{supp}(p_i)$  are best responses, then since the mixed strategy is a convex combination of them, it will have the same expected payoff and also be a best response.

On the other hand, if mixed strategy  $p_i$  is a best response, all pure strategies in its support are best responses: suppose this was not the case, then by decreasing the probability of the pure strategy with *worst* expected payoff, and redistributing the remaining probability proportionally for the other pure strategies in the support, we could improve the expected payoff. But then  $p_i$  would not be a best response.  $\square$

Thus the *hard part in finding a MNE is finding the right supports*.

Suppose that we are given a *finite two-player game*. This can be completely specified by two *payoff matrices*  $A = (a_{ij})_{ij}, B = (b_{ij})_{ij} \in \mathbb{R}^{m_1 \times m_2}$  (which is why these games are also called *bimatrix games*), where  $S_1 = \{1, \dots, m_1\}$  and  $S_2 = \{1, \dots, m_2\}$  are the *strategy sets*. If we knew the supports  $I \subseteq S_1$  and  $J \subseteq S_2$ , to check whether there is a MNE with these supports it would be enough to check whether the following system has a solution in the (vector) variables  $x, y$ :

game with  $A = \{a_{ij}\}$

$$B = \{b_{ij}\}$$

supports:  $I \subseteq S_1, J \subseteq S_2$

variables:  $x_i, i \in I$

$$y_j, j \in J$$

$$\sum_{j \in J} y_j a_{kj} \leq \sum_{j \in J} y_j a_{ij} \quad \forall k \in S_1, \forall i \in I \quad (1)$$

$$\sum_{i \in I} x_i b_{ik} \leq \sum_{i \in I} x_i b_{ij} \quad \forall k \in S_2, \forall j \in J \quad (2)$$

$$\sum_{i \in I} x_i = 1 \quad x_i \text{ must be a probability distribution} \quad (3)$$

$$\sum_{j \in J} y_j = 1 \quad y_j \text{ same} \quad (4)$$

$$x_i \geq 0 \quad \forall i \in I \quad (5)$$

$$y_j \geq 0 \quad \forall j \in J. \quad (6)$$

Intuitively, the equations (1) state that every pure strategy in the support of  $x$  (that is,  $I$ ) is a best response to mixed strategy  $y$ : no other pure strategy  $k$  in  $S_1$  can achieve better expected payoff. Similarly, equations (2) state that every pure strategy in the support of  $y$  is a best response to mixed strategy  $x$ . Equations (3)–(6) simply state that  $x$  and  $y$  are in fact mixed strategies (probability distributions on  $S_1$  and  $S_2$ , respectively).

**Theorem 2.3.** There is an algorithm that finds a MNE of a bimatrix game  $(A, B)$  where  $A, B \in \mathbb{Q}^{m_1 \times m_2}$  in time  $2^{m_1+m_2} \cdot \text{poly}(\text{bits}(A) + \text{bits}(B))$ .

*Proof.* We simply enumerate all possible supports  $I \subseteq S_1, J \subseteq S_2$  and for each of them check whether the above linear system is feasible. If it is, then  $(x, y)$  is a MNE.  $\square$

**Example 2.12.** Let's go back to the Chicken game.

$u_1, u_2$	deviate	straight
deviate	0, 0	-1, 5
straight	5, -1	-100, -100

We saw that the game has two pure Nash equilibria: (deviate,straight) and (straight,deviate). Let's see if it has one mixed Nash equilibrium. It is easy to see that in this case, when the support of one of the players has size one, the other player best response is a single pure strategy. Thus, since we already investigated the PNE of this game, any other MNE (if there is one) will necessarily have a support of size at least two for both players. So there is no need to enumerate all the possible supports  $I$  and  $J$ : we can directly take  $I = J = \{d, s\}$  where  $d$  and  $s$  are shorthand for "deviate" and "straight".

The linear program then gives us:

$$\begin{aligned} y_d \cdot 0 + y_s \cdot (-1) &= y_d \cdot 5 + y_s \cdot (-100) \\ x_d \cdot 0 + x_s \cdot (-1) &= x_d \cdot 5 + x_s \cdot (-100) \\ y_d + y_s &= 1 \\ x_d + x_s &= 1 \\ x_d, x_s, y_d, y_s &\geq 0. \end{aligned}$$

$$\begin{aligned} \text{since: } x_d &= 1 - x_s \\ -x_s &= 5(1-x_s) - 100x_s \\ 104x_s &= 5 \\ x_s &= \frac{5}{104} \Rightarrow x_d = \frac{99}{104} \end{aligned}$$

The solution is  $x_d = y_d = 99/104$ ,  $x_s = y_s = 5/104$ . So we discovered another equilibrium. In this equilibrium both players will deviate from their route with high probability, but each of them has a small probability ( $5/104$ ) of going straight.

**Exercise 2.3.** Find all MNE of the following bimatrix game.

$u_1, u_2$	Action 1	Action 2
Action 1	2, 1	0, 3
Action 2	1, 2	4, 1

*Remark 2.1.* It is not known whether finding a Nash equilibrium of a finite two-player game is a problem that can be solved in polynomial time, although there is some complexity-theoretic evidence that it is not. However the problem is definitely not NP-hard: in fact, the associated decision problem is trivial, as the answer is always "yes"!

### 2.3.2 The case of zero-sum games

In the special case where we have a zero-sum two-player game, it turns out that there is a polynomial time algorithm to find a Nash equilibrium. In fact, consider such a game; this can be specified by a single matrix  $A = (a_{ij})_{ij}$ , whose entries represent simultaneously the payoffs for the first (row) player as well as the costs for the second (column) player.

Assume that the column player knows that row player is playing mixed strategy  $x$ . Then the column player will look at the expected payoff vector  $xA$ , and since he wants to minimize his loss, he will only play strategies that correspond to minimum entries in this vector. So if we now consider things from the point of view of the row player, he can secure himself a payoff of  $v$  if he selects a mixed strategy  $x$  such that no matter what the second player plays, the payoff will be at least  $v$ . We thus have the following linear program for maximizing the “safety level”  $v$ :

$$\begin{aligned} v^* &= \max v \\ \sum_{i \in S_1} x_i a_{ij} &\geq v \quad \forall j \in S_2 \\ \sum_{i \in S_1} x_i &= 1 \\ x_i &\geq 0 \quad \forall i \in S_1. \end{aligned}$$

Similarly, for the column player we get the following program:

$$\begin{aligned} u^* &= \min u \\ \sum_{j \in S_2} y_j a_{ij} &\leq u \quad \forall i \in S_1 \\ \sum_{j \in S_2} y_j &= 1 \\ y_j &\geq 0 \quad \forall j \in S_2. \end{aligned}$$

↗ is the dual  
of above

We notice that these linear programs are duals of each other! We can now prove the following.

**Theorem 2.4.** Optimum solutions for the above pair of linear programs give mixed strategies that form a Nash equilibrium of the two-person zero-sum game.

*Proof.* By linear duality  $v^* = u^*$ . If the players play this pair of strategies, the row player cannot increase his win, as the column player is guaranteed by his strategy not to lose more than  $u^*$ . Similarly, the column player cannot decrease his loss under  $v^*$ . This means that the pair of strategies is at equilibrium.  $\square$

**Corollary 2.5.** There is a polynomial time algorithm for finding a mixed Nash equilibrium in a two-player zero-sum game.

In fact, the quantity  $v^* = u^*$  is called the *value* of the zero-sum game: it is the payoff that the first player can ensure for himself by playing the game at the best of his possibilities. Notice that the value might be negative: in that case it would be better for the first player not to play at all!

**Example 2.13.** Consider the following zero-sum game.

$u_1 = -u_2$	Action 1	Action 2
Action 1	2	-1
Action 2	1	3

→ No pure Nash equil.

"1  
 2, -2      -1, 1  
 1, -1      3, -3

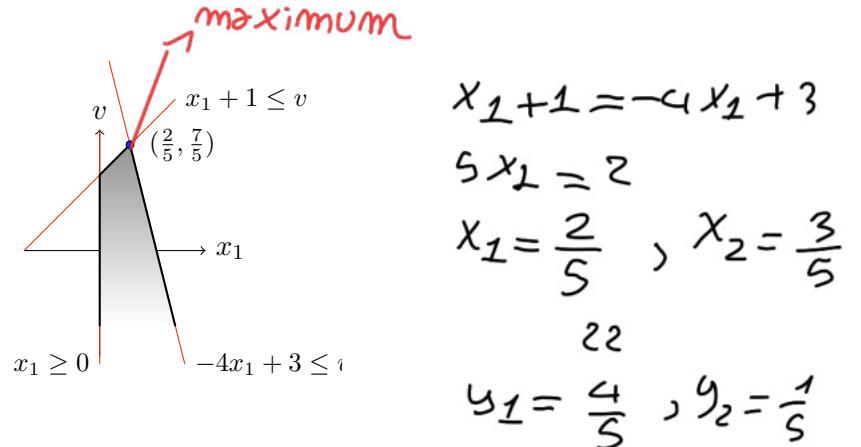


Figure 1: Example 2.13

The first LP becomes:

$$\begin{aligned} \max v \\ x_1 \cdot 2 + x_2 \cdot 1 &\geq v \\ x_1 \cdot (-1) + x_2 \cdot 3 &\geq v \\ x_1 + x_2 &= 1 \\ x_1, x_2 &\geq 0. \end{aligned}$$

$$\begin{aligned} x_2 &= 1 - x_1 \\ 2x_1 + 1 - x_1 &= x_1 + 1 \geq v \\ -x_1 + 3 - 3x_1 &= -4x_1 + 3 \geq v \end{aligned}$$

If we substitute  $x_2 = 1 - x_1$ , we can rewrite the main inequalities as  $x_1 + 1 \geq v$  and  $-4x_1 + 3 \geq v$ . From these we find out (Figure 1) that the best possible value of  $v$  is  $7/5$ , obtained when  $x_1 = 2/5$ ,  $x_2 = 3/5$ . Similarly for the column player, we obtain that  $y_1 = 4/5$ ,  $y_2 = 1/5$ .

### 2.3.3 Degenerate games

Sometimes it can happen that we have a “degenerate” kind of equilibrium. Consider a zero-sum game given by the following matrix:

$$\begin{pmatrix} 2 & 4 \\ 2 & 5 \end{pmatrix}$$

By solving for the second player’s equilibrium strategy we find  $y_1 = 1$ ,  $y_2 = 0$ . However if we write the LP for the first player:

$$\begin{aligned} \max v \\ 2x_1 + 2x_2 &\geq v \\ 4x_1 + 5x_2 &\geq v \\ x_1 + x_2 &= 1 \\ x_1, x_2 &\geq 0. \end{aligned}$$

we find out that any probability distribution  $(x_1, x_2)$  is feasible (and  $v = 2$ ). What is happening? The point is that, as the second player will play the first column, it does not really matter which row the first player selects. So we have infinitely many MNE, of the form  $((\epsilon, 1 - \epsilon), (1, 0))$  for any  $\epsilon \in [0, 1]$ .

### 2.3.4 Dominated strategies

Another useful concept to keep in mind in the study of equilibria is that of a *dominated strategy*.

**Definition 2.14.** A pure strategy  $s_i$  of a player  $i \in N$  is *strictly dominated* by a strategy  $s'_i$  of the same player if, for each combination  $s_{-i}$  of strategies of the remaining players,

$$u_i(s_i, s_{-i}) < u_i(s'_i, s_{-i}).$$

Thus, a strictly dominated strategy is a strategy for which there is an alternative that is always strictly better for the player, independently of the actions of the others. As such, it is not rational to play a strictly dominated strategy and in fact it can be easily proven that they are never part of a Nash equilibrium.

A useful preprocessing step, when analyzing a game, is then to eliminate from it strategies that are strictly dominated. Since the strategy is strictly dominated, we are not “forgetting” any equilibrium in this way. This procedure can be iterated until no strategy is strictly dominated.

**Example 2.15.** Consider the following bimatrix game  $(A, B)$ :

$$A = \begin{pmatrix} 0 & 2 & 5 \\ 2 & 4 & -1 \end{pmatrix}, \quad B = \begin{pmatrix} 4 & 5 & -1 \\ 2 & -3 & 0 \end{pmatrix}$$

Looking at  $A$ , we see that no strategy of the row player is strictly dominated. Looking at  $B$ , we see that the third column is strictly dominated by the first one. We can thus eliminate the third column and obtain the simpler, but equivalent, game:

$$A = \begin{pmatrix} 0 & 2 \\ 2 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} 4 & 5 \\ 2 & -3 \end{pmatrix}$$

In this new game, the first row in matrix  $A$  is strictly dominated by the second row. Thus player 1 will never play the first row. We obtain the even simpler game:

$$A = (2 \ 4), \quad B = (2 \ -3)$$

We can finally conclude that player 2 will select the column that gives payoff 2. We have thus reached a pure Nash equilibrium where both players have payoff 2. This is also a PNE of the original game, and from what we said it must be the *only* one (whether pure or mixed), because all the strategies we discarded cannot be part of an equilibrium.

The notion of strictly dominated strategy can be weakened to allow for equality of the payoffs, as follows.

**Definition 2.16.** A pure strategy  $s_i$  of a player  $i \in N$  is *weakly dominated* by a strategy  $s'_i$  of the same player if, for each combination  $s_{-i}$  of strategies of the remaining players,

$$u_i(s_i, s_{-i}) \leq u_i(s'_i, s_{-i}).$$

Differently from strictly dominated strategies, we *cannot* eliminate weakly dominated strategies from a game while being sure that we do not remove a Nash equilibrium. In fact, if we did so in the example of Section 2.3.3, we would end up with a single pure equilibrium (where player 1 plays the second row and player 2 the first column), while we have already seen that the game had infinitely many Nash equilibria.

### 2.3.5 Other game examples

**Example 2.17** (Auction). An object is to be assigned to a player in the set  $\{1, \dots, n\}$  in exchange for a payment. Player  $i$ 's valuation of the object is  $v_i$ , and  $v_1 > v_2 > \dots > v_n > 0$ . The mechanism used to assign the object is a (sealed-bid) auction: the players simultaneously submit bids (nonnegative numbers), and the object is given to the player with the lowest index among those who submit the highest bid, in exchange for a payment.

In a *first price* auction the payment that the winner makes is the price that he bids.

**Exercise 2.4.** Formulate a first price auction as a strategic game and analyze its pure Nash equilibria. Show that in all equilibria player 1 obtains the object.

In a *second price* auction the payment that the winner makes is the highest bid among those submitted by the players who do not win (so that if only one player submits the highest bid then the price paid is the *second* highest bid). Notice that a very similar second price auction is used by online auction sites like eBay.

**Exercise 2.5.** Show that in a second price auction the bid  $v_i$  of any player  $i$  is a weakly dominant action. Show that nevertheless there are equilibria in which the winner is not player 1.

**Example 2.18** (Catching the votes). Each of  $n$  people chooses whether or not to become a political candidate, and if so which position to take. There is a continuum of citizens, each of whom has a favorite position; the distribution of favorite positions is given by a density function. A candidate attracts the votes of those citizens whose favorite positions are closer to his position than to the position of any other candidate; if  $k$  candidates choose the same position then each receives the fraction  $1/k$  of the votes that the position attracts. The winner of the competition is the candidate who receives the most votes. Each person prefers to be the unique winning candidate than to tie for first place, prefers to tie for first place than to stay out of the competition, and prefers to stay out of the competition than to enter and lose.

**Exercise 2.6.** Formulate this situation as a strategic game and find the set of pure Nash equilibria when  $n = 2$ .

# QUALITY OF EQUILIBRIA

## Computational Game Theory

Vincenzo Bonifaci

December 15, 2008

### 2 Inefficiency of equilibria

The tragedy of the commons and the price of anarchy. We have seen in the preceding lectures how it is possible to model, using games and solution concepts, the kind of behavior that can arise in situations where multiple self-interested agents interact. What happens to the global behavior of the system? The concept of *social utility* can be used to measure the quality of a given state of a game.

**Definition 2.1.** The *social utility* of a game in state  $s \in S$  is the quantity  $\sum_{i \in N} u_i(s)$ .

*Remark 2.1.* There are actually other possible definitions of social utility. The one we gave might be called *utilitarian* or *egalitarian*, as it gives the same weight to all players.

In general, the social utility that derives from the behavior at equilibrium is not the best possible one; the players of the game would globally be better off if a central optimal coordination was possible. The question is, how much worse than optimal can the social utility become at an equilibrium? This question is interesting because, if we knew that equilibria in a game have high social utility, comparable to the optimal one, then we can obtain more or less the same result as the optimal one without the need of enforcing the players to choose particular actions – which can often be expensive or impossible.

For example, consider the Bandwidth Sharing game. We have seen that the game has a pure Nash equilibrium where each of the  $n$  players uses a fraction  $1/(n+1)$  of the bandwidth. The payoff for every player is then  $1/(n+1)^2$ , which means that the social utility at the equilibrium is  $n/(n+1)^2 = \Theta(1/n)$ . On the other hand, if every player used only a fraction  $1/(2n)$  of the bandwidth, the payoff of each player would be  $1/(4n)$ , so the corresponding social utility would be  $1/4$ . Notice that is  $\Theta(n)$  times larger than the social utility at the equilibrium. Thus, the price paid for the selfishness of the users is a dramatically decreased social utility. This well-known phenomenon is called the *tragedy of the commons* in Economics.

Although for some games the price of selfishness is high, this does not hold in general. To quantify the degradation of the social utility, Koutsoupias and Papadimitriou have proposed the concept of *price of anarchy*, which is analogue to the notion of approximation ratio for optimization problems.

**Definition 2.2.** Let  $\Gamma$  be a normal-form game having a set of states  $S$  and let  $E \subseteq S$  be a set of equilibria such that  $E \neq \emptyset$ . The *price of anarchy* of  $\Gamma$  (with respect to  $E$ ) is the quantity

$$\text{PoA} = \frac{\max_{s \in S} \sum_{i \in N} u_i(s)}{\min_{s \in E} \sum_{i \in N} u_i(s)} \xrightarrow{\text{maximum social utility}} \xrightarrow{\text{worse equilibrium}}$$

1

$$s^* = \operatorname{argmax}_{s \in S} \sum_{i=1}^N u_i(s)$$

↳ could not be the equilibrium. How far the s.u. in the equilibrium from opt?

### 3 SELFISH ROUTING

**equilibrium of Pollution game:**

FOR EACH COUNTRY THE DOMINANT STRATEGY is TO POLLUTE:

$$\bar{s} = (s_2^1, s_2^2, \dots, s_2^m)$$

$$\text{COST}(\bar{s}) = \sum_i \text{cost}_i(s) = m^2$$

$$\hookrightarrow P_oA = \frac{m^2}{3m} = \frac{m}{3}$$

When the game is defined in terms of costs  $(c_i)_{i \in N}$ , we instead use the definition

$$P_oA = \frac{\max_{s \in E} \sum_{i \in N} c_i(s)}{\min_{s \in S} \sum_{i \in N} c_i(s)}$$

→ maximum equilibrium cost, worst case  
→ minimum cost of social utility

Thus with this terminology the price of anarchy of the Bandwidth Sharing game (with respect to pure Nash equilibria) is  $\Theta(n)$ . Notice from the definition that the price of anarchy of any game is always at least 1. Notice also that in case the game admits multiple equilibria, the definition implicitly assumes that the worst one will occur (so that the guarantee always holds).

has 2 strategy

**Exercise 2.1** (The pollution game). In this game there are  $n$  countries (the players). Each country can decide of either passing the legislation to control pollution or not. Pollution control has a cost 3 for the country, while every country that pollutes adds 1 to the cost of all countries. Find the price of anarchy of the game.

### 3 Selfish routing

$S_1: \text{POLLUTION CONTROL}$ $\text{COST}_i(S_1) = 3$	$S_2: \text{POLLUTE}$ $\text{COST}_i(S_2) = 1$ $\forall j \neq i, \text{COST}_j(S_2) = +1$	IF $n \geq 4$ , (for $m < 4$ best solution is to all pollute) $S^* = (S_1^1, S_1^2, \dots, S_1^n)$ $\text{COST}(S^*) = 3m$
---	--	--

We have seen that the inefficiency of equilibria can be in general high and might not scale well with the dimension of the system being analyzed. However, it is possible to identify games for which equilibria are in fact approximately optimal; that is, games with bounded price of anarchy. We will now discuss a model of network games where this happens.

**Pigou's example.** Consider the simple network shown in Figure 1. Two disjoint arcs connect a source vertex  $s$  to a destination vertex  $t$ . Each arc is labeled with a cost function  $c(\cdot)$  describing the cost (for example, the travel time) incurred by users of the arc, as a function of the amount of traffic routed on the arc. In the example, the upper arc has constant cost  $c_1(x) = 1$ . The cost of the lower arc is  $c_2(x) = x$  and thus increases as the arc gets more congested. In fact, the lower route is cheaper than the upper route as long as less than one unit of traffic uses it.

$t = 1$  unit of flow

Assume that one unit of traffic has to be routed in the network. This traffic is controlled by a very large population of players, each player controlling a negligible (infinitesimal) amount of flow from  $s$  to  $t$ . In fact, this is called a "nonatomic" selfish routing game because the decision of an individual player alone has no significant effect on the game. We assume that each player is minimizing its own cost, which is the sum of the costs of the arcs of the route he selects. Then we can see that the lower route is a dominant strategy for all players, so in the resulting dominating strategy equilibrium every

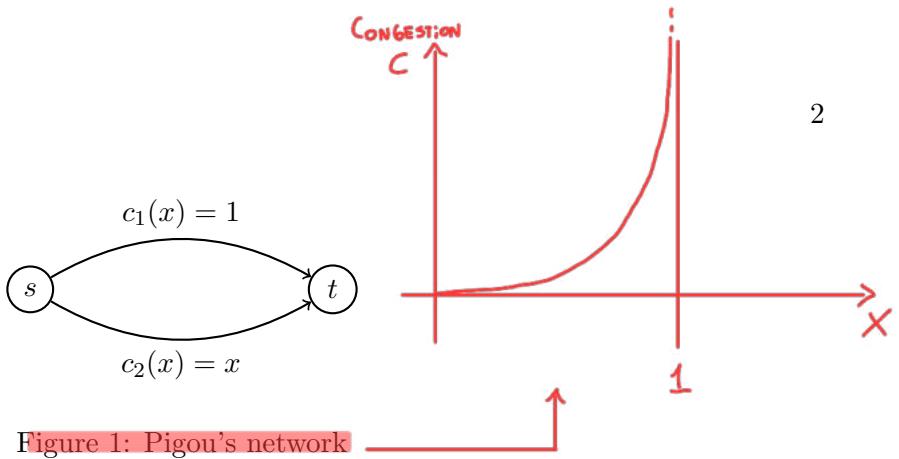


Figure 1: Pigou's network

$$\bar{x} = 1 \text{ DOMINANT STRATEGY} \rightarrow C(\bar{x}) = 1 \rightarrow P_0 A = \frac{1}{\frac{3}{4}} = \frac{4}{3}$$

$$x^* = \frac{1}{2} \text{ best social utility} \rightarrow C(x^*) = 3/4$$

all unit to lower link

$$3 \text{ SELFISH ROUTING } C_2(x) = x \cdot C_2(x) = x^2, C(x) = (1-x) \cdot 1 + x \cdot x = 1 - x + x^2$$

↓  
cost of send x unit to lower link  
↓  
1-x unit to upper link

player incurs one unit of cost. The social cost at the equilibrium is 1 (there is one unit of traffic, and all traffic incurs a cost of 1). In fact this is the only pure Nash equilibrium of the game.

What is the optimal social cost? If we send an amount of  $x$  ( $0 \leq x \leq 1$ ) on the upper link, and  $1-x$  on the lower, we obtain a cost of 1 for the players using the upper link, and a cost of  $1-x$  for the players using the lower link. The social cost is thus

$$C(x) = x \cdot 1 + (1-x) \cdot (1-x) = 1 - x + x^2.$$

Since  $C(x) = 2x - 1$  and  $C''(x) > 0$ , the social cost is minimized when  $x = 1/2$ , that is, when the traffic is evenly split. In this case we get a social cost of  $3/4$ . Thus the price of anarchy in this game (with respect to pure Nash equilibria) is equal to  $4/3$ .

What happens to the price of anarchy in more complex networks, when we increase the number arcs, or when there are multiple sources and destinations, or when we have non-linear latency functions? This is what we will study in the following.

### 3.1 The selfish routing model

A selfish routing game is specified by a network consisting of a directed graph  $G = (V, A)$  with node set  $V$  and arc set  $A$ , together with a set  $(s_1, t_1), \dots, (s_k, t_k)$  of source-sink pairs (commodities). Each player carries an infinitesimal amount of flow associated with one commodity. We denote with  $\mathcal{P}_i$  the set of  $s_i-t_i$  paths of the network. We define  $\mathcal{P} := \bigcup_{i=1}^k \mathcal{P}_i$ . We allow the graph to contain parallel arcs between the same pair of nodes.

A state of the game is represented by a flow, that is a function  $f : \mathcal{P} \rightarrow \mathbb{R}_+$ . If  $f$  is a flow and  $P \in \mathcal{P}_i$ , we denote with  $f_P$  the amount of traffic of commodity  $i$  that travels along the path from  $s_i$  to  $t_i$ . The game specifies a fixed demand  $r_i \geq 0$  of traffic corresponding to each commodity  $i$ . A flow is feasible for a demand vector  $r \in \mathbb{R}_+^k$  if for all commodities  $i$ ,  $\sum_{P \in \mathcal{P}_i} f_P = r_i$ .

Each arc  $a \in A$  has an associated cost function  $c_a : \mathbb{R}_+ \rightarrow \mathbb{R}_+$ . The cost functions are assumed to be nonnegative, continuous and nondecreasing. A nonatomic selfish routing game can thus be summarized by the triple  $(G, r, c)$ , where  $G$  is the graph,  $r$  is the demand vector and  $c$  is the vector of cost functions.

What is the cost incurred by the players? If a player is routing along a path  $P$  and the global state of the game is represented by the flow  $f$ , the cost of the player is

$$c_P(f) := \sum_{a \in P} c_a(f_a)$$

where  $f_a := \sum_{P \in \mathcal{P}: a \in P} f_P$  denotes the total amount of traffic using arc  $a$ .

*Remark 3.1.* Be careful not to confuse the two notations  $f_P$  and  $f_a$ . The first gives the flow traveling along a certain path  $P$ , without considering flows associated to other paths, even when they have some arc in common with the path  $P$ . The quantity  $f_a$  gives instead the total amount of flow traveling along a given arc  $a$ . We will avoid confusion by using capital letters for paths and small letters for arcs.

The social cost is measured as follows: the cost of a flow  $f$  is given by

$$C(f) := \sum_{P \in \mathcal{P}} f_P c_P(f).$$

$$C_p(F) = \sum_{a \in P} c_a(f_a)$$

model:

$$G = (V, A)$$

$$(s_1, t_1) \dots (s_k, t_k)$$

$r_1, \dots, r_k$  demand of flow from  $s_i$  to  $t_i$

$P_i$ : set of paths from  $s_i$  to  $t_i$

$$P = \bigcup_{i=1}^k P_i, F : P \rightarrow \mathbb{R}^+ \text{ state of game equal to flow}$$

cost of a flow

$$c_a : \mathbb{R}^+ \rightarrow \mathbb{R}^+$$

$$c_a(f_a) = c_a \left( \sum_{P \in \mathcal{P}} f_P \right)$$

strategy of agent i

$$\sum_{a \in P} f_a = r_i = \text{sum of flow in each path} \rightarrow \text{feasible solution must be } r_i$$

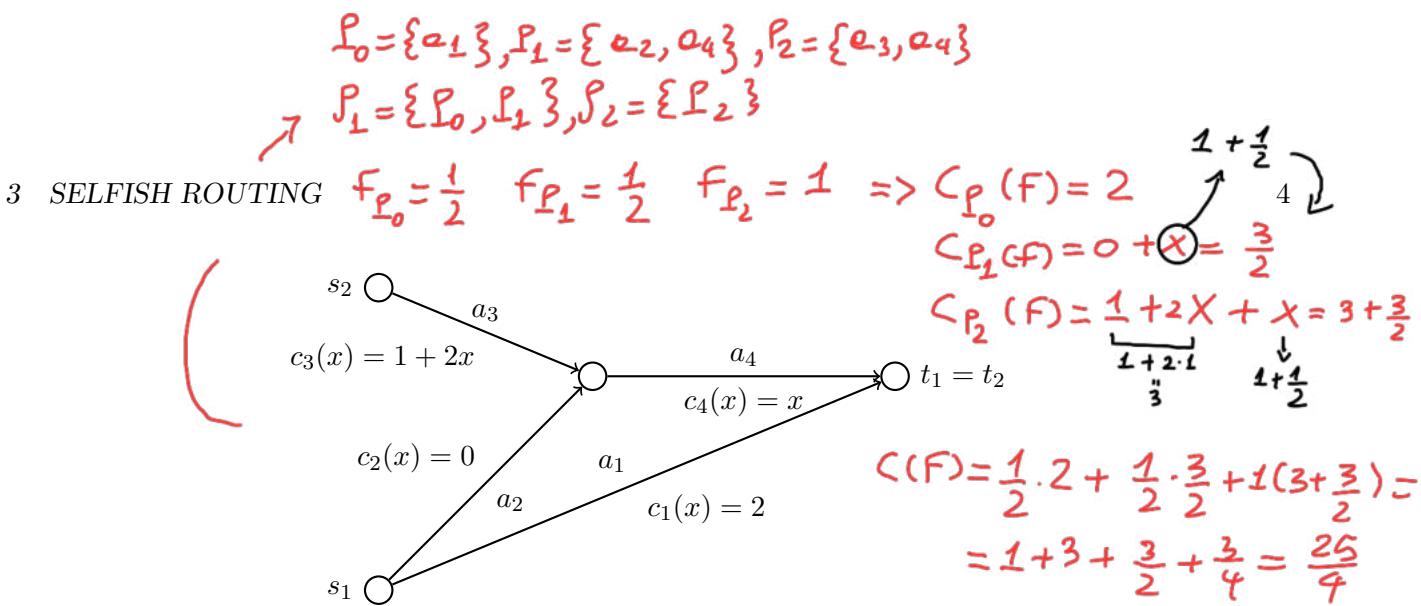


Figure 2: An example network

Equivalently (prove this), this can be expressed as the sum over all arcs

$$C(f) = \sum_{a \in A} f_a c_a(f_a).$$

**Example 3.1.** Consider the network in Figure 2. Here we assume two commodities ( $k = 2$ ) with a demand of one each (demand vector  $r = (1, 1)$ ). The source and destinations are  $(s_1, t_1)$  for the first commodity and  $(s_2, t_2)$  for the second. The graph contains two  $s_1-t_1$  paths ( $\mathcal{P}_1 = \{P_0, P_1\}$ ) and one  $s_2-t_2$  path ( $\mathcal{P}_2 = \{P_2\}$ ). The paths are the following:  $P_0 = \{a_1\}$ ,  $P_1 = \{a_2, a_4\}$ ,  $P_2 = \{a_3, a_4\}$ . Finally, the cost functions are  $c_1(x) = 2$ ,  $c_2(x) = 0$ ,  $c_3(x) = 1 + 2x$ ,  $c_4(x) = x$ .

An example of feasible flow for the network is the flow  $f$  defined by  $f_{P_0} = 1/2$ ,  $f_{P_1} = 1/2$ ,  $f_{P_2} = 1$ . Notice that in this case  $f_{a_1} = f_{P_0} = 1/2$ ,  $f_{a_2} = f_{P_1} = 1/2$ ,  $f_{a_3} = f_{P_2} = 1$ ,  $f_{a_4} = f_{P_1} + f_{P_2} = 3/2$ . The cost of this flow is thus  $C(f) = (1/2) \cdot 2 + (1/2) \cdot 0 + 1 \cdot 3 + (3/2) \cdot (3/2) = 25/4$ .

We can now define an equilibrium concept for nonatomic selfish routing games.

**Definition 3.2.** Let  $f$  be a feasible flow for the nonatomic instance  $(G, r, c)$ . The flow  $f$  is an equilibrium flow if, for every commodities  $i = 1, \dots, k$  and every pair of paths  $P, P' \in \mathcal{P}_i$ ,

↳ always exist!      if  $f_P > 0$  then  $c_P(f) \leq c_{P'}(f)$ .

Notice that a consequence of the definition is that in an equilibrium flow, all nonzero  $s_i-t_i$  path flows  $f_P$  have the same cost, irrespective of  $P$ , for any  $P \in \mathcal{P}_i$ .

**Example 3.3.** In Pigou's example (Figure 1), the flow that sends all the traffic along the lower arc is an equilibrium flow.

**Example 3.4.** Consider again Example 3.1. The flow  $f$  mentioned in the example is not an equilibrium flow: consider the paths  $P_0$  and  $P_1$ . We have  $f_{P_0} > 0$  while  $c_{P_0}(f) = 2 > c_{P_1}(f) = 3/2$ . Thus, the players on the path  $P_0$  have an incentive to change their route to  $P_1$ .

Instead, the flow  $g$  defined by  $g_{P_0} = 0$ ,  $g_{P_1} = 1$ ,  $g_{P_2} = 1$  is an equilibrium flow (check the inequalities!).

$$\bar{x} = \frac{1}{2} \text{ only equilibrium here}$$

$$CC(\bar{x}) = \frac{1}{2}(1+\frac{1}{2}) + \frac{1}{2}(\frac{1}{2}+1) = \frac{3}{2}$$

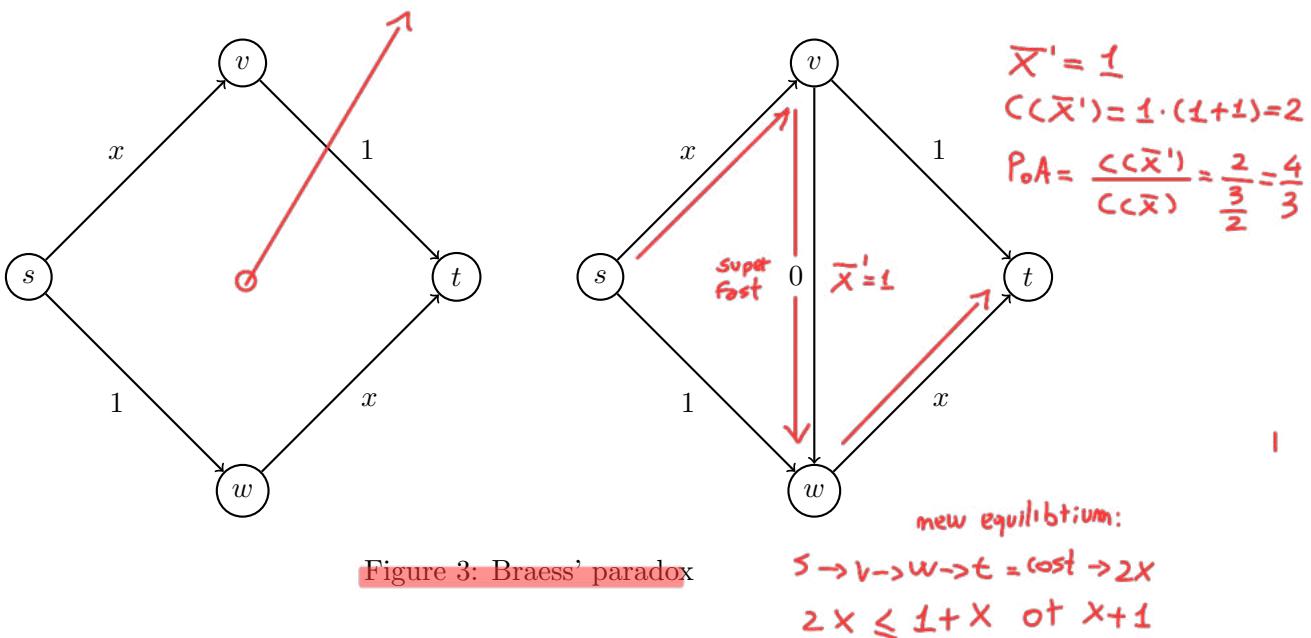


Figure 3: Braess' paradox

**Example 3.5** (Braess' paradox). The following example shows that in selfish routing games counter-intuitive phenomena can arise. Consider the network on the left of Figure 3. There are two routes, each with combined cost  $1 + x$  when  $x$  traffic is sent along the route. Assume that there is a unit demand of traffic to be routed. Then the equilibrium flow is to split the traffic evenly (why?) and the total cost experienced by the traffic is  $3/2$ .

Now suppose that, in order to decrease the cost encountered by the traffic, we build a zero-cost arc connecting the midpoints of the existing routes, as on the right of Figure 3. What is the new equilibrium? There is a new route  $s \rightarrow v \rightarrow w \rightarrow t$  and using this route is a (weakly) dominant strategy. Thus at equilibrium all the traffic is sent along this path. The total cost becomes thus 2, which is higher than before!

### 3.2 Existence of equilibrium flows

Our aim of this section is to show that a nonatomic selfish routing game always admits an equilibrium flow. We will make use of a characterization of optimal flows.

**Definition 3.6.** An optimal flow for instance  $(G, r, c)$  is a feasible flow  $f$  that minimizes  $\sum_{a \in A} f_a c_a(f_a)$ .

To simplify the discussion, assume that for any arc  $a$ , the function  $x \cdot c_a(x)$  is continuously differentiable and convex. Recall that  $x \cdot c_a(x)$  is the contribution to the social cost of the traffic on arc  $a$ . Let  $c_a^*(x) := (x \cdot c_a(x))' = c_a(x) + x c_a'(x)$  denote the marginal cost function for arc  $a$ . For example, if  $c_a(x) = x^3$  then  $c_a^*(x) = 4x^2$ . Recall that with  $c_P^*(f)$  we mean the quantity  $\sum_{a \in P} c_a^*(f_a)$ . Then it is possible to prove the following characterization (we omit the proof).

**Proposition 3.1.** Under the above hypotheses, a feasible flow  $f^*$  is an optimal flow for  $(G, r, c)$  if and only if, for every commodity  $i = 1, 2, \dots, k$  and every pair of paths  $P, P' \in \mathcal{P}_i$ ,

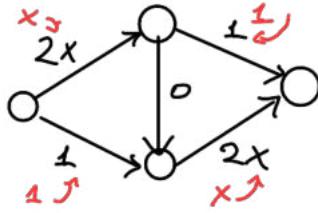
$$\text{if } f_P^* > 0 \text{ then } c_P^*(f^*) \leq c_{P'}^*(f^*).$$

$f^*$  is a feasible flow for marginal network. Replace the cost of each arc with marginal cost.

DEF.: marginal cost

$$c_a^*(x) = (x \cdot c_a(x))' = c_a(x) + x \cdot c_a'(x)$$

### 3 SELFISH ROUTING



$$\begin{aligned} c_a^*(x) &= (x^2)' = 2x \\ c_{a_1}^*(x) &= (1 \cdot x)' = 1 \\ c_1 &= 1 + 2x \\ c_2 &= 1 + 2x \quad \text{equivalent } \frac{1}{2} \end{aligned}$$

Notice the striking similarity between the above condition and the one in Definition 3.2! In fact, the conditions are the same, except that one is defined on the original costs  $c$  of the network, and the other on the marginal costs  $c^*$ . So we could say that optimal flows and equilibrium flows can be defined in the “same” way, but with respect to different cost functions.

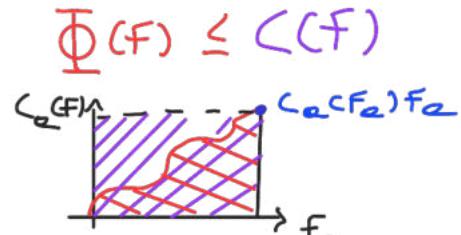
**Corollary 3.2.** A feasible flow is an optimal flow for  $(G, r, c)$  if and only if it is an equilibrium flow for  $(G, r, c^*)$ .

The idea now is the following: since optimal flows always exist (no matter which global function we use), we show that equilibrium flows exist by seeing them as optimal flows for a different notion of social cost. That is, we want to make a statement similar to Corollary 3.2, but in the “opposite” direction.

Let  $h_a(x) := \int_0^x c_a(y) dy$ . We make this choice because now  $h'_a(x) = c_a(x)$ . We also use the following definition.

**Definition 3.7.** The function

$$\Phi(f) := \sum_{a \in A} \int_0^{f_a} c_a(x) dx = \sum_{a \in A} h_a(f_a)$$



is called the *potential function* of a nonatomic instance  $(G, r, c)$ .

We can now invoke Proposition 3.1, except that we consider the minimization of  $\Phi(\cdot)$  instead of the minimization of  $C(\cdot)$ ; that is, we use the function  $h_a(x)$  in place of  $x \cdot c_a(x)$ . Then what was before  $c_a^*(x) = (x \cdot c_a(x))'$  is now  $h'_a(x) = c_a(x)$ , so that the second part of Proposition 3.1 is stating that we have an equilibrium flow with respect to costs  $(c_a)_{a \in A}$ . We have reformulated Proposition 3.1 as follows.

**Proposition 3.3.** A feasible flow is an equilibrium flow for  $(G, r, c)$  if and only if it is a global minimum of the corresponding potential function  $\Phi$  given in Definition 3.7.

Armed with this proposition we can now prove our main existence result.

**Theorem 3.4.** Let  $(G, r, c)$  be a nonatomic instance. Then

1. The instance  $(G, r, c)$  admits at least one equilibrium flow.
2. If  $f$  and  $g$  are equilibrium flows for  $(G, r, c)$  then  $c_a(f_a) = c_a(g_a)$  for every arc  $a$ .

*Proof.* By its definition, the set of feasible flows is a compact (= closed and bounded) subset of  $\mathbb{R}^{|P|}$ . Arc cost functions are continuous, so the potential function is also continuous. By Weierstrass’ Theorem from elementary analysis (remember?),  $\Phi$  achieves a minimum on this set. By Proposition 3.3, every one of this minima corresponds to an equilibrium flow of  $(G, r, c)$ . This proves (1).

Part (2) can be proved by using the fact that  $f$  and  $g$  are both minimizers of  $\Phi$  and the fact that  $\Phi$  is convex, thus it must be constant between  $f$  and  $g$ . Moreover  $\Phi$  is the sum of convex terms, so every term  $\int_0^x c_a(x) dx$  must be linear between  $f$  and  $g$ , which implies that  $c_a$  is constant between  $f$  and  $g$ .  $\square$

**Example 3.8.** We can use Corollary 3.2 to find an optimal flow for the Braess network (right part of Figure 3). If we compute the marginal cost functions, we obtain  $(x \cdot x)' = 2x$  for the arcs that had cost  $x$ , while the arcs with constant cost (0 and 1) remain unaltered (because for example  $(x \cdot 1)' = 1$ ). If we search an equilibrium flow in the network with modified costs, we obtain the flow that sends half unit of traffic on the upper path, half unit on the lower path, and no unit on the zig-zag path. By Corollary 3.2 this flow is the optimal flow for the network with the original cost functions. It has cost  $3/2$ . On the other hand, we saw in Example 3.5 that the equilibrium flow has cost 2. So the price of anarchy for the Braess graph is  $4/3$  (as it was for the Pigou network).

## 4 The price of anarchy for selfish routing

### 4.1 Bound using the potential function

Having proved in the previous section that an equilibrium flow always exists, we can now analyze the price of anarchy of selfish routing.

We first show that the type of cost functions plays an important role.

**Example 4.1** (Nonlinear Pigou). Consider again Pigou's network, except that now the linear cost function  $c_2(x) = x$  is replaced by the quadratic function  $c_2(x) = x^2$ .

It is easy to see that once more, an equilibrium flow sends all the demand (which is equal to 1) on the lower link, for a social cost of 1. What is the optimal flow? If we send an amount of  $x$  ( $0 \leq x \leq 1$ ) on the upper link, and  $1 - x$  on the lower, we obtain a cost of  $1$  for the players using the upper link, and a cost of  $(1 - x)^2$  for the players using the lower link. The social cost is thus

$$C(x) = x \cdot 1 + (1 - x) \cdot (1 - x)^2 = x + (1 - x)^3.$$

As  $C'(x) = 1 - 3(1 - x)^2$ , we obtain the minimum when  $(1 - x)^2 = 1/3$ , that is when  $x = 1 - 1/\sqrt{3}$ . The cost in this case is  $1 - (2/3)(1/\sqrt{3}) \simeq 0.615$ . Thus the price of anarchy is  $\simeq 1/0.615 \simeq 1.626$ . Notice that this is larger than the  $4/3$  we obtained with a linear cost function.

In fact, if we generalize the example with  $c_2(x) = x^p$ , we obtain a price of anarchy growing roughly as  $p/\ln p$ . Thus the price of anarchy is not bounded if we do not limit the class of cost functions.

The example shows that if the cost functions can be “highly nonlinear”, the price of anarchy can be very high even on very simple networks. But what if the cost functions are for example linear or quadratic? Can we obtain a useful bound in this case?

**Theorem 4.1.** Suppose that  $x \cdot c_a(x) \leq \gamma \cdot \int_0^x c_a(y) dy$  for all  $a \in A$  and  $x \geq 0$ . Then the price of anarchy of  $(G, r, c)$  is at most  $\gamma$ .

*Proof.* Let  $f$  be an equilibrium flow and let  $f^*$  be an optimal flow. Since cost functions are non-decreasing, the cost of a flow is always at least its potential function value (why?), in particular  $C(f^*) \geq \Phi(f^*)$ . By the hypothesis, the cost of any flow is at most  $\gamma$  times the potential value of the flow. We can conclude, using Proposition 3.3,

$$C(f) \leq \gamma \cdot \Phi(f) \leq \gamma \cdot \Phi(f^*) \leq \gamma \cdot C(f^*).$$

□

$$c_a(x) = x^p \quad x^{p+1} \leq y \int_0^x x^p dx = \frac{1}{p+1} x^{p+1}$$

$$y = p+1$$

**Corollary 4.2.** *The price of anarchy in nonatomic instances with cost functions that are polynomials of degree at most  $p$  (with nonnegative coefficients) is at most  $p + 1$ .*

*Proof.* Let  $c_a$  be such a cost function. Then for any  $y \geq 0$

$$y \cdot c'_a(y) \leq p \cdot c_a(y)$$

so

$$c_a(y) + y \cdot c'_a(y) \leq (p+1) \cdot c_a(y).$$

If we integrate both sides between 0 and  $x$  we obtain

$$x \cdot c_a(x) = \int_0^x (y \cdot c_a(y))' dy \leq (p+1) \int_0^x c_a(y) dy$$

for all  $x \geq 0$ .  $\square$

## 4.2 Bound using the variational inequality

In this section we improve the upper bound for *affine* cost functions (with nonnegative coefficients), that is, functions of the form  $c_a(x_a) = b_{1a}x_a + b_{0a}$  where  $b_{0a}, b_{1a} \geq 0$ . In this case Corollary 4.2 gives an upper bound of 2 on the price of anarchy. We will show an improved upper bound of  $4/3$ . Since this matches the lower bound from Pigou's example, this improved upper bound is best possible.

We will use another characterization of equilibrium flows, given by the following proposition.

**Proposition 4.3.** *Let  $f$  be a feasible flow for  $(G, r, c)$ . Then  $f$  is an equilibrium flow if and only if*

$$\sum_{a \in A} f_a c_a(f_a) \leq \sum_{a \in A} g_a c_a(f_a) \tag{1}$$

for every flow  $g$  feasible for  $(G, r, c)$ .

*Proof.* Fix  $f$  and define the function  $H_f$  on the set of feasible flows as follows:

$$H_f(g) = \sum_{i=1}^k \sum_{P \in \mathcal{P}_i} g_P c_P(f) = \sum_{a \in A} g_a c_a(f_a).$$

The value  $H_f(g)$  denotes the cost of a flow  $g$  after the cost function of each arc  $a$  has been changed to the constant function everywhere equal to  $c_a(f_a)$ . By the second definition of  $H_f$ , the claim we have to prove is equivalent to the assertion that a flow  $f$  is an equilibrium flow if and only if it minimizes the function  $H_f(\cdot)$  over all feasible flows.

Consider now the first definition of  $H_f$ . From this we see that a flow  $g$  minimizes  $H_f$  if and only if it is a minimum cost flow with respect to the constant cost functions  $c_a(f_a)$ . That is, it is the optimum of a minimum cost flow problem without any capacity constraints or integrality constraints. Such a flow  $g$  is optimum if and only if it only sends flow along paths of minimum cost, that is  $g_P > 0$  only for paths  $P$  that minimize  $c_P(f)$  over all  $s_i-t_i$  paths. But such a flow is in fact an equilibrium flow, and every equilibrium flow has this property.  $\square$

Inequality (1) is called the *variational inequality*.

We can now prove our improved bound (note: the proof given here is easier than the one in the book by Nisan et al.).

**Theorem 4.4.** *Let  $f$  be an equilibrium flow of instance  $(G, r, c)$  where the functions  $(c_a)_{a \in A}$  are affine, and let  $f^*$  be an optimal flow for the same instance. Then  $C(f) \leq (4/3)C(f^*)$ .*

*Proof.* Thanks to Proposition 4.3, we have

$$\begin{aligned} C(f) &= \sum_{a \in A} f_a c_a(f_a) \\ &\leq \sum_{a \in A} f_a^* c_a(f_a) = \\ &= \sum_{a \in A} f_a^* c_a(f_a^*) + \sum_{a \in A} f_a^*(c_a(f_a) - c_a(f_a^*)) = \\ &= C(f^*) + \sum_{a \in A} f_a^*(c_a(f_a) - c_a(f_a^*)). \end{aligned}$$

For the arcs where  $f_a^* \geq f_a$ , this bound is already sufficient because the cost functions are nondecreasing and so the second term of the sum becomes negative. So it is enough to focus on arcs for which  $f_a^* < f_a$ . In this case,  $f_a^*(c_a(f_a) - c_a(f_a^*))$  is equal to the area of the shaded rectangle in Figure 4. Note that the area of any rectangle whose upper-left corner point is  $(0, c_a(f_a))$  and whose lower-right corner point lies on the line representing  $c_a(y) = b_{1a}y_a + b_{0a}$ , is at most half of the triangle defined by the three points  $(0, c_a(f_a))$ ,  $(0, b_{0a})$  and  $(f_a, c_a(f_a))$ . In particular,

$$f_a^*(c_a(f_a) - c_a(f_a^*)) \leq \frac{1}{4} f_a c_a(f_a).$$

This completes the proof, as we obtain

$$C(f) \leq C(f^*) + \frac{1}{4} C(f),$$

that is,  $C(f) \leq (4/3)C(f^*)$ . □

### 4.3 Bicriteria bound for general cost functions

As we have seen, for general cost functions it is not possible to give a bound on the price of anarchy, not even on very simple networks. However one can prove the following (we omit the proof).

**Theorem 4.5.** *If  $f$  is an equilibrium flow for  $(G, r, c)$  and  $f^*$  is any feasible flow for  $(G, 2r, c)$ , then  $C(f) \leq C(f^*)$ .*

That is, the equilibrium flow has better cost than any flow that has to send *twice as much* traffic (including the optimal flow for twice the traffic). Intuitively, this means that the inefficiency due to selfish routing can be repaired by using links that have twice the “capacity” of the original ones.

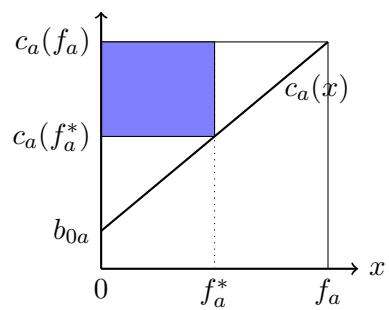


Figure 4: Illustration of the proof of Theorem 4.4

# HOMEWORK 2 2020/2021 Exercise 3

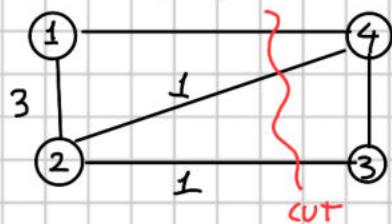
We have an undirected graph  $G = (V, E)$  with  $|V|$  players, in which each player  $i$  controls exactly one, distinct vertex  $v_i \in V$ . Each edge  $e \in E$  is associated with a nonnegative weight  $w_e$ . A cut is a partition of the set of vertices into two disjoint sets  $LEFT$  and  $RIGHT$ . Each player selects the set in which his controlled vertex will be, i.e. the strategy space of each player  $i$  is  $\alpha_i = \{LEFT, RIGHT\}$  and let  $\alpha = (\alpha_1, \dots, \alpha_n)$  be the corresponding strategy profile.

Let  $CUT(\alpha)$  be the set of edges that have one endpoint in each set and let  $N_i$  be the set of neighbours of  $v_i$  in the graph  $G$ . The payoff of player  $i$  is defined as

$$u_i(\alpha) = \sum_{e \in CUT(\alpha) \cap N_i} w_e.$$

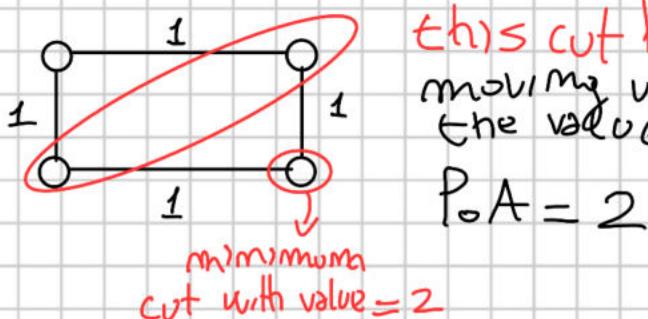
1. Design a cut game in which the Price of Anarchy is 2. Hint: Use a graph with four vertices and unit weights.
2. Prove that the Price of Anarchy of cut games is at most 2 in the general case, not only for the game with 4 vertices. Hint: First argue that in a pure Nash equilibrium, the total weight of the neighbouring vertices of a vertex  $v_i$  in the cut is at least half the total weight of all the neighbouring vertices of the vertex.

$U(2)$  given that 2 play LEFT, and 1  $\rightarrow$  LEFT , 3 and 4 play RIGHT  $\Rightarrow U(2) = 1 + 1 = 2$



$U(2) = 2$  is not an equilibrium, 2 can move to the right and obtain  $U(2) = 3$ , increase the reward

## 1. Answer



$\nearrow$  value  
 $v(OPT) \leq 2 \sum_{e \in E} w_e$

$v(EQ) \geq \sum_{v \in V} \frac{1}{2} \sum_{e \in EN(v)} w_e$

$= \frac{1}{2} \cdot 2 \sum_{e \in E} w_e = \sum_{e \in E} w_e$

