

Algorithms (Adv)

Lecture 12: Summary



THE UNIVERSITY OF
SYDNEY

Exam

Time: 10 minutes reading time
2.5 hours writing

Number of problems: 6 (ordered from easiest to hardest...imo)
First 5 problems, same as comp2007
Problem 6 is different (harder)

Advanced material

All the general techniques in comp2007...but more advanced!

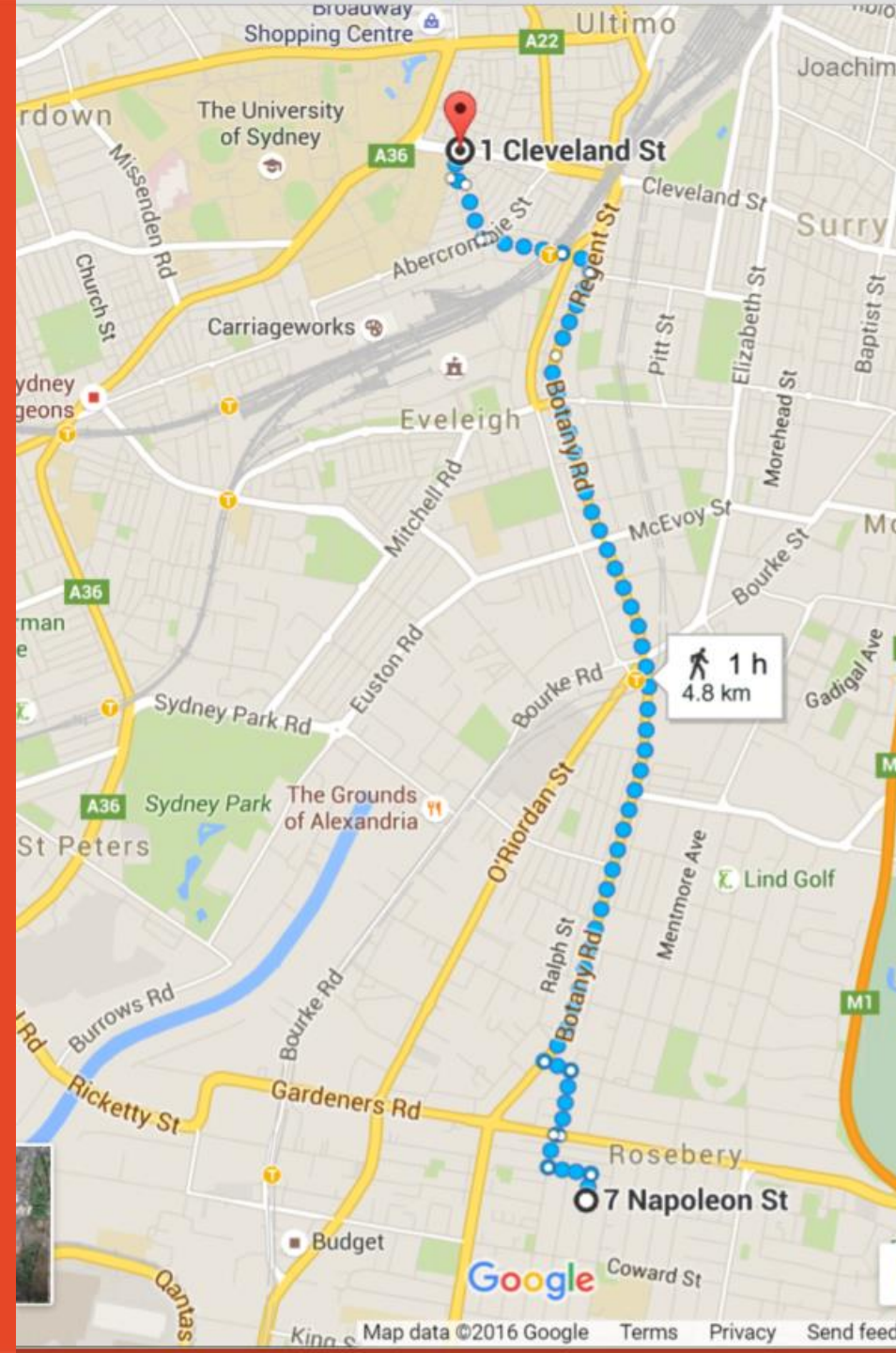
And

1. Approximation algorithms
2. Randomized algorithms (Karger's)
3. Exponential time algorithms
4. PSPACE
5. ...

Lecture 3: Greedy algorithms (Adv.)



THE UNIVERSITY OF
SYDNEY



Approximation algorithms

$$\text{Approximation ratio} = \frac{\text{Cost of apx solution}}{\text{Cost of optimal solutions}}$$

An approximation algorithm for a minimization problem requires an approximation guarantee:

- Approximation ratio $\leq c$
- Approximation solution $\leq c \cdot \text{value of optimal solution}$

Euclidean Travelling Salesman Problem (TSP)

- Vertices are points in some Euclidean space (assume 2D)
- Distance between vertices is the usual Euclidean distance.
- A set of n points (described by their (x,y) coordinates in the plane), can be considered as a **complete** weighted undirected graph on n vertices.
- **TSP problem:** Given a set of n locations (coordinates in 2D) find a travelling salesperson tour of minimum length.
- Algorithm for this problem?

Note: Metric or distance functions

A cost function $d(u,v)$ is a metric if it satisfies the following properties:

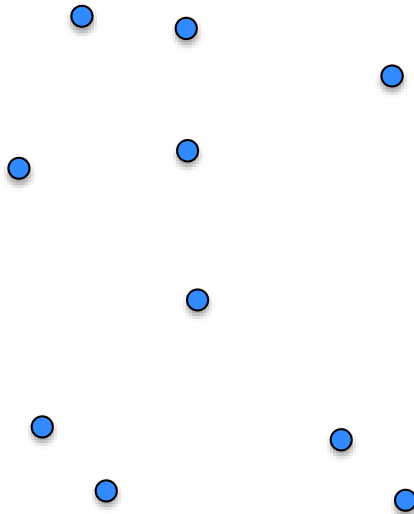
- $d(u,v) \geq 0$
- $d(u,u) = 0$
- $d(u,v) = d(v,u)$
- $d(u,v) \leq d(u,x) + d(x,v)$

Metric TSP

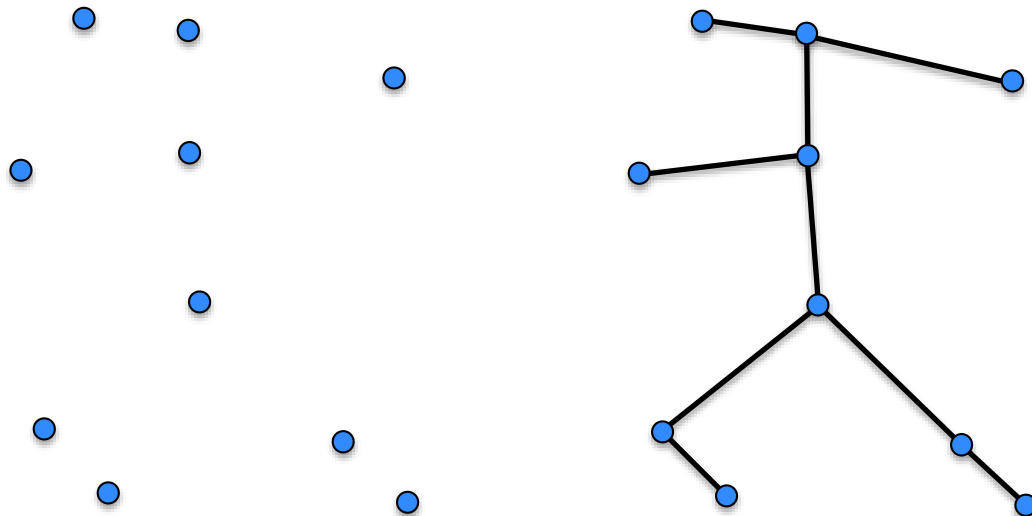
Approximate TSP(G)

1. $T := \text{MST}(G)$
2. “Double” the tree T to make it into a tour H
3. Return H as the TSP tour

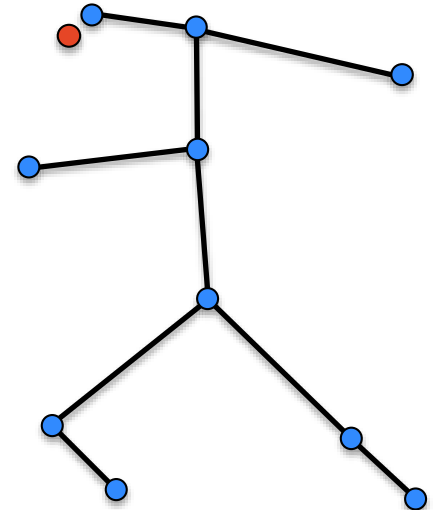
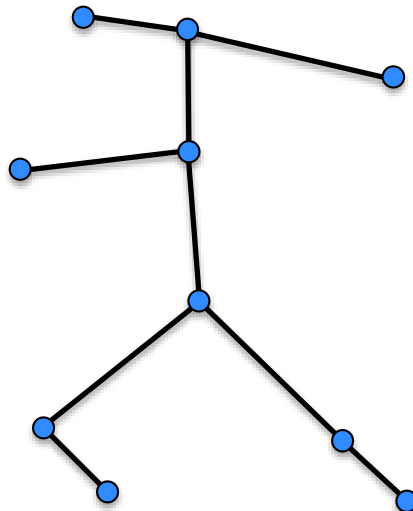
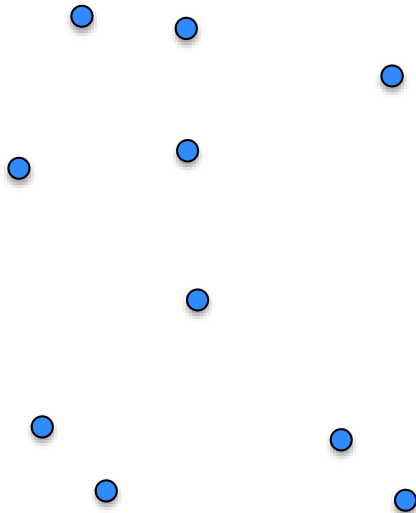
The input



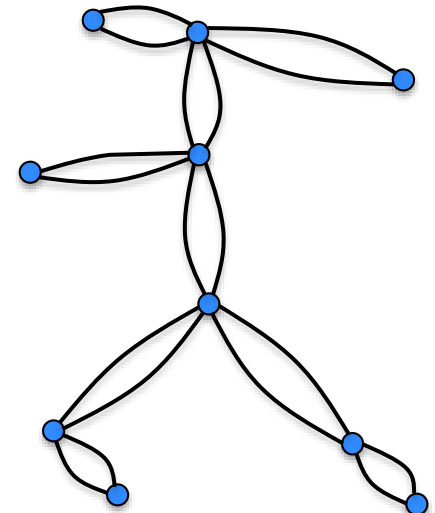
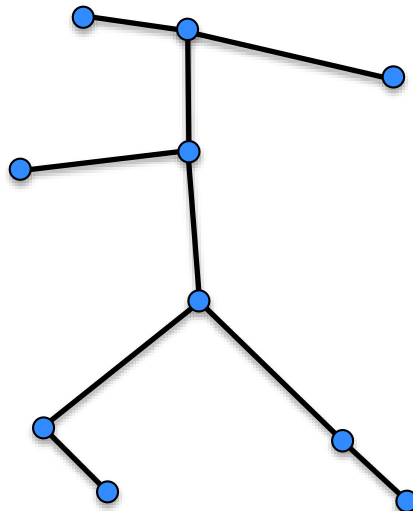
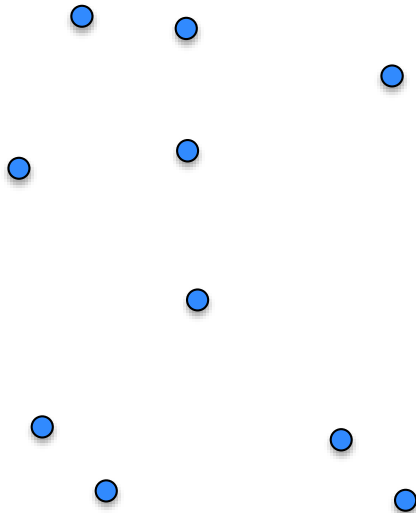
The MST



The tour



The tour



Is this a good tour?

- The tour may not be optimal (shortest possible).
- How do we prove that statement?
- We only need provide a counterexample:
A graph (input instance) for which the algorithm will return a sub-optimal tour.

2-approximation

Theorem: The MST based approximation algorithm described on the previous slides is a 2-approximation algorithm for the Metric TSP.

2-approximation

Theorem: The MST based approximation algorithm described on the previous slides is a 2-approximation algorithm for the Metric TSP.

Proof:

Assume H_{opt} is the optimal tour and that H_A is the tour returned by the approximation algorithm.

We want to prove that $\text{cost}(H_A) \leq \text{cost}(H_{\text{opt}})$.

Let T denote the MST and let W denote the walk around the MST

$$\begin{aligned} \Rightarrow \quad & \text{cost}(T) \leq \text{cost}(H_{\text{opt}}) \\ & \text{cost}(W) \leq 2 \cdot \text{cost}(T) \leq 2 \cdot \text{cost}(H_{\text{opt}}) \end{aligned}$$

SET-COVER

Instance: a finite set X and a family F of subsets of X , such that

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

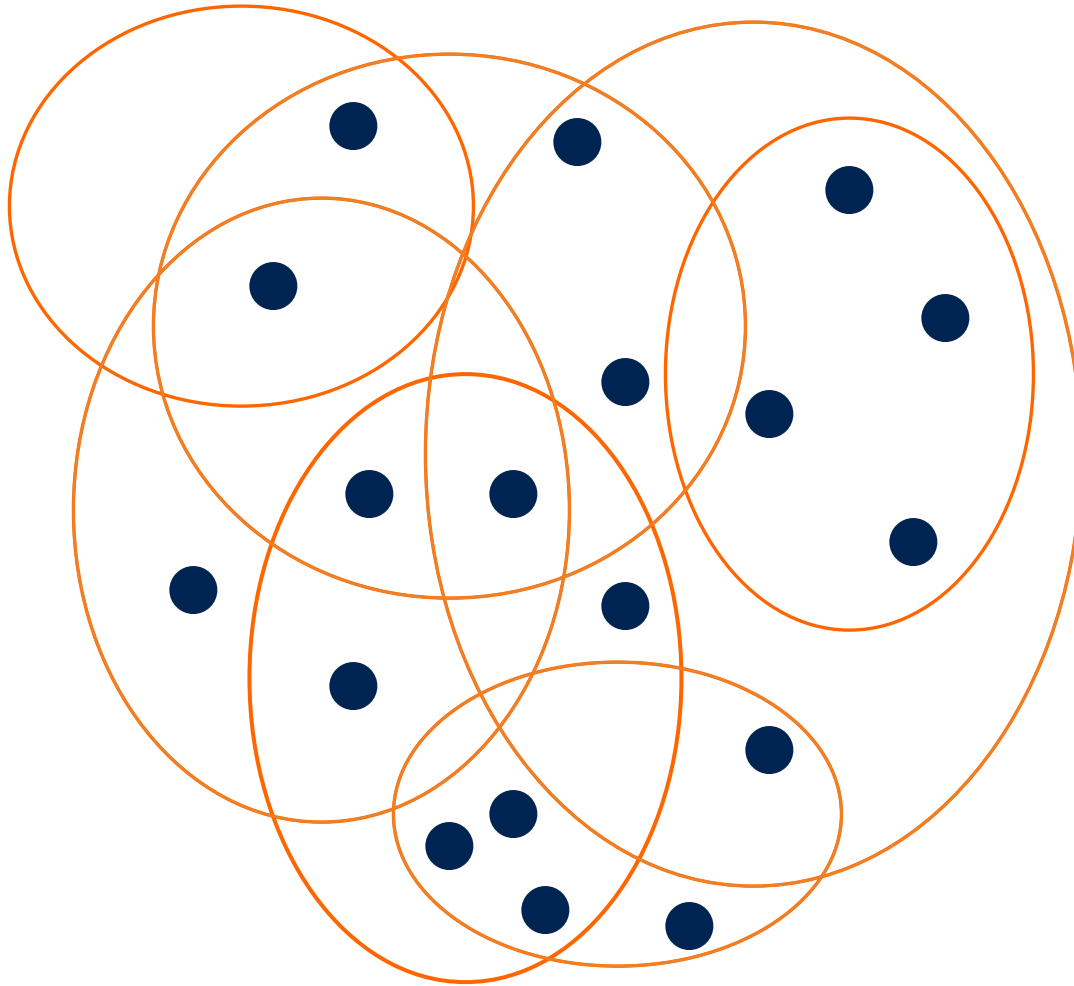
Problem: find a set $C \subseteq F$ of minimal size which covers X , i.e.

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

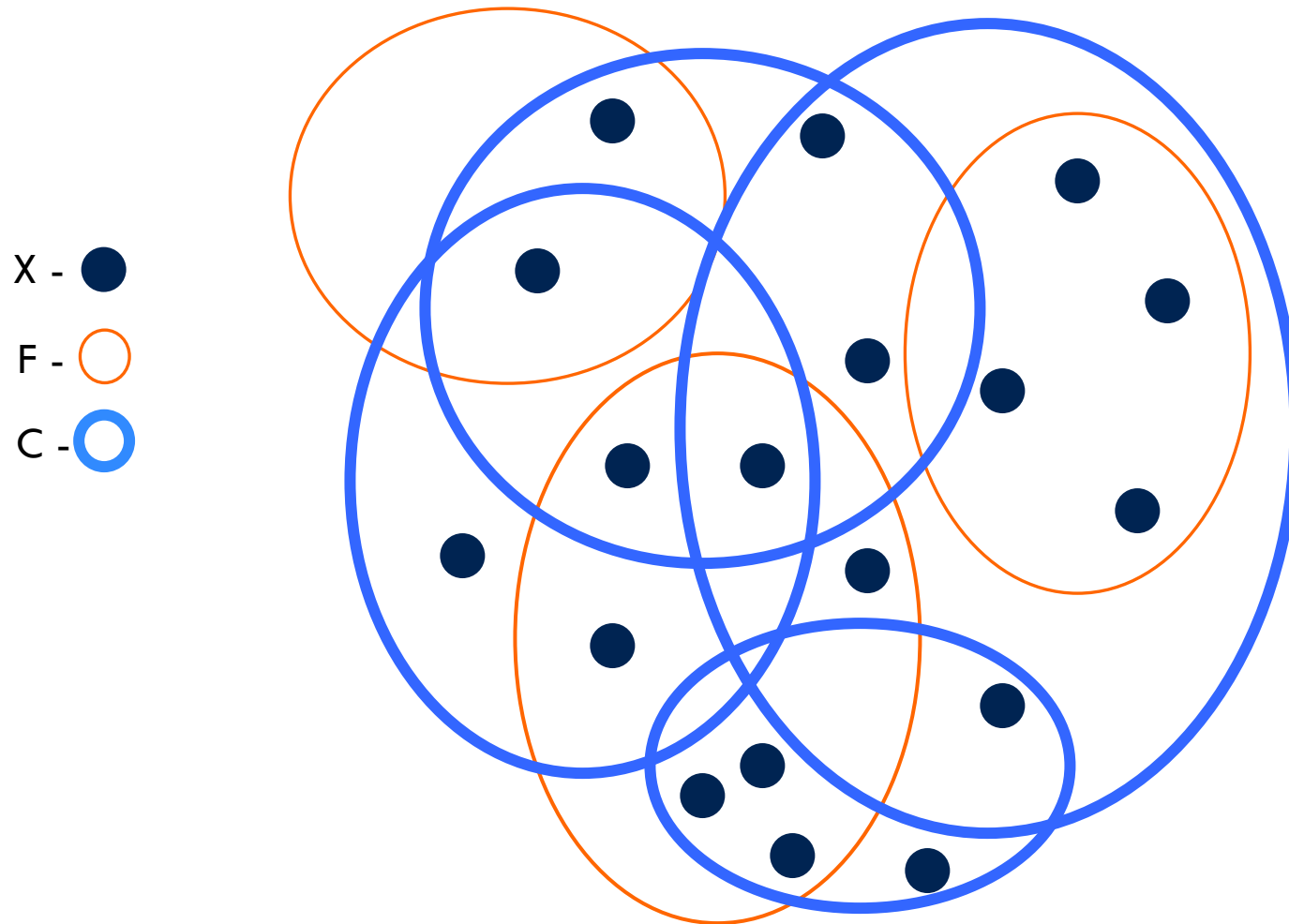
SET-COVER: Example

X - ●

F - ○



SET-COVER: Example



The Greedy Algorithm

- $C \leftarrow \emptyset$
- $U \leftarrow X$
- **while** $U \neq \emptyset$ **do**
 - select $S \in F$ that maximizes $|S \cap U|$
 - $C \leftarrow C \cup \{S\}$
 - $U \leftarrow U - S$
- **return** C

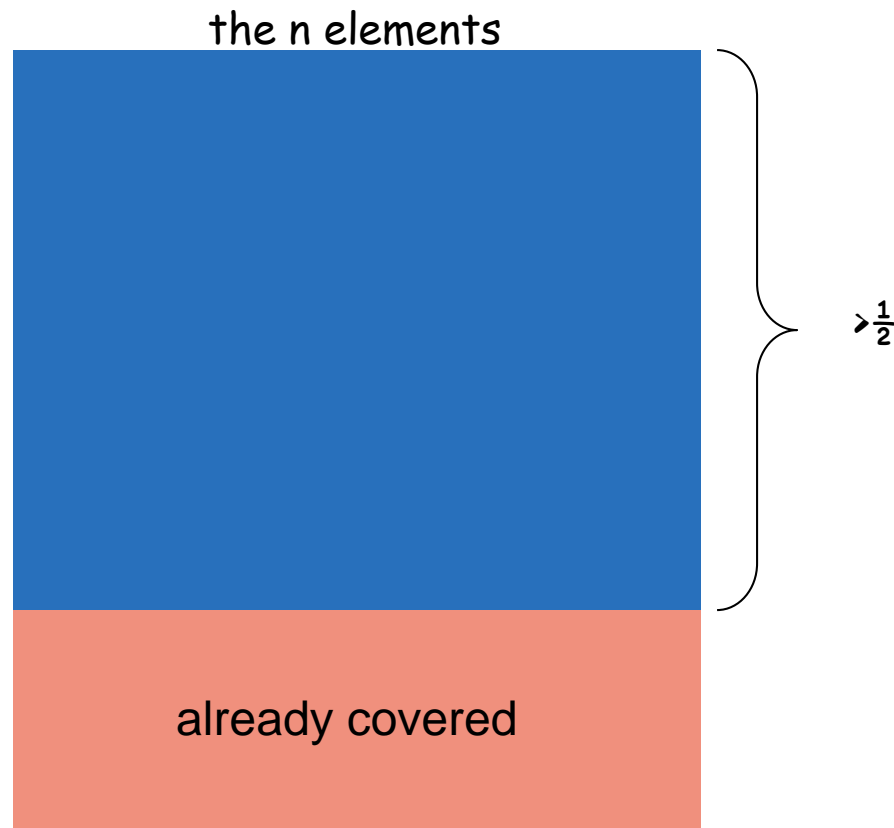
$O(|F| \cdot |X|)$

$\min\{|X|, |F|\}$

Loose Ratio-Bound

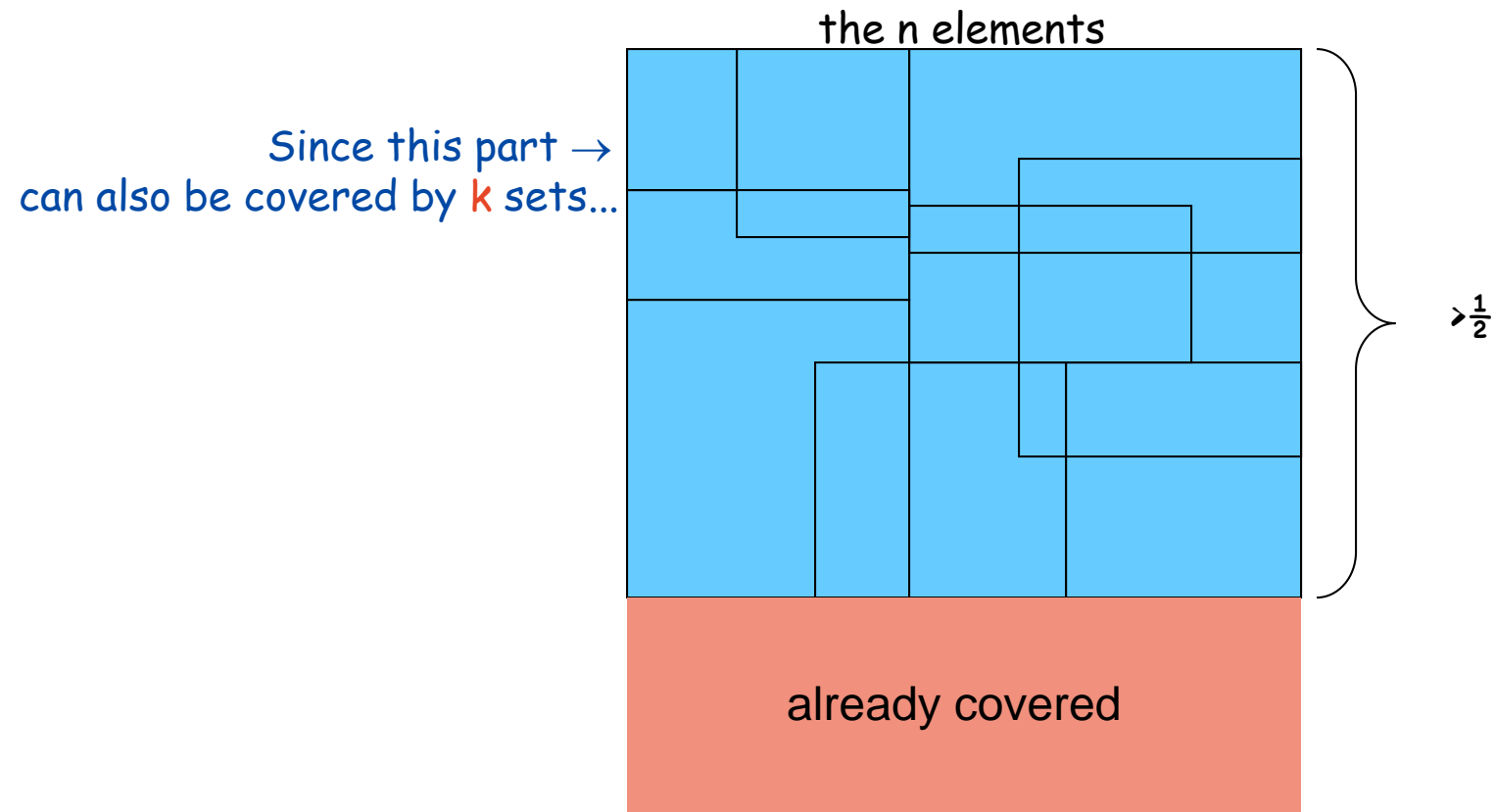
Claim: If \exists cover of size k , then after k iterations the algorithm has covered at least $\frac{1}{2}$ of the elements

Assume the opposite and observe the situation after k iterations:



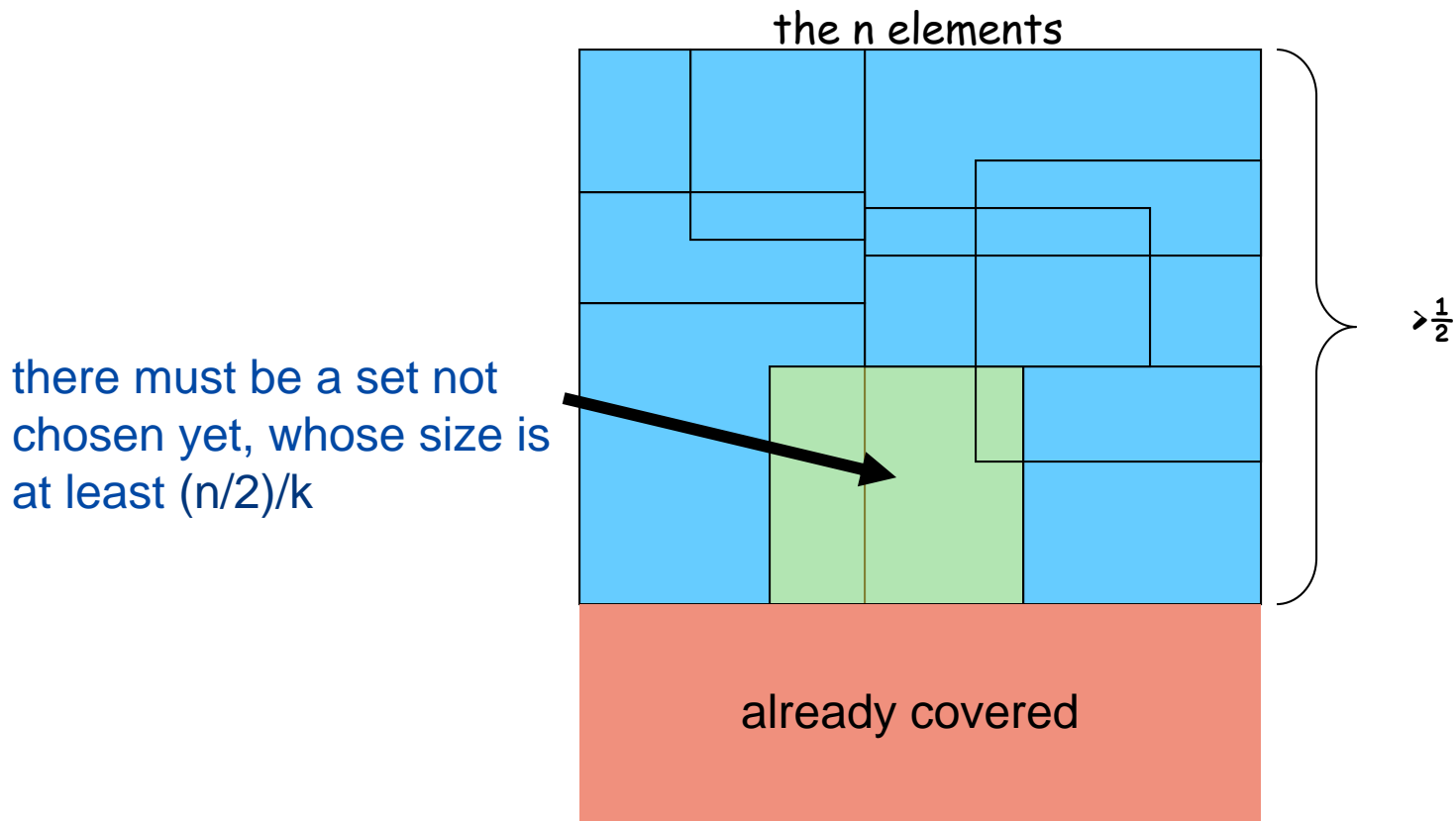
Loose Ratio-Bound

Claim: If \exists cover of size k , then after k iterations the algorithm has covered at least $\frac{1}{2}$ of the elements



Loose Ratio-Bound

Claim: If \exists cover of size k , then after k iterations the algorithm has covered at least $\frac{1}{2}$ of the elements

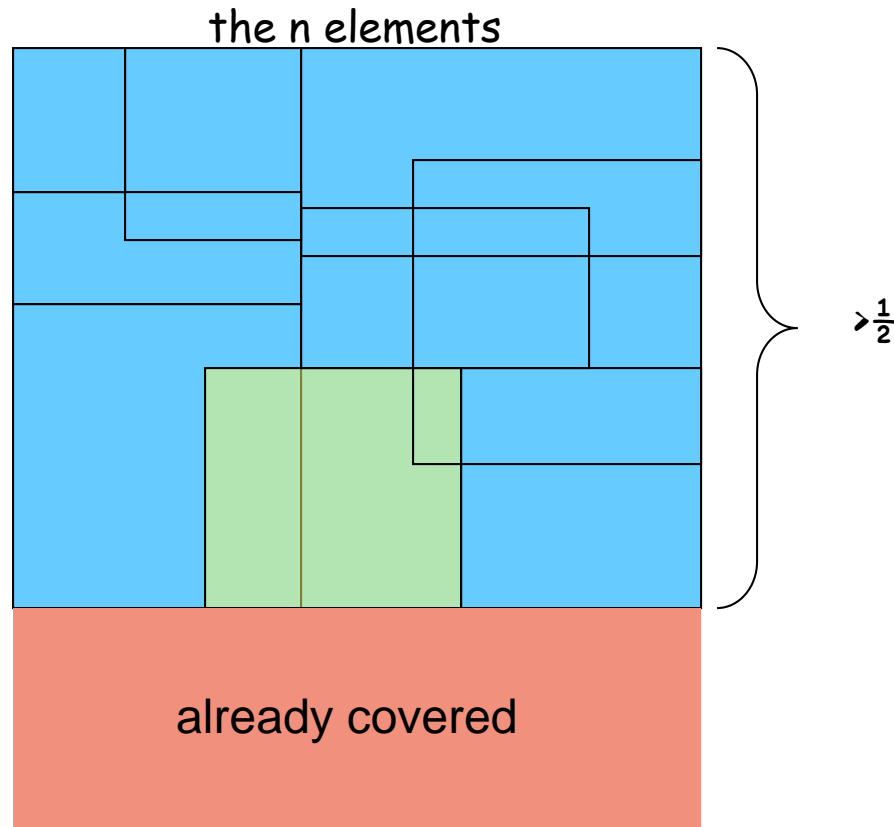


Loose Ratio-Bound

Claim: If \exists cover of size k , then after k iterations the algorithm has covered at least $\frac{1}{2}$ of the elements

and the claim is proven!

Thus in each of the k iterations we've covered at least $(n/2)/k$ new elements



Loose Ratio-Bound

Claim: If \exists cover of size k , then after k iterations the algorithm covered at least $\frac{1}{2}$ of the elements.

How many times can we half the set? $O(\log n)$ times!

Each time we perform at most k iterations.

Therefore after $k \log n$ iterations (i.e - after choosing $k \log n$ sets) all the n elements must be covered, and the bound is proved.



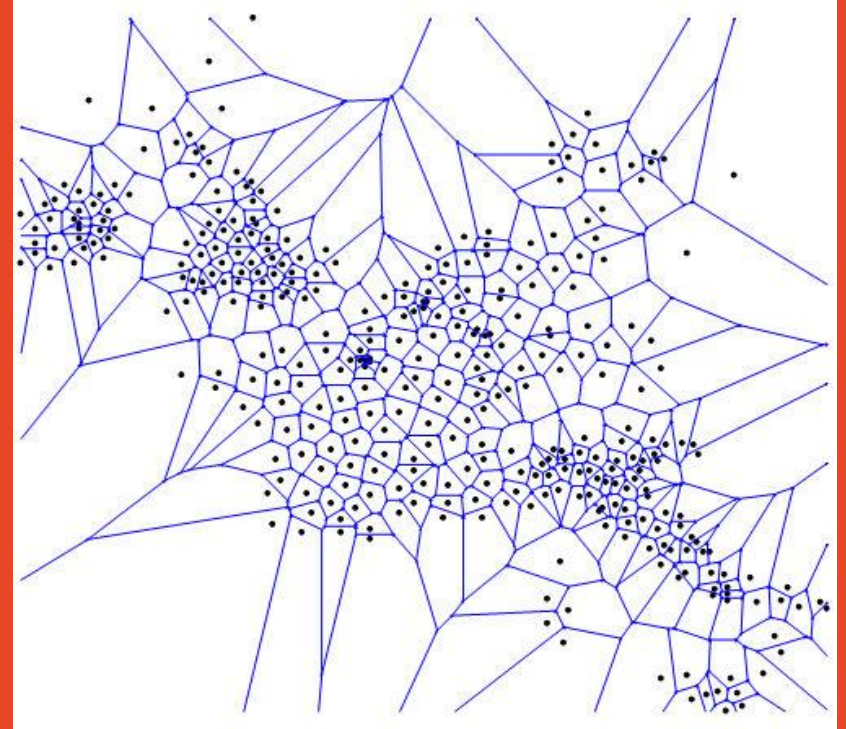
Summary: Greedy algorithms

A greedy algorithm is an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum.

Problems

- Euclidean TSP
- Set Cover
- ...

Lecture 4: Divide and Conquer (Adv.)



Matrix Multiplication

- Matrix multiplication. Given two n -by- n matrices A and B , compute $C = AB$.

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

- Brute force. $\Theta(n^3)$ arithmetic operations.
- Fundamental question. Can we improve upon brute force?

Matrix Multiplication: Key Idea

- Key idea. multiply 2-by-2 block matrices with only **7** multiplications.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &= A_{11} \times (B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12}) \times B_{22} \\ P_3 &= (A_{21} + A_{22}) \times B_{11} \\ P_4 &= A_{22} \times (B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

- 7 multiplications.
- $18 = 10 + 8$ additions (or subtractions).

Fast Matrix Multiplication

- Fast matrix multiplication. (Strassen, 1969)
 - Divide: partition A and B into $\frac{1}{2}n$ -by- $\frac{1}{2}n$ blocks.
 - Compute: 14 $\frac{1}{2}n$ -by- $\frac{1}{2}n$ matrices via 10 matrix additions.
 - Conquer: multiply 7 $\frac{1}{2}n$ -by- $\frac{1}{2}n$ matrices recursively.
 - Combine: 7 products into 4 terms using 8 matrix additions.
- Analysis.
 - Assume n is a power of 2.
 - $T(n) = \#$ arithmetic operations.

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

Fast Matrix Multiplication

Strassen(A, B)

A. If $n = 1$ Output $A \times B$

B. Else

C. Compute $A_{11}, B_{11}, \dots, A_{22}, B_{22}$

1. $P_1 \leftarrow \text{Strassen}(A_{11}, B_{12} - B_{22})$

2. $P_2 \leftarrow \text{Strassen}(A_{11} + A_{12}, B_{22})$

3. $P_3 \leftarrow \text{Strassen}(A_{21} + A_{22}, B_{11})$

4. $P_4 \leftarrow \text{Strassen}(A_{22}, B_{21} - B_{11})$

5. $P_5 \leftarrow \text{Strassen}(A_{11} + A_{22}, B_{11} + B_{22})$

6. $P_6 \leftarrow \text{Strassen}(A_{12} - A_{22}, B_{21} + B_{22})$

7. $P_7 \leftarrow \text{Strassen}(A_{11} - A_{21}, B_{11} + B_{12})$

8. $C_{11} \leftarrow P_5 + P_4 - P_2 + P_6$

9. $C_{12} \leftarrow P_1 + P_2$

10. $C_{21} \leftarrow P_3 + P_4$

11. $C_{22} \leftarrow P_1 + P_5 - P_3 - P_7$

12. Output C

D. End If

$7T(n/2)$

$O(n^2)$

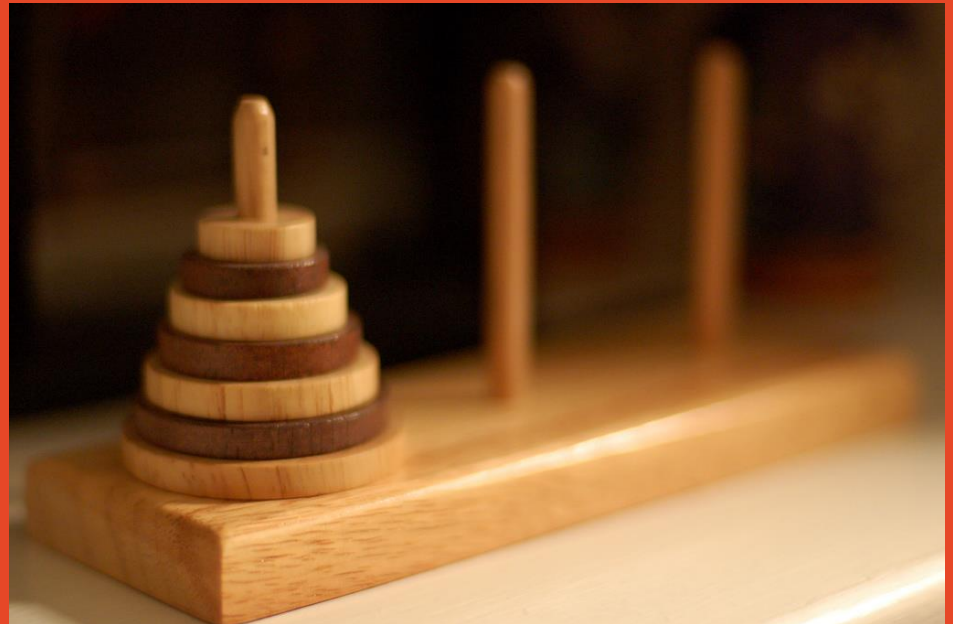
Summary: Divide-and-Conquer

- **Divide-and-conquer.**
 - Break up problem into several parts.
 - Solve each part recursively.
 - Combine solutions to sub-problems into overall solution.

Lecture 6:

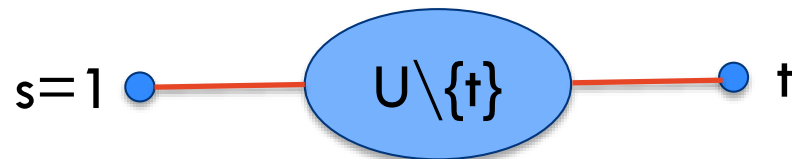
Dynamic Programming I

(Adv)



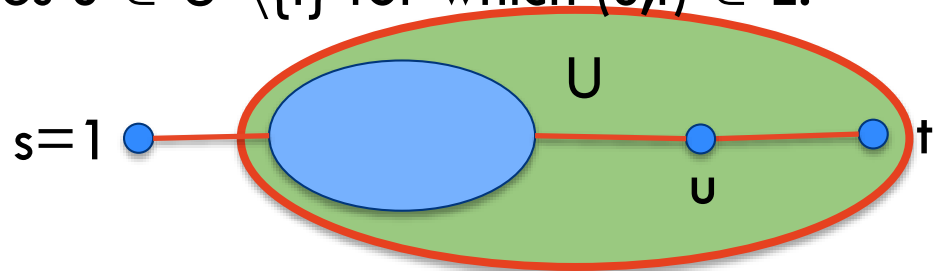
Solving TSP by Dynamic Programming

- Regard a tour to be a simple path that starts and end at vertex 1.
- Every tour consists of an edge $(1,k)$ for some $k \in V - \{1\}$ and a path from k to vertex 1. The path from vertex k to vertex 1 goes through each vertex in $V - \{1,k\}$ exactly once.
- Let $OPT[U,t]$ be the length of a shortest path starting at vertex 1, going through all vertices in U and terminating at vertex t .



Solving TSP by Dynamic Programming

- $OPT[U,t]$ = length of a shortest path starting at vertex 1 , going through all vertices in U and terminating at vertex t .
- $OPT[V,1]$ is the length of an optimal salesperson tour.
- $|U|=1 \Rightarrow OPT[U=\{t\},t] = d(s,t)$
- $|U|>1$: consider all vertices $u \in U \setminus \{t\}$ for which $(u,t) \in E$.



Observation: If a path containing (u,t) is optimal then the subpath on $U \setminus \{t\}$ ending in u must also be optimal.

Solving TSP by Dynamic Programming

$$\text{OPT}[U,t] = \min_{u \in U \setminus \{t\}} \text{OPT}[U \setminus \{t\}, u] + d(u,t)$$

Compute all solutions to subproblems in order of increasing cardinality of U .

The cost of each subproblem can be evaluated in $O(n)$ time.

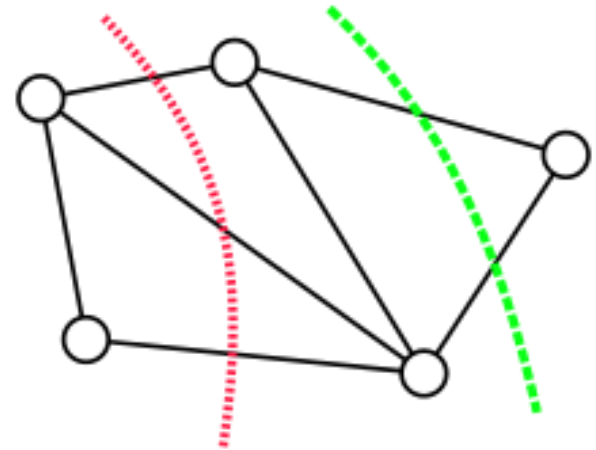
Number of subproblems?

#sets $U = 2^n$

#vertices connected to $t < n$

Total time: $O(n^2 2^n)$

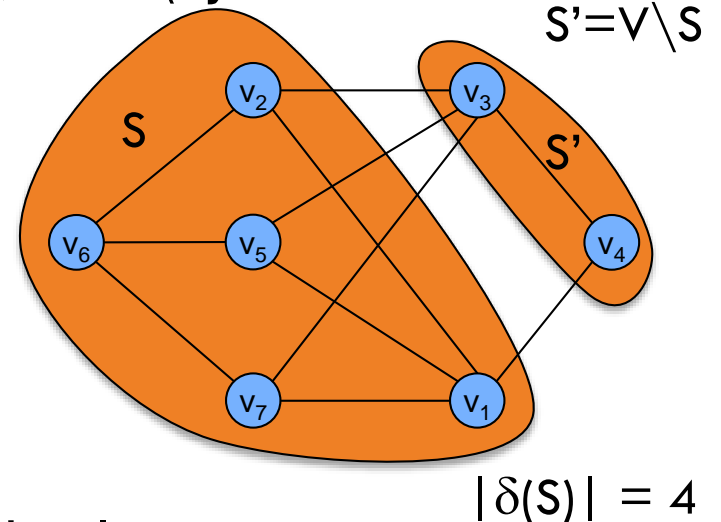
Lecture 8 (Adv): Karger's algorithms



Global Minimum Cut

Input: A connected, undirected graph $G = (V, E)$.

For a set $S \subset V$ let $\delta(S) = \{(u, v) \in E : u \in S, v \in V \setminus S\}$.



Aim: Find a cut (S, S') of minimum cardinality.

Karger's Contraction Algorithm

Definition: A multigraph is a graph that allows multiple edges between a pair of vertices.



Algorithm:

1. Start with the input graph $G=(V,E)$.
2. While $|V| > 2$ do
Contract an arbitrary edge (u,v) in G .
3. Return the cut (only one possible cut).

Karger's Contraction Algorithm

Definition: A multigraph is a graph that allows multiple edges between a pair of vertices.

Algorithm:

1. Start with the input graph $G=(V,E)$.
2. While $|V| > 2$ do
 Contract an arbitrary edge (u,v) in G .
3. Return the cut (only one possible cut).



Karger's Contraction Algorithm

Algorithm:

1. Start with the input graph $G=(V,E)$.
2. While $|V| > 2$ do
Contract an arbitrary edge (u,v) in G .
3. Return the cut S (only one possible cut).

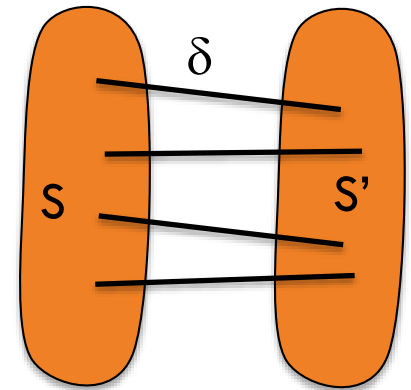
Algorithm: Since S^* is a minimum cut it has few edges!

Claim: This algorithm has a **reasonable** chance of finding a minimal cut.

Prove the claim

Claim: The algorithm returns a minimal cut with probability $\geq 2/n^2$.

Proof: Consider a global min cut (S, S') of G . Let δ be edges with one endpoint in S and the other in S' .
Let $k = |\delta|$ = size of the min cut.




Amplification

To amplify the probability of success, run the contraction algorithm many times.

Claim: If we repeat the contraction algorithm $r \binom{n}{2}$ times with independent random choices, the probability that all runs fail is at most

$$\left(1 - \frac{1}{\binom{n}{2}}\right)^{r \binom{n}{2}} \leq (1/e)^r$$



$$\left(1 - \frac{1}{x}\right)^x \leq 1/e$$

constant



Set $r = (c \ln n)$ then probability of failure is: $e^{-c \ln n} = n^{-c}$

and probability of success is: $1 - 1/n^c$

Karger's Contraction Algorithm

Algorithm:

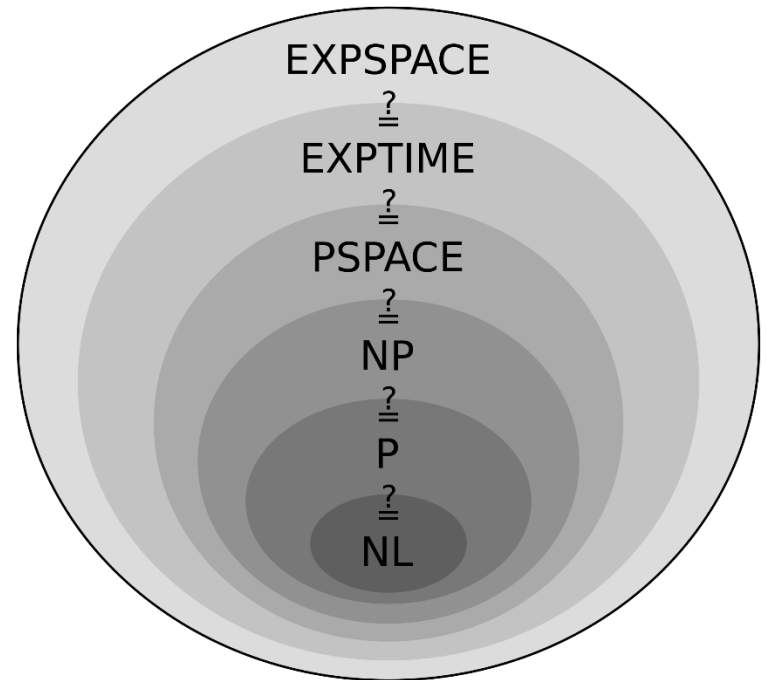
1. Start with the input graph $G=(V,E)$.
2. While $|V| > 2$ do
Contract an arbitrary edge (u,v)
3. Return the cut S (only one possible cut).

Running time: $n-2$ iterations.
each iteration requires $O(n)$ time
 $\Rightarrow O(n^2)$

The algorithm is iterated $O(n^2 \log n)$ times...total running time $O(n^4 \log n)$.

Lecture 10 (Adv)

PSPACE: A Class of Problems Beyond NP

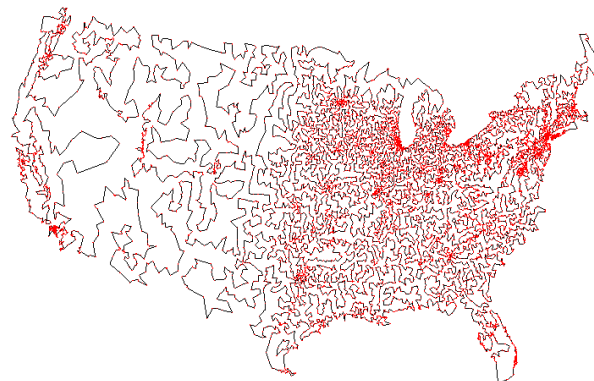


PSPACE

P: Decision problems solvable in polynomial **time**.

PSPACE: Decision problems solvable in polynomial **space**.

Example: Euclidean TSP \in PSPACE
(largest solved instance: 85,900)



Observation: $P \subseteq \text{PSPACE}$.



Since poly-time algorithm can
consume only polynomial space

PSPACE

Note that there are algorithms that might need exponential time but only polynomial space:

binary counter. Count from 0 to $2^n - 1$ in binary

Space: $O(\log_2 2^n) \Rightarrow O(n)$ space)

Theorem: 3-SAT is in PSPACE.

Proof:

- Enumerate all 2^n possible truth assignments using counter.
- Check each assignment to see if it satisfies all clauses. ▀

Important corollary: $NP \subseteq PSPACE$.

Proof: Consider arbitrary problem Y in NP.

- Since $Y \leq_p 3\text{-SAT}$, there exists algorithm that solves Y in poly-time plus polynomial number of calls to 3-SAT black box.
- Can implement black box in poly-space. ▀

Two types of problems

- Quantified SAT problems
- Planning problems

Quantified Satisfiability (QSAT)

QSAT: Let $\Phi(x_1, \dots, x_n)$ be a Boolean CNF formula. Is the following propositional formula true?

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \dots \forall x_{n-1} \exists x_n \Phi(x_1, \dots, x_n)$$

↑
assume n is odd

QSAT: $\exists x_1 \forall x_2 \exists x_3 \forall x_4 \dots \forall x_{n-1} \exists x_n \Phi(x_1, \dots, x_n)$

SAT: $\exists x_1 \exists x_2 \exists x_3 \exists x_4 \dots \exists x_{n-1} \exists x_n \Phi(x_1, \dots, x_n)$

PSPACE-Complete

PSPACE: Decision problems solvable in polynomial space.

PSPACE-complete: Problem Y is PSPACE-complete if

- (i) Y is in PSPACE and
- (ii) for every problem X in PSPACE, $X \leq_p Y$.

Theorem: QSAT is PSPACE-complete. (without proof)

Theorem: $\text{PSPACE} \subseteq \text{EXPTIME}$.

Proof: Previous algorithm solves QSAT in exponential time, and QSAT is PSPACE-complete. ▀

Summary. $P \subseteq NP \subseteq \text{PSPACE} \subseteq \text{EXPTIME}$.

↑ ↑ ↑

it is known that $P \neq \text{EXPTIME}$, but unknown which inclusion is strict;
conjectured that all are

Summary

PSPACE: Decision problems solvable in polynomial space.

$$P \subseteq NP \subseteq PSPACE$$

PSPACE-complete: Problem Y is PSPACE-complete if

- (i) Y is in PSPACE and
- (ii) for every problem X in PSPACE, $X \leq_p Y$.

Theorem: QSAT is PSPACE-complete.

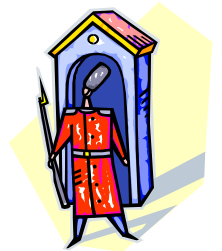
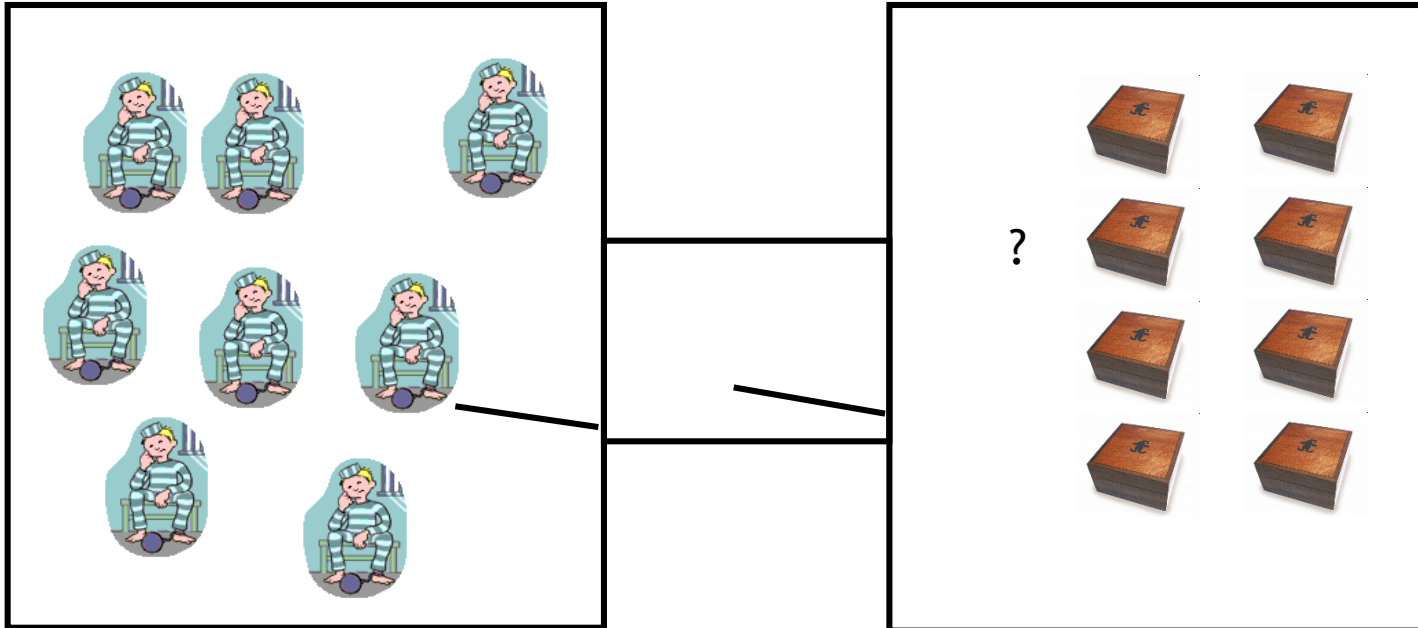
$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$$

Names in Boxes

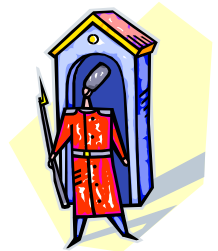
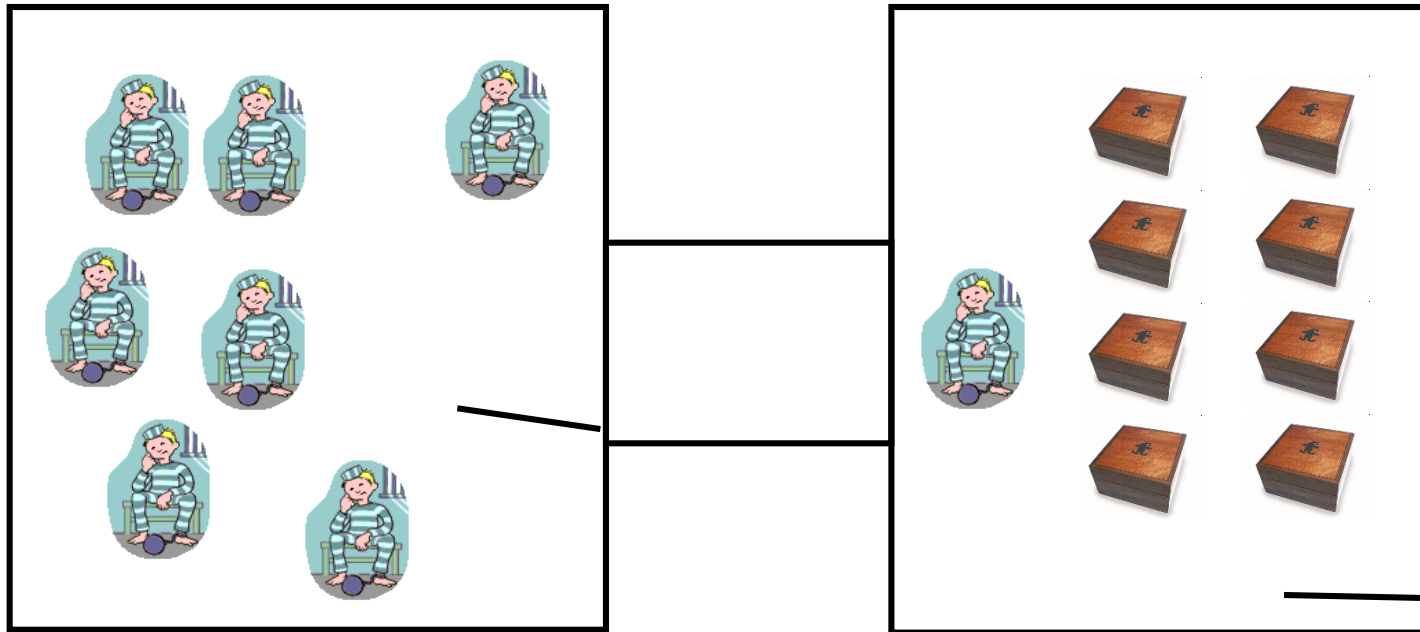
The names of 100 prisoners are placed in 100 wooden boxes, one name in each box, and the boxes are lined up on a table in a room. One by one, the prisoners are led into the room; each may look in at most 50 boxes, but must leave the room exactly as he found it and is permitted no further communication with the others.



Names in Boxes



Names in Boxes



The prisoners have a chance to plot their strategy in advance, but unless every single prisoner finds his own name all prisoners will be executed.

Names in Boxes

Any good strategies?

Choose 50 boxes randomly?

2 prisoners \rightarrow 2 boxes \rightarrow $\frac{1}{4}$ chance of surviving

100 prisoners \rightarrow 100 boxes \rightarrow $(\frac{1}{2})^{100} \approx 10^{-31}$ chance of surviving

Names in Boxes

Can we do better?

Yes we can develop a strategy that gives more than 30% probability of success! Is that possible?

Names in Boxes

Assume every prisoner is a unique integer in $[1 \dots 100]$.
All the boxes are ordered from left to right:



All the prisoners use the same scheme.

1. The prisoner opens the box corresponding to his number.
2. Then he opens the box belonging to the number he found in the box he just opened.
3. Continue step 2 until he opened 50 boxes or until he found his own number.

Names in Boxes

1	2	3	4	5	6	7	8
6	1	3	2	5	4	8	7



Prisoner: 1

Names in Boxes

1	2	3	4	5	6	7	8
6	1	3	2	5	4	8	7



Prisoner: 2

Names in Boxes

1	2	3	4	5	6	7	8
6	1	3	2	5	4	8	7



Prisoner: 3

Names in Boxes

1	2	3	4	5	6	7	8
6	1	3	2	5	4	8	7



Prisoner: 4

Names in Boxes

1	2	3	4	5	6	7	8
6	1	3	2	5	4	8	7



Prisoner: 5

Names in Boxes

1	2	3	4	5	6	7	8
6	1	3	2	5	4	8	7



Prisoner: 6

Names in Boxes

1	2	3	4	5	6	7	8
6	1	3	2	5	4	8	7



Prisoner: 7

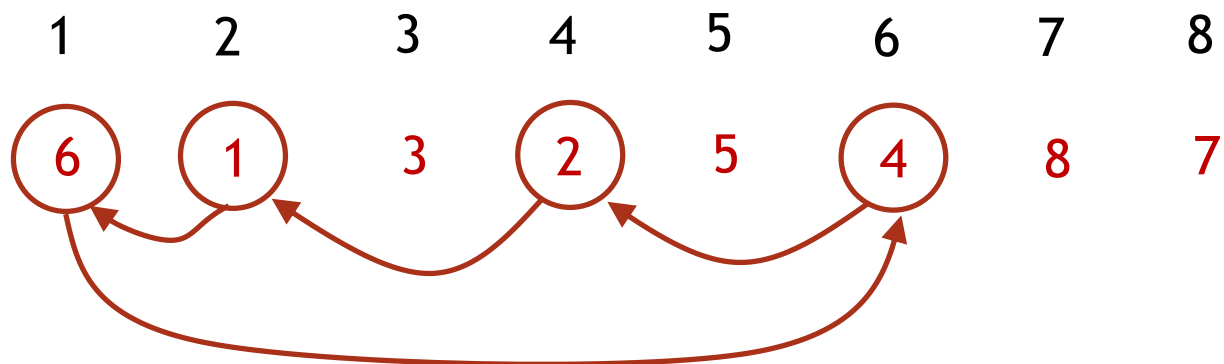
Names in Boxes

1	2	3	4	5	6	7	8
6	1	3	2	5	4	8	7



Prisoner: 8

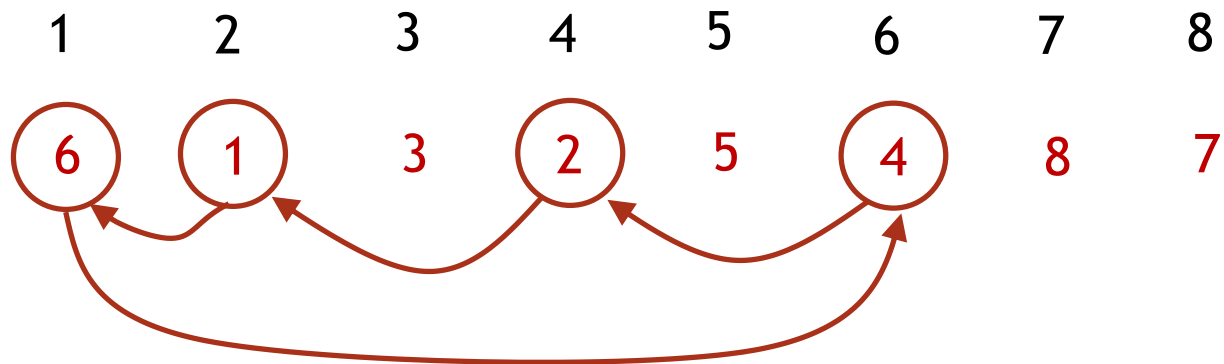
Names in Boxes



Prisoner: 2

Observation 1: Each prisoner is following a cycle in the permutation beginning with his own number.

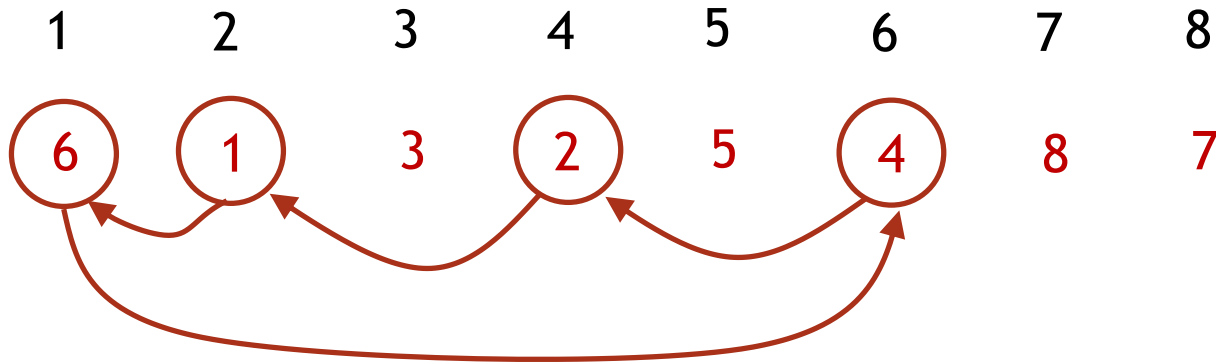
Names in Boxes



Observation 2: If the length of the cycle is at most 50 then he will find his name.

Prisoner: 2

Names in Boxes



Prisoner: 2

Observation 3: If the permutation does not have any cycle that is longer than 50 then all prisoners will find their names and be spared.

Names in Boxes

What's the probability that there is no cycle of length > 50 ?

Consider $2n$ prisoners and n boxes being opened.

$$P_k = \Pr[\exists \text{ cycle of length } k]$$

$$P_k = (\text{\#instances containing a cycle of length } k) / (\text{total \#instances})$$

Names in Boxes

$$P_k = \underbrace{(\text{\#instances containing a cycle of length } k)}_{\text{(a) \#subsets with } k \text{ elements} \bullet} / \underbrace{(\text{total \#instances})}_{\text{(b) \#permutations of } k \text{ elements forming a cycle} \bullet}$$

(2n)!

- (a) #subsets with k elements •
 (b) #permutations of k elements forming a cycle •
 (c) #permutations of the remaining elements

(a) #subsets with k elements

$$\binom{2n}{k} = \frac{(2n)!}{k! (2n-k)!}$$

	1 2 3 4	1 2 3 4
1 2 3	2 _ _ _	2 3 4 1
2 3 1	3 _ _ _	2 4 1 3
3 1 2	4 _ _ _	3 1 4 2
		3 4 2 1
		4 3 1 2
		4 1 2 3

(b) #permutations of k elements forming a cycle = $M(k) = (k-1)!$

Names in Boxes

$$P_k = \underbrace{(\text{\#instances containing a cycle of length } k)}_{\substack{(a) \text{ \#subsets with } k \text{ elements} \bullet \\ (b) \text{ \#permutations of } k \text{ elements forming a cycle} \bullet \\ (c) \text{ \#permutations of the remaining elements}}} / \underbrace{(\text{total \#instances})}_{(2n)!}$$

- (a) #subsets with k elements •
- (b) #permutations of k elements forming a cycle •
- (c) #permutations of the remaining elements

(a) #subsets with k elements

$$\binom{2n}{k} = \frac{(2n)!}{k! (2n-k)!}$$

(b) #permutations of k elements forming a cycle = $M(k) = (k-1)!$

(c) #permutations of the remaining $(2n-k)$ elements = $(2n-k)!$

Names in Boxes

$$P_k = \underbrace{(\text{\#instances containing a cycle of length } k)}_{\substack{(a) \text{ \#subsets with } k \text{ elements} \bullet \\ (b) \text{ \#permutations of } k \text{ elements forming a cycle} \bullet \\ (c) \text{ \#permutations of the remaining elements}}} / \underbrace{(\text{total \#instances})}_{(2n)!}$$

- (a) #subsets with k elements •
- (b) #permutations of k elements forming a cycle •
- (c) #permutations of the remaining elements

(a) #subsets with k elements

$$\binom{2n}{k} = \frac{(2n)!}{k! (2n-k)!}$$

(b) #permutations of k elements forming a cycle = $M(k) = (k-1)!$

(c) #permutations of the remaining $(2n-k)$ elements = $(2n-k)!$

$$\Pr[\exists \text{ cycle of length } k] = \frac{(2n)!}{k! (2n-k)!} \times (k-1)! \times (2n-k)! / (2n)! = 1/k$$

Names in Boxes

$$\Pr[\exists \text{ cycle of length } k] = \frac{(2n)!}{k! (2n-k)!} \times (k-1)! \times (2n-k)! / (2n)! = 1/k$$

[2n prisoners] Since there can be at most one cycle of length $k > n$ in a permutation we get that the probability that there is a cycle of length $> n$ is:

$$\begin{aligned} \Pr[\exists \text{ cycle of length } > n] &= \sum_{k=n+1}^{2n} P_k \\ &= 1/(n+1) + 1/(n+2) + \dots + 1/2n = H_{2n} - H_n < \ln 2n - \ln n = \ln 2 \end{aligned}$$

$$1 - \ln 2 > 0.307 \Rightarrow 30.7\% \text{ chance of surviving!}$$

More algorithms?

- COMP3530: Discrete Optimization
Lecturer: Julian Mestre
- COMP5045: Computational Geometry
Lecturer: Joachim

Please remember to fill in the unit of study evaluation

<https://student-surveys.sydney.edu.au/students/complete/>

What was good? What was bad?

Thanks for taking the class!

Good luck on the exam!