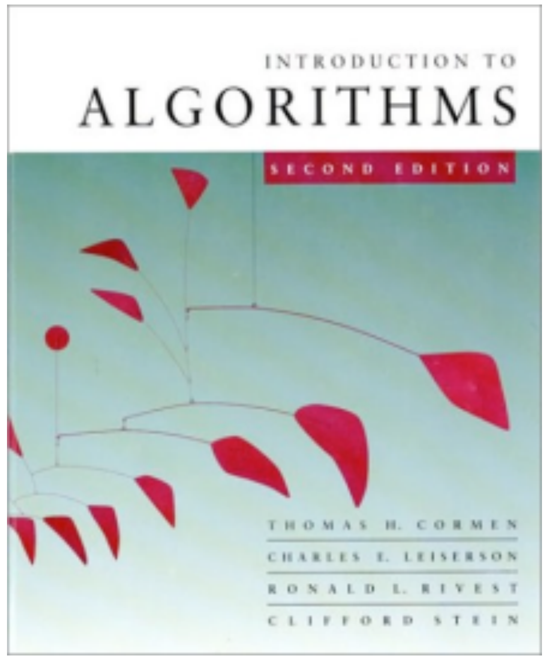
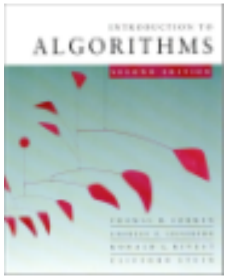
Introduction to Algorithms

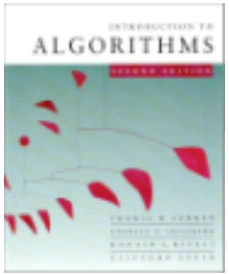


Prof. Piotr Indyk



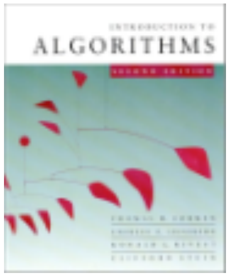
P vs NP (interconnectedness of all things)

- A whole course by itself



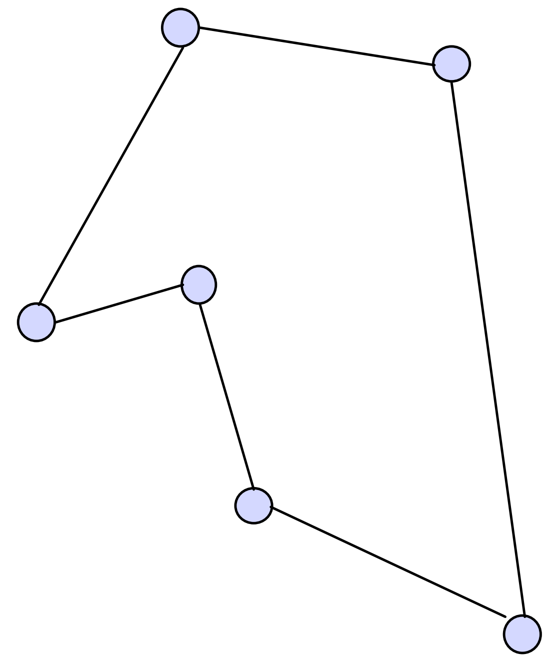
Have seen so far

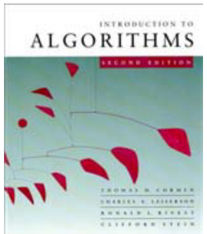
- Algorithms for many interesting problems:
  - Running times $O(nm^2)$, $O(n^2)$, $O(n \log n)$, $O(n)$, …
  - I.e., polynomial in the input size
- Can we solve all (or most of) interesting problems in polynomial time ?
- Not really…



Example difficult problem

- Traveling Salesperson Problem (TSP)
  - Input: undirected graph with lengths on edges
  - Output: shortest tour that visits each vertex exactly once
- Best known algorithm: $O(n\ 2^n)$ time.



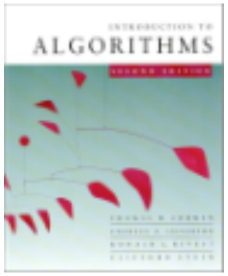


Set Covering

- Set Cover:
  - Input: subsets $S_1 \ldots S_n$ of X, $\cup_i S_i = X$, $|X|=m$
  - Output: $C \subseteq \{1 \ldots n\}$, such that $\cup_{i \in C} S_i = X$, and $|C|$ minimal
- Best known algorithm: $O(2^n\ m)$ time(?)

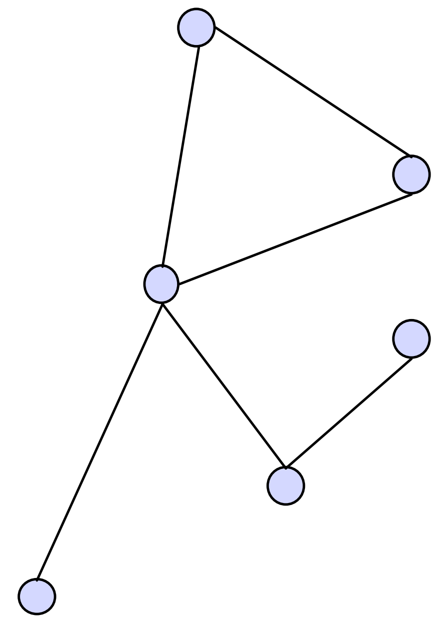
Bank robbery problem:

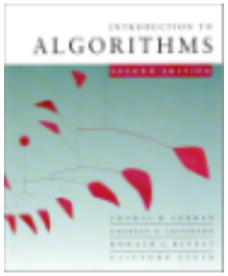
- X={plan, shoot, safe, drive, scary}
- Sets:
  - S_{Joe}={plan, safe}
  - S_{Jim}={shoot, scary, drive}
  - ….



Another difficult problem

- Clique:
  - Input: undirected graph $G=(V,E)$
  - Output: largest subset C of V such that every pair of vertices in C has an edge between them
- Best known algorithm: $O(n\ 2^n)$ time





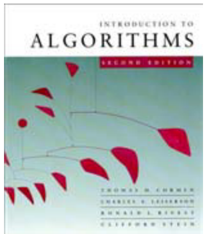
Another difficult problem

Boolean Formula Satifiability Problem (SAT):

Given a Boolean formula F(X1, X2, ..., Xn)
with n Boolean variables X1, X2, ..., Xn.

SAT Problem: Determine if there is an trueth assignment
of the n Boolean variables to 0(false) or 1(true)
that makes the formula F =1(true).

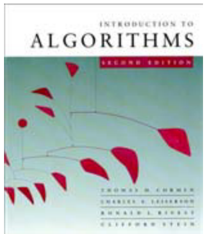
Example F = (X1 V ¬X2 V X3) ∧ (X2 V ¬X3 V ¬X5)



Dealing with Hard Problems

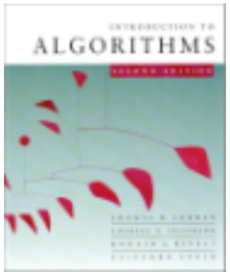
- What to do if:
  - Divide and conquer
  - Dynamic programming
  - Greedy
  - Linear Programming/Network Flows
  - …

  does not give a polynomial time algorithm?



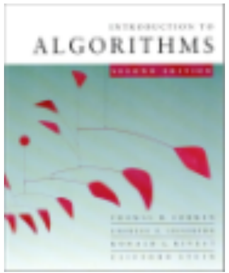
Dealing with Hard Problems

- Exponential time algorithms for small inputs. E.g., $(100/99)^n$ time is not bad for $n < 1000$.
- Polynomial time algorithms for some (e.g., average-case) inputs
- Polynomial time algorithms for all inputs, but which return approximate solutions



What can we do ?

- Spend more time and money designing efficient algorithms for those problems
  - People tried for a few decades, no luck
  - Outstanding $1000,000 prize for finding one
  - It seems very likely that such algorithms do not exist
- Prove there is no polynomial time algorithm for those problems
  - Would be great
  - Seems *really* difficult
  - Best lower bounds for "natural" problems:
    - $\Omega(n^2)$ for restricted computational models
    - $\Omega(n)$ for unrestricted computational models

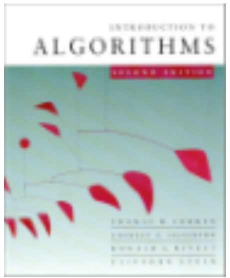


What else can we do ?

- Show that those hard problems are essentially equivalent.

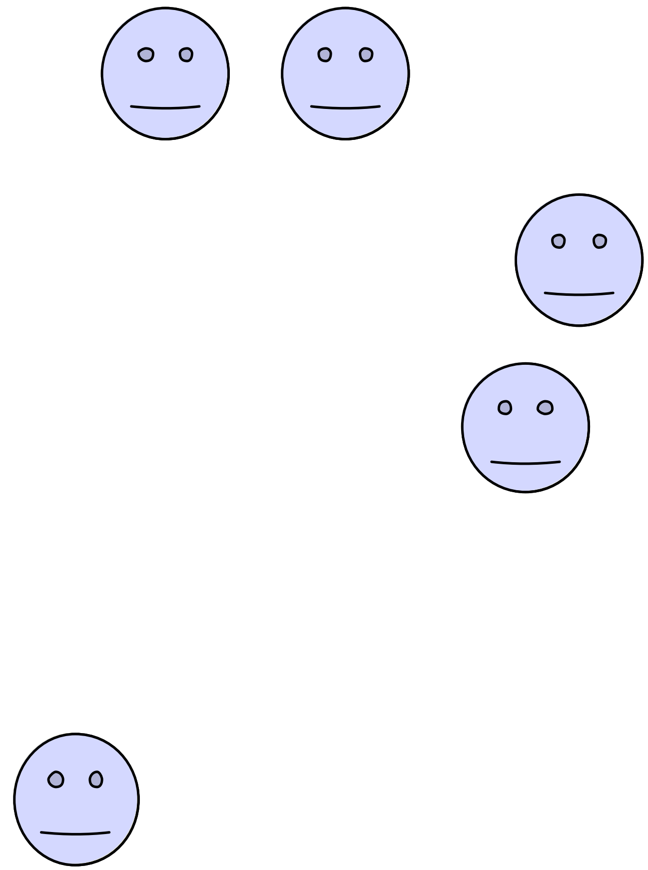
  I.e., if we can solve one of them in poly time, then all others can be solved in poly time as well.

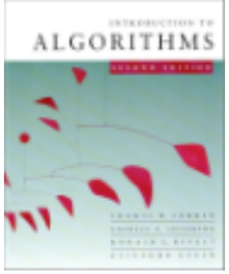
- Works for at least few thousand hard problems



The benefits of equivalence

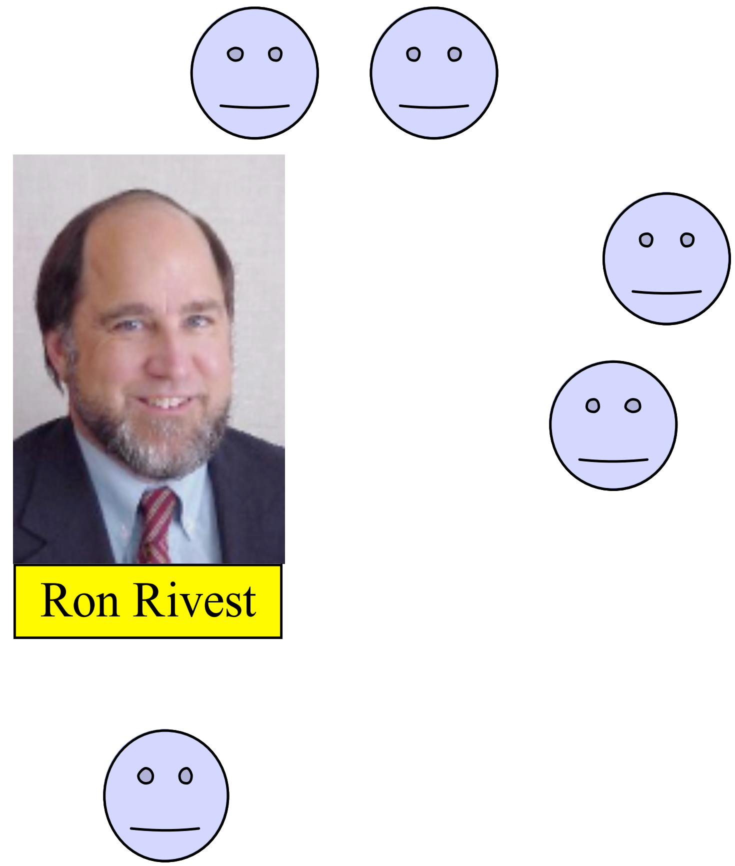
- Combines research efforts
- If one problem has polytime solution, then all of them do

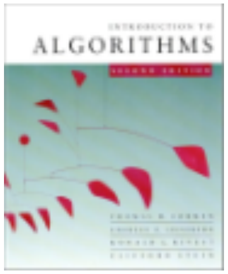




A more realistic scenario

- Once an exponential lower bound is shown for one problem, it holds for all of them
- But someone *is* happy…

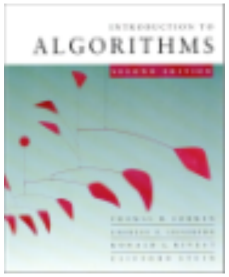




Summing up

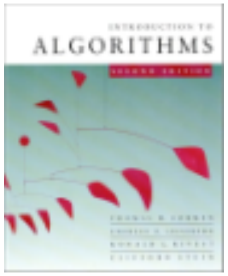
- If we show that a problem Π is equivalent to a few thousand other well studied problems without efficient algorithms, then we get a very strong evidence that Π is hard.
- We need to:

1. Identify the class of problems of interest
2. Define the notion of equivalence
3. Prove the equivalence(s)



1. Class of problems (informally)

- Decision problems: answer YES or NO. E.g.,"is there a tour of length $\leq$ K" ?
- Solvable in *non-deterministic polynomial* time:
  - Intuitively: the solution can be verified in polynomial time
  - E.g., if someone gives as a tour T, we can verify if T is a tour of length $\leq$ K.
- Therefore, TSP is in NP.



1. Class of problems: NP

Deterministic Time (P):

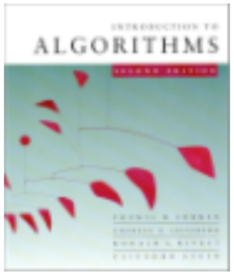
- A problem $\prod$ is solvable in poly time (or $\prod \in P$), if there is a poly time algorithm $V(.)$ such that for any input x:

$$\prod(x)=YES \text{ iff } V(x)=YES$$

Nondeterministic Time (NP):

- A problem $\prod$ is solvable in non-deterministic poly time (or $\prod \in NP$), if there is a poly time algorithm $V(.\,,.)$ such that for any input x:

$$\prod(x)=YES \text{ iff there exists a certificate } y \text{ of size } poly(|x|) \text{ such that } V(x,y)=YES$$



Examples of problems in NP

- Is "Does there exist a clique in G of size ≥K" in NP ?

  Yes: V(x,y) interprets x as a graph G, y as a set C, and checks if all vertices in C are adjacent and if |C|≥K

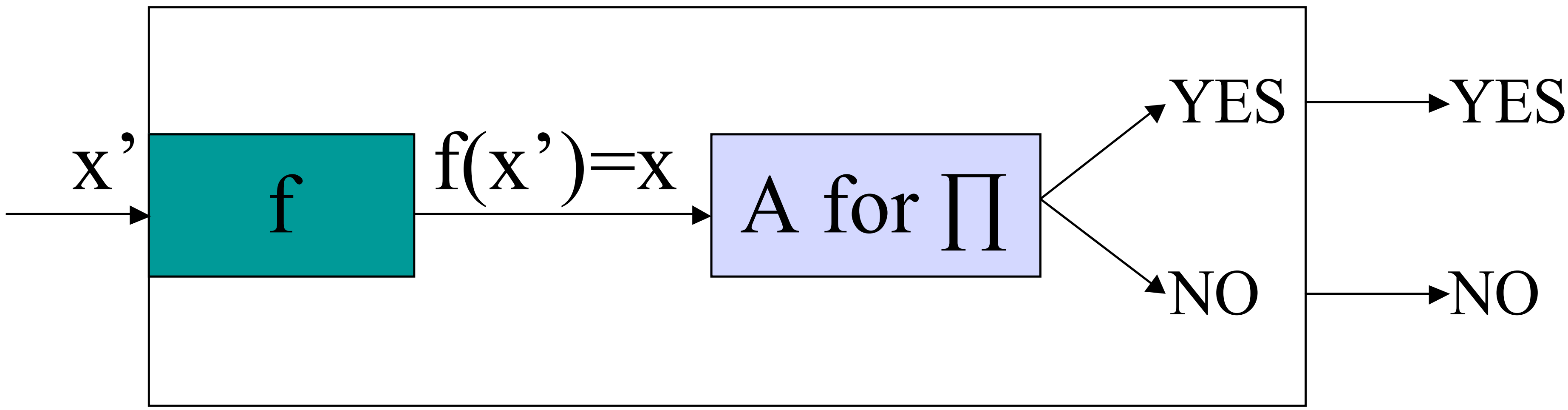
- Is Sorting in NP ?

  No, not a decision problem.

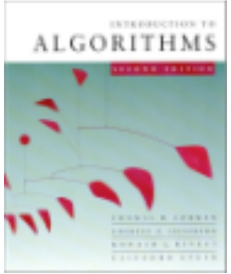
- Is "Sortedness" in NP ?

  Yes: ignore y, and check if the input x is sorted.

2. Reductions: ∏' to ∏



A' for ∏'



2. Reductions (formally)

Polynomial Time Reductions between Problem Classes:

- Π' is poly time reducible to Π ($\Pi' \leq \Pi$) iff there is a poly time function f that maps inputs x' to Π' into inputs x of Π, such that for any x'

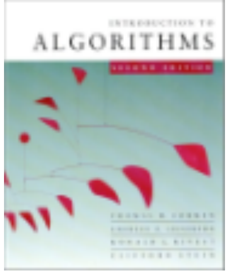
$$\Pi'(x')=\Pi(f(x'))$$

- Fact 1: if $\Pi \in P$ and $\Pi' \leq \Pi$ then $\Pi' \in P$
- Fact 2: if $\Pi \in NP$ and $\Pi' \leq \Pi$ then $\Pi' \in NP$
- Fact 3: if $\Pi' \leq \Pi$ and $\Pi'' \leq \Pi'$ then $\Pi'' \leq \Pi$

Summing up

Two Problems are Polynomial Time Equivalent:
if there Polynomial Time Reductions between the two problems

- If we show that a problem $\prod$ is equivalent to a few thousand other well studied problems without efficient algorithms, then we get a very strong evidence that $\prod$ is hard.



- Once an exponential lower bound is shown for one problem, it holds for all of them
- But someone *is* happy…