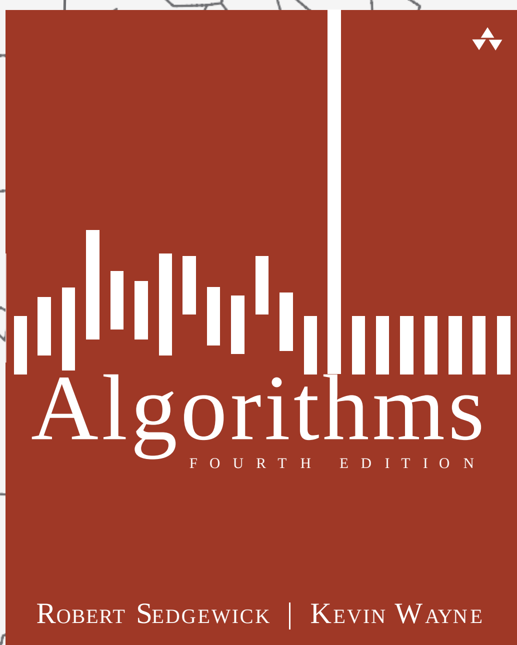


http://algs4.cs.princeton.edu

Reductions and Beyond

- Intro: P
- NP completeness
- Reductions
- Dealing with NP (or harder) problems
- P=NP
- Beyond complexity classes

Overview: Introduction to the topics

„P = NP“.

- The biggest question in your field, computer science.
- A million dollar question.
- Maybe the biggest question in all of science.

After this lecture, you should be able to explain:

- What the „P = NP“ question is.
- Why it‘s such a big deal
- Why it‘s useful for programmers to know about NP-complete problems

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

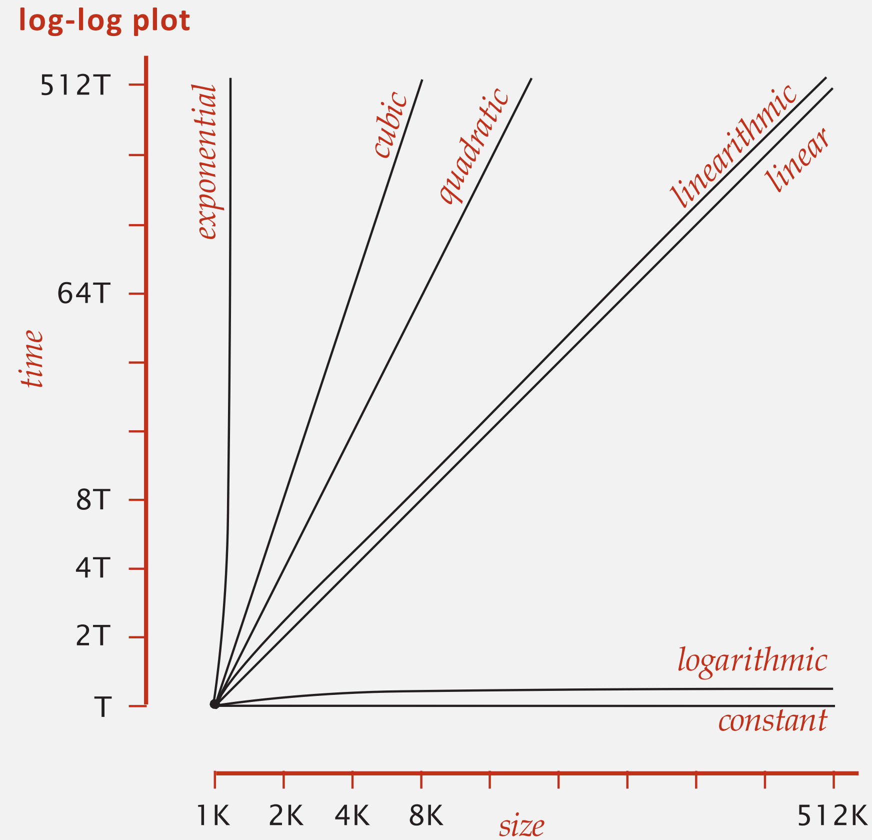
Reductions and Beyond

- Intro: P
- Reductions
- NP completeness
- Dealing with NP (or harder) problems
- P=NP
- Beyond complexity classes

Overview: introduction to advanced topics

Main topics.

- Most of our problems so far have been easy.
  - Sorting, symbol table operations (array, BST, hash table), graph search, MSTs, SPTs, etc.
- Some have been hard.
  - Hamilton path.
  - Satisfiability



Typical orders of growth

P

some constant

Polynomial Time Solvability

- A problem is in **P** if there is an algorithm that solves it in $O(N^k)$ time.
  - Worst case order of growth is $\leq N^k$.
  - N is number of bits needed to specify input.

	Order of Growth	Input bits	
Finding Maximum	Q	Q $\propto$ N	$O(N)$
Sorting with compareTo	Q log Q	Q $\propto$ N	$O(N^2)$
DFS and BFS	E + V	V, E $\propto$ N	$O(N)$

Why $O(N^k)$?

- P seems rather generous.
- $O(N^k)$ closed under addition and multiplication.
  - Consecutively run two algorithms in **P**, still in **P**.
  - Run an algorithm N times, still in **P.**
- Exponents for practical problems are typically small.

A modern standard for simplicity

Most important point

- If a practical problem is easy, it is in **P**.
- If a practical problem is in **P**, it is easy.

Def. A problem is intractable if it can't be solved in polynomial time.

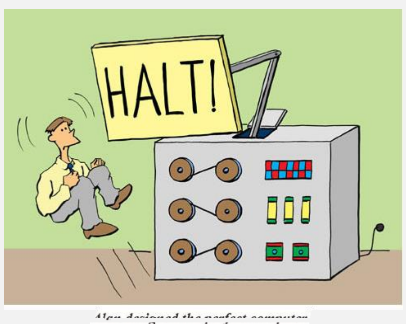
Desiderata. Prove that a problem is intractable.

Two problems that provably require exponential time.

input size = c + lg K

- Given a (constant-size) program, does it halt in at most K steps?
- Given N-by-N checkers board position, can the first player force a win?

using forced capture rule



Frustrating news. Very few successes.

Satisfiability is conjectured to be intractable

Q. How to solve an instance of *3-SAT* with n variables?

A. Exhaustive search: try all 2^n truth assignments.

Q. Can we do anything substantially more clever?

(An algorithm with significantly better worst-case time complexity)

Conjecture (P $\neq$ NP). *3-SAT* is intractable (no poly-time algorithm).

consensus opinion

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

Reductions and Beyond

- Intro
- NP and NP-Completeness
- Reductions
- Dealing with NP (or harder) problems
- P=NP

NP

The Class NP

- Decision problem.
- If answer is “Yes”, a proof exists that can be verified in polynomial time.

- Stands for “non-deterministic polynomial”
  - Name is a confusing relic. Don’t worry about it.

- **Most important detail: Verifiable in Polynomial Time.**
  - “In an ideal world it would be renamed P vs VP” - Clyde Kruskal

Verification problems

Verification problem. Problem where you can check a solution in poly-time.

Ex 1. 3-SAT.

$$\begin{array}{ccccccccc} \neg x_1 & or & x_2 & or & x_3 & & & = & true \\ x_1 & or & \neg x_2 & or & x_3 & & & = & true \\ \neg x_1 & or & \neg x_2 & or & \neg x_3 & & & = & true \\ \neg x_1 & or & \neg x_2 & or & & or & x_4 & = & true \\ & & \neg x_2 & or & x_3 & or & x_4 & = & true \end{array}$$

instance I

x_1	x_2	x_3	x_4
T	T	F	T

solution S

Ex 2. FACTOR Given an N-bit integer x, find a nontrivial factor.
.

147573952589676412927   193707721

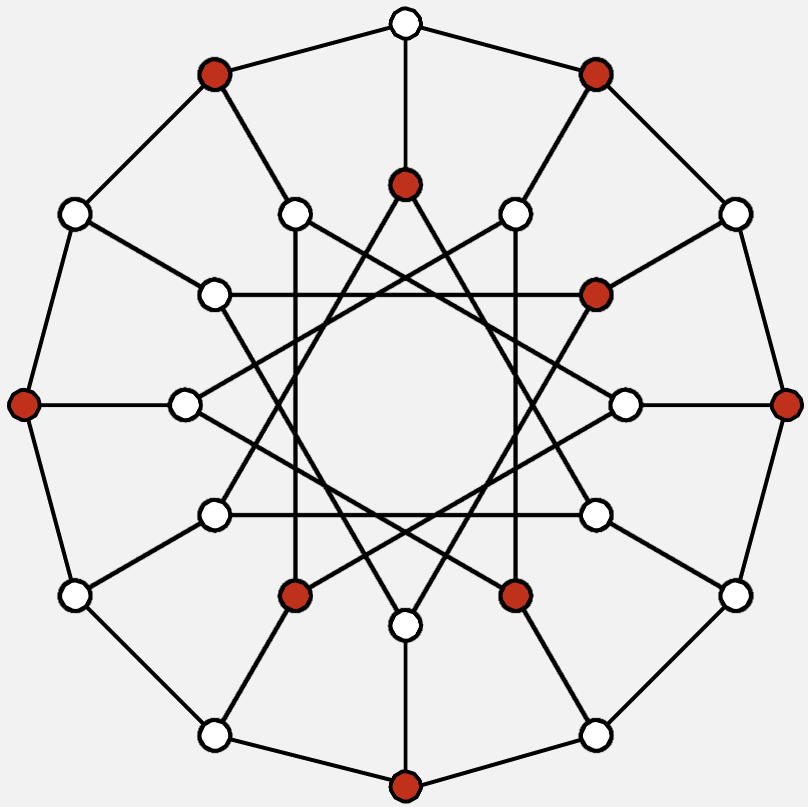
instance I   **solution S**

Independent set

An independent set is a set of vertices, no two of which are adjacent.

IND-SET. Given graph G and an integer k, find an independent set of size k.

k = 9



Applications. Scheduling, computer vision, clustering, ...

Partition problem

Want to split a set into two equal parts.

PARTITION. Partition a given set S of reals into two parts, S_1 and S_2 with the same total weight:

$$\sum_{w_i \in S_1} w_i = \sum_{w_j \in S_2} w_j$$

Example 1. $S = \{1.1, 2.3, 2.7, 3.5, 7.4\}$ Answer:

Example 2. $S = \{1.3, 2.3, 2.7, 3.5\}$ Answer:

Applications. Load balancing, dividing work among workers, ...

NP

A vast number of interesting well-defined problems are in NP.

- Hand-wavy reason: In NP if you can ask useful decision sub-problems about a solution.

3-SAT. CNF-satisfiability, where each clause has 3 literals.

Independent set. Is there a subset of *k* people none of which know each other?

Hamilton path. Is there a spanning cycle in a graph using no edge twice?

Graph Coloring. Can a given graph be colored with 3 colors?

Traveling Salesperson. Can you visit all the given towns by driving < X km?

Partition. Do these files fit on two storage devices?

Knapsack. What is the most I can rob, if I can only carry 10 kg?

Integer Linear Programming. Find an optimal ILP solution.

Many practical problems are unsolvable using the tools of this course.

- Most of these problems are actually SAT in disguise (!!).

- Learning to recognize SAT equivalent problems (NP Complete).
  - Need some rigorous notion of equivalent difficulty.

Quiz 2: NP

Which of these problems is **not** in NP?

A. Satisfiability

B. „Is array X sorted?“

C. Minimum Spanning Tree

D. „Is graph G strongly connected“?

E. „Does this program always terminate?“

Algorithms

Robert Sedgewick | Kevin Wayne

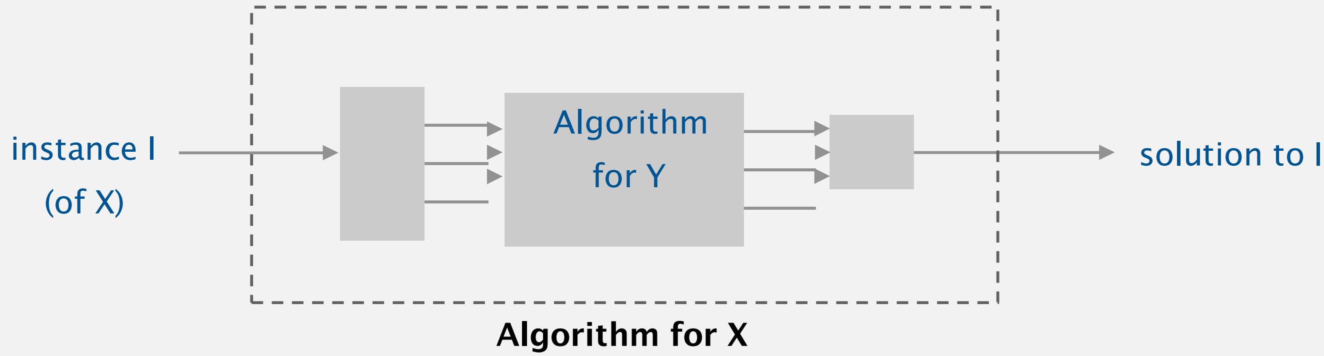
http://algs4.cs.princeton.edu

Seorting and Reductions

- Reductions

Reductions

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X.



Cost of solving X = total cost of solving Y + cost of reduction.

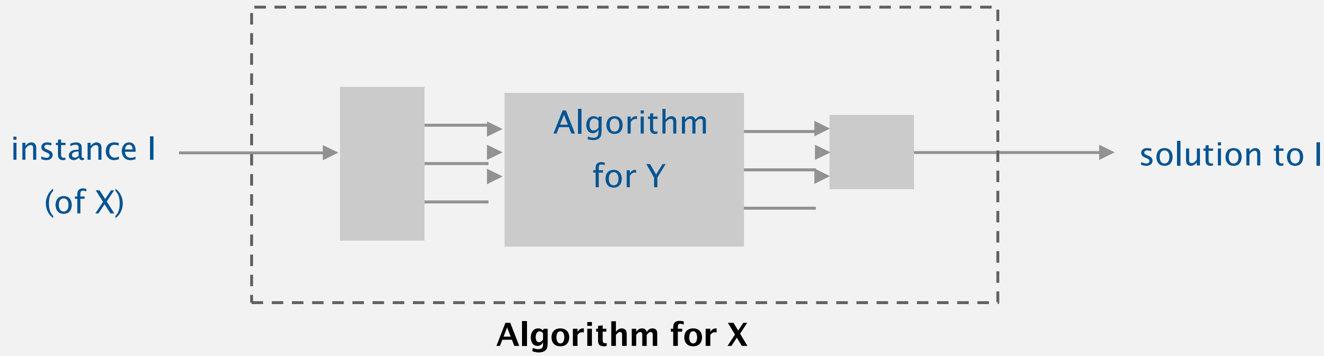
perhaps many calls to Y on problems of different sizes (though, typically only one call)

preprocessing and postprocessing (typically less than cost of solving Y)

X is no harder than Y (same or lesser difficulty).

Reductions

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X.



Ex 1. [finding the median reduces to sorting]

To find the median of N items:

- Sort N items.
- Return item in the middle.

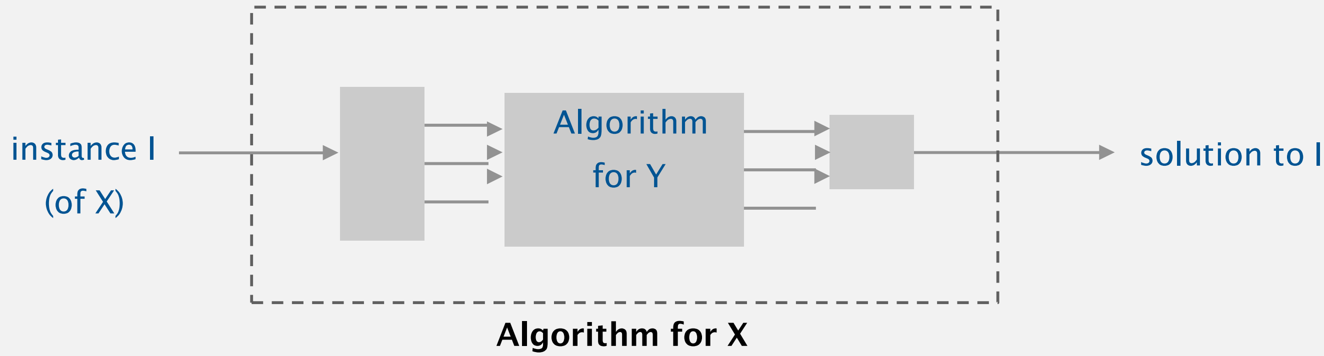
cost of sorting

cost of reduction

Cost of solving finding the median. $N \log N + 1$.

Reductions

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X.



Ex 2. [element distinctness reduces to sorting]

To solve element distinctness on N items:

- Sort N items.
- Check adjacent pairs for equality.

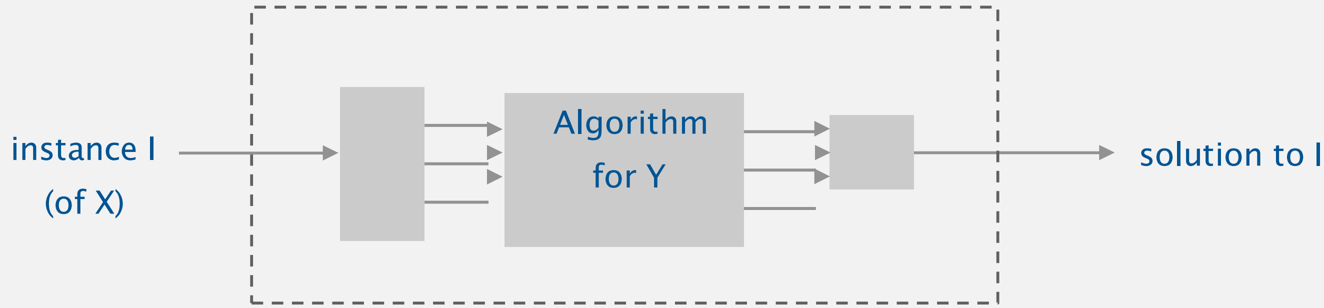
cost of sorting

cost of reduction

Cost of solving element distinctness. $N \log N + N$.

Reductions

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X.



Algorithm for X

Ex 3. [3-collinear reduces to sorting]
To solve 3-collinear instance on N points in the plane:
For each point, sort other points by polar angle.
check adjacent triples for collinearity

cost of sorting

cost of reduction

Cost of solving 3-collinear. $N^2 \log N + N^2$.

Algorithms

Robert Sedgewick | Kevin Wayne

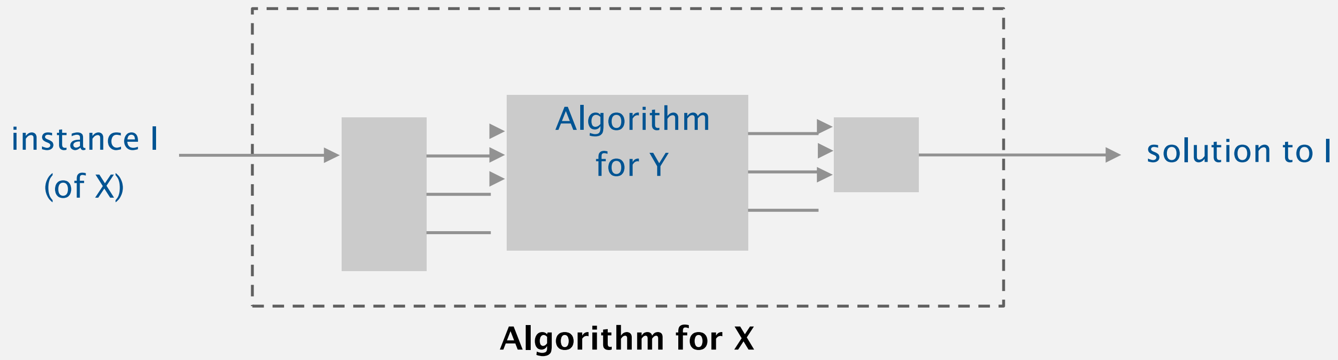
http://algs4.cs.princeton.edu

Reductions and Beyond

- Intro
- Reductions
- NP Completeness
- Dealing with NP (or harder) problems
- P=NP
- Beyond complexity classes

Problem X poly-time (Cook) reduces to problem Y if X can be solved with:

- Polynomial number of standard computational steps.
- Polynomial number of calls to Y.



Establish intractability. If *3-SAT* poly-time reduces to Y, then Y is intractable. (assuming *3-SAT* is intractable)

Mentality.

- If I could solve Y in poly-time, then I could also solve *3-SAT* in poly-time.
- *3-SAT* is believed to be intractable.
- Therefore, so is Y.

NP-complete

NP-complete

The hardest problems in NP

- A problem π is NP-complete if:
  - π is in NP.
  - All problems in NP (poly-time) reduce to π.

- Solution to an NP-complete problem would be a key to the universe!

Two questions

- Are there any NP-complete problems?
- Do we know how to solve any of them?

Existence of an NP complete problem

3SAT

Also in NP!

- Cook (71) and Levin (73) proved that every NP problem reduces to 3SAT.
  - 3SAT is at least as hard as every other problem in NP.
  - A solution to 3SAT provides a solution to every problem in NP.

- 3-SAT: CNF Satisfiability, where each clause has 3 literals

Stephen Cook

Leonid Levin

Existence of an NP complete problem

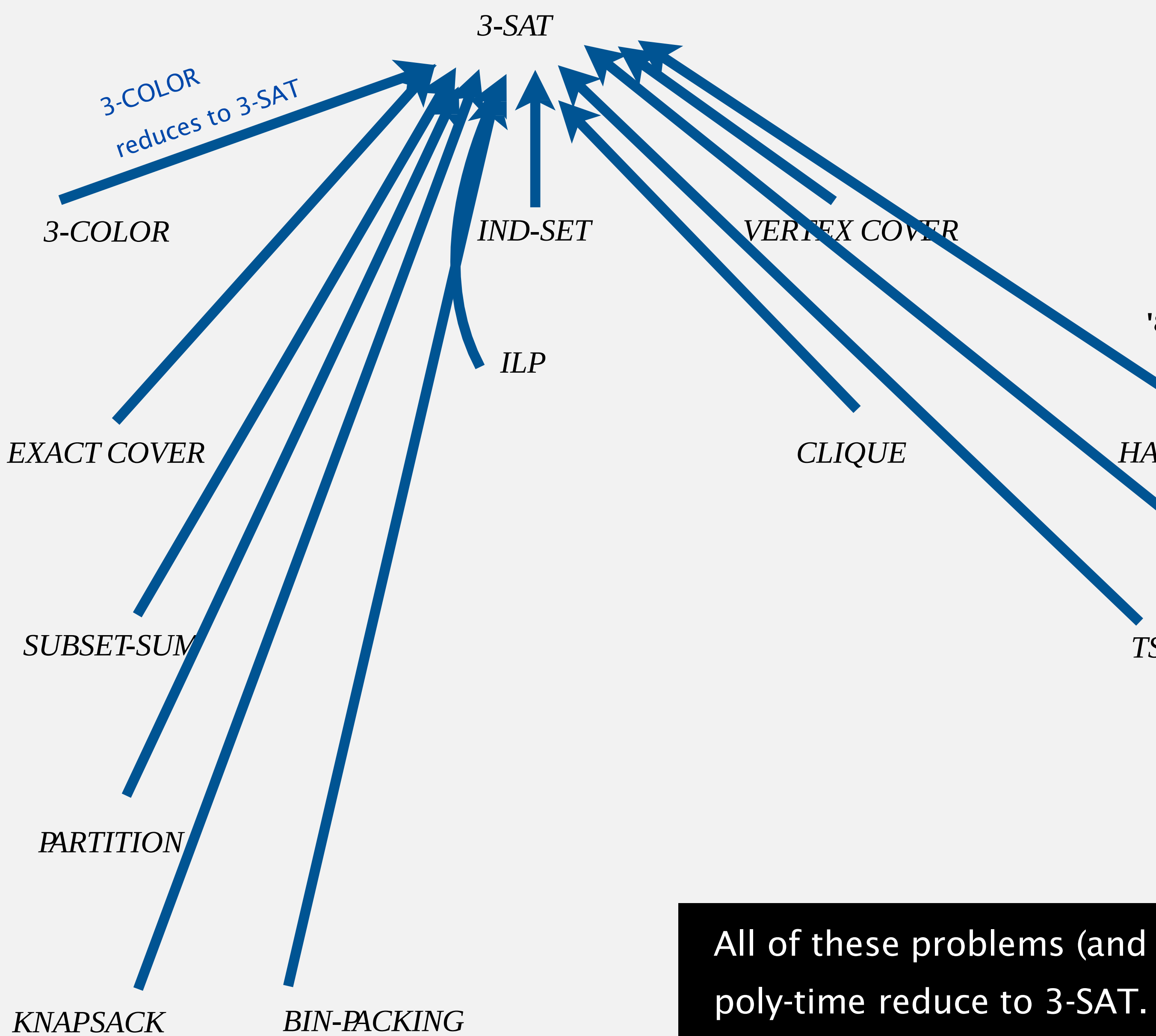
Rough idea of Cook-Levin theorem

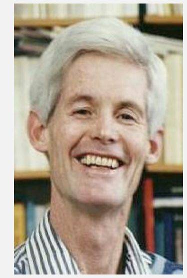
- Create giant (!!) boolean logic expression that represents entire state of your computer at every time step.
- If solution takes polynomial time, boolean logic circuit is polynomial in size.
- Example boolean logic variable: True if 57173th bit of memory is true and we're on line 38 of code during cycle 7591872 of execution.

Stephen Cook

Leonid Levin

Implications of Cook-Levin theorem





Stephen Cook
'82 Turing award

Leonid Levin

All of these problems (and many, many more) poly-time reduce to 3-SAT.

3SAT

Great, 3SAT solves most well defined problems of general interest!

Can we solve 3SAT efficiently?

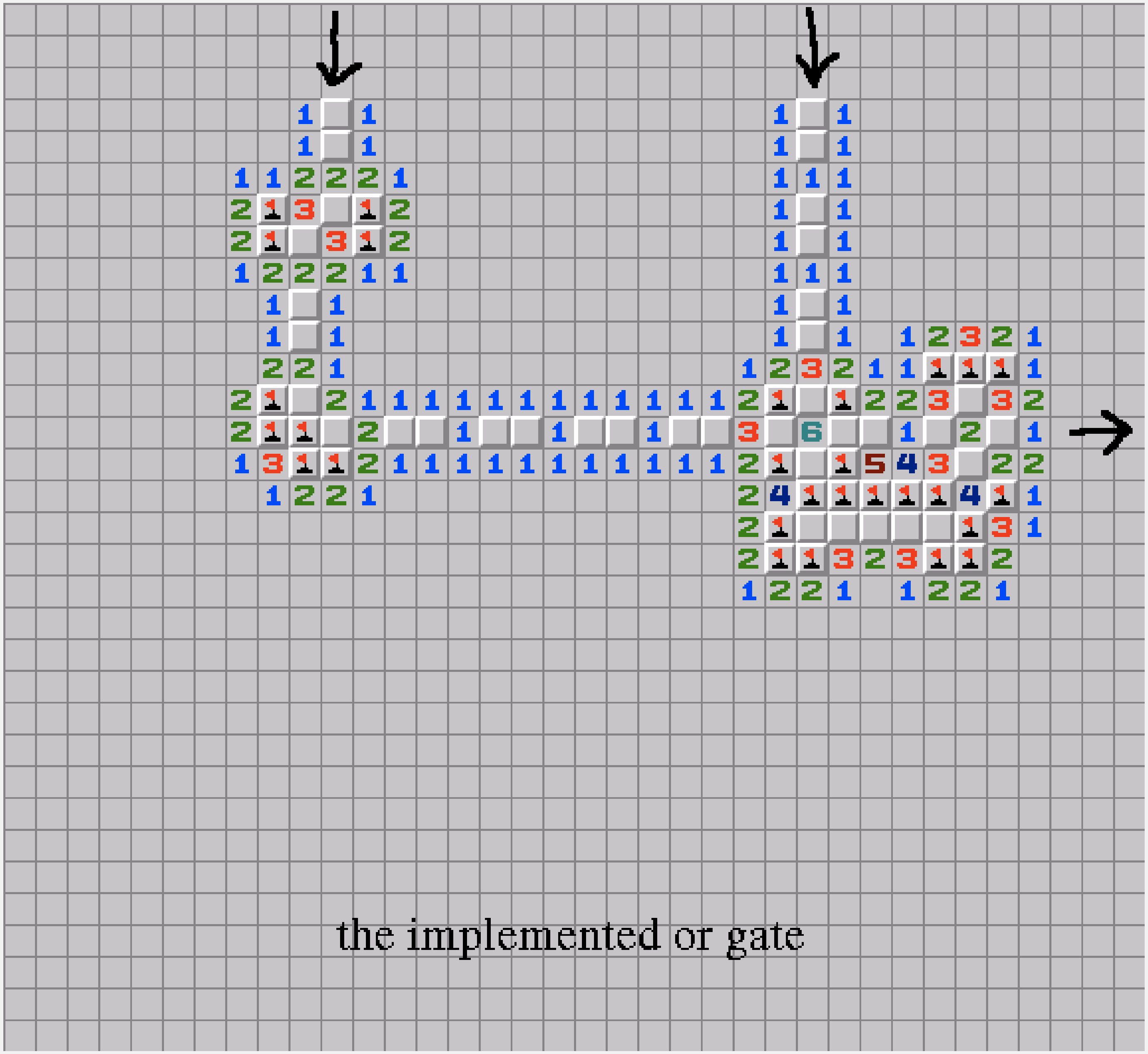
- Nobody knows how to solve 3SAT efficiently.
- Nobody knows if an efficient solution exists.
  - Unknown if 3SAT is in P.

Other NP Complete problems?

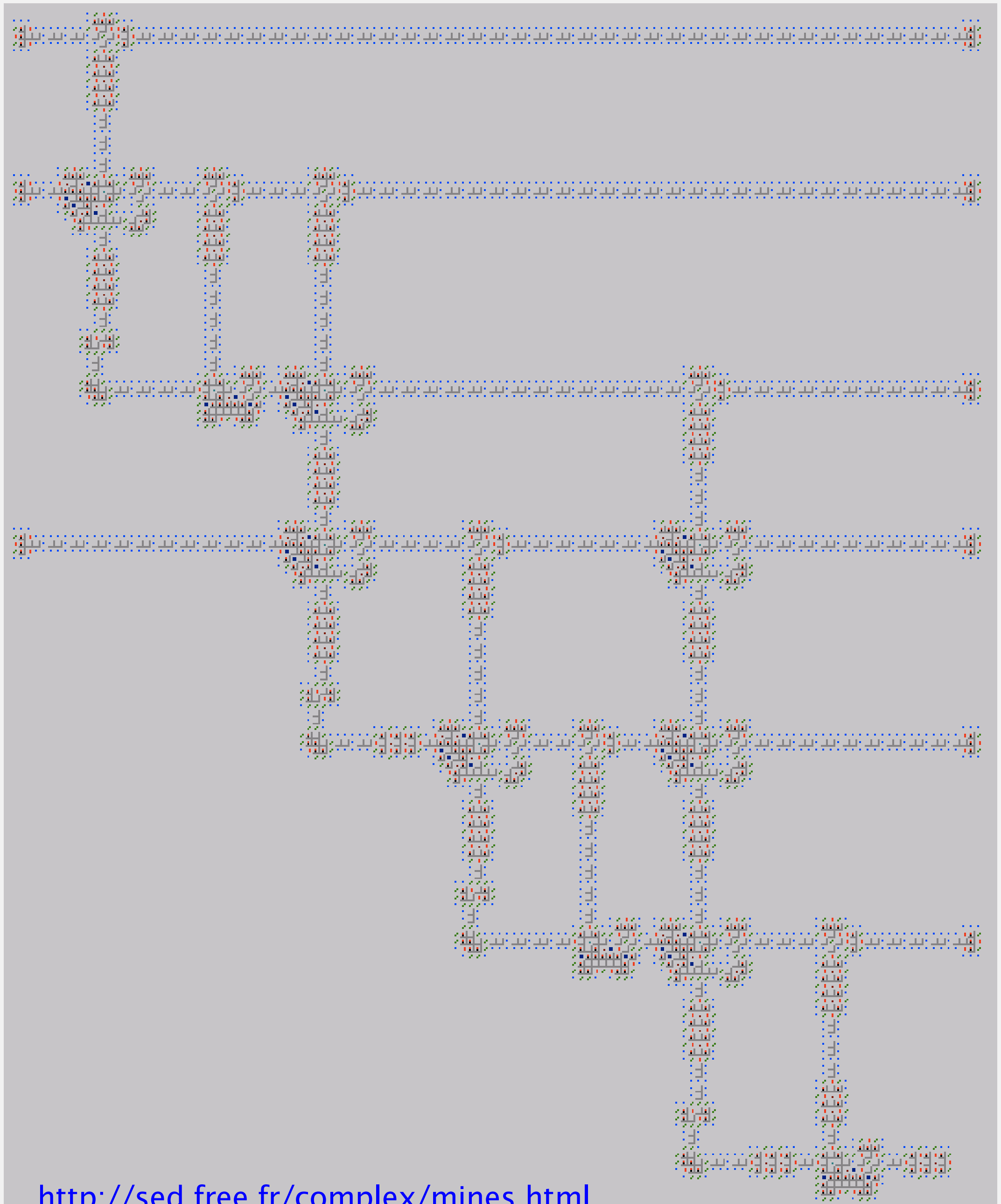
- Are there other keys to this magic kingdom?

A familiar NP-complete problem

http://sed.free.fr/complex/mines.html



A familiar NP-complete problem



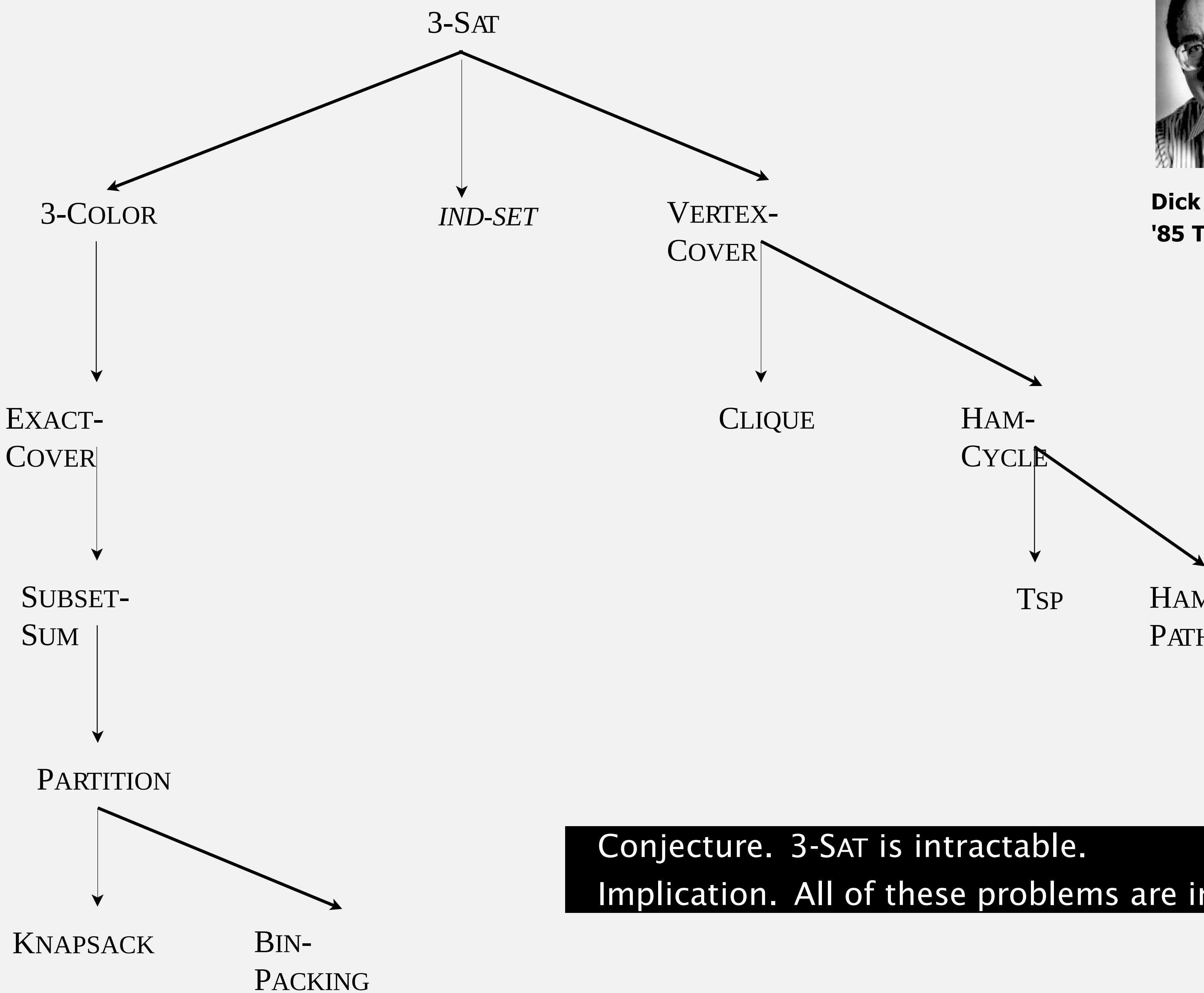
http://sed.free.fr/complex/mines.html

How to tell if your problem is NP Complete?

- Prove that it is in NP [easy].

- Prove that **some** NP Complete problem reduces to your problem [tricky!]

More poly-time reductions from 3-satisfiability

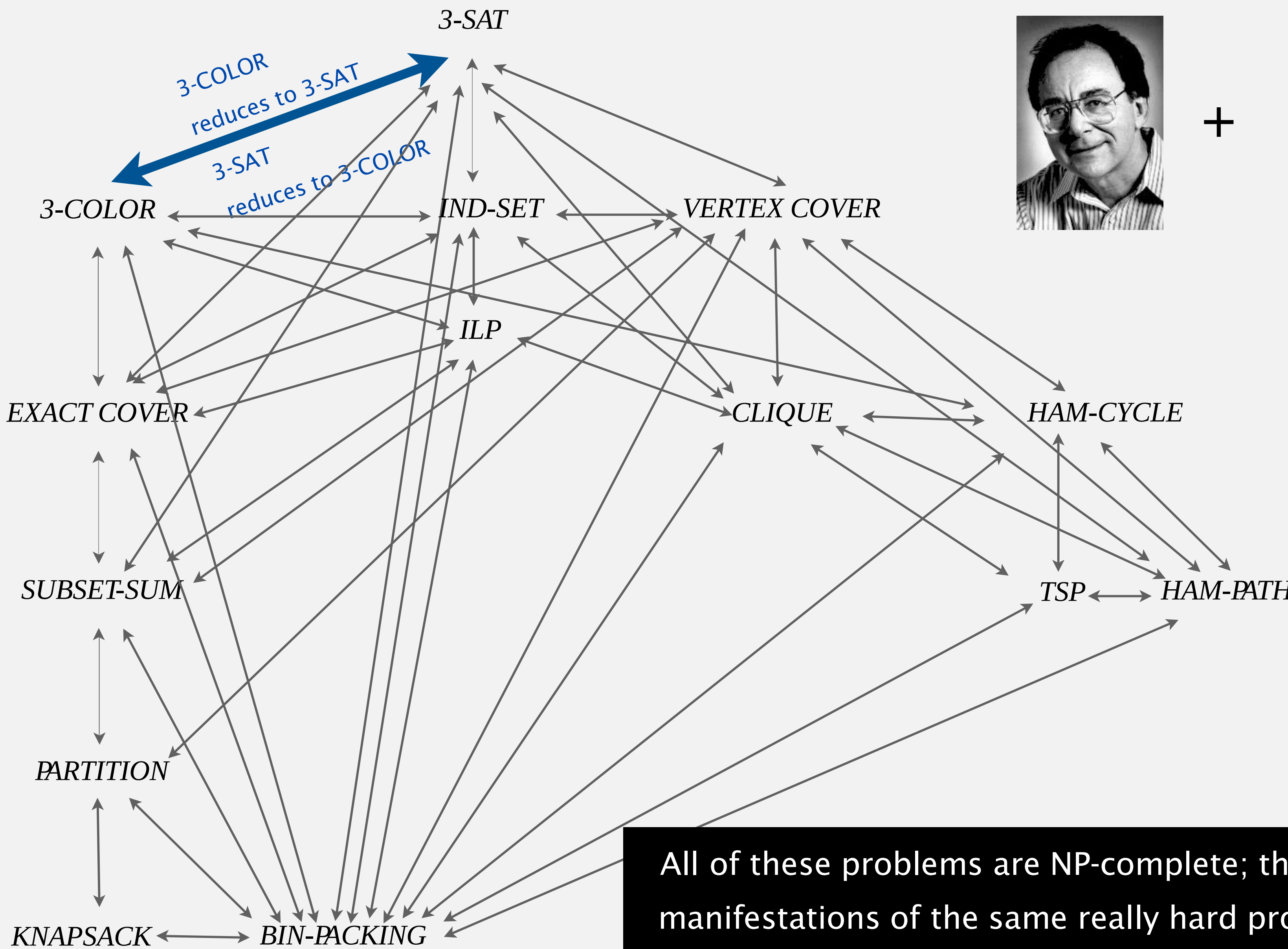


Dick Karp
'85 Turing award

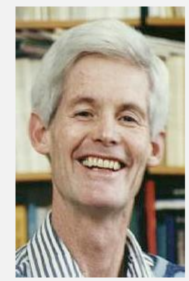
Conjecture. 3-SAT is intractable.

Implication. All of these problems are intractable.

Implications of Karp + Cook-Levin



+



All of these problems are NP-complete; they are manifestations of the same really hard problem.

Implications of poly-time reductions from 3-satisfiability

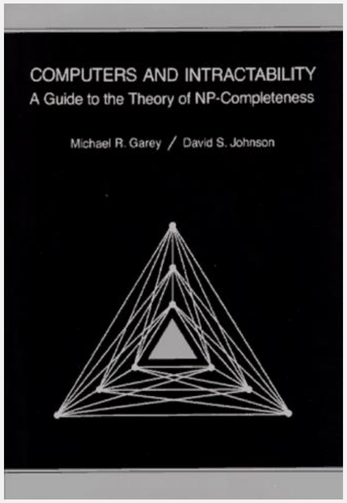
Establishing intractability through poly-time reduction is an important tool in guiding algorithm design efforts.

Q. How to convince yourself that a new problem is (probably) intractable?

A1. [hard way] Long futile search for an efficient algorithm (as for 3-SAT).

A2. [easy way] Reduction from 3-SAT.

Caveat. Intricate reductions are common.



Quiz 3: Implication of reductions

Which of these statements are true?

A. If problem X polynomial-time reduces to problem Y, and Y is solvable in polynomial time, then X is also polynomial-time solvable

B. If problem X polynomial-time reduces to problem Y, and X is polynomial-time solvable, then so is Y

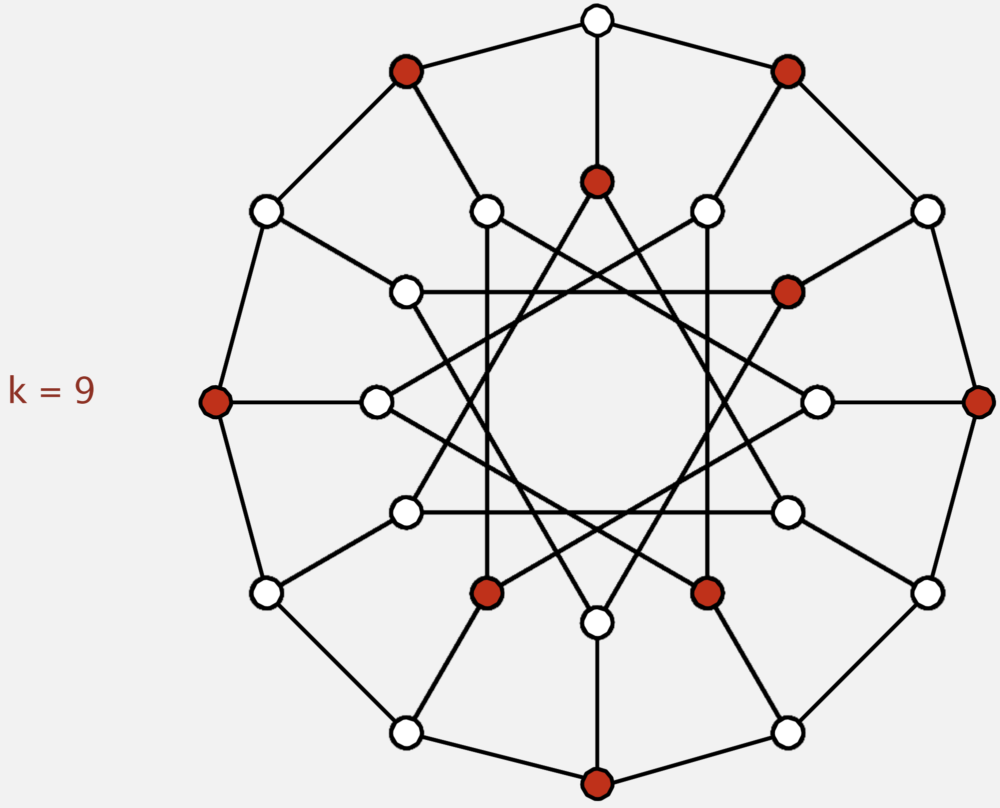
C. If problem X polynomial-time reduces to problem Y, and X is **not** polynomial-time solvable, then neither is Y

D. If problem X polynomial-time reduces to problem Y, and Y is **not** polynomial-time solvable, then neither is X

Independent set

An independent set is a set of vertices, no two of which are adjacent.

IND-SET. Given graph G and an integer k, find an independent set of size k.



Applications. Scheduling, computer vision, clustering, ...

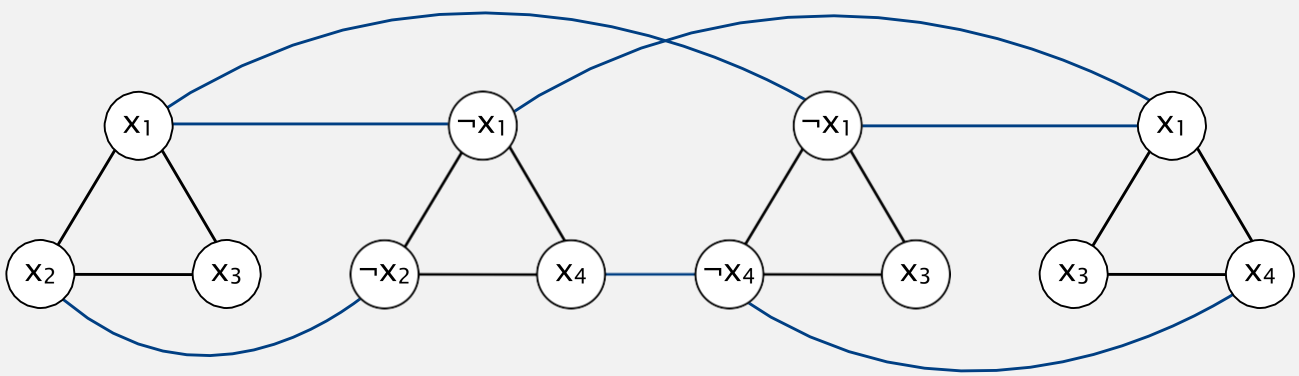
3-satisfiability reduces to independent set

Proposition. *3-SAT* poly-time reduces to *IND-SET*. ← lower-bound mentality: if I could solve IND-SET efficiently, I could solve 3-SAT efficiently

Pf. Given an instance Φ of *3-SAT*, create an instance G of *IND-SET*:

- For each clause in Φ, create 3 vertices in a triangle.
- Add an edge between each literal and its negation.

k = 4



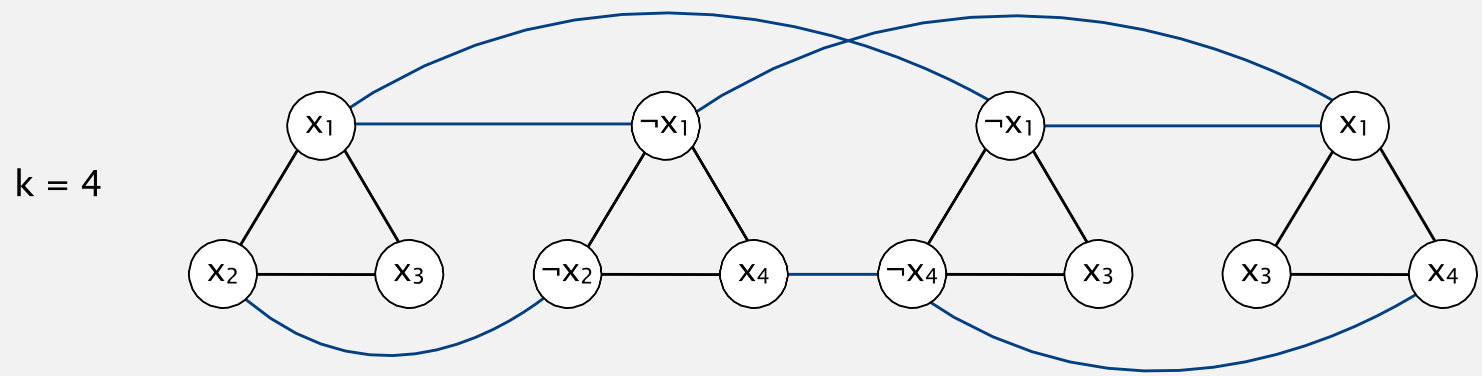
$$\Phi = (x_1 \text{ or } x_2 \text{ or } x_3) \text{ and } (\neg x_1 \text{ or } \neg x_2 \text{ or } x_4) \text{ and } (\neg x_1 \text{ or } x_3 \text{ or } \neg x_4) \text{ and } (x_1 \text{ or } x_3 \text{ or } x_4)$$

3-satisfiability reduces to independent set

Proposition. *3-SAT* poly-time reduces to *IND-SET*.

Pf. Given an instance Φ of *3-SAT*, create an instance G of *IND-SET*:

- For each clause in Φ, create 3 vertices in a triangle.
- Add an edge between each literal and its negation.



$$\Phi \;=\; (x_1 \text{ or } x_2 \text{ or } x_3) \text{ and } (\neg x_1 \text{ or } \neg x_2 \text{ or } x_4) \text{ and } (\neg x_1 \text{ or } x_3 \text{ or } \neg x_4) \text{ and } (x_1 \text{ or } x_3 \text{ or } x_4)$$

- Φ satisfiable $\Rightarrow$ G has independent set of size k.

for each of k clauses, include in independent set one vertex corresponding to a true literal

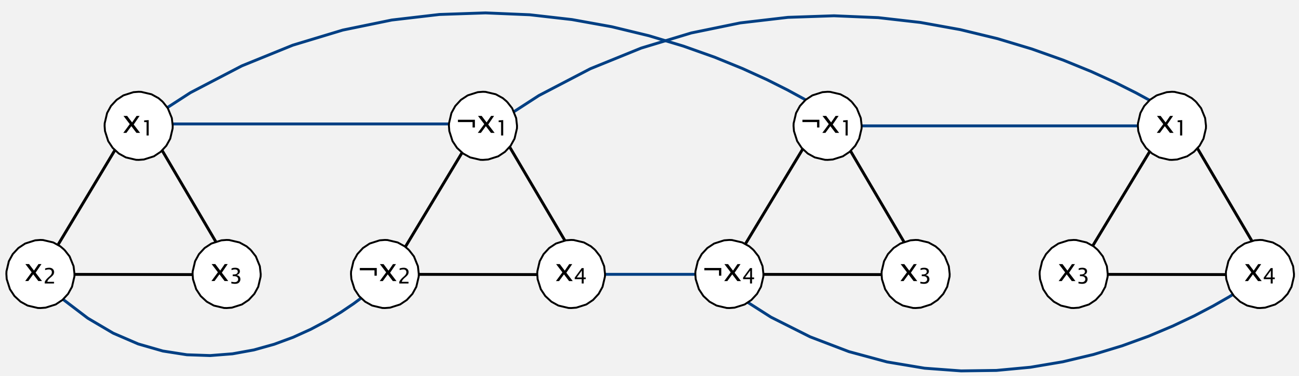
3-satisfiability reduces to independent set

Proposition. *3-SAT* poly-time reduces to *IND-SET*.

Pf. Given an instance Φ of *3-SAT*, create an instance G of *IND-SET*:

- For each clause in Φ, create 3 vertices in a triangle.
- Add an edge between each literal and its negation.

k = 4



$$\Phi \;=\; (x_1 \text{ or } x_2 \text{ or } x_3) \text{ and } (\neg x_1 \text{ or } \neg x_2 \text{ or } x_4) \text{ and } (\neg x_1 \text{ or } x_3 \text{ or } \neg x_4) \text{ and } (x_1 \text{ or } x_3 \text{ or } x_4)$$

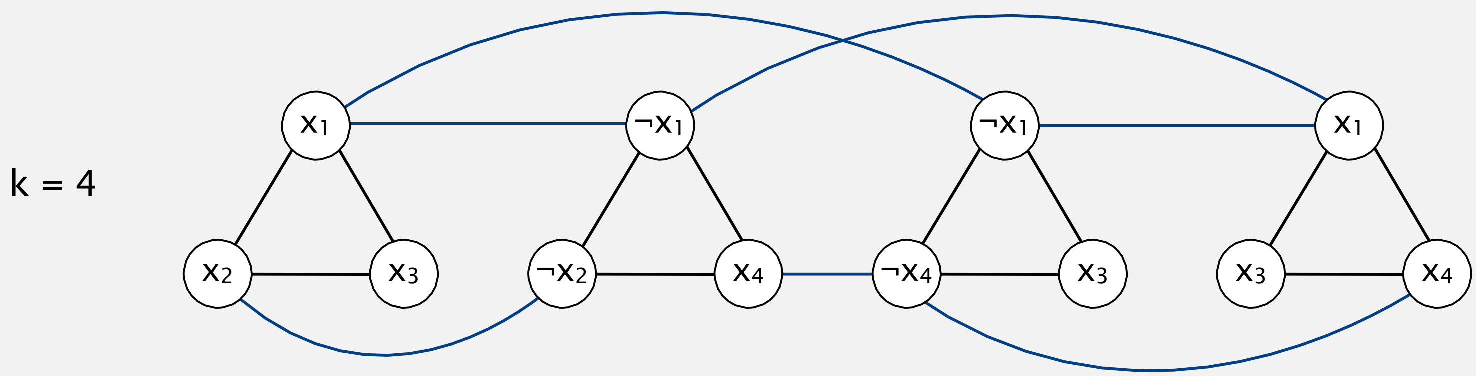
- Φ satisfiable $\Rightarrow$ G has independent set of size k.
- G has independent set of size k $\Rightarrow$ Φ satisfiable.

set literals corresponding to k vertices in independent set to true
(set remaining literals in any consistent manner)

3-satisfiability reduces to independent set

Proposition. *3-SAT* poly-time reduces to *IND-SET*.

Implication. Assuming *3-SAT* is intractable, so is *IND-SET*.



$$\Phi = (x_1 \text{ or } x_2 \text{ or } x_3) \text{ and } (\neg x_1 \text{ or } \neg x_2 \text{ or } x_4) \text{ and } (\neg x_1 \text{ or } x_3 \text{ or } \neg x_4) \text{ and } (x_1 \text{ or } x_3 \text{ or } x_4)$$

Summary

Reductions are important in theory to:

- Design algorithms.
- Establish lower bounds.
- Classify problems according to their computational requirements.

Reductions are important in practice to:

- Design algorithms.
- Design reusable software modules.
  - stacks, queues, priority queues, symbol tables, sets, graphs
  - sorting, regular expressions, suffix arrays
  - MST, shortest path, maxflow, linear programming
- Determine difficulty of your problem and choose the right tool.

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

Reductions and Beyond

- Intro
- Reductions
- NP Completeness
- Dealing with NP (or harder) problems
- P=NP

Approximation is usually OK

Most of the time, it's not "all or nothing"

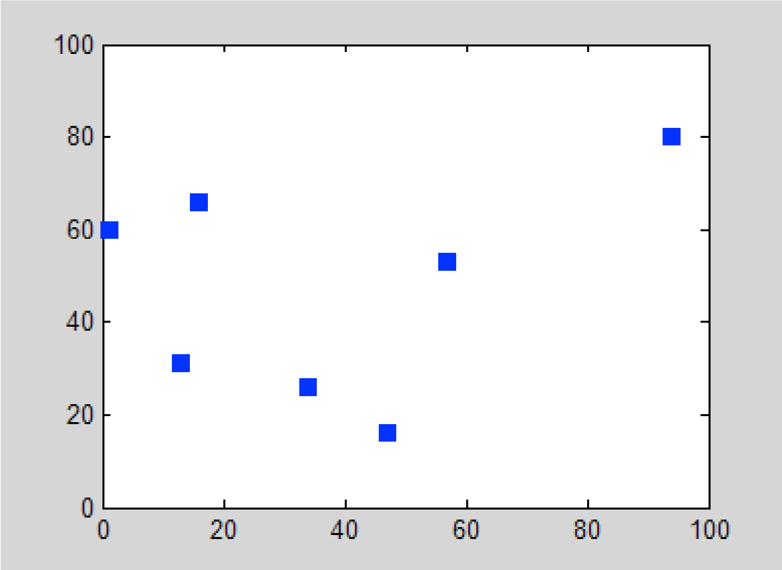
- Bin packing

- Actual traveling salesperson

Approximation

Approach 1: Approximate Heuristics

- Accept incorrect answers
  - TSP, always go to closest city next



http://en.wikipedia.org/wiki/Travelling_salesman_problem

Smarter Searching

Approach 2: Exact Heuristics

- Use a smarter (but still worst case intractable) solution technique
  - Like backtracking and DPLL

Consider restricted cases

Approach 3: Take advantage of special structure

- Realize that your problem is actually a special, solvable case.
  - Example 1: Actually in P.
  - Example 2: Worse than P, but only a little.

Examples

- 2-Satisfiability

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

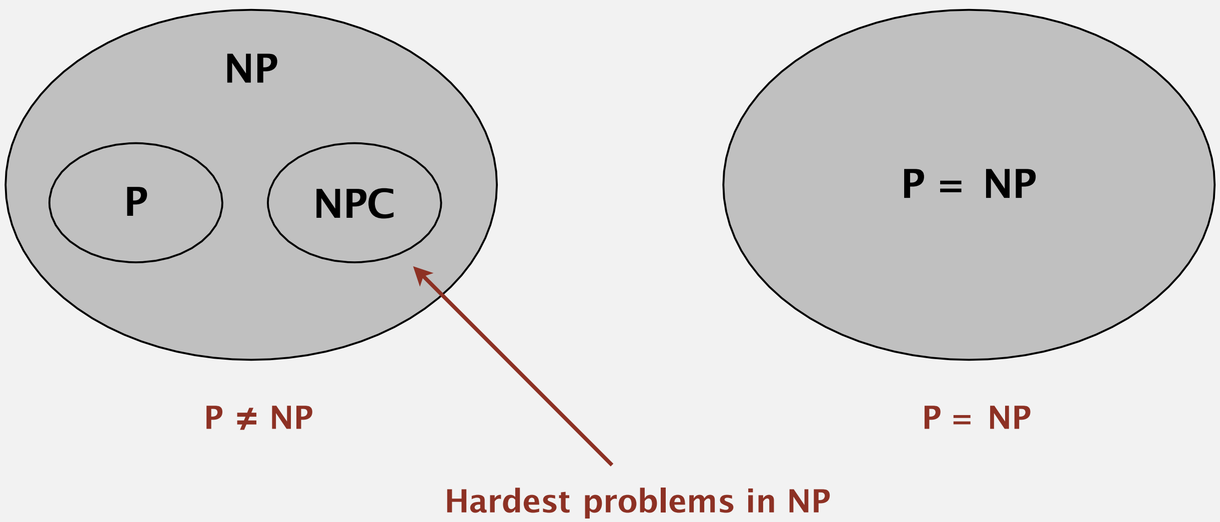
Reductions and Beyond

- Intro
- Reductions
- NP completeness
- Dealing with NP (or harder) problems
- P=NP question

P = NP?

Does P = NP?

- Equivalently: Is any NP Complete problem also in P?
- Equivalently: Efficiently verifiable ⇒ efficiently solvable?

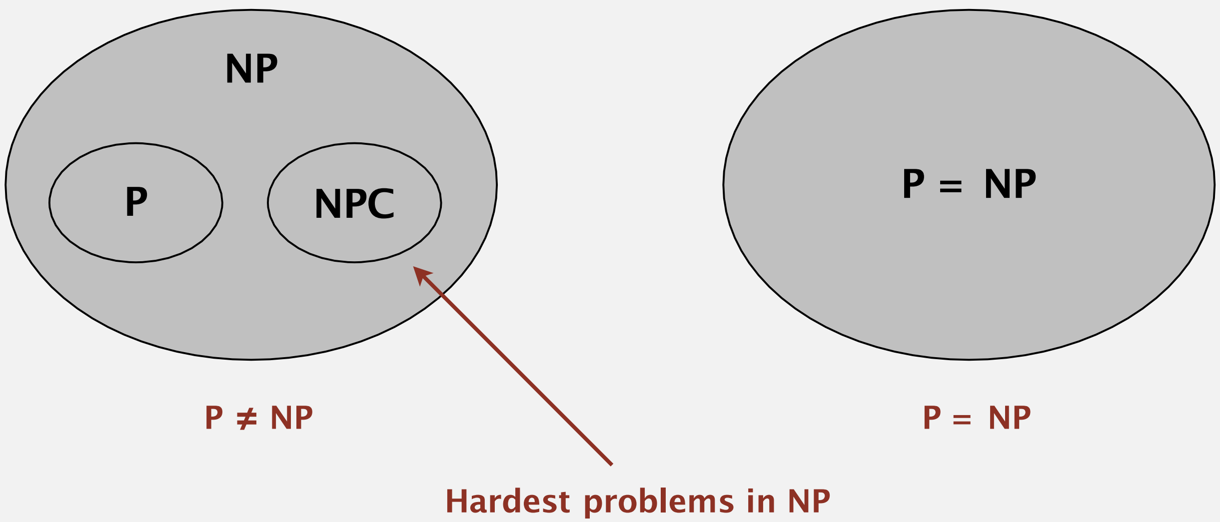


Reminder: NP may as well have been called VP for “Verifiable in Polynomial Time”

P = NP?

Does P = NP?

- Equivalently: Is any NP Complete problem also in P?
- Equivalently: Efficiently verifiable ⇒ efficiently solvable?



Why is P considered efficient? Why is exponential time inefficient?

- n^10000?
- 2^(1.00000000000000000001)?
- Known solutions to practical problems simply don't look like this.

P = NP?

Consensus Opinion (Bill Gasarch poll, 2002)

- 9 - Yes
- 61 - No
- 4 - Independent of axiomatic systems typically used in considering the problem.

Why is opinion generally negative?

- Someone would have proved it by now.
  - "The only supporting arguments I can offer are the failure of all efforts to place specific NP-complete problems in P by constructing polynomial-time algorithms." - Dick Karp
- Creation of solutions seems philosophically more difficult than verification.
- Mathematical reasons: Way beyond scope of course (and my understanding)

One of these things is not like the other...

Millenium Prize Problems

- Hodge Conjecture
- Poincare Conjecture (solved!)
- Riemann Hypothesis
- Yang-Mills existence and mass gap
- Navier-Stokes existence and smoothness
- Birch and Swinnerton-dyer conjecture
- P=NP
  - If true, proof might allow you to trivially solve all of the other problems.

But what if P = NP?

"[A linear or quadratic-time procedure for what we now call NP-complete problems would have] consequences of the greatest magnitude. [For such an procedure] would clearly indicate that, despite the unsolvability of the Entscheidungsproblem, the mental effort of the mathematician in the case of yes-or-no questions could be completely replaced by machines." — Kurt Gödel

Next week

Friday 3 Nov:

- Go over exams from 2014 and 2015
- I will post the exam from 2016 for you to practice on

Wednesday 1 Nov

- Option 1: Whirl-wind tour through the whole course
- Option 2: Go deeper into some topics, with additional problems.

Memo: Course evaluations