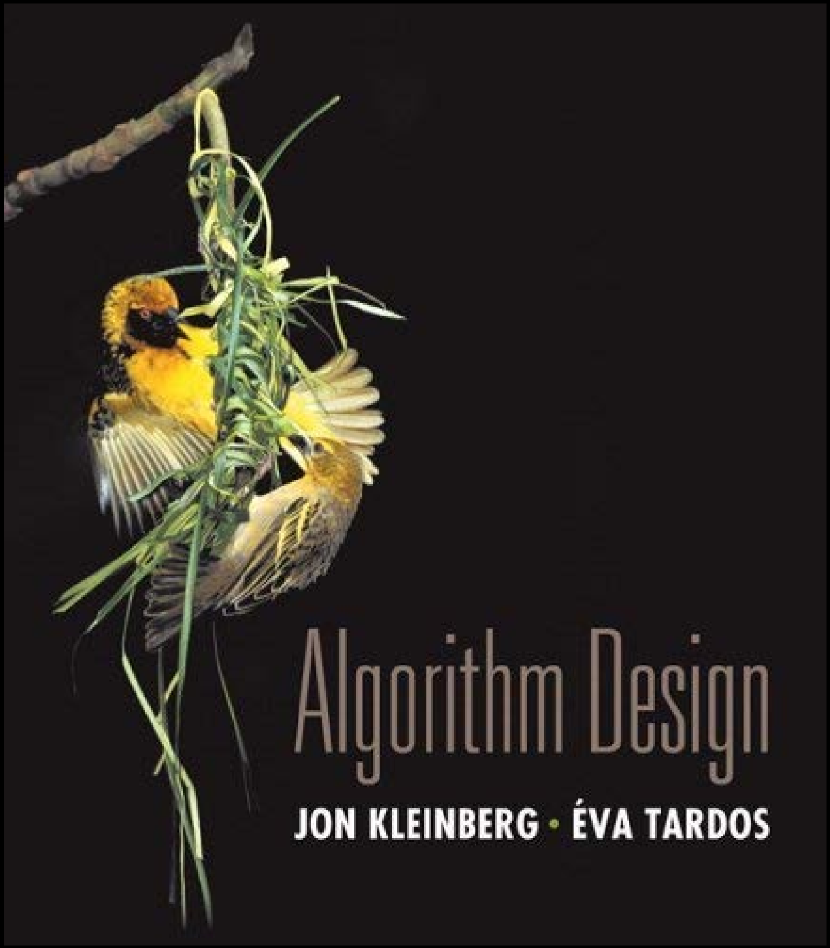


Chapter 8

NP and Computational Intractability

Slides by S. Raskhodnikova; and A. Smith

What algorithms are (im)possible?

Algorithm design patterns.	Examples.
▪ Greedy.	O(n log n) interval scheduling.
▪ Divide-and-conquer.	O(n log n) sorting.
▪ Dynamic programming.	$O(n^2)$ edit distance.
▪ Augmenting paths.	Max flow.
▪ Simplex method	Linear programming
▪ Reductions.	Maximum matcing
▪ … (lots more out there)	

New goal: understand what is hard to compute.	
▪ NP-completeness.	$O(n^k)$ algorithm unlikely.
▪ PSPACE-completeness.	$O(n^k)$ certification algorithm unlikely.
▪ Undecidability.	No algorithm possible.

Intractability: Central ideas we'll cover

- Poly-time as "feasible"
  - most natural problems either are easy (say n^3) or have no known poly-time algorithms
- P = problems that are easy to answer
  - e.g. minimum cut
- NP = {problems whose answers are easy to verify given hint}
  - e.g. graph 3-coloring
- Reductions: X is no harder than Y
- NP-completeness
  - many natural problems are easy if and only if P=NP

Polynomial-Time Reductions

Classify Problems According to Computational Requirements

Q. Which problems will we be able to solve in practice?

A working definition. [Cobham 1964, Edmonds 1965, Rabin 1966]
Those with polynomial-time algorithms.

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.

- 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 lecture. 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 **poly-time reduces 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.

computational model supplemented by special piece of hardware that solves instances of Y in a single step

Notation. $X \le_{p,\text{Cook}} Y$ (or $X \le_P Y$).
Later in the lecture. $X \le_{p,\text{Karp}} Y$.

Remarks.

- We pay for time to write down instances sent to black box $\Rightarrow$ instances of Y must be of polynomial size.

Polynomial-Time Reduction

Purpose. Classify problems according to relative difficulty.

Design algorithms. If $X \le_P Y$ and Y can be solved in polynomial-time, then X can also be solved in polynomial time.

Establish intractability. If $X \le_P Y$ and X cannot be solved in polynomial-time, then Y cannot be solved in polynomial time.

Establish equivalence. If $X \le_P Y$ and $Y \le_P X$, we use notation $X \equiv_P Y$.

up to cost of reduction

Simplifying Assumption: Decision Problems

Search problem. Find some structure.
Example. Find a minimum cut.

Decision problem.
- X is a set of strings.
- Instance: string s.
- If $x \in X$, x is a YES instance; if $x \notin X$ is a NO instance.
- Algorithm A solves problem X: $A(s)$ = yes iff $s \in X$.

Example. Does there exist a cut of size $\leq k$?

Self-reducibility. Search problem $\leq_{P,\ Cook}$ decision version.
- Applies to all (NP-complete) problems in Chapter 8 of KT.
- Justifies our focus on decision problems.

Polynomial Transformation

Def. Problem X poly-time 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 poly-time transforms (Karp) to problem Y if given any input x to X, we can construct an input y such that
x is a yes instance of X iff
y is a yes instance of Y.

we require |y| to be of size polynomial in |x|

Note. Poly-time transformation is poly-time reduction with just one call to oracle for Y, exactly at the end of the algorithm for X.

Open question. Are these two concepts the same?

Caution: KT abuses notation $\leq_p$ and blurs distinction

Basic reduction strategies

Basic reduction strategies

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

Independent Set

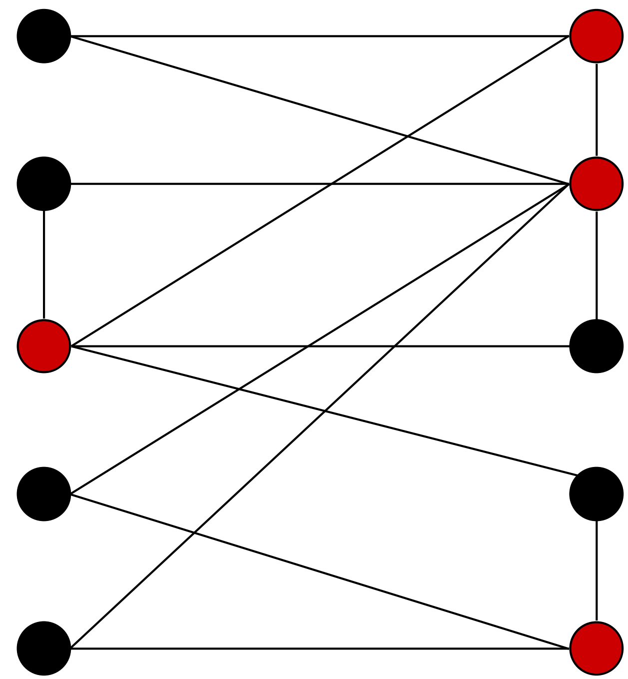
Given an undirected graph G, an **independent set** in G is a set of nodes, which includes at most one endpoint of every edge.

INDEPENDENT SET $= \{\langle G, k\rangle \mid G$ is an undirected graph which has an independent set with k nodes$\}$

- Is there an independent set of size ≥ 6?
  - Yes.

independent set

- Is there an independent set of size ≥ 7?
  - No.



Vertex Cover

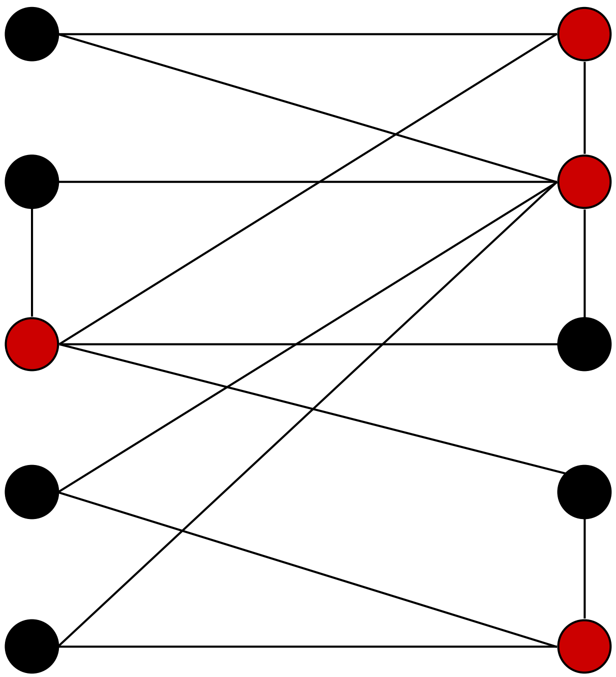
Given an undirected graph G, a **vertex cover** in G is a set of nodes, which includes at ***least*** one endpoint of every edge.

VERTEX COVER $= \{\langle G, k\rangle \mid G$ is an undirected graph which has a vertex cover with k nodes$\}$

- Is there vertex cover of size ≤ 4?
  - Yes.

vertex cover

- Is there a vertex cover of size ≤ 3?
  - No.



Independent Set and Vertex Cover

Claim. S is an independent set iff V – S is a vertex cover.

- $\Rightarrow$
  - Let S be any independent set.
  - Consider an arbitrary edge (u, v).
  - S is 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).
- $\Leftarrow$
  - Let V – S be any vertex cover.
  - Consider two nodes $u \in S$ and $v \in S$.
  - Then $(u, v) \notin E$ since V – S is a vertex cover.
  - Thus, no two nodes in S are joined by an edge $\Rightarrow$ S independent set. ▪

INDEPENDENT SET reduces to VERTEX COVER

Theorem. INDEPENDENT-SET $\leq_p$ VERTEX-COVER**.**

Proof. "On input $\langle G, k \rangle$, where G is an undirected graph and k is an integer,

1. Output $\langle G, n-k \rangle$, where n is the number of nodes in G."

Correctness:

- G has an independent set of size k iff it has a vertex cover of size $n - k$.
- Reduction runs in linear time.

Reduction: special case to general case

Basic reduction strategies

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

Set Cover

Given a set U, called a *universe*, and a collection of its subsets $S_1, S_2, \dots, S_m$, a **set cover** of U is a subcollection of subsets whose union is U.

- SET COVER=$\{\langle U, S_1, S_2, \dots, S_m; k\rangle \mid$

U has a set cover of size $k\}$

U = { 1, 2, 3, 4, 5, 6, 7 }

k = 2

S_1 = {3, 7}	S_4 = {2, 4}
S_2 = {3, 4, 5, 6}	S_5 = {5}
S_3 = {1}	S_6 = {1, 2, 6, 7}

- Sample application.
  - m available pieces of software.
  - Set U of n capabilities that we would like our system to have.
  - The ith piece of software provides the set $S_i \subseteq U$ of capabilities.
  - Goal: achieve all n capabilities using fewest pieces of software.

VERTEX COVER reduces to SET COVER

Theorem. VERTEX-COVER $\leq_P$ SET-COVER.

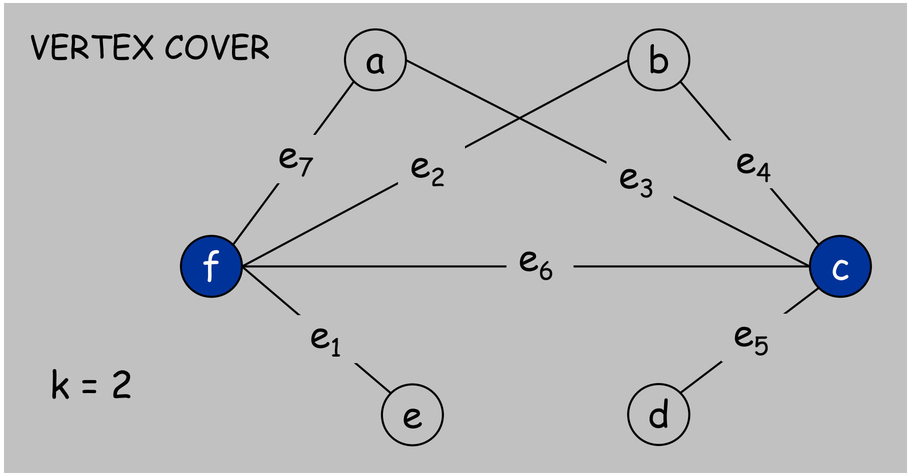
Proof. "On input $\langle G, k \rangle$, where $G = (V, E)$ is an undirected graph and k is an integer,

1. Output $\langle U, S_1, S_2, \ldots, S_m; k \rangle$, where U=E and

$$S_v = \{e \in E : e \text{ incident to } v\}"$$

Correctness:

- G has a vertex cover of size k iff U has a set cover of size k.
- Reduction runs in linear time.



SET COVER

U = { 1, 2, 3, 4, 5, 6, 7 }
k = 2

$S_a = \{3, 7\}$	$S_b = \{2, 4\}$
$S_c = \{3, 4, 5, 6\}$	$S_d = \{5\}$
$S_e = \{1\}$	$S_f = \{1, 2, 6, 7\}$

Reduction by encoding with gadgets

Basic reduction strategies

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

Satisfiability

- **Boolean variables:** variables that can take on values T/F (or 1/0)
- **Boolean operations:** ∨, ∧, and ¬
- **Boolean formula:** expression with Boolean variables and ops

SAT = {⟨Φ⟩ | Φ is a satisfiable Boolean formula}

- **Literal:** A Boolean variable or its negation. x_i or $\overline{x_i}$
- **Clause:** OR of literals. $C_j = x_1 \vee \overline{x_2} \vee x_3$
- **Conjunctive normal form (CNF):** AND of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

3SAT = {⟨Φ⟩ | Φ is a satisfiable Boolean CNF formula, where each clause contains exactly 3 literals}

each corresponds to a different variable

Ex: $\left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(x_2 \vee x_3\right) \wedge \left(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}\right)$

Yes: x_1 = true, x_2 = true x_3 = false.

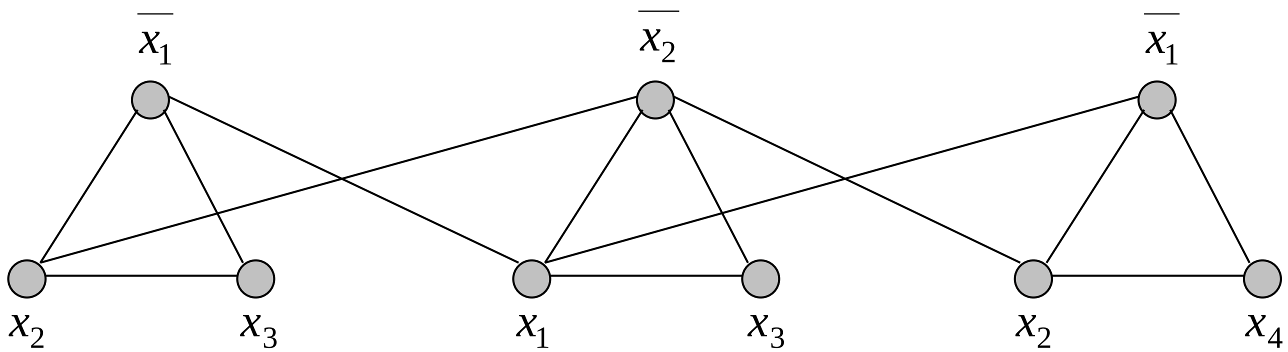
3SAT reduces to INDEPENDENT SET

Theorem. 3-SAT $\leq_P$ INDEPENDENT-SET.

Proof. "On input $\langle\Phi\rangle$, where Φ is a 3CNF formula,

1. Construct graph G from Φ
   - 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.
2. Output $\langle G, k\rangle$, where k is the number of clauses in Φ."

G



k = 3

$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4\right)$$

3SAT reduces to INDEPENDENT SET

Correctness. Let k = # of clauses and ℓ= # of literals in Φ.

Φ is satisfiable iff G contains an independent set of size k.

- $\Rightarrow$ Given satisfying assignment, select one true literal from each triangle. This is an independent set of size k.

- $\Leftarrow$ Let S be an independent set of size k.
  - S must contain exactly one vertex in each triangle.
  - Set these literals to true, and other literals in a consistent way.
  - Truth assignment is consistent and all clauses are satisfied.

Run time. O($k + \ell^2$), i.e. polynomial in the input size.

Summary

- Basic reduction strategies.
  - Simple equivalence: INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.
  - Special case to general case: VERTEX-COVER $\leq_P$ SET-COVER.
  - Encoding with gadgets: 3-SAT $\leq_P$ INDEPENDENT-SET.

- **Transitivity**. If $X \leq_P Y$ and $Y \leq_P Z$, then $X \leq_P Z$.
- Proof idea. Compose the two algorithms.
- Ex: 3-SAT $\leq_P$ INDEPENDENT-SET $\leq_P$ VERTEX-COVER $\leq_P$ 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 Chapter 8 of KT.
- 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

Definitions of P and NP

Decision Problems

Decision problem.

- X is a set of strings.
- Instance: string s.
- Algorithm A solves problem X: **A(s) = yes** iff **$s \in X$**.

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, …. }

Algorithm. [Agrawal-Kayal-Saxena, 2002] $p(|s|) = |s|^8$.

Definition of P

P. The class of decision problems for which there is a poly-time algorithm.

Examples

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	niether neither	acgggt ttttta
LSOLVE	Is there a vector x that satisfies Ax = b?	Gauss-Edmonds elimination	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

NP

Certification algorithm intuition.

- Certifier views things from "managerial" viewpoint.
- Certifier doesn't determine whether $s \in X$ on its own; 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 $C(s, t) =$ yes**.

"certificate" or "witness"

NP. Decision problems for which there exists a poly-time certifier.

$C(s, t)$ is a poly-time algorithm and $|t| \le p(|s|)$ for some polynomial $p(\cdot)$.

Remark. NP stands for nondeterministic polynomial-time.

Certifiers and Certificates: Composite

COMPOSITES. Given an integer s, is s composite?

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. ⟵ $437{,}669 = 541 \times 809$

Conclusion. COMPOSITES is in NP.

Certifiers and Certificates: 3-Satisfiability

SAT. Given a CNF formula Φ, is there a satisfying assignment?

Certificate. An assignment of truth values to the n boolean variables.

Certifier. Check that each clause in Φ has at least one true literal.

Ex.

$$\left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(x_1 \vee x_2 \vee x_4\right) \wedge \left(\overline{x_1} \vee \overline{x_3} \vee \overline{x_4}\right)$$

instance s

$$x_1 = 1,\ x_2 = 1,\ x_3 = 0,\ x_4 = 1$$

certificate t

Conclusion. SAT is in NP.

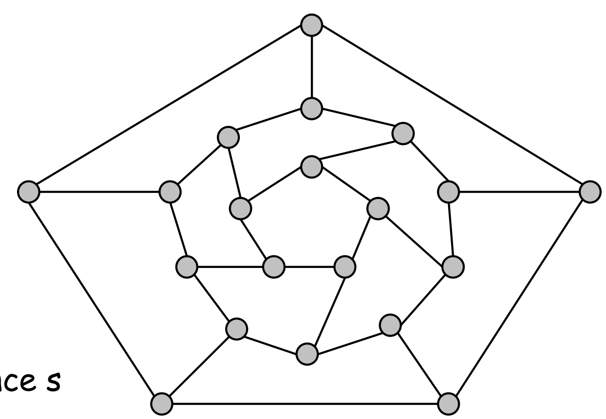
Certifiers and Certificates: Hamiltonian Cycle

HAM-CYCLE. Given an undirected graph G = (V, E), does there exist a simple cycle C that visits every node?

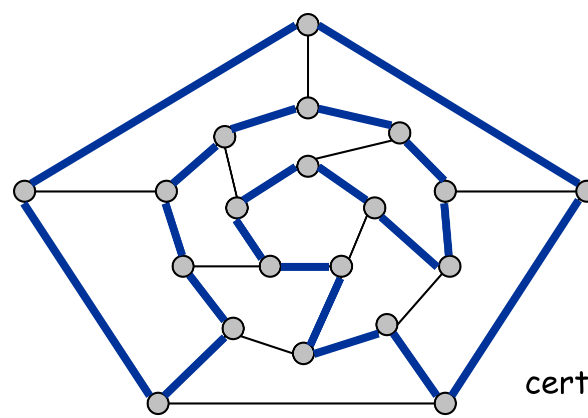
Certificate. A permutation of the n nodes.

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.



instance s



certificate t

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: $t = \varepsilon$, certifier $C(s, t) = A(s)$. ▪

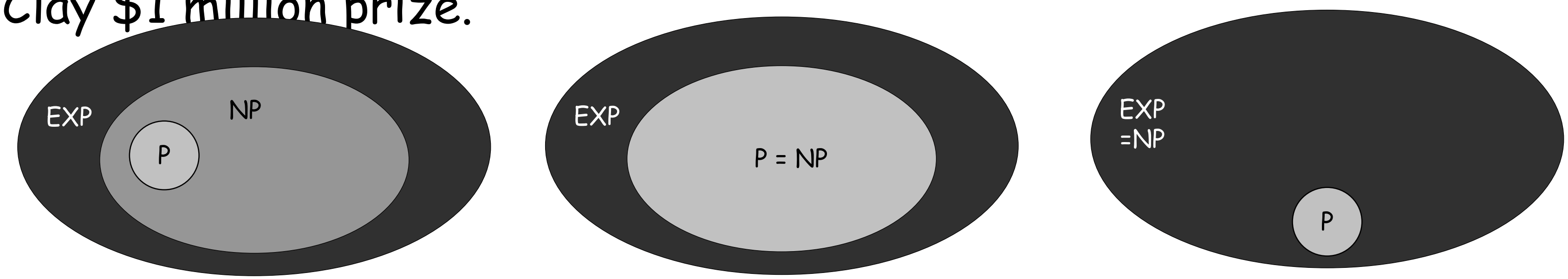
Claim. NP $\subseteq$ EXP.
Pf. Consider any problem X in NP.

- By definition, there exists a poly-time certifier $C(s, t)$ for X that runs in time $p(|s|)$.
- To solve input s, run $C(s, t)$ on all strings t with $|t| \leq p(|s|)$.
- Return yes, if $C(s, t)$ returns yes for any of these. ▪

The Big Question: P vs. NP

Does P = NP? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]

- Is the decision problem as easy as the verification problem?
- Clay $1 million prize.



If $P \neq NP$ If $P = NP$ If $P \neq NP$ and NP=EXP

If yes: Efficient algorithms for HamPath, SAT, TSP, factoring

- Cryptography is impossible*
- Creativity is automatable

If no: No efficient algorithms possible for these problems.

Consensus opinion on P = NP? Probably no.

NP-completeness

NP-Complete

NP-complete. A problem Y is NP-complete if

- Y is in NP and
- $X \leq_{p,Karp} Y$ for every problem X in NP.

Theorem. Suppose Y is an NP-complete problem. Then Y is solvable in poly-time iff P = NP.

Proof. $\Leftarrow$ If P = NP then Y can be solved in poly-time since Y is in NP.

$\Rightarrow$ Suppose Y can be solved in poly-time.

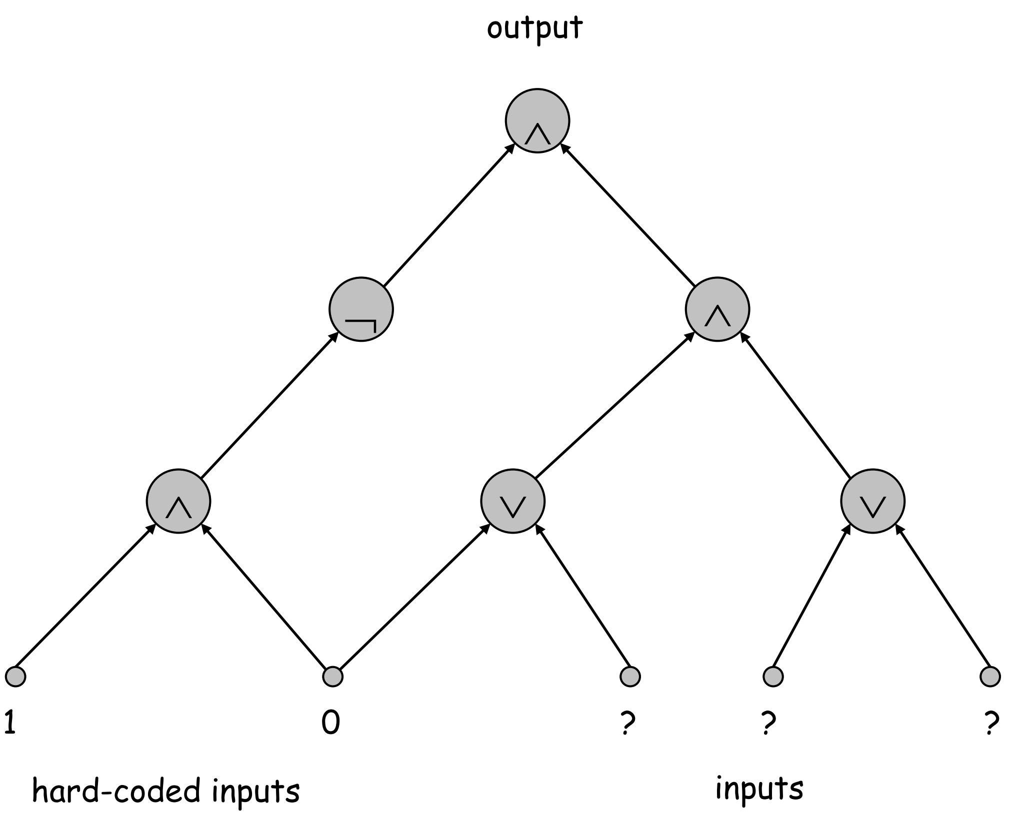
- Let X be any problem in NP. Since $X \leq_{p,Karp} Y$, we can solve X in poly-time. This implies NP $\subseteq$ P.
- We already know P $\subseteq$ NP. Thus P = NP. ▪

Fundamental question. Do there exist "natural" NP-complete problems?

Circuit Satisfiability

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?

yes: 1 0 1



The "First" NP-Complete Problem

Theorem. CIRCUIT-SAT is NP-complete. [Cook 1971, Levin 1973]

Proof sketch. CIRCUIT-SAT is in NP (certificate: input on which circuit is 1).

Reduction: **For all $X \in NP$, $A \le_{P,\text{Cook}}$ CIRCUIT-SAT.**

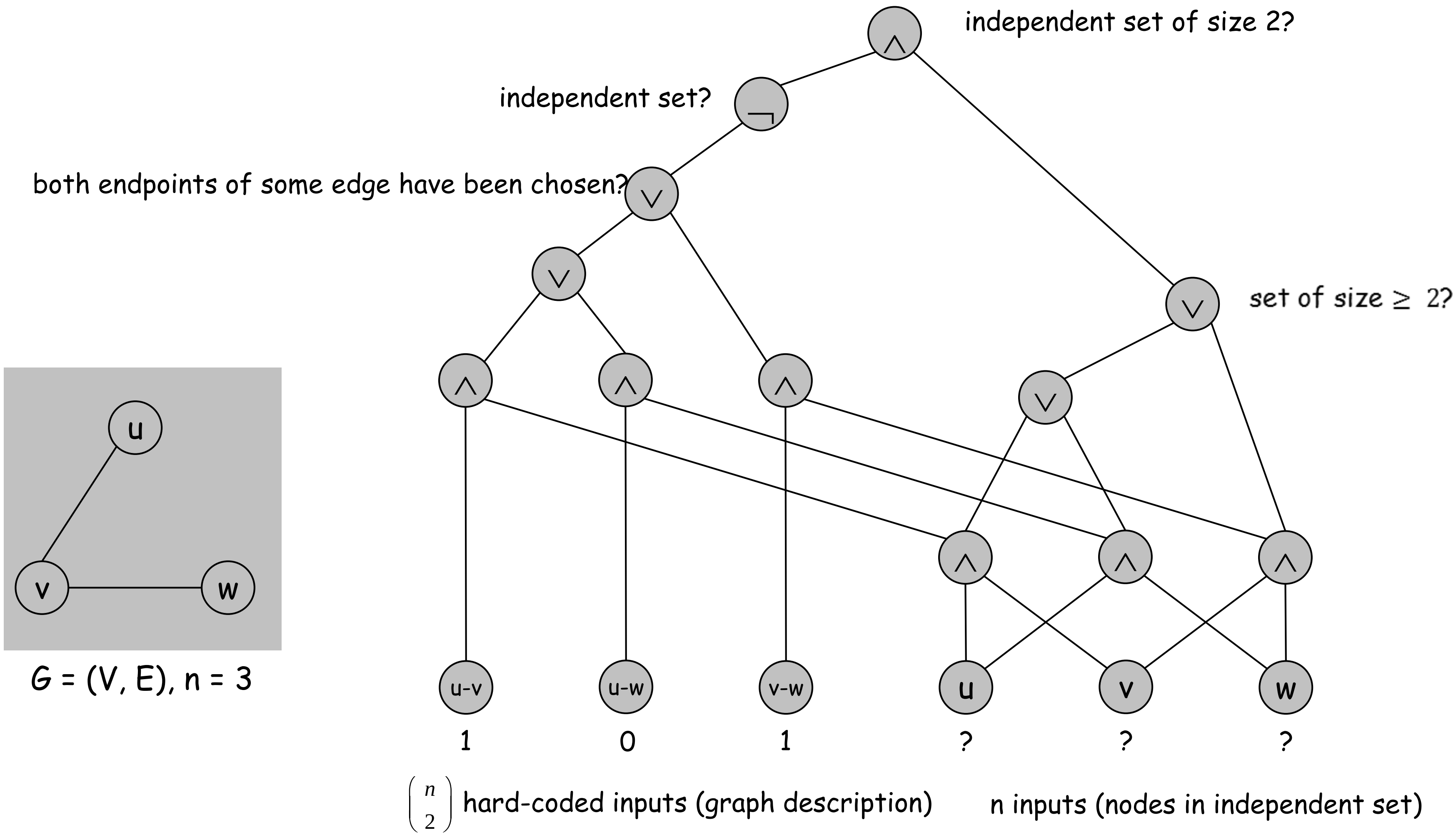
- Any algorithm that takes a fixed number of bits n as input and produces a yes/no answer can be represented by such a circuit.
  Moreover, if algorithm takes poly-time, then circuit is of poly-size.

  sketchy part of proof; fixing the number of bits is important, and reflects basic distinction between algorithms and circuits

- Since $X \in NP$, it has a poly-time certifier $C(s, t)$ that runs in time $p(|s|)$.
  To determine whether s is in X, need to know if there exists a certificate t of length $p(|s|)$ such that $C(s, t)$ = 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
- **Correctness:** Circuit K is satisfiable iff $C(s, t)$ = yes.

Example

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 $X \leq_{p,Karp} Y$.

Justification. If X is an NP-complete problem, and Y is a problem in NP with the property that $X \leq_{P,Karp} Y$ then Y is NP-complete.

Proof. Let W be any problem in NP. Then $W \underbrace{\leq_{P,Karp}}_{\text{by definition of NP-complete}} X \underbrace{\leq_{P,Karp}}_{\text{by assumption}} Y$.

- By transitivity, $W \leq_{P,Karp} Y$.
- Hence Y is NP-complete. ▪

3-SAT is NP-Complete

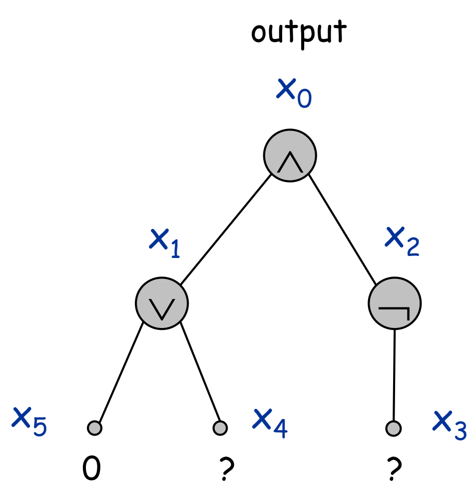
Theorem. 3-SAT is NP-complete.

Proof. 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: $\overline{x_0} \vee x_1$, $\overline{x_0} \vee x_2$, $x_0 \vee \overline{x_1} \vee \overline{x_2}$

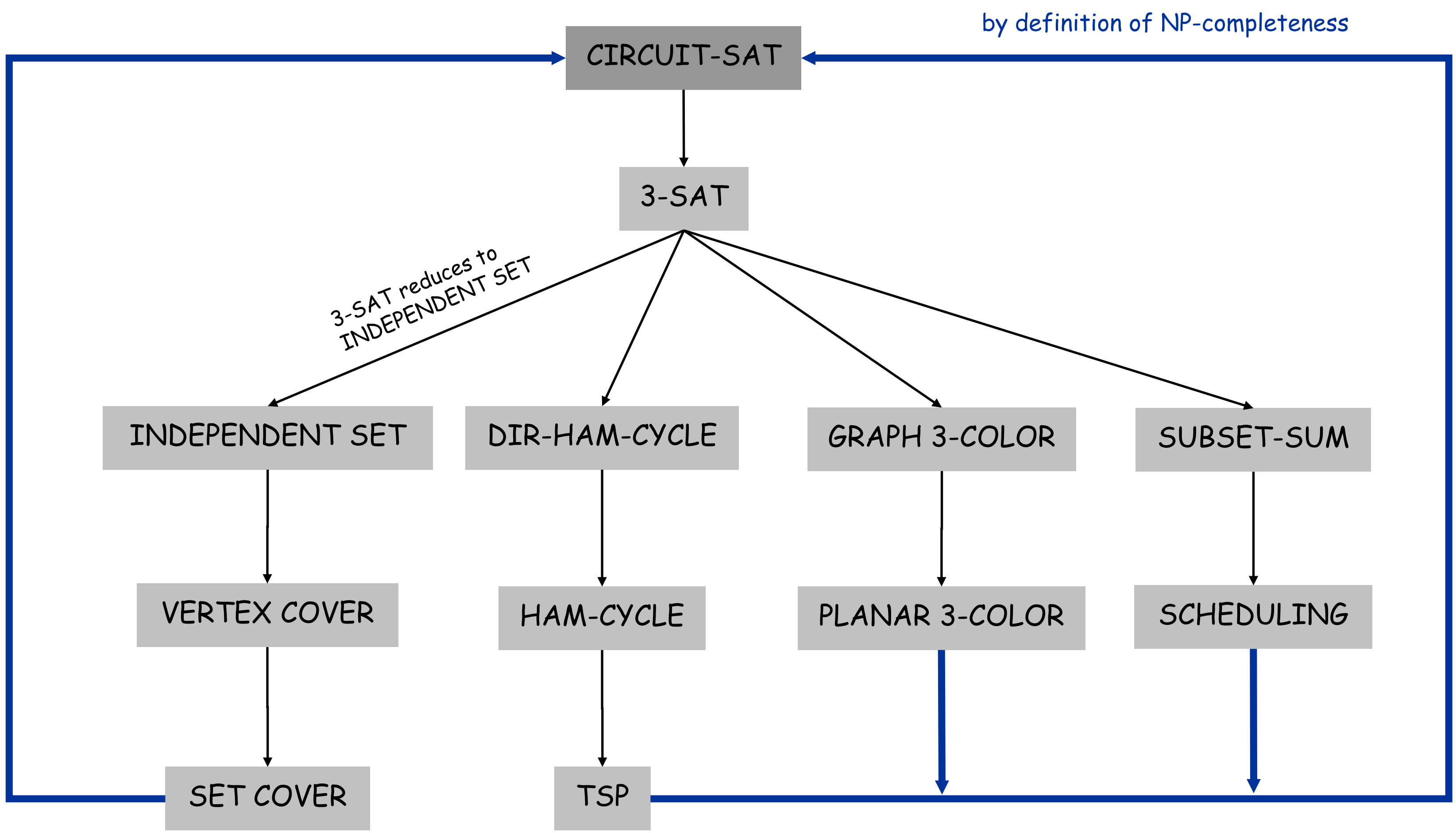
- Hard-coded input values and output value.
  - $x_5 = 0$ $\Rightarrow$ add 1 clause: $\overline{x_5}$
  - $x_0 = 1$ $\Rightarrow$ add 1 clause: x_0

- 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!



Some NP-Complete Problems

Six basic genres of NP-complete problems and paradigmatic examples.

- 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-MATCHIN,G 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

Practice. Most NP problems are either known to be in P or NP-complete.

Notable exceptions. Factoring, graph isomorphism, Nash equilibrium.

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.

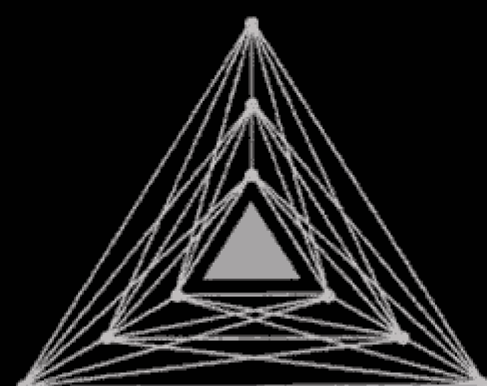
More Hard Computational Problems

Aerospace engineering: optimal mesh partitioning for finite elements.
Biology: protein folding.
Chemical engineering: heat exchanger network synthesis.
Civil engineering: equilibrium of urban traffic flow.
Economics: computation of arbitrage in financial markets with friction.
Electrical engineering: VLSI layout.
Environmental engineering: optimal placement of contaminant sensors.
Financial engineering: find minimum risk portfolio of given return.
Game theory: find Nash equilibrium that maximizes social welfare.
Genomics: phylogeny reconstruction.
Mechanical engineering: structure of turbulence in sheared flows.
Medicine: reconstructing 3-D shape from biplane angiocardiogram.
Operations research: optimal resource allocation.
Physics: partition function of 3-D Ising model in statistical mechanics.
Politics: Shapley-Shubik voting power.
Pop culture: Minesweeper consistency.
Statistics: optimal experimental design.

COMPUTERS AND INTRACTABILITY

A Guide to the Theory of NP-Completeness

Michael R. Garey / David S. Johnson



"I can't find an efficient algorithm, but neither can all these famous people."