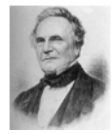
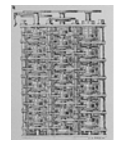
NP completeness
and computational tractability

Computational Tractability

As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise - By what course of calculation can these results be arrived at by the machine in the shortest time? - *Charles Babbage*



Charles Babbage (1864)



Analytic Engine (schematic)

How do we measure efficiency?

- platform independent, implementation-detail independent → ignore constant factors, use big O notation when we talk about running time.

- instance independent → worst-case analysis (sometimes average case analysis)

- of predictive value with respect to increasing input size, tells us how algorithm scales → want to measure rate of growth of T(n) as function of n, the input size.

Asymptotic, worst – case analysis
Seek polynomial time algorithms

Worst-Case Analysis

Worst case running time. Obtain bound on largest possible running time of algorithm on input of a given size N.

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

Average case running time. Obtain bound on running time of algorithm on random input as a function of input size N.

- Hard (or impossible) to accurately model real instances by random distributions.
- Algorithm tuned for a certain distribution may perform poorly on other inputs.

Polynomial-Time

Brute force. For many non-trivial 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.

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

> There exists constants $c > 0$ and $d > 0$ such that on every input of size N, its running time is bounded by $c N^d$ steps.

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

Worst-Case Polynomial-Time

Def. An algorithm is efficient if its running time is polynomial.

Justification: It really works in practice!

- Although $6.02 \times 10^{23} \times N^{20}$ is technically poly-time, it would be useless in practice.
- In practice, the poly-time algorithms that people develop almost always have low constants and low exponents.
- 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 are useless in practice.
- Some exponential-time (or worse) algorithms are widely used because the worst-case instances seem to be rare.

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

7

Moore's Law

The prediction that transistor density and hence the speed of computers will double every 18 months or so.

- Based on observation of 1960-- 1965
- Has pretty much held for last 40 years

Does this provide disincentive to develop efficient (polynomial time) algorithms?

8

Moore's Law

Does Moore's Law provide disincentive to develop efficient (polynomial time) algorithms?

NO!!

Running time of alg	Max input size in time T	2x speedup	2^{10}x speedup

Exponential algorithms make polynomially slow progress, while polynomial algorithms advance exponentially fast!

9

Asymptotic Order of Growth

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

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

Tight bounds. $T(n)$ is $\Theta(f(n))$ if $T(n)$ is both $O(f(n))$ and $\Omega(f(n))$.

Ex: $T(n) = 32n^2 + 17n + 32$.
- $T(n)$ is $O(n^2)$, $O(n^3)$, $\Omega(n^2)$, $\Omega(n)$, and $\Theta(n^2)$.
- $T(n)$ is not $O(n)$, $\Omega(n^3)$, $\Theta(n)$, or $\Theta(n^3)$.

10

Grand challenge: 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

11

Classify Problems

Desiderata. Classify problems according to those that can be solved in polynomial-time and those that cannot.

For any nice function T(n)
There are problems that require more than T(n) time to solve.

Frustrating news. Huge number of fundamental problems have defied classification for decades.

NP-completeness: Show that these fundamental problems are "computationally equivalent" and appear to be different manifestations of one really hard problem.

12

Polynomial-Time Reduction

Desiderata'. Suppose we could solve X in polynomial-time. What else could we solve in polynomial time?

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
- Polynomial number of calls to oracle that solves problem Y.

Notation. $X \le_P Y$.

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 $X \le_P Y$ 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

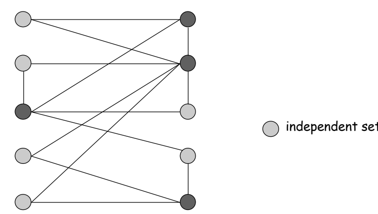
Reduction By Simple Equivalence

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 $|S| \ge k$, and for each edge at most one of its endpoints is in S?

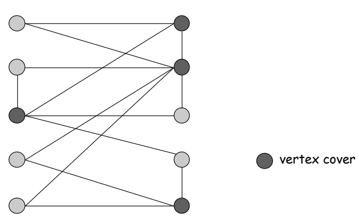
Ex. Is there an independent set of size ≥ 6? Yes.
Ex. Is there an independent set of size ≥ 7? No.



Vertex Cover

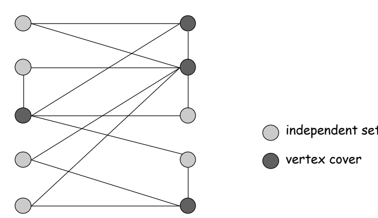
VERTEX COVER: Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \le k$, and for each edge, at least one of its endpoints is in S?

Ex. Is there a vertex cover of size ≤ 4? Yes.
Ex. Is there a vertex cover of size ≤ 3? No.



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.

$\Rightarrow$

- Let S be any independent set.
- Consider an arbitrary edge (u, v).
- 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).

$\Leftarrow$

- Let V − S be any vertex cover.
- Consider two nodes $u \in S$ and $v \in S$.
- 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.

Set Cover

SET COVER: Given a set U of elements, a collection $S_1, S_2, \ldots, S_m$ 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?

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.

Ex:

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

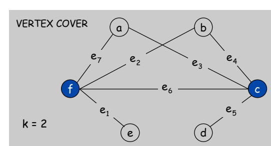
$S_1 = \{3, 7\}$	$S_4 = \{2, 4\}$
$S_2 = \{3, 4, 5, 6\}$	$S_5 = \{5\}$
$S_3 = \{1\}$	$S_6 = \{1, 2, 6, 7\}$

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.

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 $\leq k$. ▪



SET COVER

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

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

Polynomial-Time Reduction

Basic strategies.

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

Reductions via "Gadgets"

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction via "gadgets."

Satisfiability

Literal: A Boolean variable or its negation. x_i or $\overline{x_i}$

Clause: A disjunction of literals. $C_j = x_1 \lor \overline{x_2} \lor x_3$

Conjunctive normal form: A propositional formula Φ that is the conjunction of clauses. $\Phi = C_1 \land C_2 \land C_3 \land C_4$

SAT: Given CNF formula Φ, does it have a satisfying truth assignment?

3-SAT: SAT where each clause contains exactly 3 literals.

each corresponds to a different variable

Ex: $\left(\overline{x_1} \lor x_2 \lor x_3\right) \land \left(x_1 \lor \overline{x_2} \lor x_3\right) \land \left(x_2 \lor x_3\right) \land \left(\overline{x_1} \lor \overline{x_2} \lor \overline{x_3}\right)$

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

25

3 Satisfiability Reduces to Independent Set

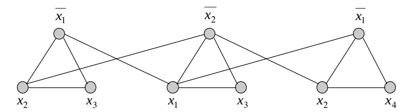
Claim. 3-SAT $\le_P$ INDEPENDENT-SET.

Pf. Given an instance Φ of 3-SAT, we construct an instance (G, k) of INDEPENDENT-SET that has an independent set of size k iff Φ 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.

G



k = 3

$\Phi = \left(\overline{x_1} \lor x_2 \lor x_3\right) \land \left(x_1 \lor \overline{x_2} \lor x_3\right) \land \left(\overline{x_1} \lor x_2 \lor x_4\right)$

26

3 Satisfiability Reduces to Independent Set

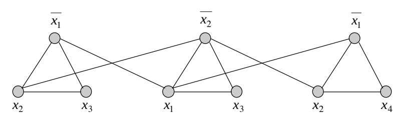
Claim. G contains independent set of size k = $|\Phi|$ iff Φ is satisfiable.

Pf. $\Rightarrow$ Let S be independent set of size k.

- S must contain exactly one vertex in each triangle.
- Set these literals to true. ← 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. ▪

G



k = 3

$\Phi = \left(\overline{x_1} \lor x_2 \lor x_3\right) \land \left(x_1 \lor \overline{x_2} \lor x_3\right) \land \left(\overline{x_1} \lor x_2 \lor x_4\right)$

27

Review

Basic reduction strategies.

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

Transitivity. If X $\le_P$ Y and Y $\le_P$ Z, then X $\le_P$ Z.
Pf idea. Compose the two algorithms.

Ex: 3-SAT $\le_P$ INDEPENDENT-SET $\le_P$ VERTEX-COVER $\le_P$ SET-COVER.

28

Self-Reducibility

Decision problem. Does there exist a vertex cover of size $\le$ k?
Search problem. Find vertex cover of minimum cardinality.

Self-reducibility. Search problem $\le_P$ decision version.

- Applies to all (NP-complete) problems we discuss.
- 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 $\le$ 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

29

Definition of NP

Decision Problems

Decision problem.
- X is a set of strings (a language).
- 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] $p(|s|) = |s|^8$.

31

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

32

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.

33

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 $|t| \le |s|$.

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 × 809

Conclusion. COMPOSITES is in NP.

34

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.

35

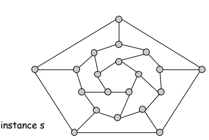
Certifiers and Certificates: Hamiltonian Cycle

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

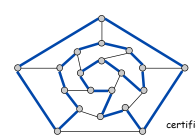
Certificate. A permutation of the n nodes.

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

36

P, NP, EXP

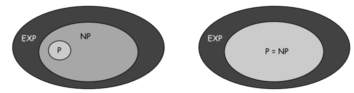
P. Decision problems for which there is a poly-time algorithm.
EXP. Decision problems for which there is an exponential-time algorithm.
NP. Decision problems for which there is a poly-time certifier.

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.
- 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 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.

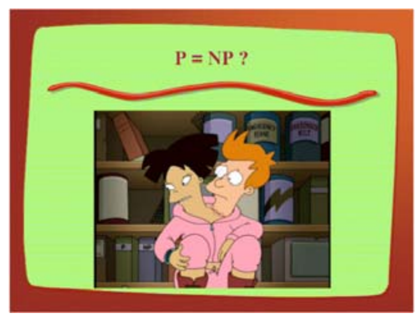


If $P \neq NP$ If $P = NP$

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.



Looking for a Job?

Some writers for the Simpsons and Futurama.
- J. Steward Burns. M.S. in mathematics, Berkeley, 1993.
- David X. Cohen. M.S. in computer science, Berkeley, 1992.
- Al Jean. B.S. in mathematics, Harvard, 1981.
- Ken Keeler. Ph.D. in applied mathematics, Harvard, 1990.
- Jeff Westbrook. Ph.D. in computer science, Princeton, 1989.

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 yes instance of X iff y is a yes instance of Y.

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

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 $\le_p$ and blur distinction

NP-Complete

NP-complete. A problem Y in NP with the property that for every problem X in NP, $X \le_p Y$.

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 $X \le_p Y$, 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?

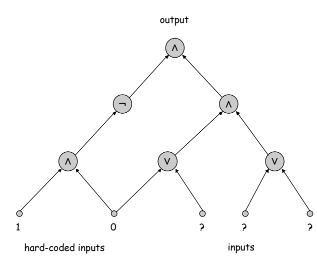
The "First" NP-Complete Problem

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

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]

Pf. (sketch)

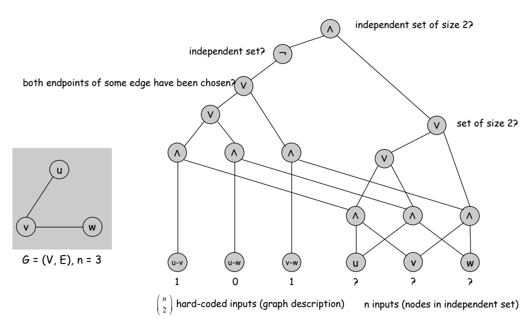
- Any algorithm that takes a fixed number of bits n as input and produces a yes/no answer can be represented by such a circuit. Moreover, if algorithm takes poly-time, then circuit is of poly-size.

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 $C(s, t)$ = 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 $C(s, t)$ = 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 $X \le_P Y$.

Justification. If X is an NP-complete problem, and Y is a problem in NP with the property that $X \le_P Y$ then Y is NP-complete.

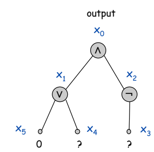
Pf. Let W be any problem in NP. Then $W \underbrace{\le_P}_{\text{by definition of NP-complete}} X \underbrace{\le_P}_{\text{by assumption}} Y$.
- By transitivity, $W \le_P Y$.
- Hence Y is NP-complete. ▪

49

3-SAT is NP-Complete

Theorem. 3-SAT is NP-complete.

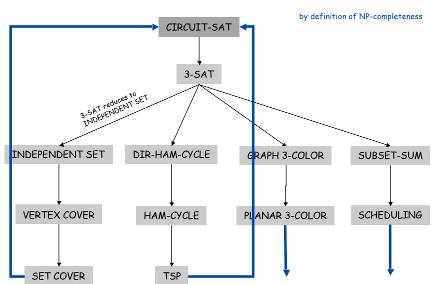
Pf. Suffices to show that CIRCUIT-SAT $\le_P$ 3-SAT since 3-SAT is in NP.
- Let K be any circuit.
- Create a 3-SAT variable x_i for each circuit element i.
- Make circuit compute correct values at each node:
  - $x_2 = \neg\, x_3 \Rightarrow$ add 2 clauses: $x_2 \vee x_3$, $\overline{x_2} \vee \overline{x_3}$
  - $x_1 = x_4 \vee x_5 \Rightarrow$ 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. ▪



50

NP-Completeness

Observation. All problems below are NP-complete and polynomial reduce to one another!



51

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-MATCHING 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.

52

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.

53

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.

54