CSCI 355: Algorithm Design and Analysis
10. Intractability

- *poly-time reductions*
- *P vs. NP*
- *NP-completeness*

Algorithm design patterns and antipatterns

Algorithm design patterns.

- Greedy.
- Divide and conquer.
- Dynamic programming.
- Duality.
- Reductions.
- Local search.
- Randomization.

Algorithm design antipatterns.

- **NP**-completeness. $O(n^k)$ algorithm unlikely.
- **PSPACE**-completeness. $O(n^k)$ certification algorithm unlikely.
- Undecidability. No algorithm possible.

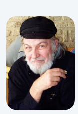
Historical significance: Edmonds' *Paths, Trees, and Flowers*, 1965

2. Digression. An explanation is due on the use of the words "efficient algorithm." First, what I present is a conceptual description of an algorithm and not a particular formalized algorithm or "code."

For practical purposes computational details are vital. However, my purpose is only to show as attractively as I can that there is an efficient algorithm. According to the dictionary, "efficient" means "adequate in operation or performance." This is roughly the meaning I want—in the sense that it is conceivable for maximum matching to have no efficient algorithm. Perhaps a better word is "good."

I am claiming, as a mathematical result, the existence of a *good* algorithm for finding a maximum cardinality matching in a graph.

There is an obvious finite algorithm, but that algorithm increases in difficulty exponentially with the size of the graph. It is by no means obvious whether *or not* there exists an algorithm whose difficulty increases only algebraically with the size of the graph.



Edmonds

Classifying problems according to computational requirements

Q. Which problems will we be able to solve in practice?
A working definition. Those with poly-time algorithms.

Turing machine, word RAM, uniform circuits, …

Theory. Definition is broad and robust.

constants tend to be small, e.g., $3n^2$

Practice. Poly-time algorithms scale to huge problems.

Classifying problems according to computational requirements

Q. Which problems will we be able to solve in practice?
A working definition. Those with poly-time algorithms.

yes	(probably) no
shortest path	longest path
min cut	max cut
2-satisfiability	3-satisfiability
planar 4-colourability	planar 3-colourability
bipartite vertex cover	vertex cover
matching	3d-matching
primality testing	factoring
linear programming	integer linear programming

Classifying problems

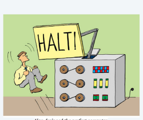
Desiderata. Classify problems according to those that can be solved in polynomial time and those that cannot.

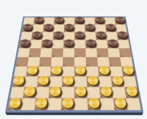
Problems that provably require exponential time.

input size = $c + \log k$

- Given a constant-size program, does it halt in at most k steps?
- Given a board position in an n-by-n generalization of checkers, can black guarantee a win?

using forced capture rule





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

Poly-time reductions

Precise desiderata. Suppose we could solve a problem Y in polynomial time. What other problems could we solve in polynomial time?

Reduction. Problem X is polynomial-time reducible to problem Y if arbitrary instances of problem X can be solved using:

- a polynomial number of standard computational steps, plus
- a polynomial number of calls to an oracle that solves problem Y.

Notation. $X \leq_P Y$.

Note. We pay for the time to write down instances of Y sent to oracle $\Rightarrow$ instances of Y must be of polynomial size.

Common mistake. Confusing $X \leq_P Y$ with $Y \leq_P X$.

Poly-time reductions

Designing algorithms. If $X \leq_P Y$ and Y can be solved in polynomial time, then X can be solved in polynomial time.

Establishing intractability. If $X \leq_P Y$ and X cannot be solved in polynomial time, then Y cannot be solved in polynomial time.

Proving equivalence. If both $X \leq_P Y$ and $Y \leq_P X$, then X can be solved in polynomial time iff Y can be solved in polynomial time; we write $X \equiv_P Y$.

Bottom line. Reductions classify problems according to relative difficulty.

Examples of problems

Satisfiability.

- **SAT**. Given a CNF formula Φ, does it have a satisfying truth assignment?
- **3-SAT**. An instance of SAT where each clause contains exactly 3 literals (and each literal corresponds to a different variable).

Packing and covering.

- **INDEPENDENT-SET**. Given a graph $G = (V, E)$ and an integer k, is there a subset of k (or more) vertices such that no two are adjacent?
- **VERTEX-COVER**. Given a graph $G = (V, E)$ and an integer k, is there a subset of k (or fewer) vertices such that each edge is incident to at least one vertex in the subset?
- **SET-COVER**. Given a set U of elements, a collection S of subsets of U, and an integer k, are there $\leq k$ of these subsets whose union is equal to U?

Examples of problems

Sequencing.

- **HAMILTON-CYCLE.** Given an undirected graph $G = (V, E)$, does there exist a cycle Γ that visits every vertex exactly once?
- **DIRECTED-HAMILTON-CYCLE.** Given a directed graph $G = (V, E)$, does there exist a directed cycle Γ that visits every vertex exactly once?

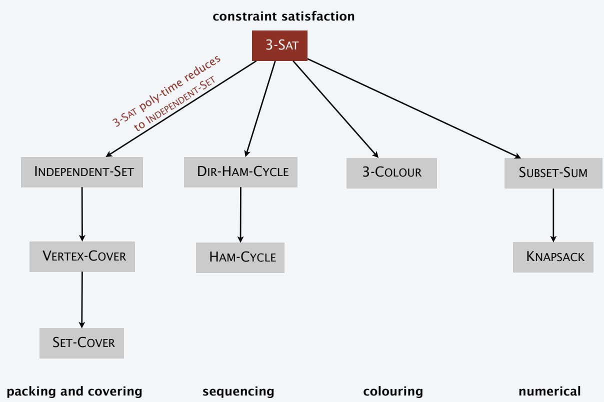
Colouring.

- **3-COLOUR.** Given an undirected graph G, can the vertices be coloured black, white, and blue so that no adjacent vertices have the same colour?

Numerical.

- **SUBSET-SUM.** Given n natural numbers $w_1, \ldots, w_n$ and an integer W, is there a subset that adds up to exactly W?
- **KNAPSACK.** Given $2n$ natural numbers $w_1, \ldots, w_n, v_1, \ldots, v_n$ and an integer W, is there a subset that maximizes v_i while adding up all values w_i to exactly W?

Tree of poly-time reductions between problems



Karp's *Reducibility Among Combinatorial Problems*, 1972

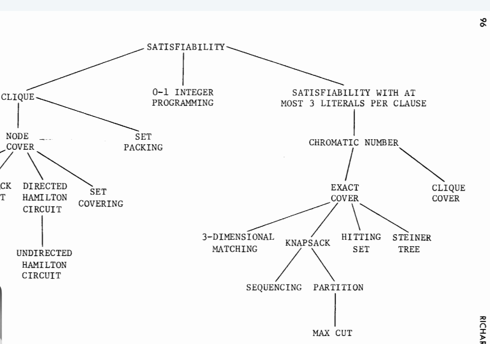


FIGURE 1 - Complete Problems

Karp
1985 Turing Award

CSCI 355: ALGORITHM DESIGN AND ANALYSIS
10. INTRACTABILITY

- *poly-time reductions*
- *P vs. NP*
- *NP-completeness*

The class P

Decision problems.

- A problem X is a set of strings.
- An instance s of a problem is one string.
- An algorithm A solves problem X: $A(s) = \begin{cases} yes & \text{if } s \in X \\ no & \text{if } s \notin X \end{cases}$

Def. Algorithm A runs in polynomial time if, for every string s, $A(s)$ terminates in $\leq \underbrace{p(|s|)}_{\text{length of } s}$ "steps," where $p(\cdot)$ is some polynomial function.

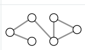
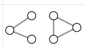
Def. P = set of decision problems for which there exists a poly-time algorithm.

on a deterministic Turing machine

problem PRIMES:	$\{ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, \ldots \}$
instance s:	592335744548702854681
algorithm:	Agrawal–Kayal–Saxena (2002)

Some problems in P

P. Set of decision problems for which there exists a poly-time algorithm.

problem	description	poly-time algorithm	yes	no
MULTIPLE	Is x a multiple of y ?	grade-school division	51, 17	51, 16
REL-PRIME	Are x and y relatively prime ?	Euclid's algorithm	34, 39	34, 51
PRIMES	Is x prime ?	Agrawal–Kayal–Saxena	53	51
EDIT-DISTANCE	Is the edit distance between x and y less than 5 ?	Needleman–Wunsch	niether neither	acgggt ttttta
L-SOLVE	Is there a vector x that satisfies $Ax = b$?	Gauss–Edmonds elimination	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
U-CONN	Is an undirected graph G connected?	depth-first search		

The class NP

Def. An 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"

Def. **NP** = set of decision problems for which there exists a poly-time certifier.

- $C(s, t)$ is a poly-time algorithm.
- Certificate t is of polynomial size: $|t| \leq p(|s|)$ for some polynomial $p(\cdot)$.

problem COMPOSITES:	$\{ 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, \ldots \}$
instance s:	437669
certificate t:	541 ⟵ 437,669 = 541 × 809
certifier C(s, t):	grade-school division

Certifiers and certificates: satisfiability

SAT. Given a CNF formula Φ, does it have a satisfying truth assignment?
3-SAT. SAT where each clause contains exactly 3 literals.

Certificate. An assignment of truth values to the Boolean variables.

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

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

certificate t $x_1 = true,\ x_2 = true,\ x_3 = false,\ x_4 = false$

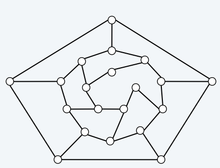
Conclusions. SAT $\in$ **NP**, 3-SAT $\in$ **NP**.

Certifiers and certificates: Hamiltonian path

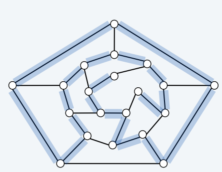
HAMILTON-PATH. Given an undirected graph $G = (V, E)$, does there exist a simple path P that visits every vertex?

Certificate. A permutation π of the n vertices.

Certifier. Checks that π contains each vertex in V exactly once, and that G contains an edge between each pair of adjacent vertices.



instance s

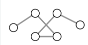
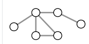


certificate t

Conclusion. HAMILTON-PATH $\in$ **NP**.

Some problems in NP

NP. Set of decision problems for which there exists a poly-time certifier.

problem	description	poly-time algorithm	yes	no
L-SOLVE	Is there a vector x that satisfies $Ax = b$?	Gauss–Edmonds elimination	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
COMPOSITES	Is x composite ?	Agrawal–Kayal–Saxena	51	53
FACTOR	Does x have a nontrivial factor less than y ?	???	(56159, 50)	(55687, 50)
SAT	Given a CNF formula, does it have a satisfying truth assignment?	???	$\neg x_1 \lor x_2 \lor \neg x_3$ $x_1 \lor \neg x_2 \lor x_3$ $\neg x_1 \lor \neg x_2 \lor x_3$	$\neg x_2$ $x_1 \lor x_2$ $\neg x_1 \lor x_2$
HAMILTON-PATH	Is there a simple path between u and v that visits every vertex?	???		

The classes P, NP, and EXP

P. Set of decision problems for which there exists a poly-time algorithm.
NP. Set of decision problems for which there exists a poly-time certifier.
EXP. Set of decision problems for which there exists an exp-time algorithm.

Proposition. **P** $\subseteq$ **NP**.
Pf. Consider any problem $X \in$ **P**.
- By definition, there exists a poly-time algorithm $A(s)$ that solves X.
- Certificate is $t = \varepsilon$, certifier is $C(s, t) = A(s)$. ▪

Proposition. **NP** $\subseteq$ **EXP**.
Pf. Consider any problem $X \in$ **NP**.
- By definition, there exists a poly-time certifier $C(s, t)$ for X where a certificate t satisfies $|t| \le p(|s|)$ for some polynomial $p(\cdot)$.
- To solve the instance s, run $C(s, t)$ on all strings t with $|t| \le p(|s|)$.
- Return *yes* iff $C(s, t)$ returns *yes* for any of these potential certificates. ▪

Fact. **P** $\neq$ **EXP** $\Rightarrow$ either **P** $\neq$ **NP**, or **NP** $\neq$ **EXP**, or both.

The big question: P vs. NP

Q. How do we 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?
Conjecture. There exists $\underbrace{\text{no poly-time algorithm}}_{\text{“intractable”}}$ for 3-SAT.

The big question: P vs. NP

Does P = NP? [Cook, Levin, ...]
Is the decision problem as easy as the certification problem?

If yes... Efficient algorithms exist for 3-SAT, TSP, VERTEX-COVER, FACTOR, ...
If no... No efficient algorithms are possible for 3-SAT, TSP, VERTEX-COVER, ...

Consensus opinion. Probably no.

CSCI 355: ALGORITHM DESIGN AND ANALYSIS
10. INTRACTABILITY

- poly-time reductions
- P vs. NP
- *NP-completeness*

NP-completeness

NP-completeness. A problem $Y \in$ **NP** is NP-complete if it has the property that for every problem $X \in$ **NP**, $X \leq_P Y$.

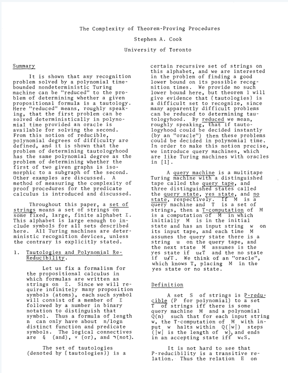
Proposition. Suppose $Y \in$ **NP**-complete. Then $Y \in$ **P** iff **P** = **NP**.

Pf.
[⇐] If **P** = **NP**, then $Y \in$ **P** because $Y \in$ **NP**.
[⇒] Suppose $Y \in$ **P**.
- Consider any problem $X \in$ **NP**. Since $X \leq_P Y$, we have $X \in$ **P**.
- This implies **NP** ⊆ **P**.
- We already know **P** ⊆ **NP**. Thus **P** = **NP**. ▪

Fundamental question. Are there any "natural" **NP**-complete problems?

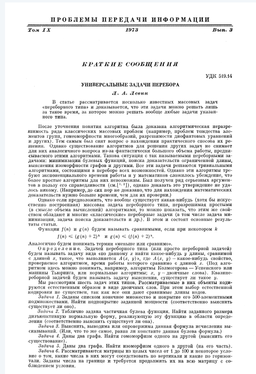
The first NP-complete problem

Theorem. [Cook 1971, Levin 1973] SAT $\in$ **NP**-complete.


The Complexity of Theorem-Proving Procedures

Stephen A. Cook

University of Toronto


ПРОБЛЕМЫ ПЕРЕДАЧИ ИНФОРМАЦИИ

Том IX 1973 Вып. 3

КРАТКИЕ СООБЩЕНИЯ

УДК 519.14

УНИВЕРСАЛЬНЫЕ ЗАДАЧИ ПЕРЕБОРА

Л. А. Левин

Establishing NP-completeness

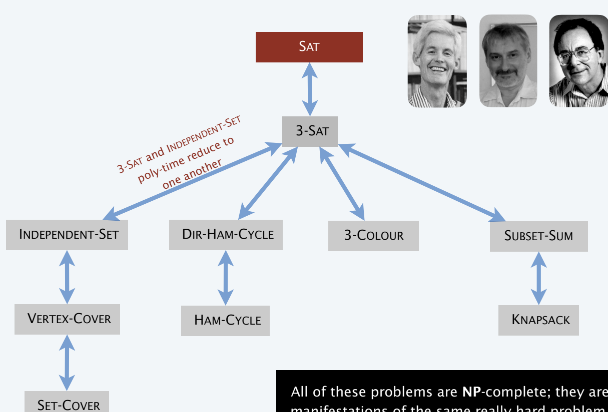
Remark. Once we establish the first "natural" **NP**-complete problem, the others fall like dominoes.

Recipe. To prove that $Y \in$ **NP**-complete:
- Step 1. Show that $Y \in$ **NP**.
- Step 2. Choose an **NP**-complete problem X.
- Step 3. Prove that $X \leq_P Y$.

Proposition. If $Y \in$ **NP**, $X \in$ **NP**-complete, and $X \leq_P Y$, then $Y \in$ **NP**-complete.

Pf. Consider any problem $W \in$ **NP**. Then, both $\underbrace{W \leq_P X}_{\text{by definition of NP-completeness}}$ and $\underbrace{X \leq_P Y}_{\text{by assumption}}$.
- By transitivity, $W \leq_P Y$.
- Hence $Y \in$ **NP**-complete. ▪

Implications of Karp + Cook–Levin



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

Some NP-complete problems

Basic classes of NP-complete problems and examples.

- Packing/covering problems: SET-COVER, VERTEX-COVER, INDEPENDENT-SET.
- Constraint satisfaction problems: SAT, 3-SAT, CIRCUIT-SAT.
- Sequencing problems: HAMILTON-CYCLE, TSP.
- Partitioning problems: 3-COLOUR, 3D-MATCHING.
- Numerical problems: SUBSET-SUM, KNAPSACK.

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

"NP-intermediate" problems? FACTOR, DISCRETE-LOG, GRAPH-ISOMORPHISM, …

Theorem. [Ladner 1975] Unless **P** = **NP**, there exist problems in **NP** that are neither in **P** nor **NP**-complete.

On the Structure of Polynomial Time Reducibility

RICHARD E. LADNER

University of Washington, Seattle, Washington

More hard computational problems

M. R. Garey and D. S. Johnson. *Computers and Intractability.*

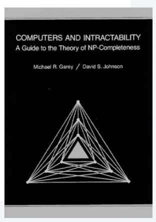
- Appendix includes over 300 **NP**-complete problems.
- Most cited reference in computer science literature.

Most Cited Computer Science Citations

This list is generated from documents in the CiteSeer^x database as of January 17, 2013. This list is automatically generated and may contain errors. The list is generated in batch mode and citation counts may differ from those currently in the CiteSeer^x database, since the database is continuously updated.

All Years | 1990 | 1991 | 1992 | 1993 | 1994 | 1995 | 1996 | 1997 | 1998 | 1999 | 2000 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2011 | 2012 | 2013

1. M R Garey, D S Johnson
   Computers and Intractability. A Guide to the Theory of NP-Completeness 1979
   8665
2. T Cormen, C E Leiserson, R Rivest
   Introduction to Algorithms 1990
   7210
3. V N Vapnik
   The nature of statistical learning theory 1998
   6580
4. A P Dempster, N M Laird, D B Rubin
   Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society, 1977
   6082
5. T Cover, J Thomas
   Elements of Information Theory 1991
   6075
6. D E Goldberg
   Genetic Algorithms in Search, Optimization, and Machine Learning, 1989
   5998
7. J Pearl
   Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference 1988
   5582
8. E Gamma, R Helm, R Johnson, J Vlissides
   Design Patterns: Elements of Reusable Object-Oriented Software 1995
   4614
9. C E Shannon
   A mathematical theory of communication Bell Syst. Tech. J, 1948
   4118
10. J R Quinlan
    C4.5: Programs for Machine Learning 1993
    4018



More hard computational problems

Aerospace engineering. Optimal mesh partitioning for finite elements.
Biology. Phylogeny reconstruction.
Chemical engineering. Heat exchanger network synthesis.
Chemistry. Protein folding.
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. Minimum risk portfolio of given return.
Game theory. Nash equilibrium that maximizes social welfare.
Mathematics. Given integer $a_1, \ldots, a_n$, compute $\int_0^{2\pi} \cos(a_1\theta) \times \cos(a_2\theta) \times \cdots \times \cos(a_n\theta)\, d\theta$
Mechanical engineering. Structure of turbulence in sheared flows.
Medicine. Reconstructing 3d shape from biplane angiocardiogram.
Operations research. Traveling salesperson problem.
Physics. Partition function of 3d Ising model.
Politics. Shapley–Shubik voting power.
Recreation. Versions of Sudoku, Checkers, Minesweeper, Tetris, Rubik's Cube.
Statistics. Optimal experimental design.

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 (more than "compiler", "OS", "database").
- Broad applicability and classification power.

NP-completeness can guide scientific inquiry.

- 1926: Ising introduces a simple model for phase transitions.
- 1944: Onsager finds a closed-form solution to 2D-ISING.
- 19xx: Top minds seek a solution to 3D-ISING. ← a holy grail of statistical mechanics
- 2000: Istrail proves 3D-ISING $\in$ **NP**-complete.

the search for a closed formula appears doomed

Ising Onsager Istrail