CSE202
Design and Analysis of Algorithms

Week 13 — P vs NP

Model for Polynomial Time Computation

Def 1. A function from $\{0,1\}^\star$ to $\{0,1\}^\star$ is computable in polynomial time if there exists a k and a program computing the output in time $O(n^k)$ for all input of size n.

Input: a word in $\{0,1\}^n$

Or any encoding of polynomial size

Computation: a Python program

Python, C++, Java, Turing machines,… can simulate each other in polynomial time

Output: a word in $\{0,1\}^m$

Def 2. Decision problem: $m = 1$

Necessarily $m = O(n^k)$

Def 3. P is the class of all decision problems computable in polynomial time.

I. The Class NP

Verifier and Certificate

Decision problem: $A : w \in \{0,1\}^\star \to \{0,1\}$

Verifier: a program $V : \{0,1\}^\star \times \{0,1\}^\star \to \{0,1\}$ s.t.

$$\forall w \in \{0,1\}^\star, \quad A(w) = 1 \iff \exists C \in \{0,1\}^\star, V(w, C) = 1.$$

certificate,
witness,
proof

Note the asymmetry between 0 & 1.

Def. NP is the class of problems A s.t. there exists a verifier V in P with $|C| = \text{poly}(|w|)$.

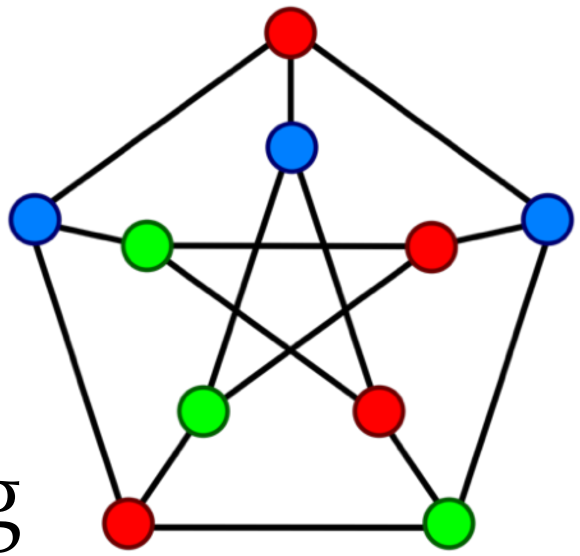
Prop. P $\subseteq$ NP.

Example 1: k-Coloring

Problem k-Coloring:

Input: a graph

Output: yes iff there exists a proper coloring with $\leq k$ colors (no unicolor edge).



$\boxed{k\text{-Coloring} \in \mathrm{NP}}$

Proof: a proper coloring is easy to check.

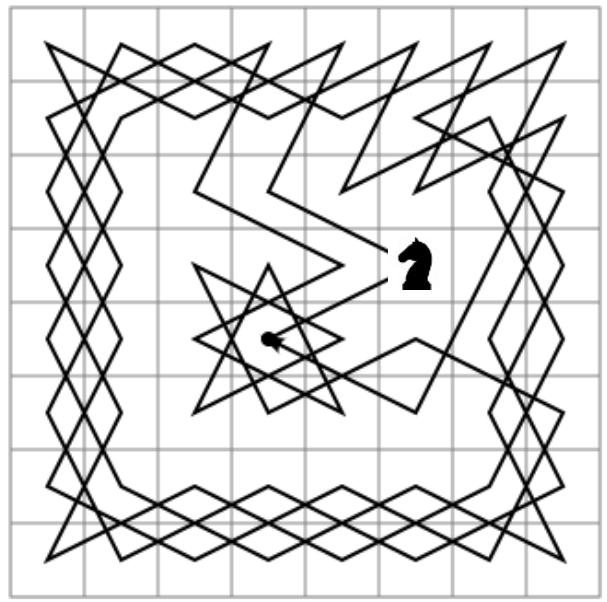
No polynomial algorithm known to find a coloring for $k \geq 3$, or to check non-k-Coloring.

Example 2: Hamiltonian Cycle

Problem Hamiltonian Cycle:

Input: a graph

Output: yes iff there exists a circuit visiting each vertex exactly once.

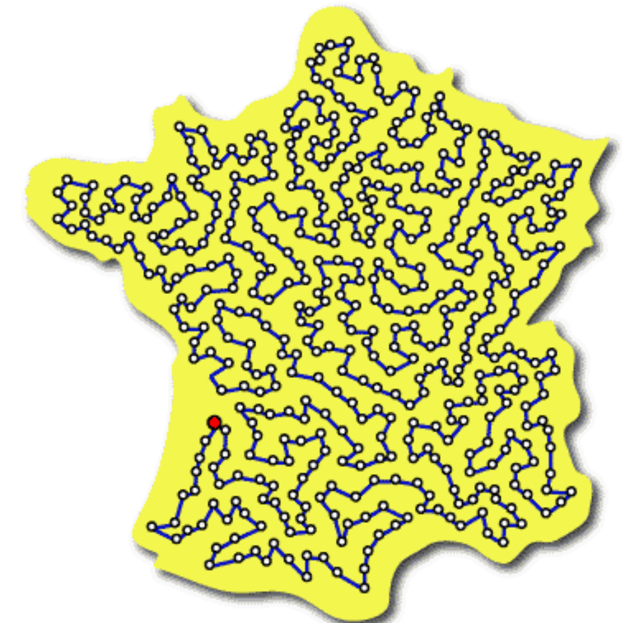


$\boxed{\text{Hamiltonian Cycle} \in \text{NP}}$

Proof: a circuit is easy to check.

No polynomial algorithm known to find a Hamiltonian cycle, or to check that none exists.

Example 3: Traveling Salesman Problem



Problem TSP:

Input: $n \times n$ matrix M of positive integers,
k in $\mathbb{N}$

Output: yes iff there exists a cyclic permutation σ s.t. $\sum_{1 \le i \le n} M_{i,\sigma(i)} \le k$.

$\boxed{\text{TSP} \in \text{NP}}$

Proof: a cycle is easy to check.

No polynomial algorithm known to find a TS tour, or to check that none exists.

Example 4: Subset-Sum

[14, 18, 15, 8, 10, 14, 12, 9, 13, 16]

14+15+10+14=53

Problem SubsetSum:

Input: $L = (x_1, \ldots, x_\ell) \in \mathbb{N}^\ell$,
k in $\mathbb{N}$

Output: yes iff there exists a subset $A \subset \{1, \ldots, \ell\}$ s.t.

$$\sum_{i \in A} x_i = k\,.$$

$\boxed{\text{SubsetSum} \in \text{NP}}$

Proof: a solution is easy to check.

No polynomial algorithm known to find a solution, or to check that none exists.

Example 5: Prime

Eratosthenes' sieve has exponential complexity

Problem Prime:

Input: $n \in \mathbb{N}$

Output: yes iff n is prime

$\boxed{\text{Prime} \in \text{NP}}$

Lehmer's theorem: n is prime iff $\exists a \in \{2,\ldots,n-1\}$ s.t.

$a^{n-1} \equiv 1 \bmod n$ and

$\forall q$ prime factor of $n-1$, $\quad a^{(n-1)/q} \not\equiv 1 \bmod n$.

$\longrightarrow$ recursive certificate

(proof polynomial verifier not obvious).

Prime $\in$ P proved in 2002

NP is at most Exponential Time

Def. NP is the class of problems A s.t. there exists a verifier V in P with $|C| = \text{poly}(|w|)$.

Algorithm:

For $m = 1,2,\ldots$

For all C of length m

If $V(w, C) = 1$ return YES

proves

$\boxed{\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{TIME}(2^{\text{poly}(n)})\,.}$

No better upper bound known.

The `N' in NP

N stands for non-deterministic

The program can flip coins to guess the certificate.

NP is about:

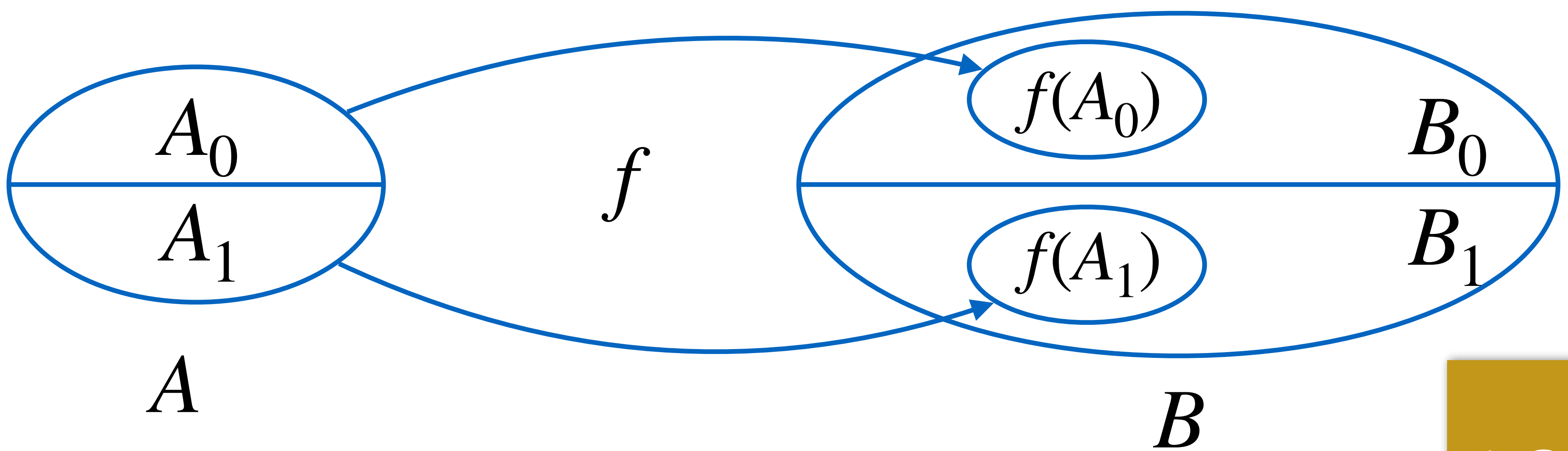
. the power of nondeterminism;
. the difference of complexity between finding and checking;
. and this applies to mathematical proofs as special cases.

At its root, our belief that P≠NP is about the nature of intelligence itself. Moore & Mertens (2011)

II. NP-Complete Problems

Reductions Comparing Hardness

Def. A & B two decisions problems.
A reduces to B (denoted $A \leq B$), if there exists f computable in polynomial time s.t. $A(x) = 1 \Leftrightarrow B(f(x)) = 1$.



Exercise:
k-Coloring $\leq (k+1)$-Coloring

$$A \leq B \text{ and } B \in \mathrm{P} \Rightarrow A \in \mathrm{P}$$

Def: B in **NP** is NP-complete if for all A in NP, $A \leq B$.

If one **NP**-complete problem is in **P**, they all are!

$$A \leq B, B \in \mathrm{NP} \text{ and } A \text{ NP-complete} \Rightarrow B \text{ NP-complete}$$

3-SAT is NP-complete

Ex.: $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$

Clause: disjunction ($\vee$) of variables or their negations.

Recall: Conjunctive normal form: conjunction ($\wedge$) of clauses.

k-SAT: every clause involves k of the n variables.

Thm. (Cook-Levin, early 70's) 3-SAT is NP-complete.

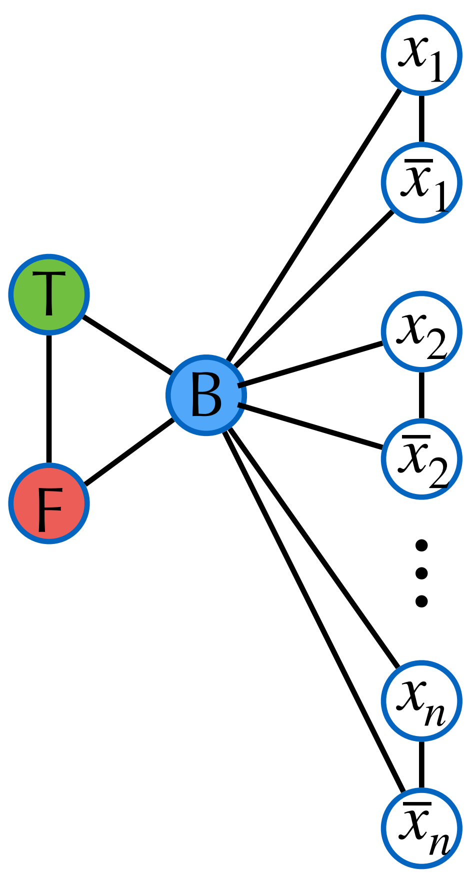
NP-complete problems exist!

Proof requires a more rigorous/precise definition of algorithm, e.g., Turing machine

3-SAT ≤ 3-Coloring

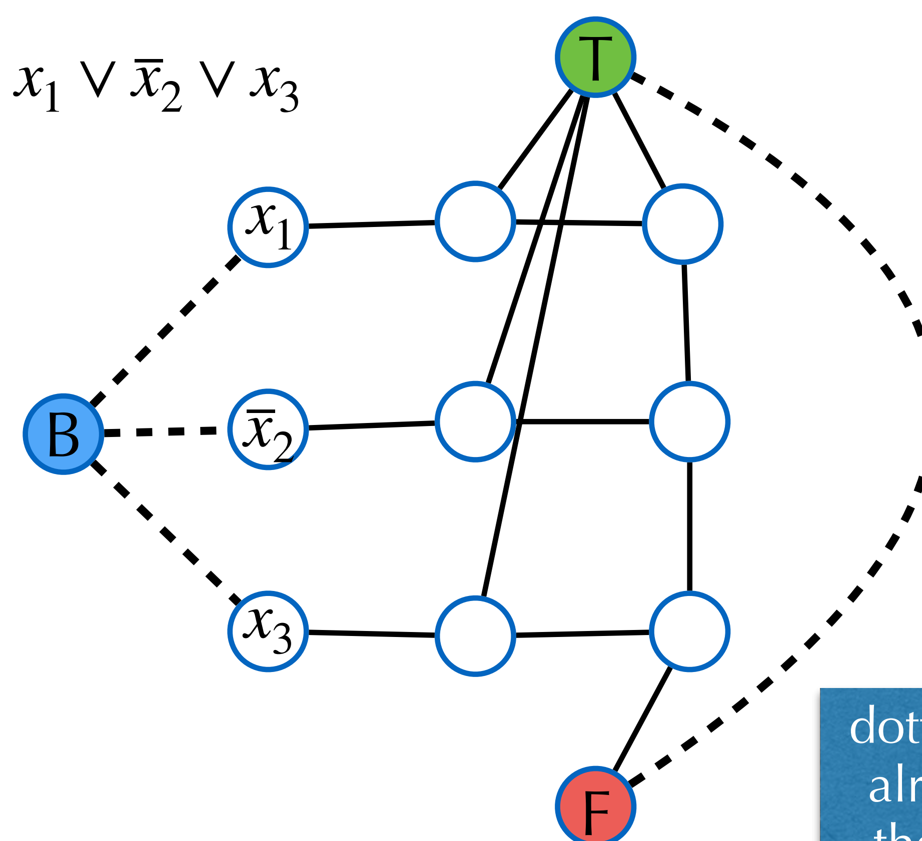
Proof. Encode a 3-SAT problem into a graph coloring problem.

1. encode the variables



2. encode the clauses (in the same graph)

Ex: $x_1 \vee \bar{x}_2 \vee x_3$



dotted lines already in the graph

3-Coloring is NP-complete

3-SAT $\leq$ SubsetSum

n variables $x_1, \ldots, x_n$, $\quad k$ clauses $C_1, \ldots, C_k$

Ex.: $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$

1. To each x_i, associate two integers t_i, f_i of $n + k$ decimal digits:

   $t_{ii} = f_{ii} = 1, \quad t_{i,n+j} = 1$ if $x_i \in C_j$,

   $f_{i,n+j} = 1$ if $\bar{x}_i \in C_j$.

2. To each C_j, associate 2 integers s_j, s_j':

   $s_{j,k+j} = s'_{j,k+j} = 1.$

3. All other digits are 0.

4. Target sum is $\underbrace{11\cdots1}_{n}\underbrace{33\cdots3}_{k}$.

	x_1	x_2	x_3	C_1	C_2	C_3	C_4
t_1	1	0	0	1	0	0	1
f_1	1	0	0	0	1	1	0
t_2	0	1	0	1	0	1	0
f_2	0	1	0	0	1	0	1
t_3	0	0	1	1	1	0	1
f_3	0	0	1	0	0	1	0
s_1	0	0	0	1	0	0	0
s_1'	0	0	0	1	0	0	0
s_2	0	0	0	0	1	0	0
s_2'	0	0	0	0	1	0	0
s_3	0	0	0	0	0	1	0
s_3'	0	0	0	0	0	1	0
s_4	0	0	0	0	0	0	1
s_4'	0	0	0	0	0	0	1

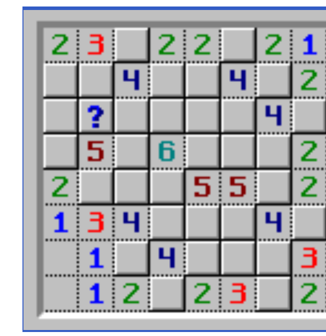
$\boxed{\text{SubsetSum is NP-complete}}$

Other NP-Complete Examples

All the examples of Part I that are not in P;

k-SAT for $k \geq 3$; SAT;

Shortest total path length spanning tree;



Quadratic diophantine equations ($ax^2 + by = c$?);

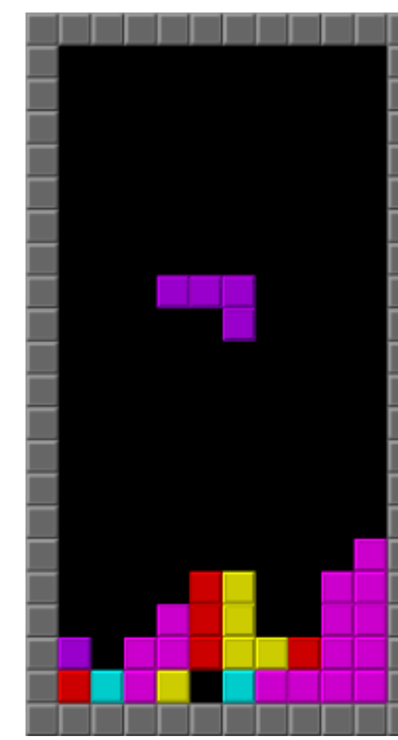
Integer linear programming;

Other graph problems: VertexCover, Clique,…;

Minesweeper consistency;

Tetris;

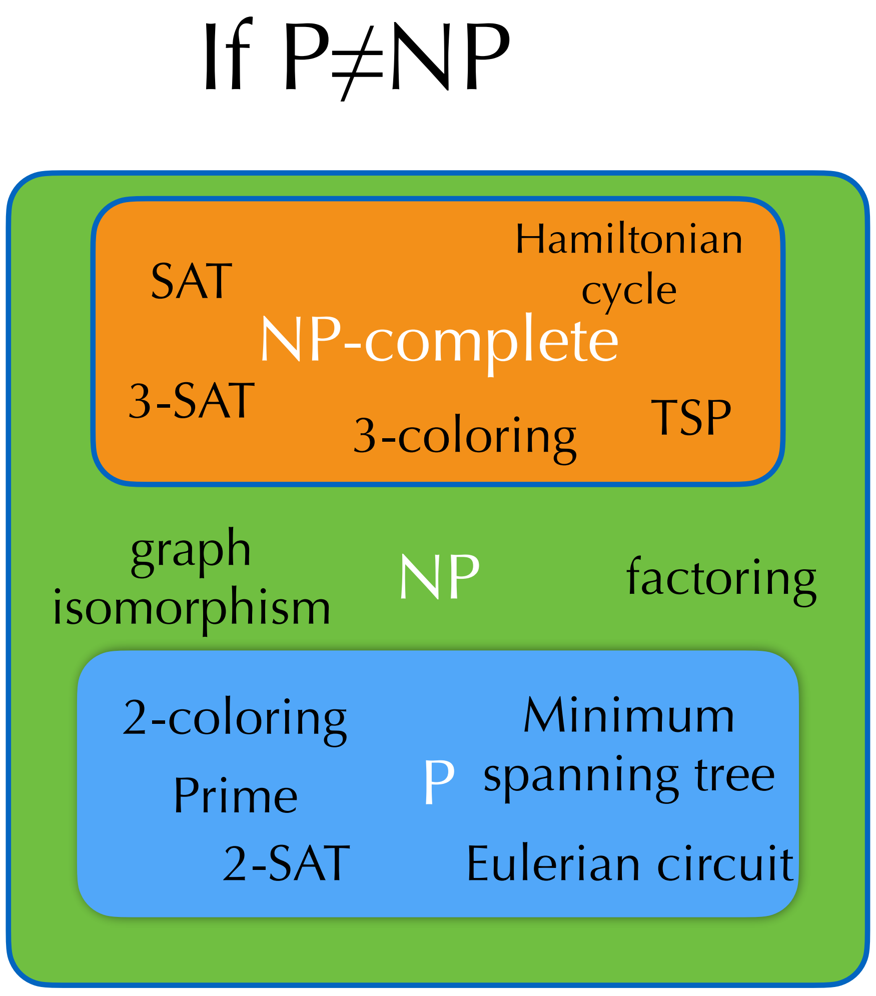
Other games…

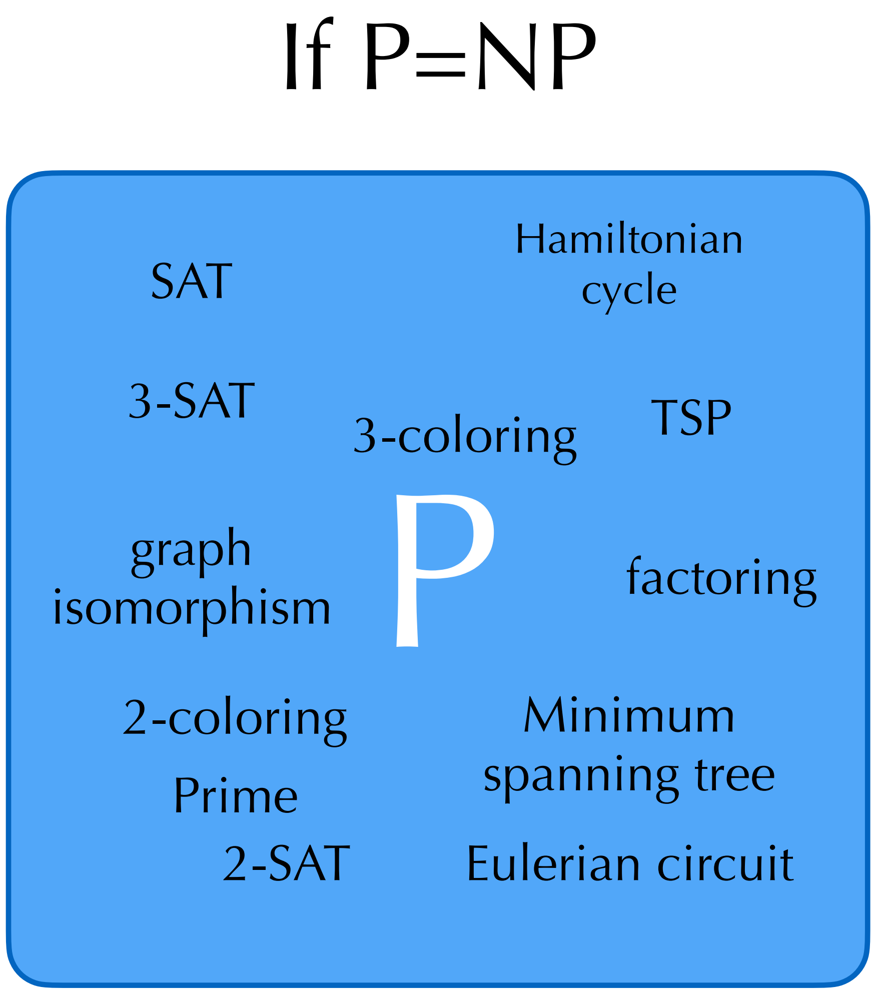


If your problem is in this list, simple solutions will probably be inefficient.

More than 300 entries in Garey & Johnson (1979).

P vs. NP





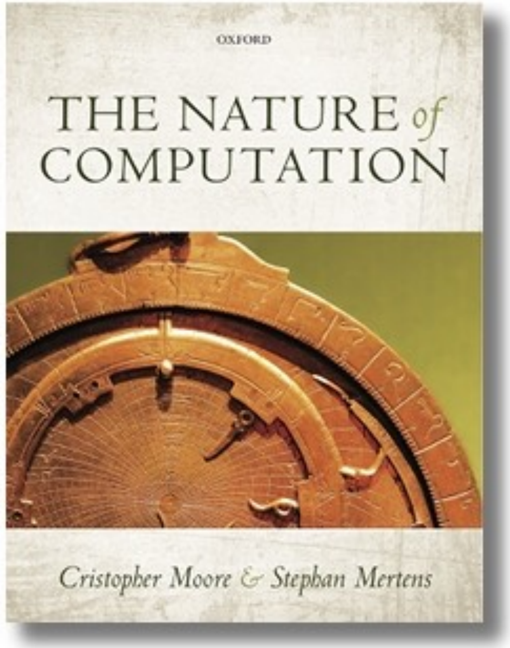
If P=NP, then the world would be a profoundly different place than we usually assume it to be. Scott Aaronson.

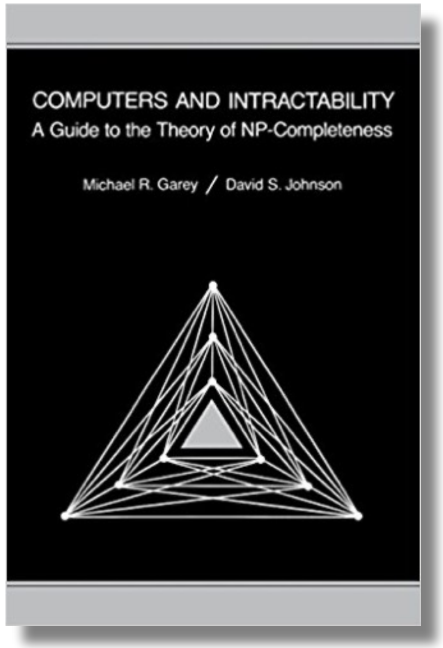
I've come to believe that P = NP. Donald Knuth.

References for this lecture

The slides are designed to be self-contained.

They were prepared using the following
books that I recommend if you want to learn more:





this one was not used
but gives a nice introduction
through games

Next

Assignment: yet another NP-complete problem

Next tutorial: faster Traveling Salesman

Next lecture: Approximation Algorithms for NP-hard problems

Next week: 

Feedback

Moodle for the slides, tutorials and exercises.

Questions or comments: Bruno.Salvy@inria.fr