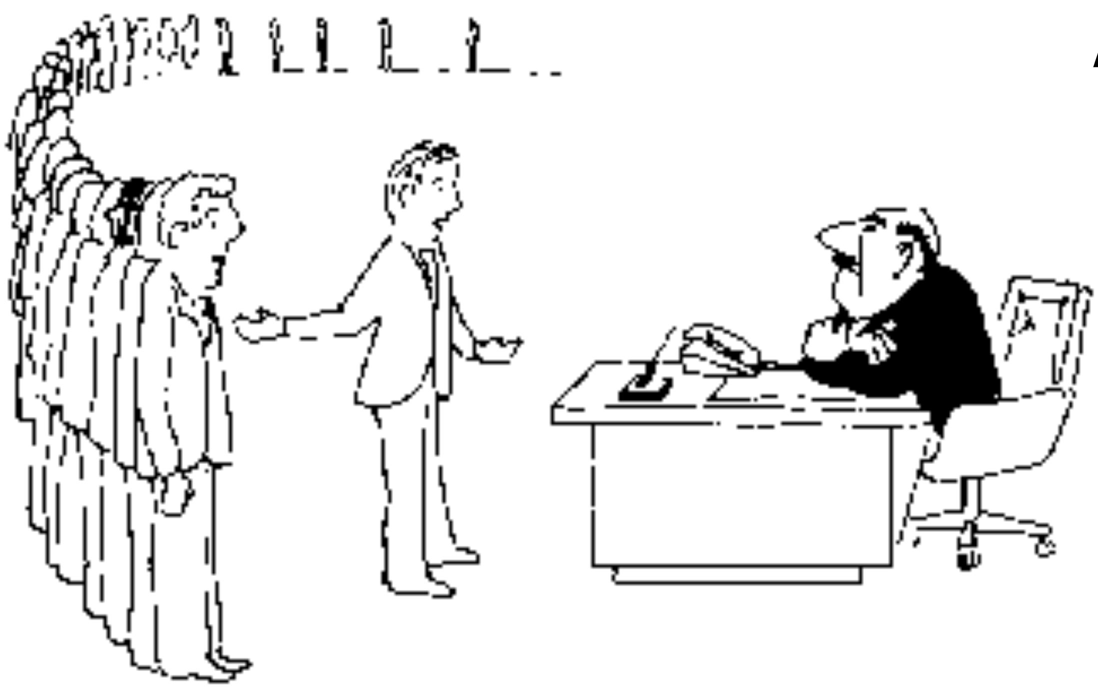
15-251
Great Theoretical Ideas in Computer Science

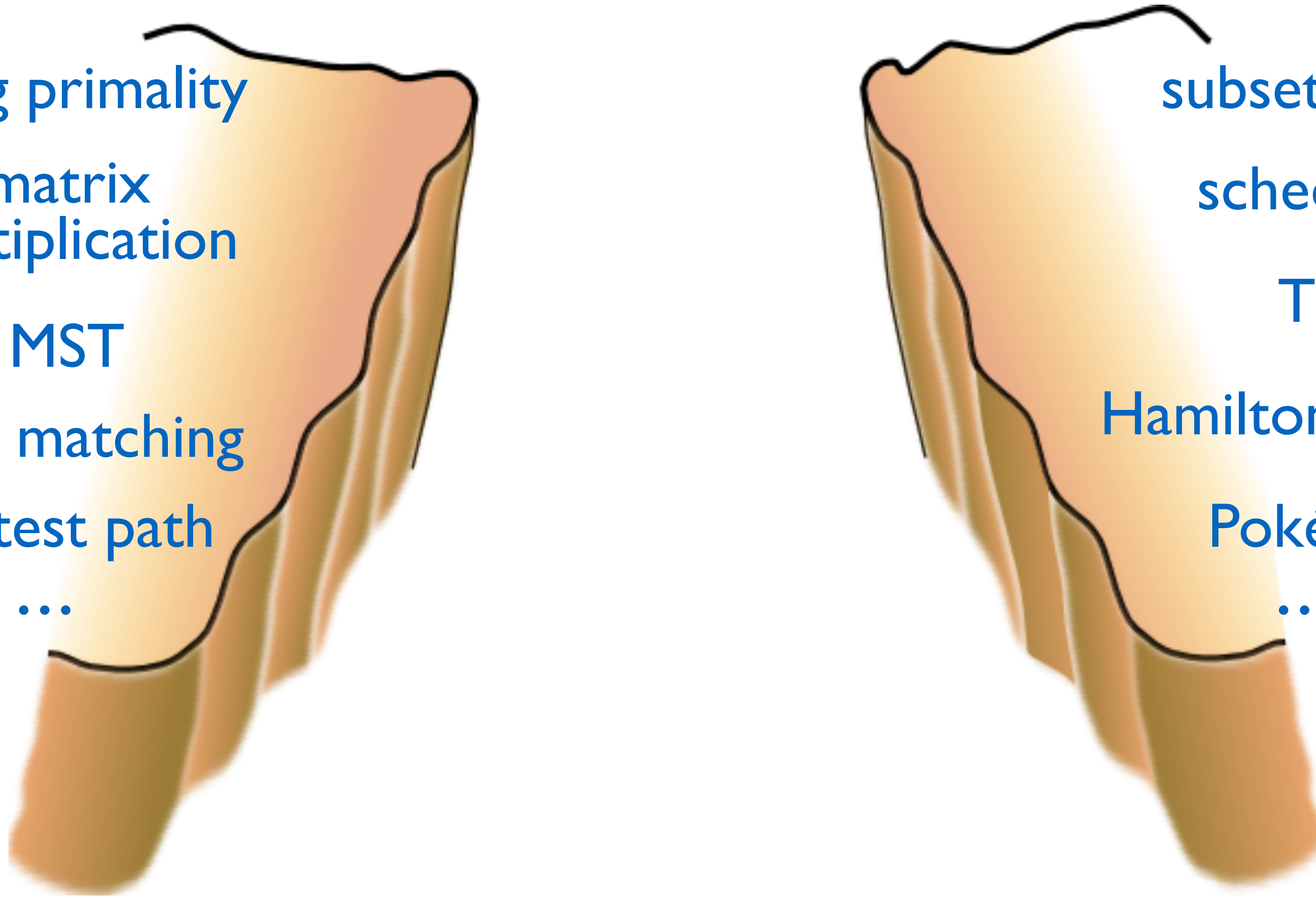
Lecture 17:
NP and NP-completeness 1

March 21st, 2017



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

The big chasm between poly-time and exp-time.



poly-time solvable

best we can say:
exp-time solvable

What is P ?

P

The set of <u>languages</u> that can be decided in $O(n^k)$ steps for some constant k.

"complexity class"

The theoretical divide between **efficient** and **inefficient**:

$L \in$ **P** $\longrightarrow$ **efficiently** solvable (tractable).

$L \notin$ **P** $\longrightarrow$ **not** efficiently solvable.

Exponential running time examples

Subset Sum Problem

Given a list of integers, determine if there is a subset of the integers that sum to 0.

4	-3	-2	7	99	5	1

Exponential running time examples

Subset Sum Problem

Given a list of integers, determine if there is a subset of the integers that sum to 0.

4	-3	-2	7	99	5	1

Exhaustive Search (Brute Force Search):

> Try every possible subset and see if it sums to 0.

subsets is 2^n $\implies$ running time at least 2^n

Note: checking if a given subset sums to 0 is **easy**.

Exponential running time examples

Theorem Proving Problem
(informal description)

Given a mathematical proposition P and an integer k, determine if P has a proof of length at most k.

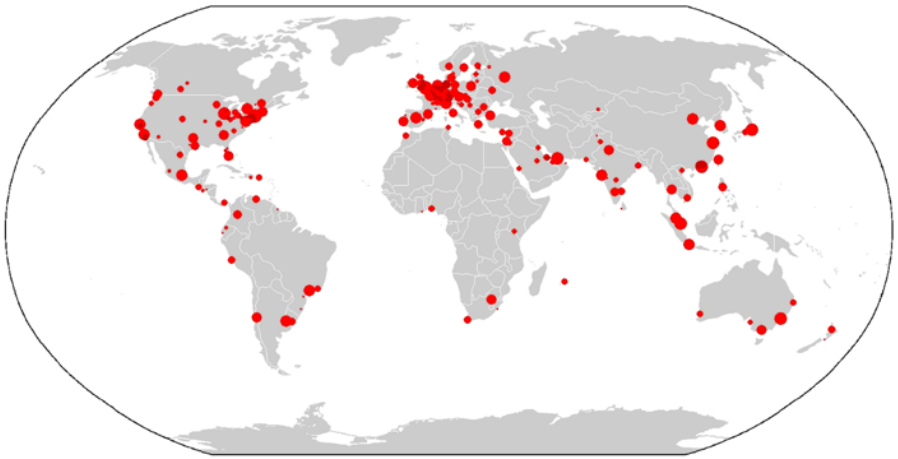
Exhaustive Search (Brute Force Search):

> Try every possible "proof" of length at most k, and check if it corresponds to a valid proof.

Note: checking if a given proof is correct is **easy**.

Exponential running time examples

Traveling Salesperson Problem (TSP)



Is there an order in which you can visit the cities so that ticket cost is < $50000?

Exhaustive Search (Brute Force Search):

> Try every possible order and compute the cost.

Note: checking if a given solution has the desired cost is **easy**.

Exponential running time examples

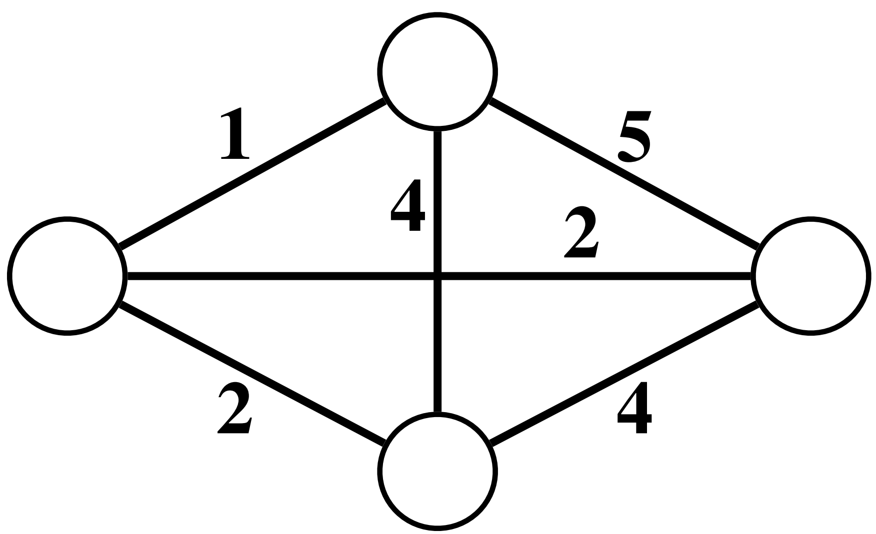
Traveling Salesperson Problem (TSP)

<u>Input:</u>

A graph $G = (V, E)$, edge weights w_e (non-negative, integral) and target t.

<u>Output:</u>

Yes, iff there is a cycle of cost at most t that visits every vertex exactly once.



Exponential running time examples

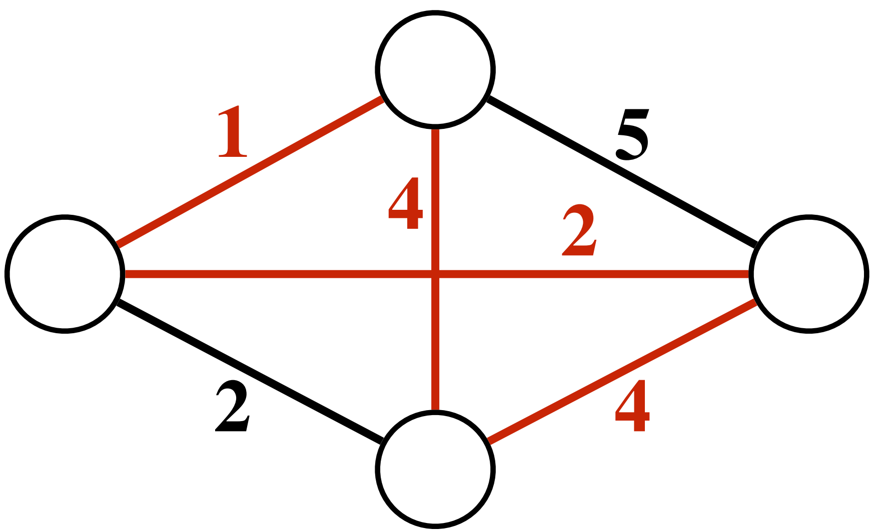
Traveling Salesperson Problem (TSP)

Input:

A graph $G = (V, E)$, edge weights w_e (non-negative, integral) and target t.

Output:

Yes, iff there is a cycle of cost at most t that visits every vertex exactly once.



Exponential running time examples

Satisfiability Problem (SAT)

Input: A Boolean propositional formula.

e.g. $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_3 \wedge x_4) \vee x_3$

Output: Yes iff there is an assignment to the variables that makes the formula True.

Exhaustive Search (Brute Force Search):

> Try every possible truth assignment to the input variables. Evaluate the formula to see the output.

Note: checking if a given truth assignment makes the formula True is **easy**.

Exponential running time examples

Circuit Satisfiability Problem (Circuit-SAT)

Input: A Boolean circuit.

Output: Yes iff there is an assignment to the input gates that makes the circuit output 1.

Exhaustive Search (Brute Force Search):

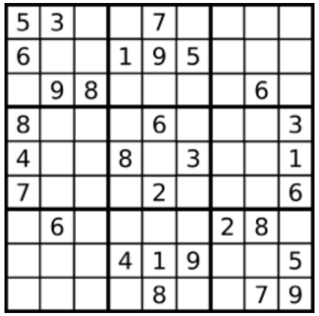
> Try every possible truth assignment to the input gates. Evaluate the circuit to see the output.

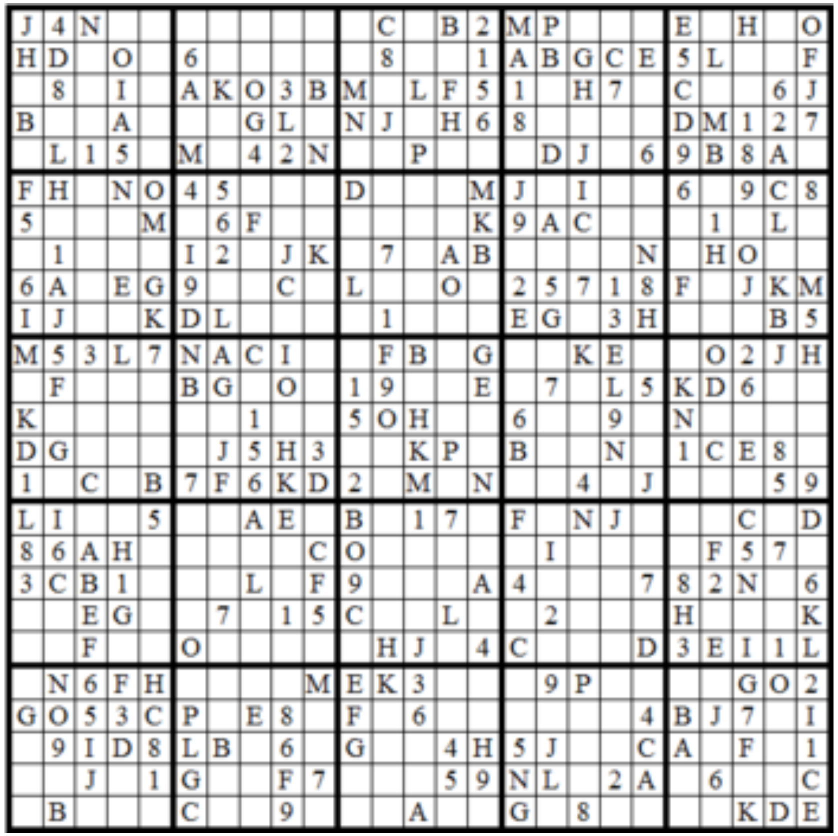
Note: checking if a given assignment makes the circuit output 1 is **easy**.

Exponential running time examples

Sudoku Problem

Given a partially filled *n* by *n* sudoku board, determine if there is a solution.





Exponential running time examples

Sudoku Problem

Given a partially filled *n* by *n* sudoku board, determine if there is a solution.

Exhaustive Search (Brute Force Search):

> Try every possible way of filling the empty cells and check if it is valid.

Note: checking if a given solution is correct is **easy**.

In our quest to understand efficient computation,

(and life, the universe, and everything)

we come across:

$\boxed{\textbf{P vs NP problem}}$

"Can creativity be automated?"

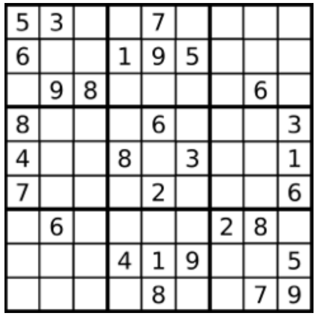
Biggest open problem in all of Computer Science.

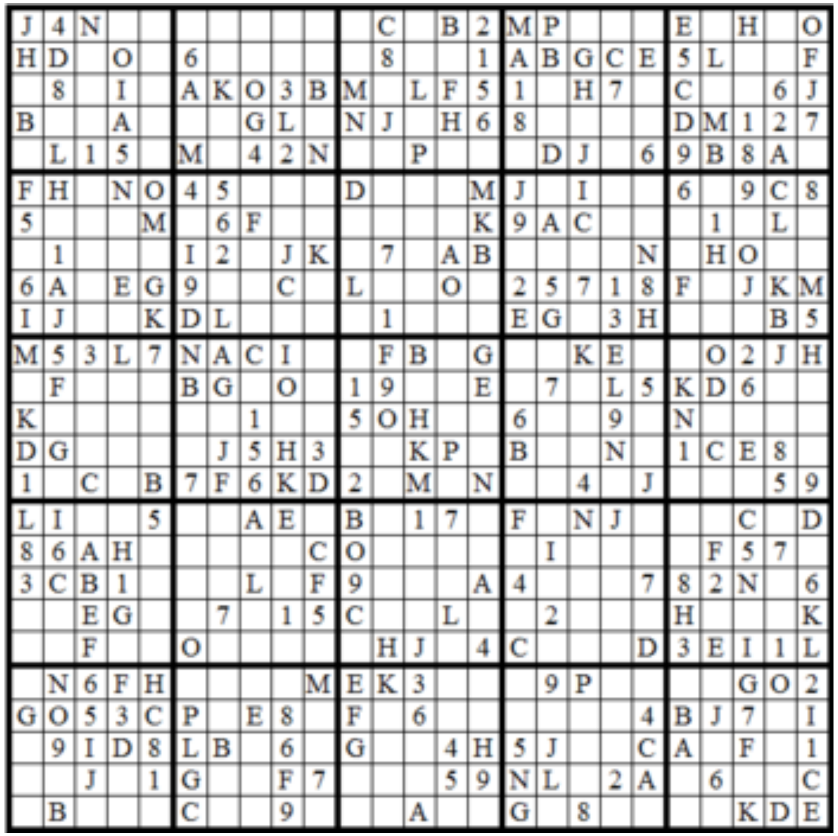
One of the biggest open problems in all of Mathematics.

So what is the **P** vs **NP** question?

The P vs NP question is the following:

Can the Sudoku problem be solved in polynomial time?





WTF?!

So what is the **P** vs **NP** question?

The P vs NP question is the following:

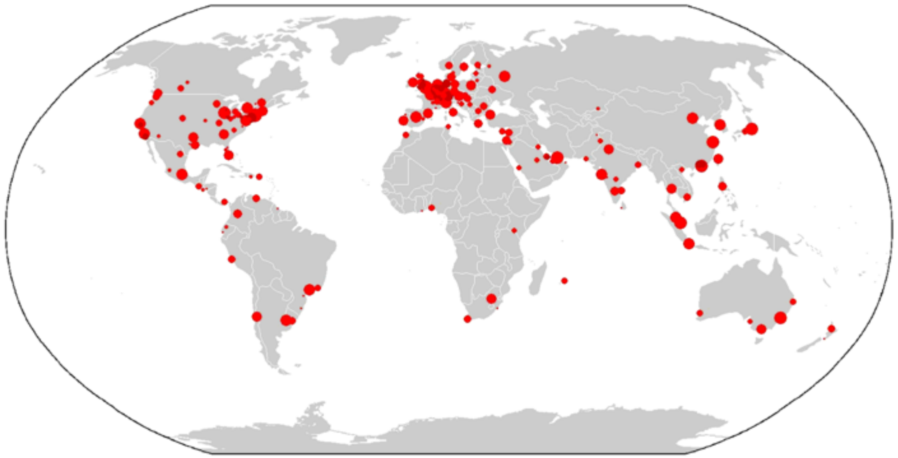
Can the Subset Sum problem be solved in poly-time?

4	-3	-2	7	99	5	1

So what is the **P** vs **NP** question?

The P vs NP question is the following:

Can TSP be solved in poly-time?



So what is the **P** vs **NP** question?

The P vs NP question is the following:

Can the Theorem Proving problem be solved in poly-time?

So what is the **P** vs **NP** question?

The P vs NP question is the following:

Can SAT be solved in poly-time?

$$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_3 \wedge x_4) \vee x_3$$

What the bleep is going on?!?

An important goal for a computer scientist

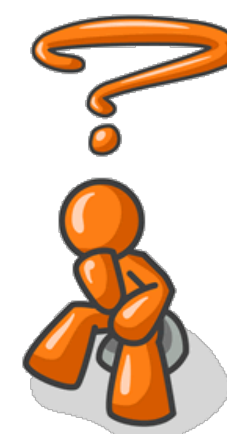
Identifying and dealing with intractable problems

After decades of research and billions of dollars of funding, **no** poly-time algs for:

SAT, Theorem Proving, TSP, Sudoku, …

Can we <u>prove</u> there is no poly-time alg?

poly-time algs.



Goal:

Find *evidence* these problems are computationally hard (i.e., they are not in **P**)

Revisiting reductions

A central concept for comparing the "difficulty" of problems.

will differ based on context

Right now we are interested in poly-time decidability **vs**
not poly-time decidability

<u>Want to define:</u> $A \leq B$ (B is at least as hard as A w.r.t. poly-time decidability.)

B poly-time decidable $\implies$ A poly-time decidable

$B \in$ **P** $\implies$ $A \in$ **P**

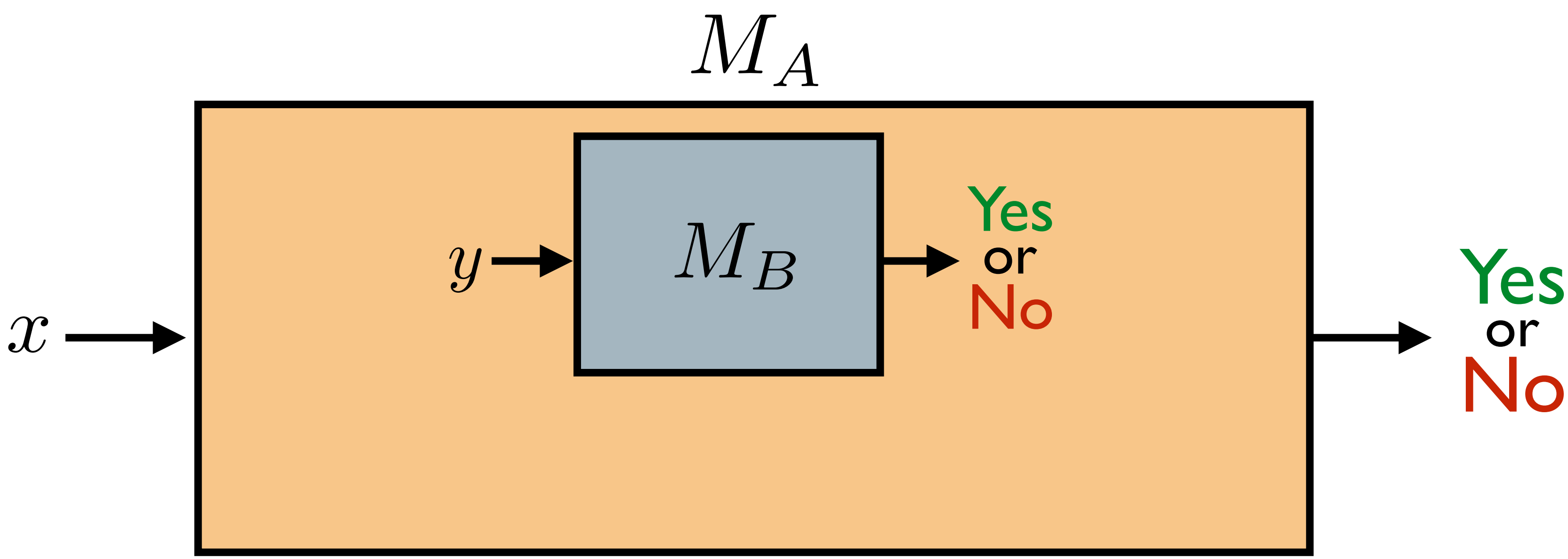
A not poly-time decidable $\implies$ B not poly-time decidable

$A \notin$ **P** $\implies$ $B \notin$ **P**

Revisiting reductions

Notation: $A \leq_T^P B$ (A poly-time reduces to B)

if there is a poly-time machine M_A that decides A using an oracle M_B for B as a black-box subroutine.



B in **P** $\implies$ A in **P**

A not in **P** $\implies$ B not in **P**

Revisiting reductions

Example

A:

Given a graph and an integer k, does there exist at least k pairs of vertices connected to each other?
(by a path)

B:

Given a graph and a pair of vertices (s,t), is s and t connected?

A poly-time reduces to *B*

Revisiting reductions

The 2 sides of reductions

1. Expand the landscape of tractable problems.

If $A \leq_T^P B$ and B is tractable, then A is tractable.

$$B \in \textbf{P} \implies A \in \textbf{P}$$

Revisiting reductions

The 2 sides of reductions

2. Expand the landscape of intractable problems.

If $A \leq_T^P B$ and A is intractable, then B is intractable.

$$A \notin \mathbf{P} \implies B \notin \mathbf{P}$$

But we are pretty lousy at showing a problem is intractable.

Maybe we can still make good use of this…

Gathering evidence for intractability

including some that we think should not be in **P**

If we can show $L \leq_T^P A$ for **many** L

then that would be good evidence that $A \notin$ **P**.

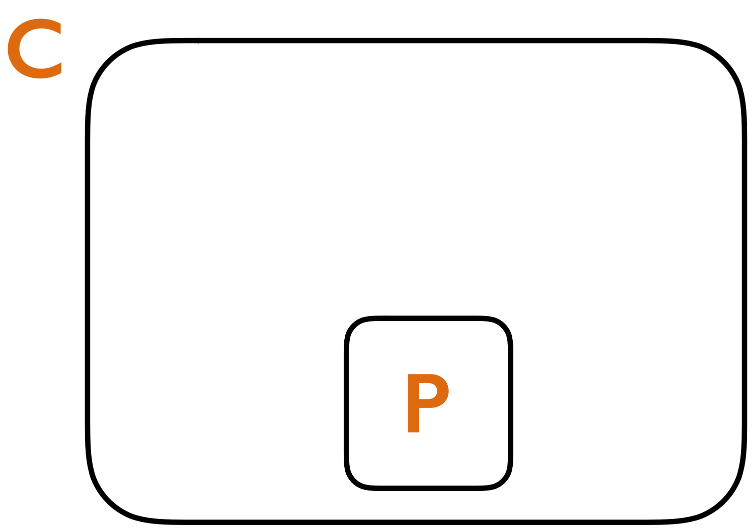
Definitions of C-hard and C-complete

Definition: Let C be a set of languages containing P.

We say that language A is C-hard if

for all $L \in$ C , $\quad L \leq_T^P A$.

> "A is at least as hard as every language in C."



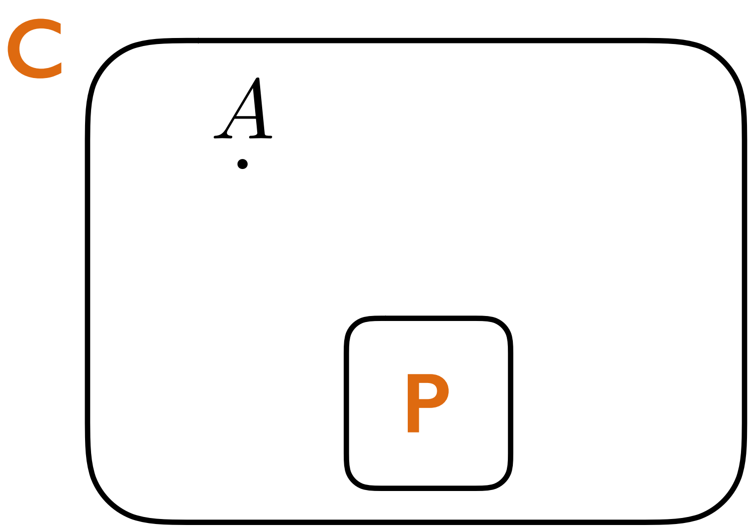
$\leq_T^P A$

Definitions of C-hard and C-complete

Definition: Let C be a set of languages containing P.
We say that language A is C-complete if

- A is C-hard;
- $A \in$ C.

> “A is a representative for hardest languages in C.”



$\leq_T^P A$

Definitions of C-hard and C-complete

Observation:

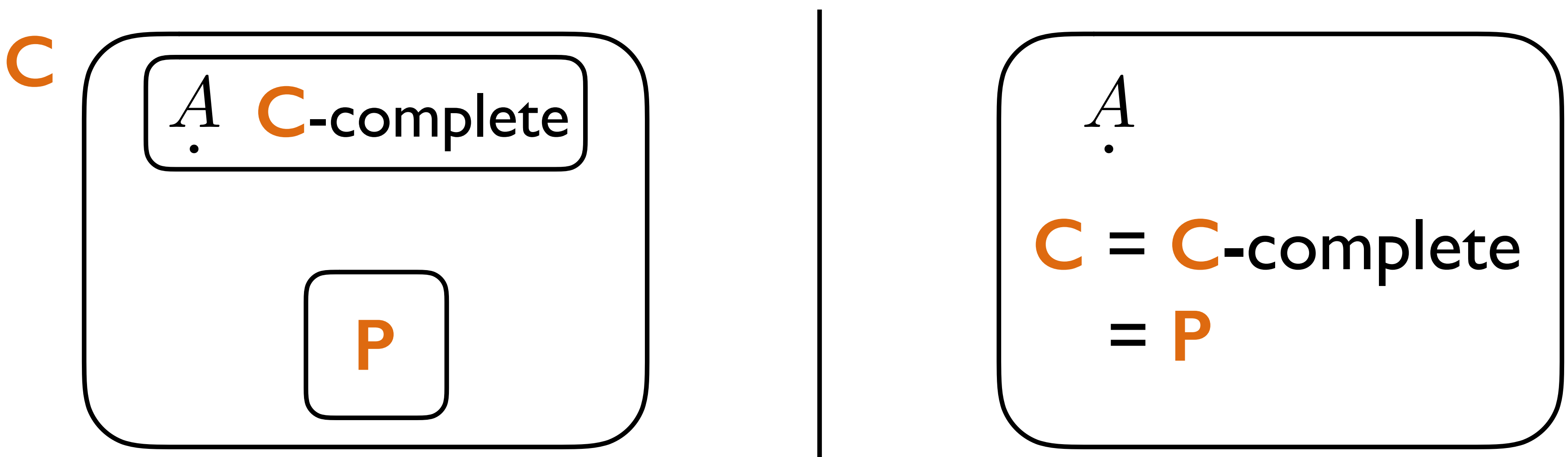
Suppose A is C-complete.

- If C = P, then $A \in$ P.
- If $A \in$ P, then C = P.

$\boxed{\text{C} = \text{P} \iff A \in \text{P}}$

(If we believe C $\neq$ P, then we must believe $A \notin$ P.)

2 possible worlds



Recall the goal

Good evidence for $A \notin$ **P** :

- A is **C**-complete for a really rich/large set **C**

  (a set **C** such that we believe **C** $\neq$ **P**)

So what is a good choice for **C** ?

(if we want to show *SAT, Theorem Proving, TSP,* … are **C**-complete?)




Main Goal Reduces to:

Find a good choice for C

(if we want to show *SAT, Theorem Proving, TSP*, … are C-complete)




Finding the right complexity class C

<u>Try 1:</u>

C = the set of all languages

Can it be true that SAT is C-complete?

<u>Try 2:</u>

C = the set of all languages
"decidable using Brute Force Search (BFS)"

Can it be true that SAT is C-complete?

A complexity class for BFS?

What would be a reasonable definition for:

"class of problems decidable using BFS" ?

What is common about
SAT, Theorem Proving, TSP, Sudoku, etc…?

Seems hard to **find** a correct solution
(solution space is too big!)

BUT, easy to **verify** a given solution.

The complexity class NP

Informally:

A language is in **NP** if:

whenever we have a Yes input/instance,
there is a "*simple*" proof (solution) for this fact.

1. The length of the proof is polynomial in the input size.
2. The proof can be verified/checked in polynomial time.

Poll: Test your intuition

Which of these are in **NP**?

- Subset Sum
- TSP
- SAT
- Circuit-SAT
- Sudoku
- HALTS
- $\{0^k 1^k : k \in \mathbb{N}\}$

The complexity class NP

Formally:

Definition:

A language A is in **NP** if

- there is a polynomial-time TM V
- a constant k

such that for all $x \in \Sigma^*$:

$$x \in A \iff \exists u \text{ with } |u| \leq |x|^k \text{ s.t. } V(x,u) = 1.$$

If $x \in A$, there is some poly-length proof that leads V to accept.

If $x \notin A$, every "proof" leads V to reject.

The complexity class **NP**

Formally:

Definition:

A language A is in **NP** if

- there is a polynomial-time TM V
- a constant k

such that for all $x \in \Sigma^*$:

$$x \in A \iff \exists u \text{ with } |u| \leq |x|^k \text{ s.t. } V(x, u) = 1.$$

The following are synonyms in this context:

proof = solution = certificate

Examples of languages in **NP**

CLIQUE

Input: $\langle G, c \rangle$ where *G* is a graph and *c* is a positive int.

Output: Yes iff *G* contains a clique of size *c*.

Fact: CLIQUE is in **NP**.

Examples of languages in NP

Proof: We need to show a verifier TM V exists as specified in the definition of **NP**.

def $V(x, u)$:

- if x is not an encoding $\langle G = (V, E), c \rangle$ of a valid graph *G* and a positive integer *c*, REJECT.

- if u is not an encoding of a set $S \subseteq V$ of size *c*, REJECT.

- for each pair of vertices in S:

  - if the vertices are not neighbors, REJECT.

- ACCEPT

Examples of languages in NP

Proof (continued):

Need to show:

1. if $x \in$ CLIQUE, there is some proof u (of poly-length) that makes V ACCEPT.
2. if $x \notin$ CLIQUE, no matter what u is, V REJECTS.
3. V is polynomial-time.

   (we leave 3 as an exercise)

Examples of languages in NP

Proof (continued):

Need to show:

1. if $x \in$ CLIQUE, there is some proof u (of poly-length) that makes V ACCEPT.

if $x \in$ CLIQUE, then $x = \langle G, c \rangle$ is a valid encoding, and *G* contains a clique of size *c*.

Then when u is a valid encoding of this clique, the verifier will accept.

Examples of languages in **NP**

Proof (continued):

Need to show:

2. if $x \notin$ CLIQUE, no matter what u is, V REJECTS.

if $x \notin$ CLIQUE, then there are 2 options:

- x is not a valid encoding $\langle G, c \rangle$.
- x is a valid encoding, but *G* does not contain a clique of size *c*.

In either case, V rejects for any u.
(add a couple of lines of justification)

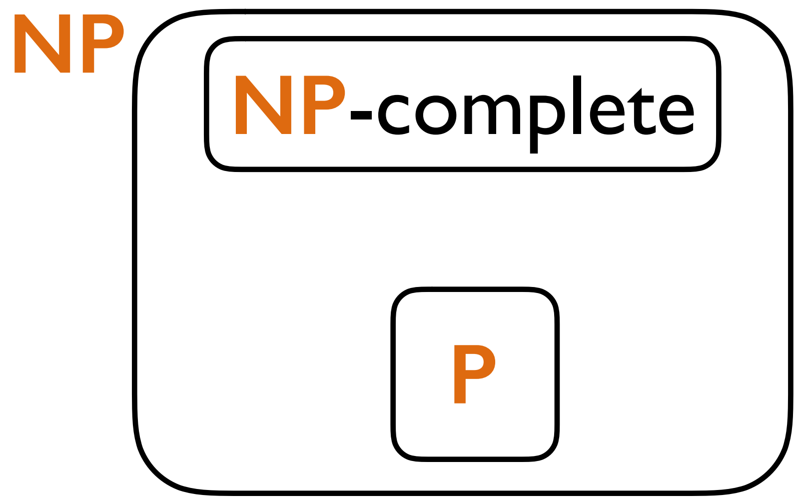
$\square$

The complexity class NP

2 Observations:

1. Every decision problem in NP can be solved using BFS.
   - Go through all potential proofs u, and run $V(x, u)$

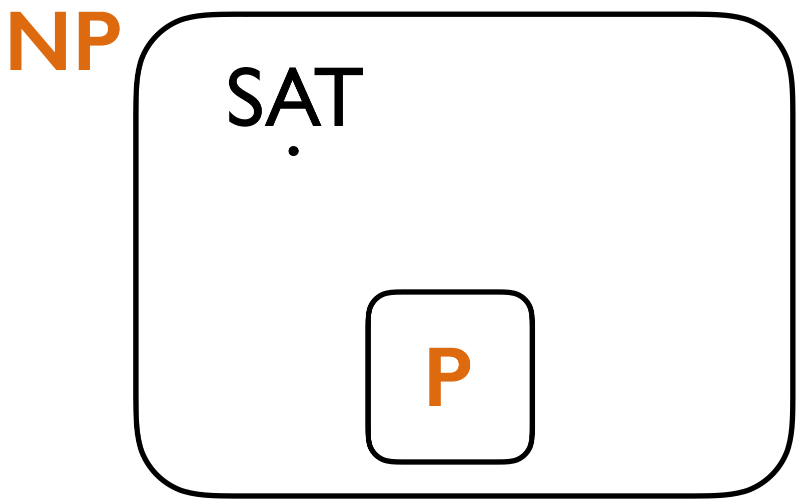
2. This is a HYUUGE class! (believe me!)
   Contains everything in P.



People expect NP contains much more than P.

Coming back to our main goal

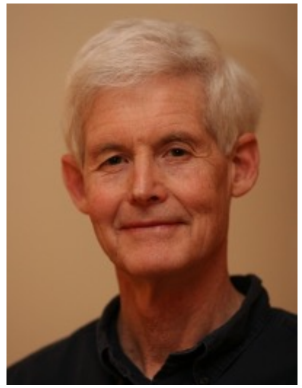
Could it be true that one of
SAT, Theorem Proving, TSP, Sudoku, etc.
is **NP**-complete?

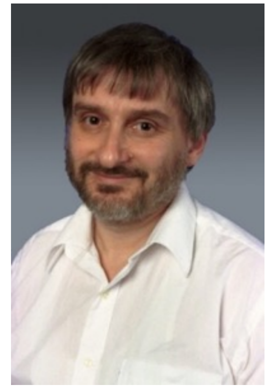


$$\overset{?}{\leq}{}^P_T \quad \text{SAT}$$

Is there **any** language that is **NP**-complete??

The Cook-Levin Theorem





Theorem (Cook 1971 - Levin 1973):

SAT is **NP**-complete.

So SAT is in **NP**. (easy)

And for every L in **NP**, L $\leq_T^P$ SAT .

Karp's 21 NP-complete problems

1972: "Reducibility Among Combinatorial Problems"

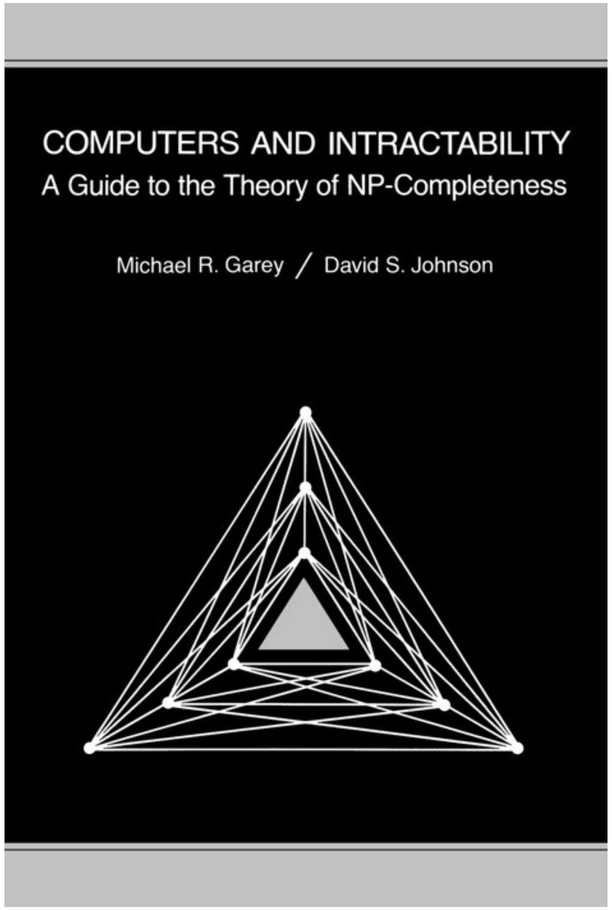
0-1 Integer Programming
Clique
Set Packing
Vertex Cover
Set Covering
Feedback Node Set
Feedback Arc Set
Directed Hamiltonian Cycle
Undirected Hamiltonian Cycle
3SAT

Partition
Clique Cover
Exact Cover
Hitting Set
Knapsack
Steiner Tree
3-Dimensional Matching
Job Sequencing
Max Cut
Chromatic Number

Today

Thousands of problems are known to be NP-complete.

(including the problems mentioned at the beginning of lecture)



1979

Some other "interesting" examples

Super Mario Bros

Given a Super Mario Bros level, is it completable?

arXiv:1203.1895v1 [cs.CC] 8 Mar 2012

Classic Nintendo Games are (NP-)Hard

Greg Aloupis* Erik D. Demaine† Alan Guo†‡

March 9, 2012

Abstract

We prove NP-hardness results for five of Nintendo's largest video game franchises: Mario, Donkey Kong, Legend of Zelda, Metroid, and Pokémon. Our results apply to Super Mario Bros. 1, 3, Lost Levels, and Super Mario World; Donkey Kong Country 1–3; all Legend of Zelda games except Zelda II: The Adventure of Link; all Metroid games; and all Pokémon role-playing games. For Mario and Donkey Kong, we show NP-completeness. In addition, we observe that several games in the Zelda series are PSPACE-complete.

1 Introduction

A series of recent papers have analyzed the computational complexity of playing many different video games [1, 4, 5], yet the most well-known Nintendo games of our youth have yet to be included among these results. In this paper, we consider some of the best-known Nintendo games of all time—Mario, Donkey Kong, Legend of Zelda, Metroid, and Pokémon—and prove that it is NP-hard to play generalized versions of many games in these series. In particular, our results for Mario apply to the NES games Super Mario Bros., Super Mario Bros.: The Lost Levels, Super Mario Bros. 3, and Super Mario World (developed by Nintendo); our results for Donkey Kong apply to the SNES games Donkey Kong Country 1–3 (developed by Rare Ltd.); our results for Legend of Zelda apply to all Legend of Zelda games (developed by Nintendo) except the side-scrolling Zelda II: The Adventure of Link; our results for Metroid apply to all Metroid games (developed by Nintendo); and our results for Pokémon apply to all Pokémon role-playing games (developed by Game Freak and Creatures Inc.).[1]

Our results are motivated in particular by tool-assisted speed runs for video games. A *speed run* of a game is a play through that aims to achieve a fast completion time, usually performed by a human. In a *tool-assisted* speed run, the player uses special tools, such as emulators, to allow them to slow the game down to a frame-by-frame rate in order to achieve superhuman reflexes and timing. In some sense, tool assistance is not cheating because the rules of the game are still obeyed. The resulting speed runs are impressive to watch, as the paths taken by a tool-assisted player are

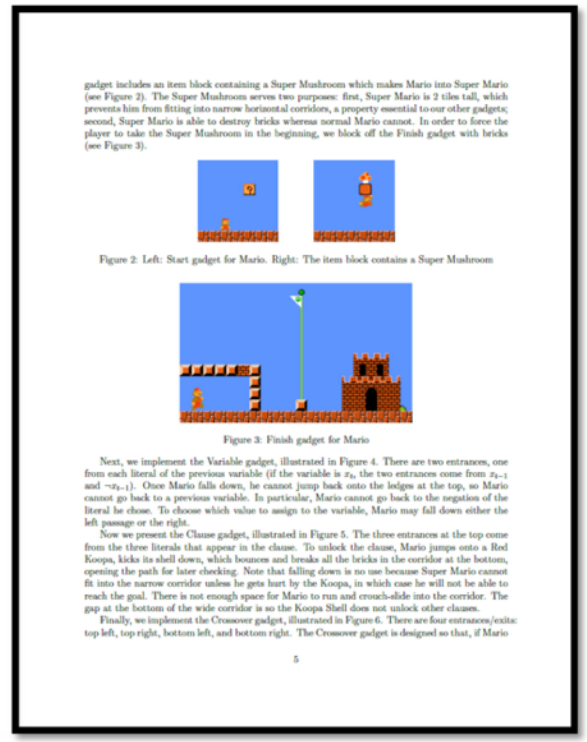
*Chargé de Recherches du FNRS, Département d'Informatique, Université Libre de Bruxelles, aloupis.greg@gmail.com. Work initiated while at Institute of Information Science, Academia Sinica.

†MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA, {edemaine,aguo}@mit.edu

‡Partially supported by NSF grants CCF-0829672, CCF-1065125, and CCF-6922462.

[1]All products, company names, brand names, trademarks, and sprites are properties of their respective owners. Sprites are used here under Fair Use for the educational purpose of illustrating mathematical theorems.

1



gadget includes an item block containing a Super Mushroom which makes Mario into Super Mario (see Figure 2). The Super Mushroom serves two purposes: first, Super Mario is 2 tiles tall, which prevents him from fitting into narrow horizontal corridors, a property essential to our other gadgets; second, Super Mario is able to destroy bricks whereas normal Mario cannot. In order to force the player to take the Super Mushroom in the beginning, we block off the Finish gadget with bricks (see Figure 3).

Figure 2: Left: Start gadget for Mario. Right: The item block contains a Super Mushroom

Figure 3: Finish gadget for Mario

Next, we implement the Variable gadget, illustrated in Figure 4. There are two entrances, one from each literal of the previous variable (if the variable is x_i, the two entrances come from x_{i-1} and $\neg x_{i-1}$). Once Mario falls down, he cannot jump back onto the ledges at the top, so Mario cannot go back to a previous variable. In particular, Mario cannot go back to the negation of the literal he chose. To choose which value to assign to the variable, Mario may fall down either the left passage or the right.

Now we present the Clause gadget, illustrated in Figure 5. The three entrances at the top come from the three literals that appear in the clause. To unlock the clause, Mario jumps onto a Red Koopa, kicks its shell down, which bounces and breaks all the bricks in the corridor at the bottom, opening the path for later checking. Note that falling down is no use because Super Mario cannot fit into the narrow corridor unless he gets hurt by the Koopa, in which case he will not be able to reach the goal. There is not enough space for Mario to run and crouch-slide into the corridor. The gap at the bottom of the wide corridor is so the Koopa Shell does not unlock other clauses.

Finally, we implement the Crossover gadget, illustrated in Figure 6. There are four entrances/exits: top left, top right, bottom left, and bottom right. The Crossover gadget is designed so that, if Mario

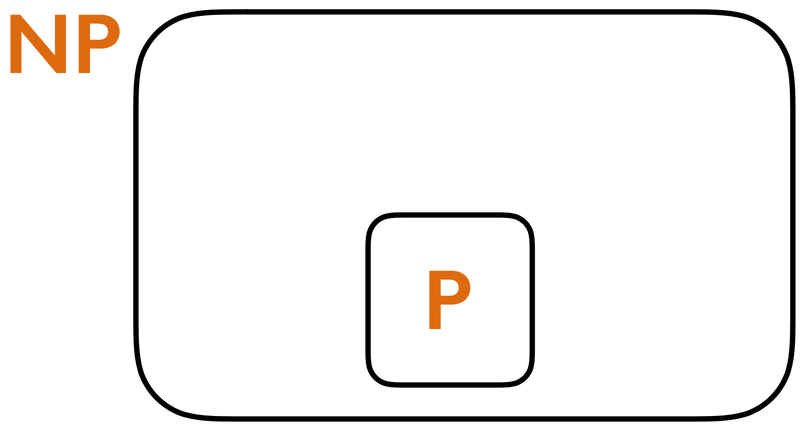
5

Tetris

Given a sequence of Tetris pieces, and a number k, can you clear more than k lines?

How do you show a language is **NP**-complete?

How did Cook and Levin do it ?!?



$\leq_T^P$ SAT

How did Karp do it ?!?

IMPORTANT NOTE:

If SAT $\leq_T^P$ L, then L is **NP**-hard.

(transitivity of $\leq_T^P$)

How do you show a language is **NP**-complete?

It is similar to showing undecidability.

- need an initial direct proof that a language is **NP**-hard. (Cook-Levin Theorem)

- to show other languages are **NP**-hard, use poly-time reductions.

These are the topics of next 2 lectures.

The P vs NP Question

Good evidence for intractability?

If A is **NP**-hard,
that seems to be good evidence that $A \notin$ **P** ...

if you believe **P** $\neq$ **NP**

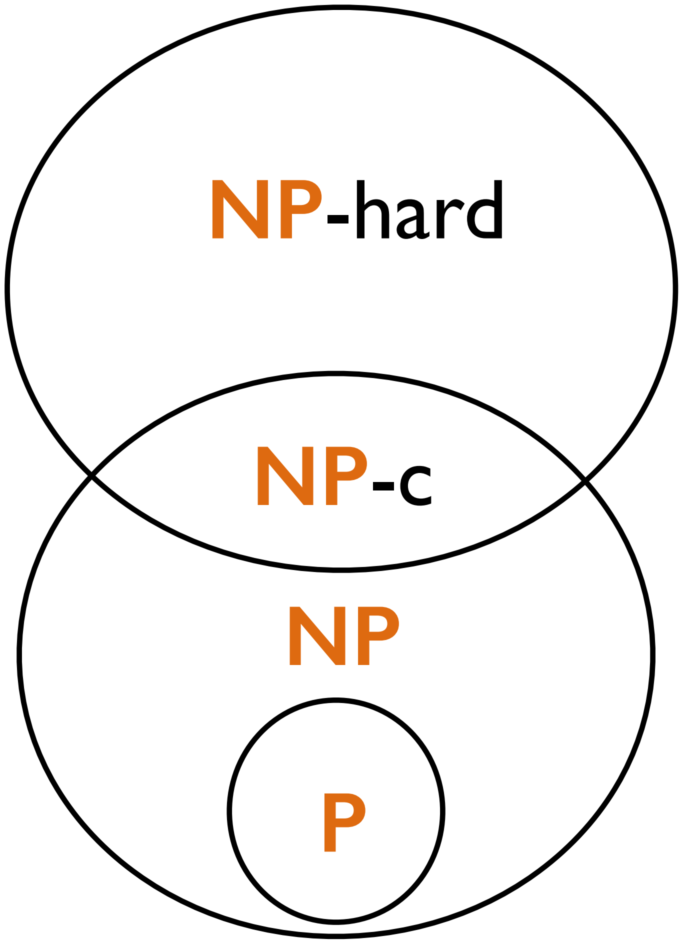
But is **P** $\neq$ **NP**???

The P vs NP question

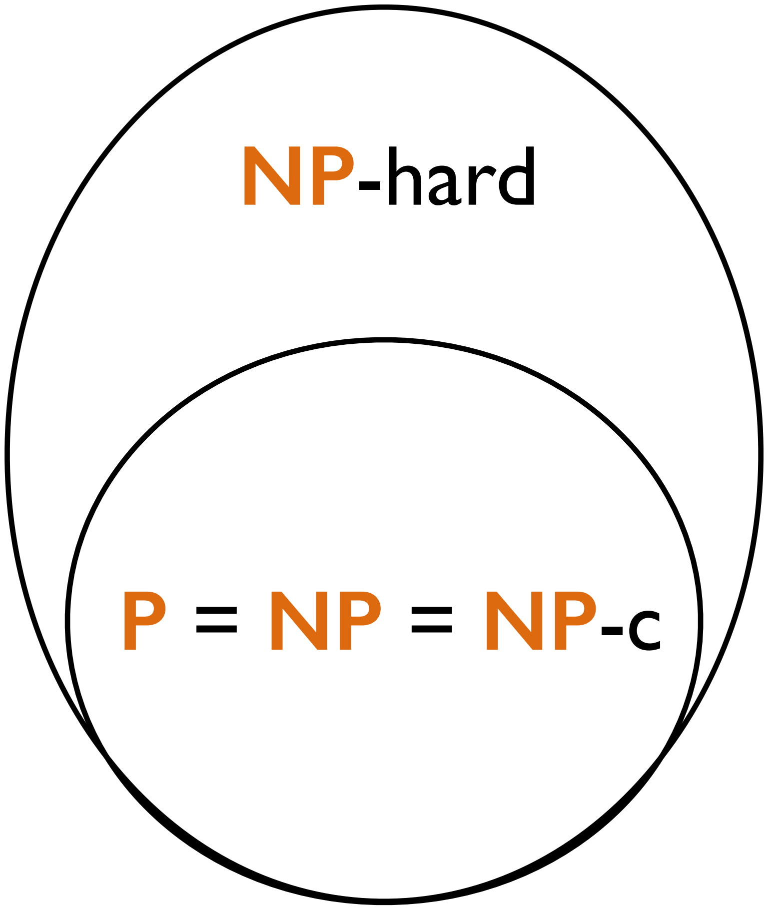
After years of research:

We are pretty confident that this is one of the deepest questions we have ever asked.

The two possible worlds



P ≠ NP



P = NP

What do experts think?

Two polls from **2002** and **2012**

respondents in **2002**: 100

respondents in **2012**: 152

	$P \neq NP$	$P = NP$	Ind	DC	DK
2002	61(61%)	9(9%)	4(4%)	1(1%)	22(22%)
2012	126 (83%)	12 (9%)	5 (3%)	5 (3%)	1(0.6%)

What does **NP** stand for anyway?

Not Polynomial?

None Polynomial?

No Polynomial?

No Problem?

Nurse Practitioner?

It stands for **N**ondeterministic **P**olynomial time.

> Languages in **NP** are the languages decidable in polynomial time by a **nondeterministic TM**.

What does **NP** stand for anyway?

Other contenders for the name of the class:

Herculean

Formidable

Hard-boiled

PET “possibly exponential time”

“provably exponential time”

“previously exponential time”

Next 2 Lectures

How did Cook-Levin show SAT is **NP**-complete?

How do you show other problems are **NP**-complete?