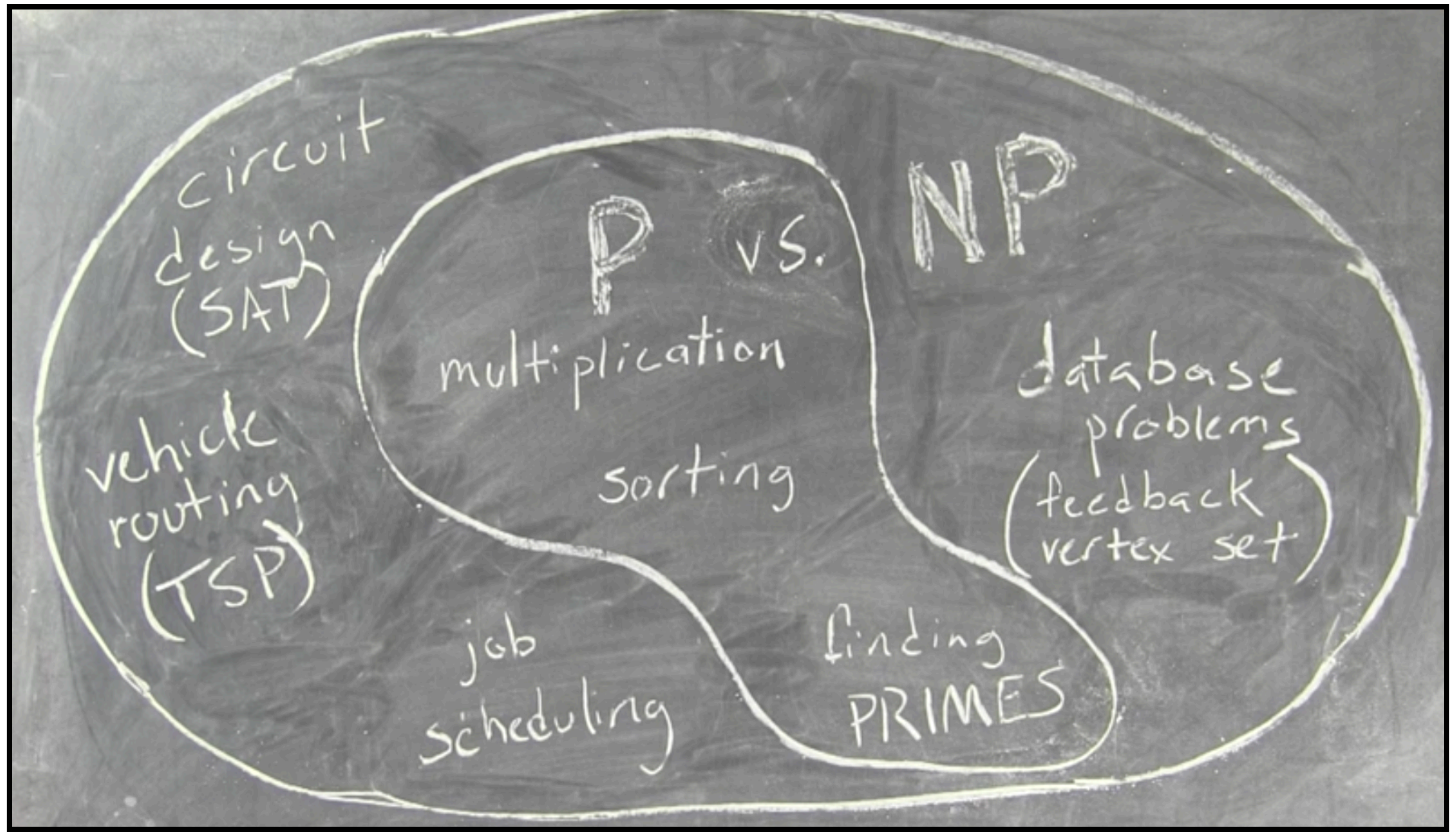
CS251

Great Ideas in *Theoretical* Computer Science



Limits of Efficient Computation: P vs NP

We have a decent understanding of computable vs uncomputable.

computable

uncomputable

<u>NEW GOAL</u>: Understand the divide between poly-time and **not** poly-time

testing primality

matrix multiplication

MST

max matching

shortest path

...

poly-time solvable

Hamiltonian cycle

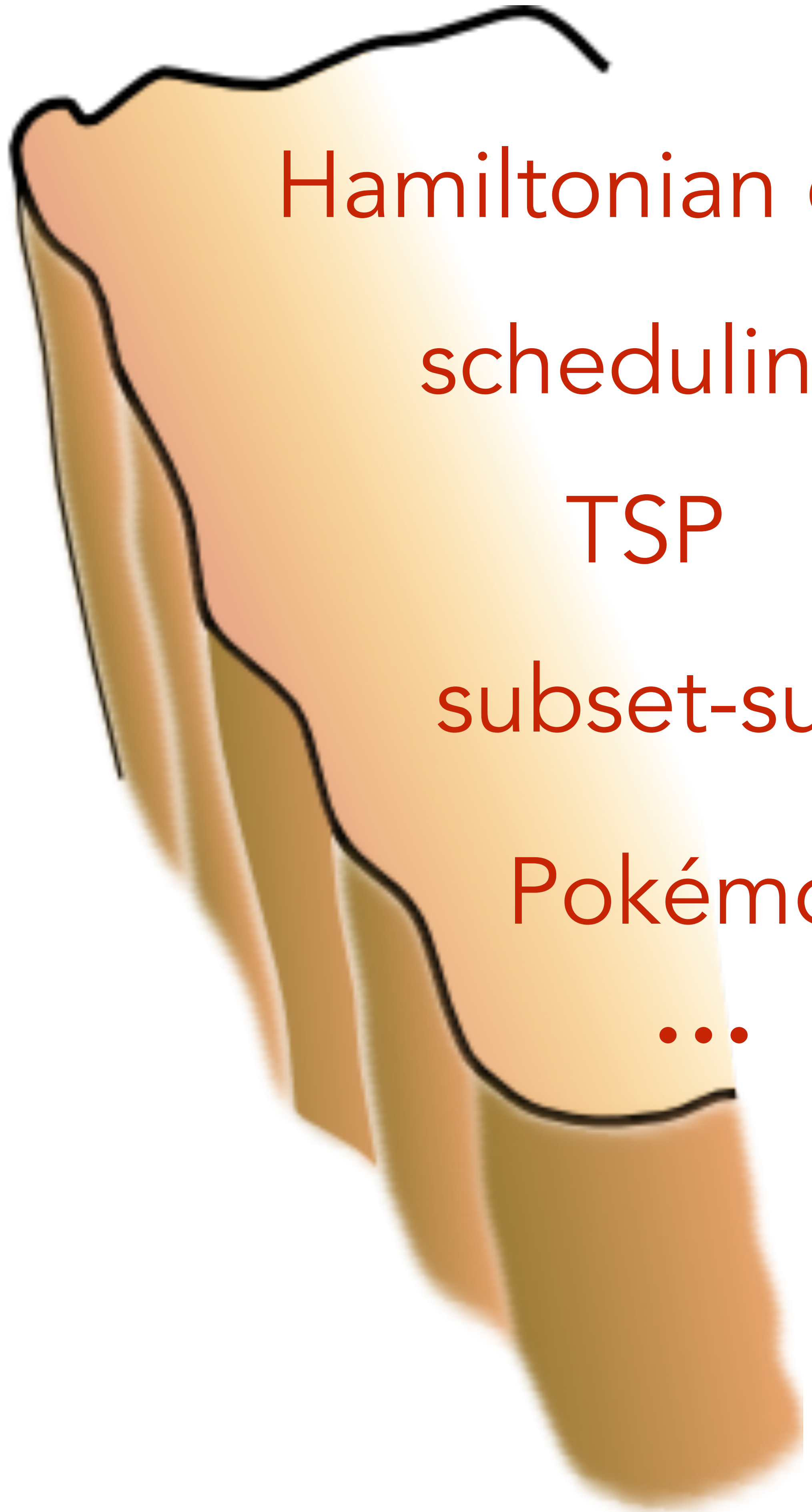
scheduling

TSP

subset-sum

Pokémon

...



best we can say:

exp-time solvable

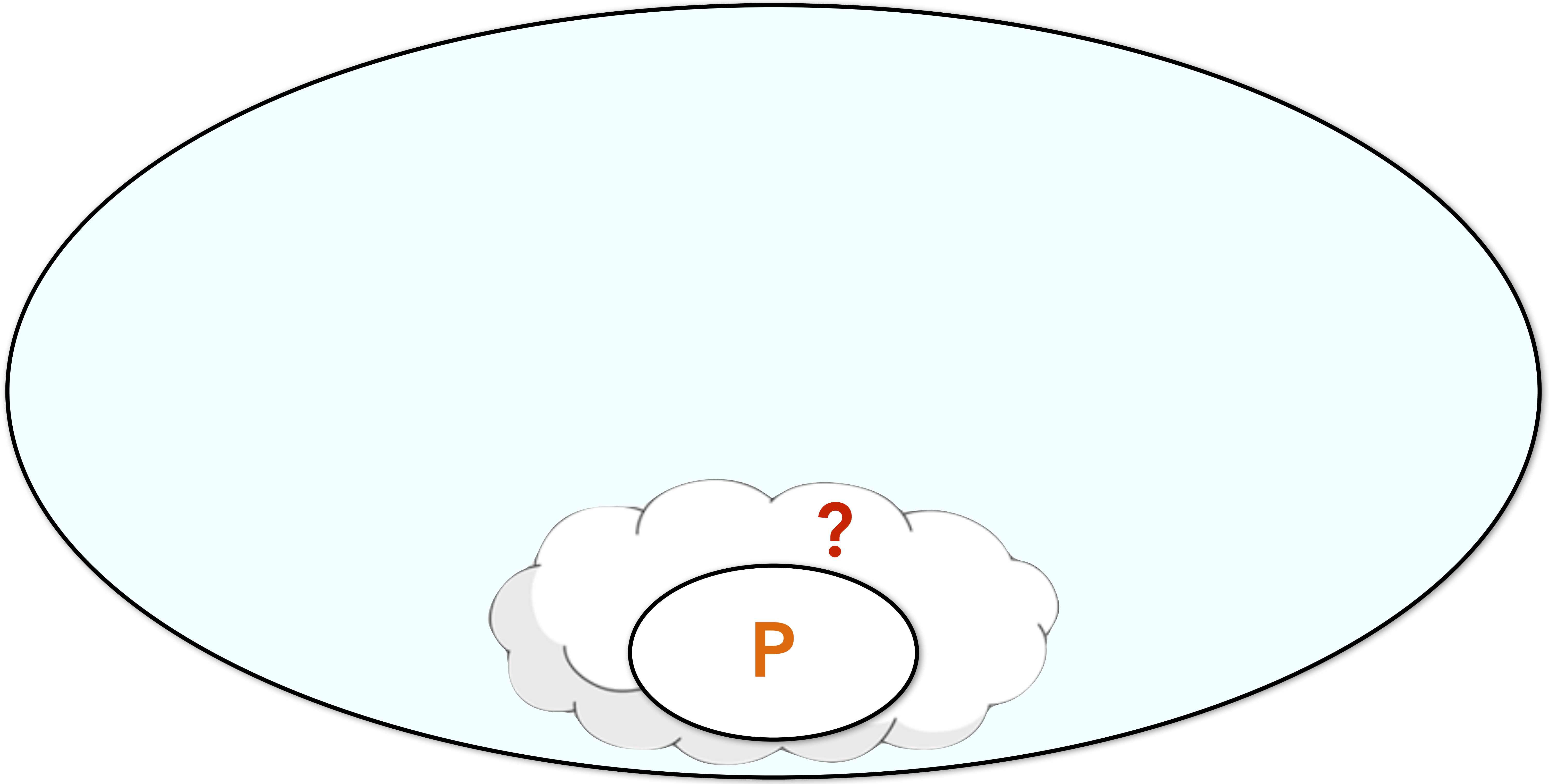
Why poly-time?

Poly-time: **The right level of abstraction!**

Complexity class for poly-time

P : The set of languages that can be solved in $O(n^k)$ steps for some constant k.

Decidable problems



Exponential running time examples

Bounded Entscheidungsproblem

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

Brute Force Search:

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

Verifying if a given string is a correct proof is **easy**.

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

Brute Force Search:

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

Verifying if a given subset sums to 0 is **easy**.

Exponential running time examples

Boolean Satisfiability Problem (SAT)

Given a Boolean propositional formula

e.g. $(x_1 \land \neg x_2) \lor (\neg x_1 \land x_3 \land x_4) \lor x_3$

determine if it is satisfiable?

Brute Force Search:

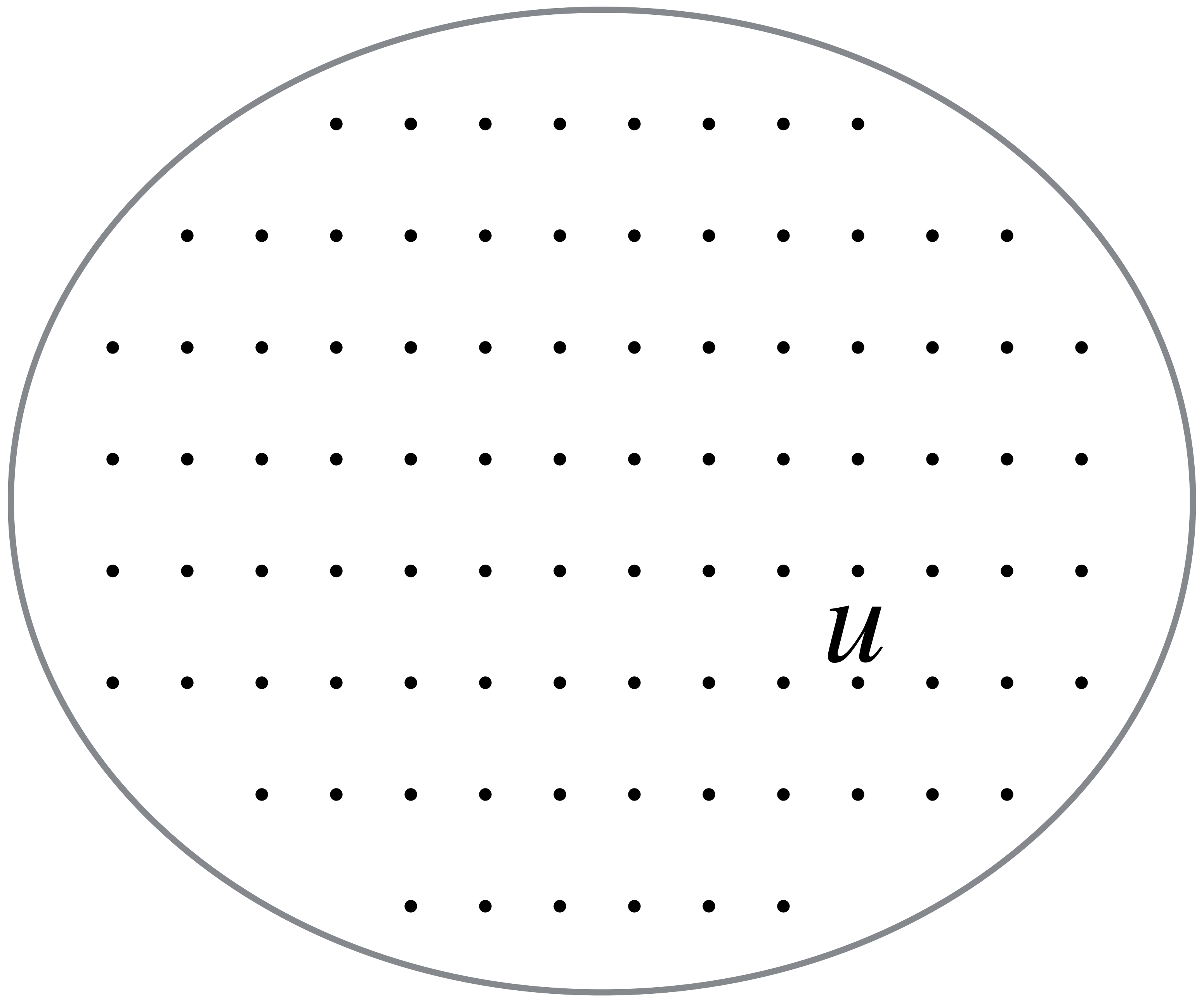
Try every possible truth assignment to the variables.

Evaluate the formula to see the output.

Verifying if a given truth assignment makes the formula True is **easy**.

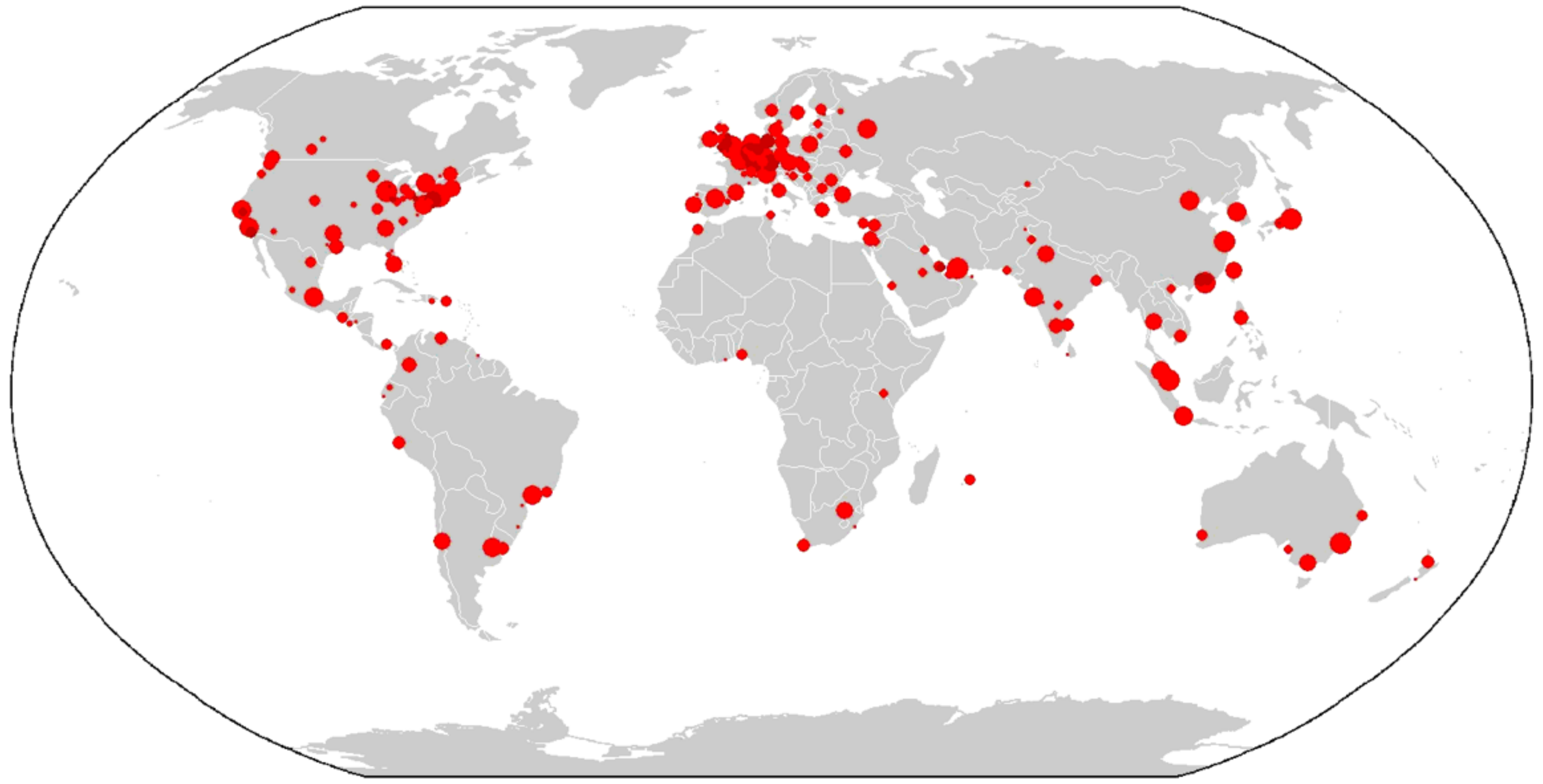
Exponential running time examples

Input x induces a "*possible solutions space*"



Exponential running time examples

Traveling Salesperson Problem (TSP)



Is there an order in which you can visit the cities so that ticket cost is < $15000?

Brute Force Search:

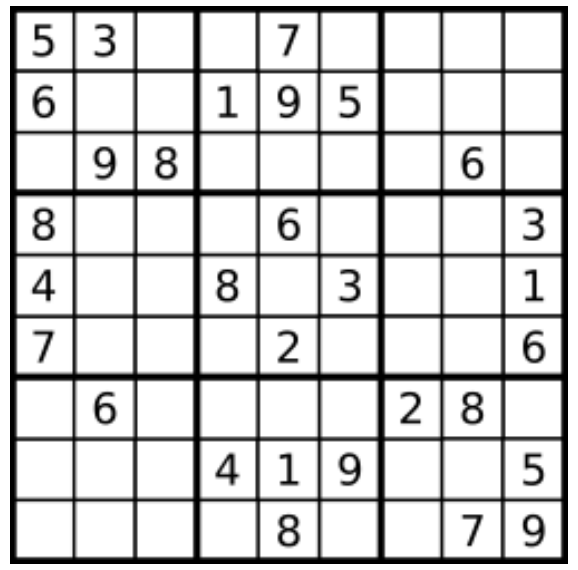
Try every possible order and compute the cost.

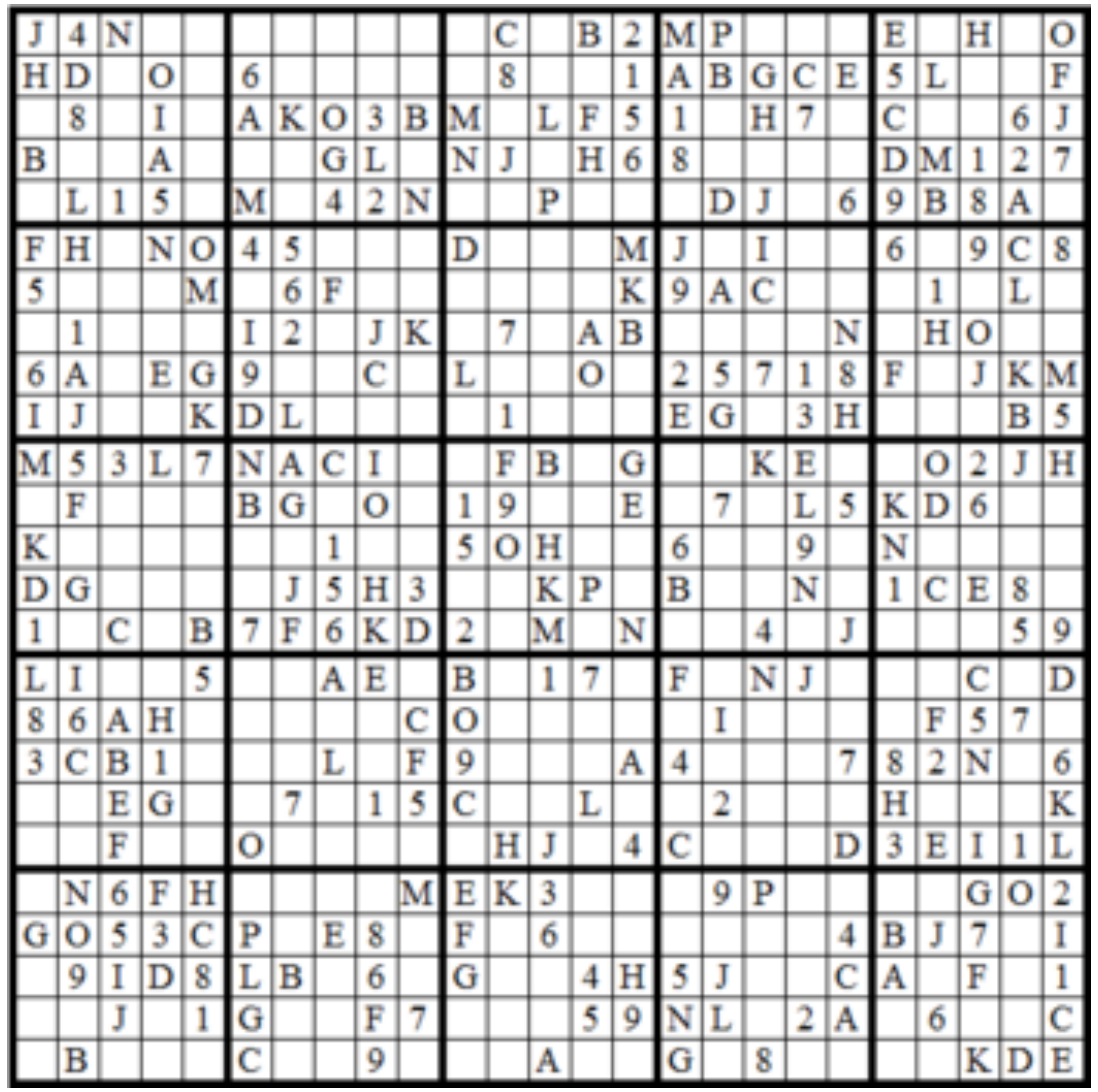
N **Verifying** if a given tour has the desired cost is **easy**.

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.

Brute Force Search:

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

N **Verifying** if a given solution is correct is **easy**.

Exponential running time examples

Scheduling Problem

Given n students, m courses, k time slots.

Each student is taking a subset of the courses.

Can we schedule final exams so that no student has a conflict?

Brute Force Search:

Try every possible way of scheduling the final exams.

N **Verifying** if a given solution is correct is **easy**.

Exponential running time examples

And many many many others in

math, physics, chemistry, biology, medicine, economics, artificial intelligence, cryptography, all sorts of engineering,…

… with humongous applications.

In our quest to understand efficient computation,
(and life, the universe, and everything)
we come across:

P vs NP problem

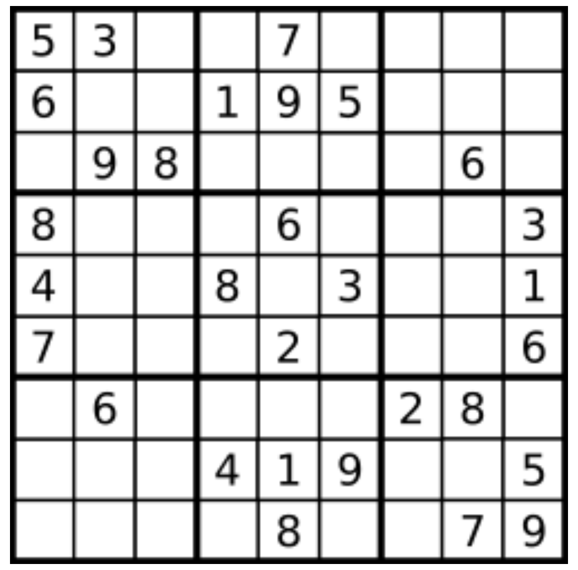
"Can creativity be automated?"

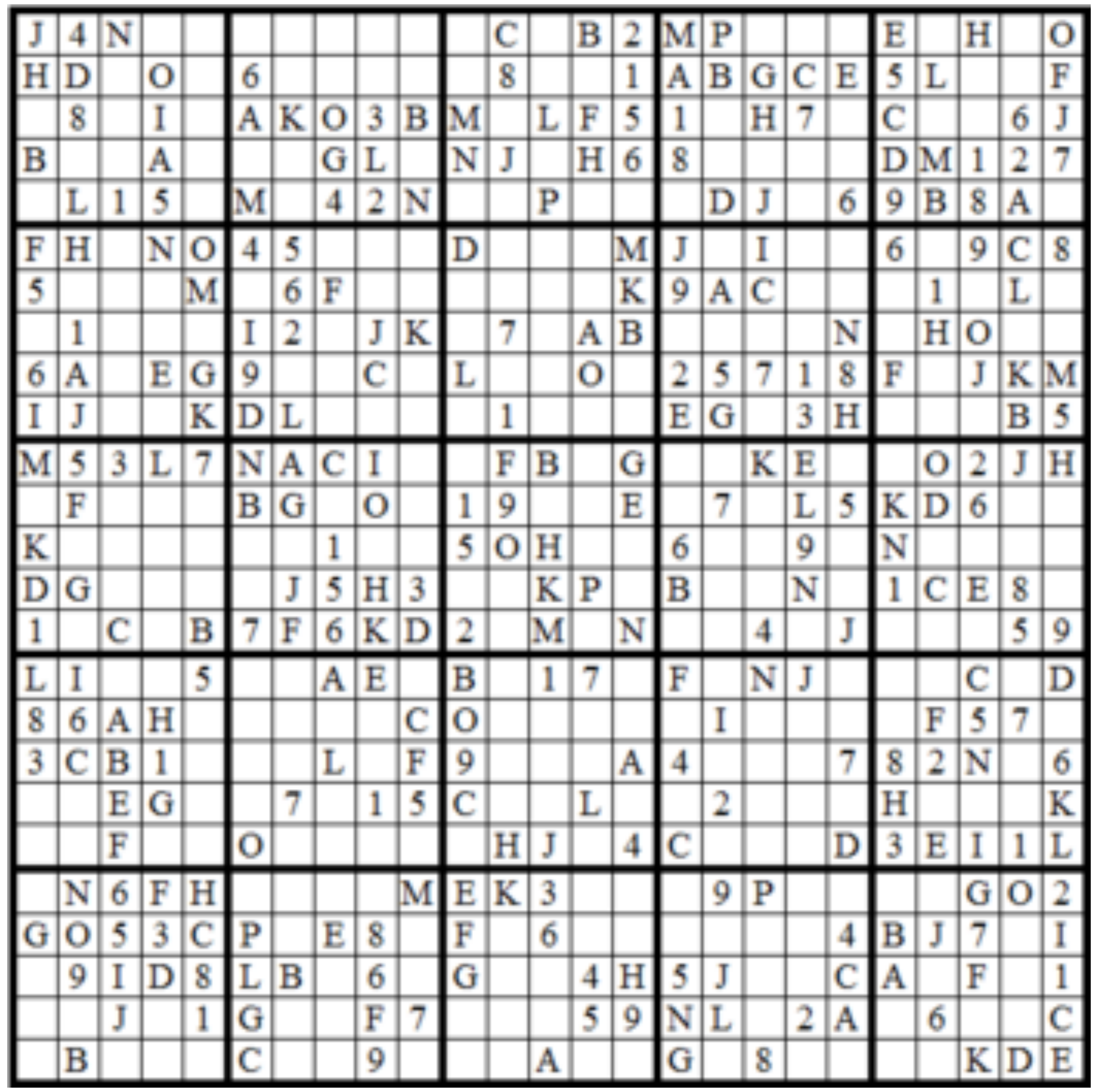
Biggest open problem in Computer Science.

One of the biggest open problems in Mathematics.

The P vs NP question is the following:

Can the Sudoku problem be solved in polynomial time?





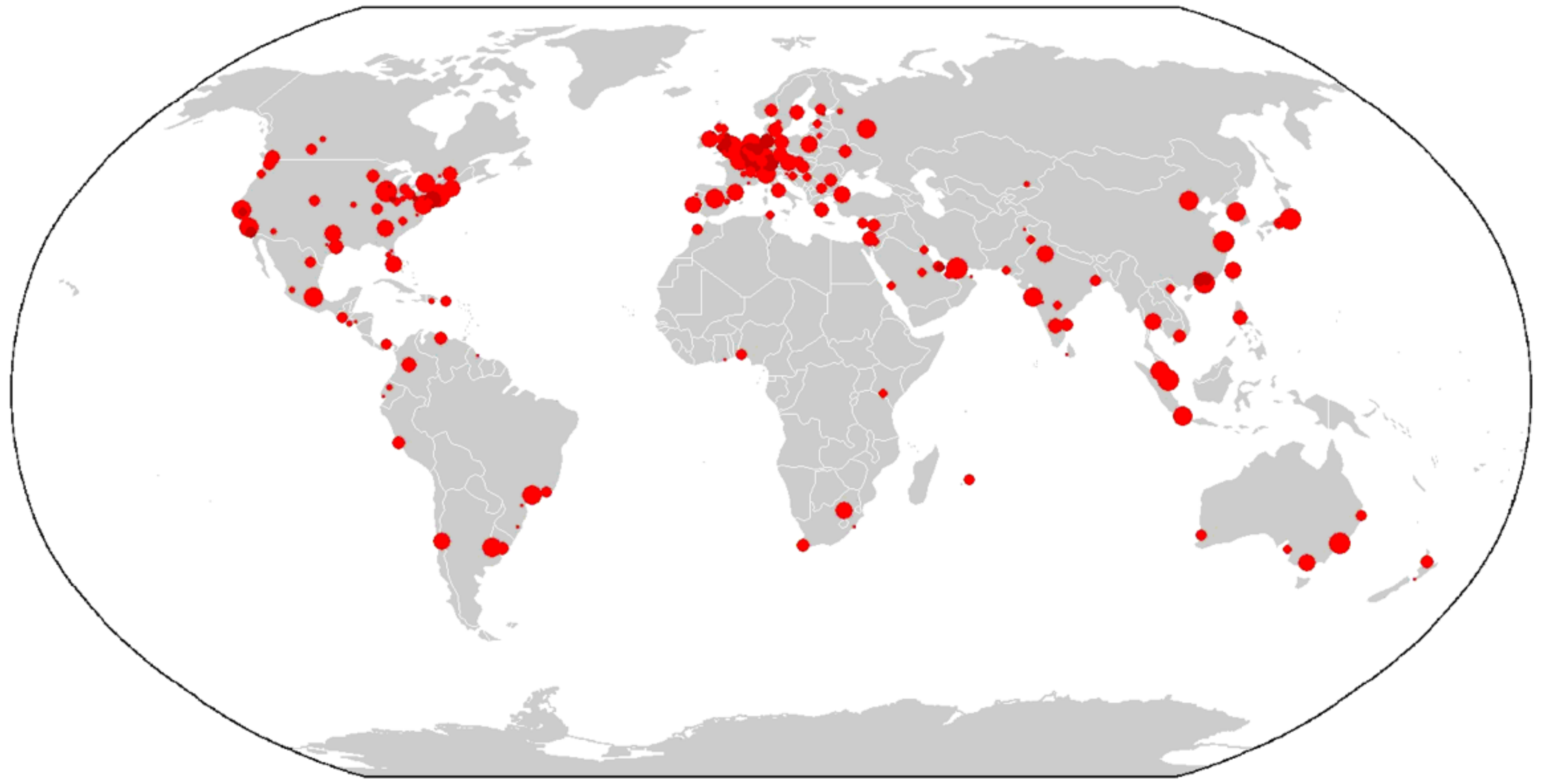
The P vs NP question is the following:

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

4	-3	-2	7	99	5	1

The P vs NP question is the following:

Can TSP be solved in poly-time?



The P vs NP question is the following:

Can SAT be solved in poly-time?

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

The P vs NP question is the following:

Can Bounded Entscheidungsproblem
be solved in poly-time?

Wut?!?

Let's start from the beginning...

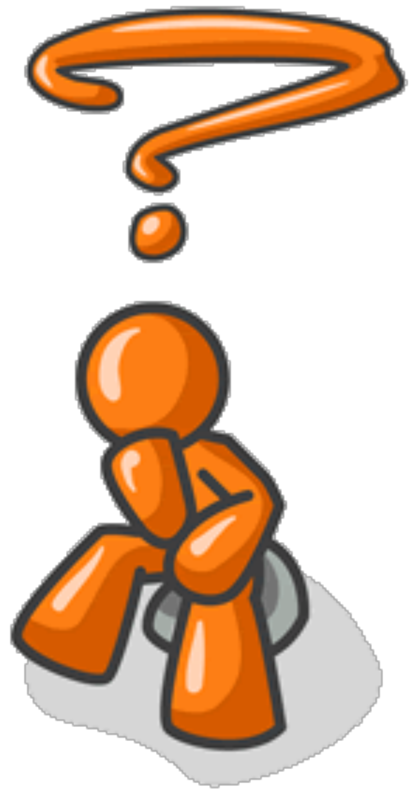
Identifying and dealing with intractable problems

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

Subset Sum, SAT, TSP, Sudoku, …

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

poly-time algs.



(A More Modest) Goal

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

Revisiting reductions

A way to compare "difficulty" of languages/problems.

differs based on context

<u>Want to define:</u> $A \leq B$ to mean

A **is no harder than** B (with respect to **poly-time** decidability).

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

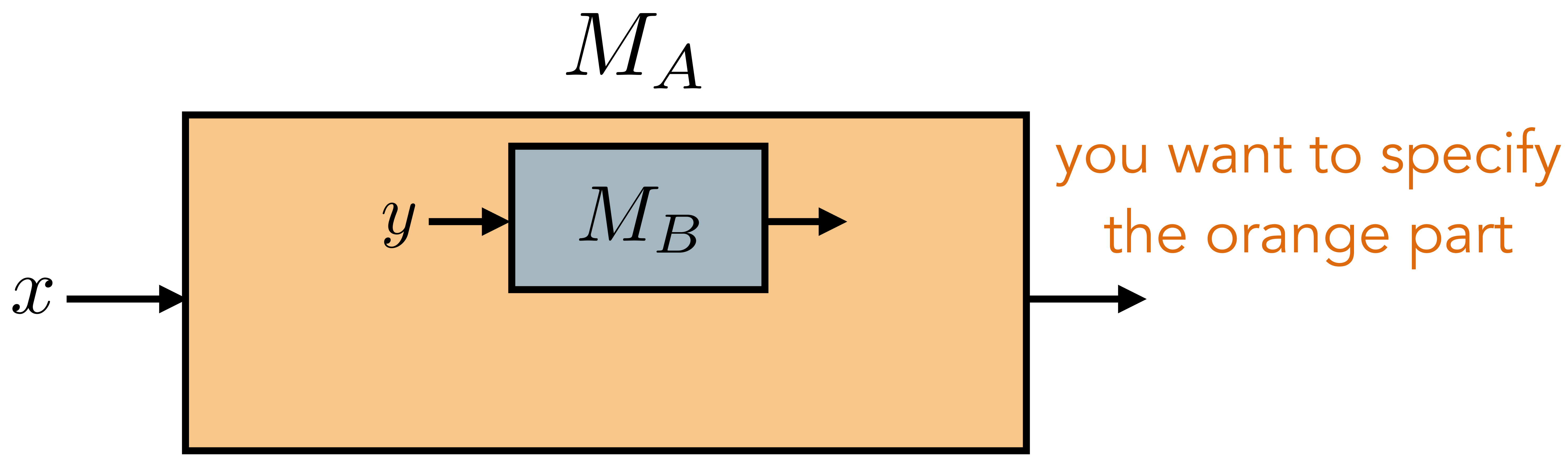
A not poly-time decidable $\implies$ B not poly-time decidable

Revisiting reductions

We write $A \leq^P B$ if you can do the following:

Construct **poly-time** M_A deciding A

that uses a "black-box" subroutine M_B for B.



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

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

The 2 sides of reductions

1. Expand the landscape of tractable problems.

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

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

The 2 sides of reductions

2. Expand the landscape of intractable problems.

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

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

? But we suck at showing a problem is intractable.

Is this still useful?

Gathering evidence for intractability

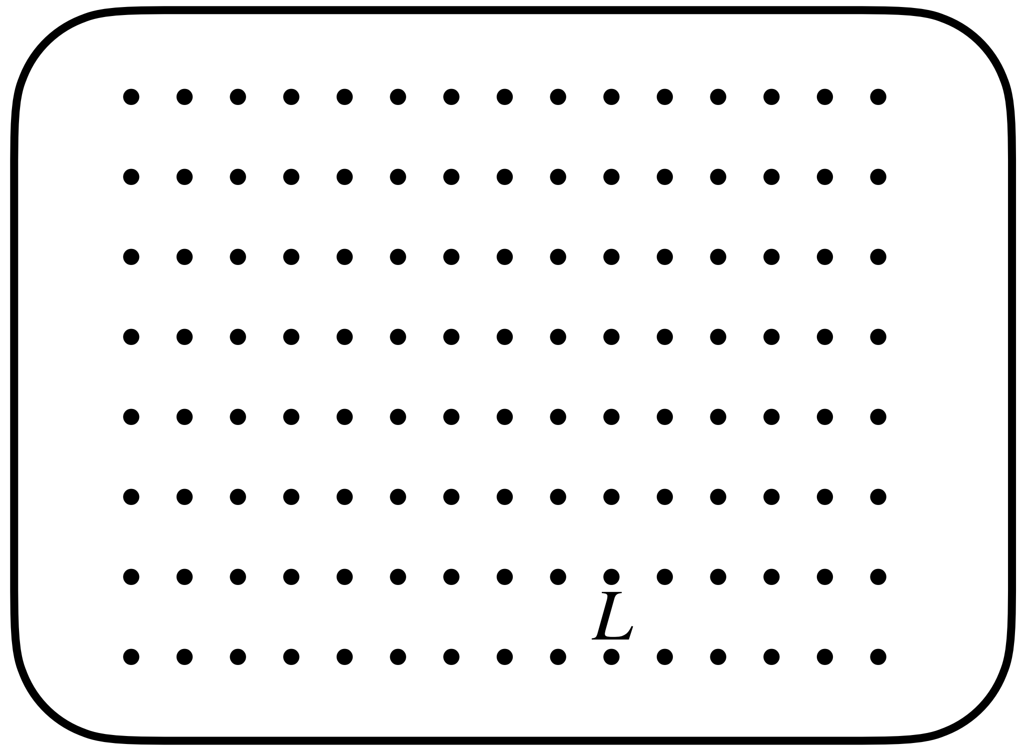
? How can we gather evidence A is intractable (using reductions)?

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

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

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

C

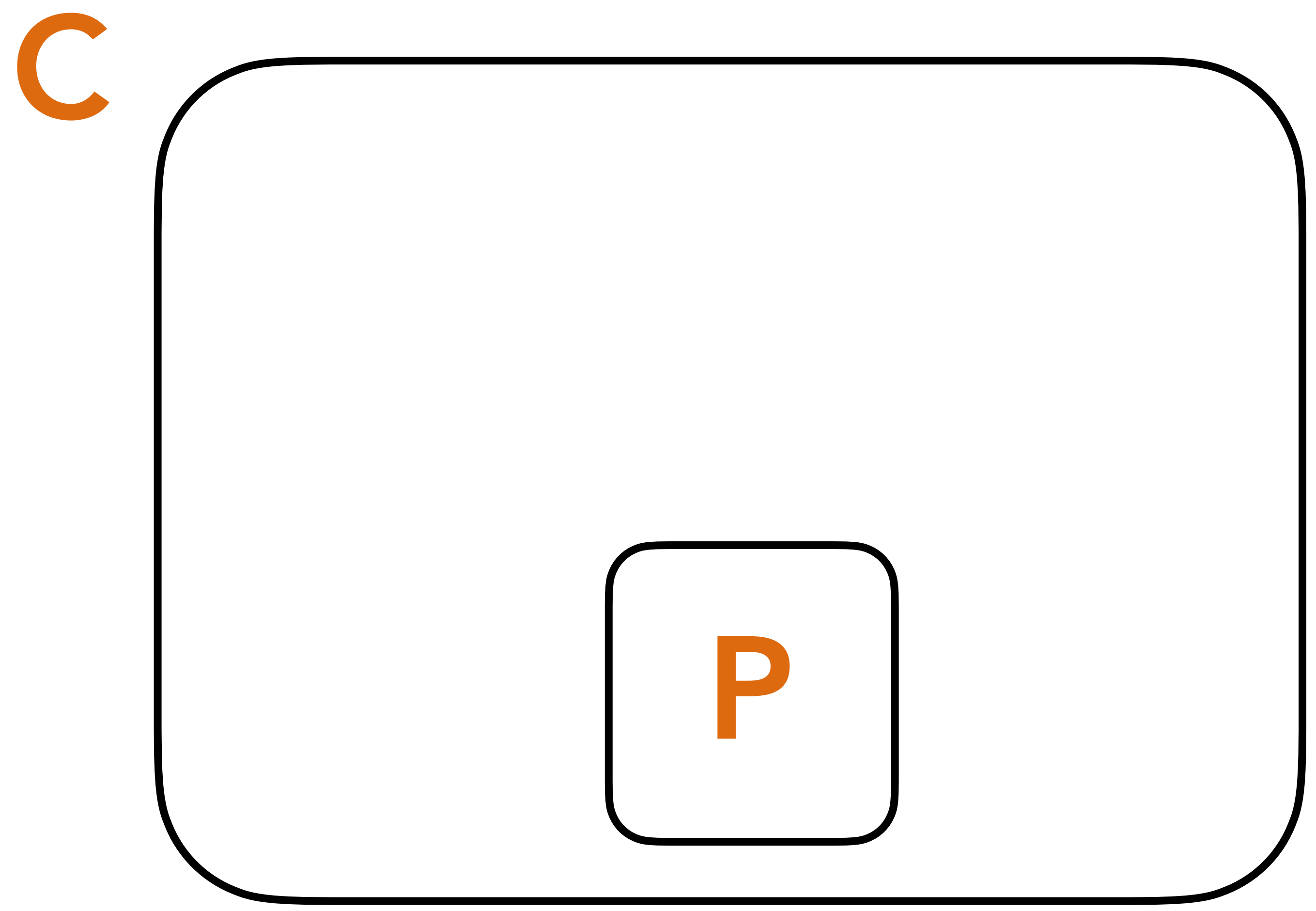


$\leq^P A$

Definitions of C-hard and C-complete

Definition: Let C be a set of languages containing P.
We say language A is C-hard if for all $L \in$ C, $L \leq^P A$.

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



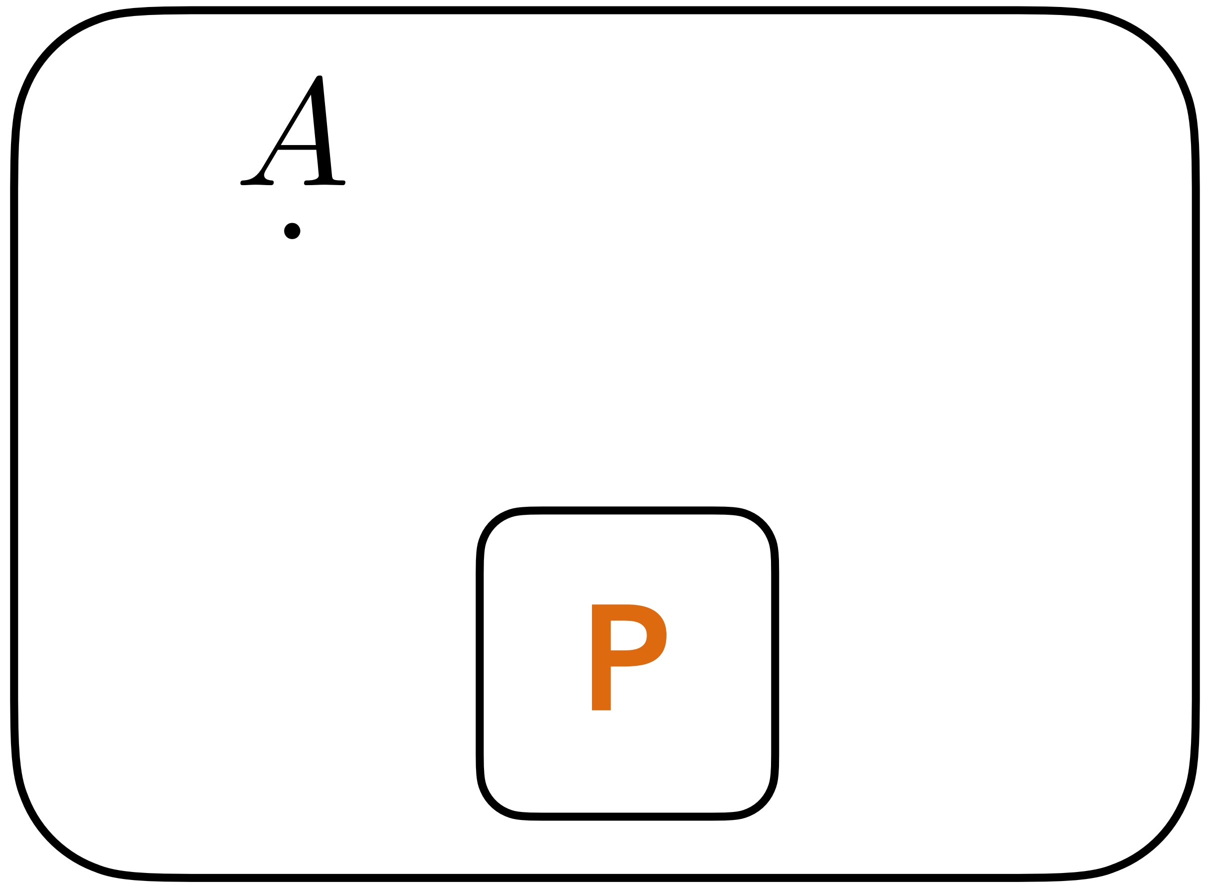
$\leq^P \ A$

Definitions of C-hard and C-complete

Definition: Let C be a set of languages containing P.
We say language A is C-complete if
- A is C-hard,
- $A \in$ C.

"A is a representative for hardest languages in C."

C



$\leq^P \ A$

Observation:

$A \in$ P $\iff$ C = P

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** ≠ **P**)

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

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

Main Goal Reduces to:

Find a good choice for **C**

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

Finding the right complexity class C

Try 1:

C = the set of all languages

SAT is C-complete???

Try 2:

C = the set of all decidable languages

SAT is C-complete???

Try 3:

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

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, Subset Sum, TSP, Sudoku, etc…?

Can be **hard** to ***find*** a correct solution
(solution space can be too big!)

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

The complexity class NP

Super-duper Informal:

NP is a set of languages that we come across all the time
and would love to solve in poly-time.

Informal:

NP is the set of languages that can be solved efficiently with "help".

(help that you do not have to trust, and can verify)

The complexity class NP

Semi-Informal:

proof can be *verified/checked* in poly-time.

A language L is in **NP** if:

- Yes inputs/instances (i.e. $x \in L$) have a "**simple**" proof (solution).
- No inputs/instances (i.e. $x \notin L$) have no proof (solution).

$L \in$ **NP**

Efficient Verifier for L

(input) x →
(proof/solution) u →

Does u certify/prove $x \in L$?

→ True or False

$L \in$ **P**

Efficient Decider for L

(input) x →

$x \in L$?

→ True or False

Poll

Efficient Verifier for L

(input) x →

(proof/solution) u →

Does u certify/prove $x \in L$?

→ True or False

Which languages/problems are in **NP**?

- Subset Sum
- Traveling Salesperson Problem (TSP)
- SAT
- Sudoku
- HALTS
- $\{0^k 1^k : k \in \mathbb{N}\}$

Formal definition of NP

Efficient Verifier for L

(input) x →
(proof/solution) u →

Does u certify/prove $x \in L$?

→ True or False

Definition: A language L is in **NP** if

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

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

$$x \in L \implies \exists u \text{ with } |u| \leq |x|^k \text{ s.t. } V(x,u) \text{ accepts},$$

$$x \notin L \implies \forall u,\ V(x,u) \text{ rejects}.$$

Formal definition of NP

Definition: A language L is in **NP** if

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

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

$$x \in L \implies \exists u \text{ with } |u| \le |x|^k \text{ s.t. } V(x,u) \text{ accepts,}$$

$$x \notin L \implies \forall u,\ V(x,u) \text{ rejects.}$$

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

If $x \notin L$, every u leads V to reject.

Formal definition of NP

Definition: A language L is in **NP** if

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

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

$x \in L \implies \exists u$ with $|u| \leq |x|^k$ s.t. $V(x, u)$ accepts,

$x \notin L \implies \forall u, V(x, u)$ rejects.

The following are synonyms in this context:

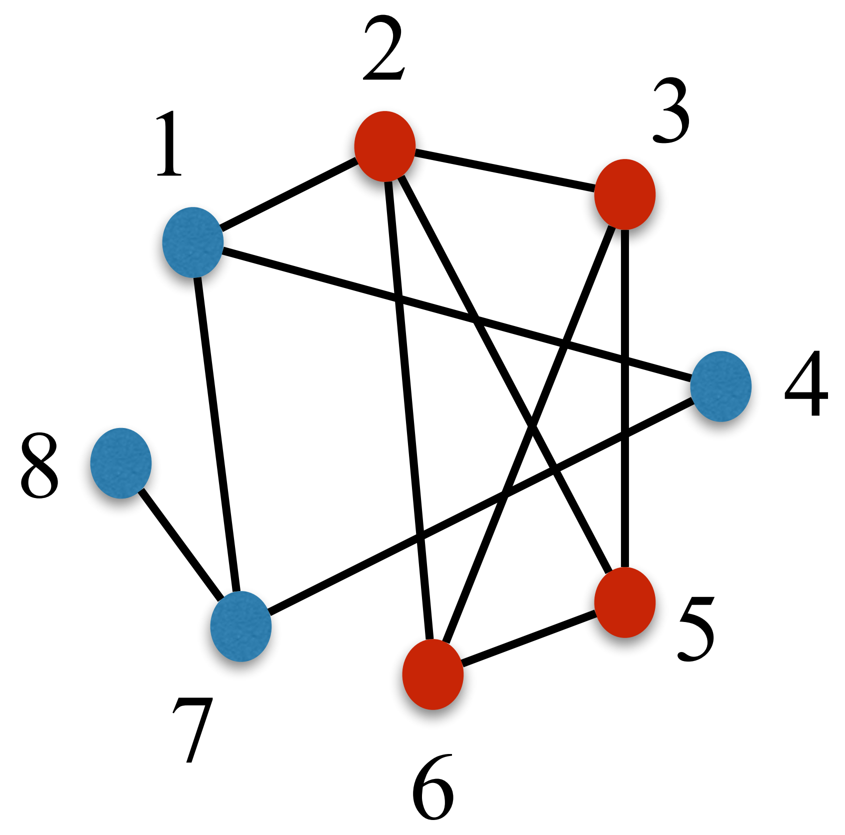
proof = solution = certificate

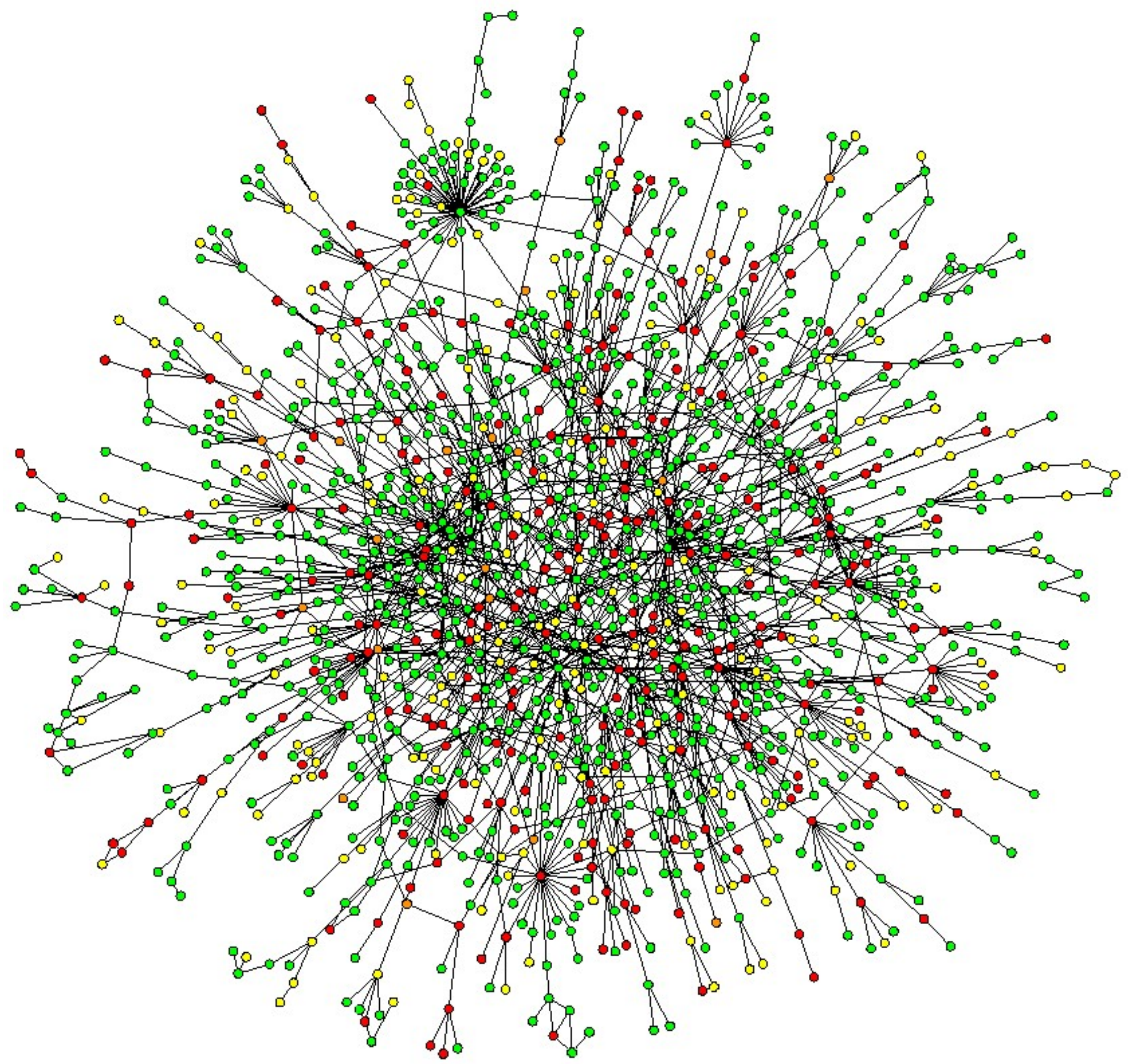
CLIQUE is in NP

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

Output: True iff G contains a clique of size c.

$$\text{CLIQUE} = \{\langle G, c \rangle : \ G \text{ is a graph that contains a clique of size } c\}$$





Fact: CLIQUE is in NP.

CLIQUE is in NP - Proof

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

def $V(\langle$graph $G = (V, E)$, natural $c\rangle$, $u)$:

- 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

CLIQUE is in NP - Proof

Need to show:

1. if $x = \langle G, c \rangle \in$ CLIQUE, <u>there exists</u> u (of poly-length) such that $V(x, u)$ ACCEPTS.

2. if $x = \langle G, c \rangle \notin$ CLIQUE, <u>for all</u> u, $V(x, u)$ REJECTS.

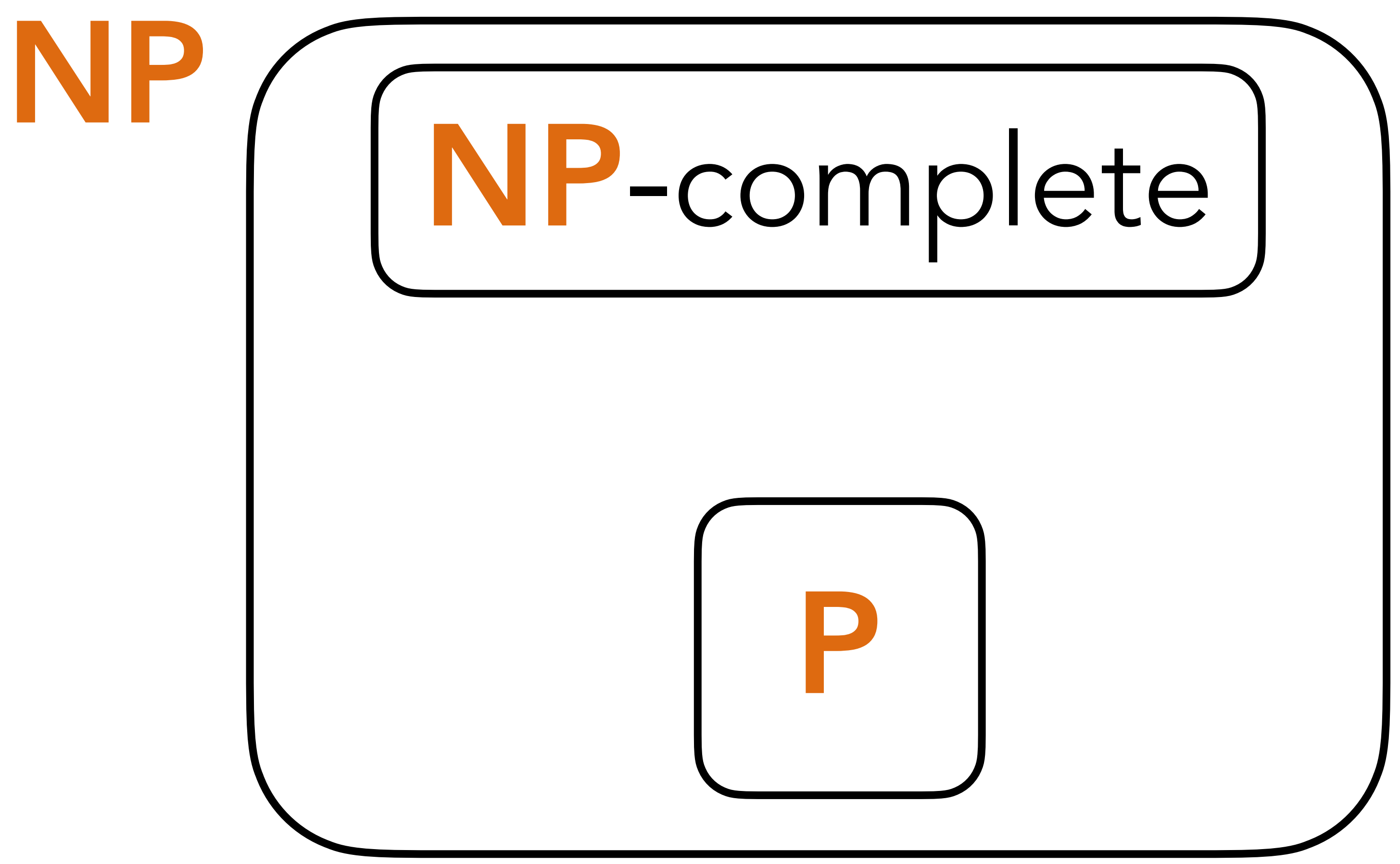
3. V is polynomial-time.

2 Observations about NP

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! And contains everything in P.



People expect NP contains much more than P.

Main Goal:

Find a good choice for **C**

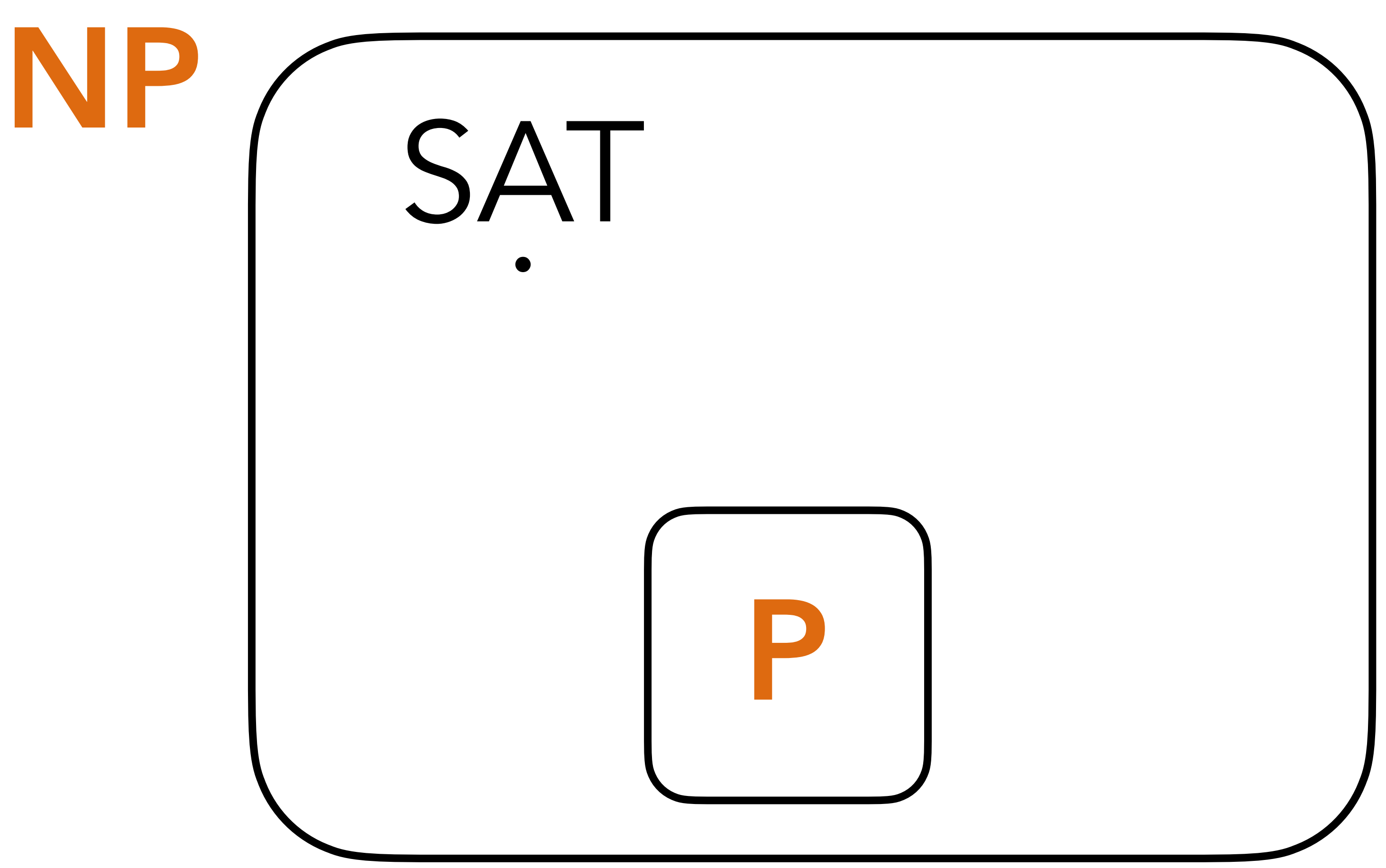
(if we want to show *SAT*, *Sudoku*, *TSP*, … are **C**-complete)

Coming back to our main goal

Could it be true that one of

SAT, *Subset Sum*, *TSP*, *Sudoku*, etc.

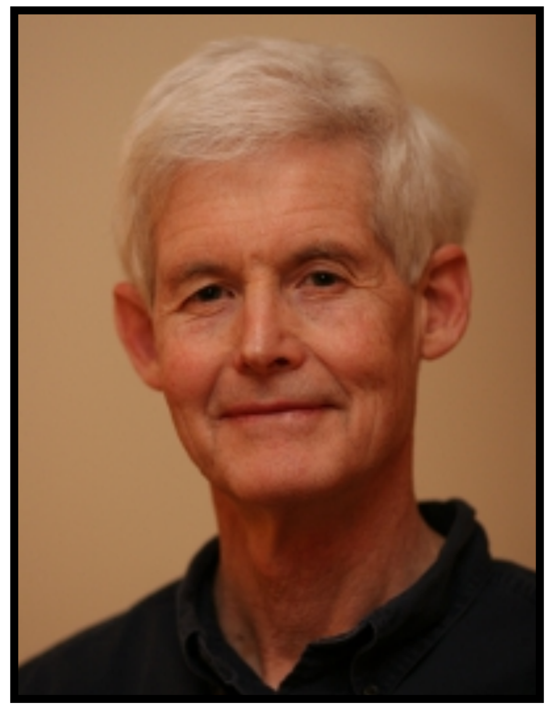
is **NP**-complete?



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

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**, $L \leq^P$ 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

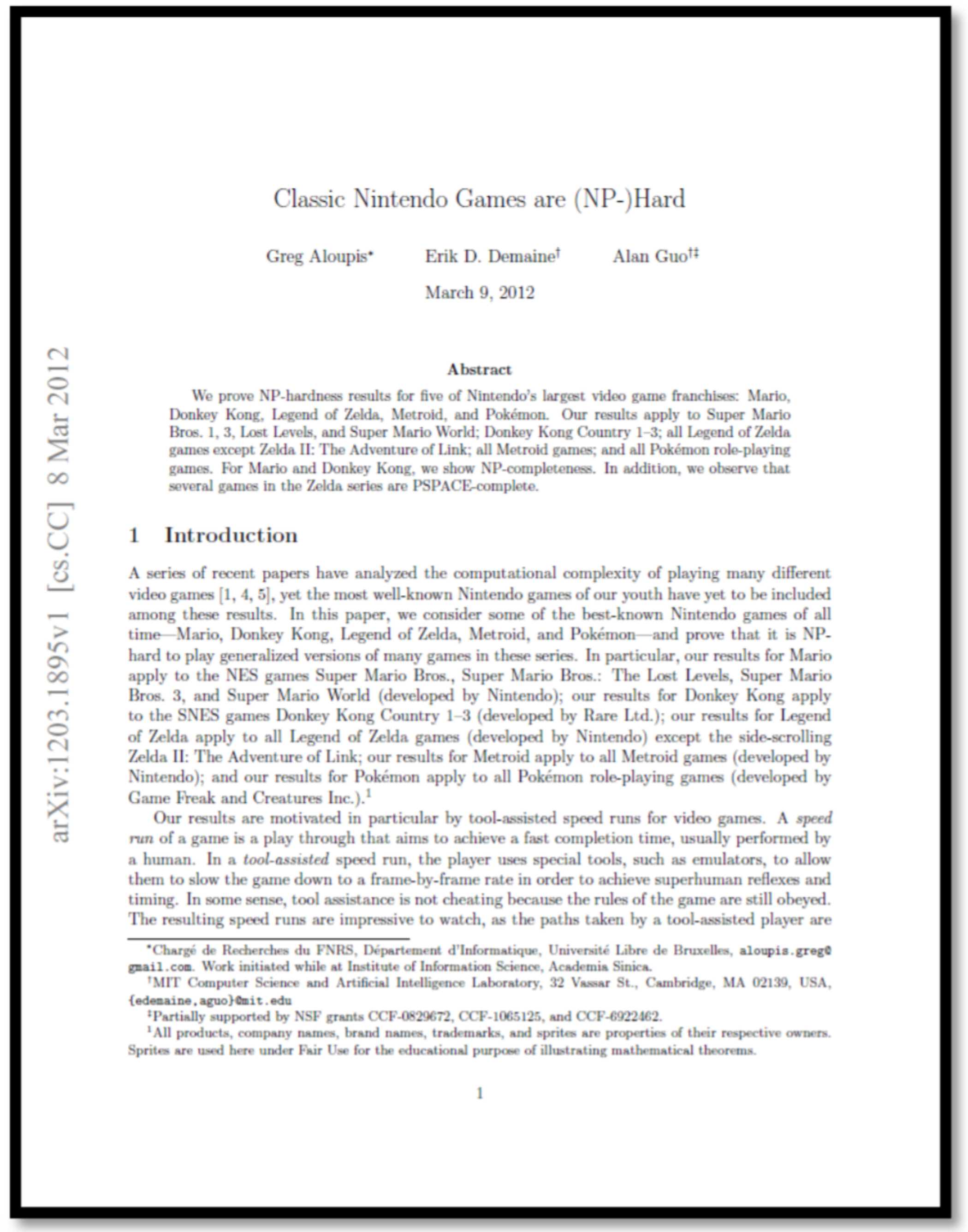
Today

Thousands of problems are known to be **NP**-complete.

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

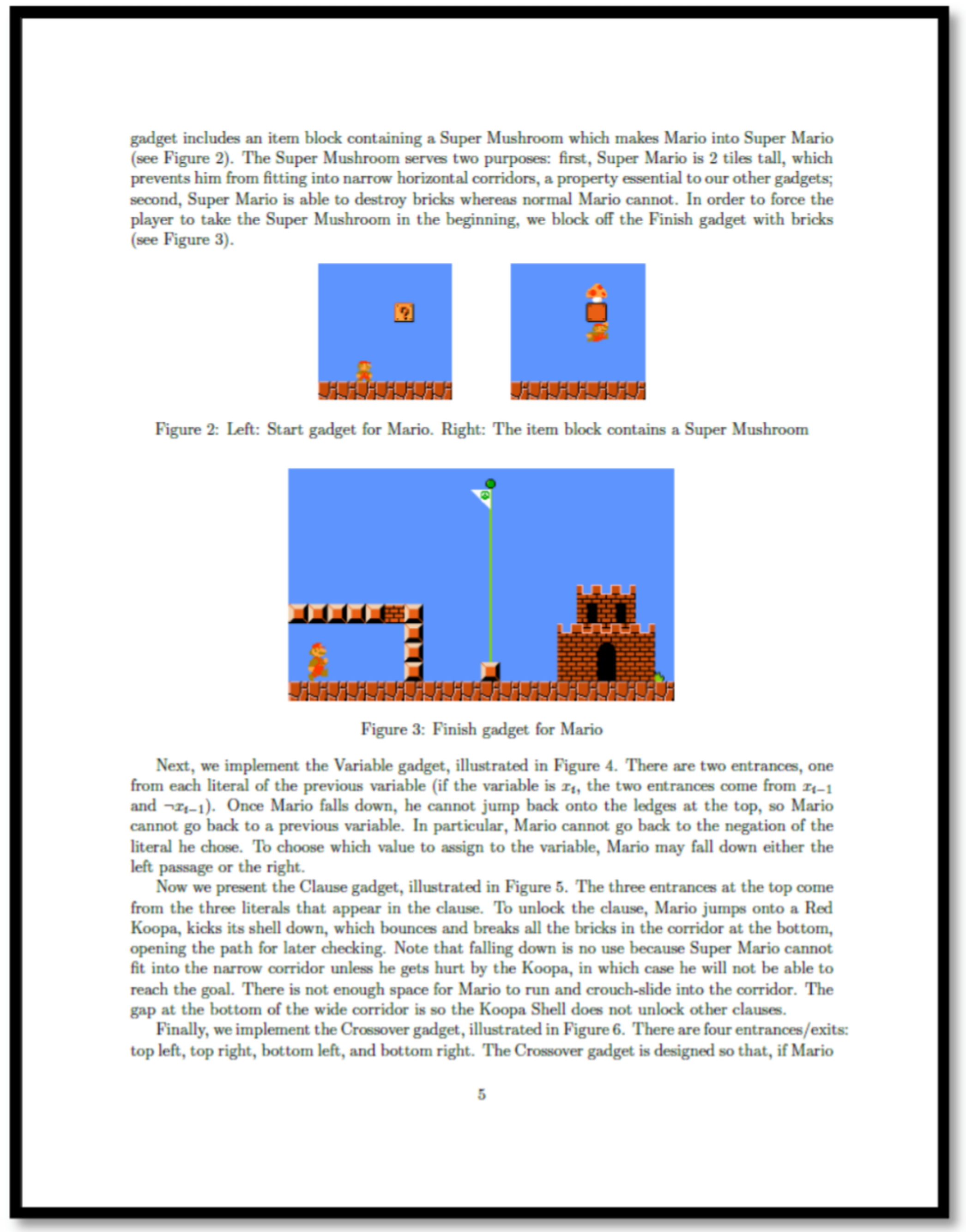
*Chargé de Recherches du FNRS, Département d'Informatique, Université Libre de Bruxelles, aloupis.greg@gmail.com. Work initiated while at Institute of Information Science, Academia Sinica.

†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

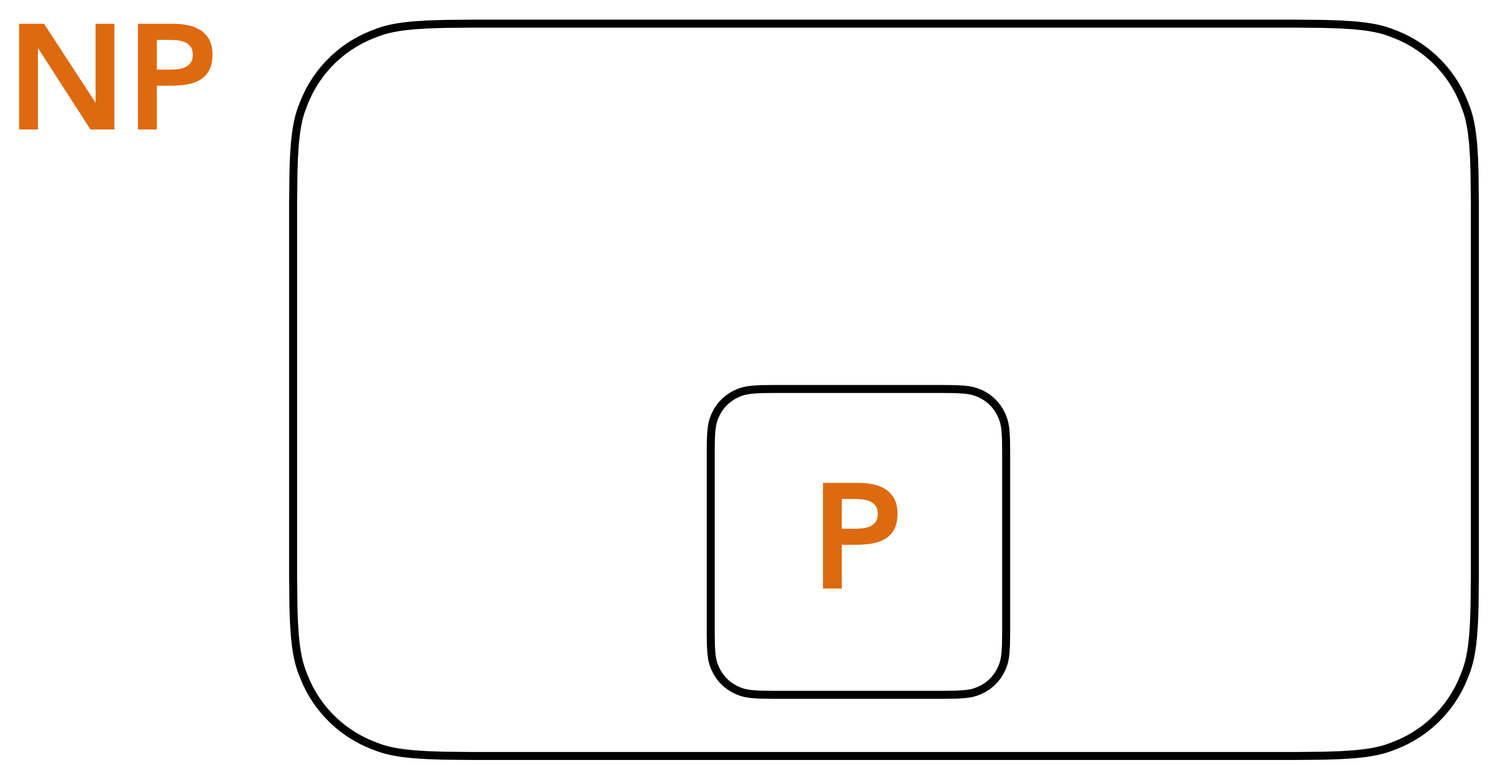
5

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 ?!?



$\leq^P$ SAT

How did Karp do it ?!?

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

(transitivity of $\leq^P$)

topics of next lecture

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**.

? Is **P** $\neq$ **NP**???

The P vs NP question

After decades of research:

Pretty confident this is one of the deepest questions we have ever asked.

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 **Nondeterministic Polynomial time**.

(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"

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

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