





Introduction to Complexity Theory: P and NP



Techniques and Paradigms in this Course

- Divide-and-conquer, greed, dynamic programming, the power of randomness
 Problems that are **easy** for a computer
- Computability  Problems that are **impossible** for a computer
- **NP-completeness** and approximation algorithms
- Cryptography  Problems that are "**probably hard**" for a computer
 Using "probably hard" problems for our benefit (hiding secrets)

The Complexity Class P

Definition: P is the set of all decision problems that can be decided in **polynomial time**.

Polynomial time = $O(n^k)$ for some constant k
E.g. $O(n)$, $O(n^2)$, $O(n^{100})$, etc.

Exponential vs. Polynomial



The λ -O-Matic performs 10^{11} operations/sec

	n=10	n=35	n=60	n=85
n^2	100 < 1 sec	1225 < 1 sec	3600 < 1 sec	7225 < 1 sec
n^3	1000 < 1 sec	43k < 1 sec	216k < 1 sec	614k < 1 sec
2^n	1024 < 1 sec	34×10^9 < 1 sec		

"Efficient": running time polynomial in input size

Why Polynomial?

- “For practical purposes the difference between algebraic [polynomial] and exponential order is often more crucial than the difference between finite and non-finite.”
 - Jack Edmonds
(defined complexity class P, 1965)
- Model-independent: A problem is solvable in polynomial time on a TM if and only if it is solvable in polynomial time any computer.
- Composable: If I run a polynomial number of polynomial-time algorithms, it's still polynomial.



We're still talking about decision problems

Optimization Problem

- **Shortest path:** Given a graph and vertices s, t in the graph, what is the length of the shortest path from s to t ?
- **Longest Common Subsequence:** Given a pair of strings, how long is the longest common subsequence?

Decision Problem

- **Shortest path:** Given a graph, vertices s, t in the graph, and a **budget b** , is the length of the shortest path from s to t **at most b** ?
- **Longest Common Subsequence:** Given a pair of strings and a **budget b** , is the longest common subsequence **at least b** ?

Solving optimization problems using decision problems

To solve the **optimization problem** (with $\text{poly}(n)$ size output), perform **binary search** using calls to the algorithm for the **decision problem**.

E.g. Suppose length of shortest path = 10

“Is there a shortest path of length at most 1?” “No”

...at most 2?” “No.”

...at most 4?” “No.”

...at most 8?” “No.”

...at most 16?” “Yes.”

(halfway between 8 and 16)

...at most 12?” “Yes.”

(halfway between 12 and 8)

...at most 10?” “Yes.”

(halfway between 8 and 10)

...at most 9?” “No.”

⇒ Answer is 10.

Number of calls to decider: $O(\log(\text{range of possible solution values}))$.

Output size of optimizer: $\Omega(\log(\text{range of possible solution values}))$.

⇒ If decider is polynomial time, then so is this optimization algorithm.

An NP-complete (“provably probably hard”) problem:

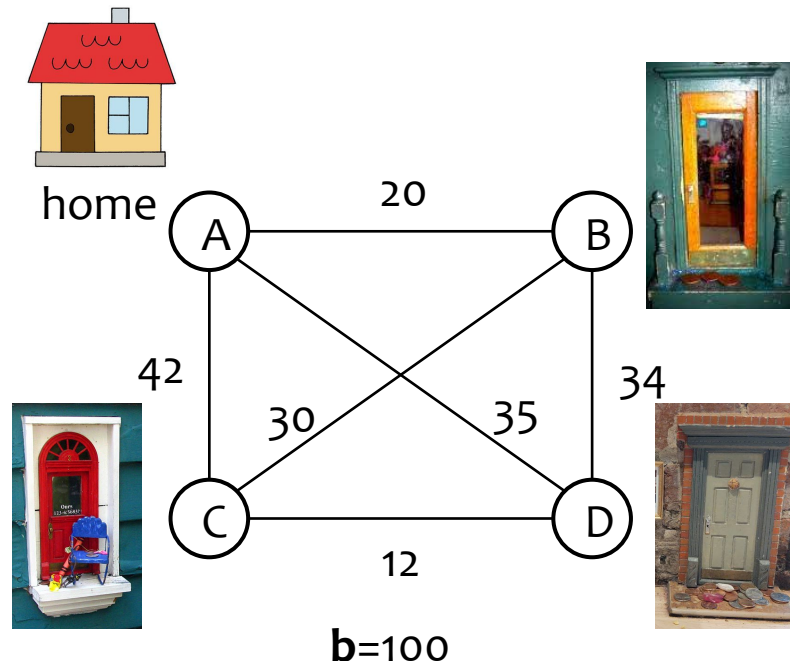
Traveling Salesperson Problem (TSP)

Suppose I want to visit all the fairy doors in Ann Arbor. What is the shortest tour to visit them all and return home?

Input: n vertices, distances between each pair of vertices, budget b

Output: Is there a length $\leq b$ cycle containing every vertex exactly once?

Greedy algorithm?



An NP-complete (“provably probably hard”) problem:

Traveling Salesperson Problem (TSP)

Suppose I want to visit all the fairy doors in Ann Arbor. What is the shortest tour to visit them all and return home?

Input: n vertices, distances between each pair of vertices, budget b

Output: Is there a length $\leq b$ cycle containing every vertex exactly once?

Dynamic programming?

An NP-complete (“provably probably hard”) problem:

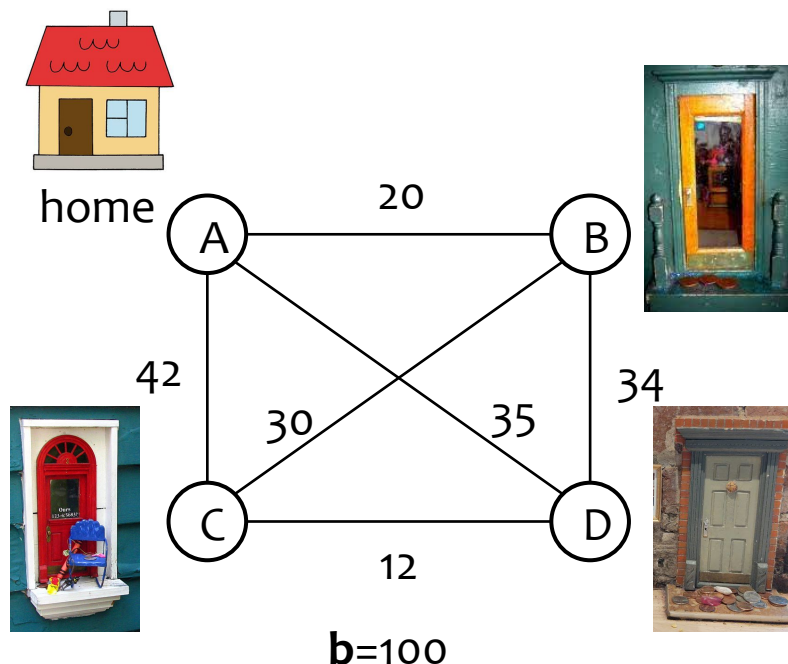
Traveling Salesperson Problem (TSP)

Suppose I want to visit all the fairy doors in Ann Arbor. What is the shortest tour to visit them all and return home?

Input: n vertices, distances between each pair of vertices, budget b

Output: Is there a length $\leq b$ cycle containing every vertex exactly once?

Brute force?



If there's a polynomial time algorithm for TSP, then there's one for tens of thousands of other open problems!



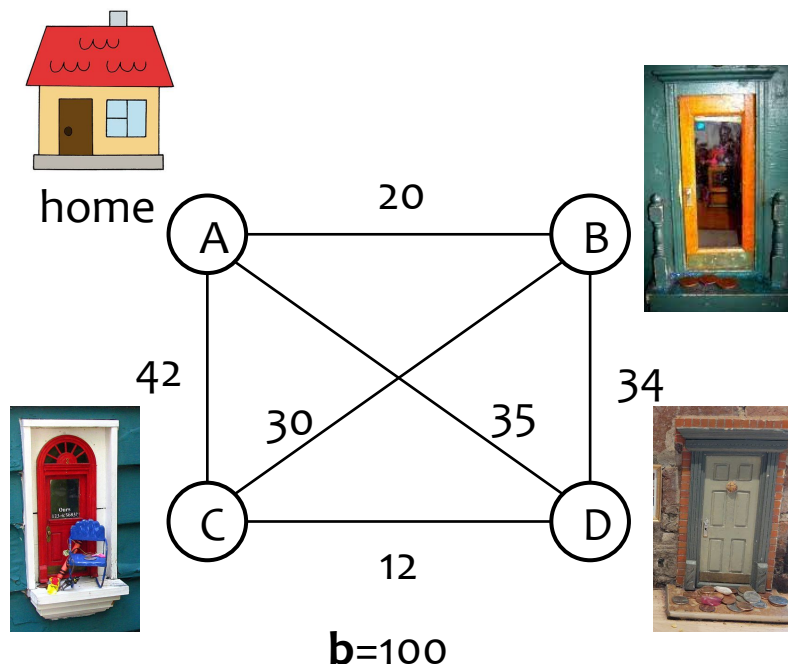
An NP-complete (“provably probably hard”) problem:

Traveling Salesperson Problem (TSP)

Input: n vertices, distances between each pair of vertices, budget b

Output: Is there a length $\leq b$ cycle containing every vertex exactly once?

Notice that TSP is **efficiently verifiable**: if you give me a solution to TSP, I can verify in polynomial time that it’s indeed a solution.



The Complexity Class **NP**

Definition: **NP** is the set of all decision problems whose solution can be **verified** in polynomial time.

Claim: Every problem in **P** is also in **NP**. (You will show on HW)

NP stands for “Nondeterministic Polynomial”

Common misconception: NP does **not** stand for “Not Polynomial”

“A better name would have been **VP: verifiable** in polynomial time.”

-Clyde Kruskal

The Complexity Class NP

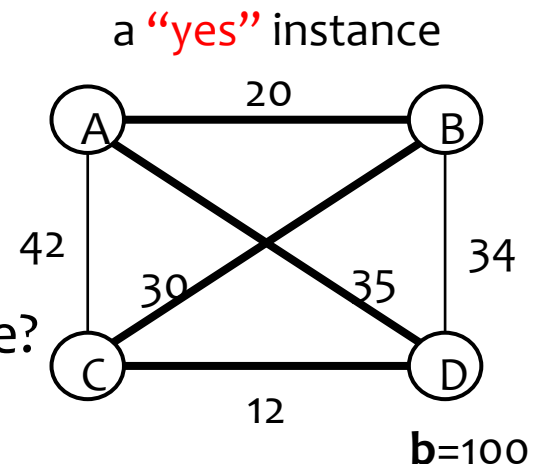
Definition: NP is the set of all decision problems whose solution can be **verified** in polynomial time.

Formally: A problem **L** is in NP if there exists a poly-time algorithm **Verify-L**:

- **Verify-L input:** instance **x** of the problem **L** and a “**certificate**” **C**
- **Verify-L output:**
 - If **x** is a “yes” instance: There **exists** **C** such that **Verify-L(x, C)** accepts
 - If **x** is a “no” instance: **Verify-L(x, C)** rejects **for every C**

Example: TSP:

- **Certificate C:** Length $\leq b$ cycle with all vertices.
- Polynomial-time algorithm **Verify-TSP($\langle G, b \rangle, C$)**:
 - Is **C** a cycle in **G**?
 - Does **C** contain every vertex in **G** exactly once?
 - Do the edge weights of **C** add up to $\leq b$?



The Complexity Class NP

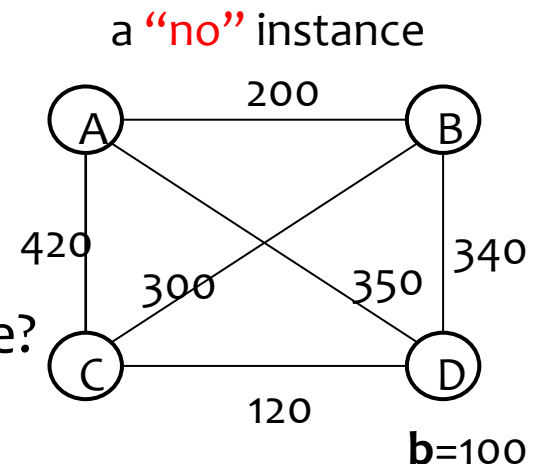
Definition: NP is the set of all decision problems whose solution can be **verified** in polynomial time.

Formally: A problem **L** is in NP if there exists a poly-time algorithm **Verify-L**:

- **Verify-L input:** instance **x** of the problem **L** and a “**certificate**” **C**
- **Verify-L output:**
 - If **x** is a “yes” instance: There **exists** **C** such that **Verify-L(x, C)** accepts
 - If **x** is a “no” instance: **Verify-L(x, C)** rejects **for every C**

Example: TSP:

- **Certificate C:** Length $\leq b$ cycle with all vertices.
- Polynomial-time algorithm **Verify-TSP($\langle G, b \rangle, C$)**:
 - Is **C** a cycle in **G**?
 - Does **C** contain every vertex in **G** exactly once?
 - Do the edge weights of **C** add up to $\leq b$?



The Complexity Class NP

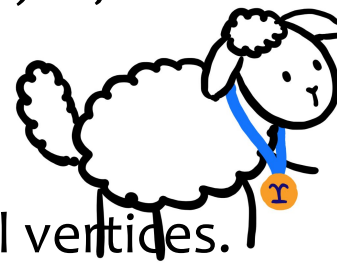
Definition: NP is the set of all decision problems whose solution can be **verified** in polynomial time.

Formally: A problem **L** is in NP if there exists a poly-time algorithm **Verify-L**:

- **Verify-L input:** instance **x** of the problem **L** and a “**certificate**” **C**
- **Verify-L output:**
 - If **x** is a “yes” instance: There **exists** **C** such that **Verify-L(x, C)** accepts
 - If **x** is a “no” instance: **Verify-L(x, C)** rejects **for every C**

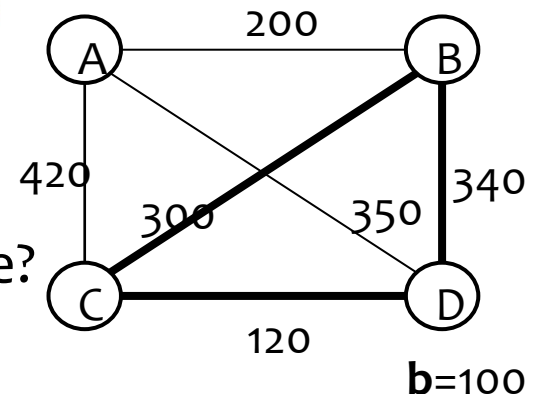
Example: TSP:

- **Certificate C:** Length $\leq b$ cycle with all vertices.
- Polynomial-time algorithm **Verify-TSP($\langle G, b \rangle, C$)**:
 - Is **C** a cycle in **G**?
 - Does **C** contain every vertex in **G** exactly once?
 - Do the edge weights of **C** add up to $\leq b$?



tada!

a “no” instance



To show that a problem is in **NP**, you need to specify (e.g. for the HW):

1. Certificate
2. Verification algorithm
3. Proof of correctness of verification algorithm
4. Proof that verification algorithm runs in polynomial time

Example: TSP:

1. **Certificate C**: Length $\leq b$ cycle containing every vertex.
2. Algorithm **Verify-TSP**($\langle G, b \rangle, C$):
 - Is **C** a cycle in **G**?
 - Does **C** contain every vertex in **G** exactly once?
 - Do the edge weights of **C** add up to $\leq b$?
 - Accept if all 3 answers are “yes”
3. TSP “yes” instance \Rightarrow exists a length $\leq b$ cycle containing every vertex
 \Rightarrow using that cycle as the certificate makes **Verify-TSP** accept

TSP “no” instance \Rightarrow no length $\leq b$ cycle containing every vertex
 \Rightarrow no certificate makes **Verify-TSP** accept
4. Checking those 3 questions takes poly time (you’d need some more detail)

The Complexity Class NP

Claim: We can assume without loss of generality that the certificate is of polynomial length. Why?

Definition reminder:

Definition: **NP** is the set of all problems whose solution can be **verified** in polynomial time by a TM.

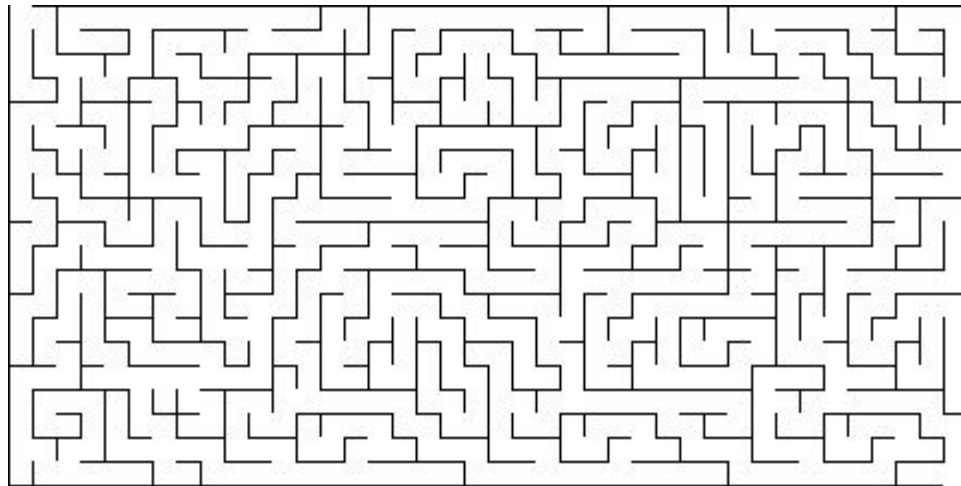
Formally: A problem **L** is in **NP** if there exists a poly-time algorithm **Verify-L**:

- **Verify-L input:** instance **x** of the problem **L** and a “**certificate**” **C**
- **Verify-L output:**
 - If **x** is a “**yes**” instance: There **exists** **C** such that **Verify-L(x, C)** accepts
 - If **x** is a “**no**” instance: **Verify-L(x, C)** rejects **for every C**

Verifiable Problems

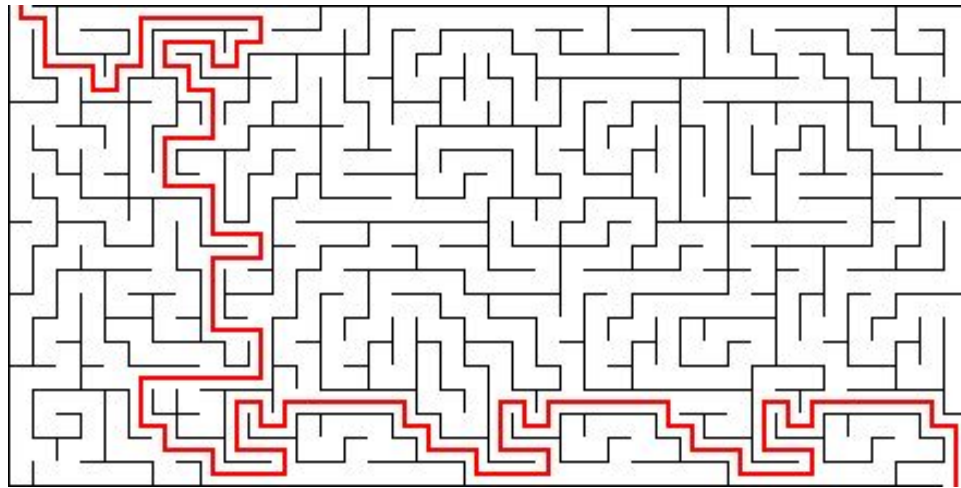
Consider a maze.

It might be hard to solve...



Verifiable Problems

But if you give me the solution, I can verify that it's a valid solution.



Another problem in NP (also happens to be NP-complete):

Subset Sum

Input: a set S of integers and a target t .

Output: Is there a subset of the integers in S whose sum is exactly t ?

To show that Subset Sum is in **NP**, we need to provide a certificate and a polynomial-time verification algorithm:

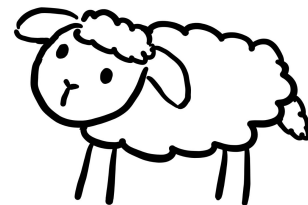
Another problem in NP (also happens to be NP-complete):

Satisfiability (SAT)

This is literally
my favorite
problem

Example SAT instance:

$$\Phi = (x \vee y) \wedge (\neg y \vee x \vee \neg z) \wedge (\neg x \vee (y \wedge \neg z))$$



A Boolean formula is made up of:

- “literals”: variables and their negations (e.g. x , y , z , $\neg x$, $\neg y$, $\neg z$)
- OR: \vee
- AND: \wedge

Input: A Boolean formula Φ

Output: Is the formula Φ *satisfiable*? That is, does there exist a true/false assignment to the variables that makes the entire formula true?

To show that SAT is in **NP**, we need to provide a certificate and a polynomial-time verification algorithm:

THE Major Open Problem in Computer Science

$$P \stackrel{?}{=} NP$$

“Is every efficiently verifiable problem also efficiently solvable?”

“If $P = NP$, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in ‘creative leaps’, no fundamental gap between solving a problem and recognizing the solution once it's found.”

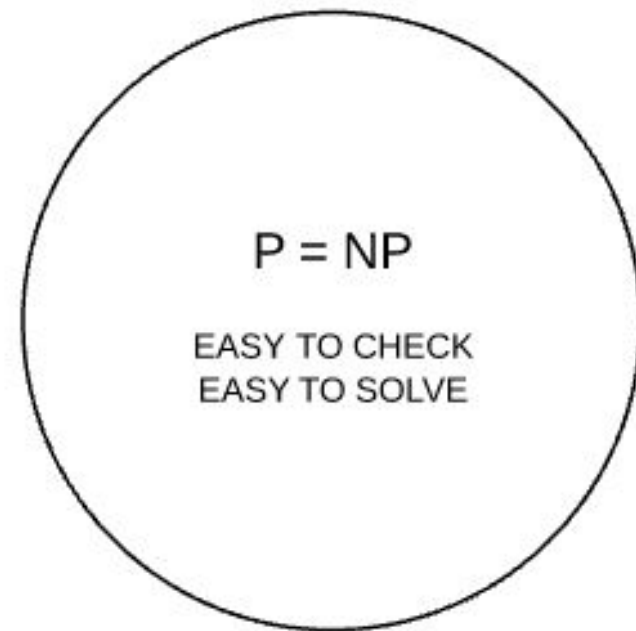
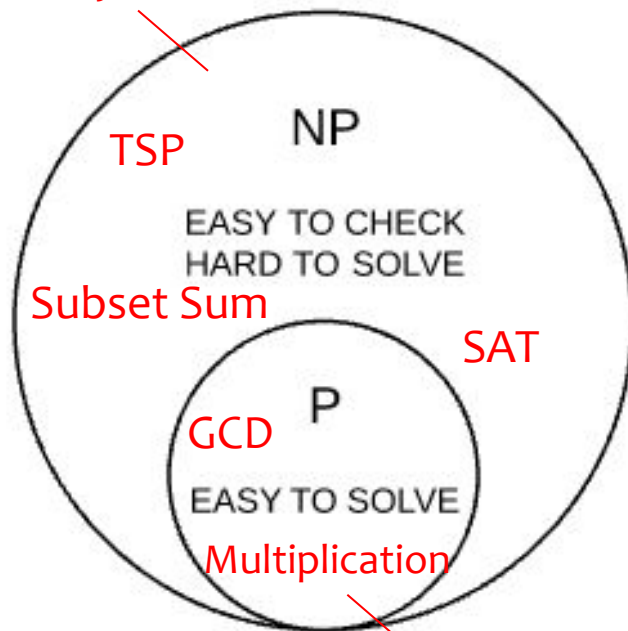
- Scott Aaronson

Two Possible Worlds

Believed: $P \neq NP$

$P = NP$

many problems you'll see next week



decision versions of many problems you've solved in this course

**\$1 million
reward**

Millennium Problems

Yang-Mills and Mass Gap

Experiment and computer simulations suggest the existence of a "mass gap" in the solution to the quantum versions of the Yang-Mills equations. But no proof of this property is known.

Riemann Hypothesis

The prime number theorem determines the average distribution of the primes. The Riemann hypothesis tells us about the deviation from the average. Formulated in Riemann's 1859 paper, it asserts that all the "non-obvious" zeros of the zeta function are complex numbers with real part $1/2$.

P vs NP Problem

If it is easy to check that a solution to a problem is correct, is it also easy to solve the problem? This is the essence of the P vs NP question. Typical of the NP problems is that of the Hamiltonian Path Problem: given N cities to visit, how can one do this without visiting a city twice? If you give me a solution, I can easily check that it is correct. But I cannot so easily find a solution.

Navier-Stokes Equation

This is the equation which governs the flow of fluids such as water and air. However, there is no proof for the most basic questions one can ask: do solutions exist, and are they unique? Why ask for a proof? Because a proof gives not only certitude, but also understanding.

Hodge Conjecture

The answer to this conjecture determines how much of the topology of the solution set of a system of algebraic equations can be defined in terms of further algebraic equations. The Hodge conjecture is known in certain special cases, e.g., when the solution set has dimension less than four. But in dimension four it is unknown.

Poincaré Conjecture

In 1904 the French mathematician Henri Poincaré asked if the three dimensional sphere is characterized as the unique simply connected three manifold. This question, the Poincaré conjecture, was a special case of Thurston's geometrization conjecture. Perelman's proof tells us that every three manifold is built from a set of standard pieces, each with one of eight well-understood geometries.

Birch and Swinnerton-Dyer Conjecture

Supported by much experimental evidence, this conjecture relates the number of points on an elliptic curve mod p to the rank of the group of rational points. Elliptic curves, defined by cubic equations in two variables, are fundamental mathematical objects that arise in many areas: Wiles' proof of the Fermat Conjecture, factorization of numbers into primes, and cryptography, to name three.



Simpsons Halloween Special, 2013

NEWS

[Home](#) | [Video](#) | [World](#) | [US & Canada](#) | [UK](#) | [Business](#) | [Tech](#) | [Science](#) | [More](#)[Magazine](#)

Homer Simpson's scary maths problems

By Simon Singh
Science writer

© 31 October 2013 | [Magazine](#)

[Share](#)

For example, in one scene, the letters P and NP can be seen over Homer's right shoulder. Although these three letters would have made no sense to most viewers, they are a deliberate nod towards a statement about one of the most important unsolved problems in theoretical computer science. Indeed, this is such a weighty puzzle that there is a reward of \$1m (£623,000) for whoever solves the mystery.

It is surprising to find a reference to P- and NP-type problems in a television sitcom, but not when the writer is Cohen, because he studied them while doing his master's degree at the University of California in Berkeley.

Why not define NP like this...

“The set of decision problems that can be solved in exponential time.”

NP-Hardness and NP-Completeness

Informal definition: A problem **L** is **NP-hard** if it is at least as hard as EVERY problem in **NP**.

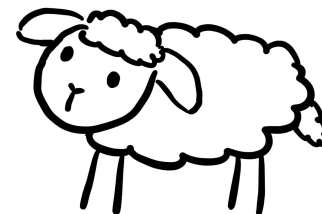
Formal definition: A problem **L** is **NP-hard** if: for EVERY problem **X** in **NP**, $X \leq_p L$.

a new type of reduction!

A problem **L** is **NP-complete** if

- **L** \in **NP**
- **L** is **NP-hard**

A polynomial-time algorithm for any **NP-hard** problem would imply **P = NP**.



Polynomial-time mapping reduction from **A** to **B**

(denoted $\mathbf{A} \leq_p \mathbf{B}$)

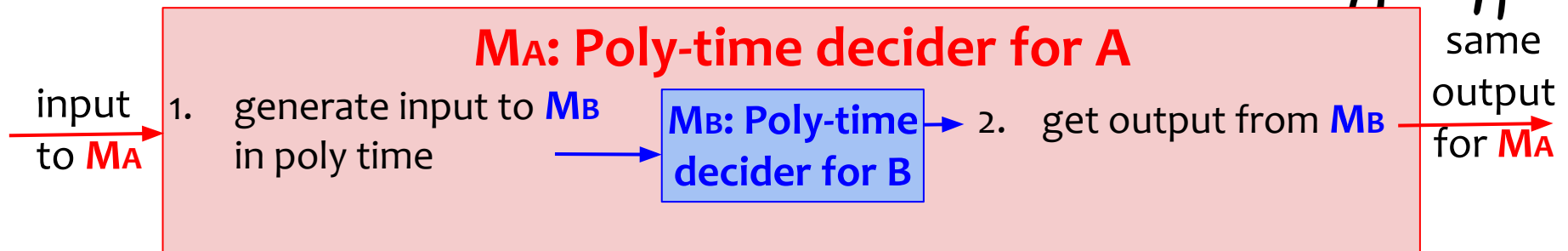
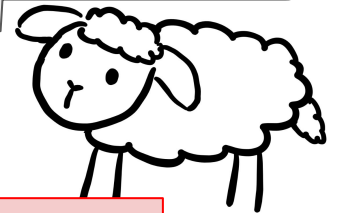
Defn: a poly-time-computable function f such that: $x \in \mathbf{A} \Leftrightarrow f(x) \in \mathbf{B}$.

*“Given any instance of **A**, in polynomial time we can construct an instance of **B** whose yes/no answer is the same.”*

What it implies:

1. If $\mathbf{B} \in \mathbf{P}$ then $\mathbf{A} \in \mathbf{P}$.
2. If **A** is NP-hard then **B** is NP-hard.

In mapping reductions, there's no “flipping” the answer, and only one call to **B**



“Problem **B** is at least as hard as Problem **A**”

Polynomial-time mapping reduction from **A** to **B**

(denoted $\mathbf{A} \leq_p \mathbf{B}$)

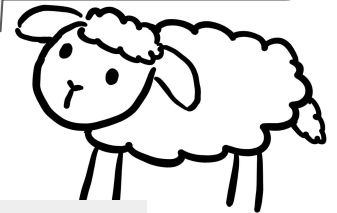
Defn: a poly-time-computable function f such that: $x \in \mathbf{A} \Leftrightarrow f(x) \in \mathbf{B}$.

*“Given any instance of **A**, in polynomial time we can construct an instance of **B** whose yes/no answer is the same.”*

What it implies:

1. If $\mathbf{B} \in \mathbf{P}$ then $\mathbf{A} \in \mathbf{P}$.
2. If \mathbf{A} is NP-hard then \mathbf{B} is NP-hard.

In mapping reductions, there's no “flipping” the answer, and only one call to **B**



Instance of **A**



Instance of **B**
with same answer

fast conversion
machine

“Problem **B** is at least as hard as Problem **A**”

Polynomial-time mapping reduction from **A** to **B**

(denoted $\mathbf{A} \leq_p \mathbf{B}$)

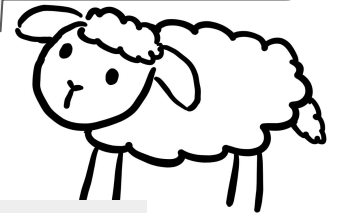
Defn: a poly-time-computable function f such that: $x \in \mathbf{A} \Leftrightarrow f(x) \in \mathbf{B}$.

*“Given any instance of **A**, in polynomial time we can construct an instance of **B** whose yes/no answer is the same.”*

What it implies:

1. If $\mathbf{B} \in \mathbf{P}$ then $\mathbf{A} \in \mathbf{P}$.
2. If \mathbf{A} is NP-hard then \mathbf{B} is NP-hard.

In mapping reductions, there's no “flipping” the answer, and only one call to **B**



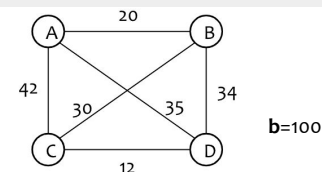
Instance of **A**

$$(x_1 \vee \bar{x}_2 \vee x_4) \wedge (x_2 \vee x_3 \vee \bar{x}_{42})$$



fast conversion
machine

Instance of **B**
with same answer



E.g. SAT Instance

TSP Instance

NP-Hardness and NP-Completeness

Informal definition: A problem **L** is **NP-hard** if it is at least as hard as EVERY problem in **NP**.

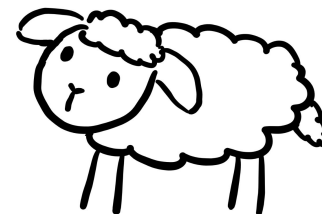
Formal definition: A problem **L** is **NP-hard** if: for EVERY problem **X** in **NP**, $X \leq_p L$.

a new type of reduction!

A problem **L** is **NP-complete** if

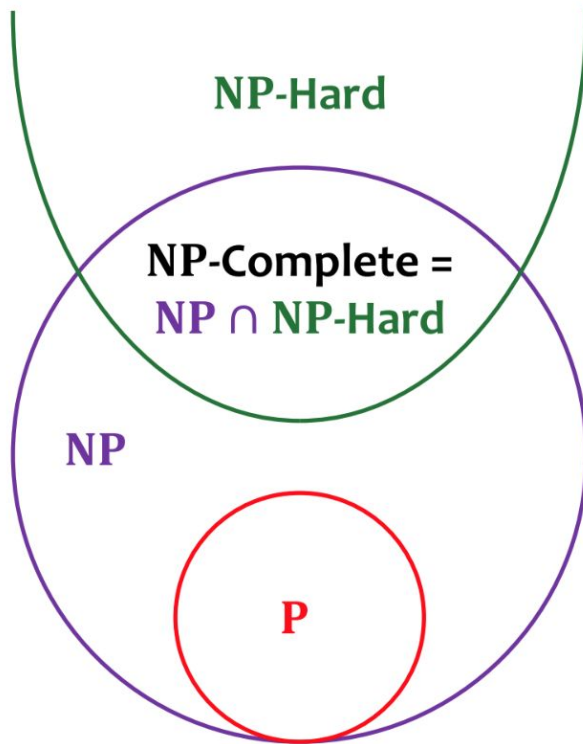
- **L** \in **NP**
- **L** is **NP-hard**

A polynomial-time algorithm for an **NP-hard** problem would imply **P = NP**.

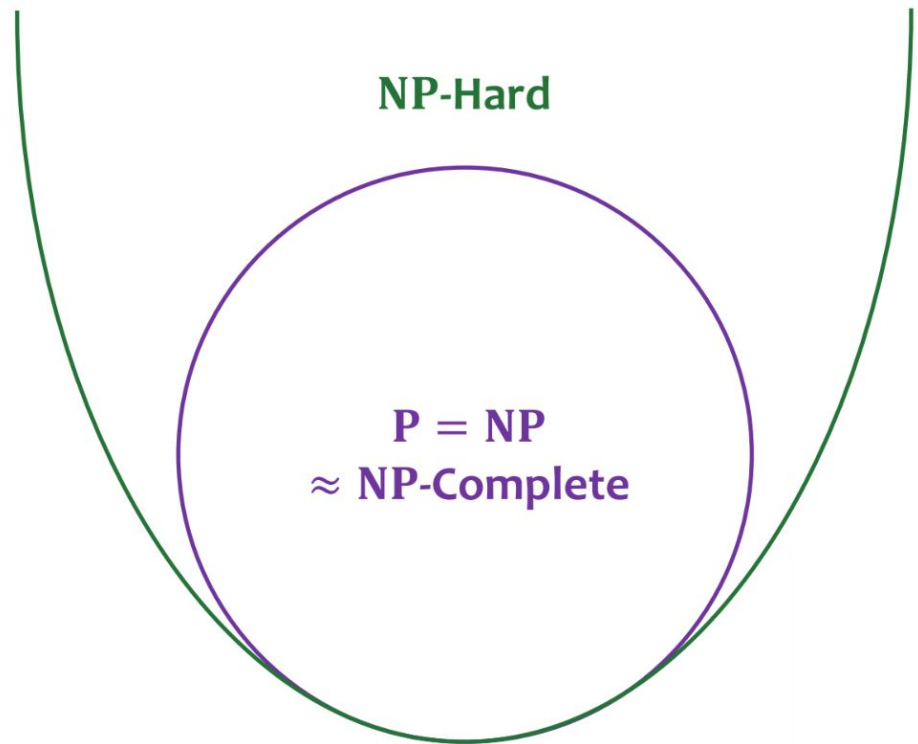


Two Possible Worlds

$P \neq NP$



$P = NP$

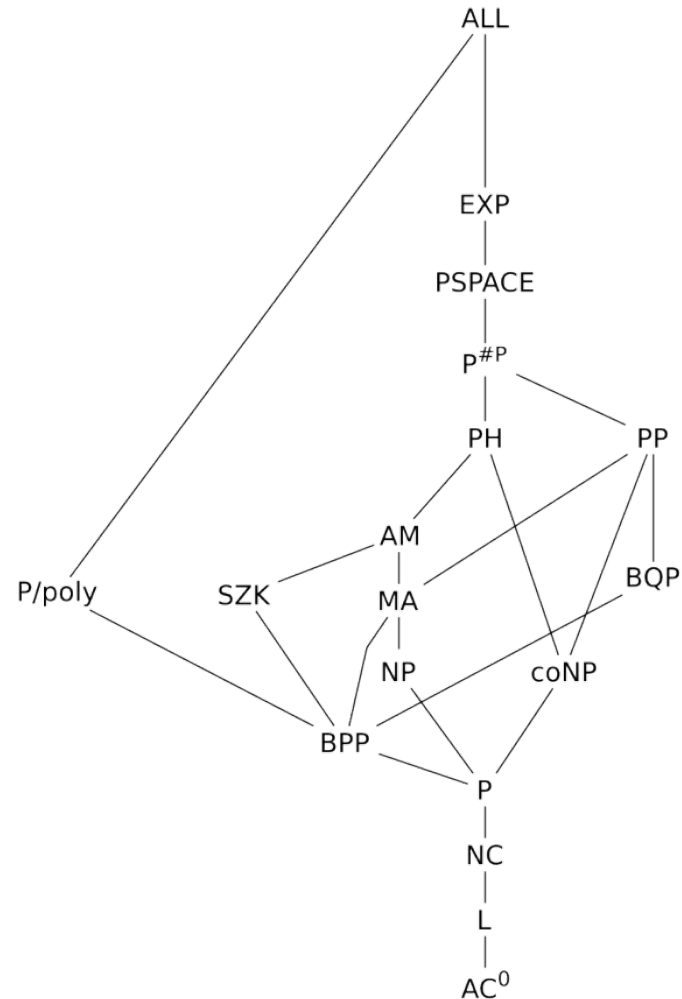
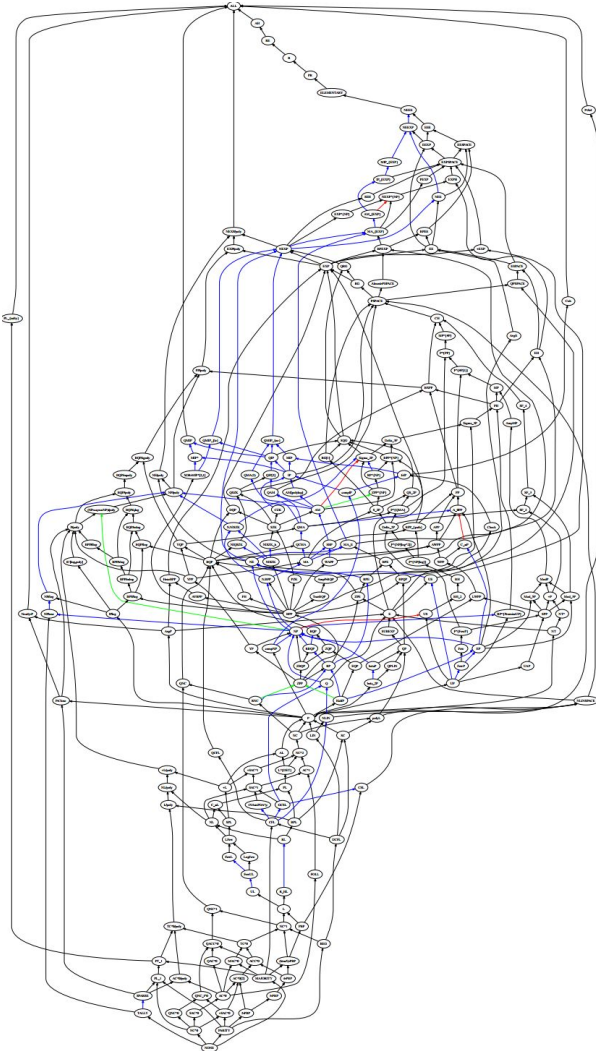


The Complexity Zoo

(a database of 547 complexity classes)



Scott Aaronson,
zookeeper



The Petting Zoo