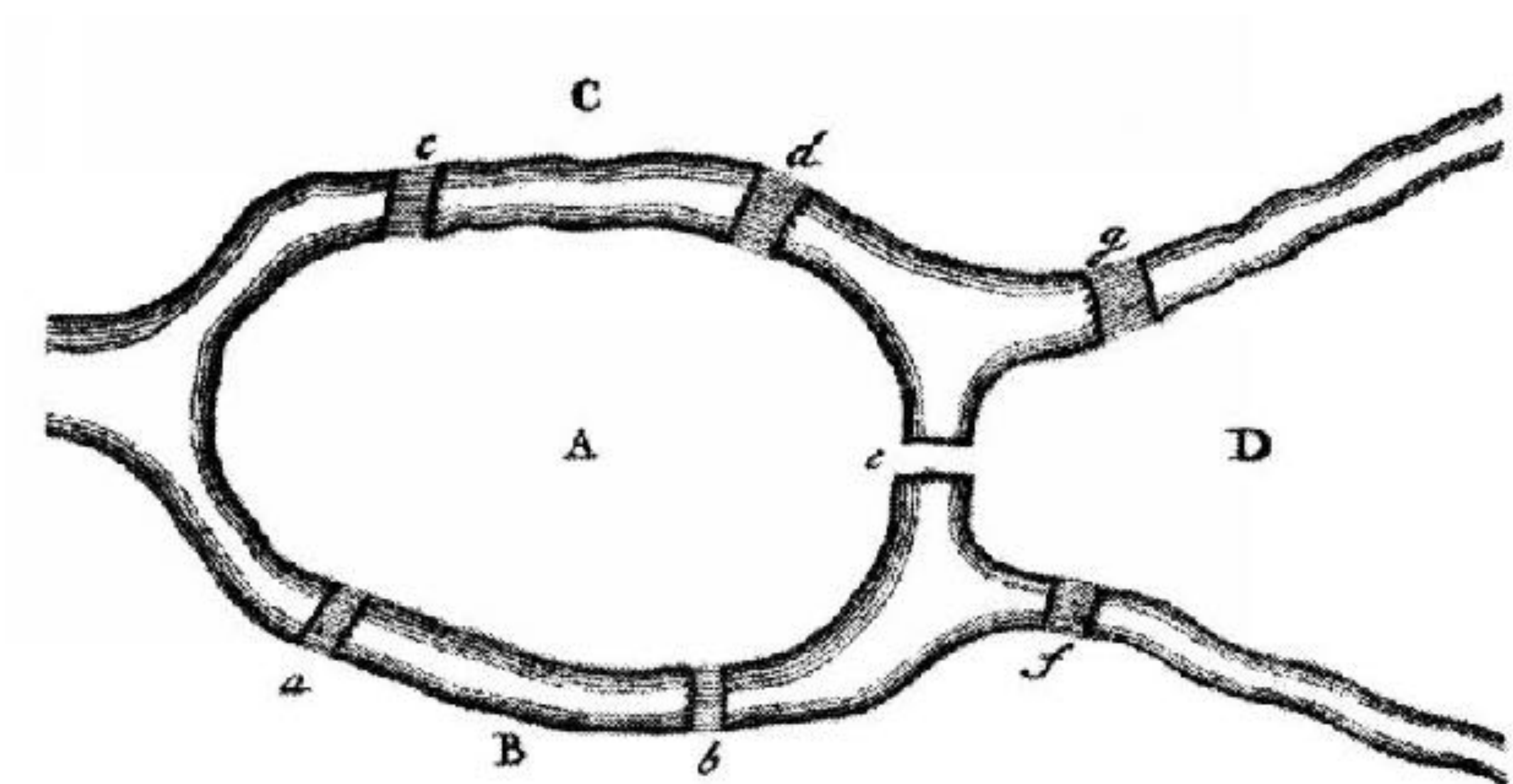
Computational Complexity 1: Algorithms and Landscapes

Cristopher Moore
Santa Fe Institute

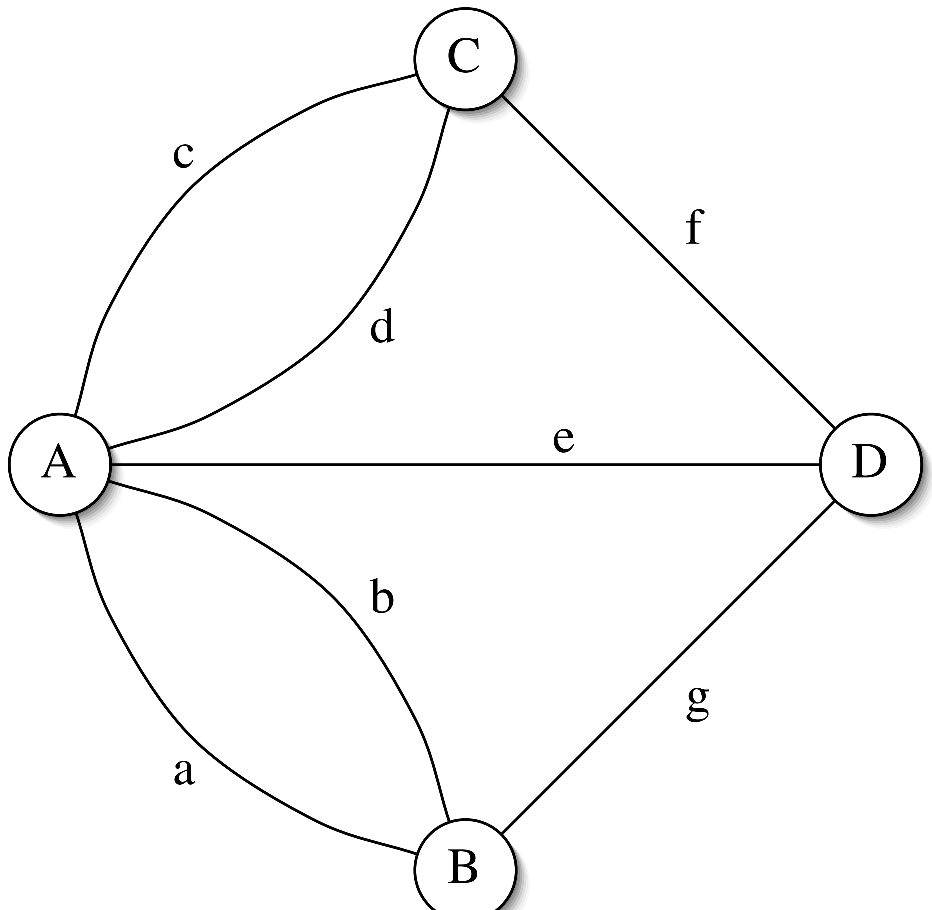
Computational complexity

Why are some problems qualitatively harder than others?



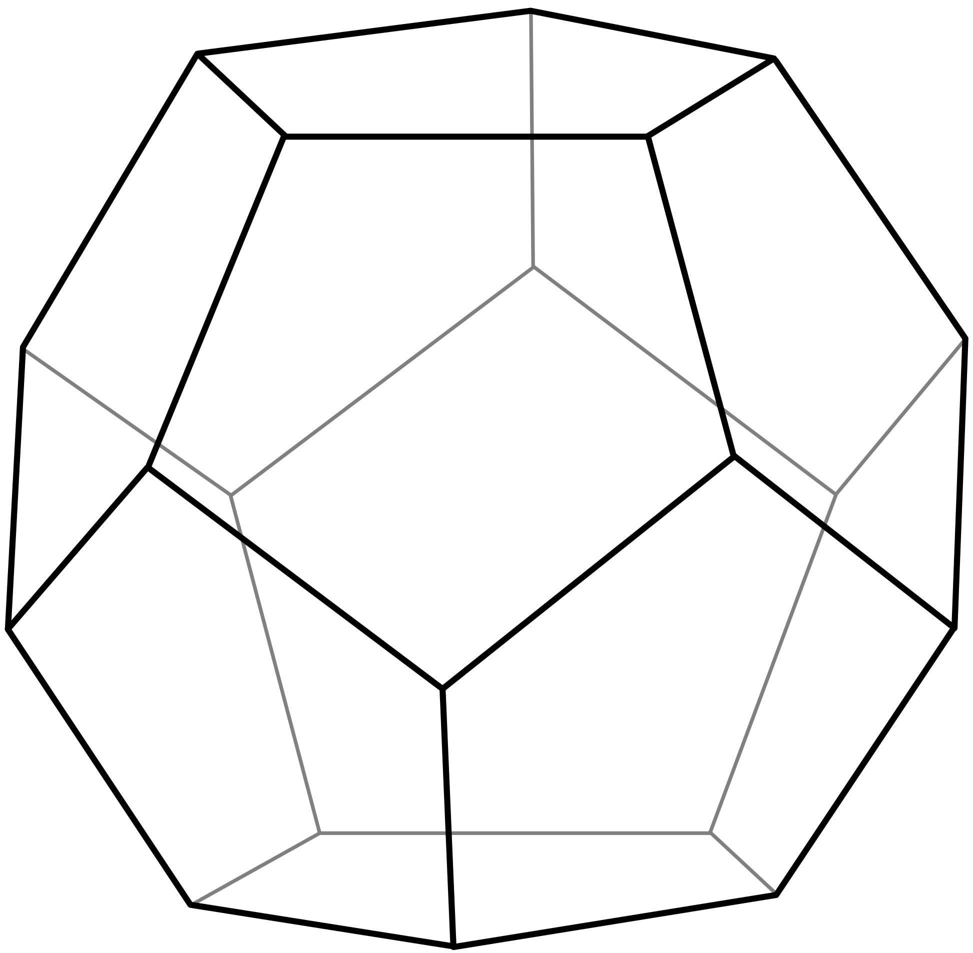
Computational complexity

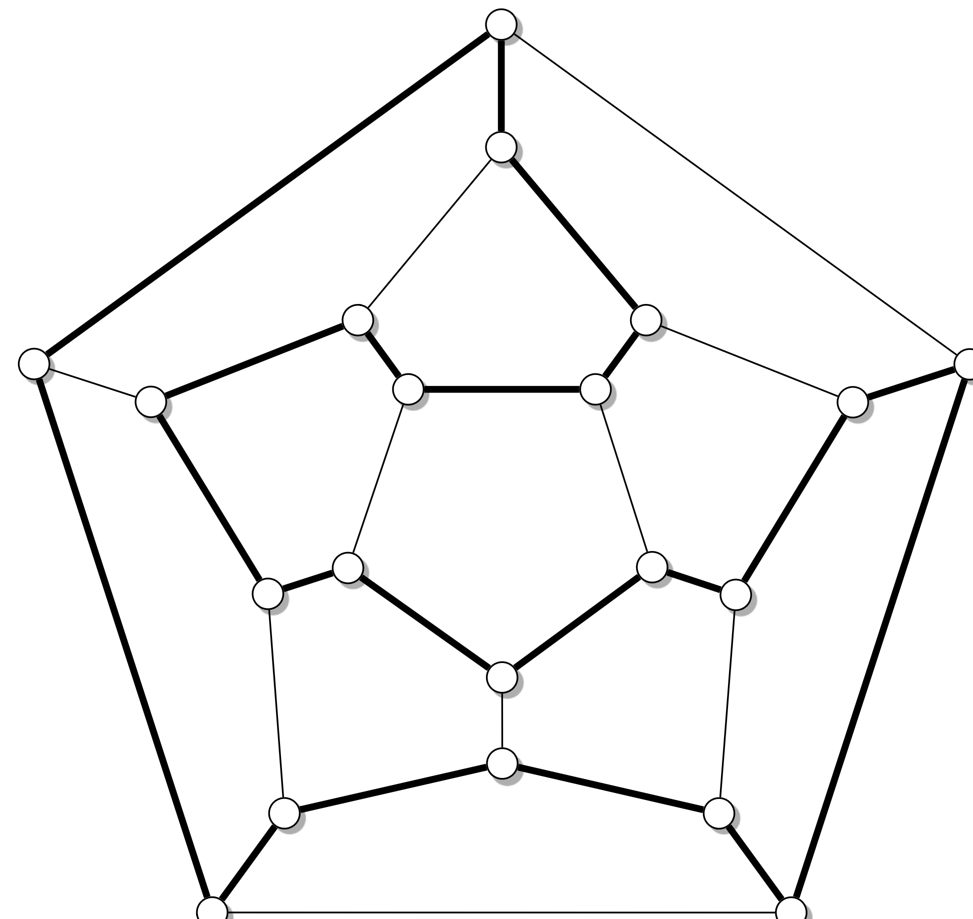
Euler: at most two nodes can have an odd number of bridges, so no tour is possible!



Computational complexity

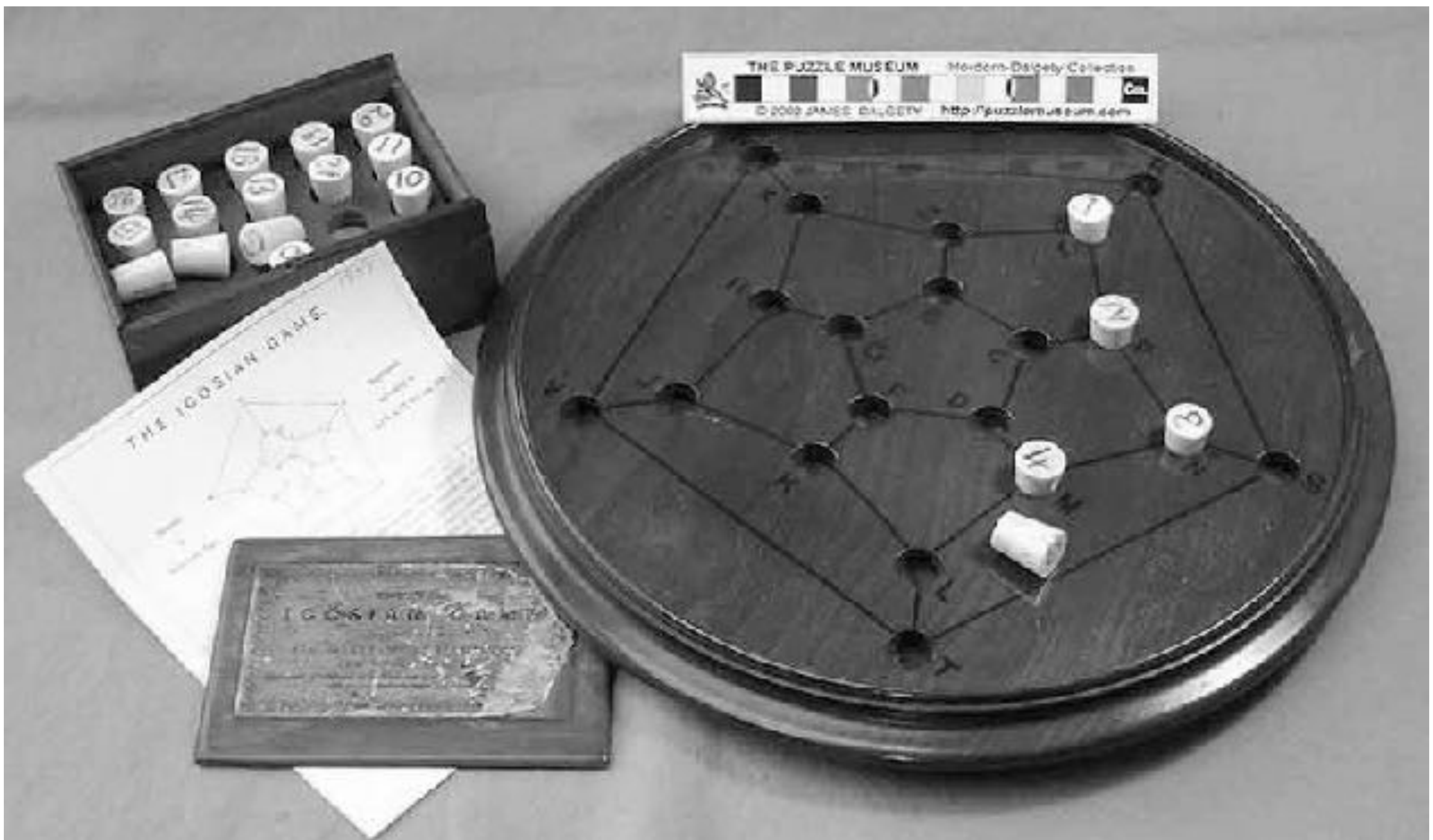
What if we want to visit every vertex, instead of every edge?





Computational complexity

As far as we know, the only way to solve this problem is (essentially) exhaustive search!

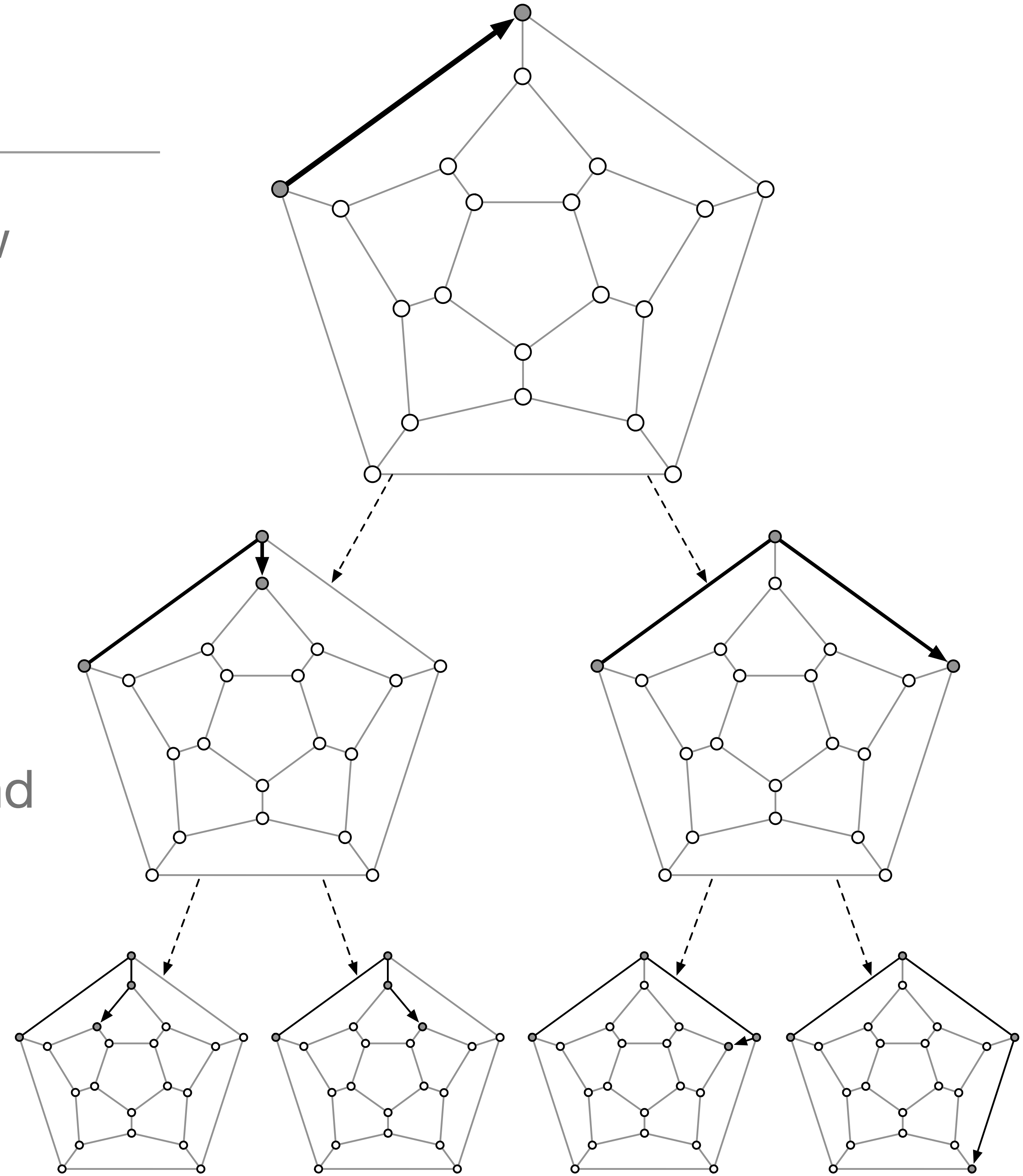


An exponential tree

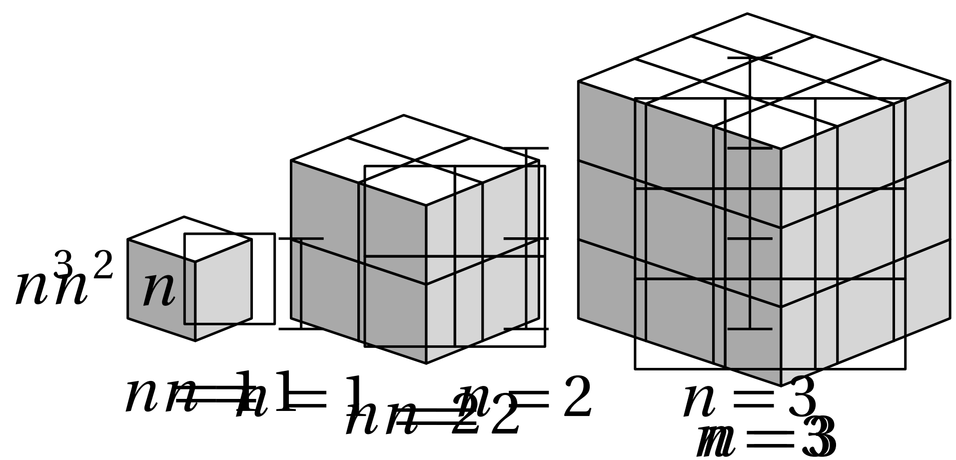
Backtracking search: follow a path until you get stuck, then backtrack to your last choice

If there are *n* nodes, this could take 2^n time

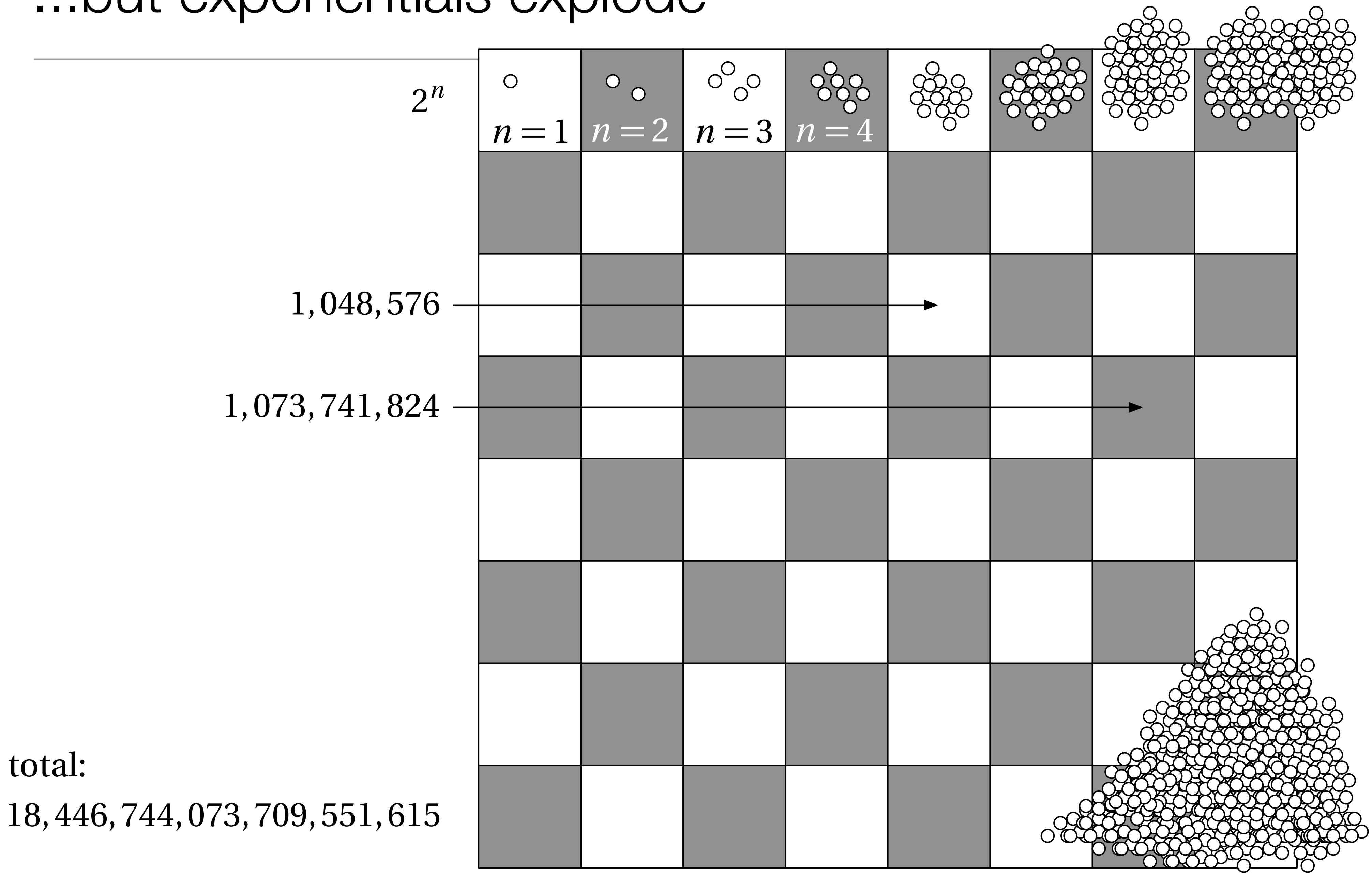
When can we avoid this kind of search?



Polynomials don’t grow too badly as n grows...



...but exponentials explode



Faster computers, bigger problems?

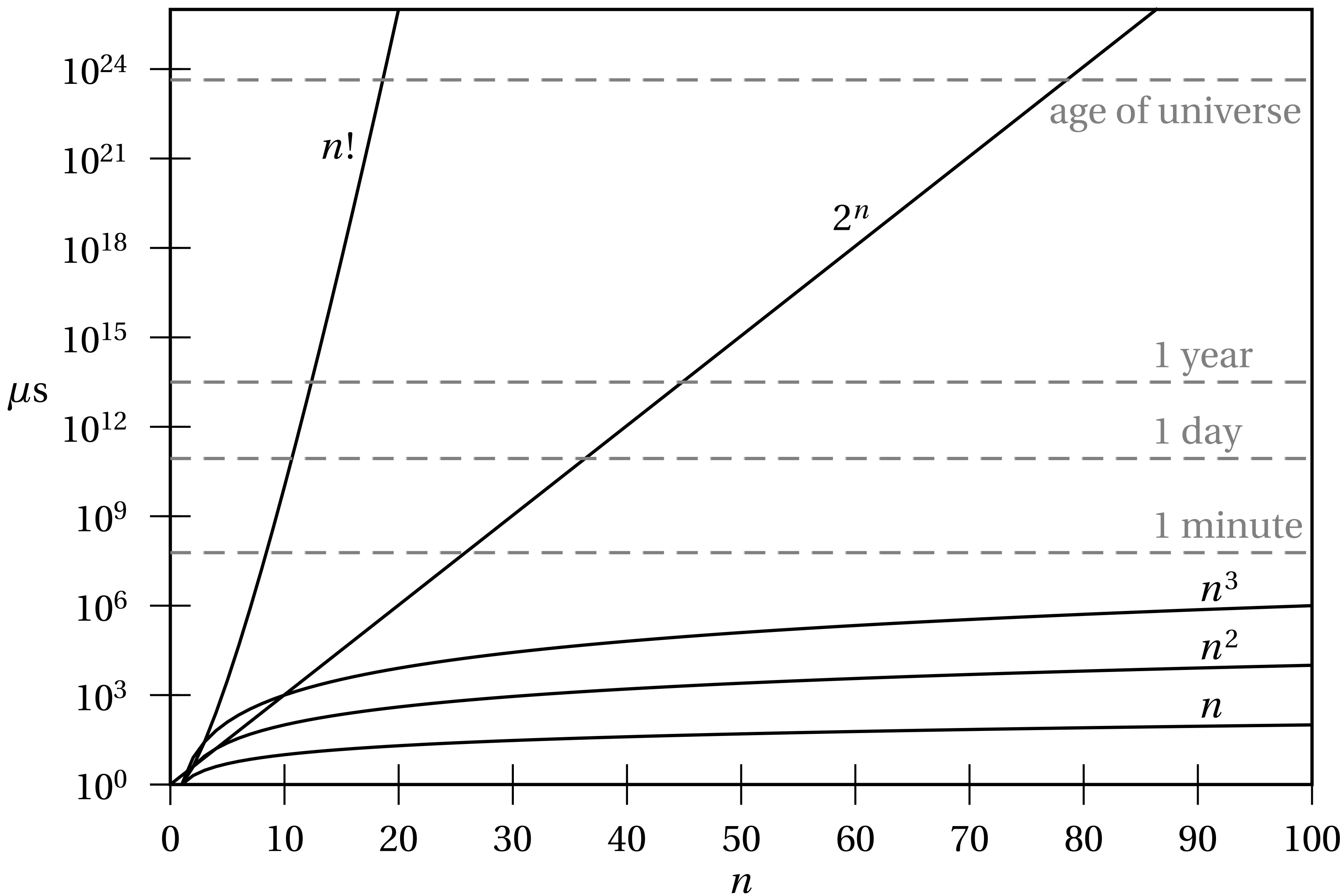
Say you can do T steps in a week with your current computer...

According to Moore's law, next year this will be $2T$

If T=O(n^2), then doubling T multiplies n by $\sqrt{2}$

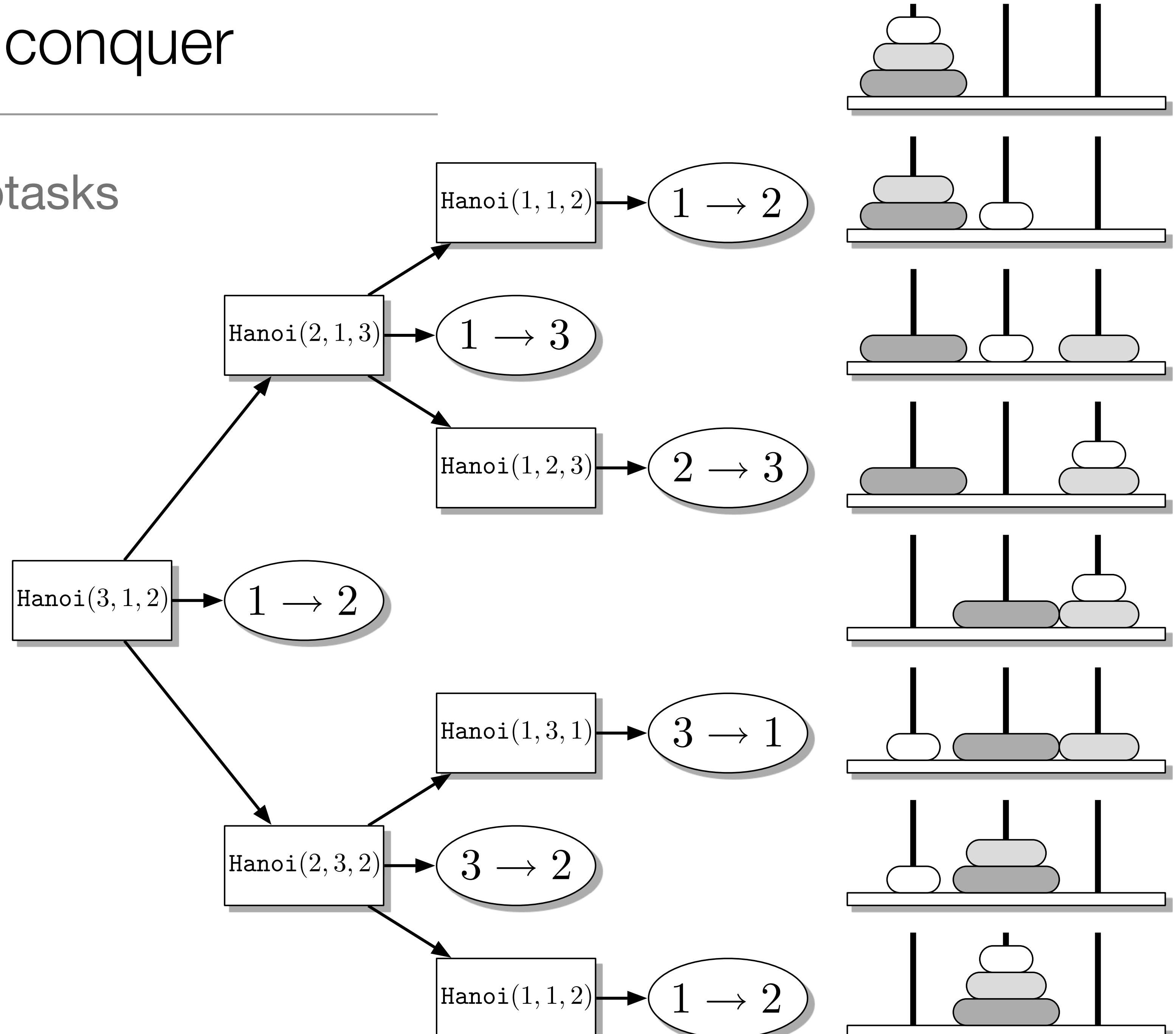
But if T=O(2^n), doubling T just changes n to n+1

Until the end of the world

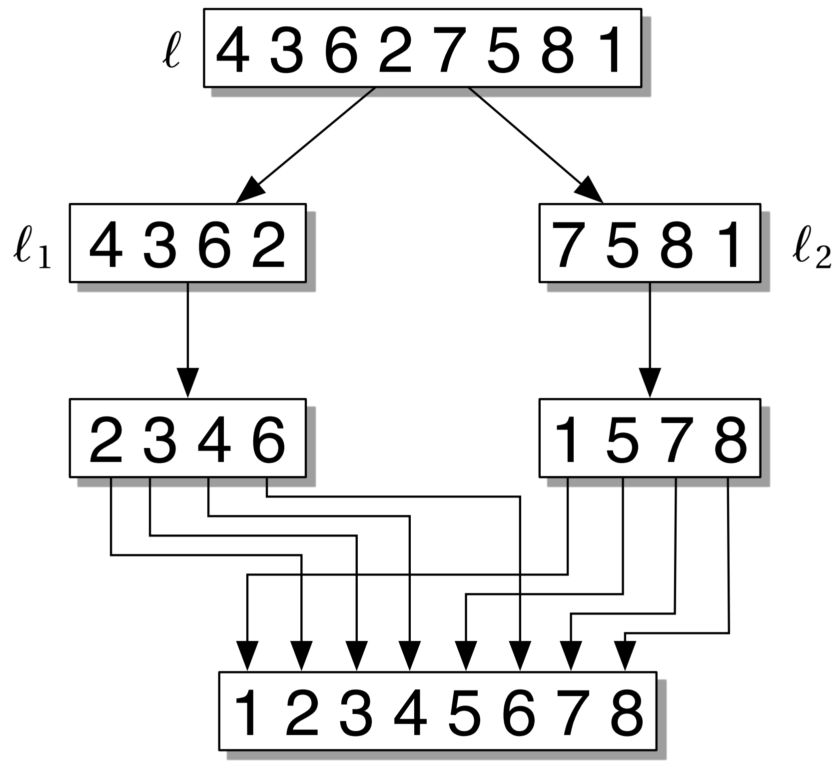


Divide and conquer

Tasks and subtasks



Divide and conquer: mergesort

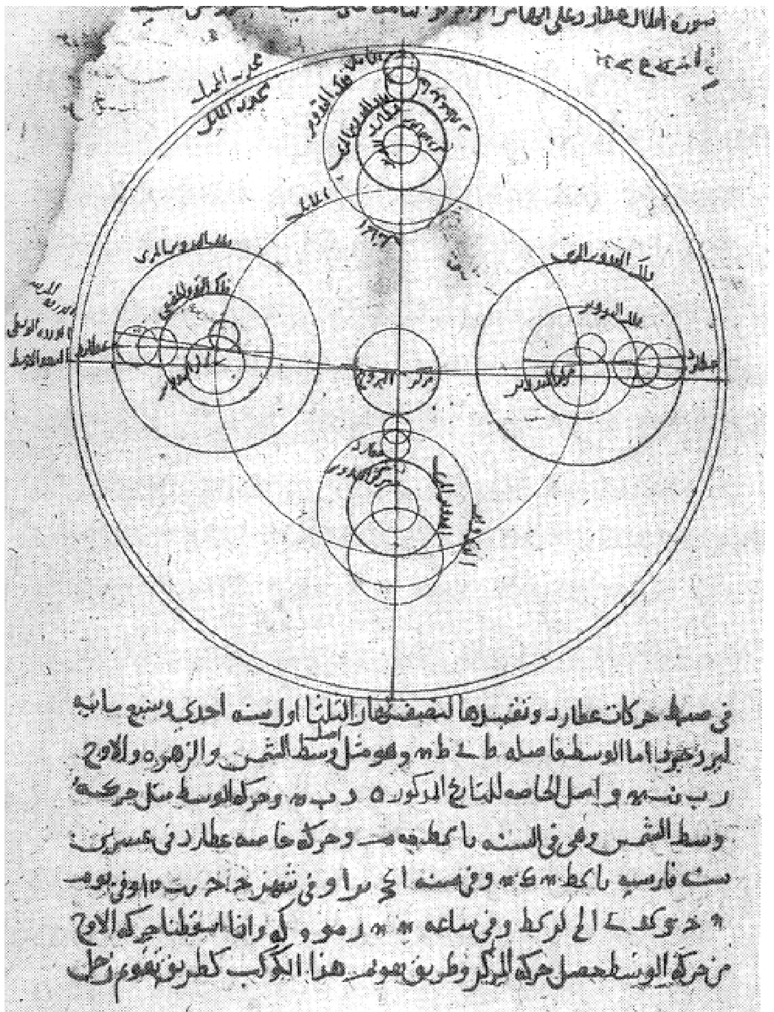


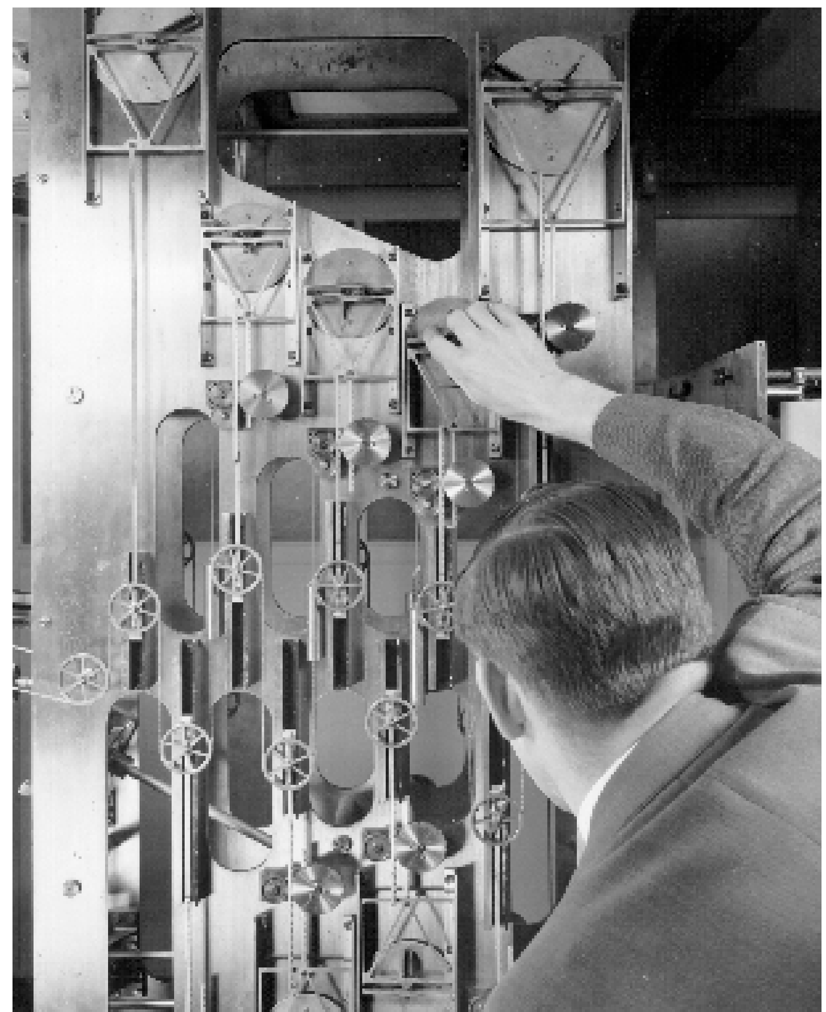
$$T(n) = 2T(n/2) + n$$

$$T(n) = n \log_2 n$$

Divide and conquer

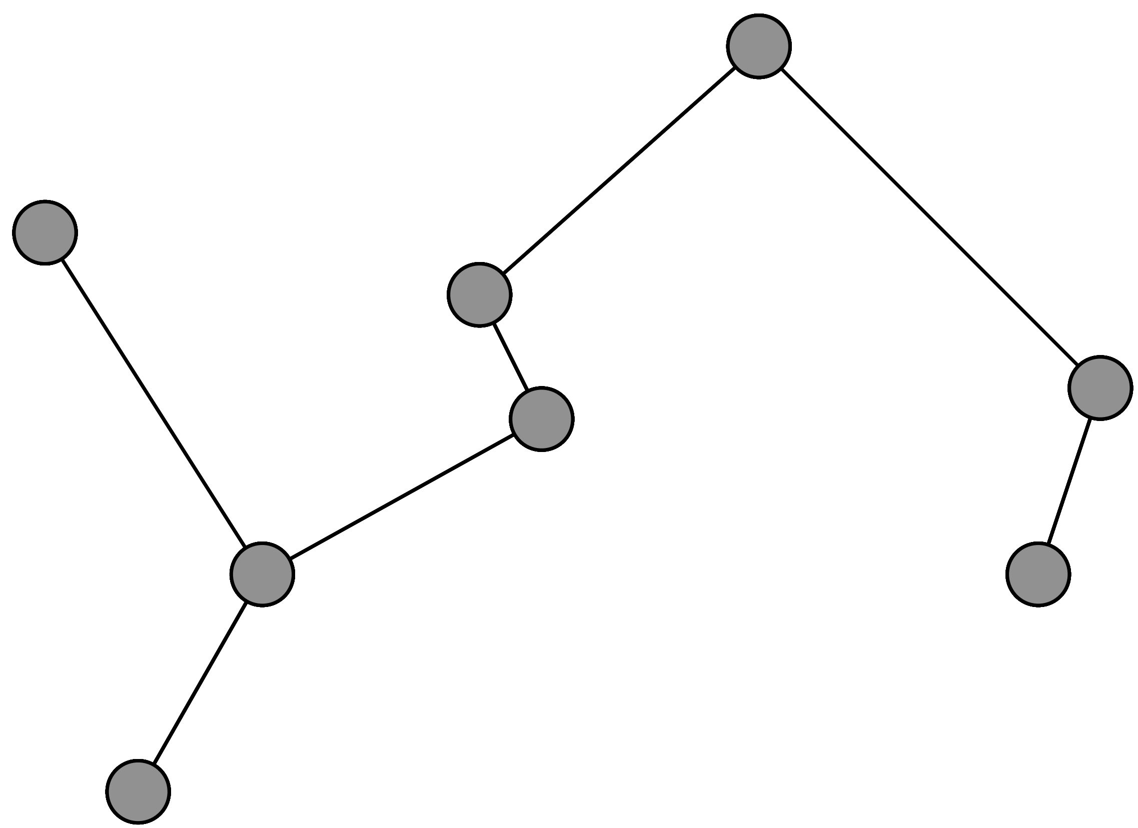
The Fast Fourier Transform





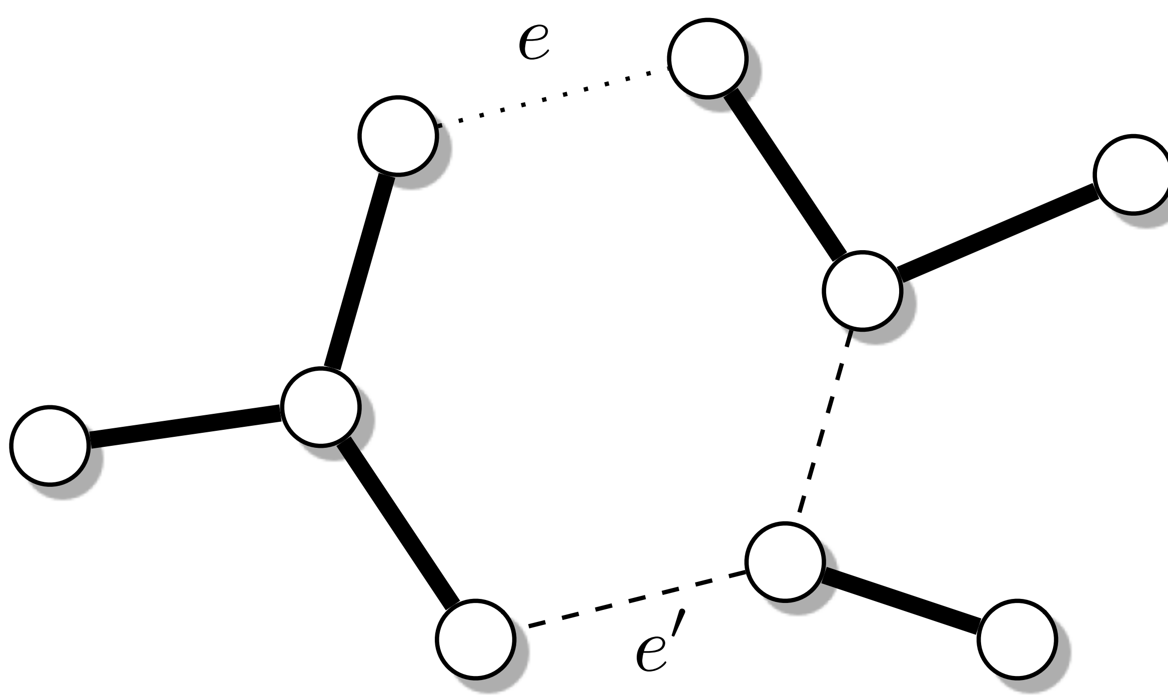
When greed is good

Minimum Spanning Tree (Boruvka, 1920): add the shortest edge



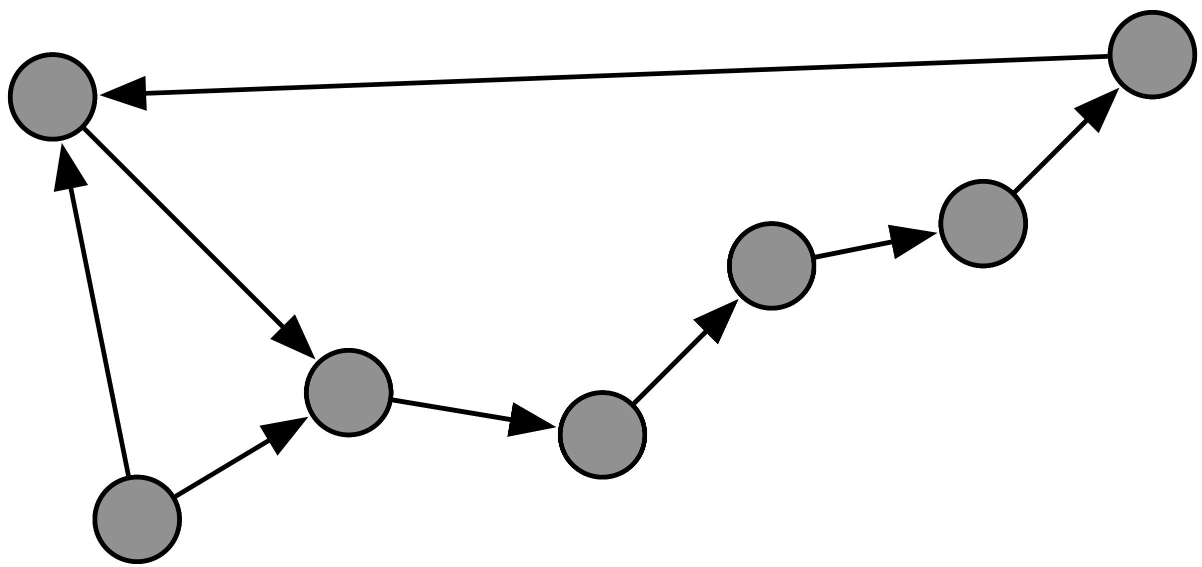
When greed is good

For Minimum Spanning Tree, doing the best thing in the short term can never lead us down the wrong path.



The primrose path

The Traveling Salesman Problem



Landscapes

A single optimum, that we can find by climbing:

Landscapes

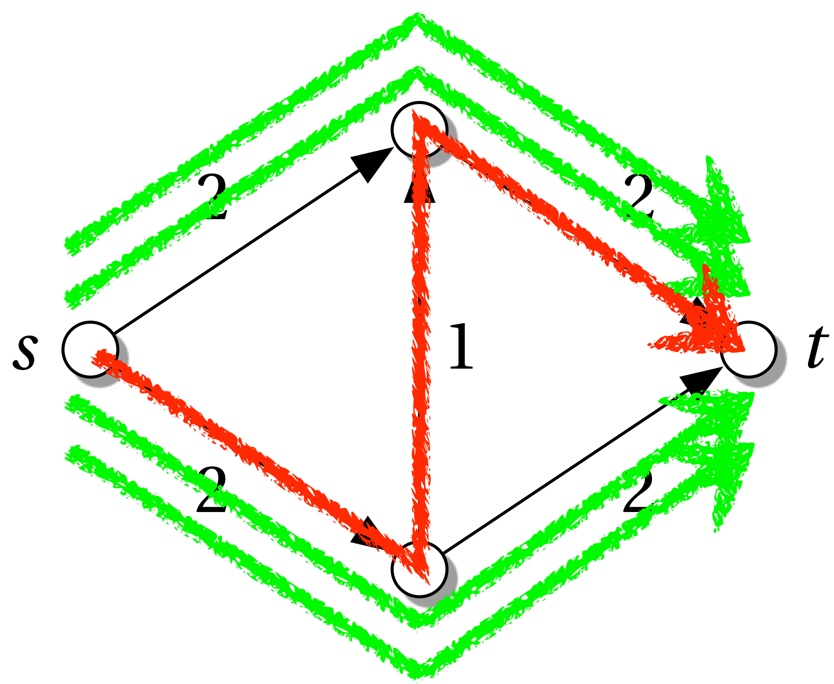
Many local optima where we can get stuck

Reorganizing the landscape: max flow

Each edge has a capacity

Greedy: push more flow along any path with excess capacity

But we can get stuck in local optima!

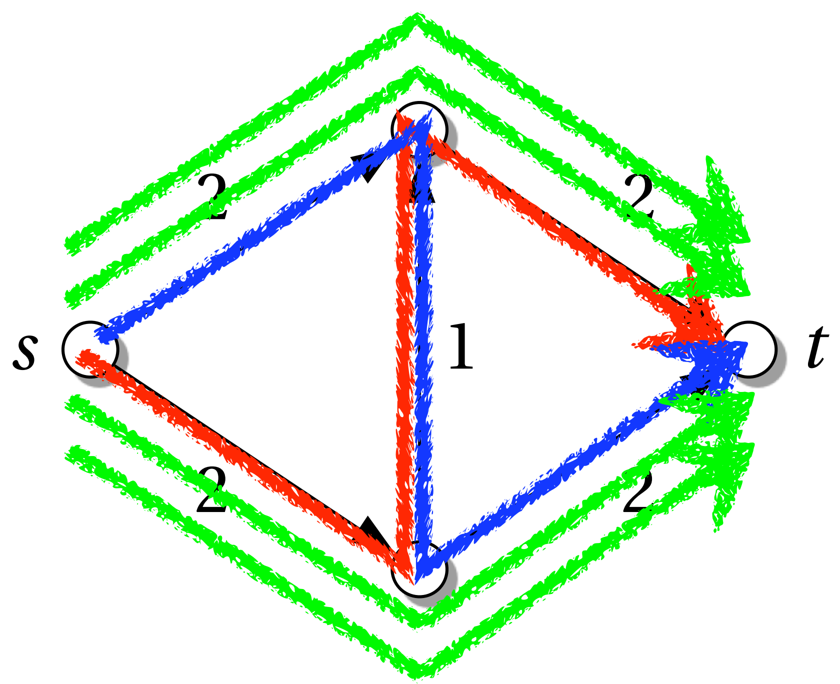


Reorganizing the landscape: max flow

Solution: allow reverse edges to cancel out previous flow

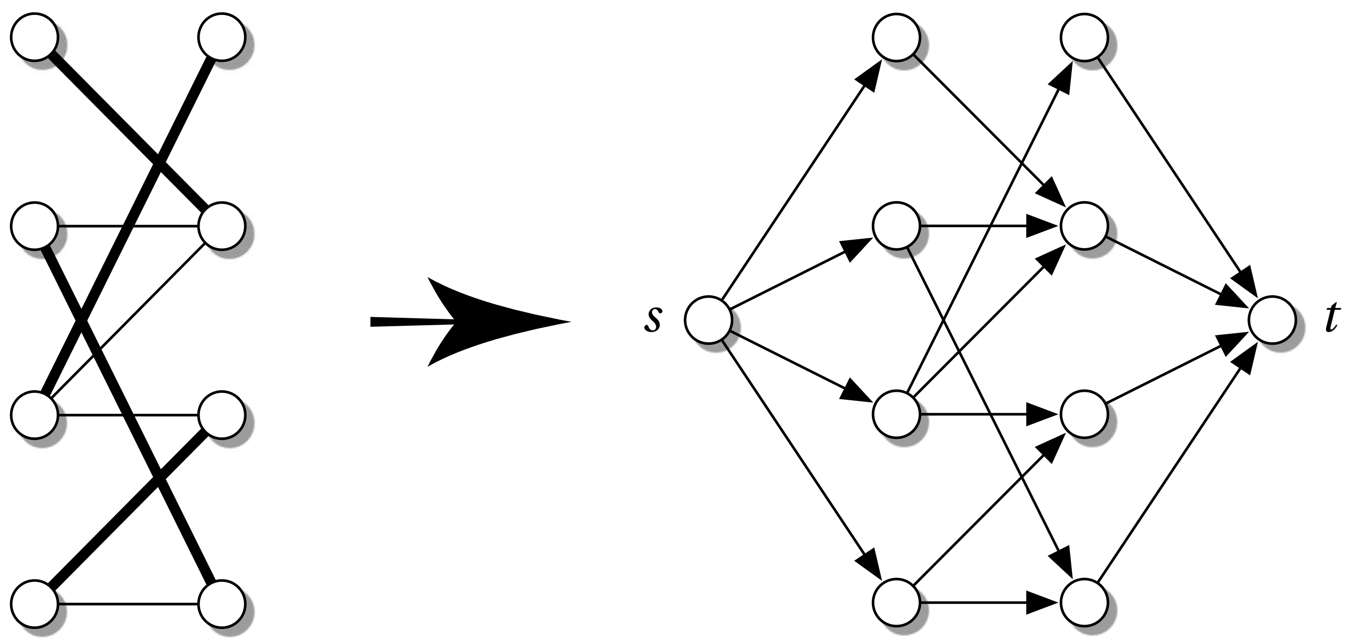
Theorem: now we can't get stuck

Sometimes we can turn the landscape from Rockies to Mt. Fuji, by defining what moves are possible—but not always!



“Reducing” one problem to another

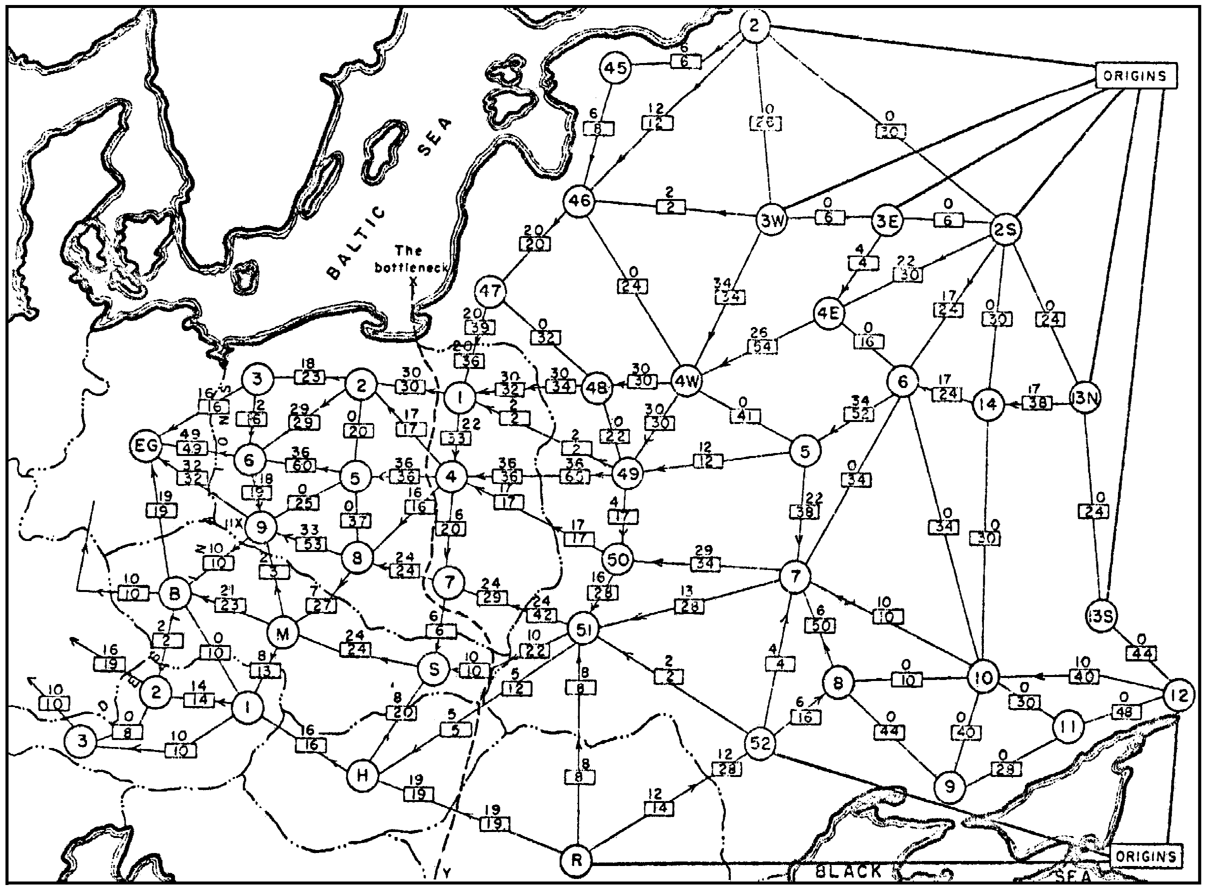
Translate new problem into one we already know how to solve



Bipartite Matching ≤ Max Flow

If Max Flow is easy, then so is Bipartite Matching

Duality: max flow and min cut



Cut the smallest set of edges that divides *s* from *t*

Find Max Flow from *s* to *t*, and cut the saturated edges

Computation and complexity

Systems aren't simple or complex; questions about them are

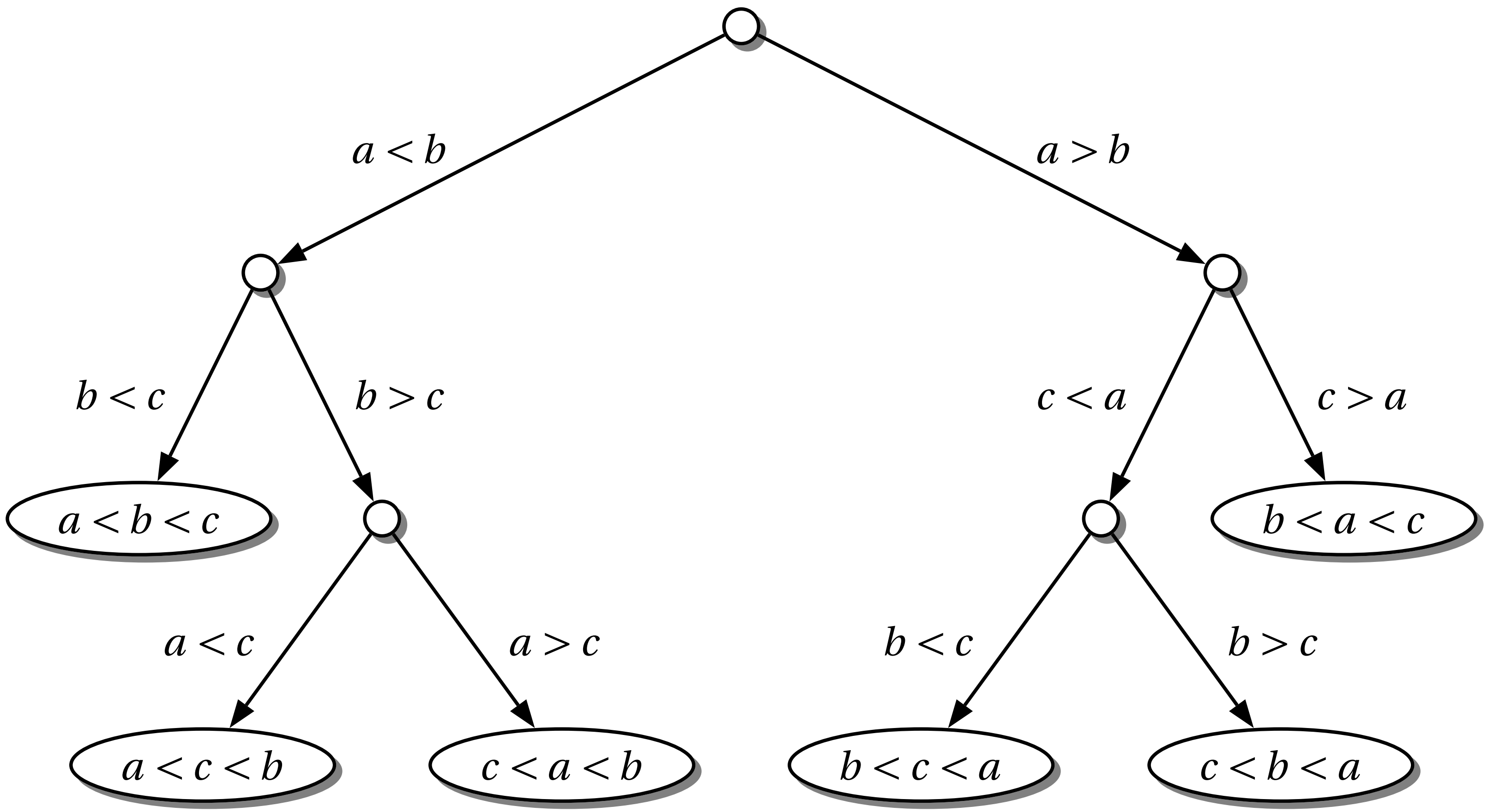
Intrinsic complexity of a problem: the running time (or memory use, or other resource) of the *best possible algorithm* for it

Worst-case! Works for all instances = works for worst ones

Upper bounds are easy: just give an algorithm

Lower bounds are hard!

How can we tell if an algorithm is optimal?



Twenty questions: can only distinguish 2^{20} situations

To sort, we need $\log_2 n! \approx n \log_2 n$ comparisons

Lessons so far

We only know of a small number of families of algorithms that are guaranteed to be efficient:

Greedy (local search)

Divide and conquer

Dynamic programming

Linear programming, convex programming, and duality

Are there others we haven't found yet? Are there fundamental limits to what efficient algorithms can do?

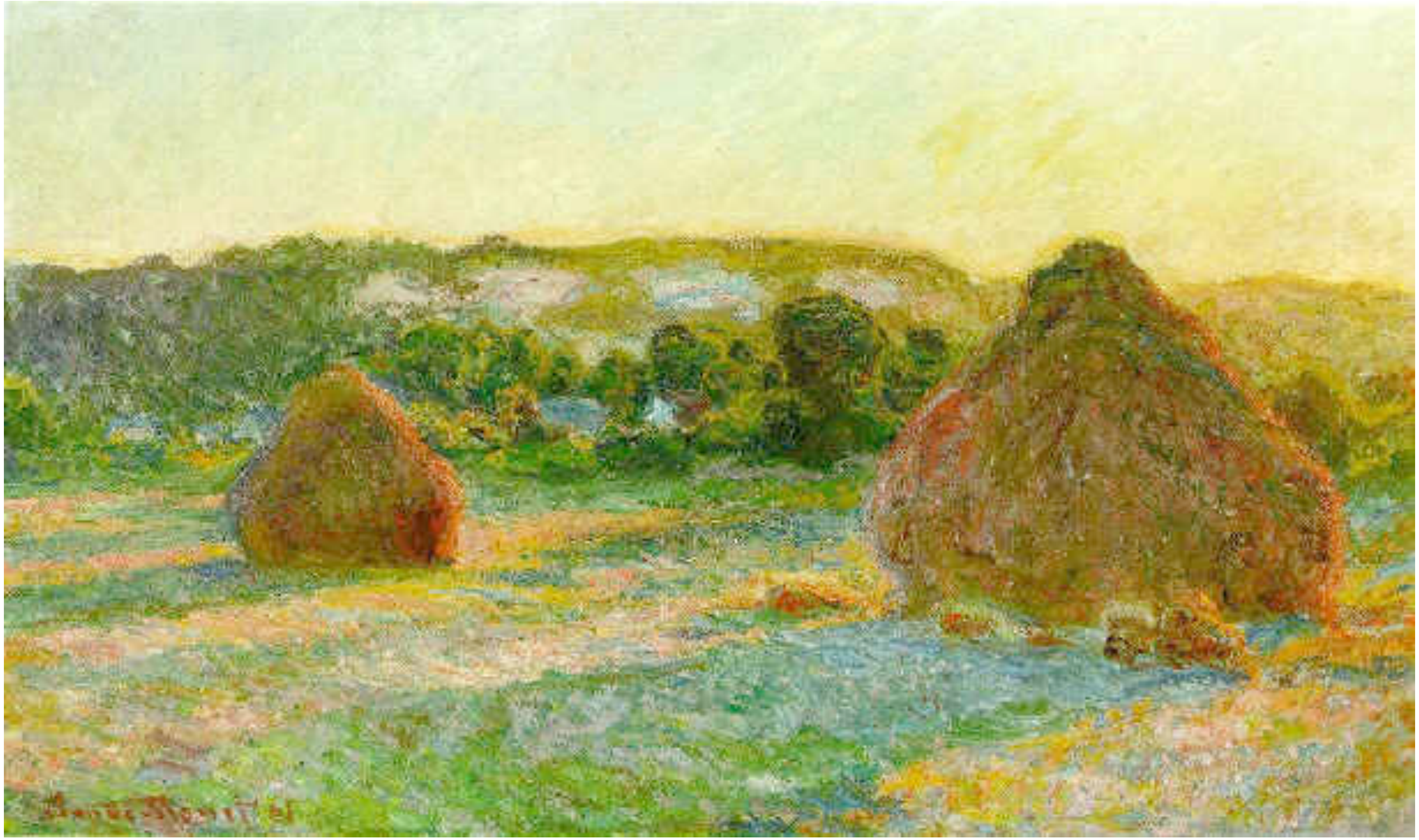
Computational Complexity 2:
NP-completeness and the P vs. NP question

Cristopher Moore
Santa Fe Institute

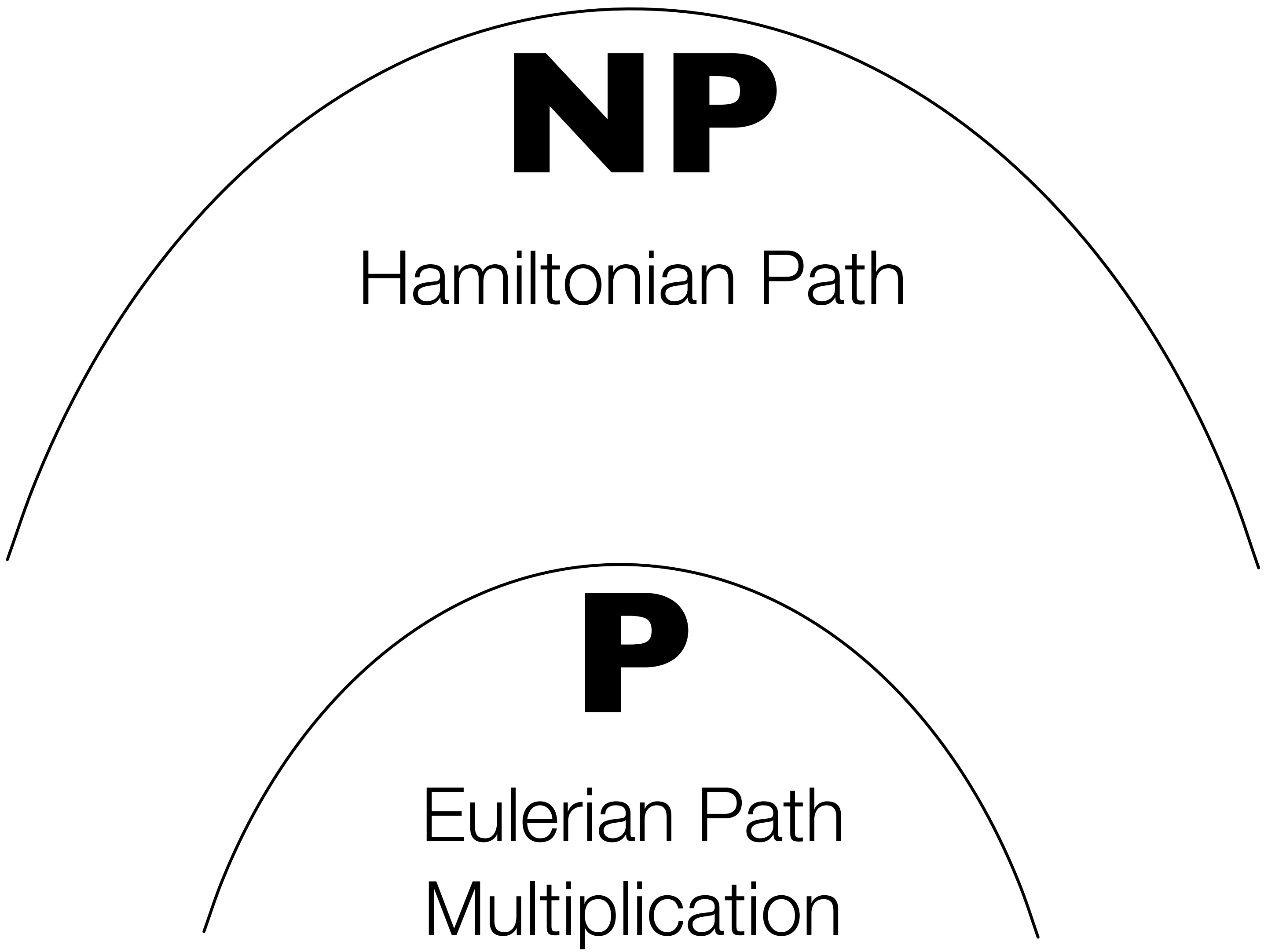
Needles in haystacks

P: we can find a solution efficiently

NP: we can *check* a solution efficiently



Complexity classes



NP-completeness

Some problems B have the amazing property that *any* problem in NP can be reduced to them: A ≤ B for all A in NP

But if A ≤ B and A is hard, then B is hard too

So if P≠NP, then B can't be solved in polynomial time

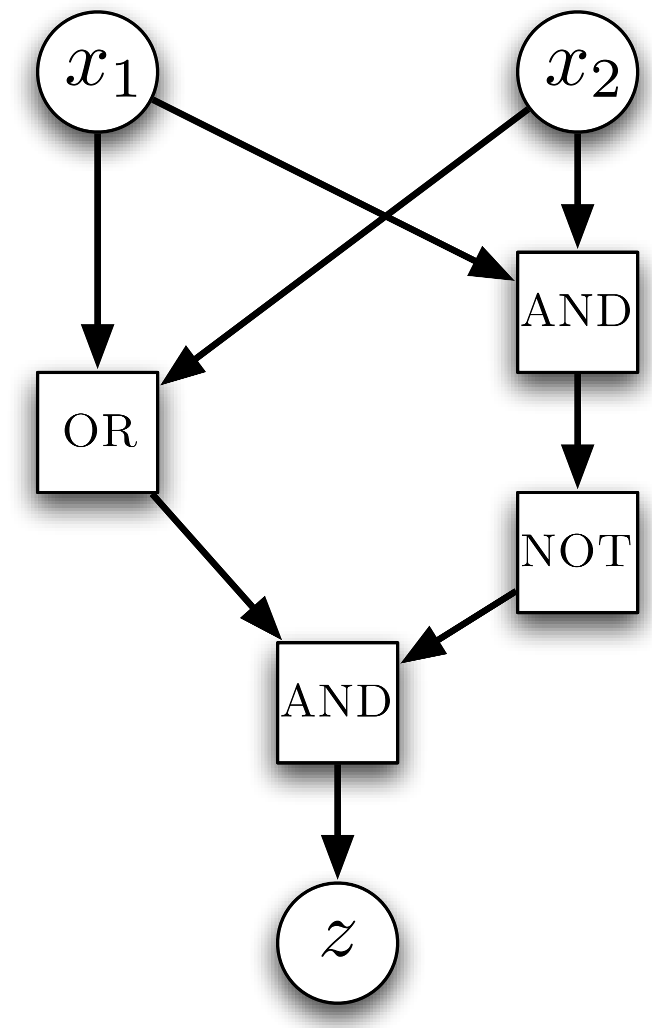
How can a single problem express every other problem in NP?

Satisfying a circuit

Any program that tests solutions (e.g. Hamiltonian paths) can be "compiled" into a Boolean circuit

The circuit outputs "true" if an input solution works

Is there a set of values for the inputs that makes the output true?

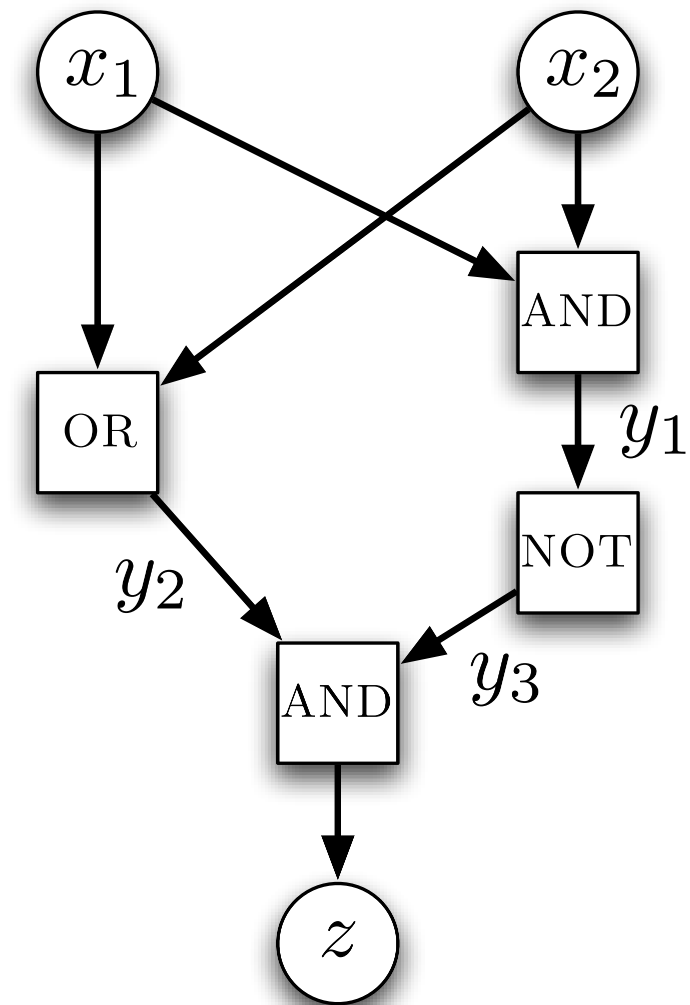


From circuits to formulas

Add variables representing the truth values of the wires

The condition that each gate works, and the output is “true,” can be written as a Boolean formula:

$$(x_1 \vee \overline{y}_1) \wedge (x_2 \vee \overline{y}_1) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee y_1) \\ \wedge \cdots \wedge z \quad .$$



3-SAT

Our first NP-complete problem!

Given a set of *clauses* with 3 variables each,

$$(x_1 \vee \overline{x}_2 \vee x_3) \wedge (x_2 \vee x_{17} \vee \overline{x}_{293}) \wedge \cdots$$

does a set of truth values for the x_i exist such that all the clauses are satisfied?

k-SAT (with *k* variables per clause) is NP-complete for $k \geq 3$

If 3-SAT were easy...

we could take any problem in NP we want to solve,

write a program that checks solutions,

convert that program into a circuit,

convert that circuit to a 3-SAT formula which is satisfiable if a solution exists,

and use our efficient algorithm for 3-SAT to solve it!

So, if 3-SAT is in **P**, then all of **NP** is too, and **P=NP**

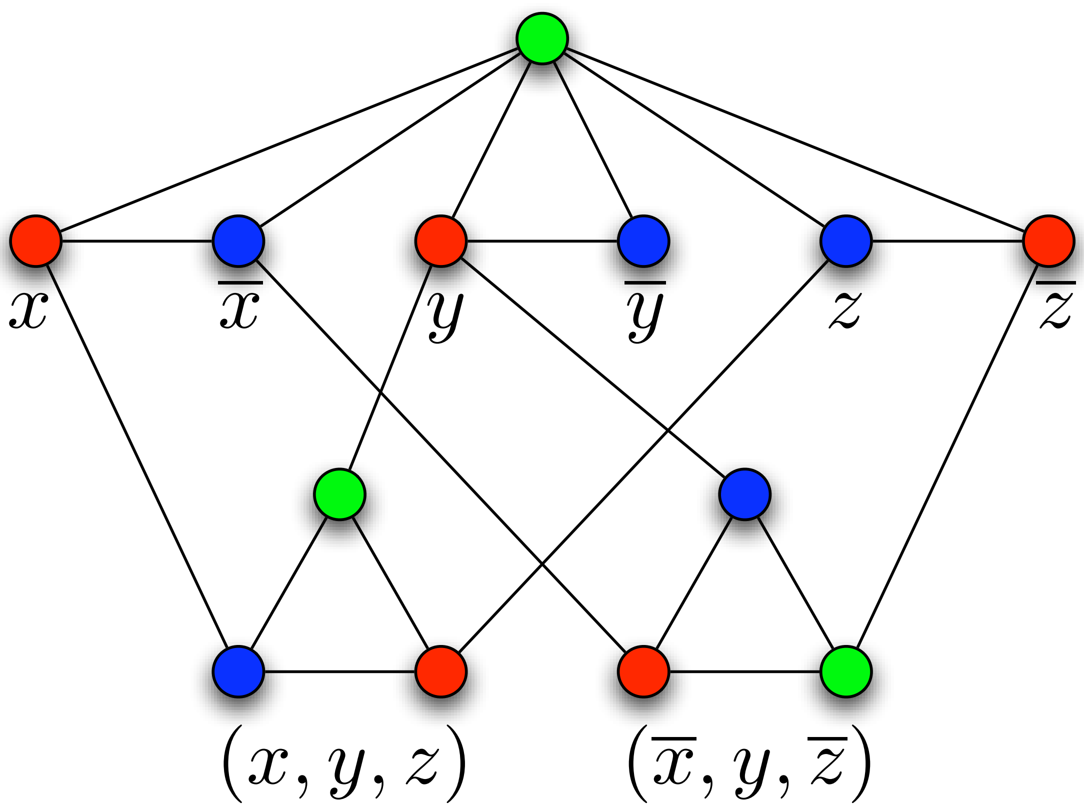
Conversely, if P≠NP, then 3-SAT cannot be solved in polynomial time: something like exhaustive search is needed

Graph coloring

Given a set of countries and borders between them, what is the smallest number of colors we need?

From SAT to Coloring

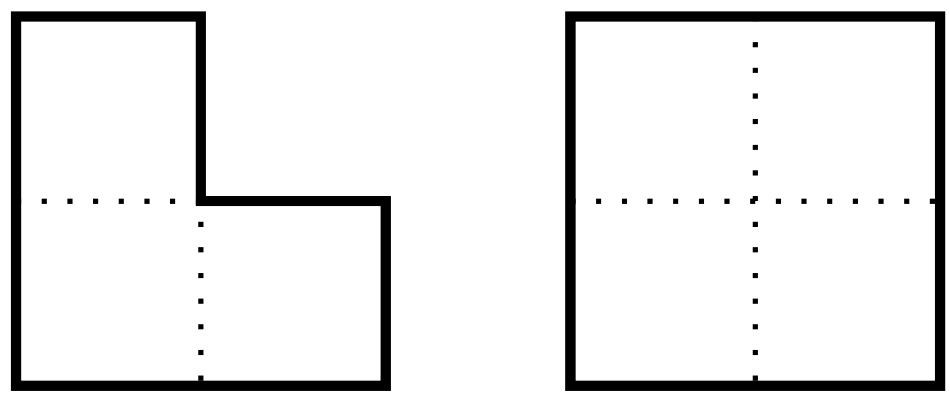
"Gadgets" enforce constraints:



Graph 3-Coloring is NP-complete

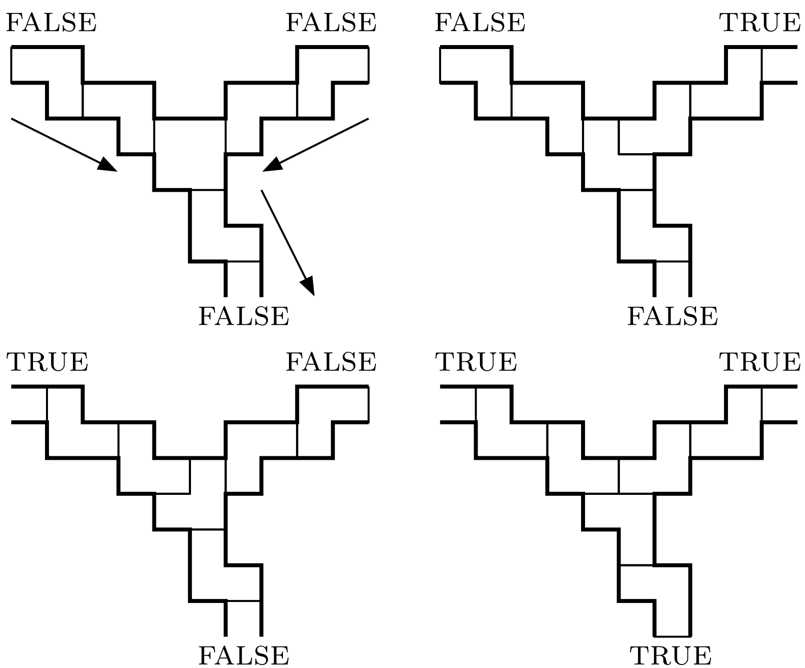
Graph 2-Coloring is in **P** (why?)

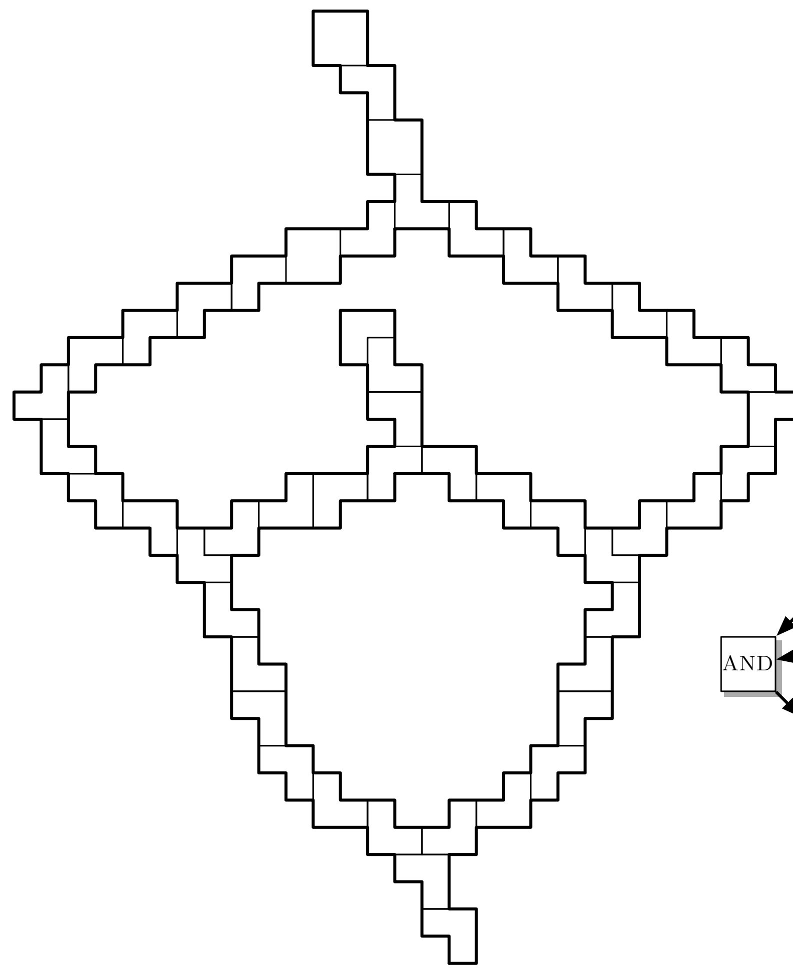
Why are some problems NP-complete?

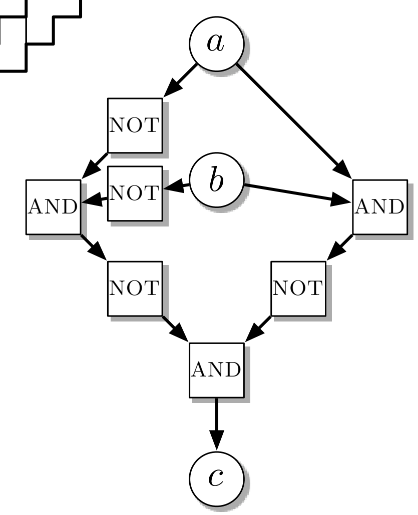


Can we tile a shape with these tiles?

Because we can use them to build a computer.

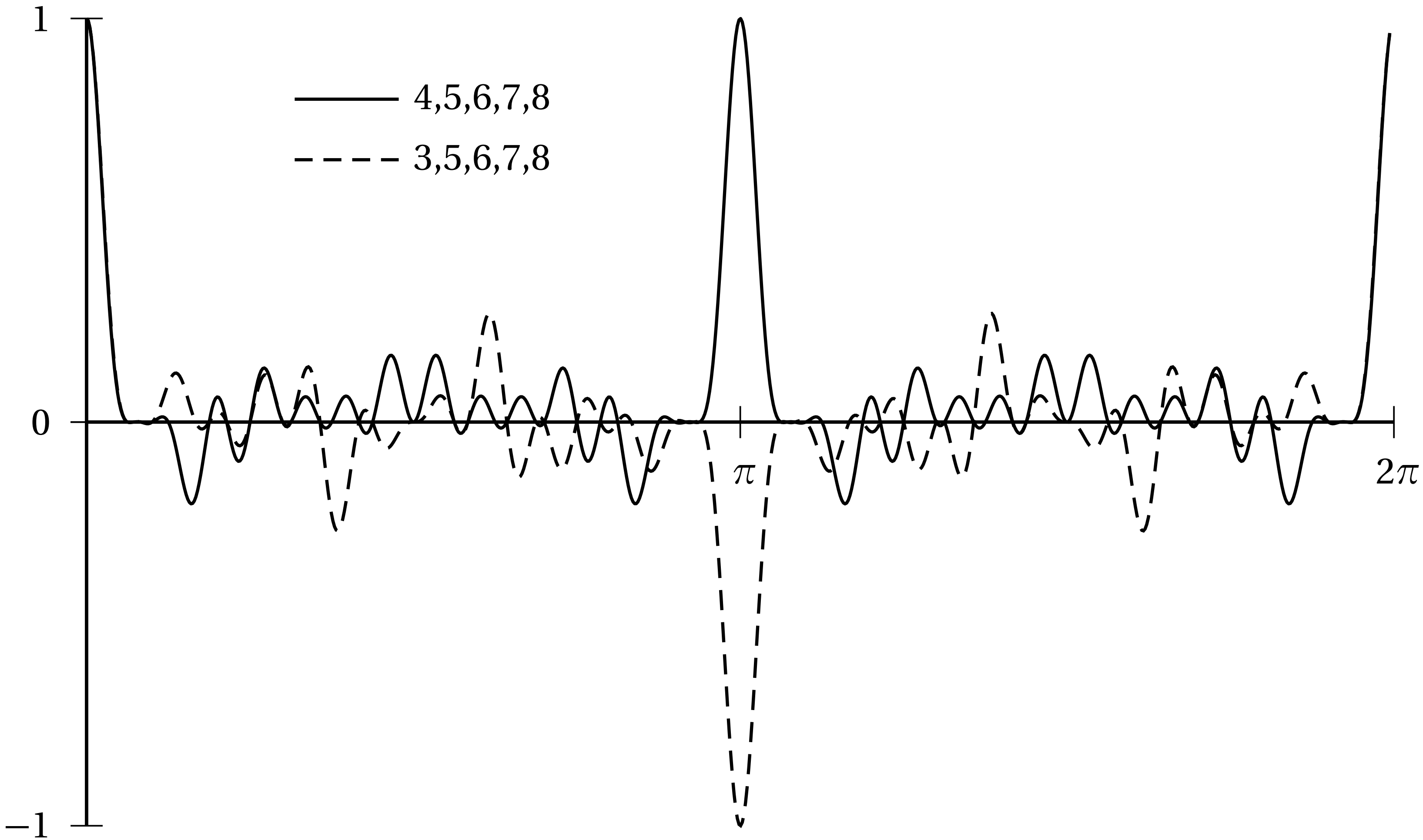




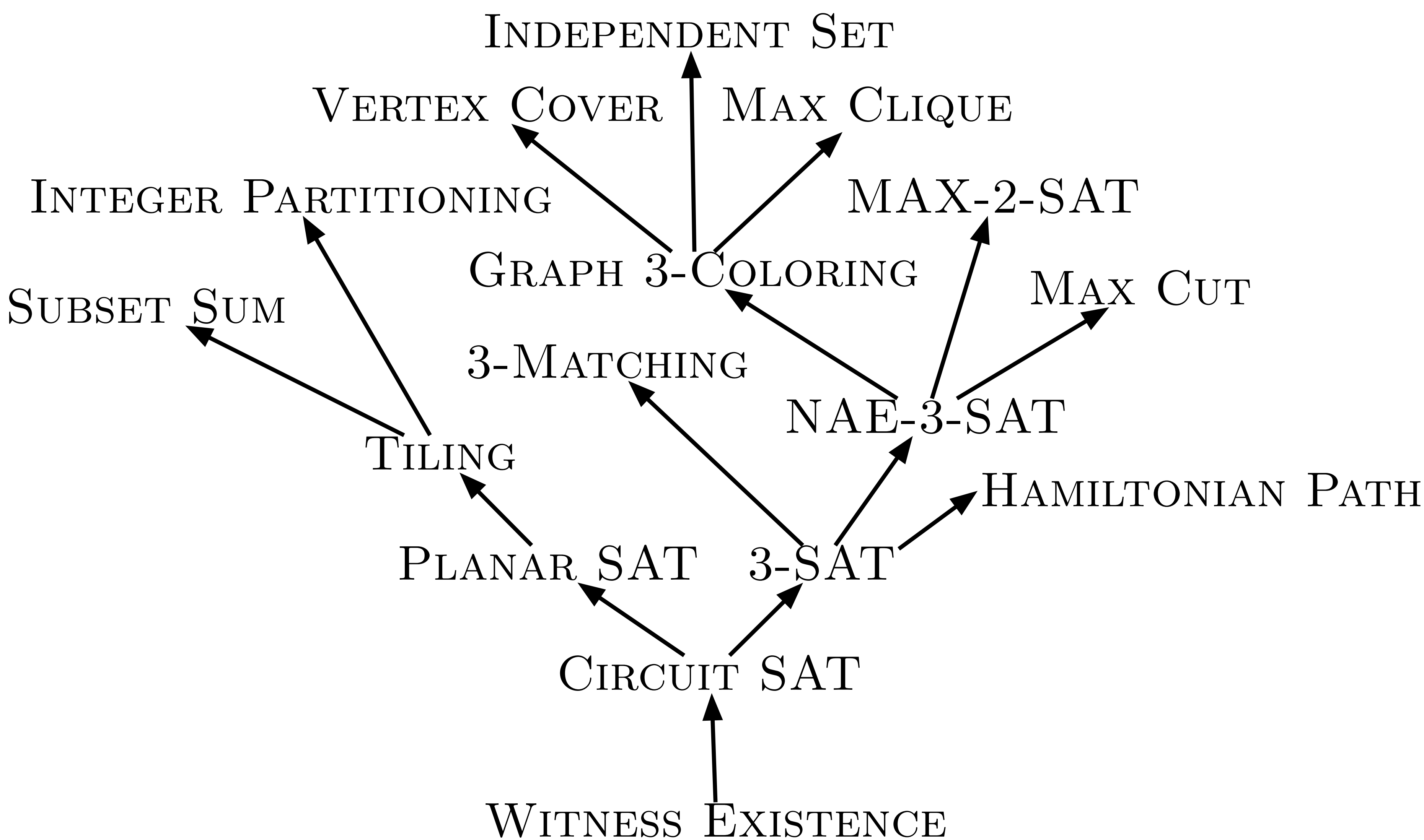


Even problems in continuous mathematics…

$$\int_{-\pi}^{\pi} (\cos a_1 x)(\cos a_2 x)\cdots(\cos a_n x)\,\mathrm{d}x \neq 0?$$



Thousands of NP-complete problems



Oh, cruel world!

NP-completeness is a worst-case notion...

We assume that instances are designed by a clever adversary to encode hard problems

A good assumption in cryptography, but not in most of nature

The scientist is always working to discover the order and organization of the universe, and is thus playing a game against the arch-enemy, disorganization. Is this devil Manichaean or Augustinian? Is it a contrary force opposed to order or is it the very absence of order itself?

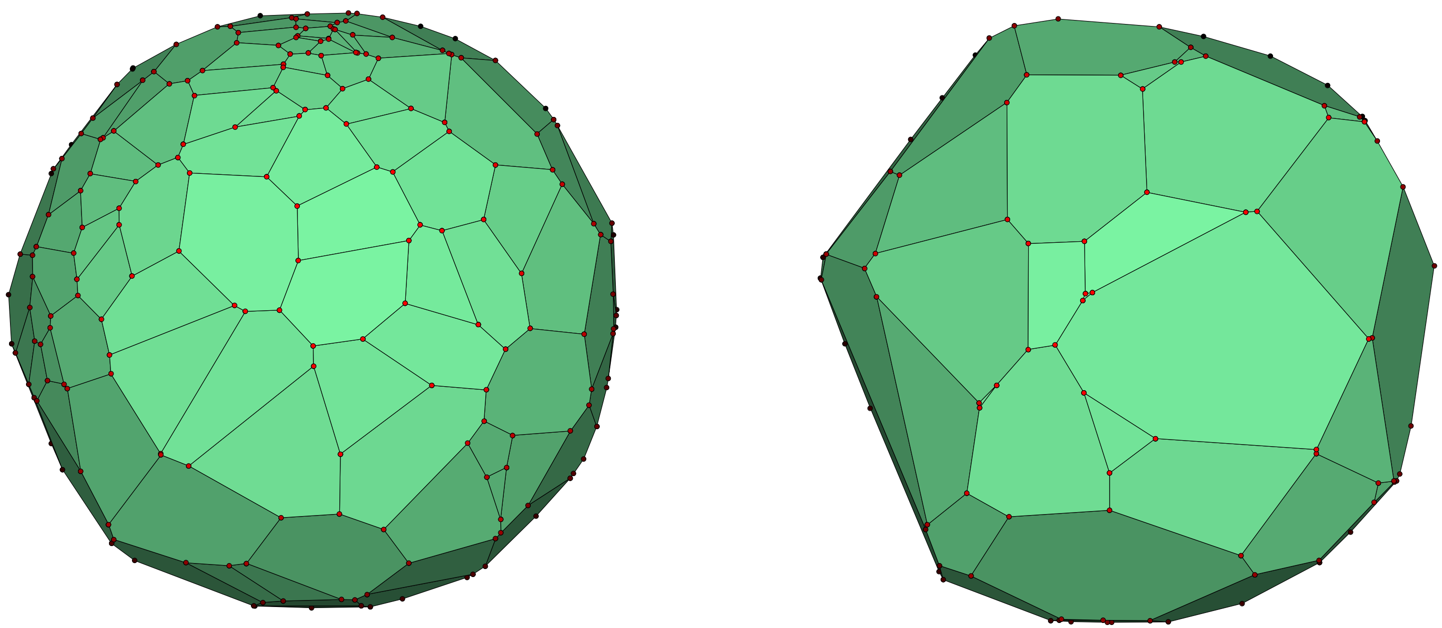
— Norbert Wiener, *Cybernetics*

From theory to the real world

Many algorithms that take exponential time in the worst case are efficient in practice

Optimization problems are like exploring a high-dimensional jewel

If we add noise to the problem, the number of facets goes down, and the path to the top gets shorter



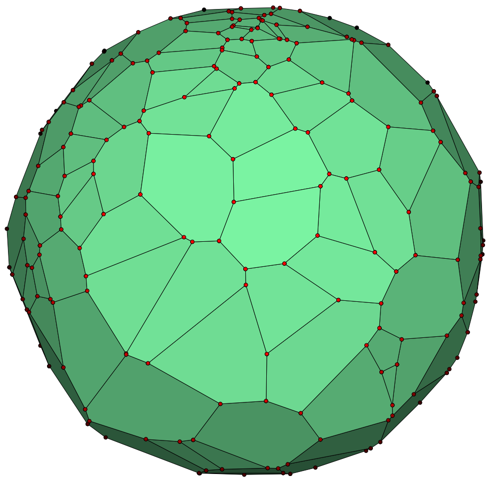
Alternatives

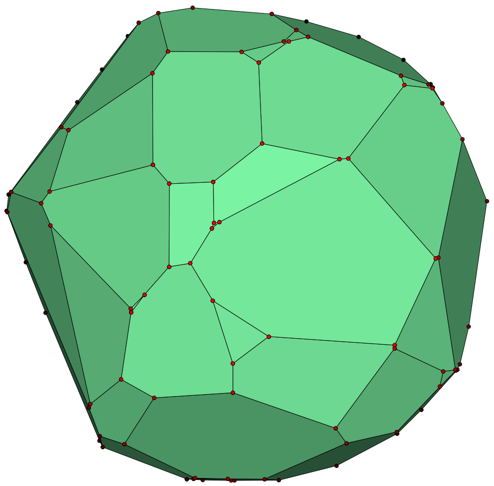
Probably Approximately Correct [Valiant]

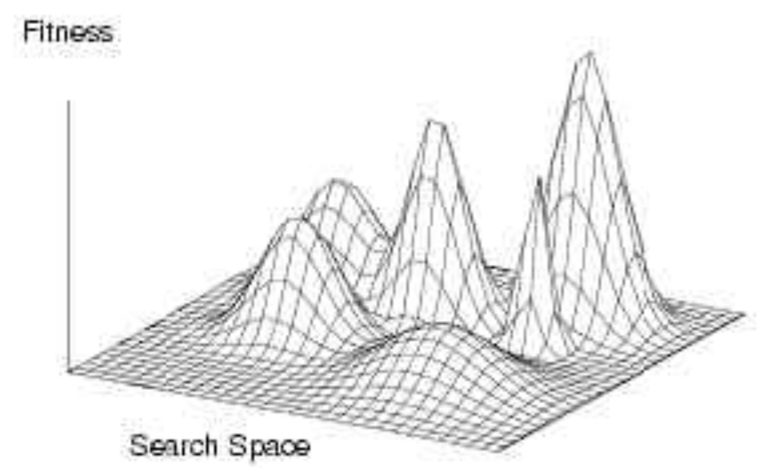
Noise can foil the adversary, producing a smoother problem [Spielman and Teng]

Landscapes are not as bumpy as they could be: good solutions are close to the optimum [Balcan, Blum, and Gupta, clustering]

In nature, problems and algorithms coevolve (e.g. protein folding)







Beyond NP

Which of these puzzles are in NP? Which has a solution that is easy to check?

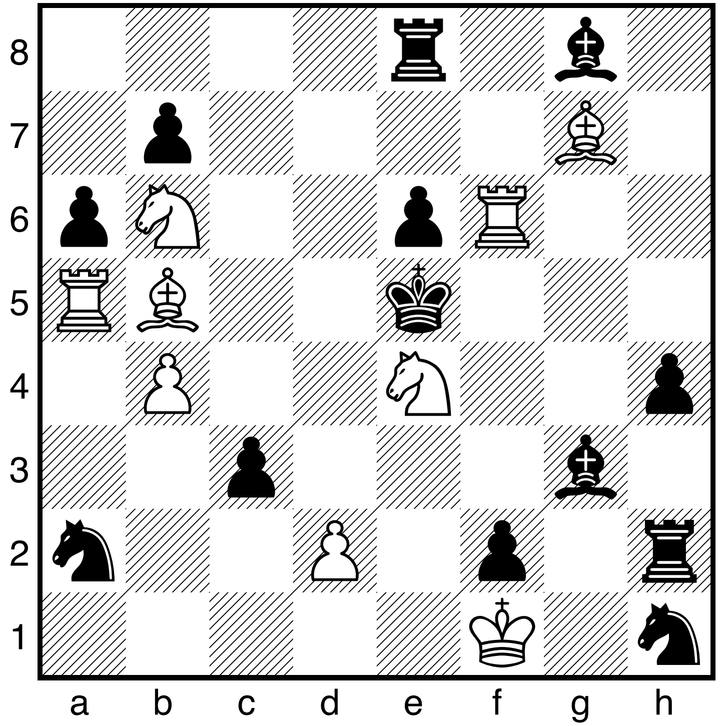
Logical hierarchies

I can win if there exists a move for me,

such that for all of your replies,

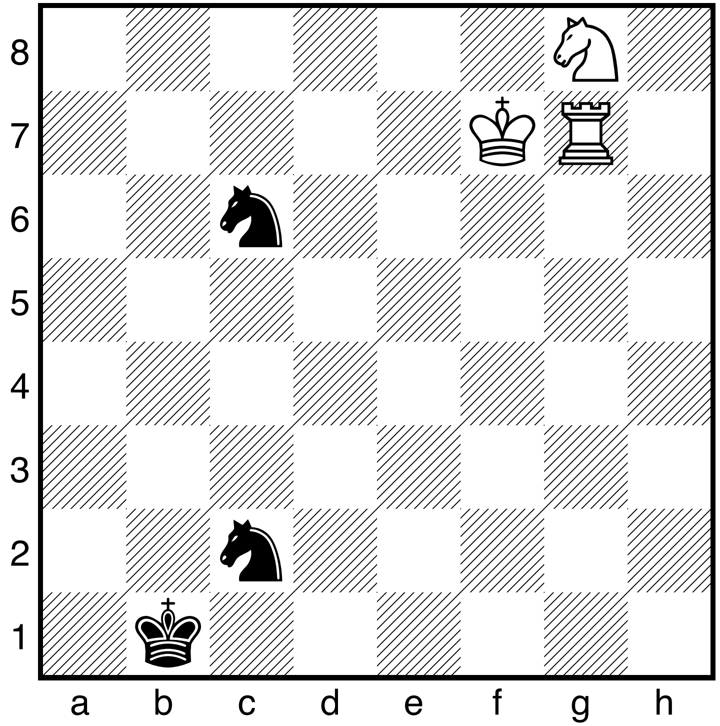
there exists a move for me...

Sam Loyd (1903)



Mate in 3

Lewis Stiller (1995)



Mate in 262

Undecidability

Suppose we could tell whether a program *p* will ever halt. This would be really handy!

Fermat
begin
$\quad t := 3;$
$\quad$ **repeat**
$\quad\quad$ **for** $n = 3$ **to** t **do**
$\quad\quad\quad$ **for** $x = 1$ **to** t **do**
$\quad\quad\quad\quad$ **for** $y = 1$ **to** t **do**
$\quad\quad\quad\quad\quad$ **for** $z = 1$ **to** t **do**
$\quad\quad\quad\quad\quad\quad$ **if** $x^n + y^n = z^n$ **then return** (x, y, z, n);
$\quad\quad\quad\quad\quad$ **end**
$\quad\quad\quad\quad$ **end**
$\quad\quad\quad$ **end**
$\quad\quad$ **end**
$\quad\quad t := t + 1;$
$\quad$ **until** forever;
end

I have discovered a marvelous proof that this program will run forever, but it is too small to fit on this slide...

Undecidability

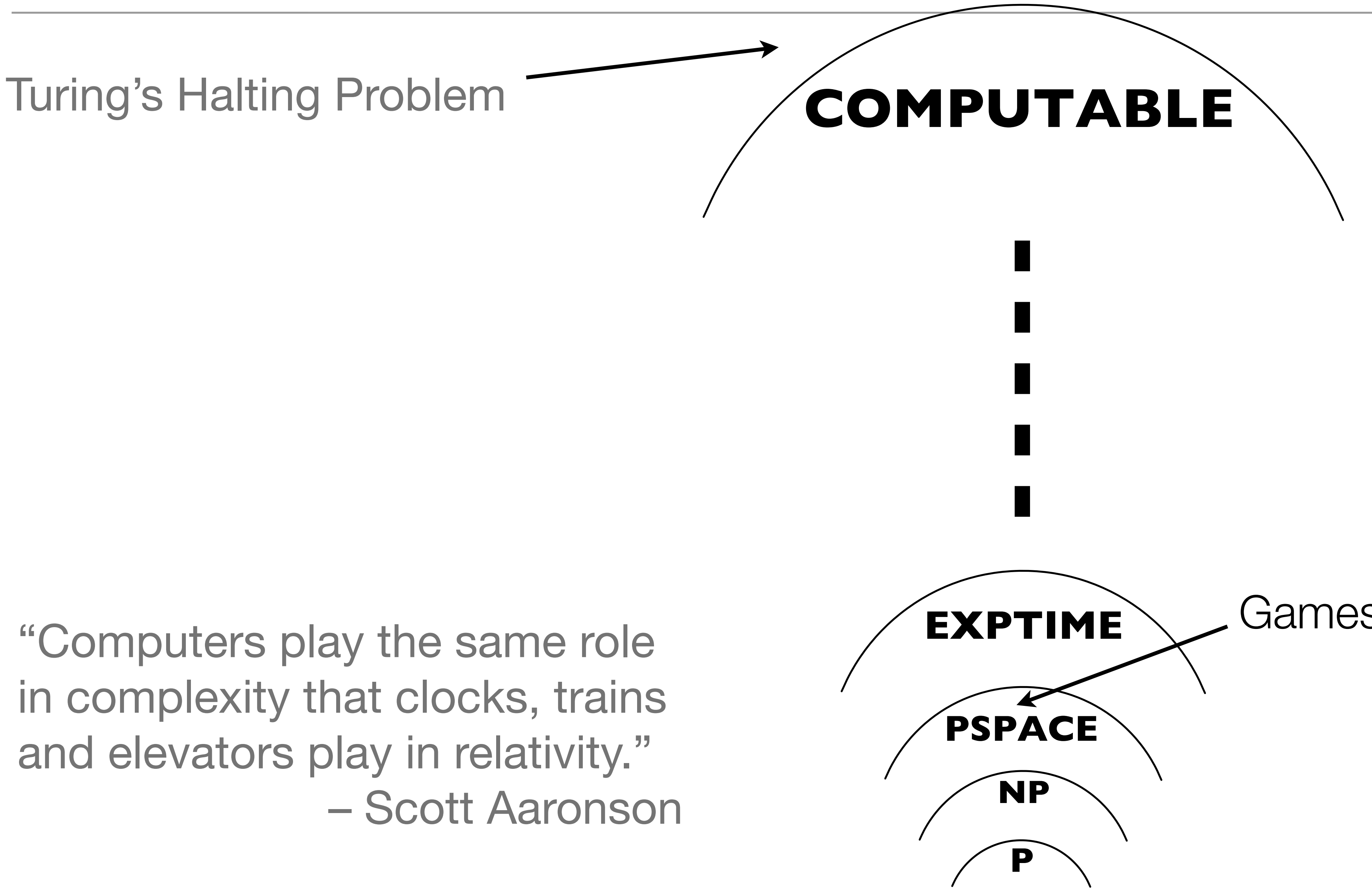
Suppose `halt(p,x)` can tell whether *p*, given input *x*, will halt. Then we could feed it to itself, and run this program instead:

```
trouble(p):
  if halt(p,p) loop forever
  else halt
```

Will **`trouble(trouble)`** halt or not?

Undecidable problems ⇒ unprovable truths!

An infinite hierarchy



"Computers play the same role in complexity that clocks, trains and elevators play in relativity."
– Scott Aaronson

Questions, questions...

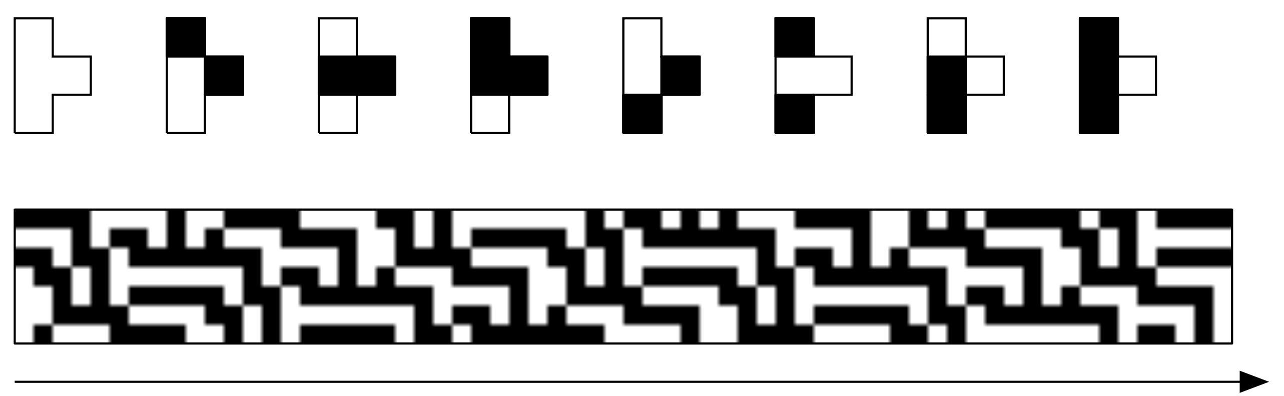
Suppose we have a cellular automaton. There are lots of questions we could ask about it:

Given an initial state *s*, what will the state be at time *t*? **P**

Does a state *s* have a predecessor? **NP**

On a lattice of size *n*, is *s* on a periodic orbit? **PSPACE**

On an infinite lattice, will *s* ever die out? **Undecidable**



Deep questions

Is finding solutions harder than checking them? When can we avoid exhaustive search?

How much memory do we need to find our way through a maze?

What if we only need good answers, instead of the best ones? Are there problems where even finding good answers is hard?

How much does it help if we can do many things at once?

If you and I are working together to solve a problem, how much do we need to communicate?

Are there good pseudorandom generators? Are there strong cryptosystems?

How much does quantum physics help?

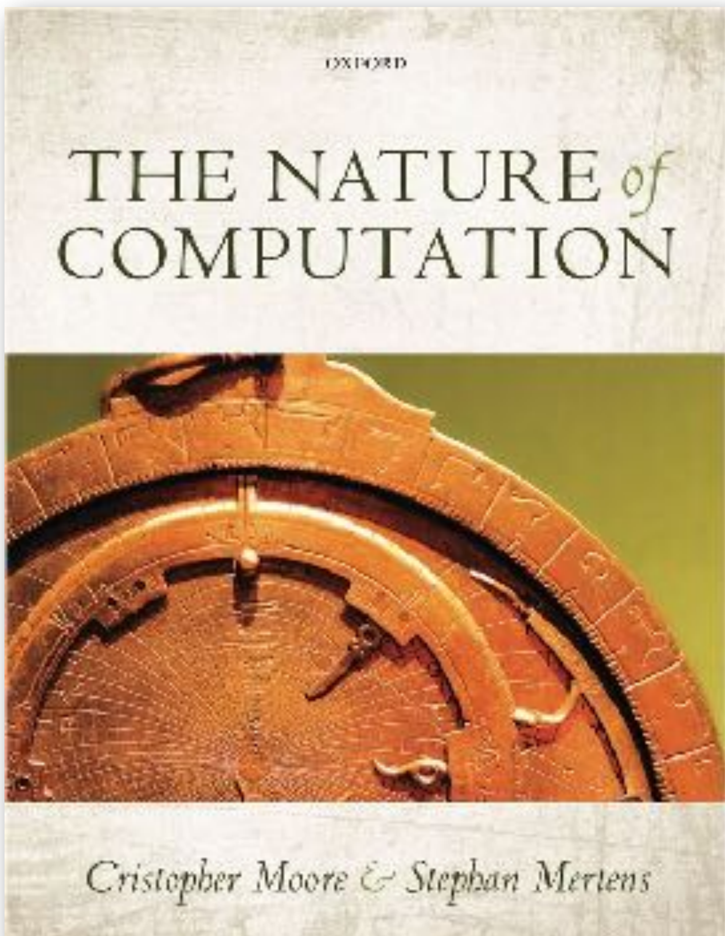
The computational lens

Do brains compute? Do cells? Do societies? Do planets?

Maybe the wrong question…

Does focusing on flows and transformations of information
(as opposed to e.g. energy) help me understand my system?

Shameless Plug



www.nature-of-computation.org

To put it bluntly: this book rocks! It somehow manages to combine the fun of a popular book with the intellectual heft of a textbook.

Scott Aaronson, MIT

This is, simply put, the best-written book on the theory of computation I have ever read; one of the best-written mathematical books I have ever read, period.

Cosma Shalizi, Carnegie Mellon

A creative, insightful, and accessible introduction to the theory of computing, written with a keen eye toward the frontiers of the field and a vivid enthusiasm for the subject matter.

Jon Kleinberg, Cornell