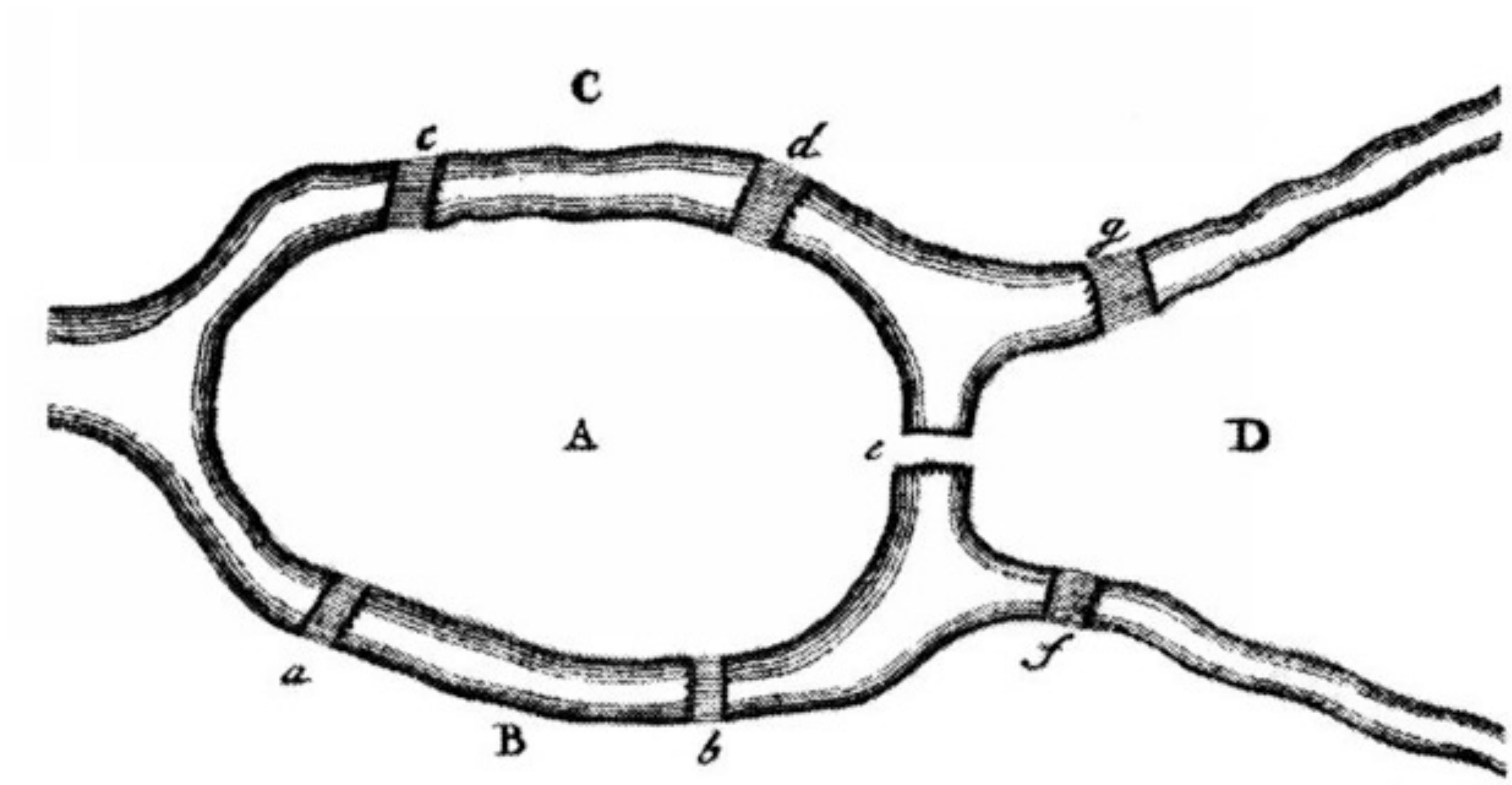
Computational Complexity 1: Algorithms and Landscapes

Cristopher Moore
Santa Fe Institute

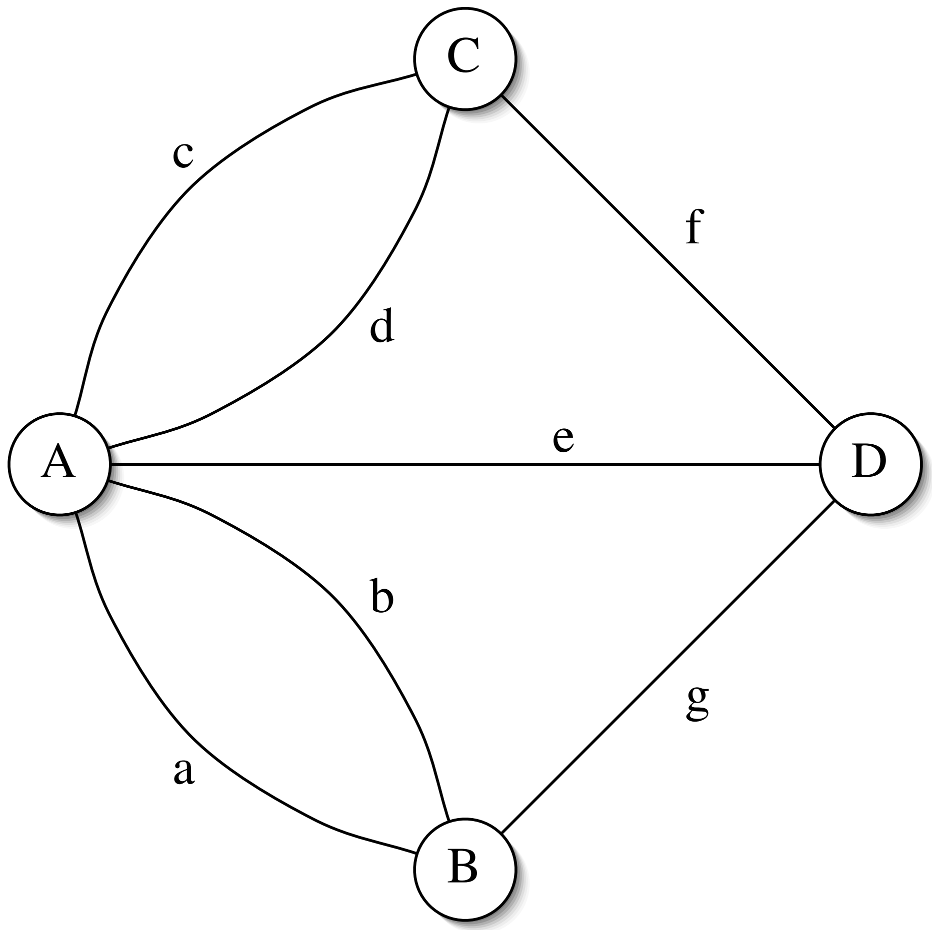
Computational complexity

Why are some problems qualitatively harder than others?



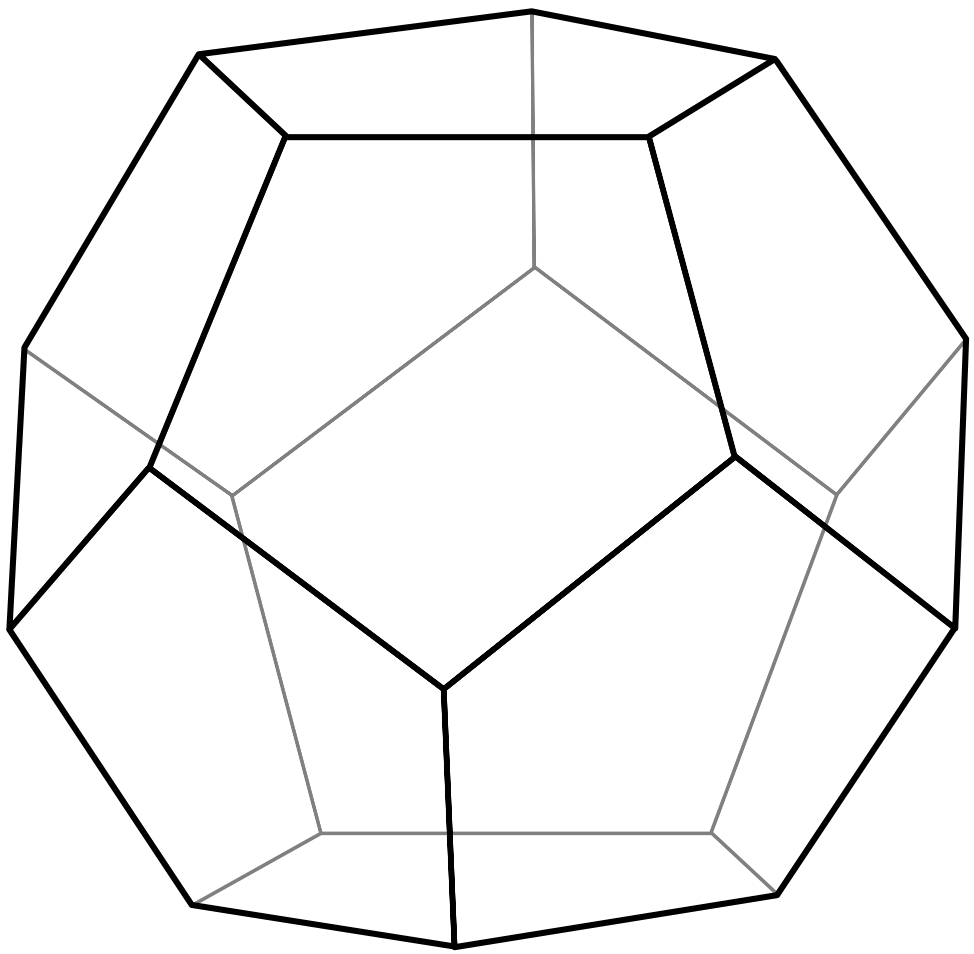
Computational complexity

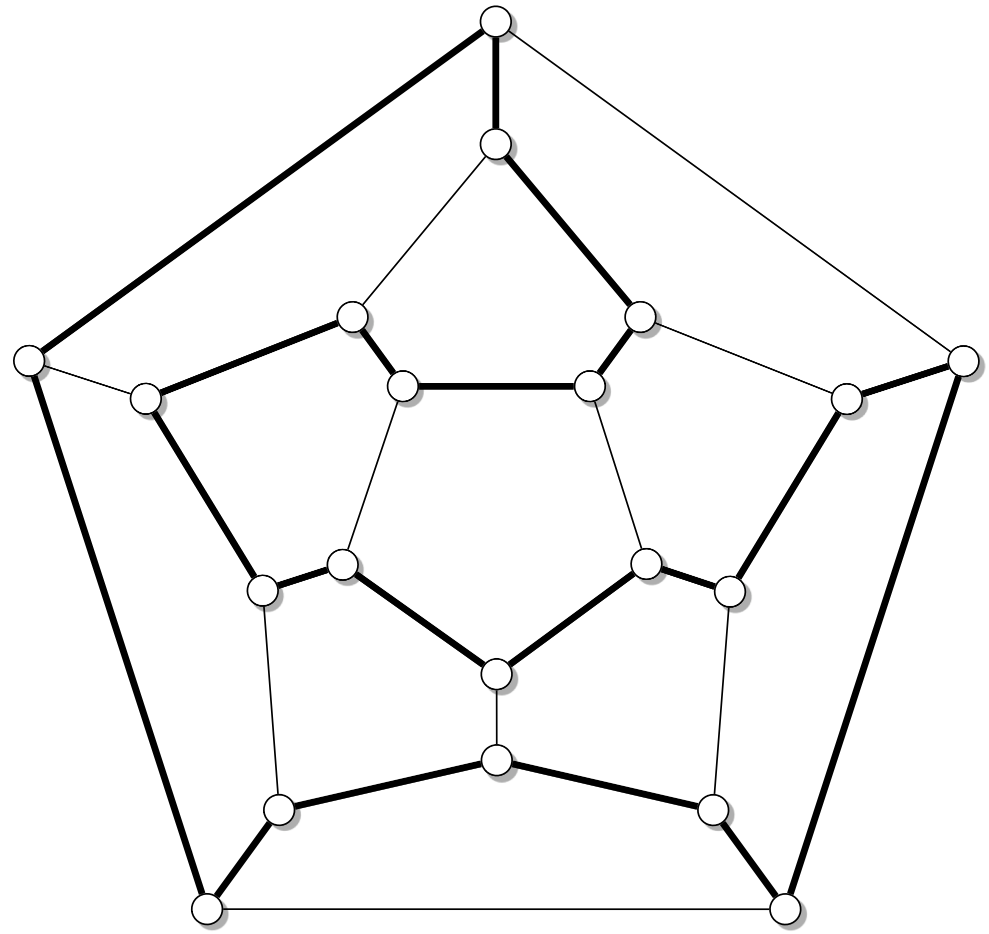
Euler: at most two nodes can have an odd number of bridges, so no tour is possible!



Computational complexity

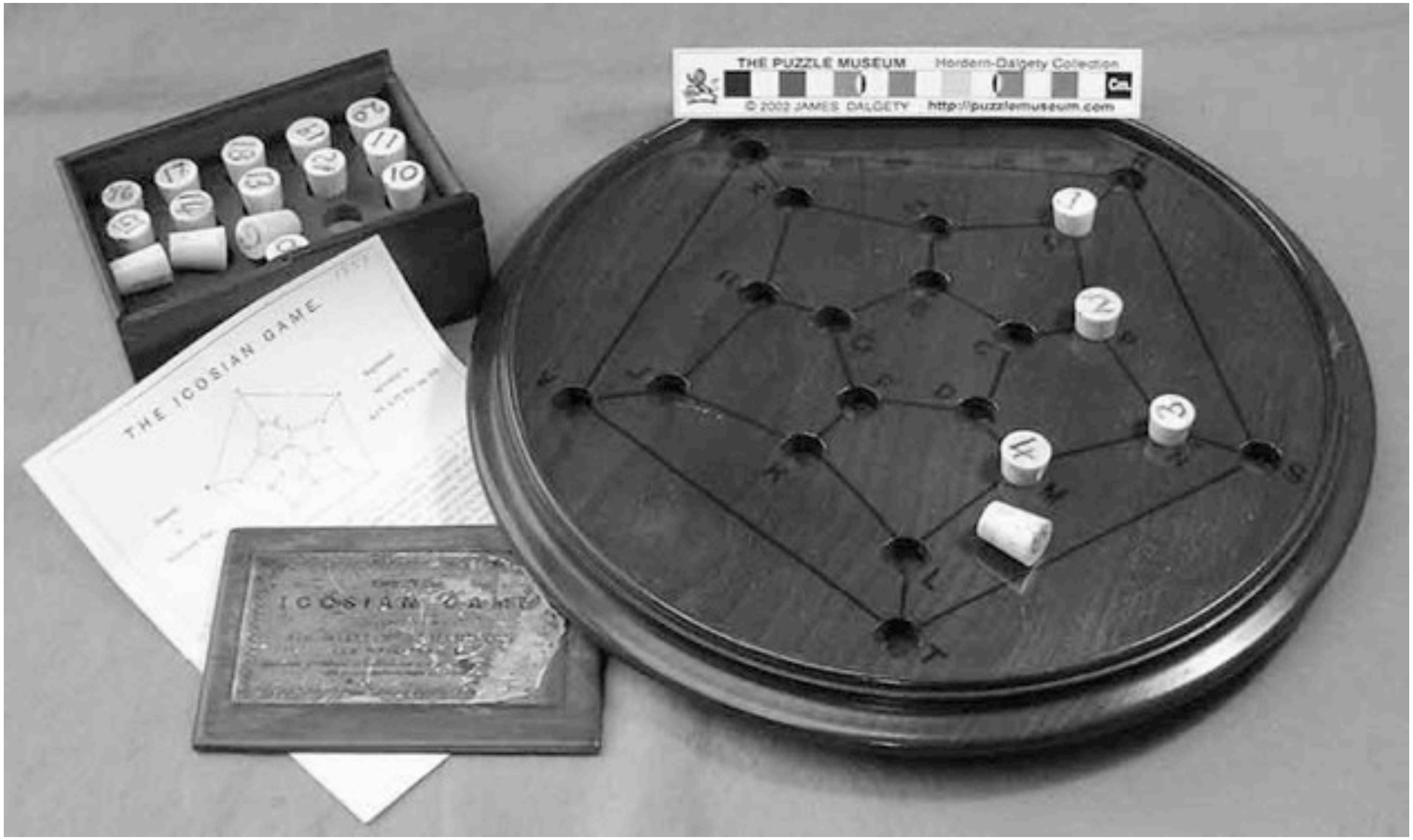
What if we want to visit every vertex, instead of every edge?





Computational complexity

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

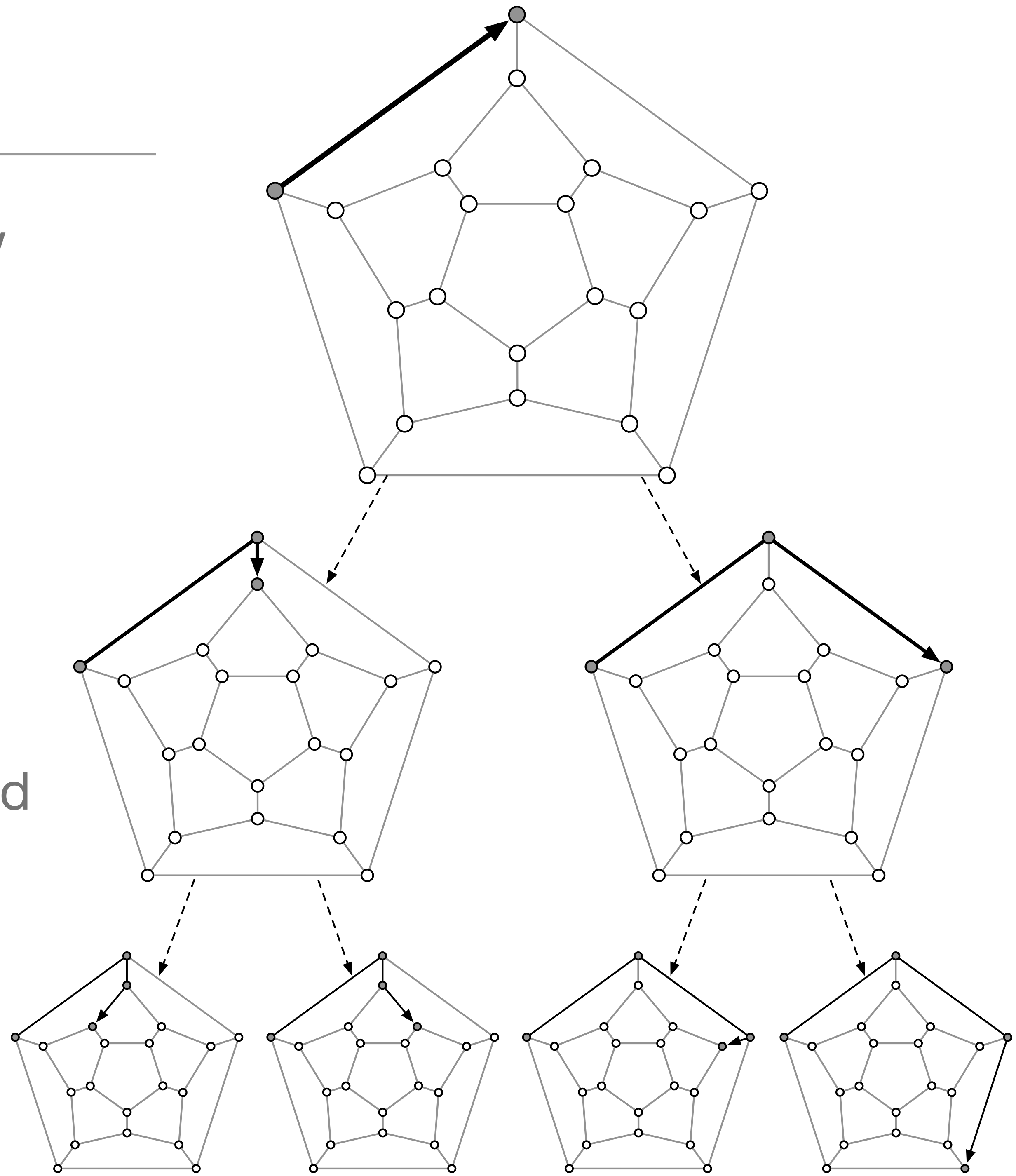


An exponential tree

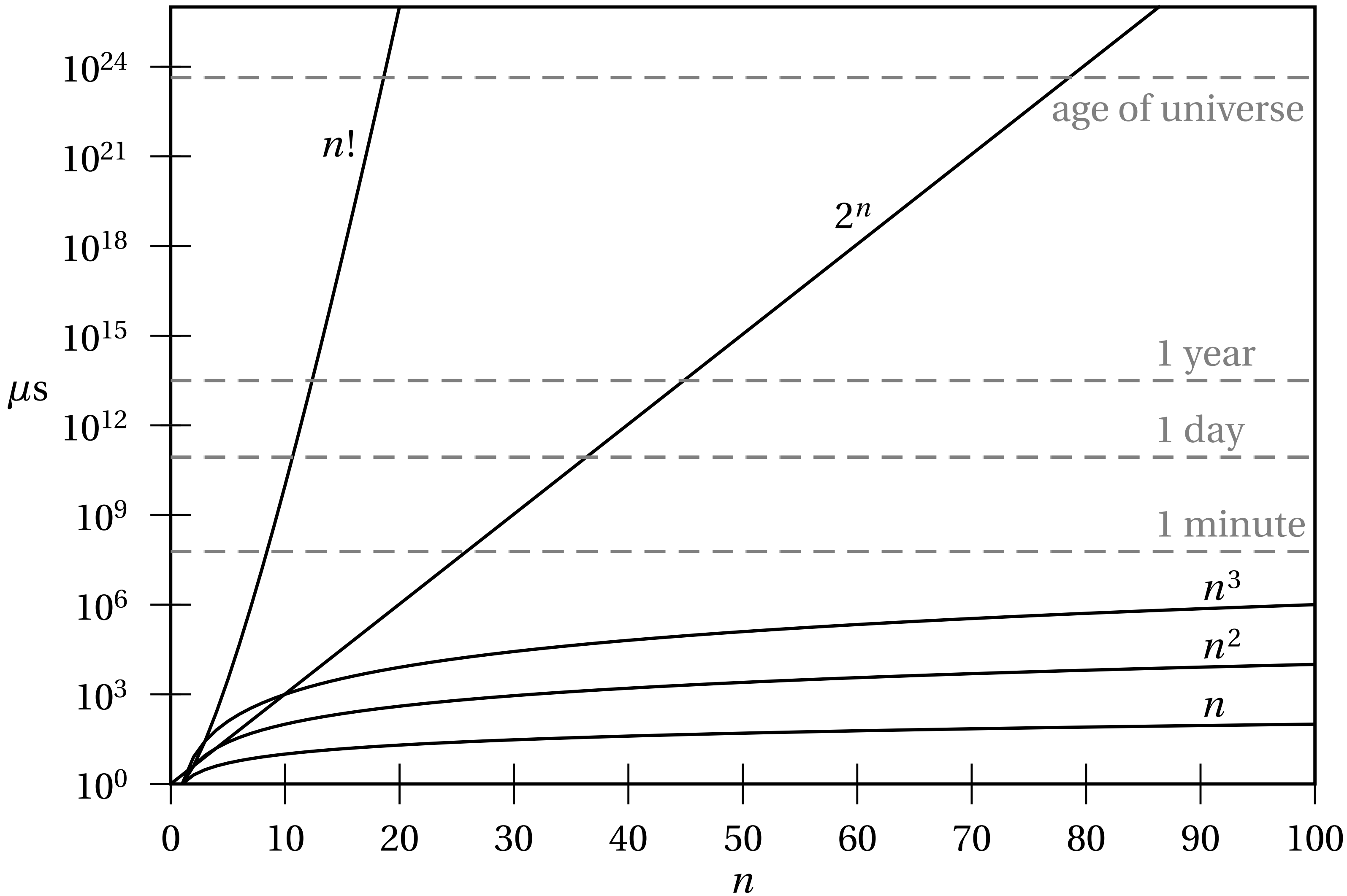
Backtracking search: follow a path until you get stuck, then backtrack to your last choice

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

When can we avoid this kind of search?

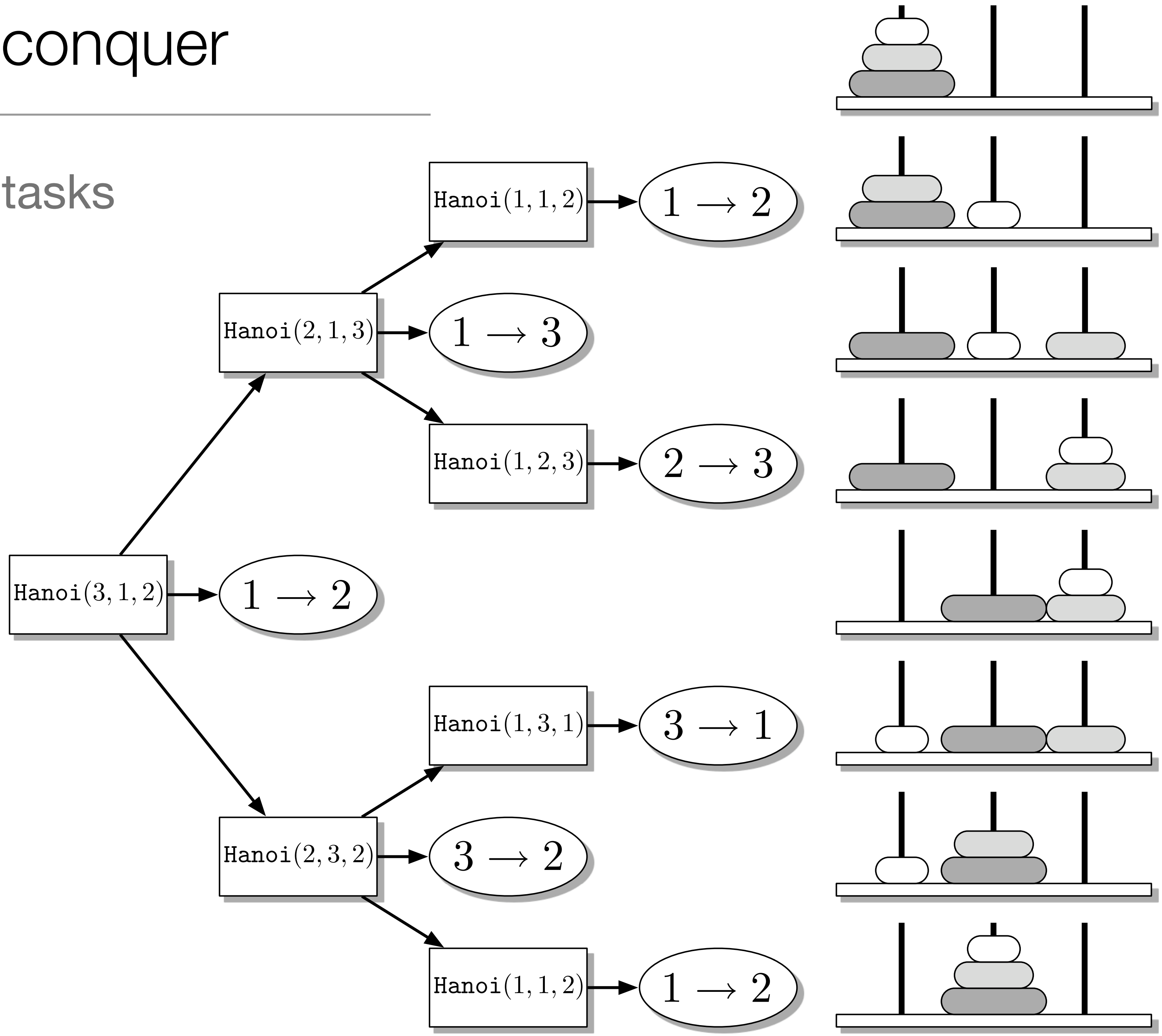


Until the end of the world

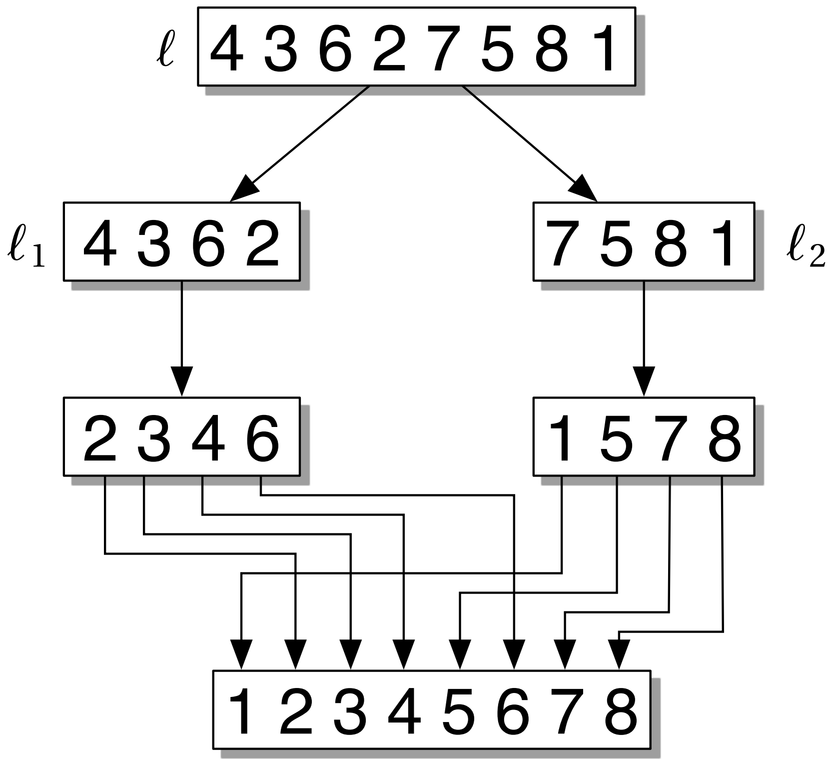


Divide and conquer

Tasks and subtasks



Divide and conquer: mergesort

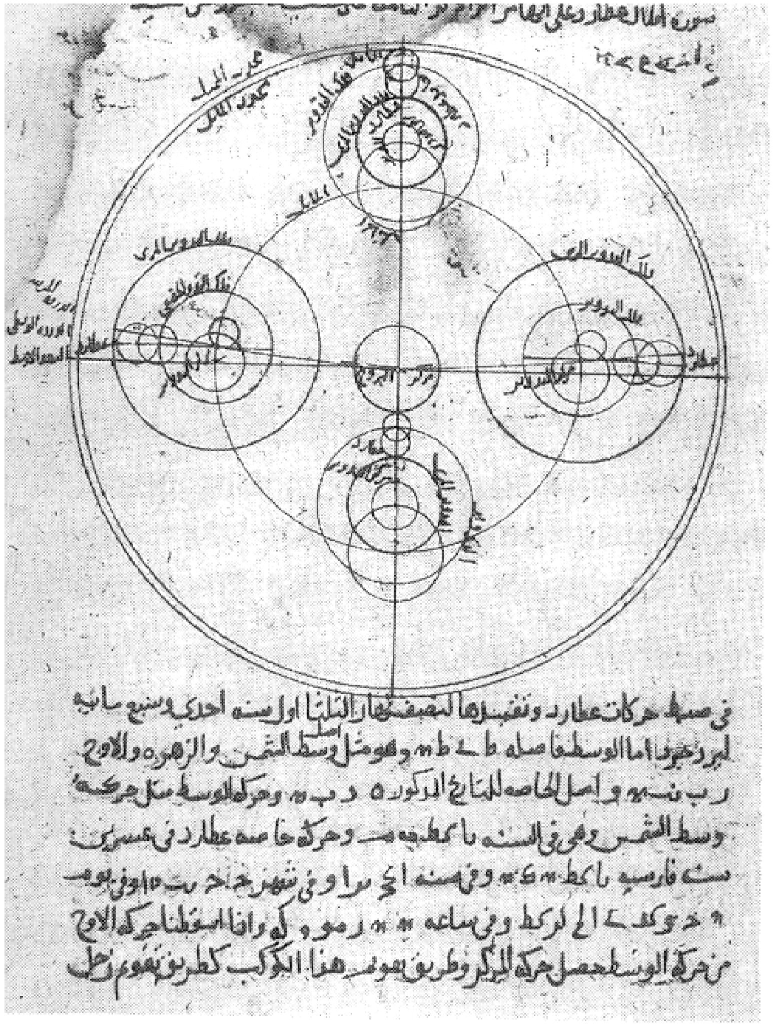


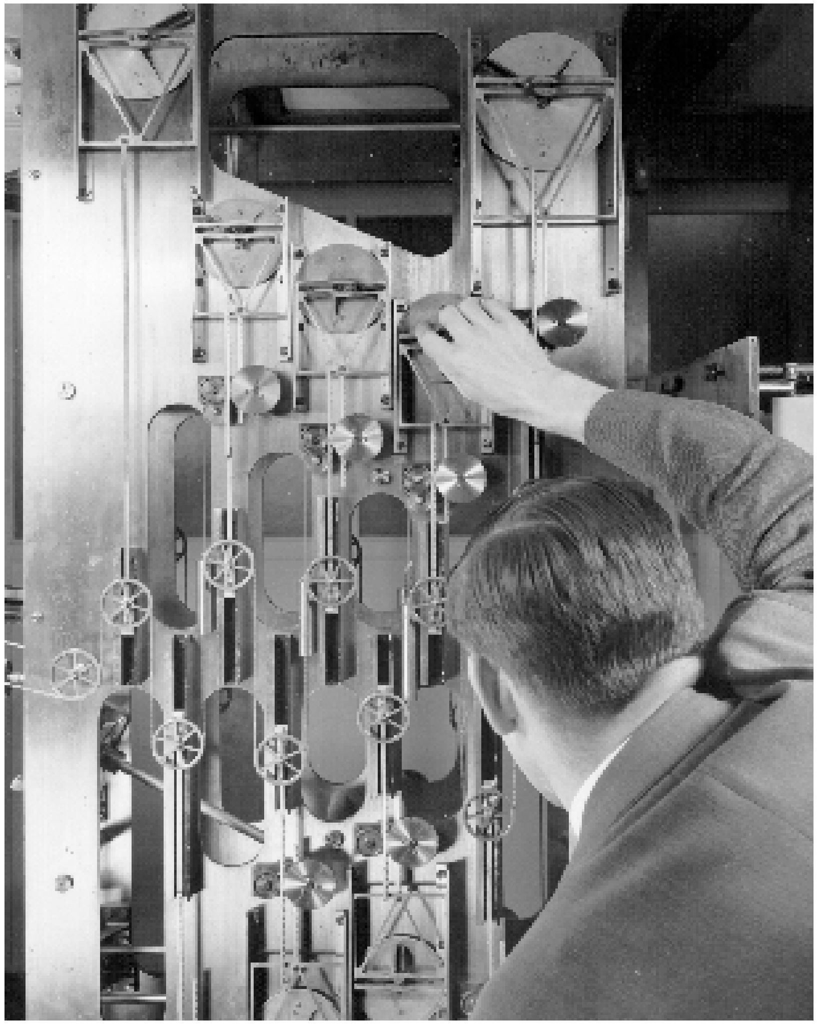
$$T(n) = 2T(n/2) + n$$

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

Divide and conquer

The Fast Fourier Transform

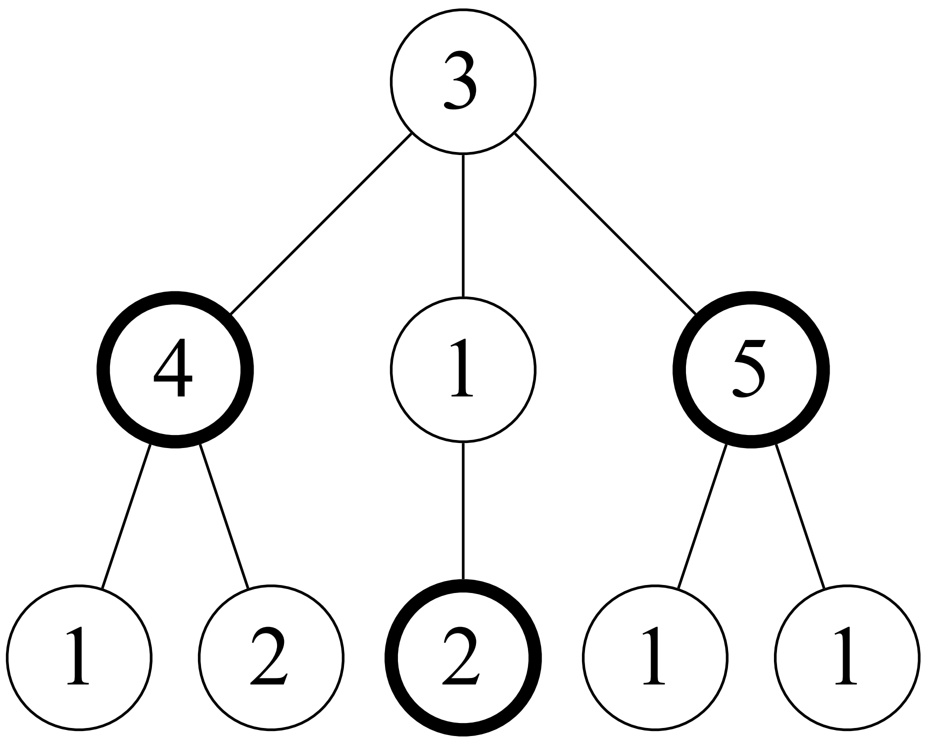


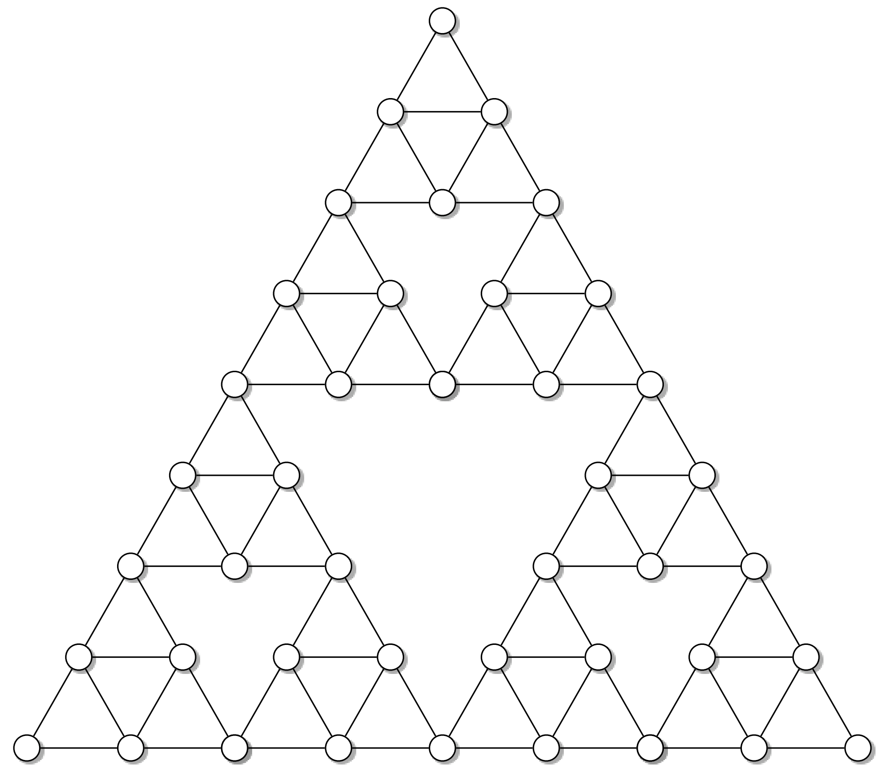


Divide and conquer again: dynamic programming

Subproblems become independent after we make some choices

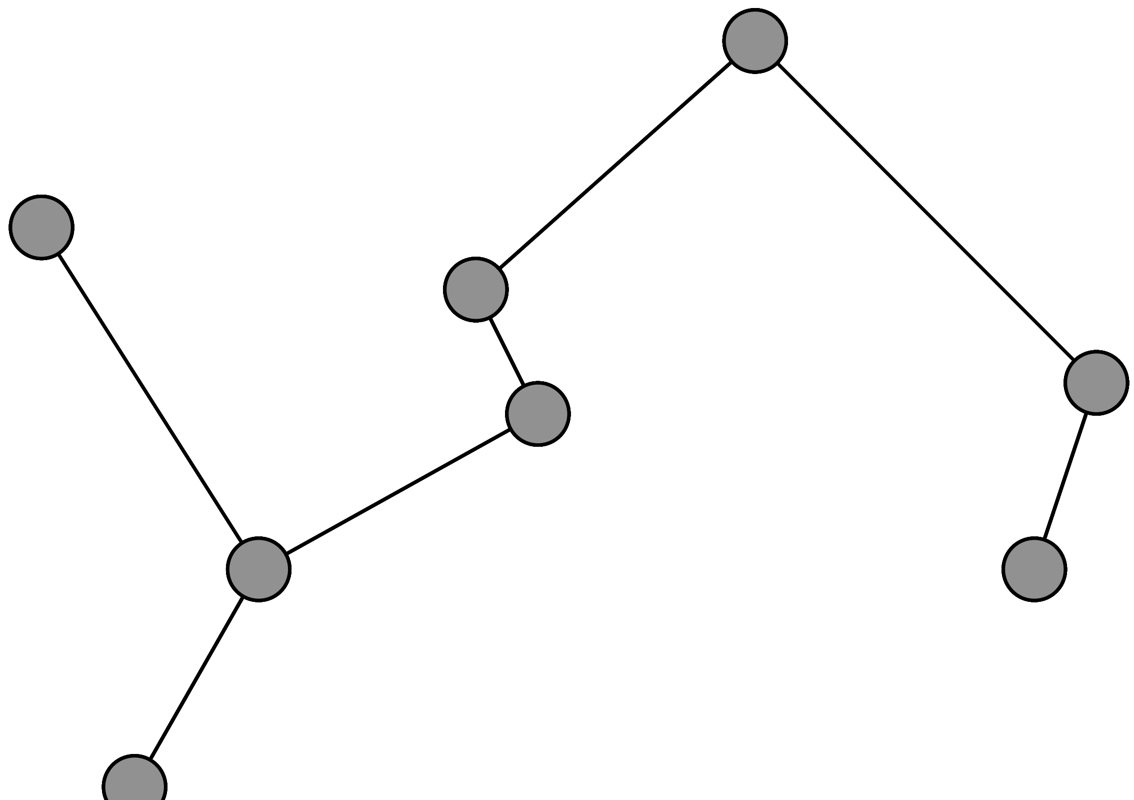
The “boundaries” between subproblems are small





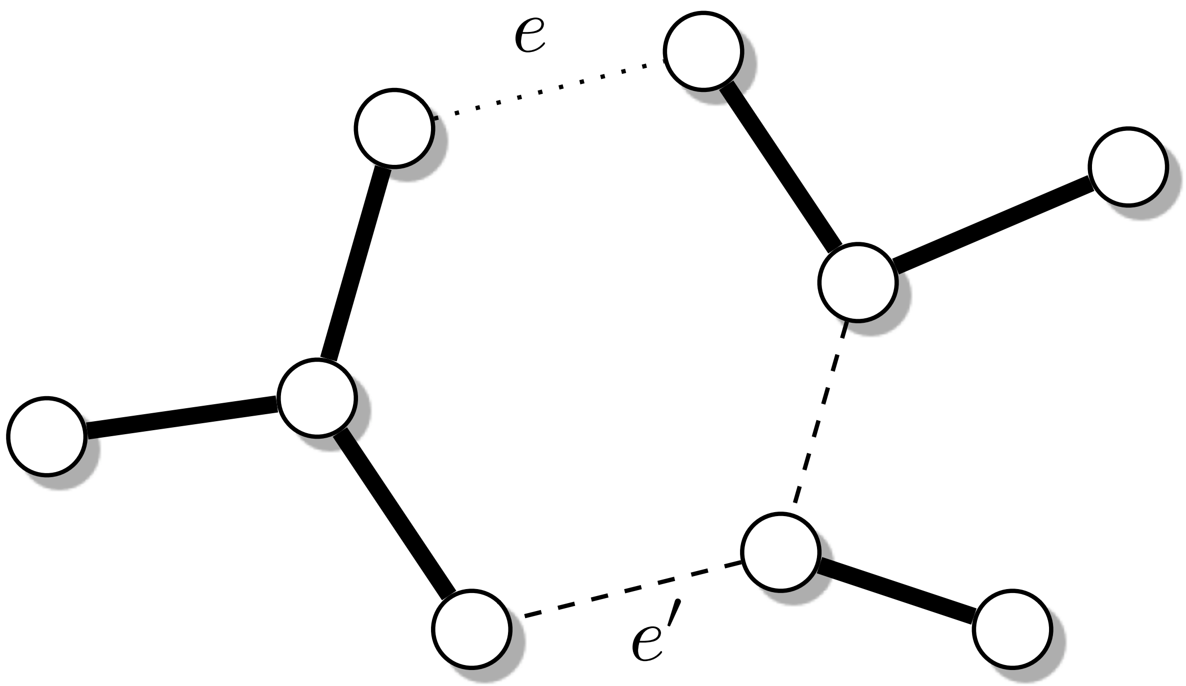
When greed is good

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



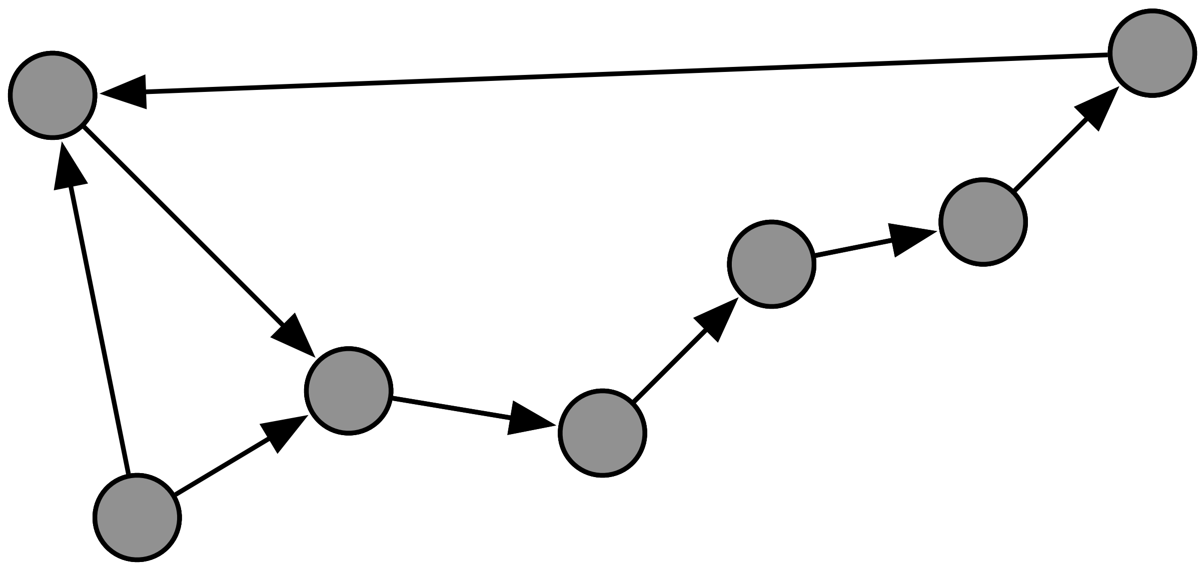
When greed is good

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

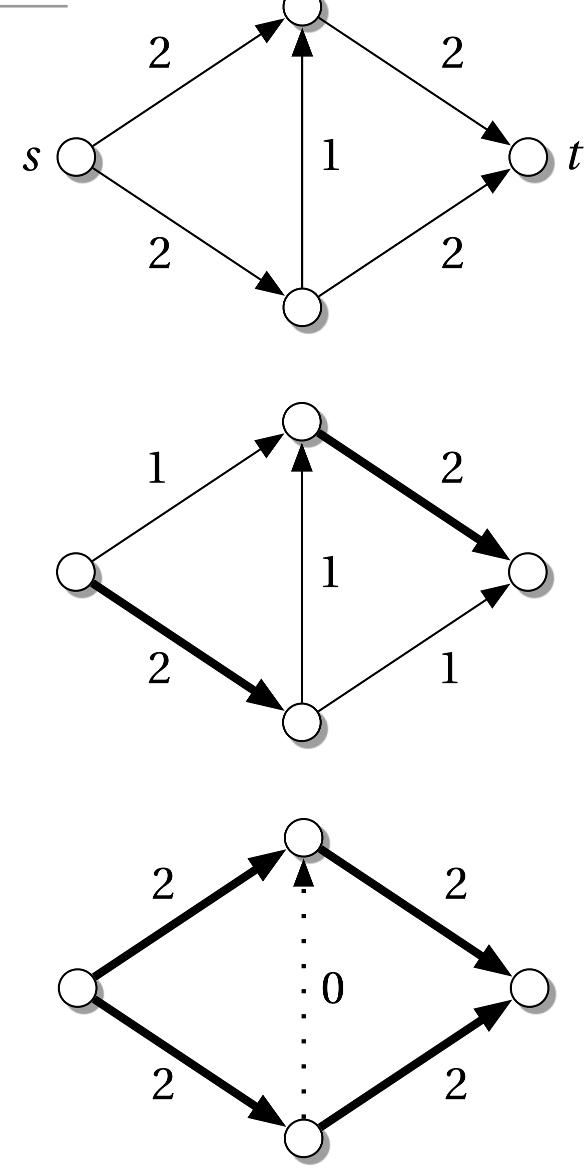
Many local optima where we can get stuck

Reorganizing the landscape: max flow

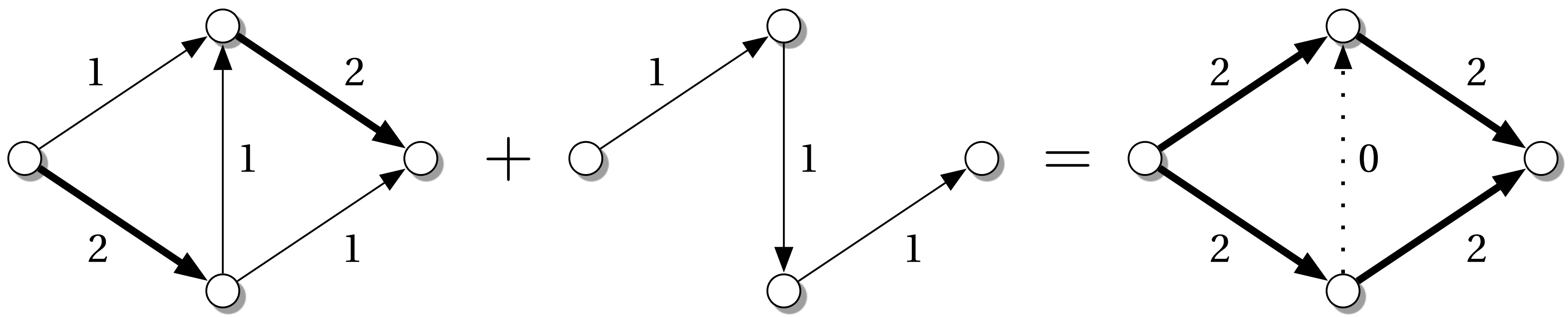
Each edge has a capacity

Greedy: push more flow along a path with excess capacity

But we can get stuck in local optima!



Reorganizing the landscape: max flow



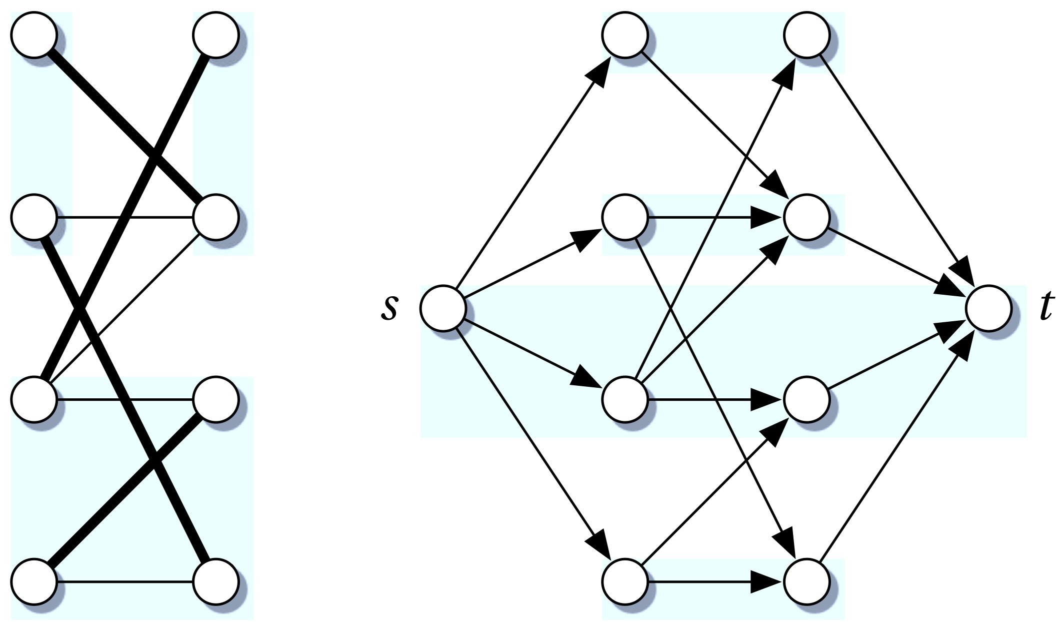
Solution: allow reverse edges to cancel out previous flow

Theorem: now we can't get stuck

Lesson: we define the neighborhood, by defining what moves are possible

“Reducing” one problem to another

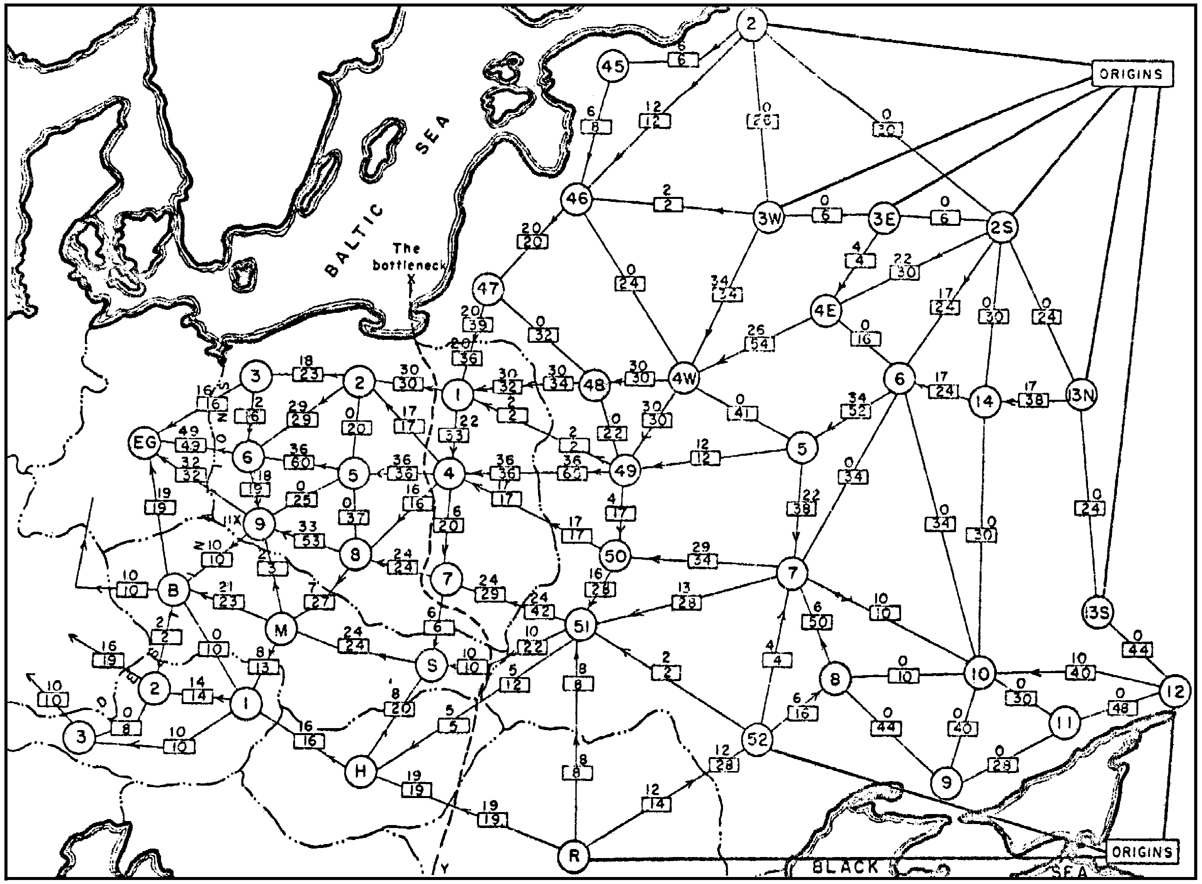
Change a new problem into one we already know how to solve:



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

What are algorithms for?

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

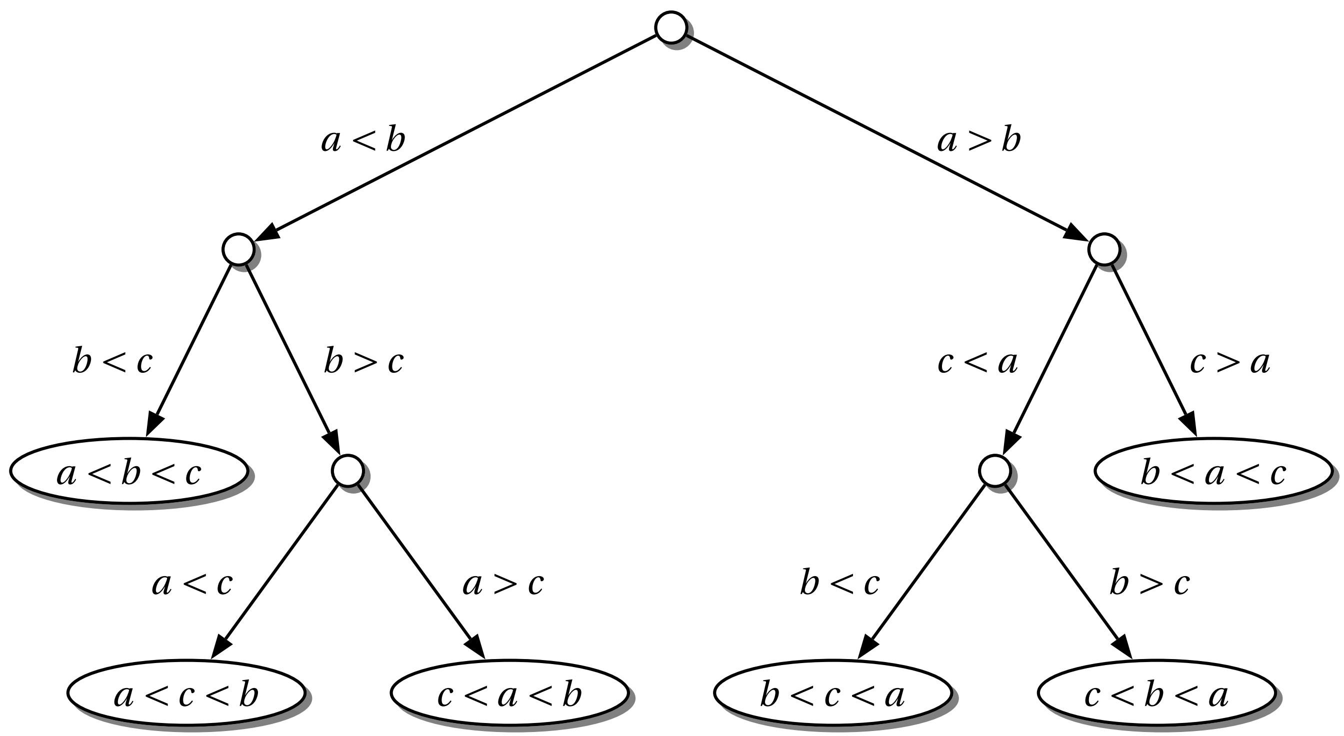
Intrinsic complexity of a problem: the running time (or memory use, or other resource) of the *best possible algorithm* for it

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 an algorithm is optimal?



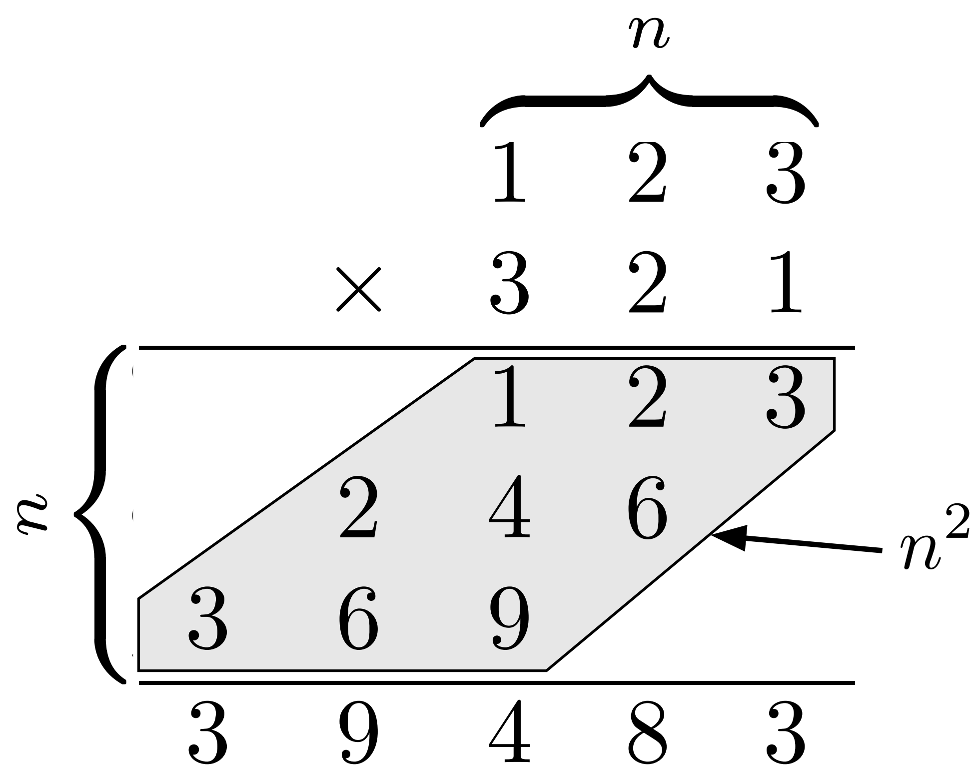
Twenty questions: can only distinguish 2^{20} situations

Sorting: need $\log_2 n! \approx n \log_2 n$ comparisons

Information, not computation!

How can we tell an algorithm is optimal?

Grade school multiplication takes O(n^2) time:



Can we do better?

Surprise!

Divide and conquer:

$$x = 2^{n/2}a + b,\ y = 2^{n/2}c + d$$

$$xy = 2^n ac + 2^{n/2}(ad + bc) + bd$$

Looks like we need ac, ad, bc, bd. But

$$(a+b)(c+d) - ac - bd = ad + bc$$

so we only need three products. Running time:

$$T(n) \approx 3T(n/2)$$

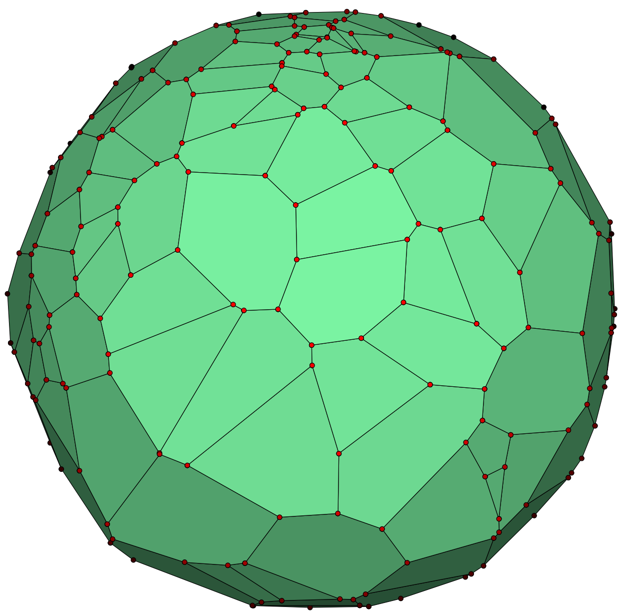
so $T(n) \sim n^{\log_2 3} = n^{1.58}$. How low can we go?

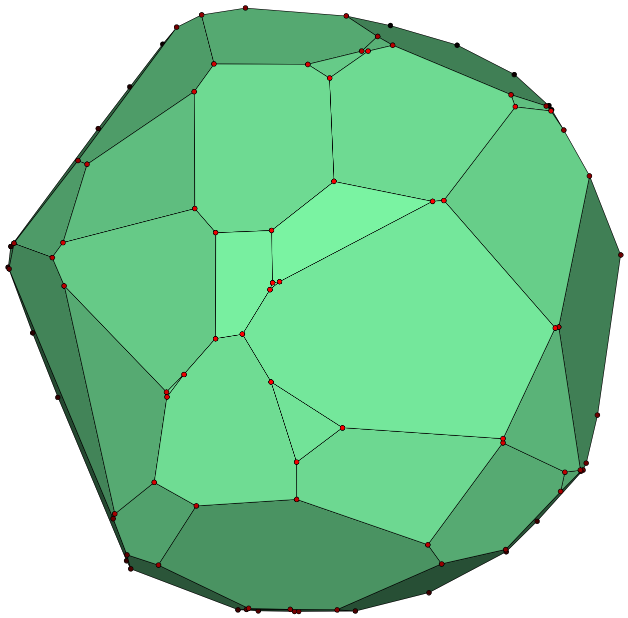
From theory to the real world

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





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

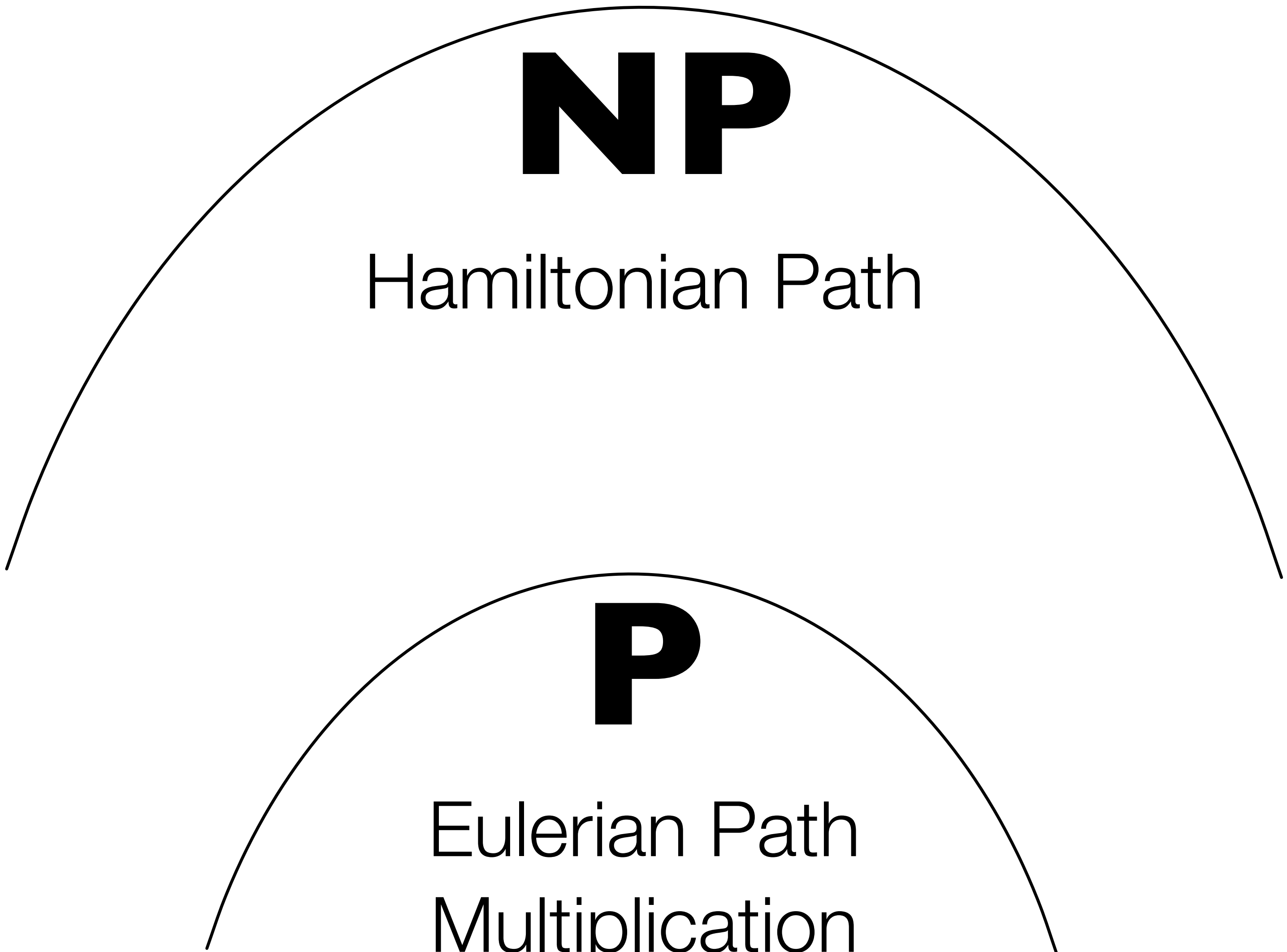
Cristopher Moore
Santa Fe Institute

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

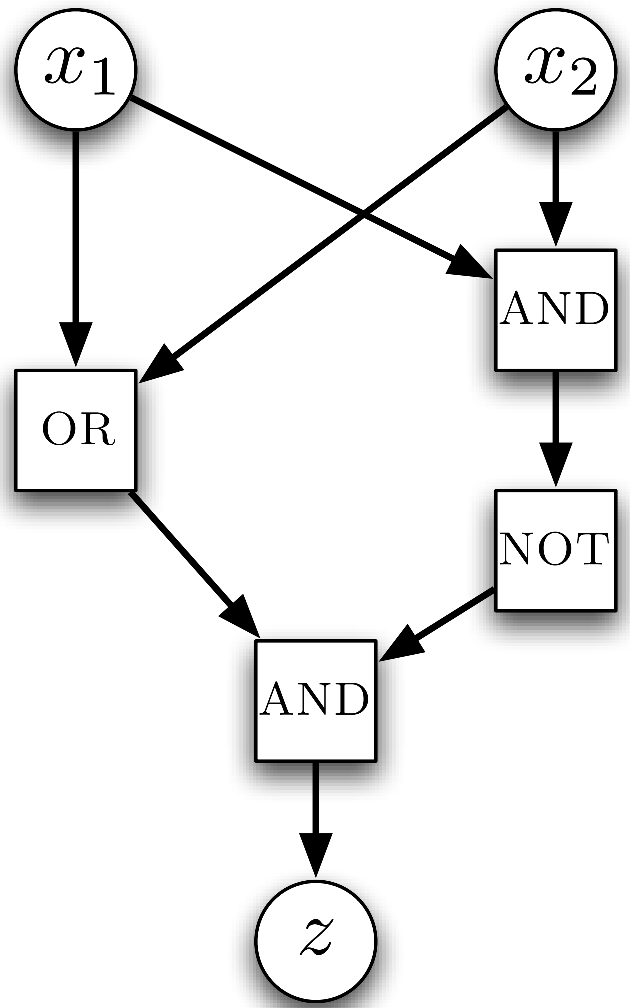
How can a single problem express every other problem in NP?

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?

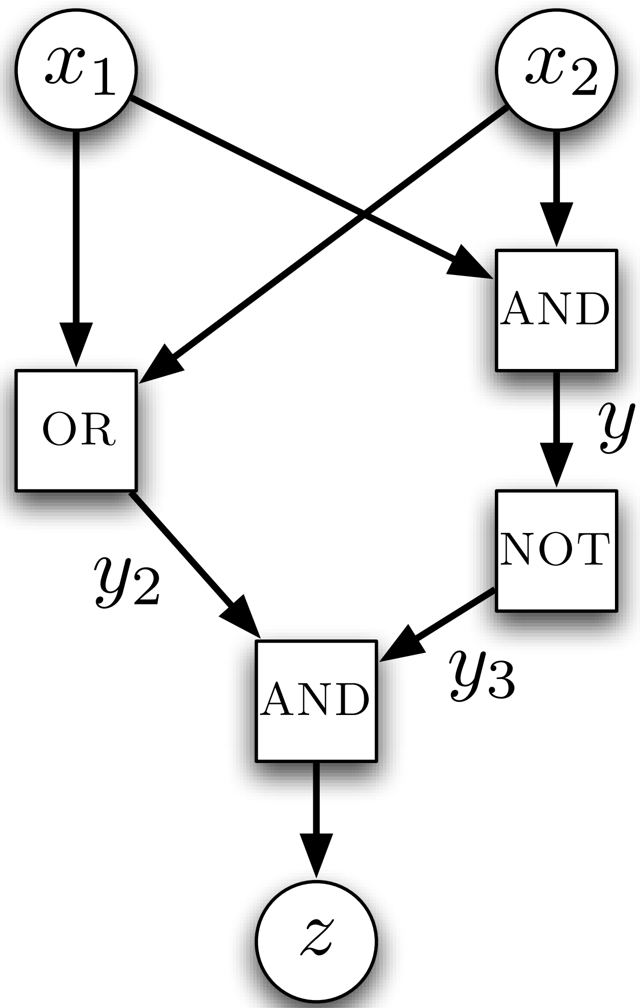


From circuits to formulas

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 convert any problem in NP to a circuit that checks solutions,

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**

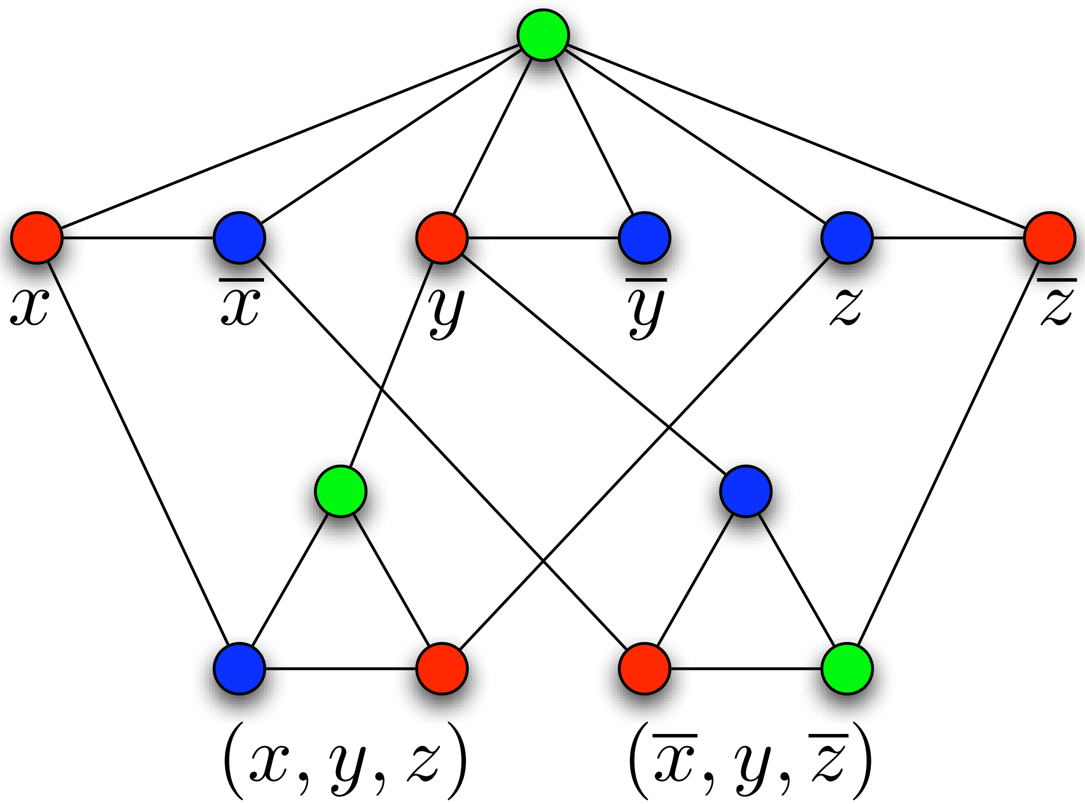
Conversely, if P≠NP, then 3-SAT cannot be solved in polynomial time: something like exhaustive search is needed

Graph coloring

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

From SAT to Coloring

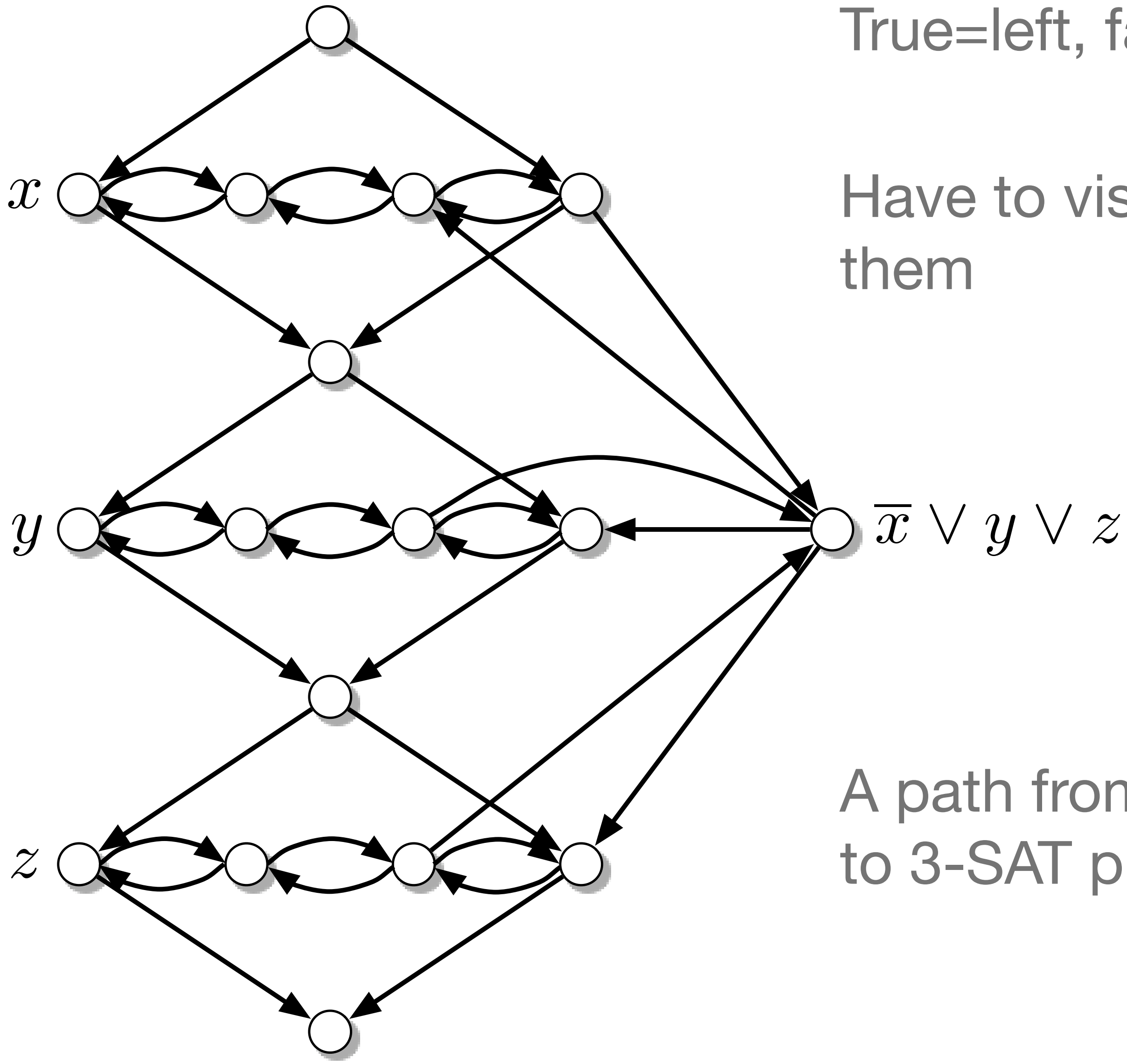
"Gadgets" enforce constraints:



Graph 3-Coloring is NP-complete

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

Traveling Salespeople

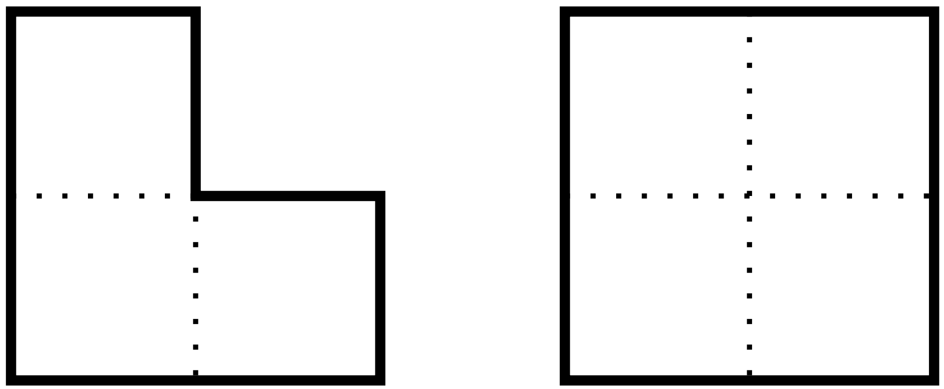


True=left, false=right

Have to visit clause vertices to satisfy them

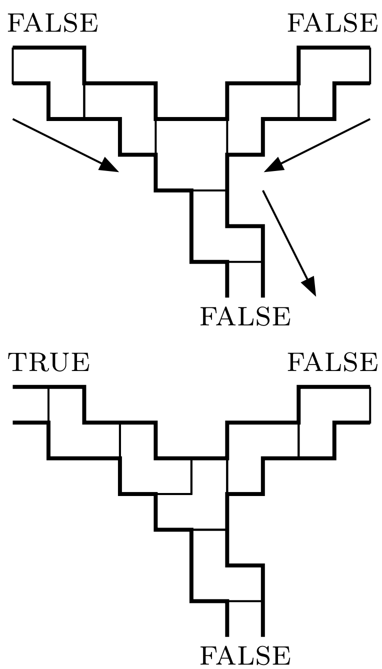
A path from top to bottom = solution to 3-SAT problem!

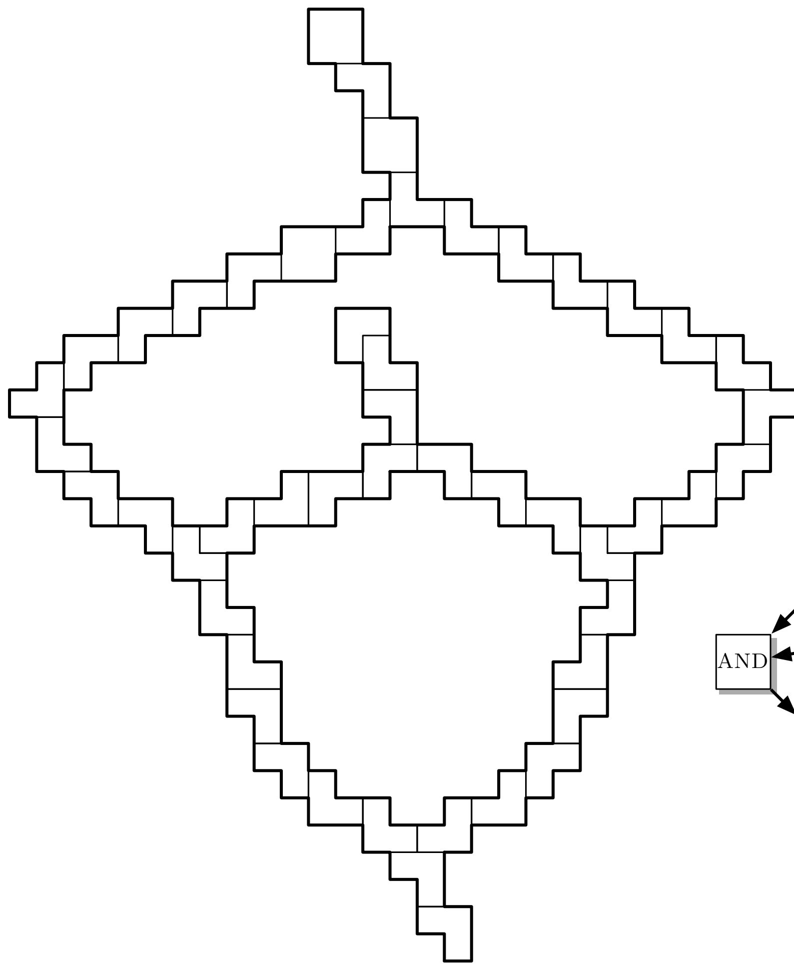
Why are some problems NP-complete?

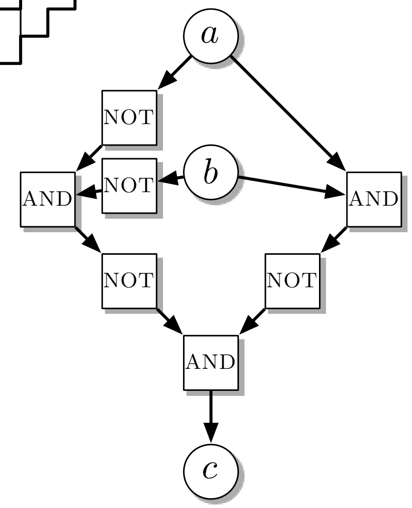


Can we tile a shape with these tiles?

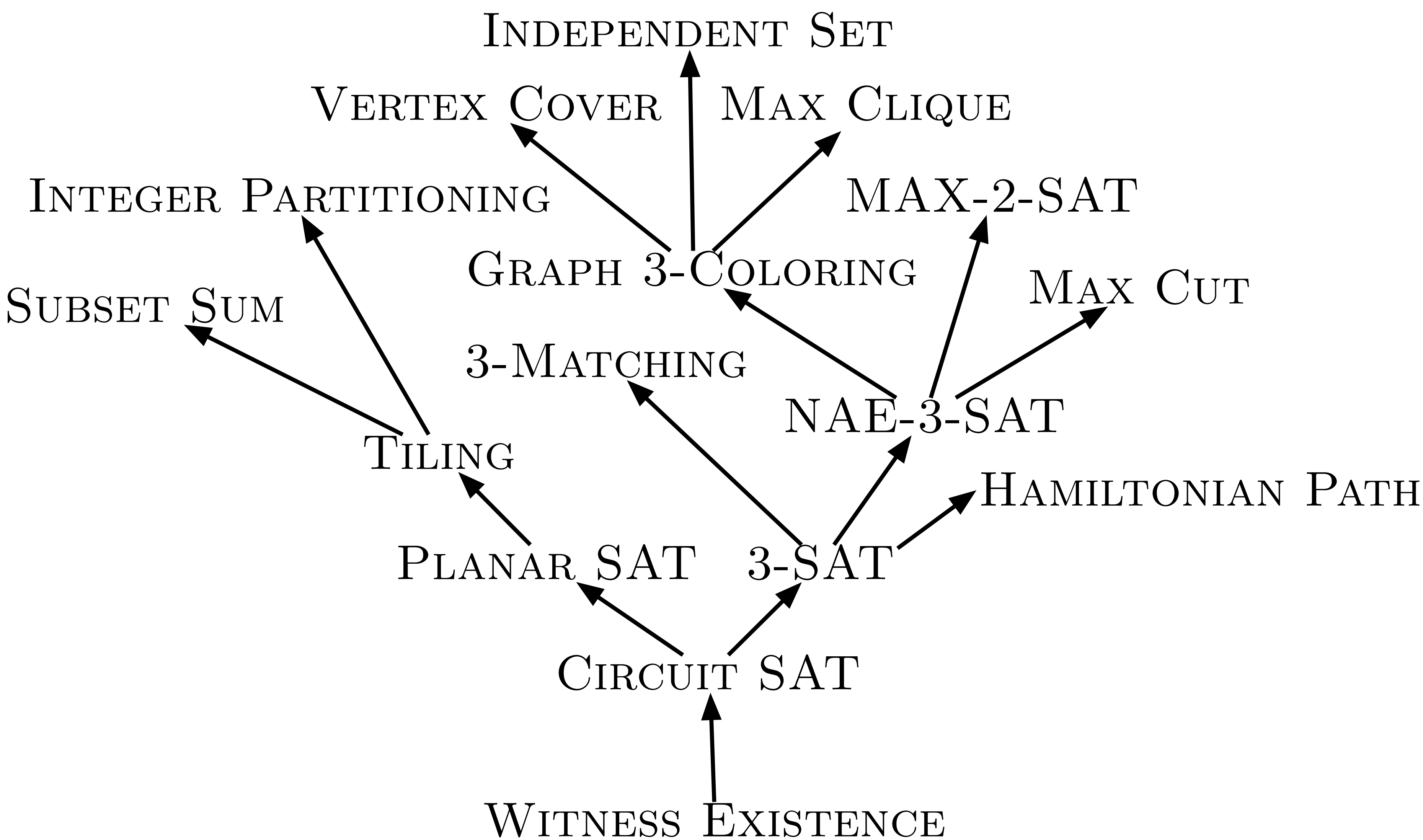
Because we can use them to build a computer.







Thousands of NP-complete problems



The P vs. NP problem

If any NP-complete problem can be solved in polynomial time, then they all can be, and P=NP

That would mean that *anything that is easy to check* — like a needle in a haystack — *is also easy to find.*

Better traveling salesmen; easier to tile bathroom floor, and pack our luggage; every cryptosystem can be broken.

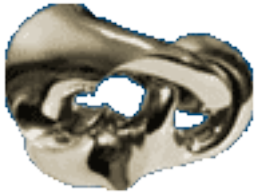
But P vs. NP turns out to be question about the nature of mathematical truth and creativity.

Gödel to Von Neumann

Let $\varphi(n)$ be the time it takes to tell if a proof of length n or less exists...

The question is, how fast does $\varphi(n)$ grow for an optimal machine. One can show that $\varphi(n) \geq Kn$. If there actually were a machine with $\varphi(n) \sim Kn$ (or even only $\varphi(n) \sim Kn^2$), this would have consequences of the greatest magnitude. That is to say, it would clearly indicate that, despite the unsolvability of the *Entscheidungsproblem*, the mental effort of the mathematician in the case of yes-or-no questions could be completely replaced by machines (footnote: apart from the postulation of axioms). One would simply have to select an n large enough that, if the machine yields no result, there would then be no reason to think further about the problem.

Millennium problems



P=NP?

Poincaré Conjecture

Riemann Hypothesis

Yang-Mills Theory

Navier-Stokes Equations

Birch and Swinnerton-Dyer Conjecture

Hodge Conjecture

Upper Bounds are Easy; Lower Bounds are Hard

Why is the P vs. NP question so hard?

Algorithms are upper bounds on complexity...

...but how do you know if you have the best algorithm?

Undecidability

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

```
Fermat
begin
    t := 3;
    repeat
        for n = 3 to t do
            for x = 1 to t do
                for y = 1 to t do
                    for z = 1 to t do
                        if x^n + y^n = z^n then return (x, y, z, n);
                    end
                end
            end
        end
        t := t + 1 ;
    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!

The time hierarchy theorem

This proves [Hartmanis and Stearns, 1965]

$$\mathrm{TIME}(n) \subset \mathrm{TIME}(n^2) \subset \mathrm{TIME}(n^{2.001}) \subset \cdots \subset \mathrm{P} \subset \mathrm{EXPTIME} \subset \cdots$$

Similar theorems for SPACE, NTIME, SPACE... can compare apples to apples

But can a similar argument separate P and NP?

We can't seem to diagonalize P within NP — but perhaps some other type of diagonalization will work?

Sadly, no...

Oracles and relativization

We can consult an oracle for a set *A*, asking her yes-or-no questions

P^A is the class of problems we can solve polynomial time, with her help

NP^A is the class of problems where we can check solutions in polynomial time, with her help

[Baker, Gill, Solovay 1975]: there exist oracles *A*, *B* such that

$$\mathrm{P}^A = \mathrm{NP}^A \quad \text{but} \quad \mathrm{P}^B \neq \mathrm{NP}^B$$

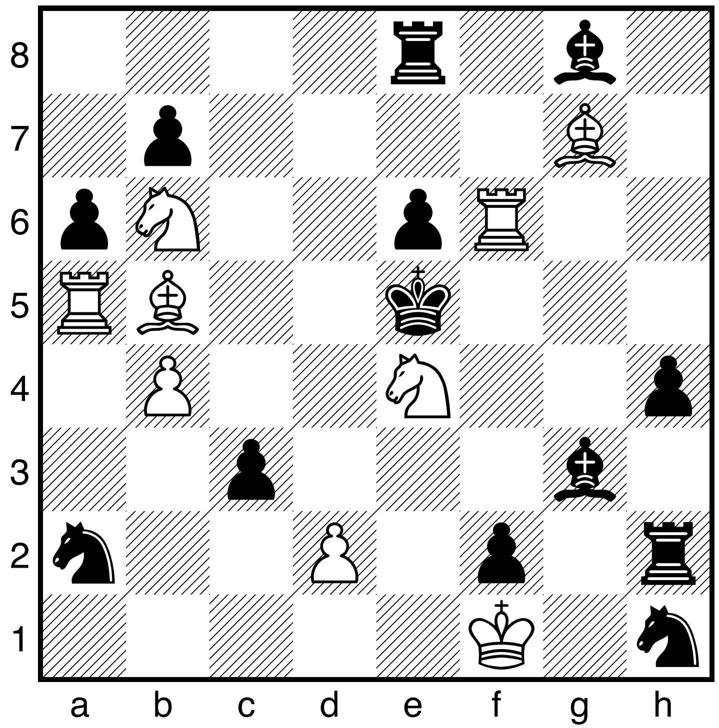
Logical hierarchies

I can win if there exists a move for me,

such that for all of your replies,

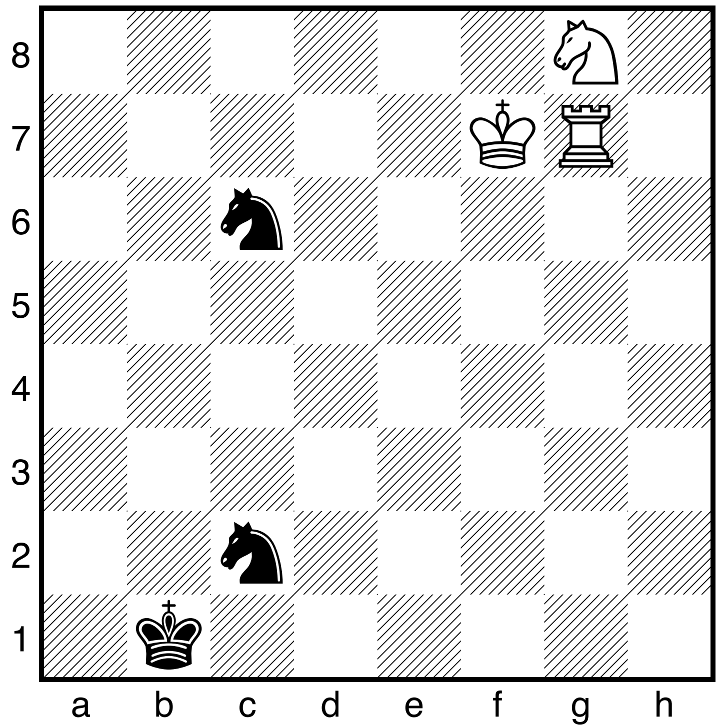
there exists a move for me...

Sam Loyd (1903)



Mate in 3

Lewis Stiller (1995)



Mate in 262

A powerful oracle

Adding poly(n) quantifiers to P gives the class

$$\mathrm{PSPACE} = \exists\forall\exists \cdots \mathrm{P}$$

Let A be a PSPACE-complete problem, such as Quantified SAT:

$$\exists x_1 : \forall y_1 : \exists x_2 : \cdots : \forall y_n : \phi(x_1, y_1, x_2, \ldots, y_n)$$

NP^A is P^A with another $\exists$. This just gives another instance of A, so

$$\mathrm{P}^A = \mathrm{NP}^A .$$

A random oracle

For each n=0, 1, 2, ... flip a coin

If Heads, choose a random string s_n of length n:
The oracle likes s_n, dislikes all others of length n

If Tails, the oracle dislikes all strings of length n

Haystack(n): is there a string of length n that the oracle likes?

In NP^B, since if the answer is “yes” we can prove it by asking her about s_n

But (with probability 1) not in P^B, since we have no hope of finding s_n among the 2^n possibilities in only poly(n) time.

$$P^B \neq NP^B .$$

A barrier to resolving the P vs. NP question

Since P vs. NP has different answers in different "possible worlds"...

...no proof technique that *relativizes* (works in the presence of oracles) can resolve it either way

includes diagonalization, and also ideas like exhaustive search:

$$\mathrm{NP} \subseteq \mathrm{EXPTIME}\,, \quad \mathrm{NTIME}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$$

"syntactic" manipulations of programs are not enough

Another barrier: natural proofs

We want to say that a function *f* is outside a complexity class C if *f* is "too complicated"

But if "complicatedness" is

common — i.e. random functions are complicated

constructive — it is fairly easy to compute from *f*'s truth table

then this gives a contradiction if C contains *pseudorandom* functions, and we think P does!

Defeats most known techniques in circuit complexity: random restrictions, Fourier methods, etc. [Razborov and Rudich, 1994]

The road ahead

It seems unlikely that P vs. NP is formally independent since it is almost first-order:

"For all $n \geq 1000$, there is no circuit of size $n^{\log n}$ that solves all SAT formulas of length n": if this is false, there is a finite proof.

Sophisticated approaches from algebraic geometry have been suggested [Mulmuley]

The most hopeful view: we will eventually prove that P≠NP...

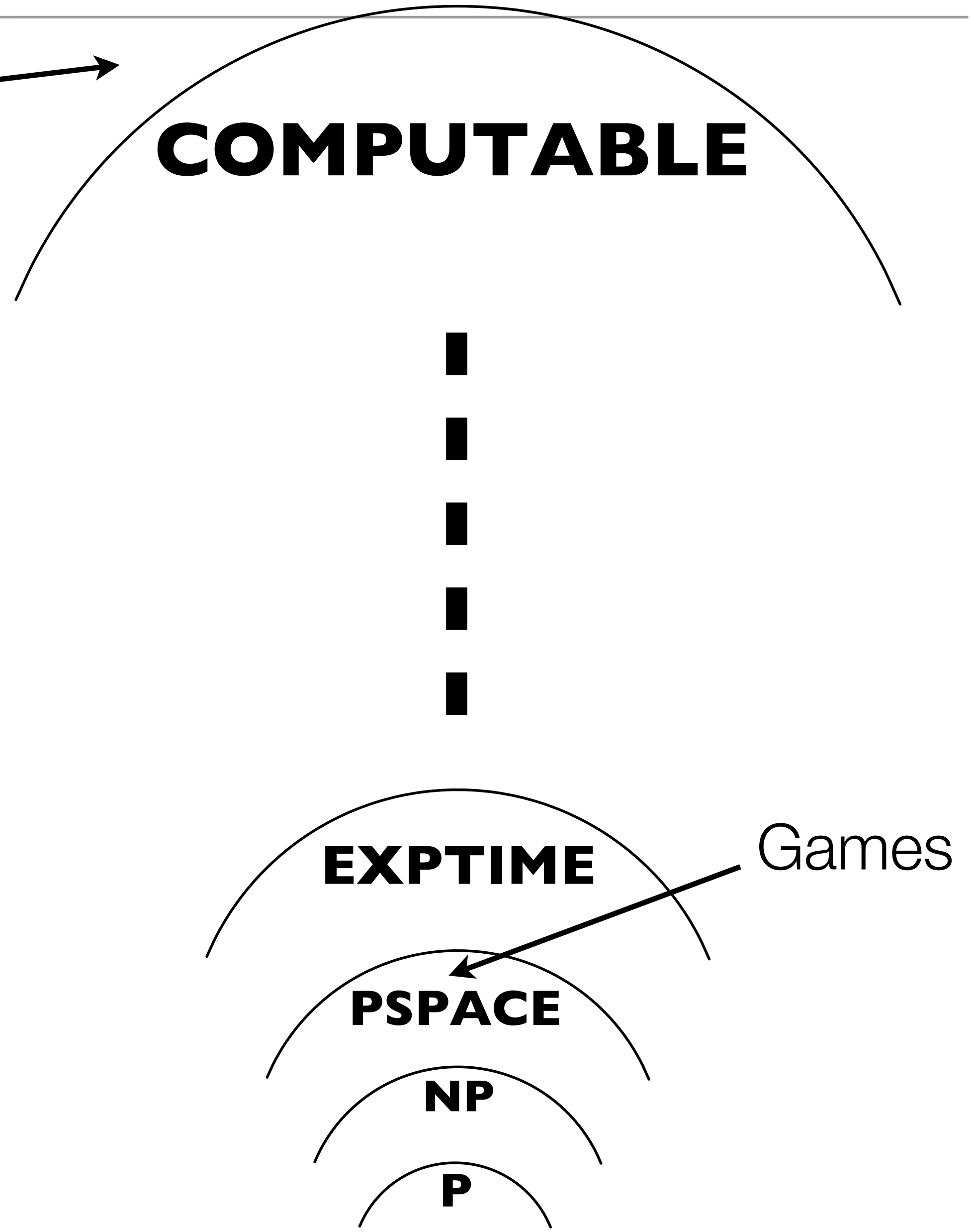
...but we will be forced to build a lot of new mathematics in the process!

Beyond NP

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

An infinite hierarchy

Turing's Halting Problem



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

Questions, questions...

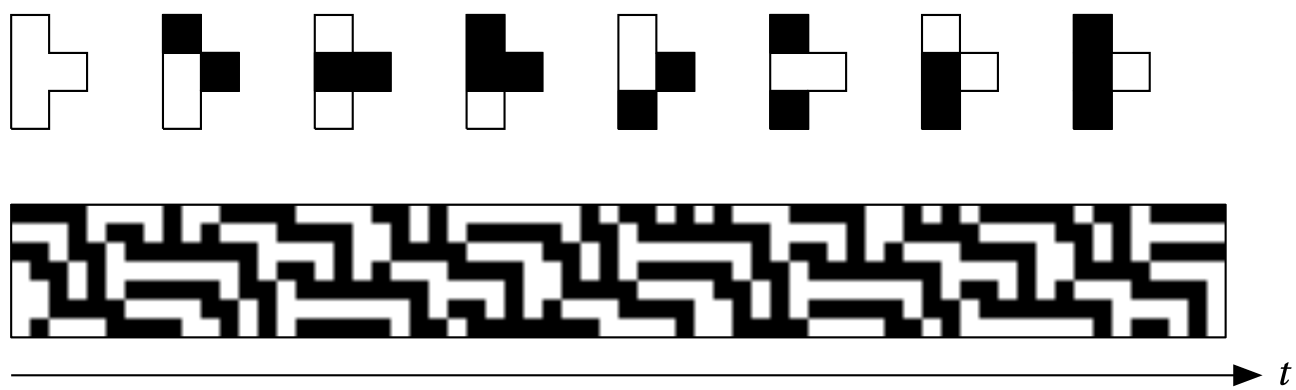
Suppose we have a cellular automaton. There are lots of questions we could ask about it:

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**



Problems in the gap

To prove that a problem is hard, we build a computer out of it

If P≠NP, problems exist that are outside P, but not NP-complete

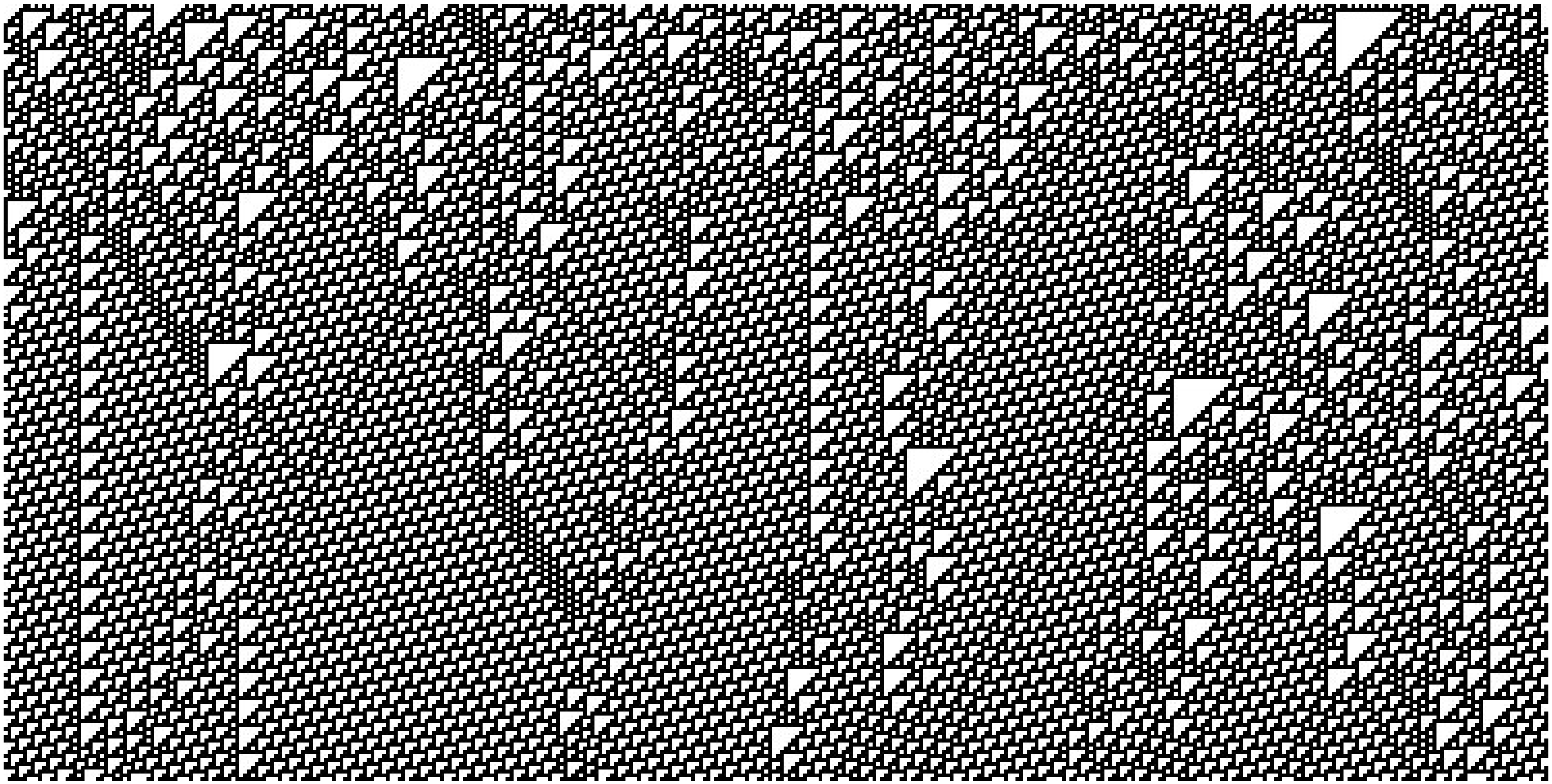
Factoring, Graph Isomorphism, Shortest Lattice Vector

Similarly there are undecidable problems to which Halting can't be reduced: "easier" than (or at least different from) than Halting

Could "naturally occurring" problems live in this middle ground?

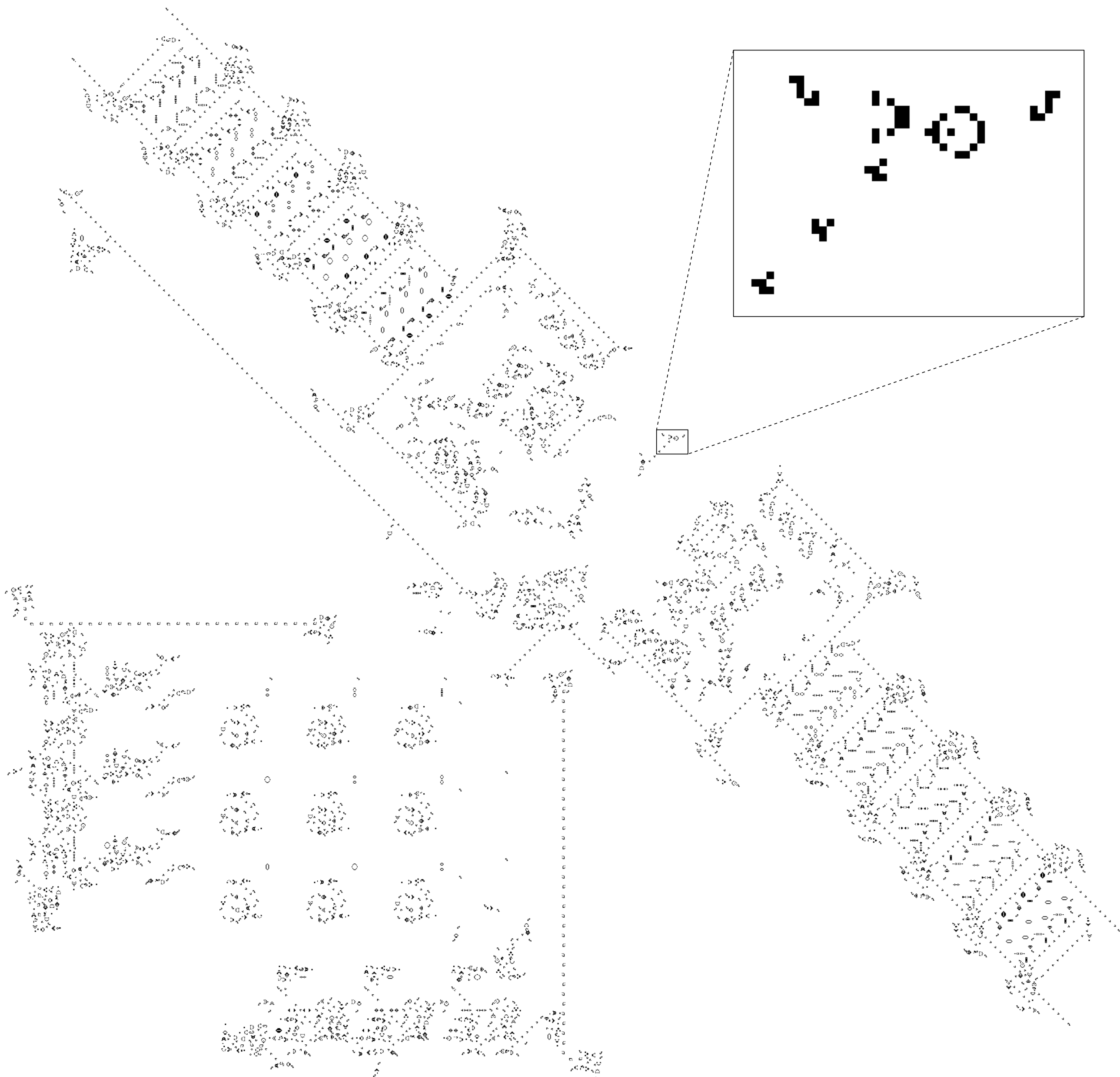
Claim: problems equivalent to Halting are easy (to think about, not to solve!)

Building a computer: cellular automata



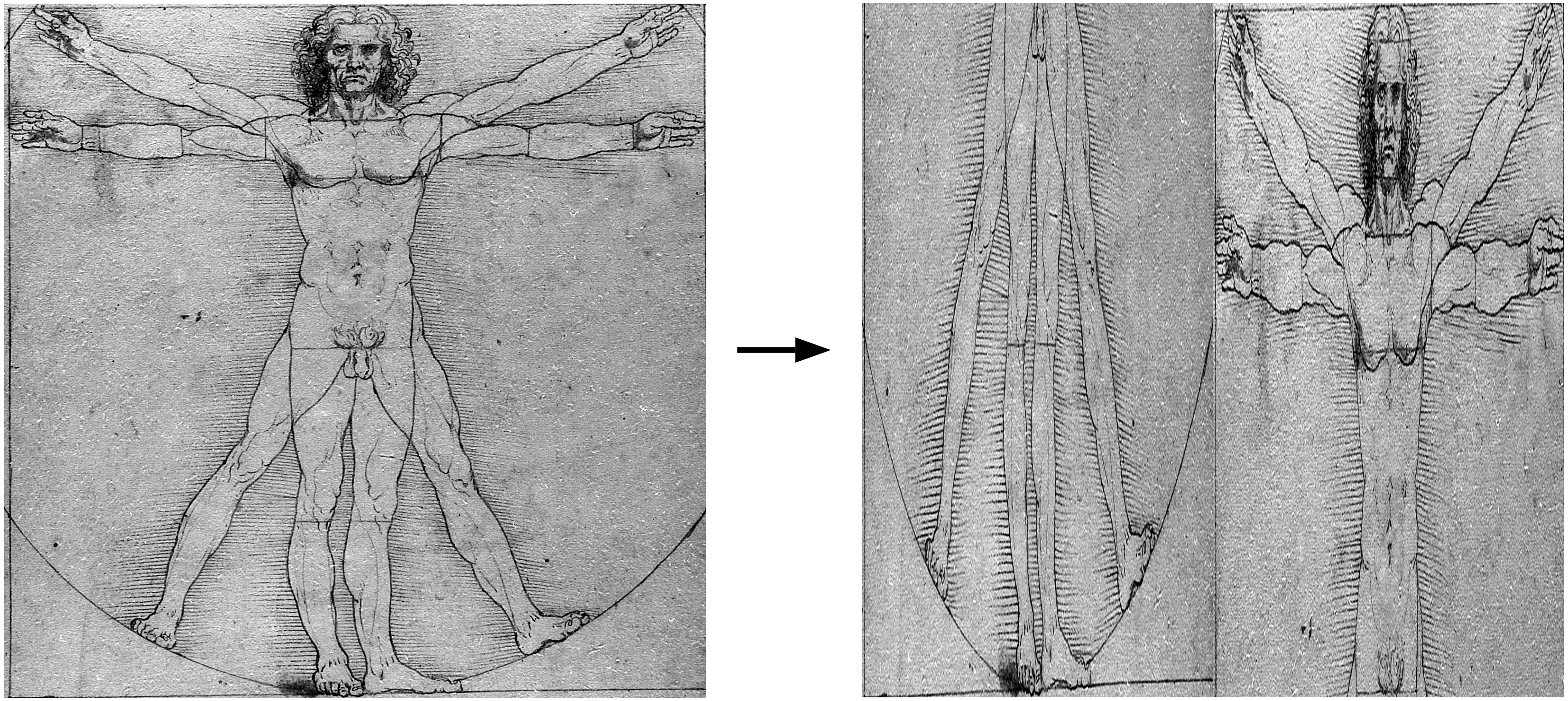
[Cook, Wolfram]

Building a computer: cellular automata



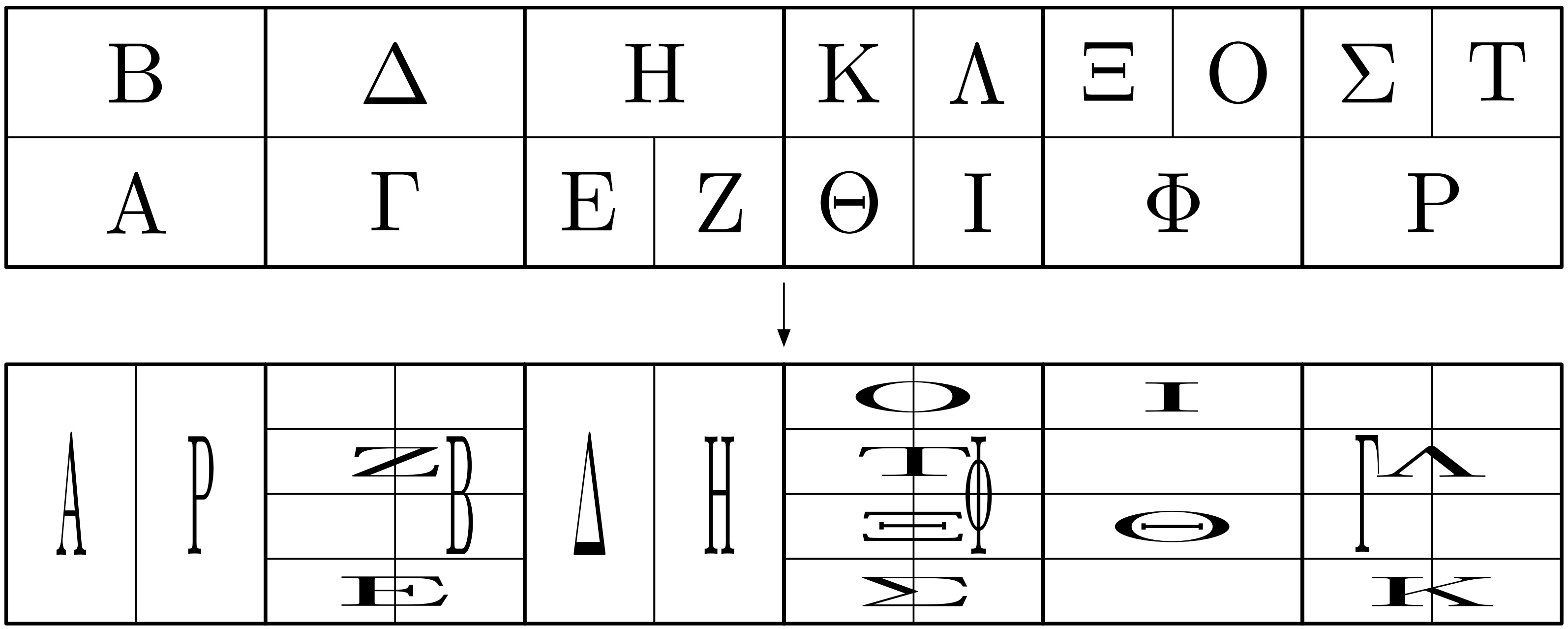
[Rendell]

Building a computer: dynamical systems



$$\ldots y_3 y_2 y_1 \,.\, x_1 x_2 x_3 \ldots$$
$$\Downarrow$$
$$\ldots y_3 y_2 \,.\, y_1 x_1 x_2 x_3 \ldots$$

Building a computer: dynamical systems

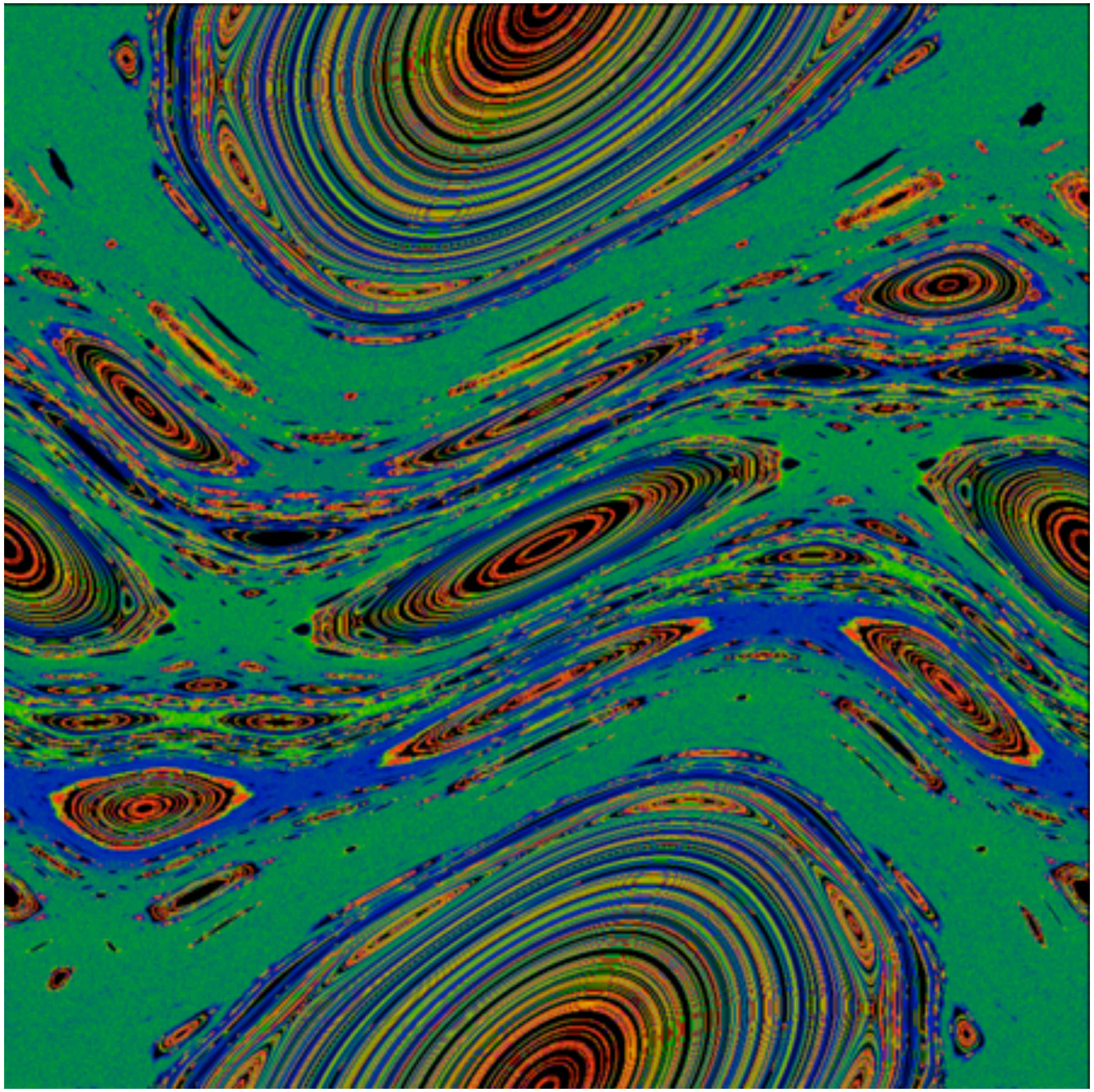


F	s_1	s_2	s_3	s_4	s_5	s_6
0	$0, s_1, L$	$0, s_6, L$	$0, s_2, R$	$1, s_5, R$	$1, s_4, L$	$1, s_1, L$
1	$1, s_2, L$	$0, s_3, L$	$1, s_3, L$	$0, s_6, R$	$1, s_4, R$	$0, s_4, R$

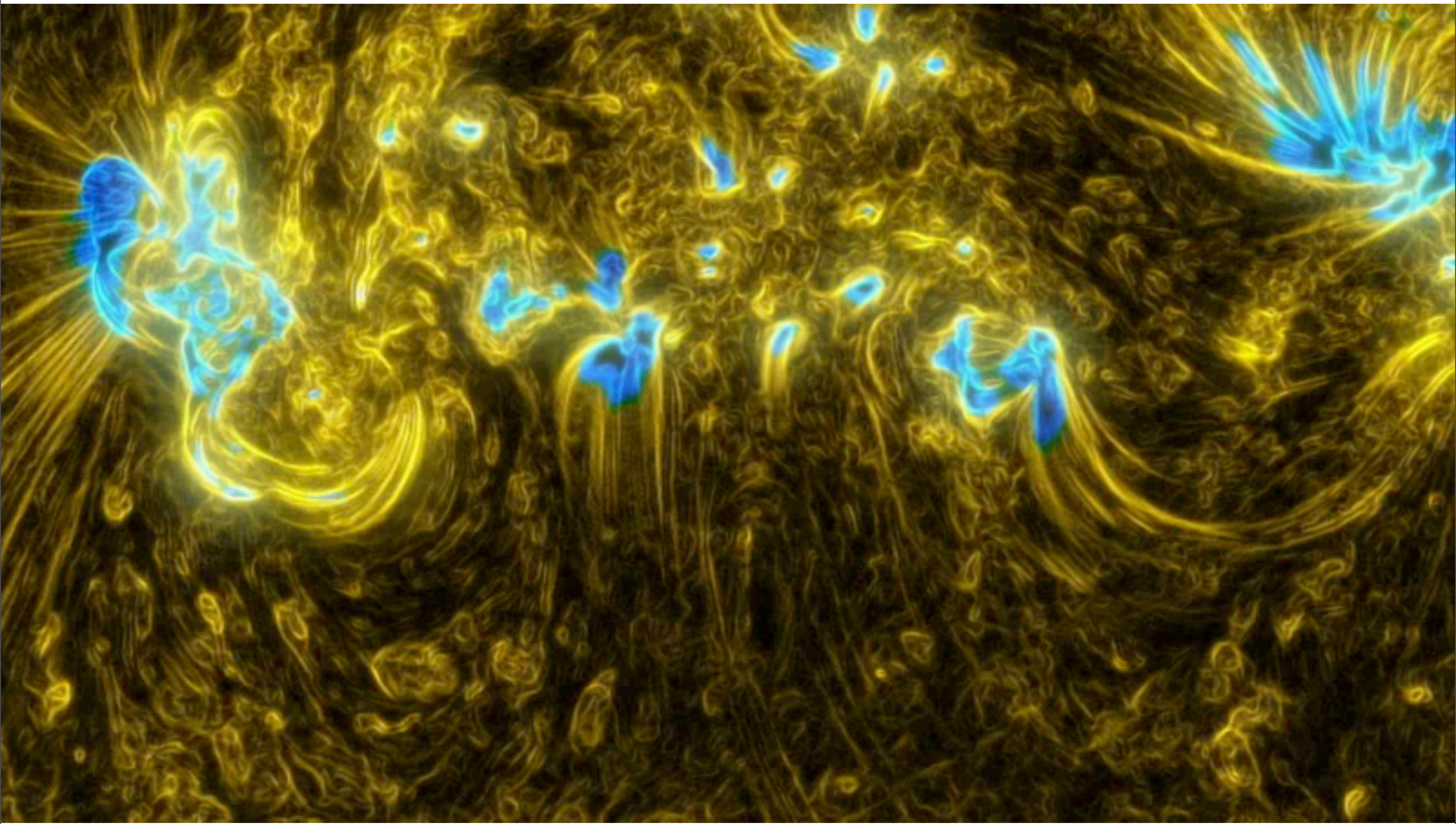
[Moore]

Wild problems

$$p_{t+1} = p_t + K \sin\theta_t$$
$$\theta_{t+1} = \theta_t + p_{t+1}$$



Wild problems



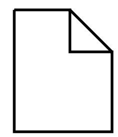
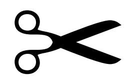
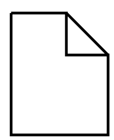
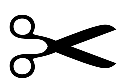
Computational Complexity 3:
The power of randomness

Cristopher Moore
Santa Fe Institute

Foiling the adversary

If the adversary knows what strategy we will use, he can create instances on which we will do badly

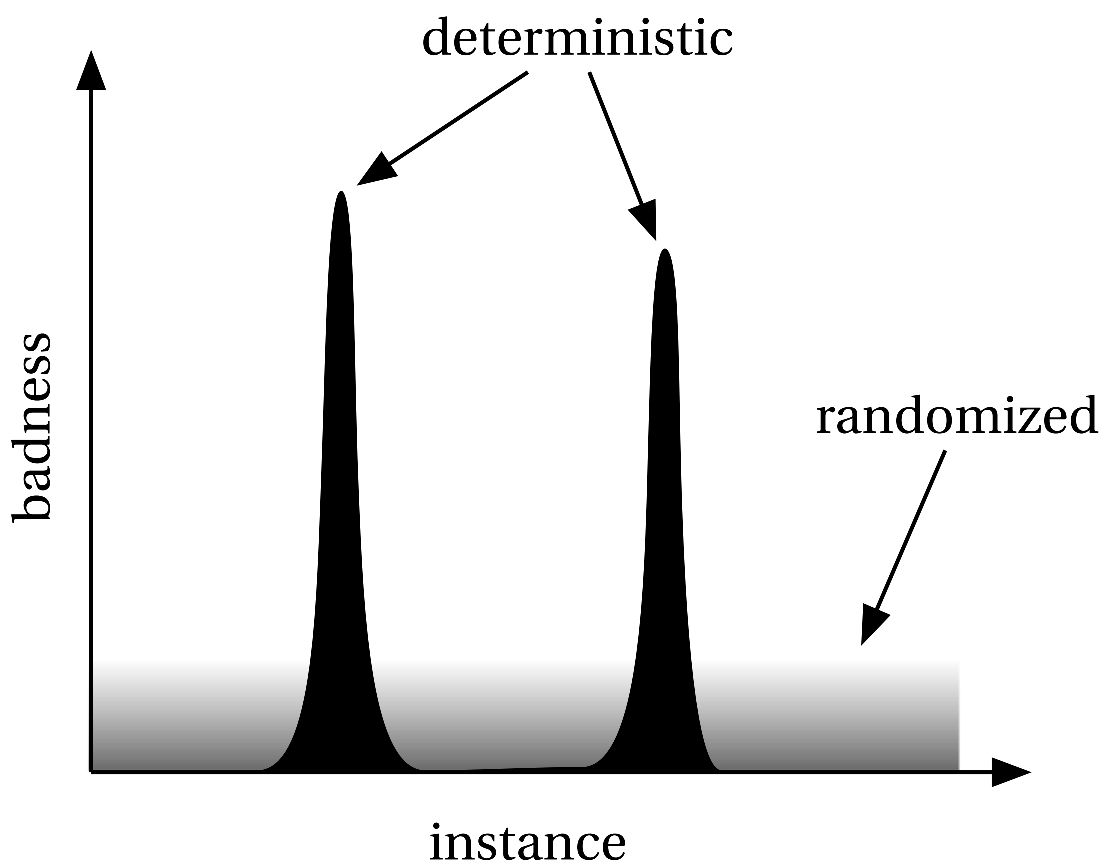
So, use an unpredictable algorithm!

			
	tie	lose	win
	win	tie	lose
	lose	win	tie

Foiling the adversary

Trade a small number of instances on which we do badly...

...for a small probability of doing badly on any instance



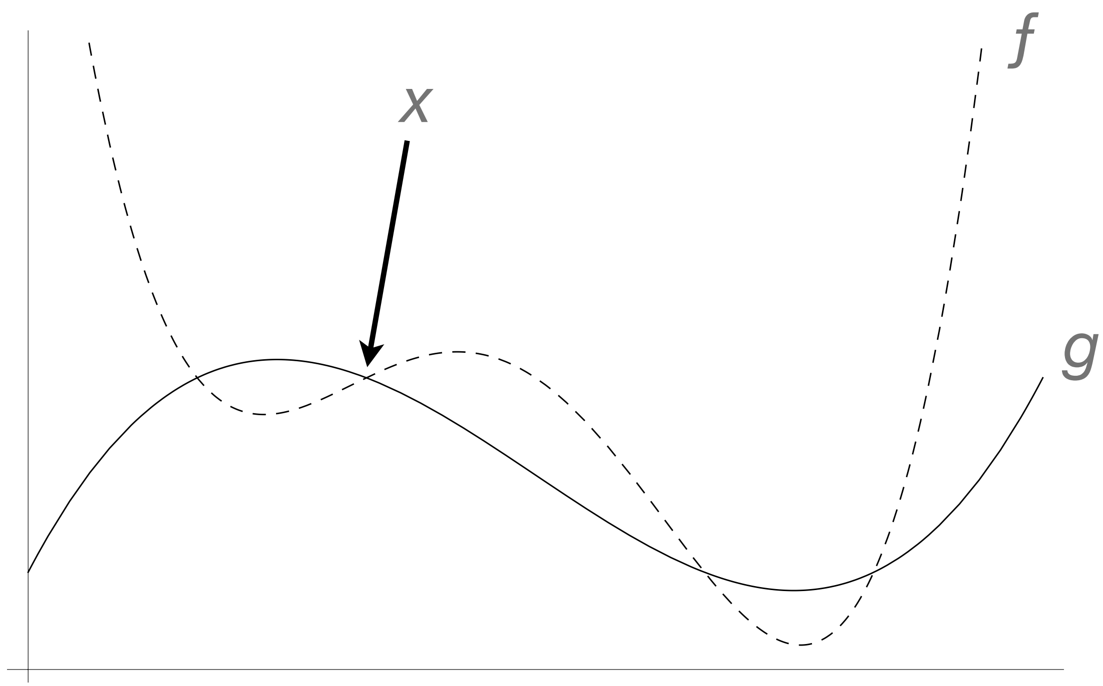
When are two functions the same?

I have two functions, $f(x)$ and $g(x)$

Both are complicated, but I want to know if $f=g$ (for all x)

Idea: choose x, and check if $f(x)=g(x)$

If the adversary knows what x we will use, he can fool us



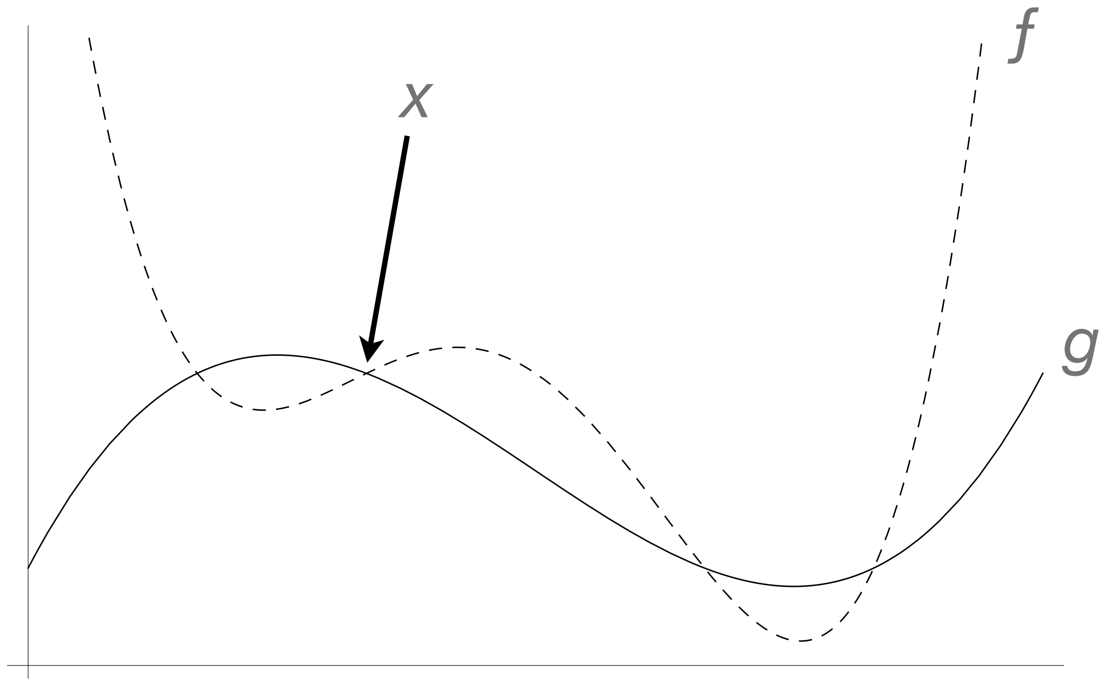
When are two functions the same?

Choose x randomly!

If $f(x)$ and $g(x)$ are polynomials of degree d, then so is $f(x)-g(x)$

But a polynomial of degree d can only have d roots, so $f(x)=g(x)$ for at most d values of x

If we choose x from t possibilities, he fools us with probability $\leq d/t$



Local search: WalkSAT

```
WalkSAT
```

input: A k-SAT formula ϕ
output: A satisfying assignment or "don't know"
begin
    start at a uniformly random truth assignment B ;
    repeat
        if B satisfies ϕ **then return** B ;
        else
            choose a clause c uniformly from among the unsatisfied clauses ;
            choose a variable x uniformly from among c's variables ;
            update B by flipping x ;
        end
    until we run out of time;
    return "don't know" ;
end

A random walk

Imagine a solution *A*

Let *d* be the Hamming distance between *A* and *B*, the number of variables on which they disagree

Each flip changes *d* by ±1: if we reach *d*=0, we're done

Worst case:

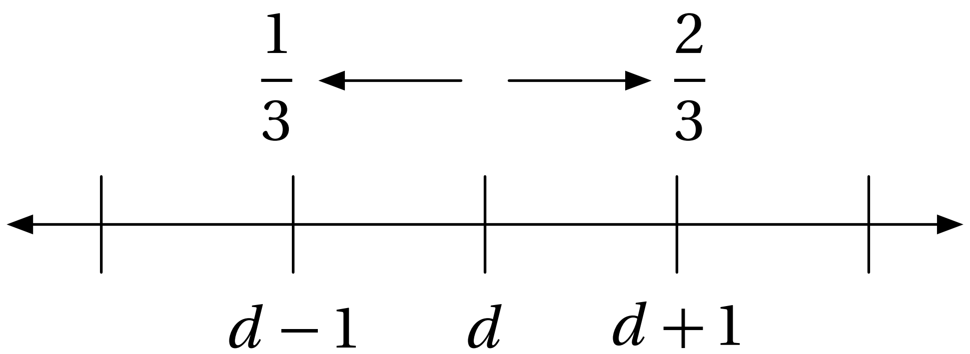
$$\Pr[\Delta d = -1] = \frac{1}{k}, \quad \Pr[\Delta d = +1] = \frac{k-1}{k}$$

For 2-SAT (k=2) this walk is balanced, we succeed in O(n^2) time

For 3-SAT, it is biased away from the solution

Can we reach the shore?

Start at distance *d*, and take a biased random walk:



Exercise: the probability that we will ever reach *d*=0 is

$$P(d) = 2^{-d}$$

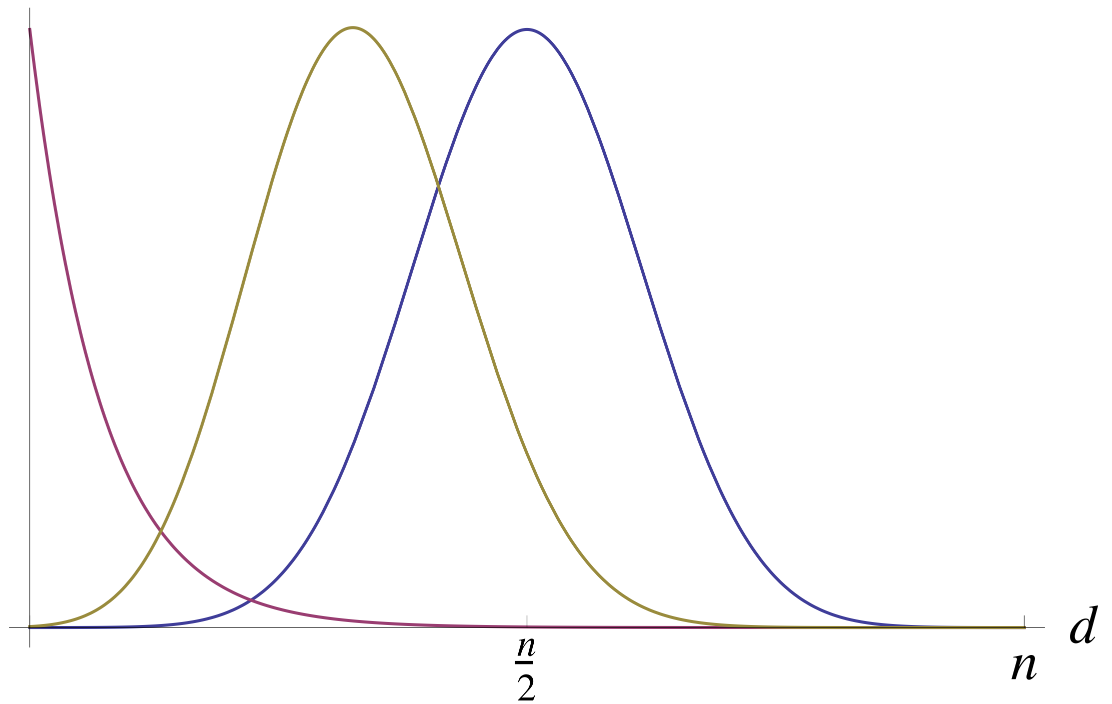
How often do we succeed?

Our initial assignment *B* is random, so on average *d*=*n*/2

Does this mean that $\overline{P(d)} = P(n/2) = 2^{-n/2}$?

No! Unless *f* is linear, $\overline{f(x)} \neq f(\overline{x})$

Rare events where *d* < *n*/2 contribute more to our success, combining two kinds of luck



Computing the weighted average

$$P_{\text{success}} = \sum_{d=0}^{n} \Pr[\text{initial distance is } d]\, P(d)$$

$$= 2^{-n} \sum_{d=0}^{n} \binom{n}{d} 2^{-d}$$

$$= \left(\frac{3}{4}\right)^n .$$

So on average, we need to make $1/P_{\text{success}} = (4/3)^n$ attempts

Each attempt only takes O(*n*) steps before choosing a new *B*

Random restarts: better to start in a new place than to keep looking

Exponential, but much better than 2^n — and almost the best known!

Derandomization

Can every randomized algorithm be derandomized?

Are there pseudorandom generators that look random to any algorithm? (not just statistical tests!)

Yes — if certain questions are hard!

One-way functions: f is in P, but f^{-1} is not

Cryptography: easy to encrypt, hard to decrypt

Pseudorandom generators “encrypt” a random seed, turning a few truly random bits into many pseudorandom ones

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?

How much does randomness help? Can we foil the adversary by being unpredictable?

How much does quantum physics help?

Shameless Plug

This book rocks! You somehow manage to combine the fun of a popular book with the intellectual heft of a textbook.

— Scott Aaronson

A treasure trove of information on algorithms and complexity, presented in the most delightful way.

— Vijay Vazirani

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

Oxford University Press, 2011