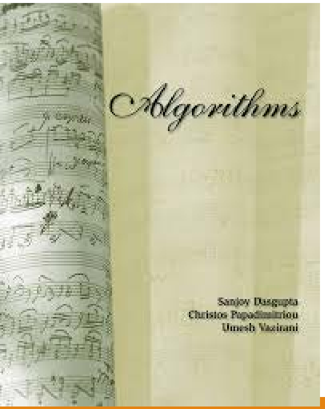
A Fine-Grained Approach to Algorithms and Complexity

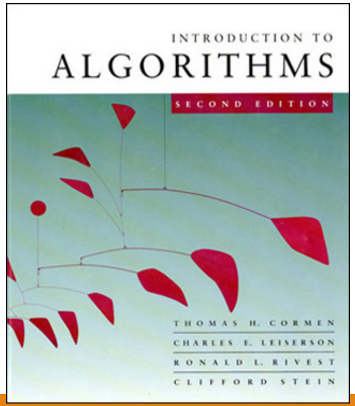
VIRGINIA VASSILEVSKA WILLIAMS

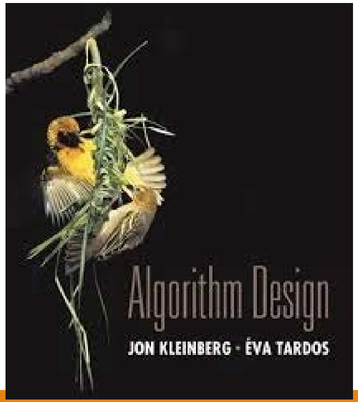
MIT

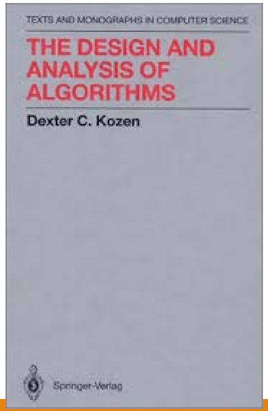
Central Question of Algorithms Research

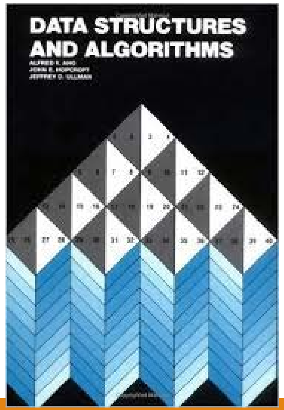
``How fast can we solve fundamental problems, in the worst case?''

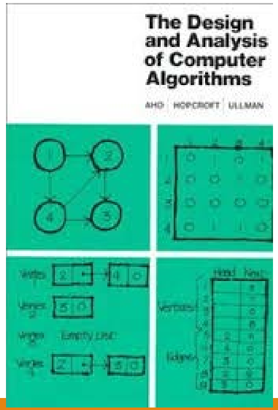












etc.

``How fast can we solve fundamental problems, in the worst case?''

$f: \{0,1\}^* \mapsto \{0,1\}^*$

f computable, search space of solutions

Reasonable model of computation M (e.g. Turing machine, RAM etc)

Defines basic constant time operations

$n \in \mathbb{Z}^+$

$x \in \{0,1\}^n$ → An **algorithm** implemented on M → $f(x)$

We study the best *worst case asymptotic running time* over all algorithms computing f, for various interesting f.

``How fast can we solve fundamental problems, in the worst case?''

Ideally, for each problem P, we want an $O(n)$ time algorithm that solves P on all instances of size n.

(Asymptotically optimal: Need to at least read the input!)

Good news: Huge Toolbox of Algorithmic Techniques!

Dynamic Programming, Divide-and-Conquer, Linear Programming, Convex Optimization, Semidefinite Programming, Fast Matrix Multiplication, and many more.

Many fundamental problems can be solved very fast.

Beating Exhaustive Search

Exhaustive Search: try all possible solutions

In a 1956 letter, Gödel wrote to Von Neumann:

"It would be interesting to know … how strongly in general the number of steps in finite combinatorial problems can be reduced with respect to simple exhaustive search."

For many fundamental problems, there are no known significant improvements over the brute-force solution!

Hard problems

Problems for which all known techniques get stuck:

- Very important problems from diverse areas
- Simple, often exhaustive search, textbook algorithms
- that are slow.
- No significant improvements in many decades!

Example 1

A canonical hard problem: Satisfiability

k-SAT

Input: variables $x_1, \ldots, x_n$ and a formula

$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ so that each C_i is of the form

$\{y_1 \vee y_2 \vee \ldots \vee y_k\}$ and $\forall i$, y_i is either x_t or $\neg x_t$ for some t.

Output: **YES** if there is a Boolean assignment to $\{x_1,\ldots,x_n\}$ that satisfies all the clauses, or **NO** if the formula is not satisfiable

No $O((2^n)^{1-\epsilon})$ time alg. known for $\epsilon > 0$.

Exhaustive Search algorithm: try all 2^n assignments: $O(2^n mn)$

Best known algorithm: $O\left(2^{n - \frac{cn}{k}} m^d\right)$ time for const c,d

Goes to 2^n as k grows.

Example 2

Another Hard problem: Longest Common Subsequence (LCS)

Given two strings on n letters

ATCGGGTTCCTTAAGGG

ATATTGGTACCTTCAGGG

Find a subsequence of both strings of maximum length.

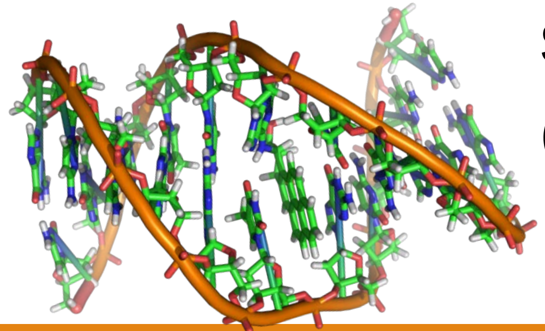
No $O((n^2)^{1-\epsilon})$ time alg. known for $\epsilon > 0$.

Algorithms:

Textbook $\boldsymbol{O(n^2)}$ time

Best algorithm: $O(n^2 / \log^2 n)$ time [MP'80]

Applications both in computational biology and in spellcheckers.



Solved daily on huge strings!

(Human genome: 3×10^9 base pairs.)

Hard problems

Why are we stuck?

Problems for which all known techniques get stuck:

Are we stuck because of the same reason?

- Very important problems from diverse areas
- Simple, often exhaustive search, textbook algorithms
- that are slow: run in $T(n) \gg n$ time.
- No $O(T(n)^{1-\epsilon})$ time alg. for $\epsilon > 0$ found in many decades!

PLAN

Traditional hardness in complexity

A fine-grained approach

Conclusion

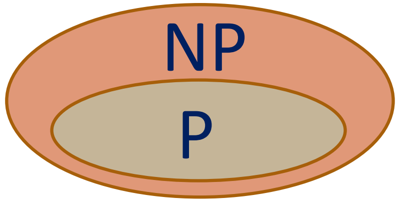
Time hierarchy theorems

For most natural computational models one can prove:

THM: **For any constant integer $c > 1$, there *exist* problems solvable in $O(n^c)$ time but not in $O(n^{c-\epsilon})$ time for any $\epsilon > 0$.**

It is completely unclear how to show that a *particular* problem is not in $O(n^{c-\epsilon})$ time for any $\epsilon > 0$.

It is not even known if k-SAT is in linear time!



N – size of input

Why is k-SAT considered hard?

Notoriously hard problem, one of the Clay Math Institute Millennium Problems:

Is P = NP?

P ≠ NP widely believed.

Theorem [Cook, Karp'72]:

k-SAT is **NP-hard** for all $k \geq 3$.

Theorem says: If k-SAT has a poly time algorithm for some $k \geq 3$, then P = NP.

Conditional Hardness!

NP-Hardness of a problem Q

1. Assume that P ≠ NP
2. Show that if Q has a polynomial time algorithm, then it can be used to solve any problem in NP in polynomial time.
3. Conclude that Q requires super-polynomial time.

No problem that can already be solved in polynomial time can be NP-hard unless P=NP.

We want problems in $O(n^2)$ time to be hard.

NP-Hardness is too coarse-grained.

In theoretical CS, polynomial time = efficient.

This is for a variety of reasons.

E.g. composing two efficient algorithms results in an efficient algorithm. Also, model-independence.

However, noone would consider an $O(n^{100})$ time algorithm efficient in practice.

If n is huge, then $O(n^2)$ is also inefficient.

We develop a *fine-grained theory of hardness* that mimics NP-hardness but is about concrete runtimes.

PLAN

Traditional hardness in complexity

A fine-grained approach

Conclusion

NP-HARDNESS

1. (Hardness Hypothesis)
   Assume that P ≠ NP
   SAT requires super-polynomial time

2. (Polynomial Time Reduction)
   Show that if Q has a polynomial time algorithm, then it can be used to solve SAT / any problem in NP in polynomial time.

3. (Conditional Hardness)
   Conclude that Q must require super-polynomial time.

FINE-GRAINED HARDNESS

1. (Hardness Hypotheses) such as hard problem H requires $h(n)^{1-o(1)}$ time on inputs of size n on a RAM

2. (Fine-Grained Reduction)
   Show that an $O(q(n)^{1-\epsilon})$ time RAM algorithm for problem Q for $\epsilon > 0$ would imply an $O(h(n)^{1-\delta})$ time RAM alg for $\delta > 0$ for problem H.

3. (Conditional Hardness) Conclude that Q must require $q(n)^{1-o(1)}$ time on a RAM.

SAT is conjectured to be really hard

Much Stronger than P $\neq$ NP!!!

Two popular conjectures about SAT on n variables [IPZ01]:

Exponential Time Hypothesis (ETH):

There exists a constant $\delta > 0$ s.t. 3-SAT cannot be solved in $O(2^{\delta n})$ time.

Strong Exponential Time Hypothesis (SETH): $\forall \epsilon > 0 \; \exists k \geq 3$ s.t. k-SAT on n variables, m clauses cannot be solved in $2^{n(1-\epsilon)} poly(m)$ time.

We can use ETH or SETH as our hardness hypothesis.

Strengthening of SETH [CGIMPS'16] suggests these are **not equivalent**...

Fix the model: word-RAM with $O(\log n)$ bit words

Given a set S of n vectors in $\{0,1\}^d$, for $d = \omega(\log n)$ are there $u, v \in S$ with $\mathbf{u} \cdot \mathbf{v} = \mathbf{0}$?

Hypothesis: Orthog. Vecs. requires $n^{2-o(1)}$ time.

[W'05]: SETH implies this hypothesis!

Easy $O(\mathbf{n^2}\, d)$ time alg
Best known [AWY'15]: $\mathbf{n^{2-\Theta(1/\log(d/\log n))}}$

Orthogonal vectors

Hypothesis: APSP requires $n^{3-o(1)}$ time.

More key problems to blame

All pairs shortest paths: given an n-node weighted graph, find the **distance** between every two nodes.

APSP

Given a set S of n numbers, are there $a, b, c \in S$ with $\boldsymbol{a+b+c=0}$?

3SUM

Easy $O(n^2)$ time alg
[BDP'05]: ~$\mathbf{n^2}$ / $\log^2 n$ time for integers
[Chan'18] : ~$\mathbf{n^2}$ / $\log^2 n$ time for reals

Hypothesis: 3SUM requires $n^{2-o(1)}$ time.

Classical algs: $O(\mathbf{n^3})$ time
[W'14]: $\mathbf{n^3} / \exp(\sqrt{\log n})$ time

NP-HARDNESS

1. (Hardness Hypothesis)
   Assume that P ≠ NP
   SAT requires super-polynomial time

2. (Polynomial Time Reduction)
   Show that if Q has a polynomial time algorithm, then it can be used to solve SAT / any problem in NP in polynomial time.

3. (Conditional Hardness)
   Conclude that Q must require super-polynomial time.

FINE-GRAINED HARDNESS

1. (Hardness Hypotheses)
   problem H requires $h(n)^{1-o(1)}$ time on inputs of size n on a RAM

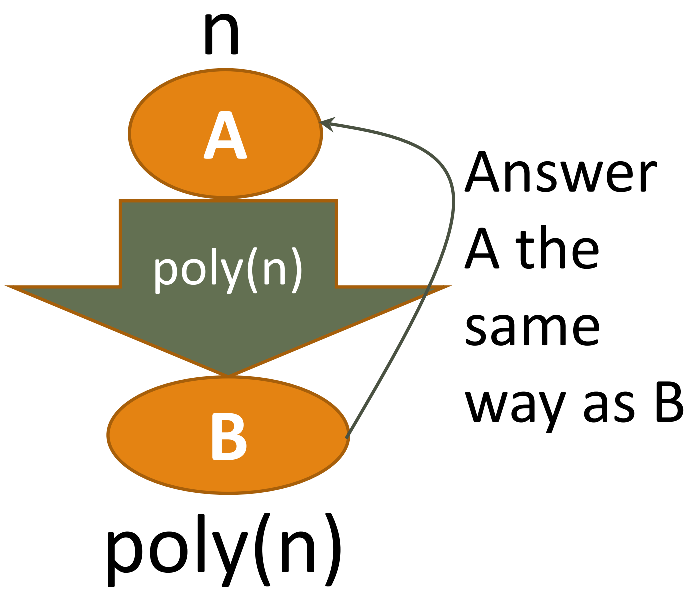
2. (Fine-Grained Reduction)
   Show that an $O(q(n)^{1-\epsilon})$ time RAM algorithm for problem Q for $\epsilon > 0$ would imply an $O(h(n)^{1-\delta})$ time RAM alg for $\delta > 0$ for problem H.

3. (Conditional Hardness)
   Conclude that Q must require $q(n)^{1-o(1)}$ time on a RAM.

Polynomial Time Reductions

Def. (Many-One Reduction) Let A and B be decision problems ($\{0,1\}^* \mapsto \{0,1\}$).

A is *polynomial time many-one reducible* to B if there is a polynomial time algorithm R that transforms any instance x of A into an instance $R(x)$ of B, so that $A(x) = 1$ if and only if $B(R(x)) = 1$.



Many-one reductions have useful properties

They also have weaknesses:

- Cannot show that a function problem can be reduced to a decision problem
- Coarse-grained: not about concrete runtimes

Polynomial Time Reductions

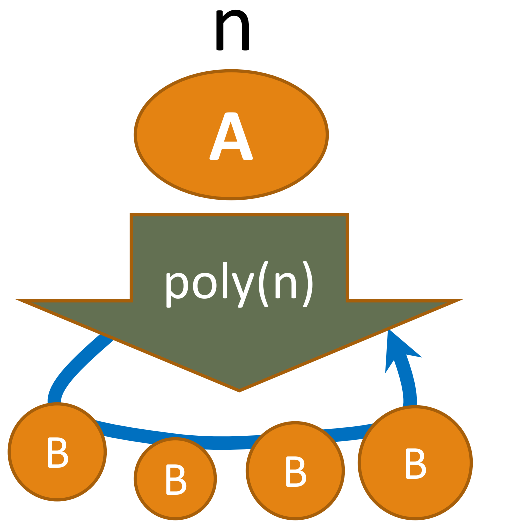
Def. (Turing Reduction) Let A and B be computational problems.

A is *polynomial time Turing reducible* to B if there is a polynomial time algorithm that solves any instance x of A using oracle access to a polynomial number of polynomial-sized instances of B.

Turing reductions have the same useful properties, and also allow a search problem to be reduced to a decision problem.

Weaknesses:

- Cannot differentiate between NP- and coNP-completeness
- Coarse-grained: not about concrete runtimes

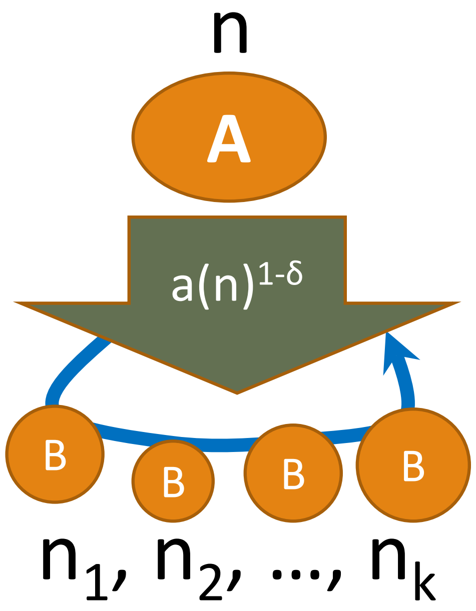


Fine-grained reductions (V-Williams'10)

Intuition: a(n),b(n) are the naive runtimes for A and B. A reducible to B implies that beating the naive runtime for B implies also beating the naive runtime for A.

Def. Let A and B be problems, and $a\colon \mathbb{N} \mapsto \mathbb{N}$ and b: $\mathbb{N} \mapsto \mathbb{N}$ be time-constructible.

A is (a,b)-reducible to B if $\forall \epsilon > 0\ \exists \delta > 0$, and an $O\left(a(n)^{1-\delta}\right)$ time algorithm that can solve A on instances of size n making calls to an oracle for B with query lengths $n_1, \dots, n_k$ so that $\sum_i b(n_i)^{1-\epsilon} < a(n)^{1-\delta}$.



Prop. 1: If A is (a,b)-reducible to B and B is in $O\left(b(n)^{1-\epsilon}\right)$ time, then A is in $O\left(a(n)^{1-\delta}\right)$ time.

Prop. 2: If A is (a,b)-reducible to B and B is (b,c)-reducible to C, then A is (a,c)-reducible to C.

NP-HARDNESS

1. (Hardness Hypothesis)
   Assume that P ≠ NP
   SAT requires super-polynomial time

2. (Polynomial Time Reduction)
   Show that if Q has a polynomial time algorithm, then it can be used to solve SAT / any problem in NP in polynomial time.

3. (Conditional Hardness)
   Conclude that Q must require super-polynomial time.

FINE-GRAINED HARDNESS

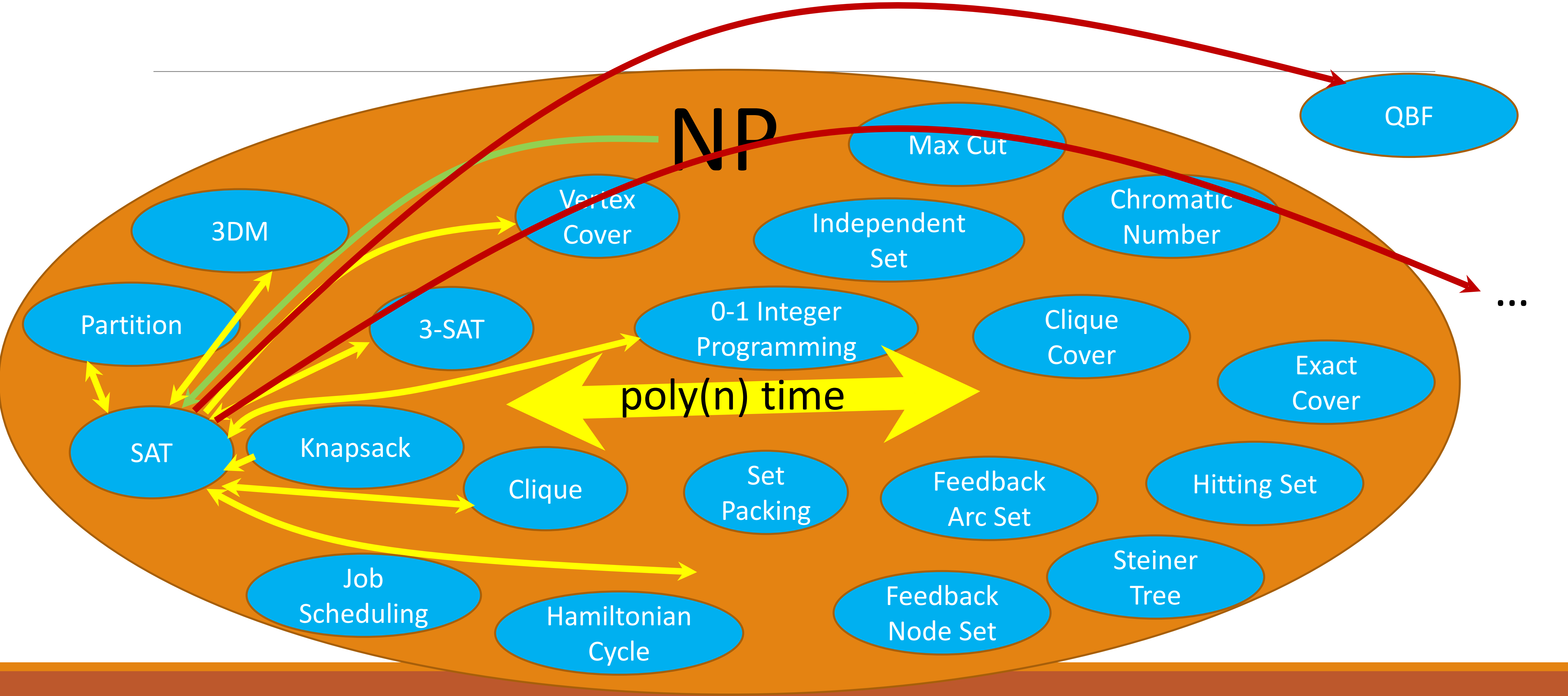
1. (Hardness Hypotheses)
   problem H requires $h(n)^{1-o(1)}$ time on inputs of size n on a RAM

2. (Fine-Grained Reduction)
   Show that an $O(q(n)^{1-\epsilon})$ time RAM algorithm for problem Q for $\epsilon > 0$ would imply an $O(h(n)^{1-\delta})$ time RAM alg for $\delta > 0$ for problem H.

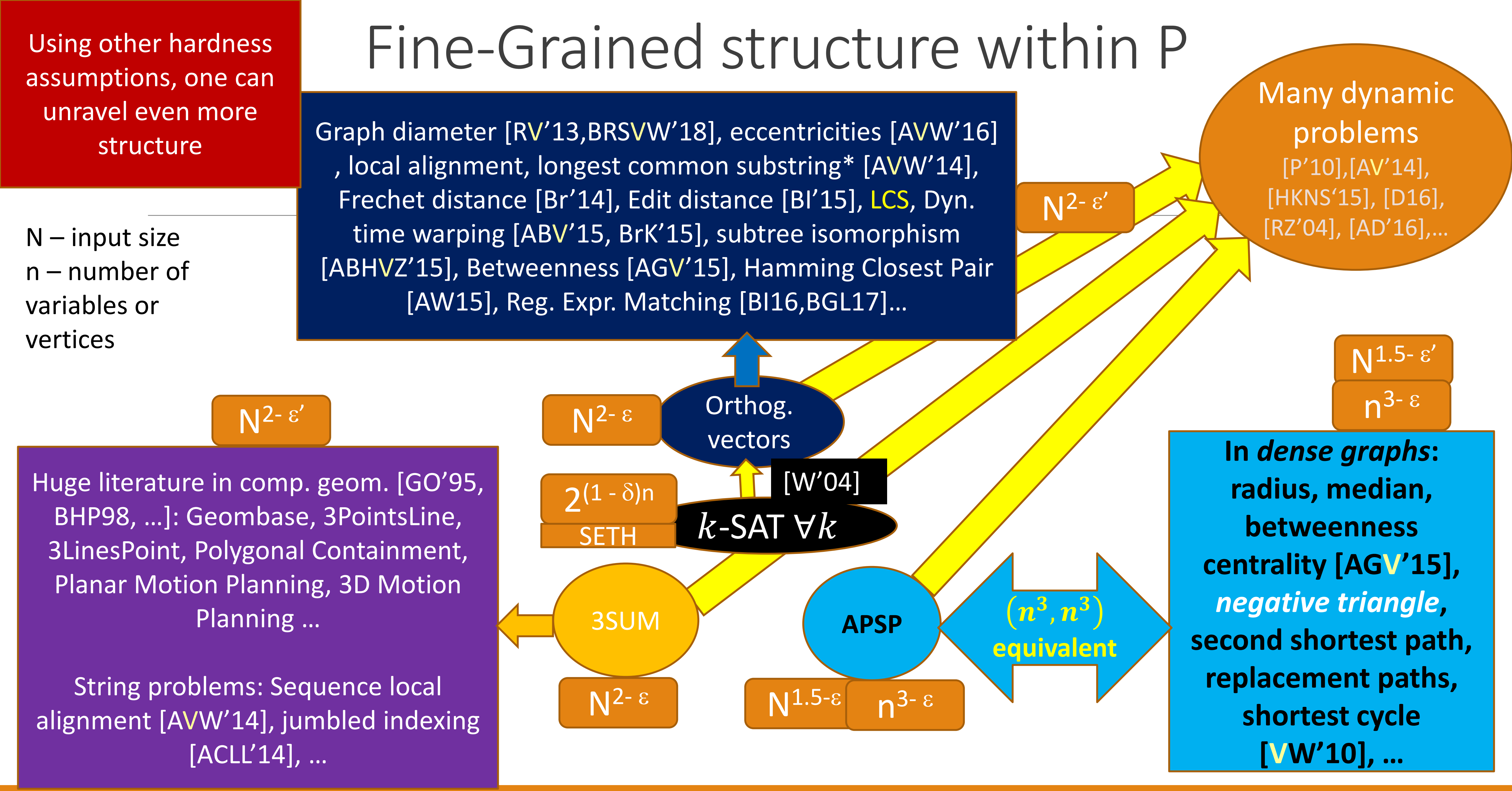
3. (Conditional Hardness)
   Conclude that Q must require $q(n)^{1-o(1)}$ time on a RAM.

1972: Karp's 21

Thousands of NP-Complete problems found since!



Fine-Grained structure within P



PLAN

Traditional hardness in complexity

A fine-grained approach

Conclusion

Fine-grained ++

Since 2010 there has been an explosion of results on fine-grained complexity: *adding many problems to the picture, strengthening the hardness hypotheses, giving connections to circuit lower bounds and more.*

Fine-grained complexity has spread: *fine-grained space complexity, approximability, I/O complexity, data structures, fine-grained cryptography …*

Recent fine-grained ideas in cryptography

Ball, Rosen, Sabin andVasudevan'17,18:

Can we build cryptographic primitives using worst case fine-grained assumptions?

YES: Assuming OV, 3SUM, APSP Hypotheses there exist related algebraic average-case hard problems, and one can get *proofs of work* based on them.

Challenge generation in ~n time, Proof in ~n^k time, Proof Validity in ~n time, Every proof needs ~n^k time.

BUT: there are barriers to getting one-way-functions…

Are the hard problems in fine-grained complexity hard on average themselves for a nice distribution?

OV is not $n^{2-o(1)}$ hard: [Kane, Williams'18] for all $p \in (0,1)$, $\exists \epsilon_p > 0$ s.t. all but $o_n(1)$ fraction of the OV instances where bits are set to 1 w.p. p can be solved in $O(n^{2-\epsilon_p})$ time.

APSP is not $n^{3-o(1)}$ hard: for various natural distributions… [Peres et al.'13] e.g. when edge weights are drawn uniformly at random from $[0,1]$, APSP is in $O(n^2)$ time.

CNF-SAT? 3SUM? Might be hard…

[Goldreich and Rothblum'18]: Counting versions of the problems might be hard. Reduce counting **k-cliques** to counting ***k-cliques on average (from a simple distribution)***. Implies proof of work.

Conclusion

Thank you!

Since 2010 there has been an explosion of results on fine-grained complexity: *adding many problems to the picture, strengthening the hardness hypotheses, giving connections to circuit lower bounds and more.*

Fine-grained complexity has spread: *fine-grained space complexity, approximability, I/O complexity, data structures, fine-grained cryptography … what's next?*