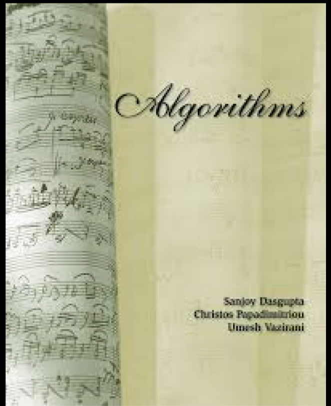
Fine-Grained Complexity

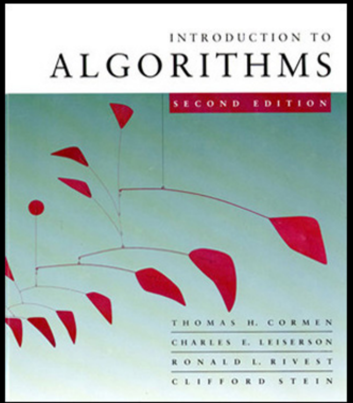
(a.k.a. "Hardness in P")

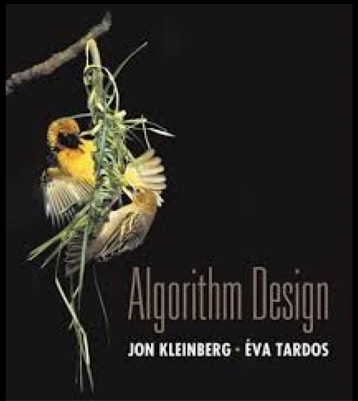
Ryan Williams
(with Virginia Vassilevska Williams' slides!)

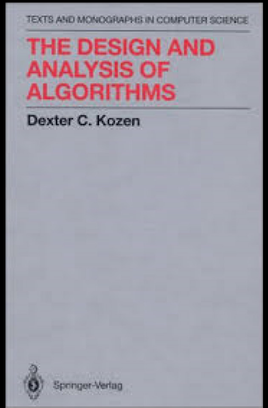
The Central Question of Algorithms Research

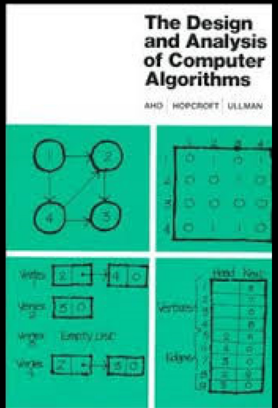
“How fast can we solve fundamental problems, in the worst case?”











etc.

HARD PROBLEMS

For many problems, the known techniques get stuck:

- Very important computational problems from diverse areas
- They have simple, often brute-force, textbook algorithms…

… that are ***too slow***.

- No improvements in many decades!

These points hold not only for NP-hard problems,
but ***polynomial-time*** problems as well!

Example 1

A Canonical (NP) Hard Problem

k-SAT

Input: variables $x_1, \ldots, x_n$ and a formula $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ over $\{x_1,\ldots,x_n\}$
where each C_i has the form $\{y_1 \vee y_2 \vee \ldots \vee y_k\}$ and each y_i is either x_t or $\neg x_t$ for some t.

Output: A boolean assignment to $\{x_1,\ldots,x_n\}$ that makes all clauses true (satisfies clauses),
or NO if the formula is not satisfiable (there is no such assignment)

Brute-force algorithm: try all 2^n assignments, plug them in one by one

Best known algorithm: $O(2^{n-(cn/k)} m^d)$ time for fixed constant 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

ATTGGTACCTTCAGG

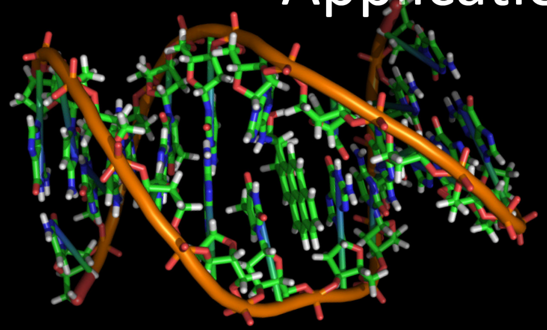
Find a subsequence of both strings of maximum length.

Algorithms:

Classical **$O(n^2)$** time

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

Applications: computational biology, spellcheckers, …



Solved daily on ***huge*** strings!

(Human genome: 3×10^9 base pairs.)

In Theoretical Computer Science, Polynomial time = efficient/easy.

- **Composition:** Composing two "efficient" algorithms always results in another "efficient" algorithm
- **Model Independence:** "Polynomial time" is the same notion over random access machines, pointer machines, Turing machines, etc.

However, nobody believes that an $O(n^{100})$ time algorithm would be efficient in practice…

If n is large enough, then $O(n^2)$ is already inefficient!

We are stuck on many problems, even O(n²)-time solvable ones!

We do not know any $N^{2-\varepsilon}$ time algorithm (for *any* $\varepsilon > 0$) for:

➨ Many *string matching* problems:
Edit distance, Sequence local alignment, LCS, jumbled indexing …

General form*: given two sequences of length n, how similar are they?*
All variants can be solved in $O(n^2)$ time by dynamic programming.

ATCGGGTTCCTTAAGGG
ATTGGTACCTTCAGG

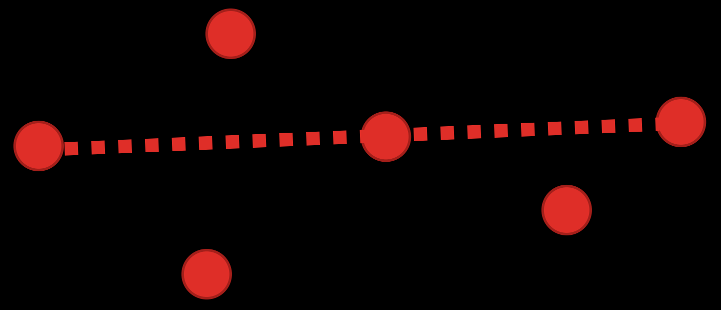
We are stuck on many problems, even $O(N^2)$-time solvable ones!

We do not know any $\mathbf{N^{2-\varepsilon}}$ time algorithm (for *any* $\varepsilon > 0$) for:

- Many *string matching* problems
- Many problems in *computational geometry*: for example,

Given n points in the plane, are any three co-linear?

A very important primitive!



We are stuck on many problems, even $O(n^2)$-time solvable ones!

We do not know any $N^{2-\varepsilon}$ time algorithm (for *any* $\varepsilon > 0$) for:

- ➨ Many *string matching* problems
- ➨ Many problems in *computational geometry*
- ➨ Many *graph problems* in sparse graphs, for example:

Given an n-node, $O(n)$-edge graph, what is its diameter?

Fundamental problem. Even approximation algorithms seem hard!

We are stuck on many problems, even O(N^2)-time solvable ones!

We do not know any $N^{2-\varepsilon}$ time algorithm (for *any* $\varepsilon > 0$) for:

- Many *string matching* problems
- Many problems in *computational geometry*
- Many *graph problems* in sparse graphs
- Many other problems …

Why are we stuck?

Are we stuck because of ***the same reason***?

"HARD" PROBLEMS IN THE REAL WORLD

I've got data. I want to solve this algorithmic problem but I'm stuck!

I'm sorry, your problem is **NP-hard**. A fast algorithm would resolve a big problem in CS/math and tilt the very axis of our universe.

Uhm… Ok, thanks, I feel better that nothing worked… I'll use some heuristics.

"EASY" PROBLEMS IN THE REAL WORLD

I've got data. I want to solve this algorithmic problem but I'm stuck!

Great news! Your problem is in **P**. Here's an **O(n²)** time algorithm!

Yo, my **n** is huge! Don't you have a faster algorithm?

Uhm, I don't know… Isn't this fast enough? I don't know a faster algorithm at the moment…

?!? … Should I wait? … should I be satisfied with heuristics?

Outline

- Traditional hardness in computational complexity theory
- A “fine-grained” approach to complexity theory
- Some simple results

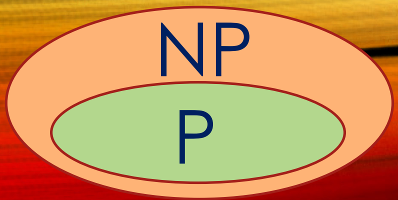
A Source of Hardness: Time Hierarchy Theorems

For most natural computational models, one can prove:

for any constant $c \geq 1$, and every $\varepsilon > 0$, there *exist* problems that are solvable in $O(n^c)$ time but not in $O(n^{c-\varepsilon})$ time.

It remains entirely unclear how to show that a *particular desired* problem in **$O(n^c)$ time** is not in $O(n^{c-\varepsilon})$ **time**.

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



Why is K-SAT Hard?

Theorem [Cook, Levin, Karp]:

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

NP-completeness addresses runtime, but it is too coarse-grained!

NP-completeness also does not apply to problems in P! Unless P=NP

Tool: poly-time reductions

That is, k-SAT is believed to be *hard because*

***poly-time algorithms for k-SAT imply poly-time algorithms for many other difficult problems*.**

A *fine-grained theory of hardness* has been developed, which is conditional and mimics NP-completeness.

Outline

- Traditional hardness in computational complexity theory
- A “fine-grained” approach to complexity theory
- Some simple results

Fine-Grained Hardness

Goal: Understand the landscape of problems in P that people want to solve

0. Mimic NP-completeness?

1. Identify **key hard problems**

2. **Reduce** these to all (?) problems believed hard

3. Try to form ***equivalence classes of problems:***

   one of them can be solved faster

   ⇔ all of them can be solved faster

Fine-Grained Hardness

Goal: Understand the landscape of problems in P that people want to solve

0. Mimic NP-completeness?

1. Identify **key hard problems**

2. **Reduce** these to all (?) problems believed hard

3. Try to form ***equivalence classes of problems:***
   one of them can be solved faster
   ⇔ all of them can be solved faster

CNF SAT IS CONJECTURED TO BE *REALLY* HARD

Two popular conjectures about SAT on n variables [IPZ'99,CIP'09]

ETH (Exponential Time Hypothesis):

3-SAT cannot be solved in $2^{\delta n}$ time for some constant $\delta > 0$.

3-SAT can't be solved in $1.0000\cdots01^n$ time (for *some* number of 0's)

SETH (Strong Exponential Time Hypothesis):

For every $\varepsilon > 0$, there is a k such that k-SAT on n variables and m clauses cannot be solved in $2^{(1-\varepsilon)n}$ poly(m) time.

CNF-SAT can't be solved in $1.9999\cdots9^n$ time (for *every* number of 9's)

One Idea: Use k-SAT as our hard problem, and ETH or SETH as the hypothesis that we base hardness on.

Strengthening of SETH [CGIMPS'16] suggests these three 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 $\langle \mathbf{u},\mathbf{v} \rangle = \mathbf{0}$?

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

[W'05]: SETH implies this hypothesis!

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

Orthogonal vectors (OV)

Three more key problems to blame

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

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

3SUM

APSP

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

Not-too-hard $O(n^2)$ time algorithm
[BDP'05]: $\approx \mathbf{n^2} / \log^2 n$ time for integers
[Chan'18] : $\approx \mathbf{n^2} / \log^2 n$ 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

Fine-Grained Hardness

0. Mimic NP-completeness?

1. Identify **key hard problems**

2. **Reduce** these to all (?) problems believed hard

3. Hopefully form ***equivalence classes of problems:***
one of them can be solved faster
⇔ all of them can be solved faster

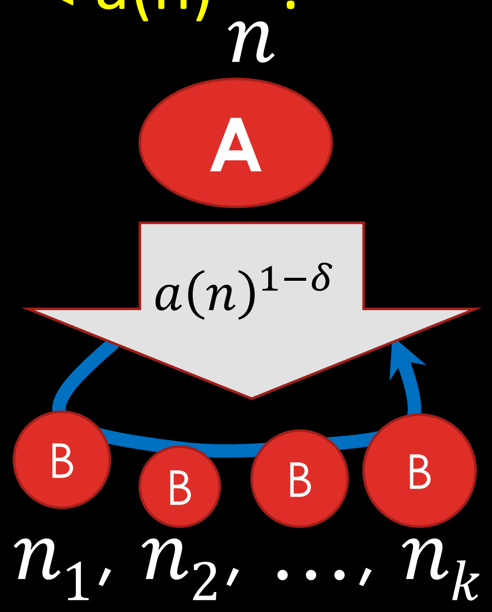
FINE-GRAINED REDUCTIONS

Intuition: a(n),b(n) are known runtimes for problems A and B. "A reducible to B" implies that beating b(n)-time for B implies also beating a(n)-time for A.

- Problem **A** is (a(n),b(n))-reducible to Problem **B** if

  for all sufficiently small $\varepsilon > 0$, there's a $\delta > 0$ and an **$O(a(n)^{1-\delta})$ time** algorithm

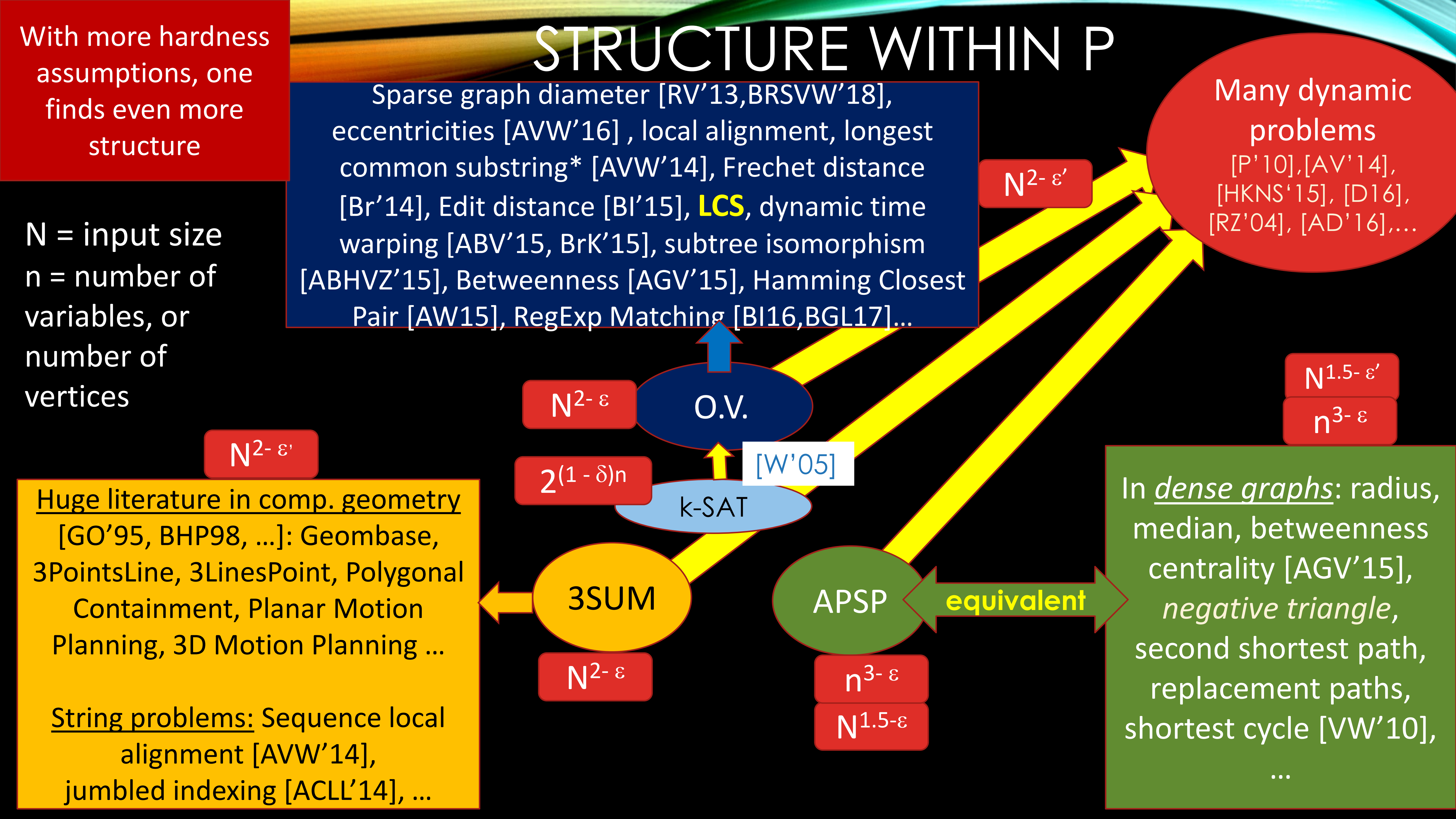
  that can solve all **A-instances of size n** by making adaptive calls solving **B-instances of size $n_1,\ldots,n_k$** satisfying $\Sigma_i\, b(n_i)^{1-\varepsilon} < a(n)^{1-\delta}$.



Key Property: If B is in $O(b(n)^{1-\varepsilon})$ **time for *some* ε, then A is in** $O(a(n)^{1-\delta})$ time for *some* δ.

- Focus on running time exponents.
- We can build more equivalences with this.

Most reductions don't need this level of generality… but some do!



STRUCTURE WITHIN P
With more hardness assumptions, one finds even more structure
N = input size
n = number of variables, or number of vertices
Sparse graph diameter [RV’13,BRSVW’18], eccentricities [AVW’16] , local alignment, longest common substring* [AVW’14], Frechet distance [Br’14], Edit distance [BI’15], LCS, dynamic time warping [ABV’15, BrK’15], subtree isomorphism [ABHVZ’15], Betweenness [AGV’15], Hamming Closest Pair [AW15], RegExp Matching [BI16,BGL17]…
Many dynamic problems
[P’10],[AV’14], [HKNS‘15], [D16], [RZ’04], [AD’16],…
$N^{2-\varepsilon'}$
$N^{2-\varepsilon}$
O.V.
[W’05]
$2^{(1-\delta)n}$
k-SAT
$N^{2-\varepsilon'}$
Huge literature in comp. geometry [GO’95, BHP98, …]: Geombase, 3PointsLine, 3LinesPoint, Polygonal Containment, Planar Motion Planning, 3D Motion Planning …
String problems: Sequence local alignment [AVW’14], jumbled indexing [ACLL’14], …
3SUM
$N^{2-\varepsilon}$
APSP
equivalent
$n^{3-\varepsilon}$
$N^{1.5-\varepsilon}$
$N^{1.5-\varepsilon'}$
$n^{3-\varepsilon}$
In dense graphs: radius, median, betweenness centrality [AGV’15], negative triangle, second shortest path, replacement paths, shortest cycle [VW’10], …

Outline

- Traditional hardness in computational complexity theory
- A "fine-grained" approach to complexity theory
- Some simple results:
  Show that SETH implies fine-grained hardness in P

Strong ETH (SETH)

SETH: for every $\varepsilon > 0$, there is a k such that k-SAT on n variables, m clauses cannot be solved in $2^{(1-\varepsilon)n}$ poly(m) time.

If there is an $\varepsilon > 0$ and an algorithm that can solve SAT on ***general*** CNF Formulas (k-SAT for all k) on n variables and m clauses in $2^{(1-\varepsilon)n}$ poly(m) time algorithm, then SETH is false.

FASTER OV IMPLIES SETH IS FALSE [W'04]

Let F be a CNF formula with n vars, m clauses

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

Split the vars into V_1 and V_2 on $n/2$ vars each

Ex: $V_1 = \{x_1, x_2\}$, $V_2 = \{x_3, x_4\}$

OV: Given a set S of N vectors in $\{0, 1\}^d$, are there u, v ∈ S with **<u, v> = 0**?

Given F, we want to create a set of vectors S in $\{0,1\}^d$ so that there is an orthogonal pair in S if and only if F is satisfiable, with $|S| \approx 2^{n/2}$ and $d \approx m$.

Consider all **partial assignments** of V_1 **and V_2: there are $2^{n/2}$ of them.**

Ex: for V_1: $\{ [x_1 = 0, x_2 = 0], [x_1 = 0, x_2 = 1], [x_1 = 1, x_2 = 0], [x_1 = 1, x_2 = 1]\}$

FASTER OV IMPLIES SETH IS FALSE [W'04]

Let F be a CNF formula with n vars, m clauses

Split the vars into V_1 and V_2 on n/2 vars each

For i=1,2 and every **partial assignment** A of V_i,
create an (m+2)-length vector v(j, A):

Ex: $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4)$
$V_1 = \{x_1, x_2\}$, $V_2 = \{x_3, x_4\}$

$v(1,[x_1 = 0, x_2 = 0]) =$
$[0, 1, 1, 0, 1]$

m clauses

0	1	0	1	0	...	...	...	...	1

for all v(**1**, A)

0 if A satisfies the clause, 1 otherwise

1	0	0	0	1	...	...	...	...	1

for all v(**2**, A)

**0 if A satisfies the clause,
1 otherwise**

The 01 and 10 gadgets imply: If there's an orthogonal pair, it must be a red vector and a blue vector

FASTER OV IMPLIES SETH IS FALSE

0	1	0	1	0	…	…	…	…	1

for all v(**1**, A)

0 if A satisfies the clause, 1 otherwise

1	0	0	0	1	…	…	…	…	1

for all v(**2**, A')

0 if A' satisfies the clause,
1 otherwise

Claim: <v(1, A), v(2, A')> = 0 iff (A, A') is a sat assignment to F.

We have an OV instance with N = $2^{n/2}$ vectors of dimension $d = O(m)$

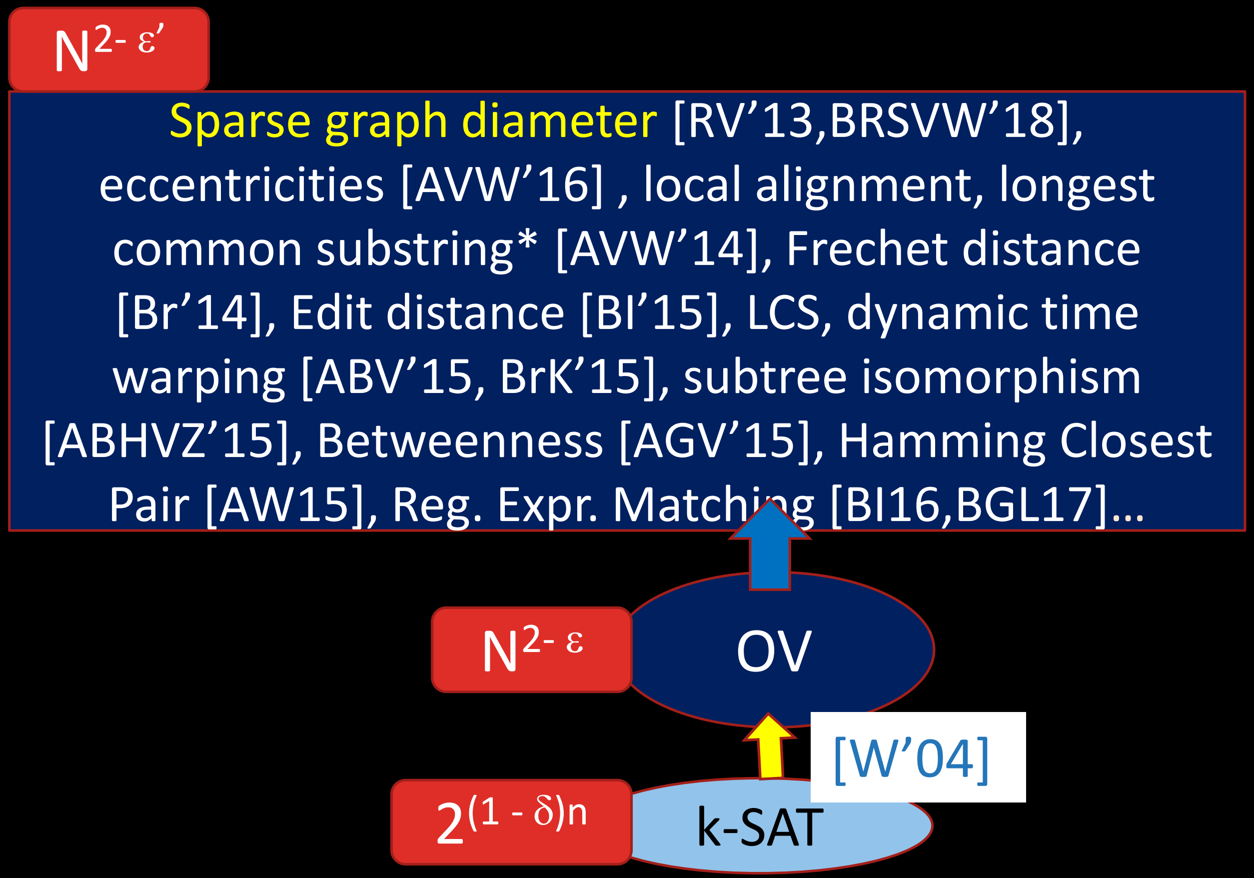
Therefore, if OV can be solved in $N^{2-\delta}\ poly(d)$ time for some $\delta > 0$

then CNF-SAT can be solved in $2^{n(1-\frac{\delta}{2})}\ poly(m)$ time, and SETH is false!

Diameter:

Given $G = (V, E)$, determine $D = \max_{u,v \in V} distance(u, v)$.

$\frac{3}{2}$ − **Approximate Diameter**: output D’ such that $\frac{2D}{3} \le D' \le D$.



Let G have m edges and n vertices.
Using BFS, can solve Diameter in $O(mn)$ time
Best known, even in sparse graphs.

[RV’13] 3/2-Approximate Diameter in $\tilde{O}\left(m^{\frac{3}{2}}\right)$ time: better than mn for sparse graphs!

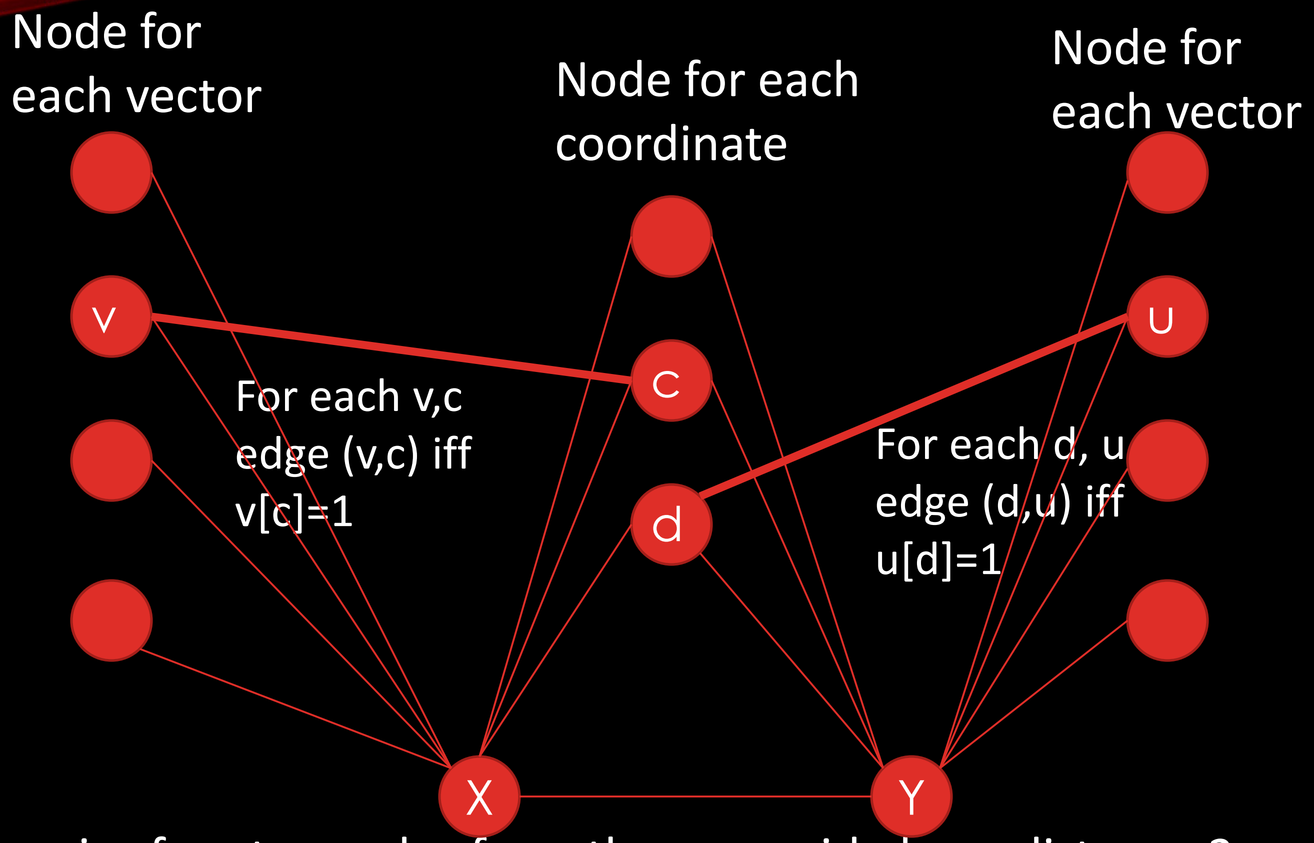
We’ll show $3/2-\epsilon$ Approximate Diameter for $\epsilon > 0$ requires $mn^{1-o(1)}$ time under SETH.

Hard: Distinguishing between sparse graphs of Diameter 2, and those with Diameter 3

Reduce from OV with n vectors and dimension d = $poly(\log n)$

[RV'13]

DIAMETER 2 OR 3



Claim: The diameter of this graph is 3 if there is an orthogonal pair, and is 2 otherwise.

Thm: Determining if a graph has diameter 2 or 3 in $O(m^{2-\varepsilon})$ time implies $O(n^{2-\varepsilon})$ time for OV, so SETH is false!

Every pair of vector nodes from the same side have distance 2.

Every coordinate node is distance 2 from everyone,
X and Y have distance 2 from everyone.

Two vector nodes u and v from different sides have

distance 2 if there's a c with **u[c]=v[c]=1**, and have **distance 3** otherwise!

Graph has $O(n)$ nodes.
Since $d = poly(\log n)$,
has $m = \tilde{O}(n)$ edges

That's All! Thank You!

Lecture Notes for a whole course @ https://people.csail.mit.edu/virgi/6.1420/