A FINE-GRAINED APPROACH TO ALGORITHMS AND COMPLEXITY

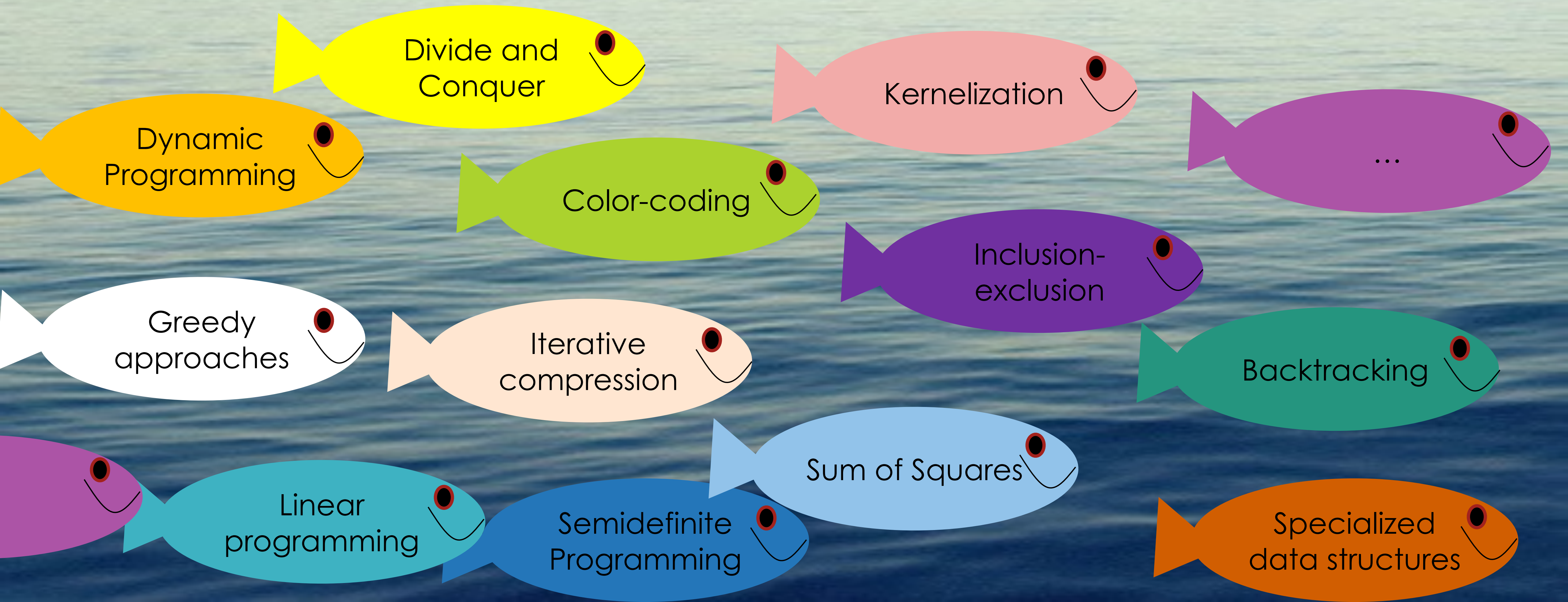
Virginia Vassilevska Williams
Stanford University

THE CENTRAL QUESTION OF ALGORITHMS RESEARCH

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

etc.

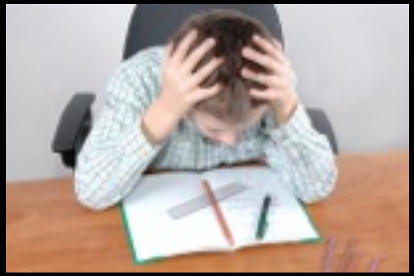
ALGORITHMIC TECHNIQUES



HARD PROBLEMS

For many problems, the known techniques get stuck:

- Very important computational problems from diverse areas
- They have simple, often brute-force, classical algorithms
- No improvements in many decades!



Example 1

A CANONICAL HARD PROBLEM

k-SAT

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

$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ 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: A boolean assignment to $\{x_1,\ldots,x_n\}$ that satisfies all the clauses, or NO if the formula is not satisfiable

Brute-force algorithm: try all 2^n assignments

Best known algorithm: $O(2^{n-(cn/k)} n^d)$ 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

ATTGGTACCTTCAGG

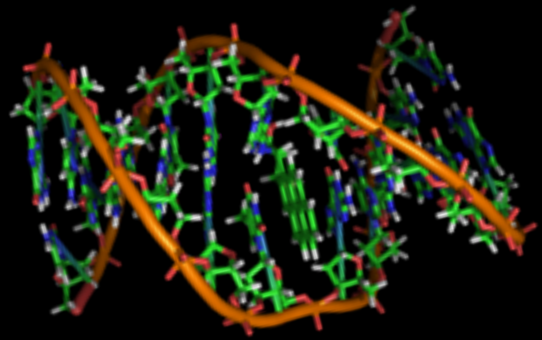
Find a subsequence of both strings of maximum length.

Algorithms:

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

THE REAL WORLD AND NP-HARD PROBLEMS LIKE K-SAT

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

Ok, thanks, I feel better that none of my attempts worked. I'll use some heuristics.

I'm sorry, this problem is NP-hard. A fast algorithm for it would resolve a hard problem in CS/math.

THE REAL WORLD AND EASIER PROBLEMS LIKE LCS

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 $O(n^2)$ time algorithm!

But my data size n is huge! Don't you have a faster algorithm?

Uhm, I don't know… This is already theoretically fast… For some reason I can't come up with a faster algorithm for it right now…

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

IN THEORETICAL CS, POLYNOMIAL TIME = EFFICIENT/EASY.

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 $O(n^2)$ can also be inefficient.

WE ARE STUCK ON MANY PROBLEMS, EVEN JUST IN $O(N^2)$ TIME

No $N^{2-\varepsilon}$ time algorithms known 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 JUST IN $O(N^2)$ TIME

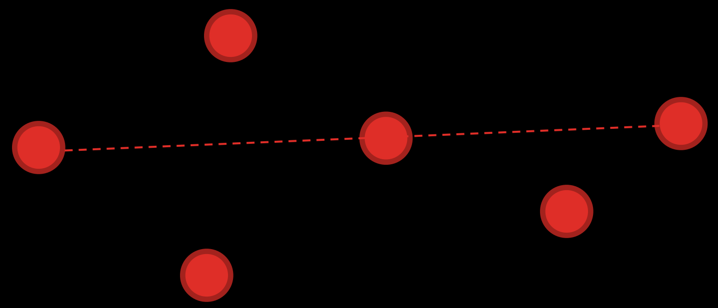
No $N^{2-\varepsilon}$ time algorithms known for:

➨Many *string matching* problems

➨Many problems in *computational geometry*: e.g

Given n points in the plane, are any three colinear?

A very important primitive!



WE ARE STUCK ON MANY PROBLEMS, EVEN JUST IN $O(N^2)$ TIME

No $N^{2-\varepsilon}$ time algorithms known for:

- Many *string matching* problems
- Many problems in *computational geometry*
- Many *graph problems* in sparse graphs: e.g.

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 JUST IN $O(N^2)$ TIME

No $N^{2-\varepsilon}$ time algorithms known for:

➨Many *string matching* problems

➨Many problems in *computational geometry*

➨Many *graph problems* in sparse graphs

➨Many other problems …

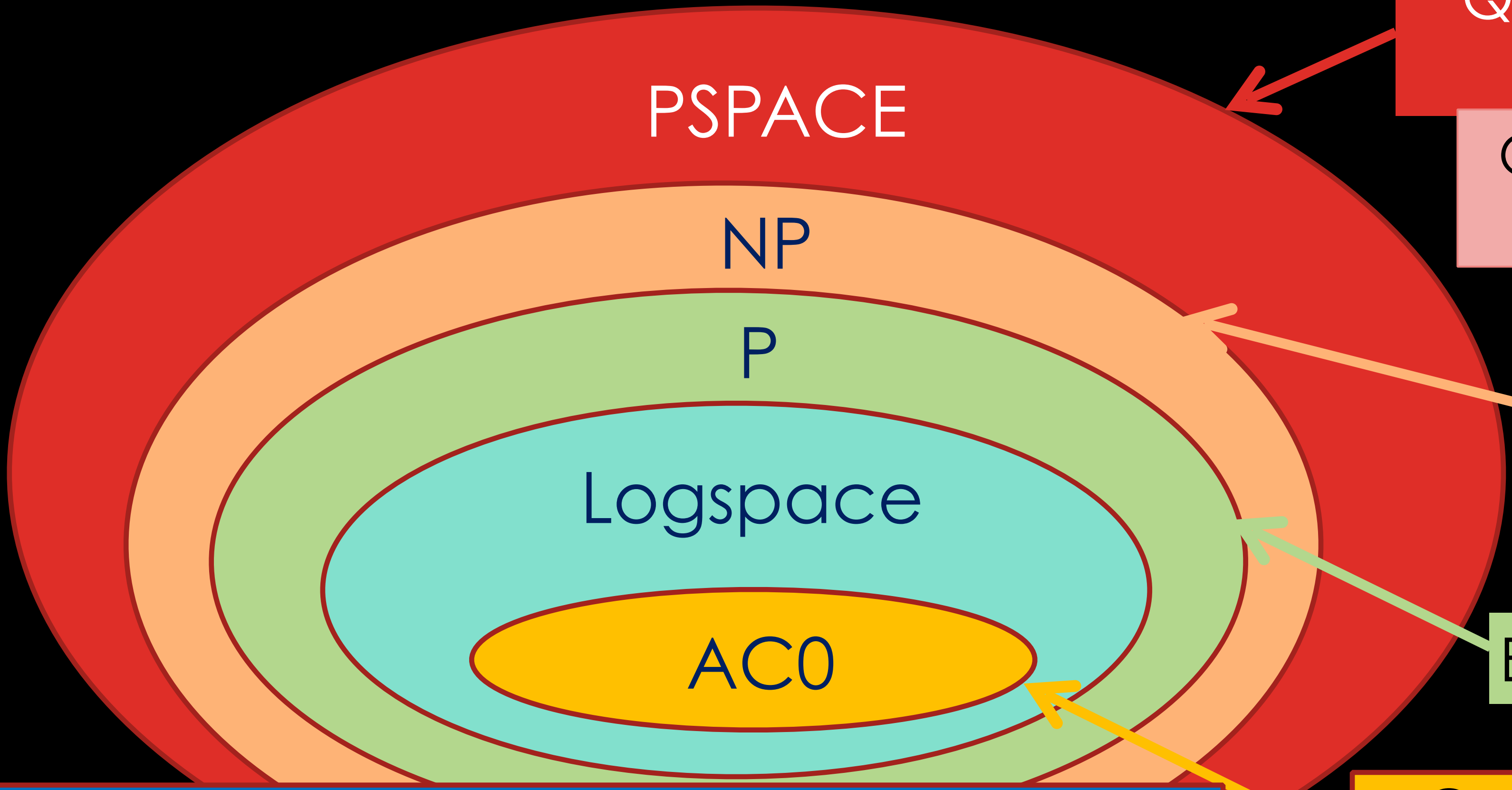
Why are we stuck?

Are we stuck because of the same reason?

PLAN

- Traditional hardness in complexity
- A fine-grained approach
- New Developments

COMPUTATIONAL COMPLEXITY



QBF with k alternations in $2^{n-\Omega(k)}$ time

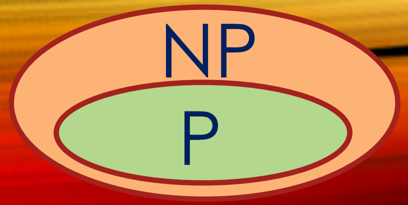
QBF gets easier as the # of quantifiers increases...

Best SAT alg: 2^n

Even $O(n^2)$ time is inefficient

Contains the "1000-clique" problem; best runtime: $\Omega(n^{790})$

This traditional complexity class approach says little about runtime!



N – size of input

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

WHY IS K-SAT HARD?

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

Theorem [Cook, Karp'72]:

k-SAT is **NP-complete** for all k ≥ 3.

That is, if there is an algorithm that solves k-SAT instances on n variables in ***poly(n)*** time, then all problems in NP have *poly(N)* time solutions, and so P=NP.

k-SAT (and all other NP-complete problems) are considered *hard because* ***fast algorithms for them imply fast algorithms for many important problems***.

TIME HIERARCHY THEOREMS

For most natural computational models one can prove:

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

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

Unconditional lower bounds seem hard.

In fact, it is not even known if SAT is in linear time!

We instead develop a *fine-grained theory of hardness* that is conditional and mimics NP-completeness.

PLAN

- Traditional hardness in complexity
- A fine-grained approach
- Fine-grained reductions lead to new algorithms

FINE-GRAINED HARDNESS *IDEA*

Idea: Mimic NP-completeness

1. Identify key hard problems

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

3. Hopefully form *equivalence classes*

Goal:

understand the landscape of algorithmic problems

CNF SAT IS CONJECTURED TO BE REALLY HARD

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

ETH: 3-SAT requires $2^{\delta n}$ time for some constant $\delta > 0$.

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.

So we can use k-SAT as our hard problem and ETH or SETH as the conjecture we base hardness on.

Recent research [CGIMPS'16] suggests these problems are **not equivalent**!

More key problems to blame

Orthogonal vectors

Given a set S of n vectors in $\{0,1\}^d$, for d = ω(log n) are there u, v 2 S with **u ¢ v = 0**?

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

[W'04]: SETH implies this conjecture!

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

3SUM

Given a set S of n integers, are there a, b, c 2 S with **a + b + c = 0**?

Easy O(n^2) time alg
[BDP'05]: ~$\mathbf{n^2}$ / $\log^2$ n time for integers
[GP'14] : ~$\mathbf{n^2}$ / log n time for reals

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

APSP

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

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

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

WORK ON APSP

Author	Runtime	Year
Floyd, Warshall	n^3	*1962*
Fredman	n^3	*1976*
Takaoka	n^3	*1992*
Dobosiewicz	n^3	*1992*
Han	n^3	*2004*
Takaoka	n^3	*2004*
Zwick	n^3	*2004*
Chan	n^3	*2005*
Han	n^3	*2006*
Chan	n^3	*2007*
Han, Takaoka	n^3	*2012*
Williams	n^3	*2014*

Classical problem
Long history

FINE-GRAINED HARDNESS

Idea: Mimic NP-completeness

1. Identify key hard problems

2. Reduce these to all (?) other hard problems

3. Hopefully form *equivalence classes*

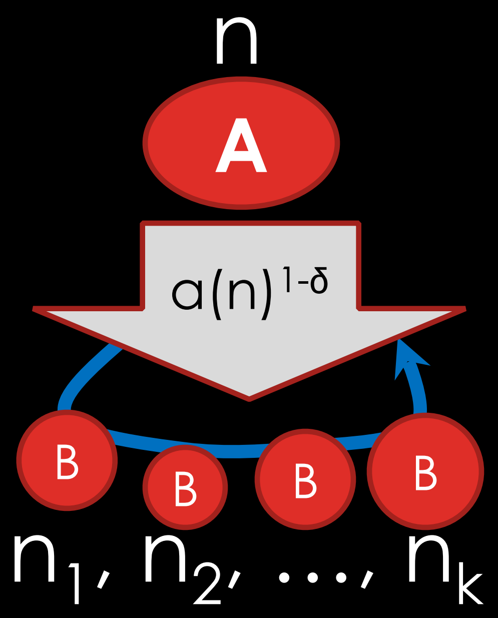
Goal:

understand the landscape of algorithmic problems

FINE-GRAINED REDUCTIONS

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.

- A is (a,b)-reducible to B if
  for every $\varepsilon>0$ $\exists$ $\delta>0$, and an $O(a(n)^{1-\delta})$ time algorithm
  that adaptively transforms any A-instance of size n to B-instances
  of size $n_1,\ldots,n_k$ so that $\sum_i b(n_i)^{1-\varepsilon} < a(n)^{1-\delta}$.



- **If B is in** $O(b(n)^{1-\varepsilon})$ **time,**
  then A is in $O(a(n)^{1-\delta})$ time.
- Focus on exponents.
- We can build equivalences.

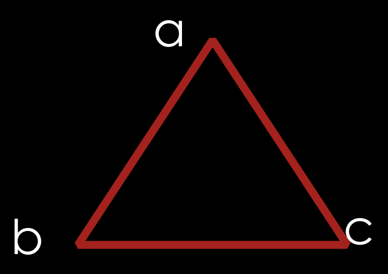
Next: an example

AN EXAMPLE FINE-GRAINED EQUIVALENCE

THEOREM [VW'10]: Boolean matrix multiplication (BMM) is equivalent to Triangle detection under *subcubic* fine-grained reductions.

BMM: Given two n x n Boolean matrices X and Y, return an n x n matrix Z where for all i and j, $Z[i , j] = OR_k (X[i , k] \text{ AND } Y[k , j])$.

Triangle detection: Given an n node graph G, does it contain three vertices a, b, c, such that (a, b), (b, c), (c, a) are all edges?



We will show that
(1) an $O(n^{3-e})$ time alg for BMM can give an $O(n^{3-e})$ time triangle alg, and
(2) an $O(n^{3-e})$ time alg for triangle can give an $O(n^{3-e/3})$ time BMM alg.

BMM: Given two n x n Boolean matrices X and Y, return an n x n matrix Z where for all i and j,
$Z[i , j] = OR_k (X[i , k] \text{ AND } Y[k , j])$.

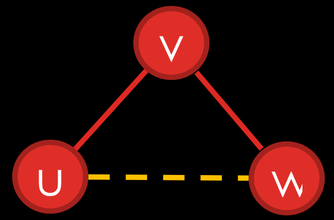
If one can multiply Boolean matrices in $O(n^c)$ time, then one can find a triangle in a graph in $O(n^c)$ time.

BMM CAN SOLVE TRIANGLE (ITAI, RODEH'1978)

G=(V,E) - n node graph. A – n x n adjacency matrix: for all pairs of nodes u,v

A[u, v] = 1 if (u, v) is an edge and 0 otherwise.

Say **Z = Boolean product of A with itself.** Then for all pairs of nodes u ≠ w,



Z[u, w] =

$OR_v (A[u , v] \text{ AND } A[v , w]) =$
- 1 if there is a path of length 2 from u to w.
- and
- 0 otherwise.

So G has a triangle iff there is some edge (u, w) in G s.t. Z[u, w] = 1.

BMM: Given two n x n Boolean matrices X and Y, return an n x n matrix Z where for all i and j,

$$Z[i , j] = OR_k (X[i , k] \text{ AND } Y[k , j]).$$

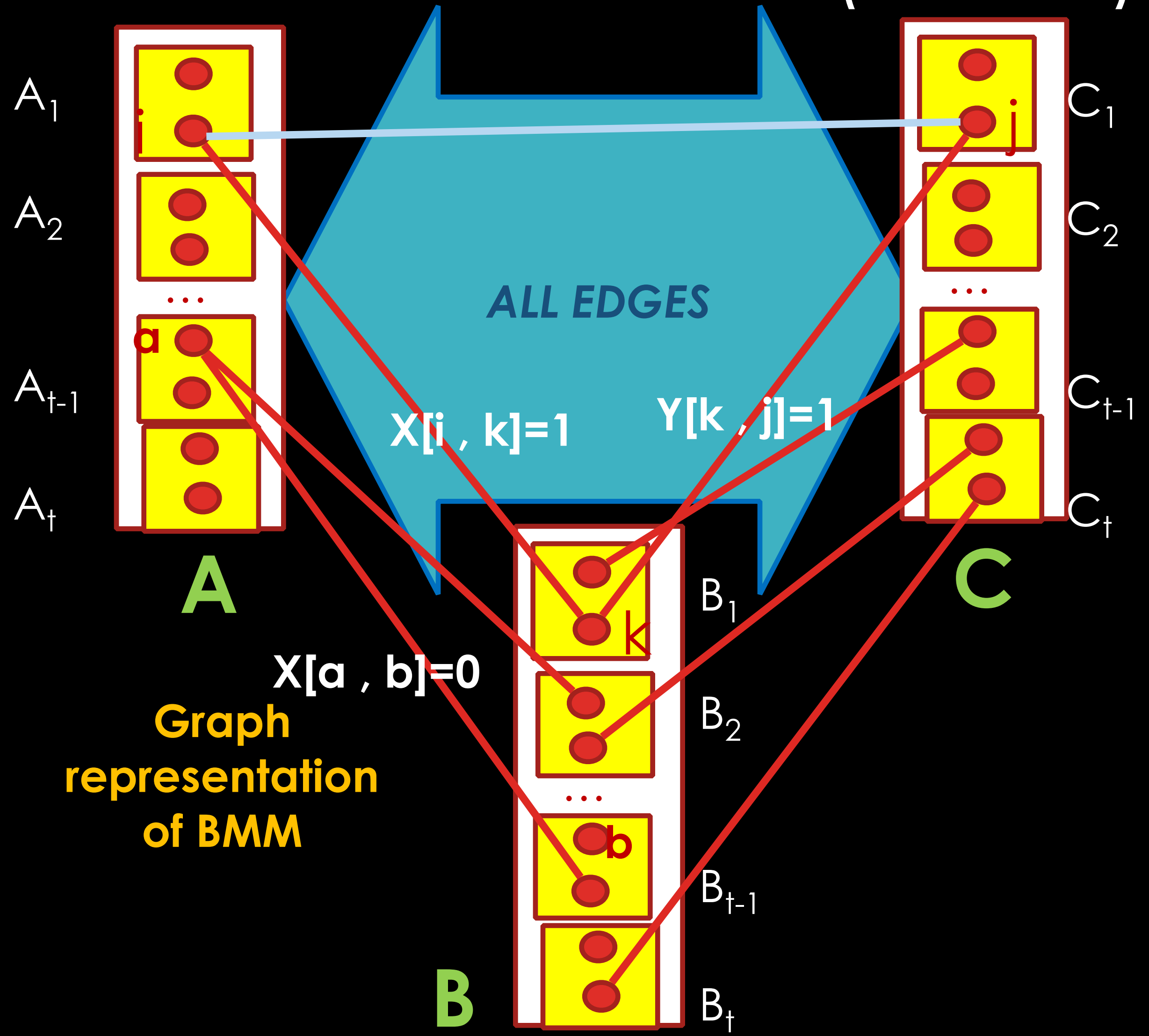
A: rows of X,
B: cols of X and rows of Y,
C: cols of Y

TRIANGLE CAN SOLVE BMM (VW'10)

Reduction from BMM to triangle finding:

$t = n^{2/3}$

- Split A into pieces $A_1,\ldots,A_t$ of size n/t
- Split B into pieces $B_1,\ldots,B_t$ of size n/t
- Split C into pieces $C_1,\ldots,C_t$ of size n/t
- Place an edge between every i in A and every j in C
- Z – all zeros matrix
- For all **triples** A_p,B_q,C_r in turn:
  - While A_p [B_q [C_r has a triangle,
    - Let (i, j, k) be a triangle in A_p [B_q [C_r
    - Set Z[i , j] = 1
    - **Remove (i , j) from the graph.**



Z – all zeros matrix
For all **triples** A_p,B_q,C_r in turn:
While **A_p [B_q [C_r** has a triangle,
Let (i, j, k) be a triangle in A_p [B_q [C_r
Set Z[i , j] = 1
Remove (i , j) from the graph.

BMM TO TRIANGLE REDUCTION

Correctness: Every triple of nodes i, j, k appears in some examined **A_p [B_q [C_r**

Runtime: Every call to the Triangle finding algorithm is due to either
(1) Setting an entry Z[i, j] to 1, or

this happens at most once per pair i, j

(2) Determining that some triple **A_p [B_q [C_r** doesn't have any more triangles

this happens at most once per triple **A_p [B_q [C_r**

If the runtime for detecting a triangle is **$T(n) = O(n^{3-\varepsilon})$**, then the reduction time is

$(n^2 + t^3)\ T(n/t)$. Setting $t=n^{2/3}$, we get: $O(n^{3 - \varepsilon/3})$.

FINE-GRAINED HARDNESS

Idea: Mimic NP-completeness

1. Identify key hard problems

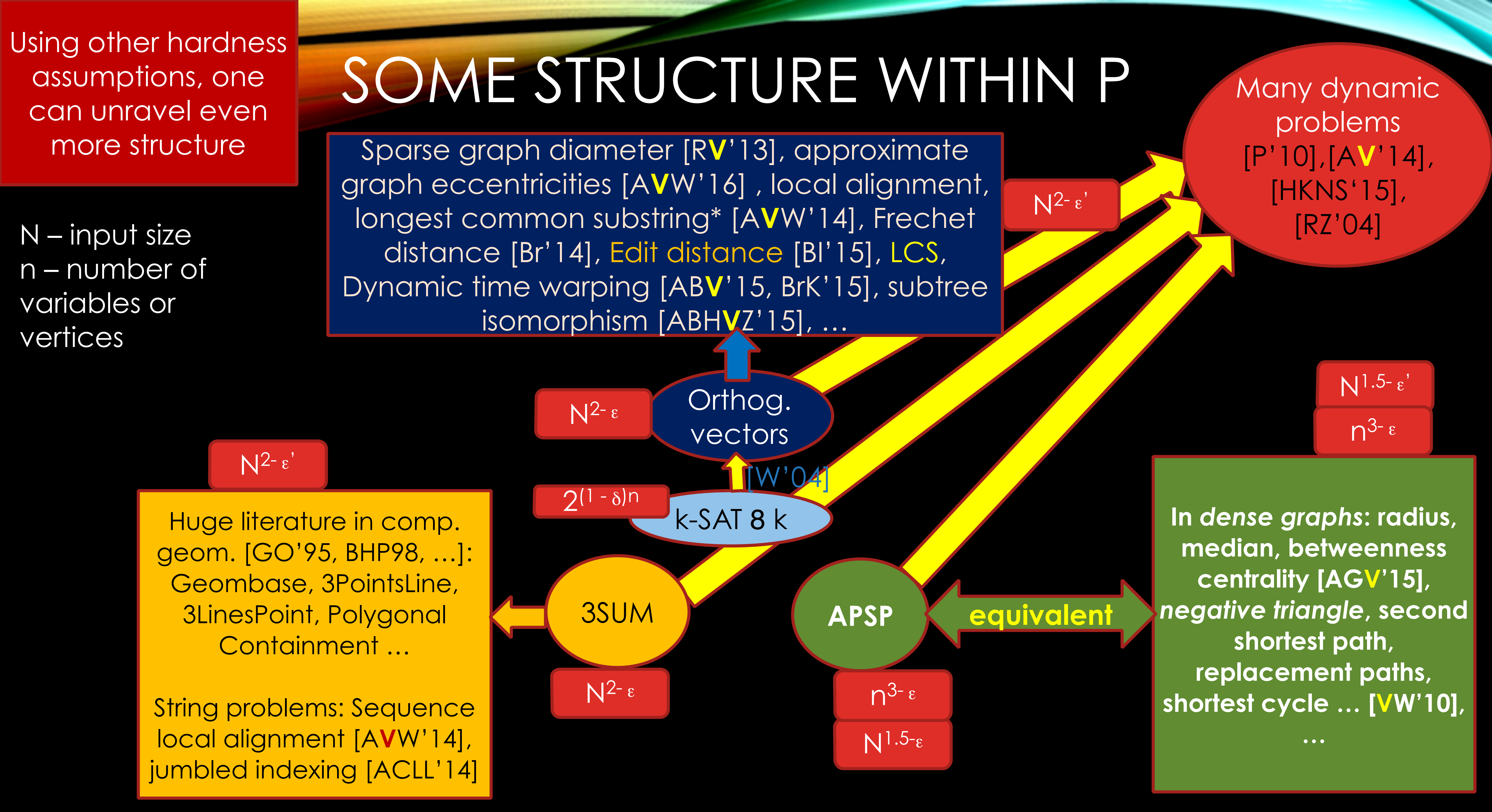
2. Reduce these to all (?) other hard problems

3. Hopefully form *equivalence classes*

Goal:

understand the landscape of algorithmic problems

SOME STRUCTURE WITHIN P



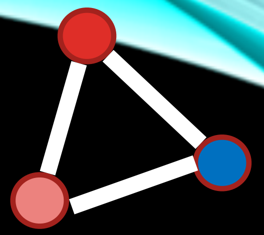
PLAN

- Traditional hardness in complexity
- A fine-grained approach
- New developments
  - The quest for more believable conjectures

THE QUEST FOR MORE PLAUSIBLE CONJECTURES

- Two problems harder than CNF-SAT,3SUM, and APSP
- Longest common subsequence, Formula SAT and Branching Programs

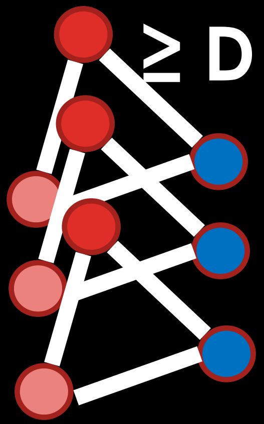
[Abboud-VW-Yu STOC'15]: Two hard problems for node-colored graphs



- 3SUM — $n^{2-\varepsilon'}$
- k-SAT for all k — $(2^n)^{1-\varepsilon'}$
- APSP — $n^{3-\varepsilon'}$

↓

- Matching Triangles — $n^{3-\varepsilon}$
- Triangle Collection — $n^{3-\varepsilon}$



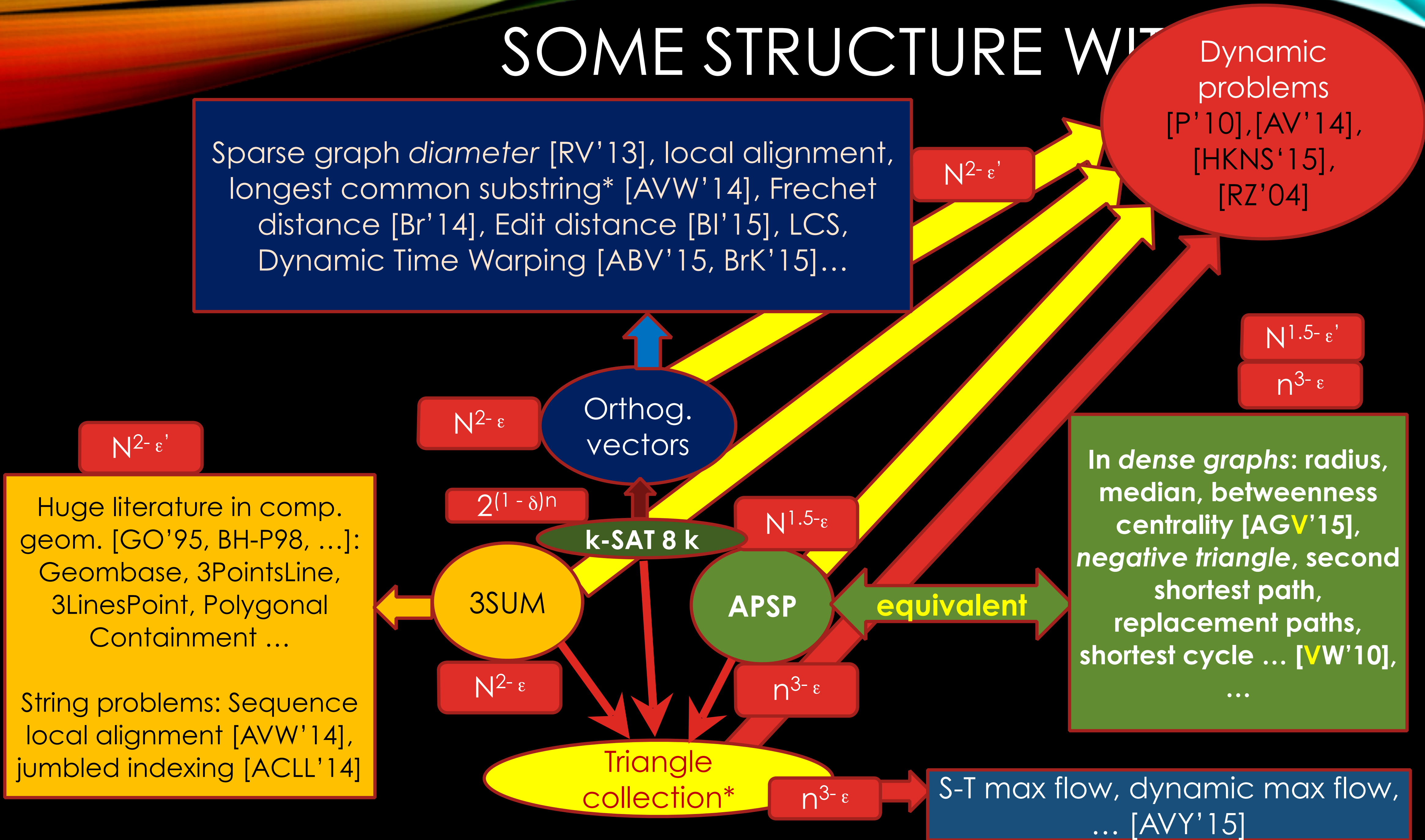
Matching Triangles: Given an n-node graph G, a color for every vertex in G, and an integer D, is there a triple of colors q1,q2,q3 such that there are at least D triangles in G with node colors exactly q1,q2,q3?

Triangle Collection: Given an n-node graph G and a color for every vertex in G, is there a triple of colors q1,q2,q3 such that there are no triangles in G with node colors exactly q1,q2,q3?

1. **Graphs don't have weights, just node colors**
2. **Any reduction from these problems would imply hardness under all three conjectures!**

NO

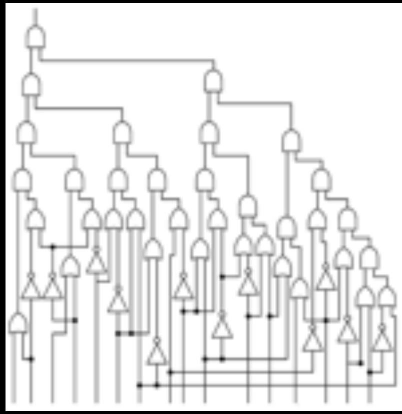
SOME STRUCTURE WIT



THE QUEST FOR MORE PLAUSIBLE CONJECTURES

- Two problems harder than CNF-SAT,3SUM, and APSP
- Longest common subsequence, Formula SAT and Branching Programs

CIRCUIT-STRONG-ETH



- The most successful hypothesis has been SETH
- It is ultimately about SAT of *linear size* CNF-formulas
- There are more difficult satisfiability problems:
  - **CIRCUIT-SAT**
  - **NC-SAT**
  - **NC1-SAT** …

C-SETH: satisfiability of circuits from circuit class C on n variables and size s requires $2^{n-o(n)}$ poly(s) time.

- [Williams'10,'11]: a $2^n/n^{10}$ time SAT algorithm implies *circuit lower bounds for C* (for E^{NP} and others); the bigger the class the stronger the lower bound

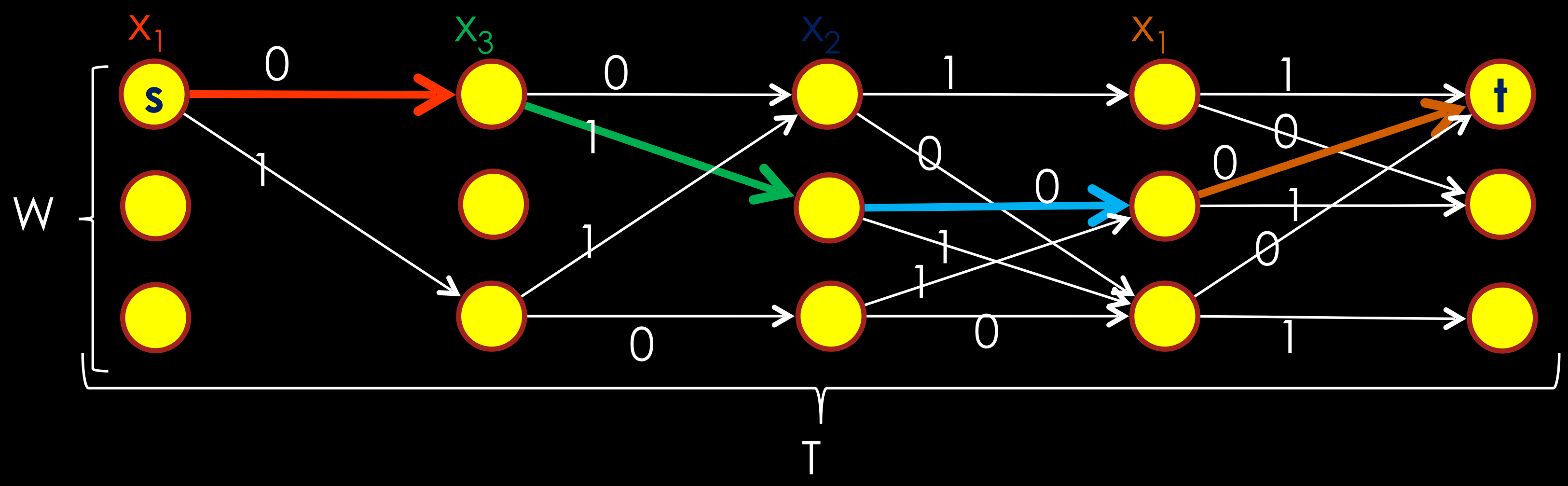
A VERY RECENT DEVELOPMENT

Theorem [Abboud-Hansen-VW-Williams'16]: There is an efficient reduction from **Satisfiability for non-deterministic branching programs** (BPs) of size **T** and width **W** and **n** input variables to the following string problems on strings of length N = $\mathbf{2^{n/2}\, T^{O(\log W)}}$:

Longest Common Subsequence, Edit Distance, Dynamic Time Warping, etc.

THEOREM [ABBOUD-HANSEN-VW-WILLIAMS'15]: THERE IS AN EFFICIENT REDUCTION FROM **SATISFIABILITY FOR NON-DETERMINISTIC BRANCHING PROGRAMS** (BPS) OF SIZE **T** AND WIDTH **W** AND **N** INPUT VARIABLES TO THE FOLLOWING STRING PROBLEMS ON STRINGS OF LENGTH N = $2^{N/2}$ $T^{O(\log W)}$:
LONGEST COMMON SUBSEQUENCE, EDIT DISTANCE, DYNAMIC TIME WARPING, ETC.

A type of reachability question. Proof encodes a Savitch-like construction into the LCS/Edit distance instance.



BP: edge-labelled, directed, layered graph. **Start** node s, **accept** node t.
Width: W nodes per layer. **Size**: T layers.
Each layer labeled with a variable. A variable can label many layers.
Each edge labeled with 0 or 1.
An input 001 is accepted if it generates an s-t path.

A VERY RECENT DEVELOPMENT

Theorem [Abboud-Hansen-VW-Williams'16]: There is an efficient reduction from **Satisfiability for non-deterministic branching programs** (BPs) of size **T** and width **W** and **n** input variables to the following string problems on strings of length N = $\mathbf{2^{n/2}\, T^{O(\log W)}}$:

Longest Common Subsequence, Edit Distance, Dynamic Time Warping, etc.

[Barrington'85]: BPs with $T=2^{\text{polylog } n}$ and W=5 capture NC.

The above problems require $\mathbf{N^{2-o(1)}}$ time under **NC-SETH**.

MORE CONSEQUENCES OF
"BP-SAT FOR N,T,W → EDIT DISTANCE ETC. ON $2^{N/2}\, T^{O(\log W)}$"

Impressive SAT algs

BPs with $T=2^{o(\text{sqrt } n)}$ and $W=2^{o(\text{sqrt } n)}$ can represent any *non-deterministic Turing machine* using o(sqrt n) space

- Edit Distance (or LCS etc) in $O(n^{2-\varepsilon})$ time implies a nontrivial improvement over exhaustive search for checking SAT of complex objects that can easily implement e.g. *cryptographic primitives*
- Much more surprising than refuting SETH!

MORE CONSEQUENCES OF

"BP-SAT FOR N,T,W → EDIT DISTANCE ETC. ON $2^{N/2}$ $T^{O(\log W)}$"

Circuit Lower Bounds

If Edit Distance (or LCS etc) has $O(n^{2-\varepsilon})$ time algorithms for any $\varepsilon > 0$, then $\mathbf{E}^{\mathbf{NP}}$ does not have:

- **Non-uniform $2^{o(n)}$-size Boolean Formulas**

  we don't even know if the enormous Σ_2EXP has $2^{o(n)}$-size depth-3 circuits
- **Non-uniform o(n)-depth circuits of bouded fan-in**
- **Non-uniform $2^{o(\mathrm{sqrt}(n))}$-size non-deterministic branching programs**

"A POLYLOG SHAVED IS A LOWER BOUND MADE"

[Williams'14, Abboud-Williams-Yu'15]: APSP can be solved in **$n^3 / \log^{\omega(1)} n$** time, OV can be solved in **$n^2/\log^{\omega(1)} n$** time

Does Edit Distance (or LCS etc) have such an algorithm?

(The current best algorithms run in ~ $n^2/\log^2 n$ time.)

PARTIAL ANSWER: An $n^2 / \log^{\omega(1)} n$ algorithm for Edit Distance (or LCS etc) implies that E^{NP} is not in NC1.

$n^2/\log^{2.1} n$?

Also meaningful for particular polylogs. E.g. if Edit Distance (or LCS etc) has an **$n^2/\log^{100} n$** time algorithm, then E^{NP} does not have non-uniform Boolean formulas of size **n^5**.

Thank you!