Hardness for easy problems

An introduction

The real world and hard problems

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 easy problems

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.

How do we explain when we are stuck?

The "easy" problems

Let's focus on $O(N^2)$ time

(N- size of the input)

What do we know about $O(N^2)$ time?

- Amazingly fast algorithms for:
  - *Classical (almost) linear time*: connectivity, planarity, minimum spanning tree, single source shortest paths, min cut, topological order of DAGs …
  - *Recent breakthroughs*: solving SDD linear systems ($m^{1+o(1)}$, [ST'04] …), approx. max flow ($m^{1+o(1)}$, [KLOS'13]), max matching ($m^{10/7}$, [M'14]), min cost flow ($\sim m\sqrt{n}$, [LS'14]), …
  - Sublinear algorithms/property testing

Good news: A lot of problems are in close to linear time!

Hard problems in $O(N^2)$ time

In general, for many problems in P, there's an ``easy'' $O(N^c)$ time solution but essentially no improvement in decades.

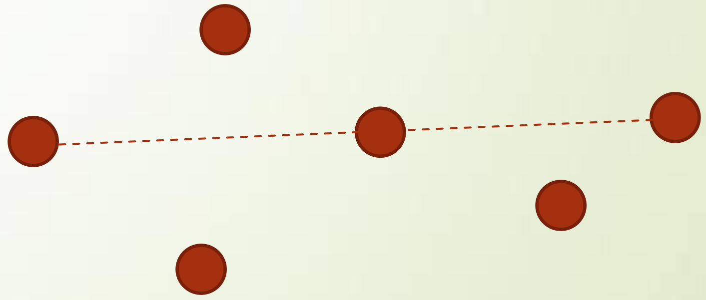
Bad news: on many problems we are stuck:

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

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

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

A very important primitive!



Hard problems in $O(N^2)$ time

Bad news: on many problems we are stuck:

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

- Many problems in *computational geometry*
- Many *string matching* problems:

Edit distance, Sequence local alignment, Longest common subsequence, jumbled indexing …

Sequence alignment

A fundamental problem from computational biology:

Given two DNA strings

ATCGGGTTCCTTAAGGG

ATTGGTACCTTCAGG

How similar are they? What do they have in common?

Several notions of similarity!

E.g. Local alignment, Edit Distance, Longest Common Subsequence

Longest Common Subsequence

Given two strings

ATCGGGTTCCTTAAGGG

ATTGGTACCTTCAGG

Find a subsequence of both strings of maximum length.

Applications both in comp. biology and in spellcheckers.

Longest Common Subsequence

Given two strings

ATCGGGTTCCTTAAGGG

ATT GG_TACCTTCA_GG

Find a subsequence of both strings of maximum length.

Applications both in comp. biology and in spellcheckers.

Solved daily on huge strings!!

Sequence problems theory/practice

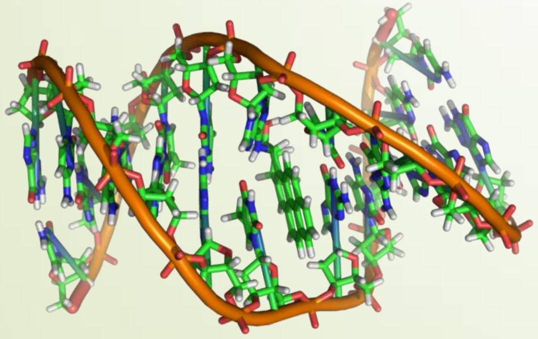
Fastest algorithm for most sequence alignment variants:

$O(n^2)$ time on length n sequences

Sequence alignment is run on whole genome sequences.

Human genome: 3×10^9 base pairs.

A quadratic time algorithm is not fast!



Hard problems in $O(N^2)$ time

Bad news: on many problems we are stuck:

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

- Many problems in *computational geometry*
- Many *string matching* problems
- 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!

Hard problems in $O(N^2)$ time

Bad news: on many problems we are stuck:

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

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

No $N^{1.5-\varepsilon}$ time algs known for many problems in dense graphs: diameter, radius, median, second shortest path, shortest cycle…

$N^{1.5}$ time algs exist

i.e. n^3

Why are we stuck?

We are stuck on many problems from different subareas of CS!

Are we stuck because of *the same reason*?

How do we address this?

How did we address this for the hard problems?

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

Trivial algorithm: try all 2^n assignments

Best known algorithm: $O(2^{n-(cn/k)}n^d)$ time for const c,d

Why is k-SAT hard?

Theorem [Cook,Karp'72]:

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

Tool: poly-time reductions

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 algs for many important problems***.

Addressing the hardness of easy problems

1. Identify key hard problems

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

3. Hopefully form *equivalence classes*

Goal: *understand the landscape of polynomial time.*

CNF SAT is conjectured to be really hard

Two popular conjectures about SAT [IPZ01]:

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

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.

Three more problems we can blame

- **3SUM**: Given a set S of n integers, are there $a,b,c \in S$ with $\mathbf{a+b+c = 0}$?

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

- **All pairs shortest paths (APSP)**: given a weighted graph, find the **distance** between every two nodes.

3SUM: Given a set S of n numbers, are there $a,b,c \in S$ with $\mathbf{a+b+c = 0}$?

- Easy $O(\mathbf{n^2})$ time algorithm
- [BDP'05]: $\sim\mathbf{n^2}/\log^2 n$ time algorithm for integers
- [GP'14] : $\sim\mathbf{n^2}/\log n$ time for real numbers
- Here we'll talk about 3SUM over the integers
- Folklore: one can assume the integers are in $\{-n^3,\ldots,n^3\}$

3SUM Conjecture: 3SUM on n integers in $\{-n^3,\ldots,n^3\}$ requires $n^{2-o(1)}$ time.

Three more problems we can blame

- **3SUM**: Given a set S of n integers, are there
  $a,b,c \in S$ with $\mathbf{a+b+c = 0}$?

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

- **All pairs shortest paths (APSP)**: given a weighted graph, find the **distance** between every two nodes.

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

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

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

- [W'04]: SETH implies the OV Conjecture.
- I'll prove this to you later

Three more problems we can blame

- **3SUM**: Given a set S of n integers, are there

  $a,b,c \in S$ with $\mathbf{a+b+c = 0}$?

- **Orthogonal vectors (OV):** Given a set S of n vectors

  in $\{0,1\}^d$, for $d = O(\log n)$ are there $u,v \in S$ with $\mathbf{u \cdot v = 0}$?

- **All pairs shortest paths (APSP)**: given a weighted graph, find the **distance** between every two nodes.

APSP: given a weighted graph, find the **distance** between every two nodes.

Classical problem
Long history

APSP Conjecture: APSP on n nodes and O(log n) bit weights requires $n^{3-o(1)}$ time.

Author	Runtime	Year
Fredman	$n^3 \log\log^{1/3} n / \log^{1/3} n$	1976
Takaoka	$n^3 \log\log^{1/2} n / \log^{1/2} n$	1992
Dobosiewicz	$n^3 / \log^{1/2} n$	1992
Han	$n^3 \log\log^{5/7} n / \log^{5/7} n$	2004
Takaoka	$n^3 \log\log^2 n / \log n$	2004
Zwick	$n^3 \log\log^{1/2} n / \log n$	2004
Chan	$n^3 / \log n$	2005
Han	$n^3 \log\log^{5/4} n / \log^{5/4} n$	2006
Chan	$n^3 \log\log^3 n / \log^2 n$	2007
Han, Takaoka	$n^3 \log\log n / \log^2 n$	2012
Williams	$n^3 / \exp(\sqrt{\log n})$	2014

Addressing the hardness of easy problems

1. Identify key hard problems

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

3. Hopefully form *equivalence classes*

Fine-grained reductions

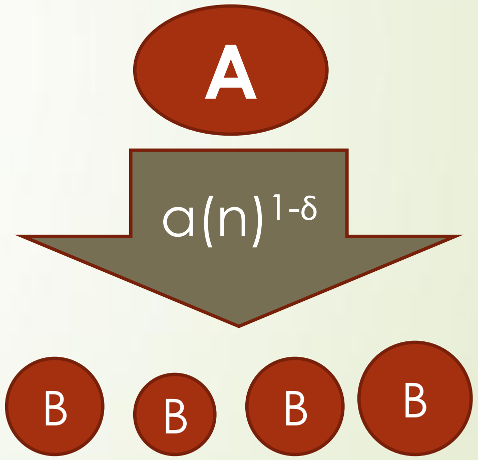
Intuition: $a(n)$,$b(n)$ are the naive runtimes for A and B. A reducible to B implies that beating the naive runtime for B implies also beating the naive runtime for A.

- 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 transforms any A-instance of size n to B-instances of size $n_1,\ldots,n_k$ so that $\Sigma_i\, b(n_i)^{1-\varepsilon} < a(n)^{1-\delta}$.

- **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.



A theory of hardness for polynomial time.

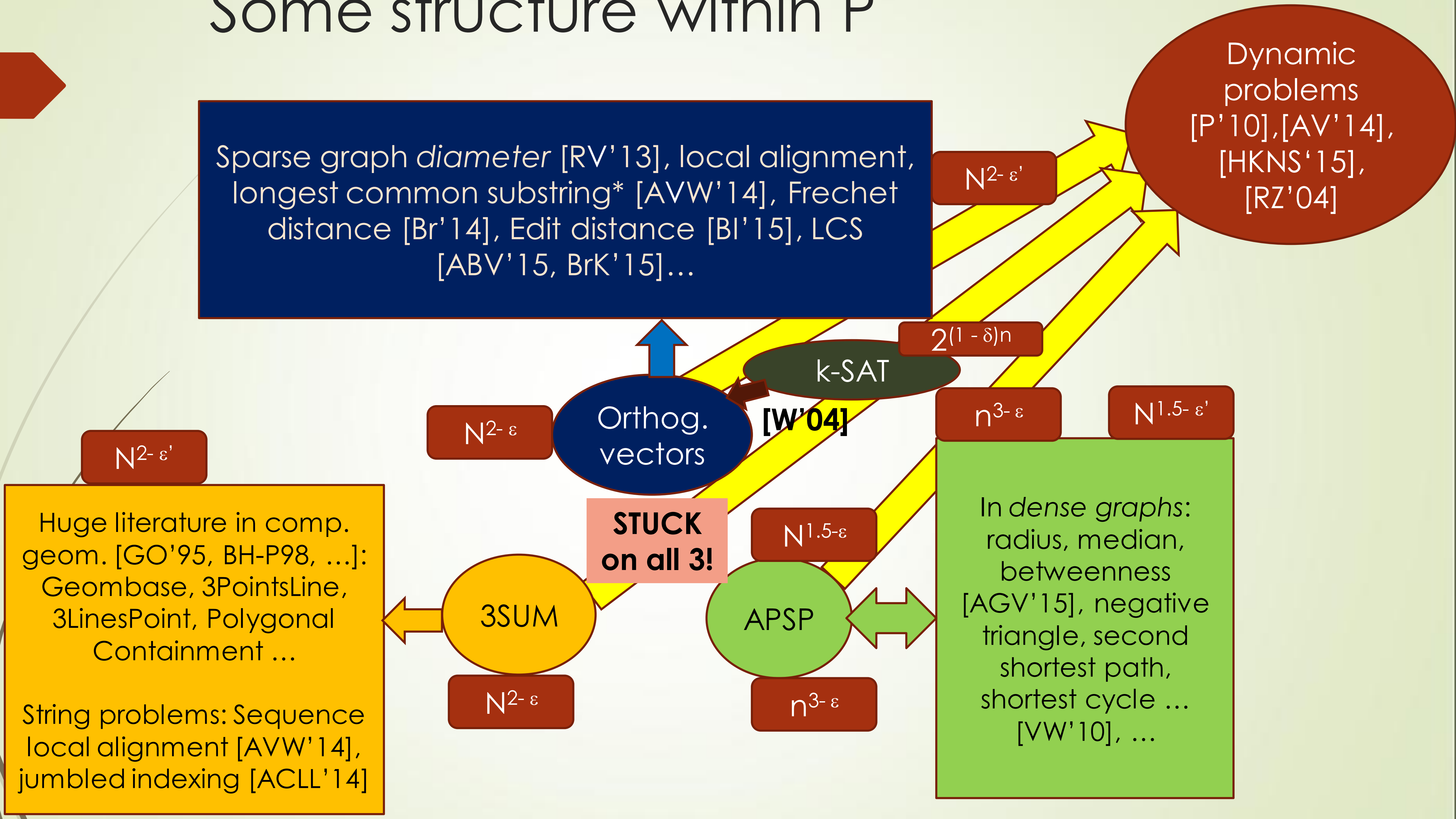
Addressing the hardness of easy problems

1. Identify key hard problems

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

3. Hopefully form *equivalence classes*

Some structure within P

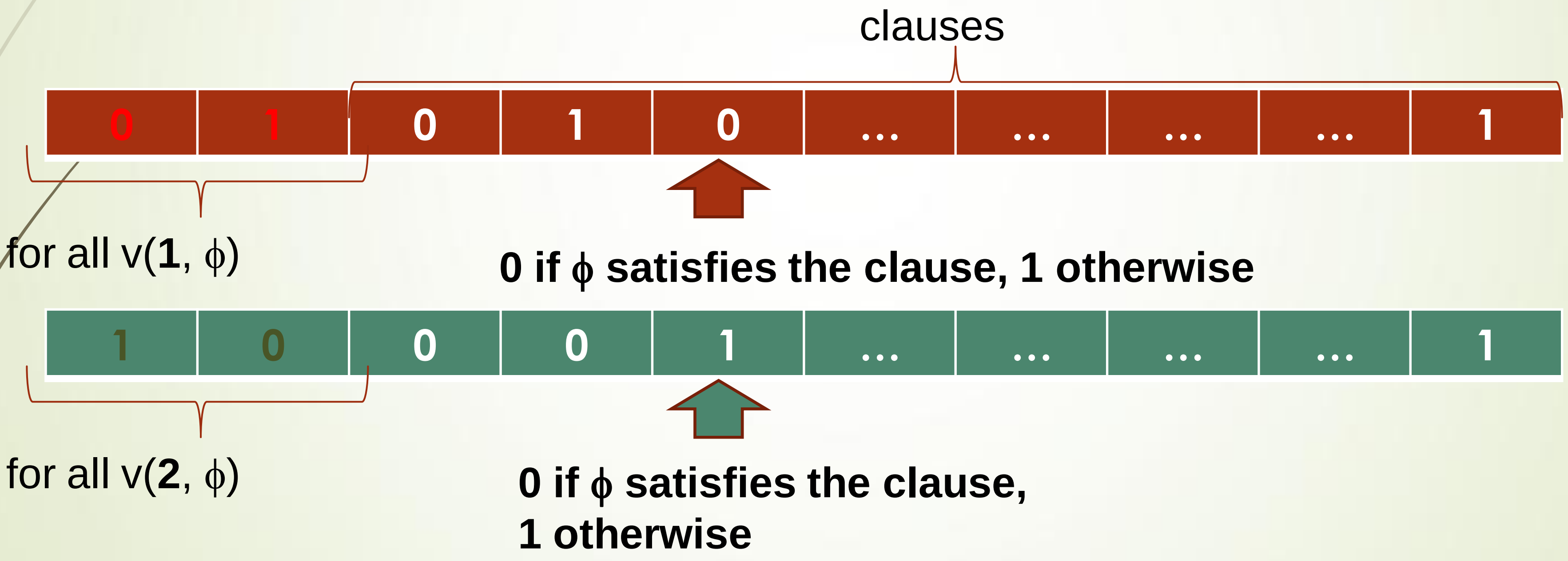


Fast OV implies SETH is false [W'04]

F- k-CNF-formula on n vars, m = O(n) clauses*

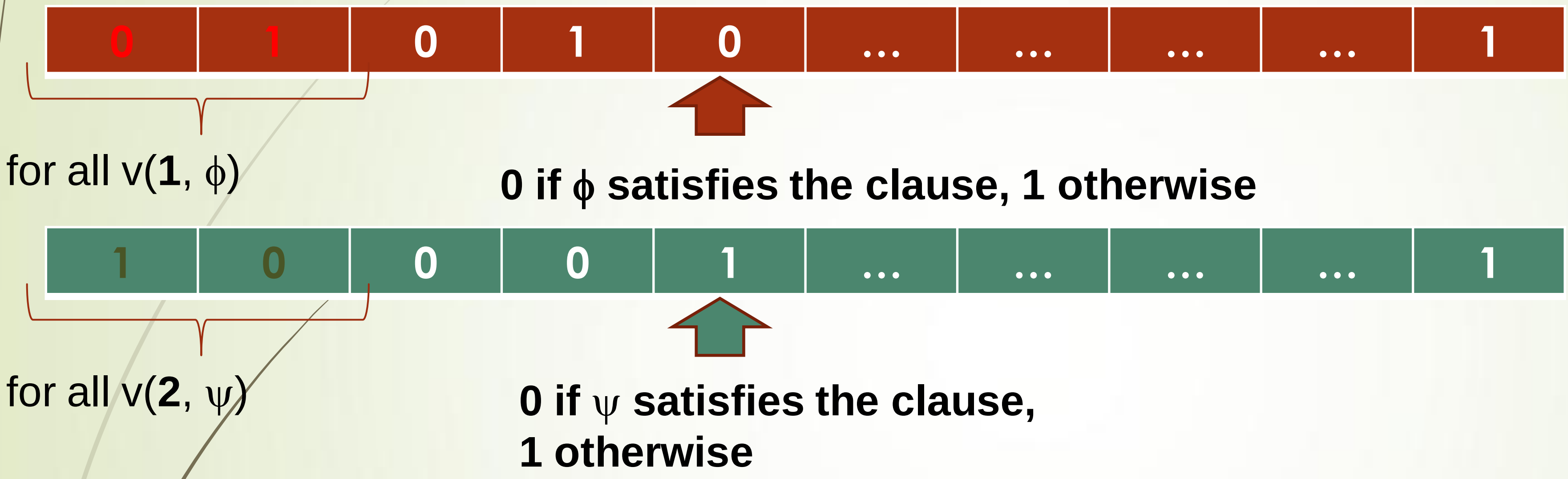
Split the vars into V_1 and V_2 on n/2 vars each

For j=1,2 and each **partial assignment** ϕ of V_j create (m+2) length vector v(j, ϕ):



*By sparsification lemma, any k-CNF can be converted into a small number of k-CNFs on O(n) clauses.

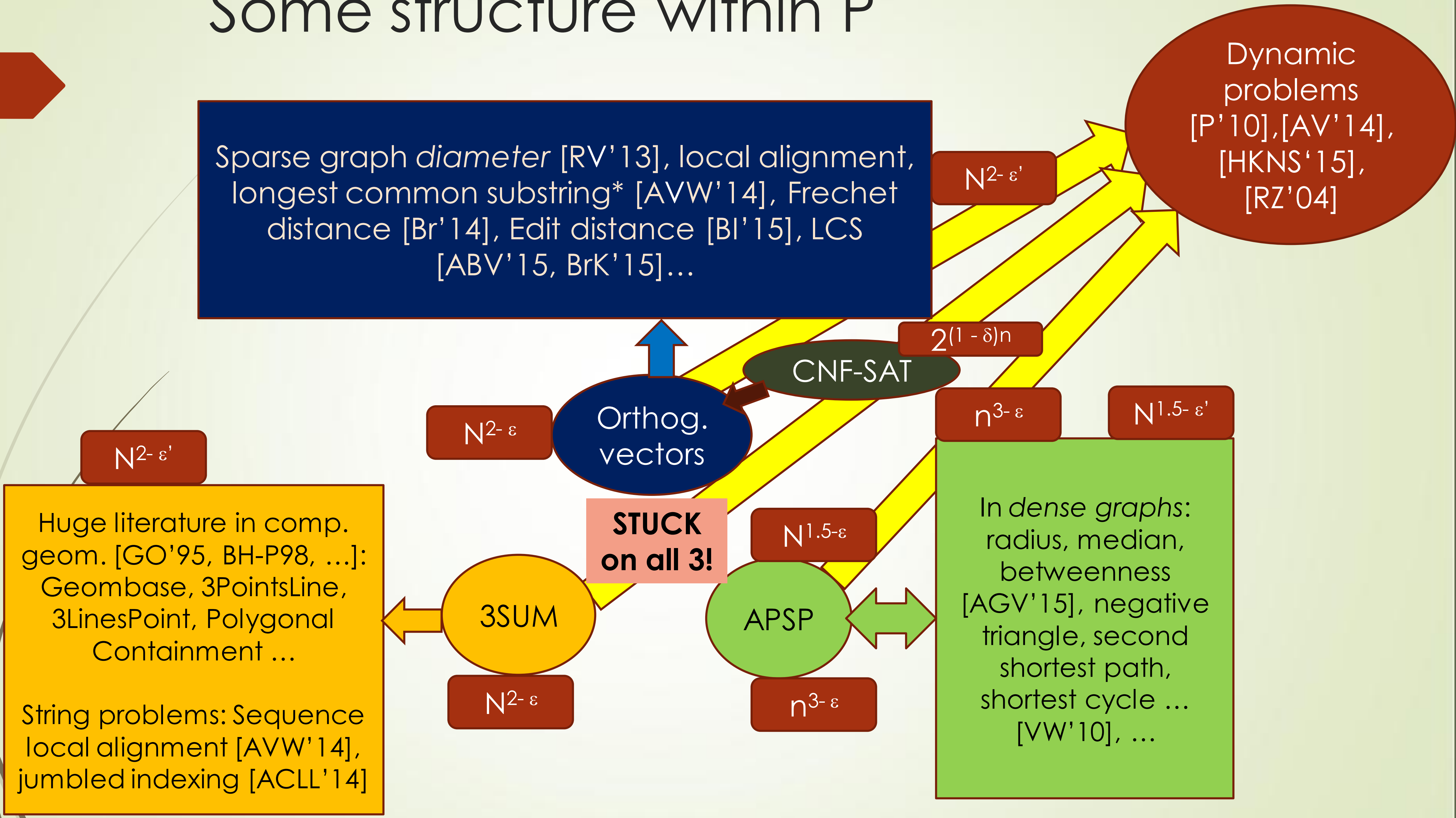
Fast OV implies SETH is false



Claim: $v(1,\phi) \cdot v(2,\psi) = 0$ iff $\phi \odot \psi$ is a sat assignment.

$N = 2^{n/2}$ vectors of dimension $O(n) = O(\log N) \rightarrow$ an OV instance.
So $O(N^{2-\delta})$ time implies SETH is false.

Some structure within P



Another popular conjecture

Boolean matrix multiplication (BMM): given two n x n boolean matrices A and B, compute their boolean product C, where $C[i, j] = \vee_k (A[i, k] \wedge B[k, j])$

BMM can be computed in $O(n^{\omega})$ time, $\omega < 2.38$. The algebraic techniques are not very practical, however.

The best known ``combinatorial'' techniques get a runtime of at best $n^3 / \log^4 n$ [Yu'15].

BMM Conjecture:

Any ``combinatorial'' algorithm for BMM requires $n^{3-o(1)}$ time.

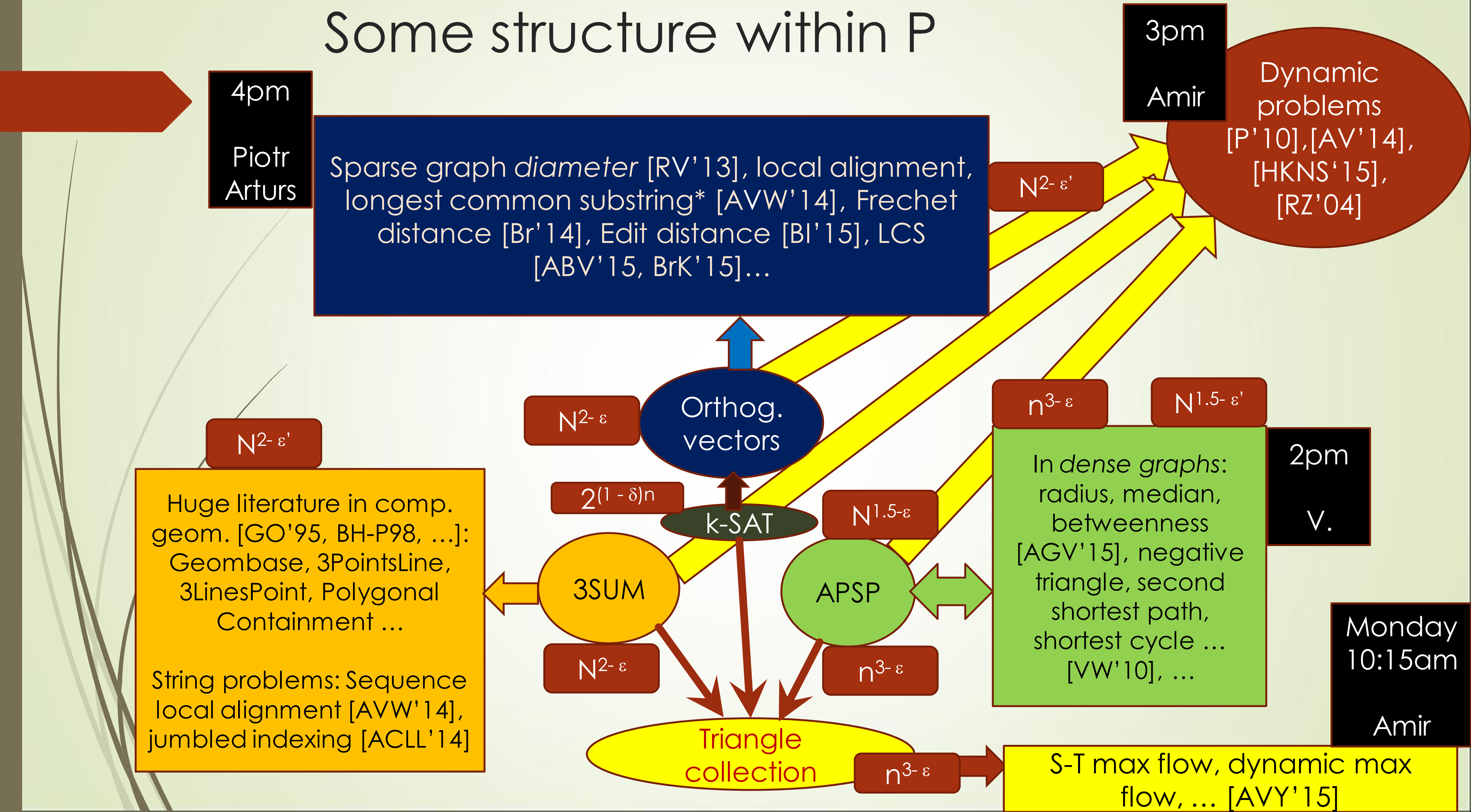
BMM conjecture consequences

- Reductions from BMM are typically used to show that fast matrix multiplication is probably required
- Some implications of the BMM conjecture:
  - A triangle in a graph cannot be found faster than in $n^{3-o(1)}$ time by any combinatorial algorithm
  - The radius of unweighted graphs requires $n^{3-o(1)}$ time via combinatorial techniques
  - Maintaining a maximum bipartite matching dynamically with nontrivial update times requires fast matrix multiplication
  - CFG parsing requires fast matrix multiplication
  - …

Which conjectures are more believable?

- Besides Ryan's proof that SETH -> OV Conjecture, there are *no other reductions* relating the conjectures known
- **It could be that one is true while all others are false**
- However:
  - The *decision tree complexities* of both 3SUM (GP'14) and APSP (Fredman'75) are low: $n^{1.5}$ and $n^{2.5}$, respectively. This is not known for OV. Perhaps this means the OV conjecture is more believable?
  - OV and APSP both admit *better than logarithmic* improvements over the naïve runtime, 3SUM does not, as far as we know. So maybe the 3SUM conjecture is more believable?
- There are natural problems that 3SUM, APSP and k-SAT **all** reduce to, *Matching Triangles & Triangle Collection*. Amir will talk about them later.

Some structure within P



Plan for the rest of the day:

- 2 pm Hardness and Equivalences for Graph Problems (Virginia)
- 3pm Hardness for Dynamic Problems (Amir)
- 4pm Intro to hardness for sequence problems (Piotr)
- 4:30pm Hardness for sequence problems (Arturs)
- 5pm Conclusion and future work (Amir)

THANKS!