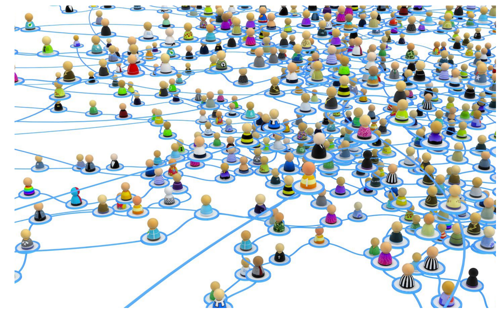
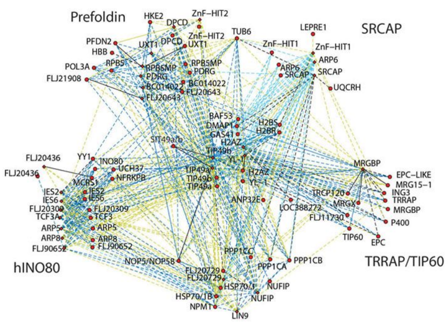
Dynamic Programming for Shortest Paths in Graphs

Why Graphs?

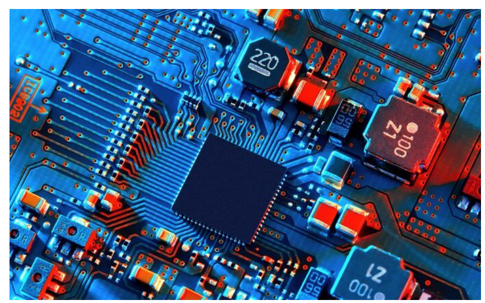
The Internet



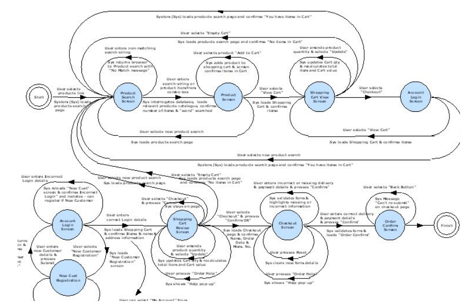
Social Networks



Biological and Chemical Networks



Circuits

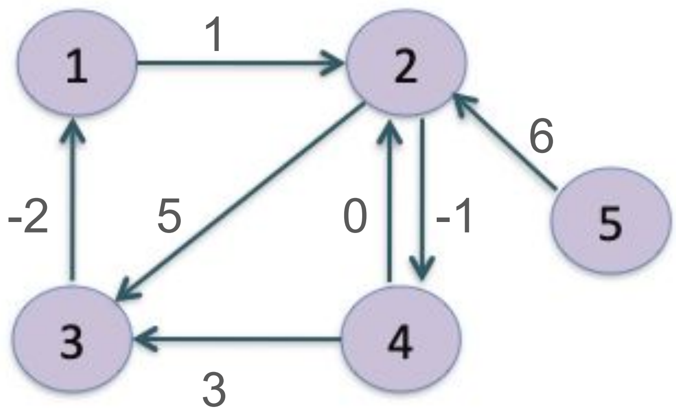


State Transition Networks

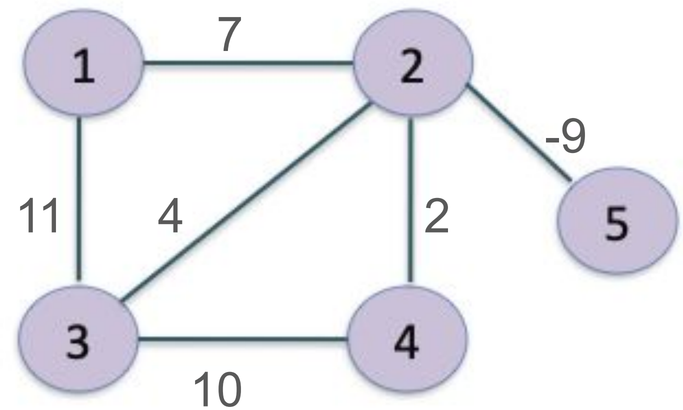
Transportation Networks

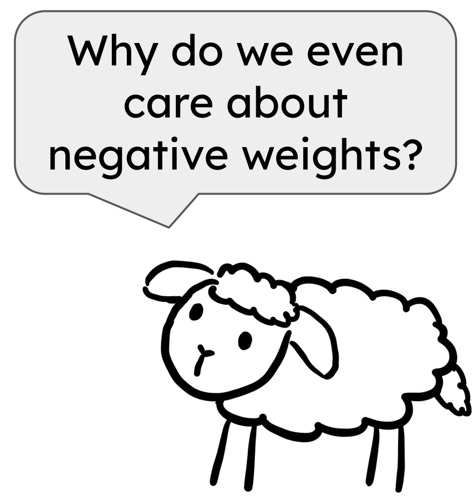
Directed and undirected graphs

Directed graph



Undirected graph





Distance from s to t (denoted **dist**(s,t)) = minimum over all paths P from s to t, of the sum of the edge weights along P.

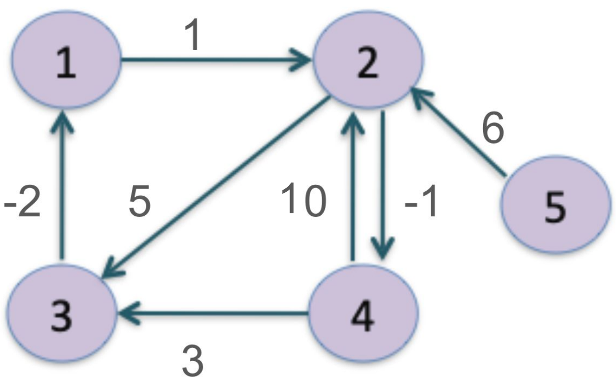
Notation: V = vertex set, E = edge set, n = |V|, m = |E|,

The shortest paths problems we'll consider

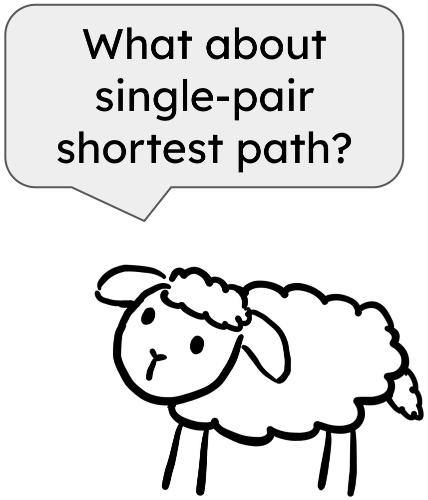
Input: Directed weighted graph. Weights can be negative but assume no negative-weight cycles.

Single-Source Shortest Paths (SSSP): Given a special vertex **s**, find a shortest path from **s** to every vertex **t**.

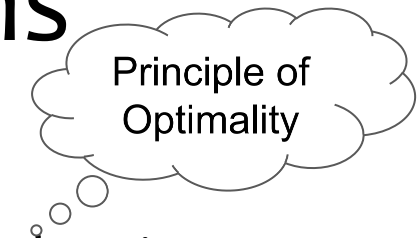
All-Pairs Shortest Paths (APSP): For every pair **s,t** of vertices, find a shortest path from **s** to **t**.



	1	2	3	4	5
1	0	1	2	0	∞
2	0	0	2	-1	∞
3	-2	-1	0	-2	∞
4	1	2	3	0	∞
5	6	6	8	5	0

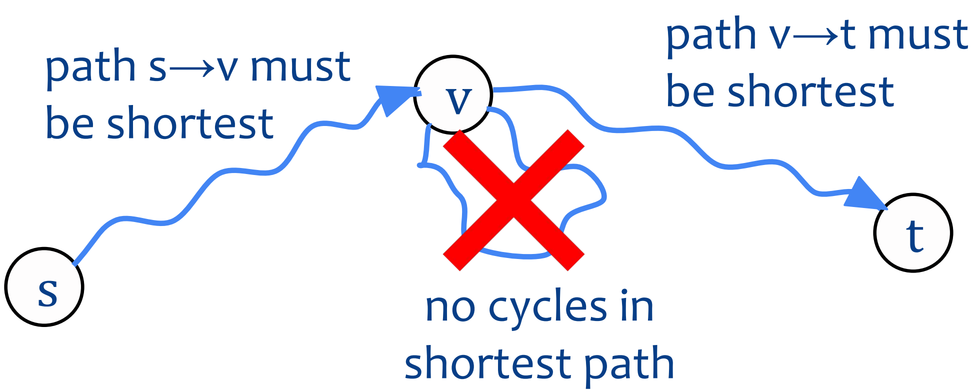


Two Key Observations

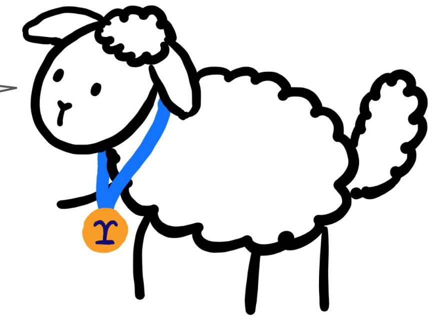


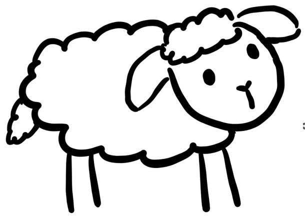
1. If a shortest path from s to t goes through vertex v, then it must take a **shortest path from s to v**, then a **shortest path from v to t.**

2. Since there are no negative-weight cycles in the graph, there is a shortest path from s to t with **no cycles** in it.



Check out my recurrence for SSSP!

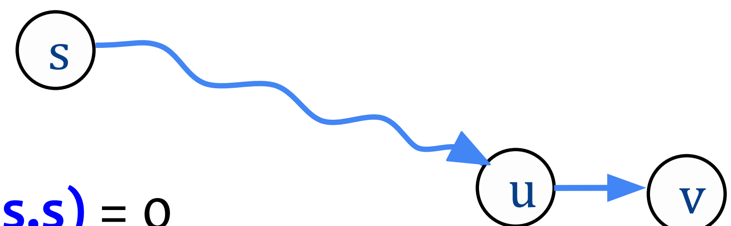




Your equation is technically correct, but it’s not really a recurrence and it doesn’t work for DP.

$$\mathrm{dist}(s,v) = \min_{(u,v)\in E}\{\mathrm{dist}(s,u) + \ell(u,v)\}$$

In the shortest s→v path,
u is the last vertex before v
(and u could be s)



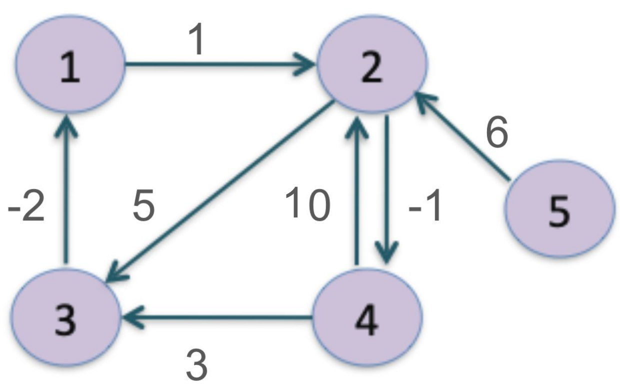
Base case: $\mathrm{dist}(s,s) = 0$

Notation:

- $\ell(y,z)$ is the weight (or “length”) of the edge y→z
- $\mathrm{dist}(y,z)$ is the distance from y to z

Recurrence for SSSP

Key Definition. Let **$dist^{(i)}(s,v)$** be the length of the shortest path from s to v that uses <u>exactly</u> i edges, or ∞ if there's no such path.



What is…

$dist^{(0)}(5,3)$?

$dist^{(1)}(5,3)$?

$dist^{(2)}(5,3)$?

$dist^{(3)}(5,3)$?

$dist^{(4)}(5,3)$?

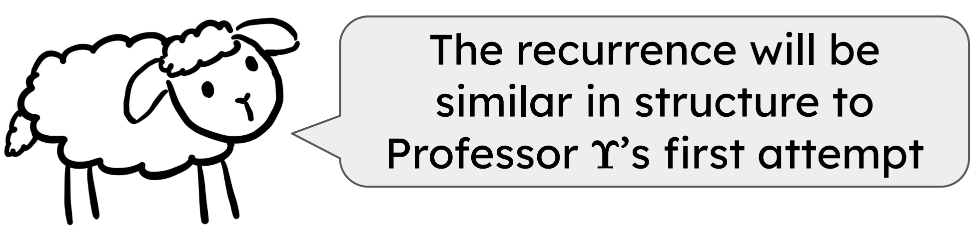
Recurrence for SSSP

Key Definition. Let **$dist^{(i)}(s,v)$** be the length of the shortest path from s to v that uses <u>exactly</u> i edges, or ∞ if there's no such path.

End goal: **$\min_{i \le n-1} dist^{(i)}(s,v)$** for each v.

$dist^{(i)}(s,v)$ =

Base case(s):



Let’s follow the DP Recipe

This is called the **Bellman-Ford Algorithm**

(Bellman, Ford, Moore, Shimbel, 1955)

The DP Recipe

1. Write recurrence ← usually the trickiest part
2. Size of table: How many dimensions? Range of each dimension?
3. What are the base cases?
4. To fill in a cell, which other cells do I look at? In which order do I fill the table?
5. Which cell(s) contain the final answer?
6. Running time = (size of table) $\cdot$ (time to fill each entry)
7. To reconstruct the solution (instead of just its size) follow arrows from final answer to base case

Pseudocode for Bellman-Ford

```
Algorithm SSSP(G, s)
    table := 2D-array indexed from 0 to n-1 in both dimensions
    // first dimension represents vertices v0,...,vn-1, where s is v0,
      second dimension represents the number i of edges
    table(0,0) = 0  // base case for s
    for k = 1 to n-1:
        table(k, 0) = ∞  // rest of base cases
    for i = 0 to n-1:    // for each number i of edges
        for k = 0 to n-1:     // for each vertex
            table(k, i) = min {table(j, i-1) + ℓ(vj,vk)}
                      (vj,vk)∈E

    Return min_i{table(k, i)} for each vertex k
```

Previously we assumed the graph has no negative-weight cycles. **But what if it does?**

With a slight modification, the Bellman-Ford (BF) Algorithm can detect whether or not there is a negative-weight cycle:

Run BF for n more iterations!

I.e calculate **$\text{dist}^{(i)}(s,v)$** for all i up to **2n-1.**

Claim: There is a negative-weight cycle (reachable from s) IFF $\min_{i \le 2n-1} \text{dist}^{(i)}(s,v) < \min_{i \le n-1} \text{dist}^{(i)}(s,v)$ for some vertex v.

*Actually it suffices to run BF for just 1 more iteration

Faster Algorithms for SSSP

- Bernstein, Nanongkai, Wulff-Nilsen, 2022: **$O(m \cdot \log^8 n)$** ← integer weights

My postdoc advisor

Thatchaphol Saranurak's PhD advisor

- Fineman, 2023: **$O(mn^{7/8})$** ← any weights
- If no negative weights, Dijkstra's algorithm: $O(m + n \log n)$

Initial idea for solving APSP: Run SSSP from every vertex

That works, but the algorithm you're about to see is faster for dense graphs: **$O(n^3)$**

Let's try essentially the same recurrence as BF

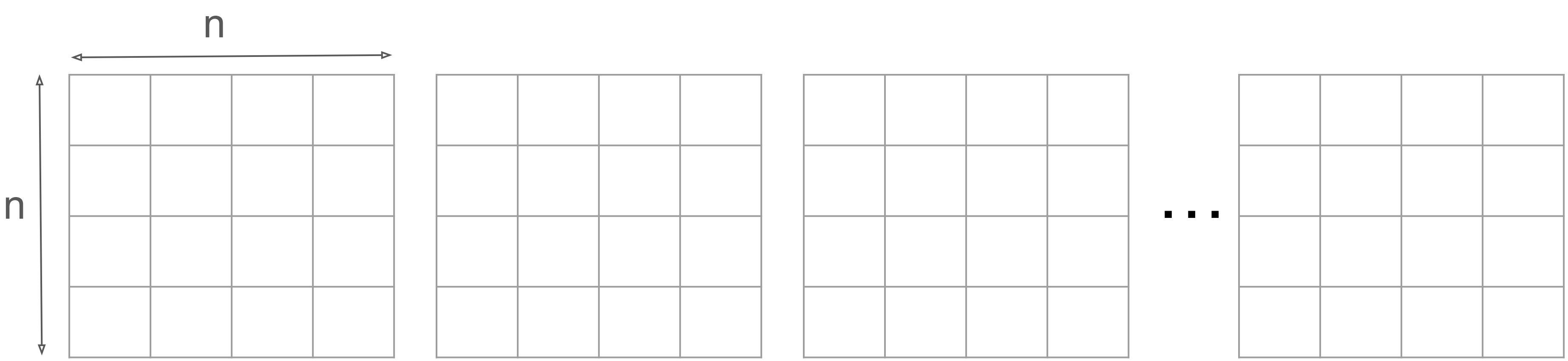
$$\mathbf{dist^{(i)}(s,v) = min_{(u,v)\in E}\{dist^{(i-1)}(s,u) + \ell(u,v)\}}$$

Base case: $\mathbf{dist^{(0)}(s,v)} = \begin{cases} 0 & \text{if } s = v \\ \infty & \text{otherwise} \end{cases}$

The only difference:

there are 3 "free" variables: s, v, and i ⇒ 3D DP table!

The DP table



$\mathbf{dist^{(0)}(s,t)}$ $\mathbf{dist^{(1)}(s,t)}$ $\mathbf{dist^{(2)}(s,t)}$ $\mathbf{dist^{(n-1)}(s,t)}$

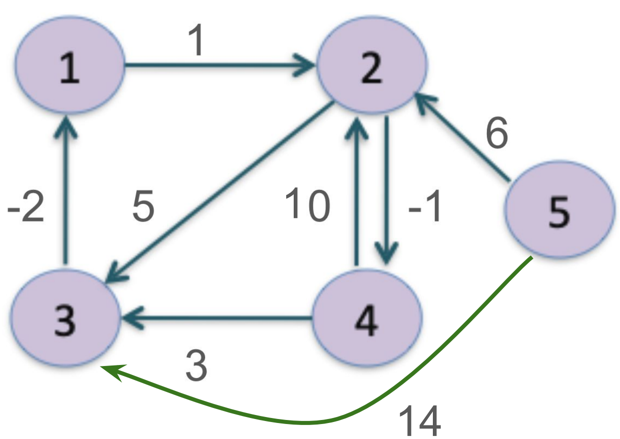
Running time:

But there's a cool trick that allows each cell to only look at **3** other cells…

Key idea: Impose an arbitrary ordering on the vertices and consider paths that **only use the first i vertices** in the ordering.

Arbitrary ordering of vertices: $v_1, v_2, \ldots, v_n$.

Key Definition: Let **$dist^{[i]}(s,t)$** be the shortest path from s to t with internal vertices ***only*** in $\{v_1,\ldots,v_i\}$, or ∞ if no such path.



What is…

$dist^{[0]}(5,3)$?

$dist^{[1]}(5,3)$?

$dist^{[2]}(5,3)$?

$dist^{[3]}(5,3)$?

$dist^{[4]}(5,3)$?

Recurrence

Arbitrary ordering of vertices: $v_1, v_2, \ldots\ v_n$.

Key Definition: Let $\mathbf{dist^{[i]}(s,t)}$ be the shortest path from s to t with internal vertices ***only*** in $\{v_1,\ldots,v_i\}$, or ∞ if no such path.

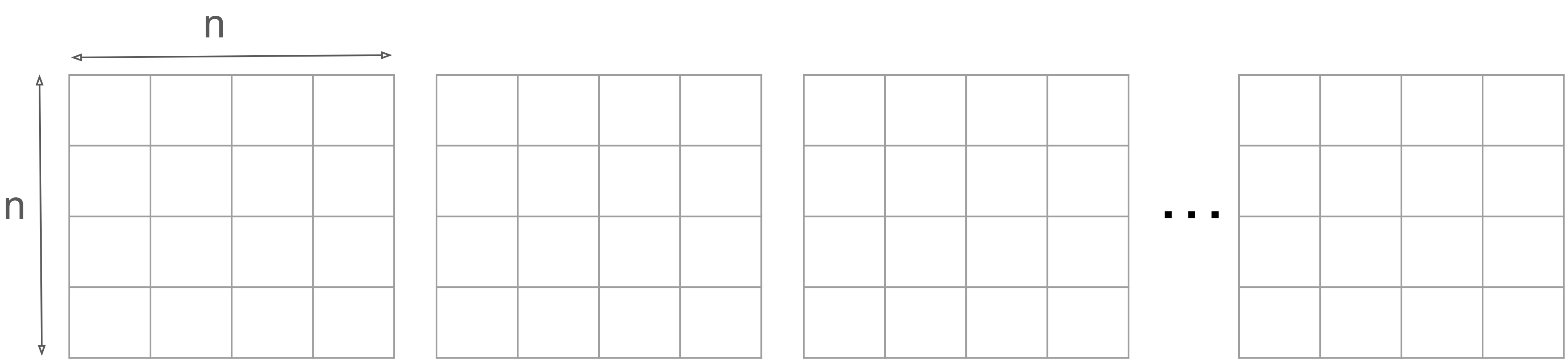
Base case: $\mathbf{dist^{[0]}(s,t)} = \begin{cases} 0 & \text{if } s = t \\ \ell(s,t) & \text{if } (s,t) \in E \\ \infty & \text{otherwise} \end{cases}$

$$\mathbf{dist^{[i]}(s,t) = \min\{\ \underbrace{\qquad\qquad}\ ,\ \underbrace{\qquad\qquad}\ \}}$$

Case 1 (**"Lose it!"**):
shortest st-path
doesn't contain v_i

Case 2 (**"Use it!"**):
shortest st-path
contains v_i

Let’s Follow the DP Recipe



$\mathbf{dist^{[0]}(s,t)}$ $\mathbf{dist^{[1]}(s,t)}$ $\mathbf{dist^{[2]}(s,t)}$ … $\mathbf{dist^{[n]}(s,t)}$

This is called the **Floyd-Warshall Algorithm** (1962)

Pseudocode for Floyd–Warshall

```
Algorithm APSP(G)
    table := 3D-array (1..n, 1..n, 0..n)
    // first two dimensions represent vertices v1,...,vn,
      third dimension represents restricting to the first i internal vertices
    for i = 1 to n:
        for j = 1 to n:
            table(i, j, 0) = ℓ(vi,vj)   // base case
    for i = 1 to n:
        for j = 1 to n:
            for k = 1 to n:
                table(i,j,k) = min{table(i,j,k-1), table(i,k,k-1) + table(k,j,k-1)}
    Return table(i, j, n) for all i,j
```

A question to ponder…

Why doesn't the trick of computing $\text{dist}^{[i]}(s,t)$ give a faster-than-BF algorithm for SSSP too?

Progress on APSP since Floyd-Warshall

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

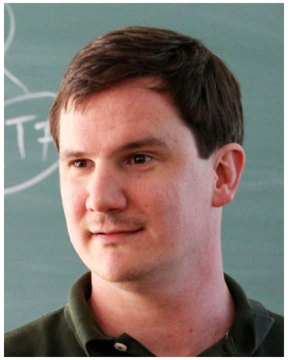
Conclusion: Maybe $O(n^{2.99})$ is impossible?

Maybe $O(n^{2.99})$ is impossible?

Either ALL of the following have $O(n^{<3})$ time algorithms or NONE of them do: (Virginia Vassilevska Williams, Ryan Williams, 2010)

1. APSP
2. Minimum Weight Triangle
3. Metricity
4. Minimum Cycle
5. Distance Product
6. Second Shortest Path
7. Replacement Paths
8. Negative Triangle Listing

…



A curious open problem:
“The Not Shortest Path problem”

Given a directed graph with positive edge weights and a pair s,t of vertices, is there a polynomial time algorithm to find a simple path from s to t that is NOT shortest?