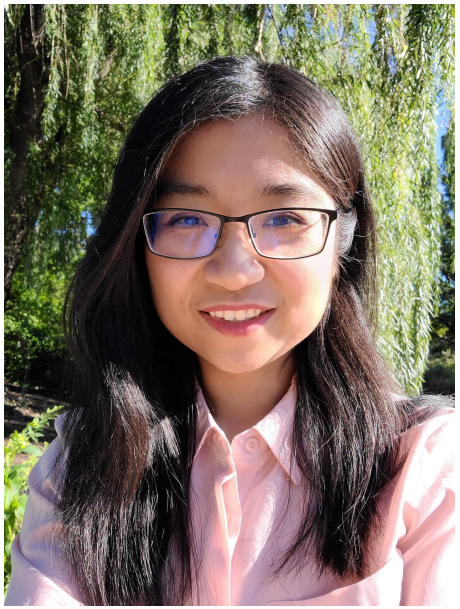
THE PREDICTED-UPDATES DYNAMIC MODEL: OFFLINE TO FULLY-DYNAMIC TRANSFORMATIONS

Dynamic Algorithms with Predictions

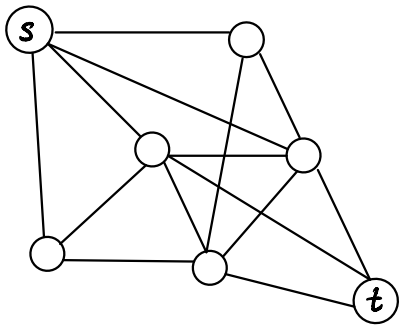


Quanquan C. Liu
Yale

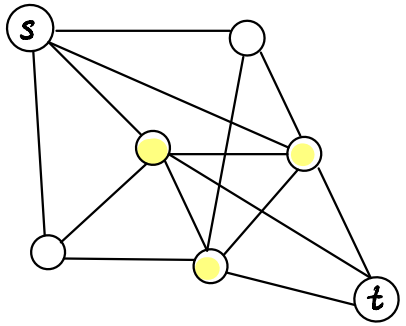
Vaidehi Srinivas
Northwestern University

CASE STUDY : TRICONNECTIVITY

updates
edge insertions and deletions



queries
"can I disconnect s and t by deleting 3 vertices from the graph?"



Best known fully-dynamic algorithm: $O(n^{2/3})$ update time
[Galil Italiano Sarnak '99]

? Is the barrier information? or computation?

A Information! Can solve offline dynamic problem in $O(T \text{ polylog } n) = \tilde{O}(T)$ time. [Peng Sandlund Sleator '17, Holm Rotenberg '20]

T = total # of updates

"$\tilde{O}(1)$ per update"

- Solve with slick divide-and-conquer algorithm
- Candidate for "algorithms with predictions"

ALGORITHMS WITH PREDICTIONS

Goal: effectively utilize extra untrusted information ("prediction") about an instance

e.g. "today's instance is likely similar to yesterday's instance"

Modeling challenge: What prediction should we ask for?

How should we measure prediction error?

Is it reasonable to get this in practice?

Ideal guarantee: [Lykouris Vassilvitskii '21] ⚠ not always possible!

① **Consistency**: if the prediction is good, do better than the worst case

② **Robustness**: if the prediction is bad, achieve a worst-case guarantee

a.k.a. "competitiveness"

③ **Graceful degradation**: smooth tradeoff between ① and ②

"Robustness" "smoothness"

Examples:

Ski-rental

[Kumar Purohit Svitkina '18]

pred: # of days able to ski

Max-Flow via Ford-Fulkerson

[Davies Moseley Vassilvitsii Wang '23]

pred.: solution to max-flow instance

"warm start"

THE (STANDARD) DYNAMIC MODEL

Goal: compute $f(G)$, for dynamic graph G

Initially: start with empty graph

On each day $t \leq T$: ("time horizon")

one edge inserted or deleted from G

algorithm outputs $f(G_t)$ — state of graph on G_t

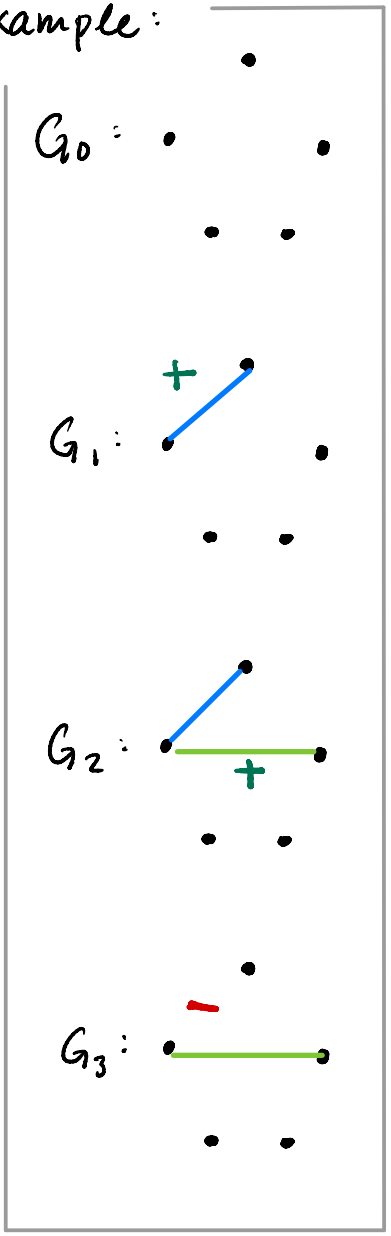
⚠ algorithm must always output correct solutions!

⚠ objective: minimize update time → runtime per update

either worst-case per update time

or amortized update time (average over all updates)

Example:



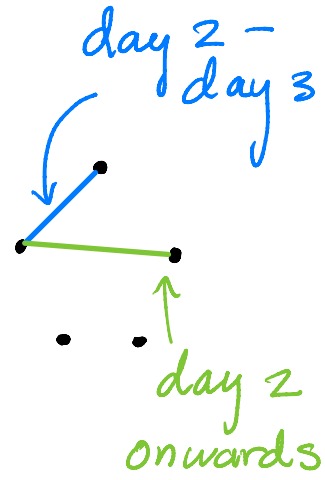
NEW! PREDICTED-UPDATES DYNAMIC MODEL

Goal: compute $f(G)$, for dynamic graph G

Initially: start with empty graph,
receive set of T predicted updates:
(update, predicted day) ← no feasibility requirement!

On each day $t \leq T$:
one edge inserted or deleted from G
algorithm outputs $f(G_t)$ ← true state of graph on G_t

Example prediction:



⚠ algorithm must always output correct solutions!

⚠ runtime of algorithm may depend on pred. error magnitude

$$\ell_1\text{-error(predictions)} = \sum_{\text{updates}} \left| \begin{matrix}\text{predicted}\\ \text{update}\\ \text{time}\end{matrix} - \begin{matrix}\text{true}\\ \text{update}\\ \text{time}\end{matrix} \right|.$$

OFFLINE TO FULLY-DYNAMIC TRANSFORMATION

Theorem [Liu S. '23] (Informal).

Given an offline divide-and-conquer* algorithm to compute $f(\cdot)$ that does $\tilde{O}(T)$ work,

and a fully-dynamic algorithm, B ("backstop"), with update time update_B,

we can design a predicted-updates dynamic algorithm that does total work

$$\tilde{O}\left(\min\left\{T + \|\text{prediction error}\|_1, \; T \cdot \text{update}_B\right\}\right)$$

- T: consistency
- $\|\text{prediction error}\|_1$: graceful degradation
- $T \cdot \text{update}_B$: robustness

* we require the divide-and-conquer to be "balanced," and satisfy a technical condition

INTERPRETATION

Work:

$$\tilde{O}\left(\min\left\{T + \|\text{prediction error}\|_1, \; T \cdot \text{update}_B\right\}\right)$$

consistency — graceful degradation — robustness

As long as average prediction error of updates $\leq \text{update}_B$, we beat the fully dynamic backstop.

Applications:

Triconnectivity	$\tilde{O}(n^{2/3})$	$\to \tilde{O}(1)$
k-Edge connectivity	$n^{o(1)}$	$\to \tilde{O}(1)$
Minimum Spanning Forest	$\tilde{O}(1)$	$\to \tilde{O}(1)$, but simpler!

Extends to incremental and decremental settings!

see also [van den Brand Forster Nazari Polak '24]

ALGOS W/ PREDS. : A LAY OF THE LAND

Warm starts (static)

\+ consistency AND robustness simultaneously

− Predictions typically large (encode problem solutions)

Online algorithms

− Inherent tradeoff between consistency and robustness

\+ Predictions typically small (encode future updates)

NEW!

Dynamic algorithms

\+ consistency AND robustness simultaneously

\+ Predictions are small (encode future updates)

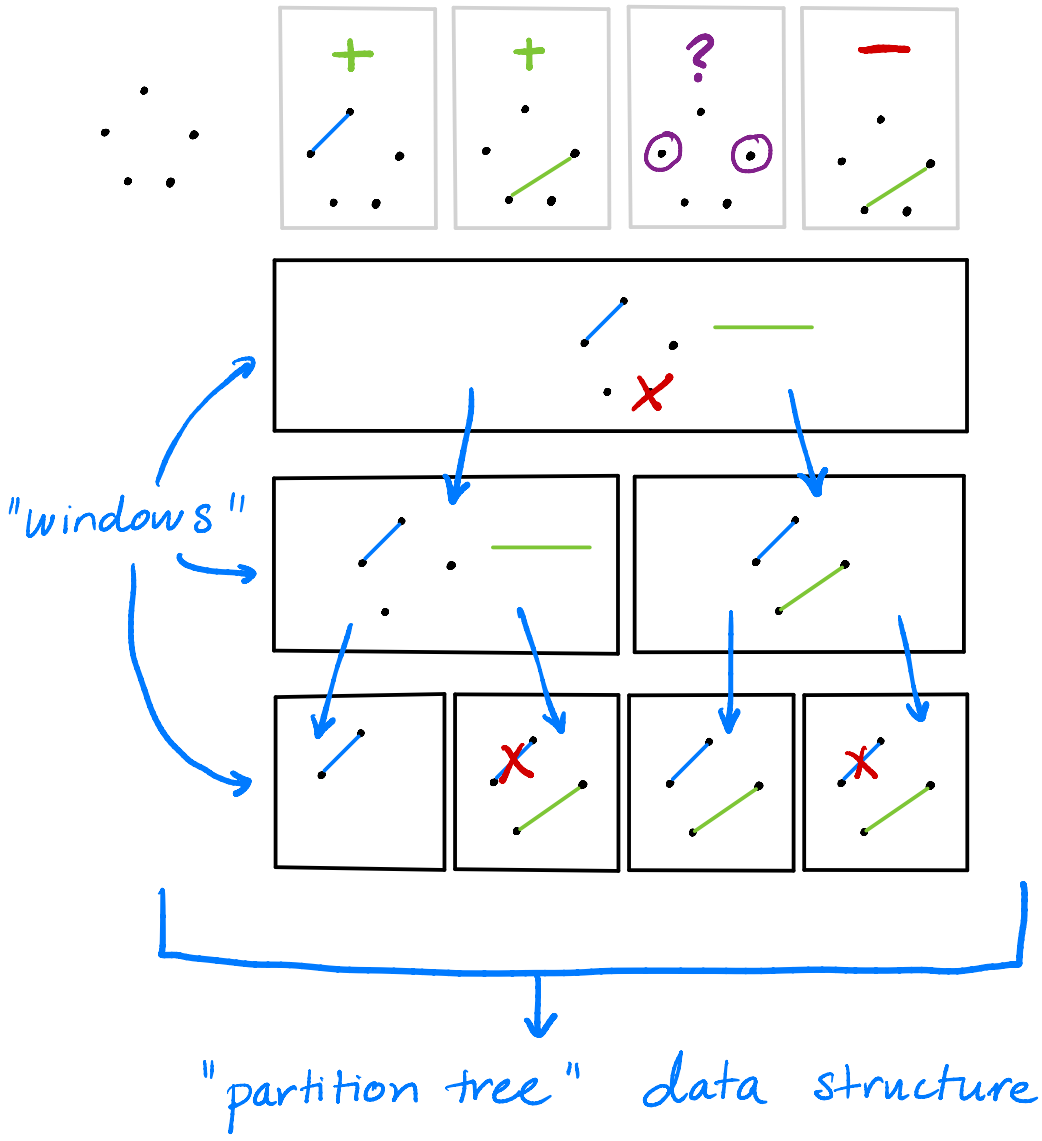
Examples:

Ski-rental
[Kumar Purohit Svitkina '18]
pred: # of days able to ski

Max-Flow via Ford-Fulkerson
[Davies Moseley Vassilvitsii Wang '23]
pred.: solution to max-flow instance
"warm start"

DIVIDE-AND-CONQUER OFFLINE DYNAMIC ALGORITHMS

Example: connectivity



Principles:

① contraction:
contract structures that are "permanent" for a window

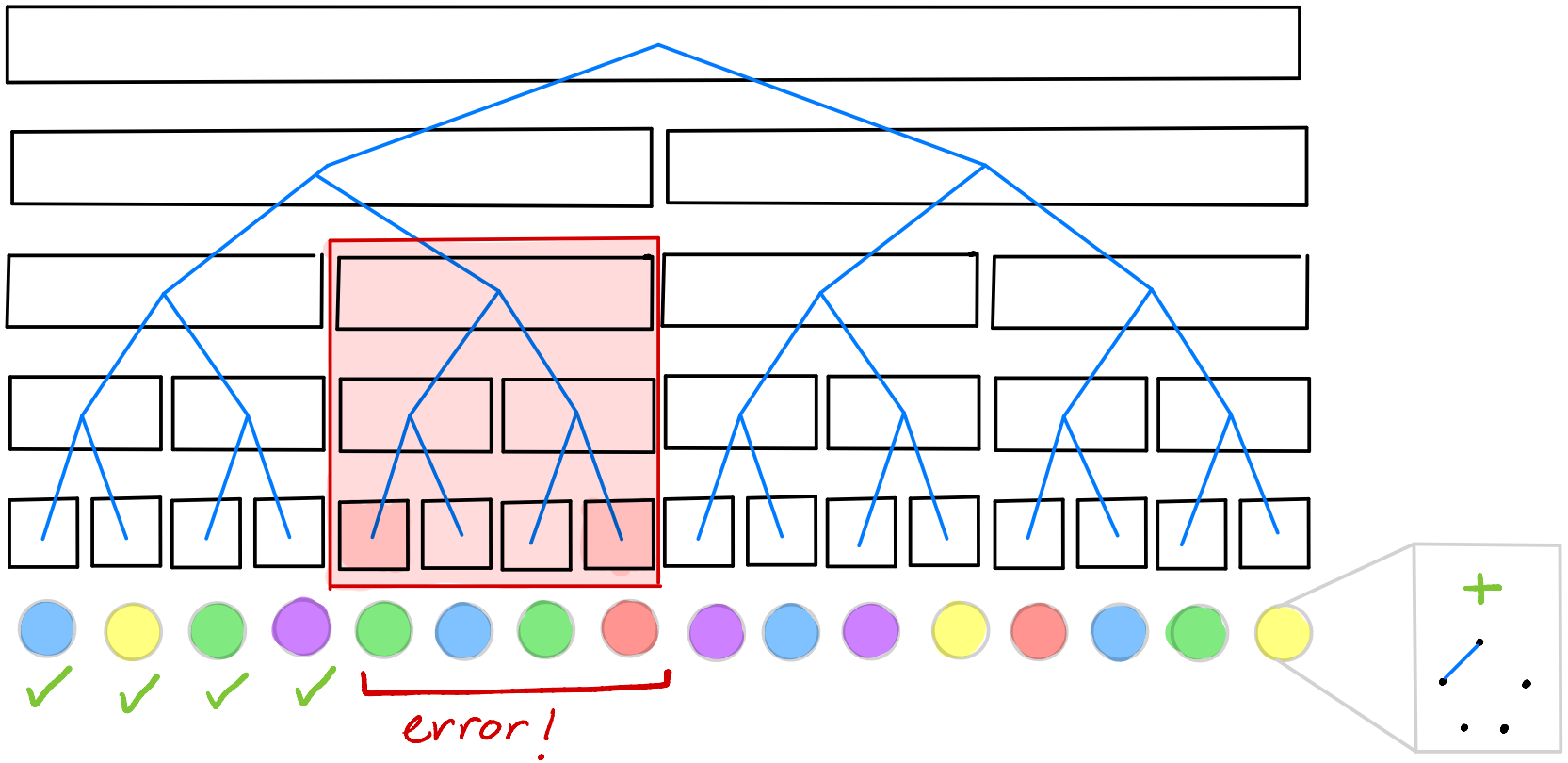
② reduction:
remove structures that are irrelevant for a window

build a sparsifier for your problem

Goal:
work done per window is linear in the # of events

A FIRST ATTEMPT

Run offline algorithm using predictions in $O(T \log T)$ work.



Observe realized updates:

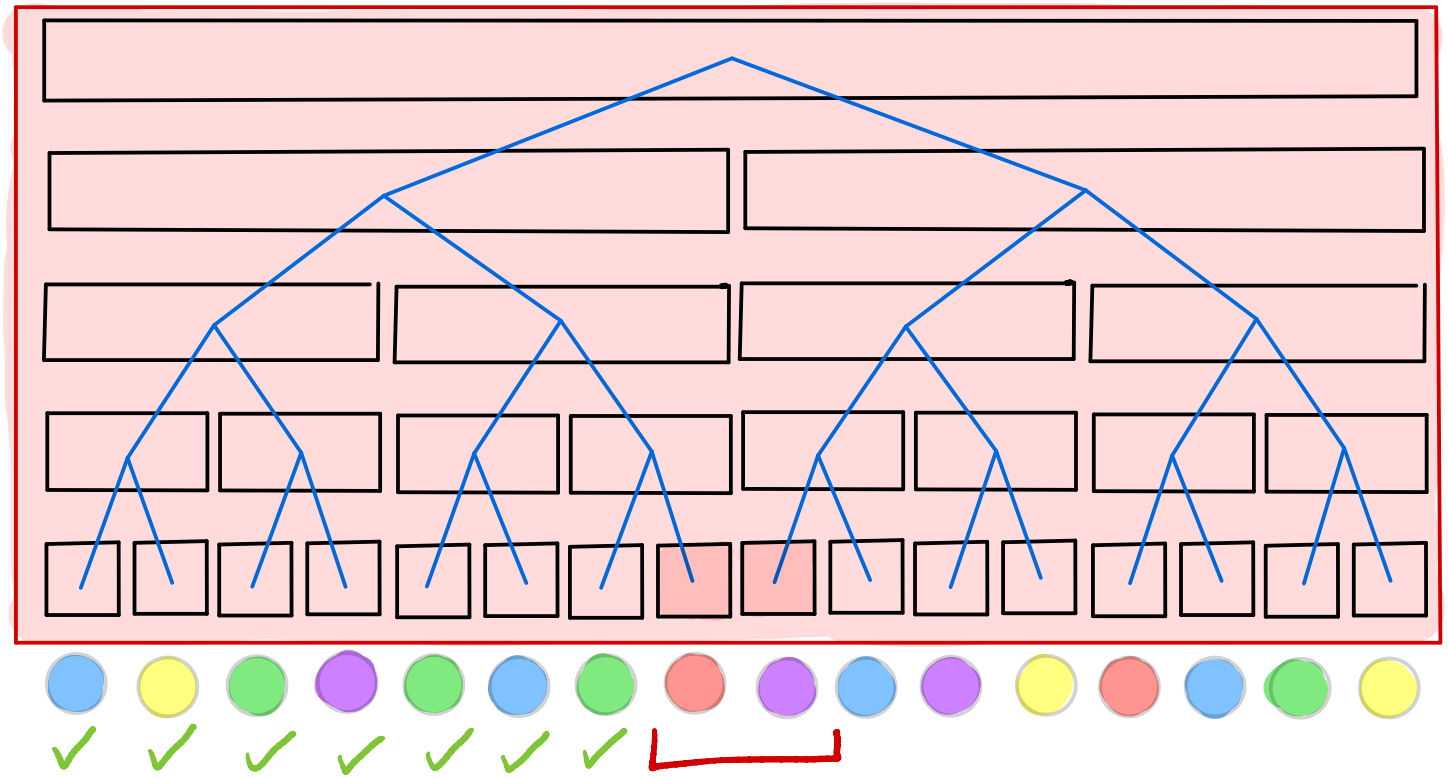
✗ Precomputed solutions incorrect

✓ Only need to fix small subtree!

Note: area of ▭ = work to recompute ▭

POTENTIAL PITFALL

Run offline algorithm using predictions in $O(T \log T)$ work.



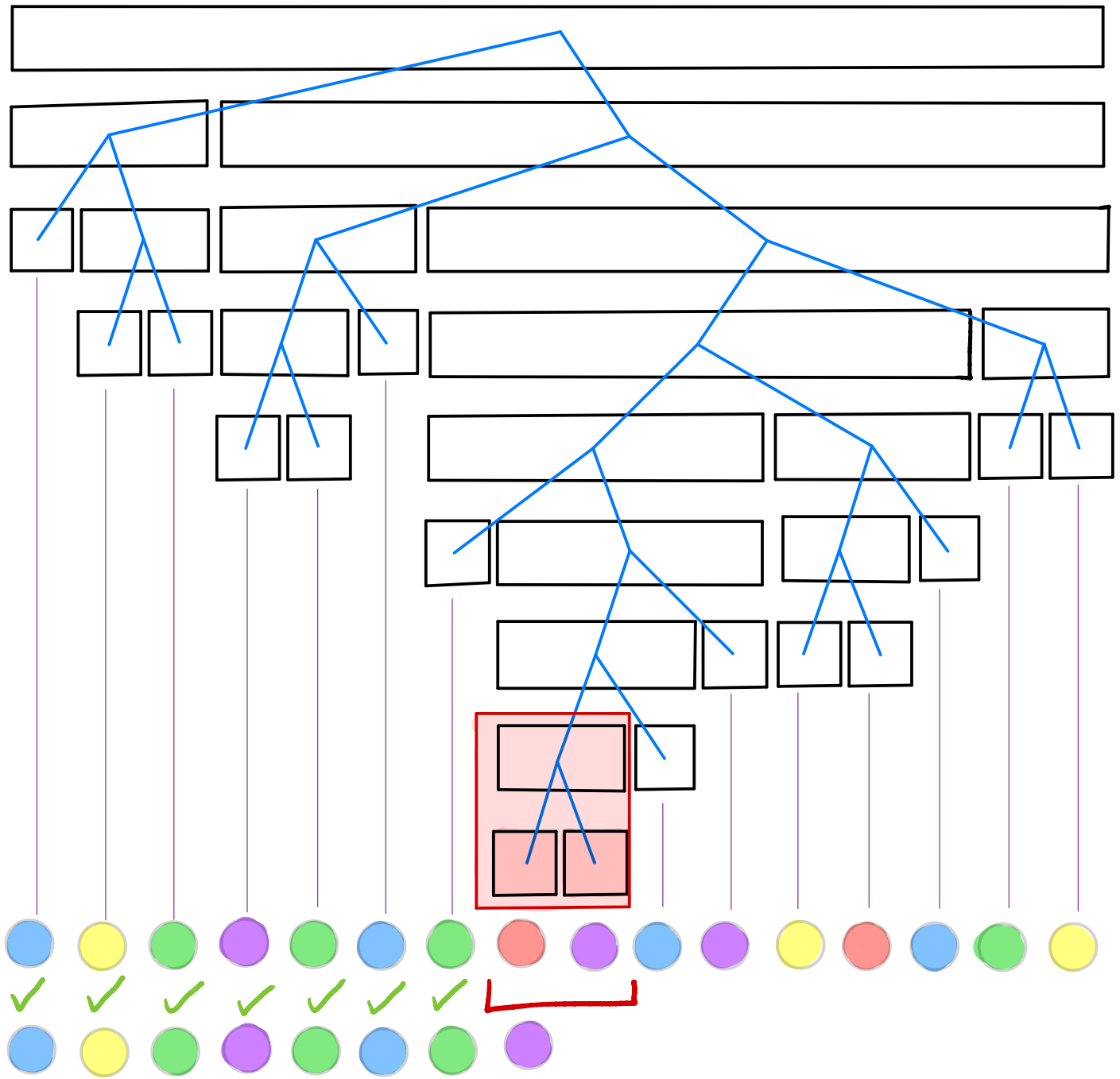
Observe realized updates:

area of □ = work to recompute □

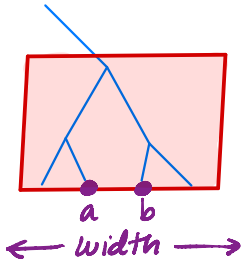
✗ Small error triggers large recomputation!

SOLUTION: RANDOM PARTITION-TREE

- divide-and-conquer still efficient!
- small errors trigger small recomputations!



PROPERTIES OF RANDOM PARTITION TREES



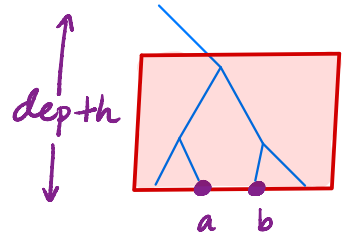
Property 1: Random trees preserve distance in expectation

$$\mathbb{E}\left[\begin{array}{c}\text{size of lowest common ancestor}\\ \text{of } a \text{ and } b\end{array}\right] = O\left(|b-a| \log T\right)$$

→ Randomized tree-embedding for the line metric
see e.g. [Fakcharoenphol Rao Talwar '04]

Property 2: Random trees are low depth.

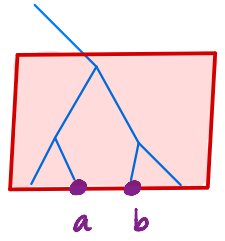
$$\mathbb{E}\left[\text{depth of random tree}\right] = O\left(\log T\right)$$



→ Divide-and-conquer is efficient

Implication: The expected work to recompute the smallest subtree containing a and b is

$$O\left(|b-a| \log^2 T\right).$$



→ work to fix error linear in magnitude of error!

WHAT DID WE SWEEP UNDER THE RUG?

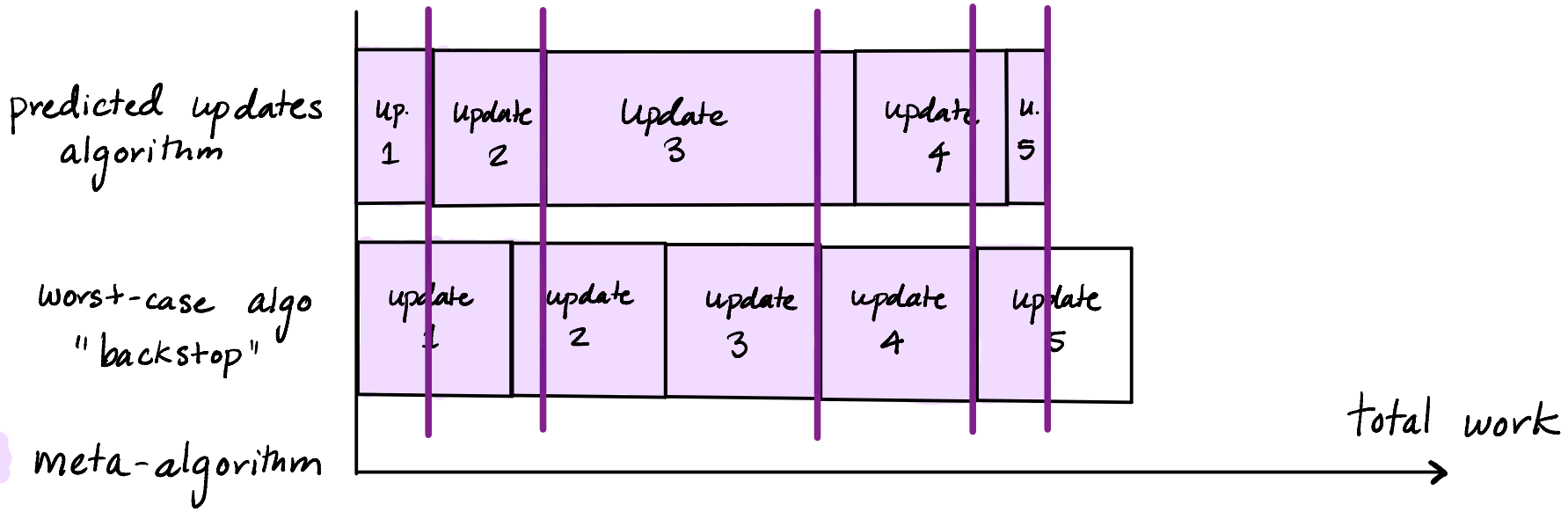
- ✓ Early update problem : (event occurs before predicted time)
  - ▷ handled by random partition tree
- ✓ Late update problem : (event occurs after prediction)
  - ▷ use doubling search, reduce to early update problem
- ⚠ each update can contribute $O(\log T)$ events over lifetime
- ⚠ predictions can assign many events to the same day
  (predictions can be infeasible)
- ✓ Overscheduling problem : Divide-and-conquer breaks if too many events on one day!
  - ▷ use online matching to make predictions feasible [Gupta Lewi '12]
  - ▷ relax data structure to handle $O(\log T)$ events per day

HEY, WHAT ABOUT ROBUSTNESS?

Work to handle T updates:

$$\tilde{O}\left(\min\left\{\underbrace{T + \|\text{prediction error}\|_1}_{\text{saw how to do this}}, \underbrace{T \cdot \text{update}_B}_{\text{know how to do this}}\right\}\right)$$

? How to get both simultaneously?



✓ also allows us to boost expectation bounds to high probability

TAKEAWAYS

Work to handle T updates:

$$\tilde{O}\left(\min\left\{T + \|\text{prediction error}\|_1, \; T \cdot \text{update}_B\right\}\right)$$

① For many dynamic problems, can take advantage of extra information "for free!"

- ▷ algorithms with predictions

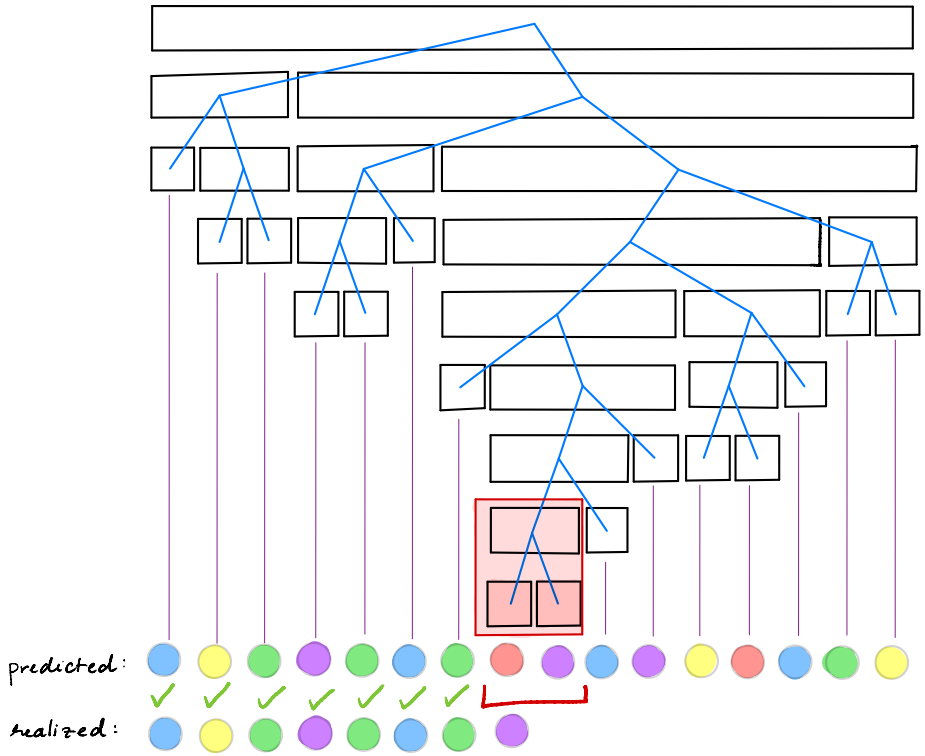
② New model smoothly interpolates between offline-dynamic and fully-dynamic settings

- ▷ "beyond-worst-case" models

③ Random partition-tree is simple and powerful

- ▷ new theoretical insight into lifting offline and partially-dynamic algorithms to fully-dynamic setting

FUTURE DIRECTIONS



- Better dependence on error?
  perhaps ℓ_0?
- Make deterministic or otherwise robust to adaptive adversaries
- Worst-case per update vs. amortized bounds
- Better dynamic subroutines for static problems

Dynamic algorithms with predictions is taking off!

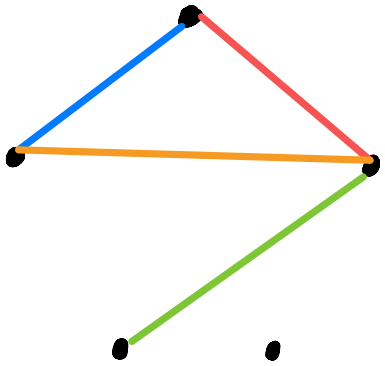
[van den Brand Forster Nazari Polak '24], [Henzinger Saha Seybold Ye '23],
[Agarwal Balkanski '23]

THANKS!

Vaidehi@u.northwestern.edu

EXTENSION TO PARTIALLY-DYNAMIC SETTINGS

Stronger assumption: incremental algorithm (vs. offline dynamic)



↳ dynamic algorithm that only handles edge insertions

worst-case per update time

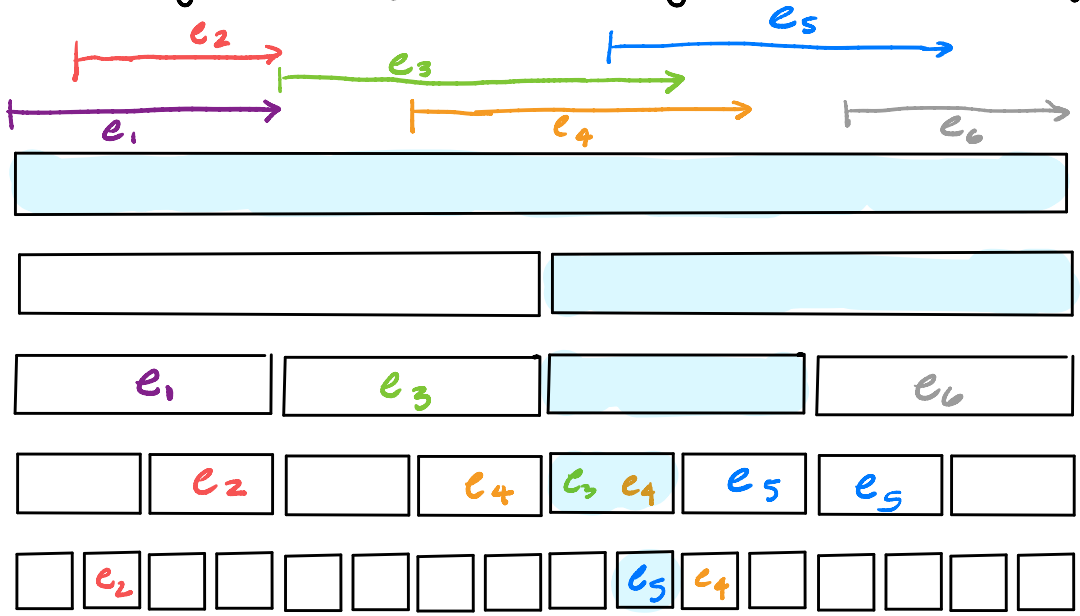
Stronger guarantee: only need predictions of deletion times

elements arrive arbitrarily, inserted element announces predicted deletion time

Same framework extends to this setting!

INCREMENTAL TO FULLY-DYNAMIC TRANSFORMATION

First, design offline dynamic algorithm using incremental algorithm A



At each window, insert "permanent edges" of that window

Total work: $\tilde{O}(T \cdot \text{update}_A)$

⚠ Apply framework to "lift" to fully-dynamic setting

✓ this particular offline solution can be run "just-in-time"

$\Rightarrow$ only need predicted deletion times

Concurrent work of [van den Brand Forster Nazari Polak '23] achieves deterministic worst-case per update bound!