The Saga of Dynamic Lower Bounds around the Logarithmic Barrier

Mihai Pătraşcu

MIT

Max Planck Institut, June 2005

Definitions

- the cell-probe model
- cells of $O(\lg n)$ bits
  - also, the bit-probe model: cells of one bit
- t_u = update time
- t_q = query time
- n = number of bits/cells in the problem instance

Definitions

- the cell-probe model
- cells of $O(\lg n)$ bits
  - also, the bit-probe model: cells of one bit
- t_u = update time
- t_q = query time
- n = number of bits/cells in the problem instance

Definitions

- the cell-probe model
- cells of $O(\lg n)$ bits
  - also, the bit-probe model: cells of one bit
- t_u = update time
- t_q = query time
- n = number of bits/cells in the problem instance

Definitions

- the cell-probe model
- cells of $O(\lg n)$ bits
  - also, the bit-probe model: cells of one bit
- t_u = update time
- t_q = query time
- n = number of bits/cells in the problem instance

The partial-sums problem

Maintain an array $A[1 \,..\, n]$ subject to:

UPDATE(k, u) modify $A[k] \leftarrow u$.

SUM(k) return the partial sum $\sum_{i=1}^{k} A[i]$.

VERIFY-SUM(k, σ) verify whether $\sum_{i=1}^{k} A[i] = \sigma$.
(decision version of SUM)

SELECT(σ) return i : SUM$(i) \leq \sigma <$ SUM$(i+1)$.

The partial-sums problem

Maintain an array $A[1\,..\,n]$ subject to:

UPDATE(k, u) modify $A[k] \leftarrow u$.

SUM(k) return the partial sum $\sum_{i=1}^{k} A[i]$.

VERIFY-SUM(k, σ) verify whether $\sum_{i=1}^{k} A[i] = \sigma$.
(decision version of SUM)

SELECT(σ) return i : SUM$(i) \le \sigma <$ SUM$(i+1)$.

The partial-sums problem

Maintain an array $A[1\,..\,n]$ subject to:

- UPDATE(k, u) modify $A[k] \leftarrow u$.
- SUM(k) return the partial sum $\sum_{i=1}^{k} A[i]$.
- VERIFY-SUM(k, σ) verify whether $\sum_{i=1}^{k} A[i] = \sigma$. (decision version of SUM)
- SELECT(σ) return i : SUM$(i) \le \sigma <$ SUM$(i+1)$.

The partial-sums problem

Maintain an array $A[1\,..\,n]$ subject to:

UPDATE(k, u) modify $A[k] \leftarrow u$.

SUM(k) return the partial sum $\sum_{i=1}^{k} A[i]$.

VERIFY-SUM(k, σ) verify whether $\sum_{i=1}^{k} A[i] = \sigma$.
(decision version of SUM)

SELECT(σ) return i : SUM$(i) \leq \sigma <$ SUM$(i+1)$.

The partial-sums problem

Maintain an array $A[1\,.\,.\,n]$ subject to:

$\textsc{update}(k, u)$	modify $A[k] \leftarrow u$.
$\textsc{sum}(k)$	return the partial sum $\sum_{i=1}^{k} A[i]$.
$\textsc{verify-sum}(k, \sigma)$	verify whether $\sum_{i=1}^{k} A[i] = \sigma$. (decision version of $\textsc{sum}$)
$\textsc{select}(\sigma)$	return $i : \textsc{sum}(i) \leq \sigma < \textsc{sum}(i+1)$.

Chapters of the saga

STOC'89 Fredman and Saks: chronogram technique
Lower Bound: $\Omega(\frac{\lg n}{\lg\lg n})$ Problem: SUM

ICALP'98 Husfeldt and Rauhe: nondeterminism
Problem: VERIFY-SUM

FOCS'98 Alstrup, Husfeldt and Rauhe: alternative histories
Problem: marked ancestor, range queries

SODA'04 M.P. and Demaine: time-tree technique
Lower Bound: $\Omega(\lg n)$ Problem: SUM

STOC'04 M.P. and Demaine: nondeterminism
Problem: VERIFY-SUM, dynamic connectivity

ICALP'05 M.P. and Corina Pătraşcu: epoch-based proof

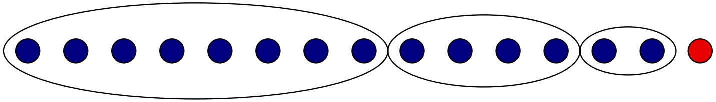
Then great $X\rho o\nu o\varsigma$ fashioned from divine $A\iota\theta\eta\rho$ a bright white egg.

Orphic Rhapsodies 66

Updates and epochs

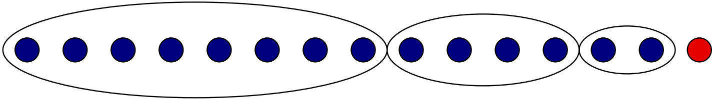
- run n updates, then one random query
  argue this query must be slow, if all updates are fast
- divide updates into exponentially growing epochs
  e.g. epoch i has $(99t_u)^i$ updates
- next slides: $(\forall)i$, the query needs to read a cell written in epoch i with $\Omega(1)$ probability
- lower bound: $t_q = \Omega(\lg_{t_u} n) \Rightarrow$
  $t_q \lg t_u = \Omega(\lg n) \Rightarrow \max\{t_u, t_q\} = \Omega(\frac{\lg n}{\lg \lg n})$.

Updates and epochs



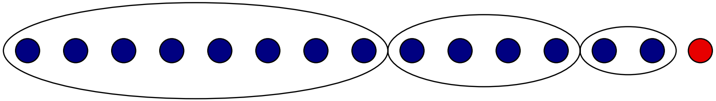
- run n updates, then one random query
  argue this query must be slow, if all updates are fast
- divide updates into exponentially growing epochs
  e.g. epoch i has $(99t_u)^i$ updates
- next slides: $(\forall)i$, the query needs to read a cell written in epoch i with $\Omega(1)$ probability
- lower bound: $t_q = \Omega(\lg_{t_u} n) \Rightarrow$
  $t_q \lg t_u = \Omega(\lg n) \Rightarrow \max\{t_u, t_q\} = \Omega(\frac{\lg n}{\lg \lg n})$.

Updates and epochs



- run n updates, then one random query
  argue this query must be slow, if all updates are fast
- divide updates into exponentially growing epochs
  e.g. epoch i has $(99t_u)^i$ updates
- next slides: $(\forall)i$, the query needs to read a cell written in epoch i with $\Omega(1)$ probability
- lower bound: $t_q = \Omega(\lg_{t_u} n) \Rightarrow$
  $t_q \lg t_u = \Omega(\lg n) \Rightarrow \max\{t_u, t_q\} = \Omega(\frac{\lg n}{\lg \lg n})$.

Updates and epochs



- run n updates, then one random query

  argue this query must be slow, if all updates are fast

- divide updates into exponentially growing epochs

  e.g. epoch i has $(99t_u)^i$ updates

- next slides: $(\forall)i$, the query needs to read a cell written in epoch i with $\Omega(1)$ probability

- lower bound: $t_q = \Omega(\lg_{t_u} n) \Rightarrow$

  $t_q \lg t_u = \Omega(\lg n) \Rightarrow \max\{t_u, t_q\} = \Omega(\frac{\lg n}{\lg \lg n})$.

The hard instance

A[1] A[n]

UPDATE(k, u) modify $A[k] \leftarrow u$.

- u is uniformly random
- k's in each epoch are uniformly spread

SUM(k) return $\sum_{i=1}^{k} A[i]$.

- k is uniformly random

"A query cares about a random prefix sum in every epoch."

The hard instance

A[1] A[n]

UPDATE(k, u) modify $A[k] \leftarrow u$.

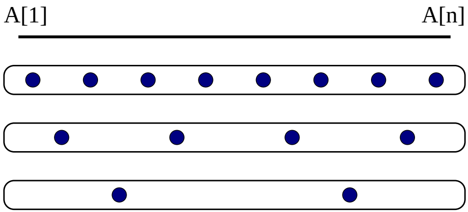
- u is uniformly random
- k's in each epoch are uniformly spread

SUM(k) return $\sum_{i=1}^{k} A[i]$.

- k is uniformly random

"A query cares about a random prefix sum in every epoch."

The hard instance



$\textsc{Update}(k, u)$ modify $A[k] \leftarrow u$.

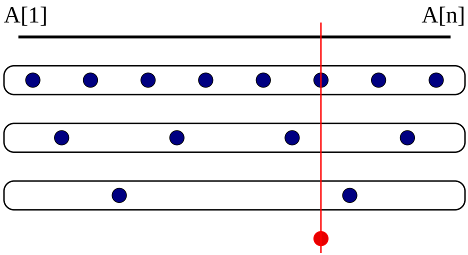
- u is uniformly random
- k's in each epoch are uniformly spread

$\textsc{Sum}(k)$ return $\sum_{i=1}^{k} A[i]$.

- k is uniformly random

"A query cares about a random prefix sum in every epoch."

The hard instance



UPDATE(k, u) modify $A[k] \leftarrow u$.

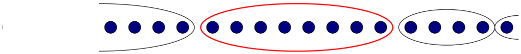
- u is uniformly random
- k's in each epoch are uniformly spread

SUM(k) return $\sum_{i=1}^{k} A[i]$.

- k is uniformly random

"A query cares about a random prefix sum in every epoch."

One cell per epoch. . .



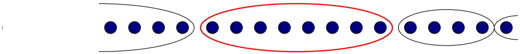
Look at some epoch i:

- cells from the past (epoch $> i$) can't contain information about updates in epoch i
- total # of cells written in the future (epoch $< i$):

$$t_u \cdot \left((99t_u)^{i-1} + (99t_u)^{i-2} + \dots \right) < t_u \cdot 2(99t_u)^{i-1}$$

- there are $(99t_u)^i$ random prefix sums in epoch i
  $\Rightarrow$ "usually", "most" prefix sums are not fixed by future cells
- a random query cares about a random prefix sum;
  if that's not fixed, need to a cell from epoch i

One cell per epoch. . .



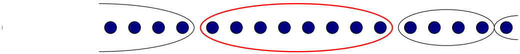
Look at some epoch i:

- cells from the past (epoch $> i$) can't contain information about updates in epoch i
- total # of cells written in the future (epoch $< i$):

$$t_u \cdot \left((99t_u)^{i-1} + (99t_u)^{i-2} + \dots \right) < t_u \cdot 2(99t_u)^{i-1}$$

- there are $(99t_u)^i$ random prefix sums in epoch i
$\Rightarrow$ "usually", "most" prefix sums are not fixed by future cells
- a random query cares about a random prefix sum;
if that's not fixed, need to a cell from epoch i

One cell per epoch. . .



Look at some epoch i:

- cells from the past (epoch $> i$) can't contain information about updates in epoch i

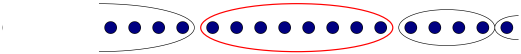
- total # of cells written in the future (epoch $< i$):

$$t_u \cdot \Big((99t_u)^{i-1} + (99t_u)^{i-2} + \dots \Big) < t_u \cdot 2(99t_u)^{i-1}$$

- there are $(99t_u)^i$ random prefix sums in epoch i
  $\Rightarrow$ "usually", "most" prefix sums are not fixed by future cells

- a random query cares about a random prefix sum;
  if that's not fixed, need to a cell from epoch i

One cell per epoch...



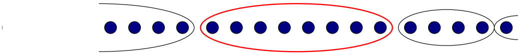
Look at some epoch i:

- cells from the past (epoch $> i$) can't contain information about updates in epoch i
- total # of cells written in the future (epoch $< i$):

$$t_u \cdot \Big((99t_u)^{i-1} + (99t_u)^{i-2} + \dots \Big) < t_u \cdot 2(99t_u)^{i-1}$$

- there are $(99t_u)^i$ random prefix sums in epoch i
  $\Rightarrow$ "usually", "most" prefix sums are not fixed by future cells
- a random query cares about a random prefix sum;
  if that's not fixed, need to a cell from epoch i

One cell per epoch...



Look at some epoch i:

- cells from the past (epoch $> i$) can't contain information about updates in epoch i
- total # of cells written in the future (epoch $< i$):

$$t_u \cdot \Big((99t_u)^{i-1} + (99t_u)^{i-2} + \dots\Big) < t_u \cdot 2(99t_u)^{i-1}$$

- there are $(99t_u)^i$ random prefix sums in epoch i
  $\Rightarrow$ "usually", "most" prefix sums are not fixed by future cells
- a random query cares about a random prefix sum;
  if that's not fixed, need to a cell from epoch i

Trouble:

- the algorithm probes cell X
- say the cell was last written before epoch i
- then it was not written in epoch i
- that's information about epoch i

Setup for Epoch Analysis

- static problem on M elements
- $o(M)$ help bits – cells from future epochs
- cell probe either returns a value, or "unavailable"
- goal: lower bound for available cell probes

Observation for lower bound of 1:

If a query makes only "unavailable" probes on two problem instances, it has the same answer on both.

Shadows of the past

Trouble:

- the algorithm probes cell X
- say the cell was last written before epoch i
- then it was not written in epoch i
- that's information about epoch i

Setup for Epoch Analysis

- static problem on M elements
- $o(M)$ help bits – cells from future epochs
- cell probe either returns a value, or "unavailable"
- goal: lower bound for available cell probes

Observation for lower bound of 1:

If a query makes only "unavailable" probes on two problem instances, it has the same answer on both.

Shadows of the past

Trouble:

- the algorithm probes cell X
- say the cell was last written before epoch i
- then it was not written in epoch i
- that's information about epoch i

Setup for Epoch Analysis

- static problem on M elements
- $o(M)$ help bits – cells from future epochs
- cell probe either returns a value, or "unavailable"
- goal: lower bound for available cell probes

Observation for lower bound of 1:

If a query makes only "unavailable" probes on two problem instances, it has the same answer on both.

Shadows of the past

Trouble:

- the algorithm probes cell X
- say the cell was last written before epoch i
- then it was not written in epoch i
- that's information about epoch i

Setup for Epoch Analysis

- static problem on M elements
- $o(M)$ help bits – cells from future epochs
- cell probe either returns a value, or "unavailable"
- goal: lower bound for available cell probes

Observation for lower bound of 1:

If a query makes only "unavailable" probes on two problem instances, it has the same answer on both.

Shadows of the past

Trouble:
- the algorithm probes cell X
- say the cell was last written before epoch i
- then it was not written in epoch i
- that's information about epoch i

Setup for Epoch Analysis

- static problem on M elements
- $o(M)$ help bits – cells from future epochs
- cell probe either returns a value, or "unavailable"
- goal: lower bound for available cell probes

Observation for lower bound of 1:

If a query makes only "unavailable" probes on two problem instances, it has the same answer on both.

Shadows of the past

Trouble:
- the algorithm probes cell X
- say the cell was last written before epoch i
- then it was not written in epoch i
- that's information about epoch i

Setup for Epoch Analysis

- static problem on M elements
- $o(M)$ help bits – cells from future epochs
- cell probe either returns a value, or “unavailable”
- goal: lower bound for available cell probes

Observation for lower bound of 1:

If a query makes only “unavailable” probes on two problem instances, it has the same answer on both.

Deeper thoughts

Queries

- lower bound holds for nondeterministic query algorithms
- nondeterministically, VERIFY-SUM equivalent to SUM

Updates

- we just bounded the number of cell writes
- the update algorithm can have perfect information
  (“hard to maintain dynamic proofs”)
- FIX: later in this talk

Hardness

- every epoch must influence query with $\Omega(1)$ probability
- FIX: alternative histories

Deeper thoughts

Queries

- lower bound holds for nondeterministic query algorithms
- nondeterministically, VERIFY-SUM equivalent to SUM

Updates

- we just bounded the number of cell writes
- the update algorithm can have perfect information
  ("hard to maintain dynamic proofs")
- FIX: later in this talk

Hardness

- every epoch must influence query with $\Omega(1)$ probability
- FIX: alternative histories

Deeper thoughts

Queries

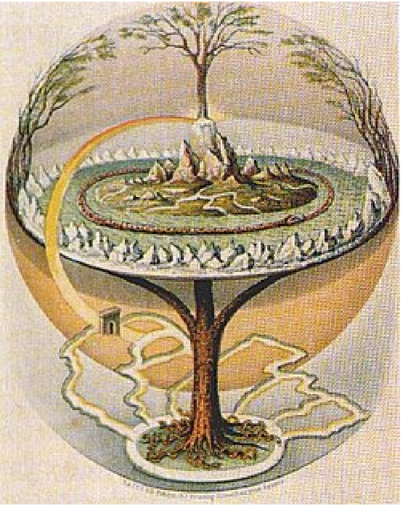
- lower bound holds for nondeterministic query algorithms
- nondeterministically, VERIFY-SUM equivalent to SUM

Updates

- we just bounded the number of cell writes
- the update algorithm can have perfect information
  (“hard to maintain dynamic proofs”)
- FIX: later in this talk

Hardness

- every epoch must influence query with $\Omega(1)$ probability
- FIX: alternative histories



Ask veit ek standa.
Heitir Yggdrasill.
(*Völuspá*)

Constructing the time tree

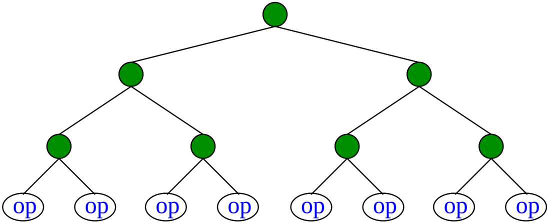
Each operation chosen randomly from:

- UPDATE(random k, random u)
- SUM(random k)

op op op op op op op op

Constructing the time tree

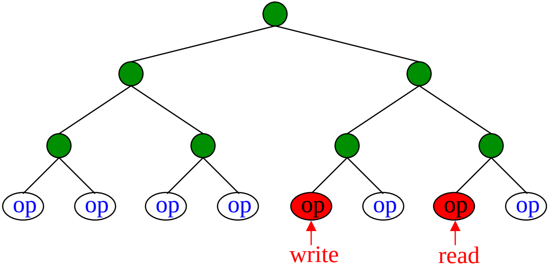
Build a balanced tree with operations in the leaves (considered in chronological order)



Constructing the time tree

A cell probe is characterized by:

- time of last write to the cell
- time when cell is read



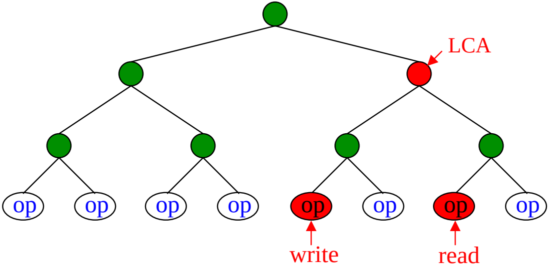
Constructing the time tree

Cell probe is “associated” with LCA

Prove lower bounds for probes associated with each node

Then sum up

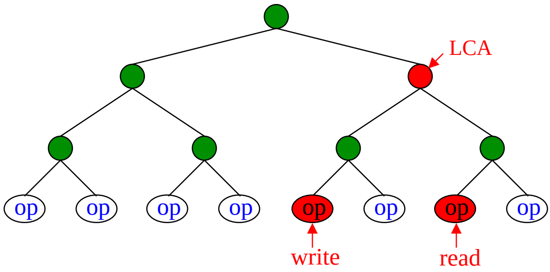
- not double counting any cell probe
- summing works for average case lower bounds



Constructing the time tree

Cell probe is "associated" with LCA
Prove lower bounds for probes associated with each node
Then sum up

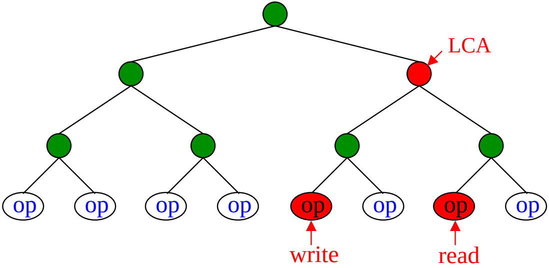
- not double counting any cell probe
- summing works for average case lower bounds



Constructing the time tree

Cell probe is "associated" with LCA
Prove lower bounds for probes associated with each node
Then sum up

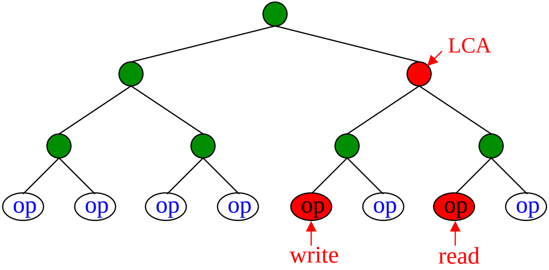
- not double counting any cell probe
- summing works for average case lower bounds



Constructing the time tree

Cell probe is “associated” with LCA
Prove lower bounds for probes associated with each node
Then sum up

- not double counting any cell probe
- summing works for average case lower bounds

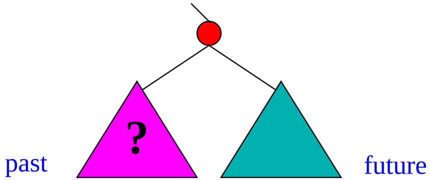


How much information do queries in the right subtree need to learn about the updates in the left subtree?

- simple analysis of a static problem

Almost true: all this information comes from cells written in left subtree, read in right subtree

Give an encoding from which we can simulate the data structure in the right subtree, not knowing the left subtree. This must include all information learned.



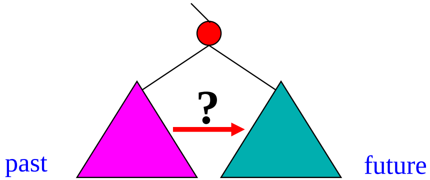
Lower bounds for the information transfer

How much information do queries in the right subtree need to learn about the updates in the left subtree?

- simple analysis of a static problem

Almost true: all this information comes from cells written in left subtree, read in right subtree

Give an encoding from which we can simulate the data structure in the right subtree, not knowing the left subtree. This must include all information learned.



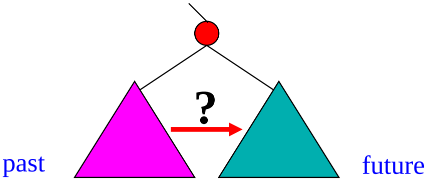
Lower bounds for the information transfer

How much information do queries in the right subtree need to learn about the updates in the left subtree?

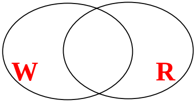
- simple analysis of a static problem

Almost true: all this information comes from cells written in left subtree, read in right subtree

Give an encoding from which we can simulate the data structure in the right subtree, not knowing the left subtree. This must include all information learned.



The Return of the Shadows



$W =$ cells written in left subtree

$R =$ cells read in right subtree
by one accepting thread per query

Encoding contains:

- complete information for $W \cap R$
- separator for $W \setminus R$ and $R \setminus W$ (Bloomier filter)

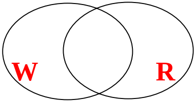
This suffices for correct simulation!
A cell probe can come from:

$W \cap R \Rightarrow$ have complete information

R's side of separator $\Rightarrow$ not in $W \Rightarrow$ value from the past

W's side of separator $\Rightarrow$ kill simulation of this thread

The Return of the Shadows



$W =$ cells written in left subtree

$R =$ cells read in right subtree
by one accepting thread per query

Encoding contains:

- complete information for $W \cap R$
- separator for $W \setminus R$ and $R \setminus W$ (Bloomier filter)

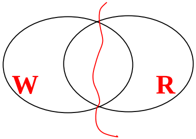
This suffices for correct simulation!
A cell probe can come from:

$W \cap R \Rightarrow$ have complete information

R's side of separator $\Rightarrow$ not in $W \Rightarrow$ value from the past

W's side of separator $\Rightarrow$ kill simulation of this thread

The Return of the Shadows



$W =$ cells written in left subtree

$R =$ cells read in right subtree
by one accepting thread per query

Encoding contains:

- complete information for $W \cap R$
- separator for $W \setminus R$ and $R \setminus W$ (Bloomier filter)

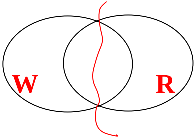
This suffices for correct simulation!

A cell probe can come from:

$W \cap R \Rightarrow$ have complete information

R's side of separator $\Rightarrow$ not in $W \Rightarrow$ value from the past

W's side of separator $\Rightarrow$ kill simulation of this thread



$W =$ cells written in left subtree

$R =$ cells read in right subtree
by one accepting thread per query

Encoding contains:

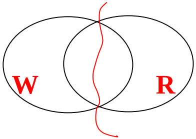
- complete information for $W \cap R$
- separator for $W \setminus R$ and $R \setminus W$ (Bloomier filter)

This suffices for correct simulation!
A cell probe can come from:

$W \cap R \Rightarrow$ have complete information

R's side of separator $\Rightarrow$ not in $W \Rightarrow$ value from the past

W's side of separator $\Rightarrow$ kill simulation of this thread



$W =$ cells written in left subtree
$R =$ cells read in right subtree
by one accepting thread per query

Encoding contains:

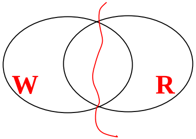
- complete information for $W \cap R$
- separator for $W \setminus R$ and $R \setminus W$ (Bloomier filter)

This suffices for correct simulation!
A cell probe can come from:

$W \cap R \Rightarrow$ have complete information

R's side of separator $\Rightarrow$ not in $W \Rightarrow$ value from the past

W's side of separator $\Rightarrow$ kill simulation of this thread



$W =$ cells written in left subtree

$R =$ cells read in right subtree

by one accepting thread per query

Encoding contains:

- complete information for $W \cap R$
- separator for $W \setminus R$ and $R \setminus W$ (Bloomier filter)

This suffices for correct simulation!

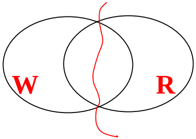
A cell probe can come from:

$W \cap R \Rightarrow$ have complete information

R's side of separator $\Rightarrow$ not in $W \Rightarrow$ value from the past

W's side of separator $\Rightarrow$ kill simulation of this thread

The Return of the Shadows



$W =$ cells written in left subtree

$R =$ cells read in right subtree

by one accepting thread per query

Encoding contains:

- complete information for $W \cap R$
- separator for $W \setminus R$ and $R \setminus W$ (Bloomier filter)

This suffices for correct simulation!

A cell probe can come from:

$W \cap R$ $\Rightarrow$ have complete information

R's side of separator $\Rightarrow$ not in W $\Rightarrow$ value from the past

W's side of separator $\Rightarrow$ kill simulation of this thread

Strategy: an epoch-based proof,

. . . but identify the real

Strategy: an epoch-based proof,

. . . but identify the real

Speaking of mythology. . .

What's wrong with the time tree?

Not combinatorial enough.

Intuitively, many problems are hard but do not hide a large encoding problem.

Concrete examples:

- the bit-probe model
- the separator (Bloomier filter) is too large to analyze the full tradeoff for VERIFY-SUM

What's wrong with the time tree?

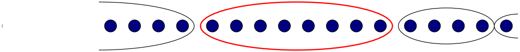
Not combinatorial enough.

Intuitively, many problems are hard but do not hide a large encoding problem.

Concrete examples:

- the bit-probe model
- the separator (Bloomier filter) is too large to analyze the full tradeoff for VERIFY-SUM

What is the time tree trying to tell us?



Information learned by future epochs about epoch i

$\leq$ cells written in future epochs

At most t_u cell writes per future update.

But also

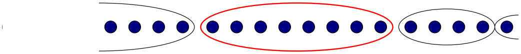
$\leq$ cells ever read from epoch i

On average $O(\frac{t_u}{\#\text{epochs}})$ cell reads per future update.

Still in old “help bits” framework.
Just reuse old analysis with better bound.

- epoch i has $(99\frac{t_u}{t_q})^i$ updates
- $t_q = \Omega(\lg n / \lg \frac{t_u}{t_q}) \Rightarrow \max\{t_u, t_q\} = \Omega(\lg n)$.

What is the time tree trying to tell us?



Information learned by future epochs about epoch i

$\leq$ cells written in future epochs

At most t_u cell writes per future update.

But also

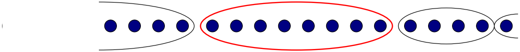
$\leq$ cells ever read from epoch i

On average $O(\frac{t_u}{\#\text{epochs}})$ cell reads per future update.

Still in old "help bits" framework.
Just reuse old analysis with better bound.

- epoch i has $(99\frac{t_u}{t_q})^i$ updates
- $t_q = \Omega(\lg n / \lg \frac{t_u}{t_q}) \Rightarrow \max\{t_u, t_q\} = \Omega(\lg n)$.

What is the time tree trying to tell us?



Information learned by future epochs about epoch i

$\leq$ cells written in future epochs

At most t_u cell writes per future update.

But also

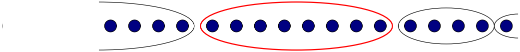
$\leq$ cells ever read from epoch i

On average $O(\frac{t_u}{\#\text{epochs}})$ cell reads per future update.

Still in old "help bits" framework.
Just reuse old analysis with better bound.

- epoch i has $(99\frac{t_u}{t_q})^i$ updates
- $t_q = \Omega(\lg n / \lg \frac{t_u}{t_q}) \Rightarrow \max\{t_u, t_q\} = \Omega(\lg n)$.

What is the time tree trying to tell us?



Information learned by future epochs about epoch i

$\leq$ cells written in future epochs

At most t_u cell writes per future update.

But also

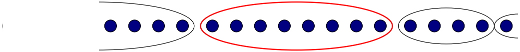
$\leq$ cells ever read from epoch i

On average $O(\frac{t_u}{\#\text{epochs}})$ cell reads per future update.

Still in old "help bits" framework.
Just reuse old analysis with better bound.

- epoch i has $(99\frac{t_u}{t_q})^i$ updates
- $t_q = \Omega(\lg n / \lg \frac{t_u}{t_q}) \Rightarrow \max\{t_u, t_q\} = \Omega(\lg n)$.

What is the time tree trying to tell us?



Information learned by future epochs about epoch i

$\leq$ cells written in future epochs

At most t_u cell writes per future update.

But also

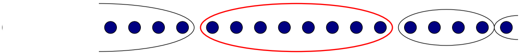
$\leq$ cells ever read from epoch i

On average $O(\frac{t_u}{\#\text{epochs}})$ cell reads per future update.

Still in old "help bits" framework.
Just reuse old analysis with better bound.

- epoch i has $(99\frac{t_u}{t_q})^i$ updates
- $t_q = \Omega(\lg n / \lg \frac{t_u}{t_q}) \Rightarrow \max\{t_u, t_q\} = \Omega(\lg n)$.

What is the time tree trying to tell us?



Information learned by future epochs about epoch i

$\leq$ cells written in future epochs

At most t_u cell writes per future update.

But also

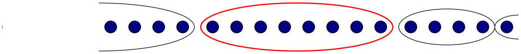
$\leq$ cells ever read from epoch i

On average $O(\frac{t_u}{\#\text{epochs}})$ cell reads per future update.

Still in old “help bits” framework.
Just reuse old analysis with better bound.

- epoch i has $(99\frac{t_u}{t_q})^i$ updates
- $t_q = \Omega(\lg n / \lg \frac{t_u}{t_q}) \Rightarrow \max\{t_u, t_q\} = \Omega(\lg n)$.

What is the time tree trying to tell us?



Information learned by future epochs about epoch i

$\leq$ cells written in future epochs

At most t_u cell writes per future update.

But also

$\leq$ cells ever read from epoch i

On average $O(\frac{t_u}{\#\text{epochs}})$ cell reads per future update.

Still in old “help bits” framework.
Just reuse old analysis with better bound.

- epoch i has $(99\frac{t_u}{t_q})^i$ updates
- $t_q = \Omega(\lg n / \lg \frac{t_u}{t_q}) \Rightarrow \max\{t_u, t_q\} = \Omega(\lg n)$.

Handwaving about the reads/epoch bound

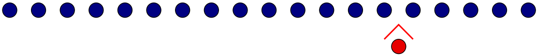
Randomize epoch construction:

- query happens at a random time
- build epochs from there

Classify cell probes by the span between read an write times.

The randomized epoch construction finds few cells of the right span, in expectation.

Handwaving about the reads/epoch bound



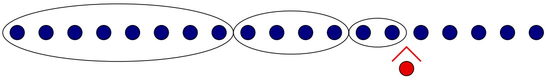
Randomize epoch construction:

- query happens at a random time
- build epochs from there

Classify cell probes by the span between read an write times.

The randomized epoch construction finds few cells of the right span, in expectation.

Handwaving about the reads/epoch bound



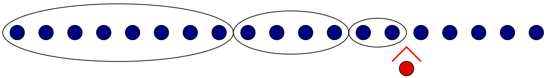
Randomize epoch construction:

- query happens at a random time
- build epochs from there

Classify cell probes by the span between read an write times.

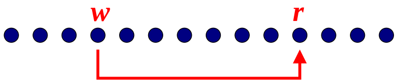
The randomized epoch construction finds few cells of the right span, in expectation.

Handwaving about the reads/epoch bound



Randomize epoch construction:

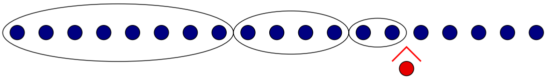
- query happens at a random time
- build epochs from there



Classify cell probes by the span between read an write times.

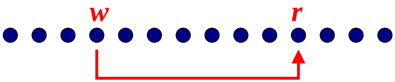
The randomized epoch construction finds few cells of the right span, in expectation.

Handwaving about the reads/epoch bound



Randomize epoch construction:

- query happens at a random time
- build epochs from there



Classify cell probes by the span between read an write times.

The randomized epoch construction finds few cells of the right span, in expectation.

What lays ahead. . .

[$\Sigma\iota\beta\upsilon\lambda\lambda\alpha$ $\Delta\varepsilon\lambda\phi\iota\varsigma$] was born before $T\rho\omega\iota\kappa\omega\nu$,
and she wrote oracles in verse.
($\Sigma o\upsilon\delta\alpha$)

Chapters to be written

- need to access $\omega(1)$ cells per epoch
  Essentially, we need a static lower bound for each epoch:
  - SELECT $\Rightarrow$ predecessor search in each epoch
  - range queries $\Rightarrow$ static range query in each epoch

  Objection: static lower bounds are hard.

- below the logarithmic barrier: dynamic search problems
  - range reporting in 1D, deterministic dictionaries
  - easy statically, nondet. and conondet.
- bounds of $n^{\Omega(1)}$ (e.g. directed graph problems)
  - epochs are useless
  - idea: conjecture on communication games with help bits

- need to access $\omega(1)$ cells per epoch
  Essentially, we need a static lower bound for each epoch:
  - SELECT $\Rightarrow$ predecessor search in each epoch
  - range queries $\Rightarrow$ static range query in each epoch

  Objection: static lower bounds are hard.
  - can dynamize range structures $\Rightarrow$ unavoidable
  - recent progress by M.P. and Thorup

- below the logarithmic barrier: dynamic search problems
  - range reporting in 1D, deterministic dictionaries
  - easy statically, nondet. and conondet.

- bounds of $n^{\Omega(1)}$ (e.g. directed graph problems)
  - epochs are useless
  - idea: conjecture on communication games with help bits

Chapters to be written

- need to access $\omega(1)$ cells per epoch
  Essentially, we need a static lower bound for each epoch:
  - SELECT $\Rightarrow$ predecessor search in each epoch
  - range queries $\Rightarrow$ static range query in each epoch

  Objection: static lower bounds are hard.
  - can dynamize range structures $\Rightarrow$ unavoidable
  - recent progress by M.P. and Thorup
- below the logarithmic barrier: dynamic search problems
  - range reporting in 1D, deterministic dictionaries
  - easy statically, nondet. and conondet.
- bounds of $n^{\Omega(1)}$ (e.g. directed graph problems)
  - epochs are useless
  - idea: conjecture on communication games with help bits

Chapters to be written

- need to access $\omega(1)$ cells per epoch
  Essentially, we need a static lower bound for each epoch:
  - SELECT $\Rightarrow$ predecessor search in each epoch
  - range queries $\Rightarrow$ static range query in each epoch

  Objection: static lower bounds are hard.
  - can dynamize range structures $\Rightarrow$ unavoidable
  - recent progress by M.P. and Thorup
- below the logarithmic barrier: dynamic search problems
  - range reporting in 1D, deterministic dictionaries
  - easy statically, nondet. and conondet.
- bounds of $n^{\Omega(1)}$ (e.g. directed graph problems)
  - epochs are useless
  - idea: conjecture on communication games with help bits

Chapters to be written

- need to access $\omega(1)$ cells per epoch
  Essentially, we need a static lower bound for each epoch:
  - SELECT $\Rightarrow$ predecessor search in each epoch
  - range queries $\Rightarrow$ static range query in each epoch

  Objection: static lower bounds are hard.
  - can dynamize range structures $\Rightarrow$ unavoidable
  - recent progress by M.P. and Thorup
- below the logarithmic barrier: dynamic search problems
  - range reporting in 1D, deterministic dictionaries
  - easy statically, nondet. and conondet.
- bounds of $n^{\Omega(1)}$ (e.g. directed graph problems)
  - epochs are useless
  - idea: conjecture on communication games with help bits

Chapters to be written

- need to access $\omega(1)$ cells per epoch
  Essentially, we need a static lower bound for each epoch:
  - SELECT $\Rightarrow$ predecessor search in each epoch
  - range queries $\Rightarrow$ static range query in each epoch

  Objection: static lower bounds are hard.
  - can dynamize range structures $\Rightarrow$ unavoidable
  - recent progress by M.P. and Thorup
- below the logarithmic barrier: dynamic search problems
  - range reporting in 1D, deterministic dictionaries
  - easy statically, nondet. and conondet.
- bounds of $n^{\Omega(1)}$ (e.g. directed graph problems)
  - epochs are useless
  - idea: conjecture on communication games with help bits

Chapters to be written

- need to access $\omega(1)$ cells per epoch
  Essentially, we need a static lower bound for each epoch:
  - SELECT $\Rightarrow$ predecessor search in each epoch
  - range queries $\Rightarrow$ static range query in each epoch

  Objection: static lower bounds are hard.
  - can dynamize range structures $\Rightarrow$ unavoidable
  - recent progress by M.P. and Thorup
- below the logarithmic barrier: dynamic search problems
  - range reporting in 1D, deterministic dictionaries
  - easy statically, nondet. and conondet.
- bounds of $n^{\Omega(1)}$ (e.g. directed graph problems)
  - epochs are useless
  - idea: conjecture on communication games with help bits

Chapters to be written

- need to access $\omega(1)$ cells per epoch
  Essentially, we need a static lower bound for each epoch:
  - SELECT $\Rightarrow$ predecessor search in each epoch
  - range queries $\Rightarrow$ static range query in each epoch

  Objection: static lower bounds are hard.
  - can dynamize range structures $\Rightarrow$ unavoidable
  - recent progress by M.P. and Thorup
- below the logarithmic barrier: dynamic search problems
  - range reporting in 1D, deterministic dictionaries
  - easy statically, nondet. and conondet.
- bounds of $n^{\Omega(1)}$ (e.g. directed graph problems)
  - epochs are useless
  - idea: conjecture on communication games with help bits

The story ends here (for now...)

THE END

Sleep tight.