Logarithmic Lower Bounds in the Cell-Probe Model

Mihai Pătraşcu

MIT CSAIL

This talk is based on two papers appearing in SODA'04 and STOC'04 by Mihai Pătraşcu and Erik Demaine.

Setup for the Problems

- only dynamic data structures (t_u vs t_q)
- cell-probe model
- dynamic language membership problems
  - n = size of problem representation in bits
  - updates take input of $O(\lg n)$ bits
  - queries take no input, return boolean answer

ways to cheat:
  - large input/output can amplify hardness
  - "n" denotes some other parameter; the size of the problem in bits is sometimes exponential in this n

Setup for the Problems

- only dynamic data structures (t_u vs t_q)
- cell-probe model
- dynamic language membership problems
  - $n =$ size of problem representation in bits
  - updates take input of $O(\lg n)$ bits
  - queries take no input, return boolean answer

Ways to cheat:

  - large input/output can amplify hardness
  - "n" denotes some other parameter; the size of the problem in bits is sometimes exponential in this n

Setup for the Problems

- only dynamic data structures (t_u vs t_q)
- cell-probe model
- dynamic language membership problems
  - $n =$ size of problem representation in bits
  - updates take input of $O(\lg n)$ bits
  - queries take no input, return boolean answer

Ways to cheat:

- large input/output can amplify hardness
- "n" denotes some other parameter; the size of the problem in bits is sometimes exponential in this n

Setup for the Problems

- only dynamic data structures (t_u vs t_q)
- cell-probe model
- dynamic language membership problems
  - $n =$ size of problem representation in bits
  - updates take input of $O(\lg n)$ bits
  - queries take no input, return boolean answer

Ways to cheat:

- large input/output can amplify hardness
- "n" denotes some other parameter; the size of the problem in bits is sometimes exponential in this n

Setup for the Problems

- only dynamic data structures (t_u vs t_q)
- cell-probe model
- dynamic language membership problems
  - $n =$ size of problem representation in bits
  - updates take input of $O(\lg n)$ bits
  - queries take no input, return boolean answer

Ways to cheat:

- large input/output can amplify hardness
- "n" denotes some other parameter; the size of the problem in bits is sometimes exponential in this n

Setup for the Problems

- only dynamic data structures (t_u vs t_q)
- cell-probe model
- dynamic language membership problems
  - $n =$ size of problem representation in bits
  - updates take input of $O(\lg n)$ bits
  - queries take no input, return boolean answer

Ways to cheat:

- large input/output can amplify hardness
- "n" denotes some other parameter; the size of the problem in bits is sometimes exponential in this n

Setup for the Problems

- only dynamic data structures (t_u vs t_q)
- cell-probe model
- dynamic language membership problems
  - $n =$ size of problem representation in bits
  - updates take input of $O(\lg n)$ bits
  - queries take no input, return boolean answer

Ways to cheat:

- large input/output can amplify hardness
- "n" denotes some other parameter; the size of the problem in bits is sometimes exponential in this n

Setup for the Problems

- only dynamic data structures (t_u vs t_q)
- cell-probe model
- dynamic language membership problems
  - $n =$ size of problem representation in bits
  - updates take input of $O(\lg n)$ bits
  - queries take no input, return boolean answer

Ways to cheat:

- large input/output can amplify hardness
- "*n*" denotes some other parameter; the size of the problem in bits is sometimes exponential in this n

Lower Bounds Timeline

1989 Fredman and Saks: $\Omega(\frac{\lg n}{\lg \lg n})$ for partial sums problem
tradeoff: $t_q \lg(t_u \lg n) = \Omega(\lg n)$

199* variations of [FS89] bound: Husfeldt and Rauhe, Ben-Amram and Galil, etc

1998 Alstrup, Husfeldt and Rauhe: marked ancestor problem
tradeoff: $t_q \lg t_u = \Omega(\lg n)$
still cannot improve bound on $\max\{t_u, t_q\}$

2004 Pătraşcu, Demaine: $\Omega(\lg n)$
tradeoff: $t_q \lg(\frac{t_u}{t_q}) = \Omega(\lg n)$
AND symmetric: $t_u \lg(\frac{t_q}{t_u}) = \Omega(\lg n)$

The Partial-Sums Problem

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

$\mathrm{update}(k, \Delta)$ modify $A[k] \leftarrow A[k] + \Delta$.

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

$\mathrm{select}(\sigma)$ return $i : \mathrm{sum}(i) \leq \sigma < \mathrm{sum}(i+1)$.

Parameters:

$n =$ size of the array

$b =$ size of cell in bits; also size of $A[i]$

$\delta =$ parameter Δ to update has this many bits

The optimal bound is: $\Theta\left(\frac{\lg n}{\lg(b/\delta)}\right)$

The Partial-Sums Problem

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

$\mathrm{update}(k, \Delta)$ modify $A[k] \leftarrow A[k] + \Delta$.

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

$\mathrm{select}(\sigma)$ return $i : \mathrm{sum}(i) \leq \sigma < \mathrm{sum}(i+1)$.

Parameters:

$n =$ size of the array

$b =$ size of cell in bits; also size of $A[i]$

$\delta =$ parameter Δ to update has this many bits

The optimal bound is: $\Theta\left(\frac{\lg n}{\lg(b/\delta)}\right)$

The Partial-Sums Problem

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

$\mathrm{update}(k, \Delta)$ modify $A[k] \leftarrow A[k] + \Delta$.

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

$\mathrm{select}(\sigma)$ return $i : \mathrm{sum}(i) \leq \sigma < \mathrm{sum}(i+1)$.

Parameters:

$n =$ size of the array

$b =$ size of cell in bits; also size of $A[i]$

$\delta =$ parameter Δ to update has this many bits

The optimal bound is: $\Theta\left(\frac{\lg n}{\lg(b/\delta)}\right)$

Will prove $\Omega(\lg n)$ for partial sums when $\delta = b$.

Ready. . . Steady. . .

Lower Bound, Version 1

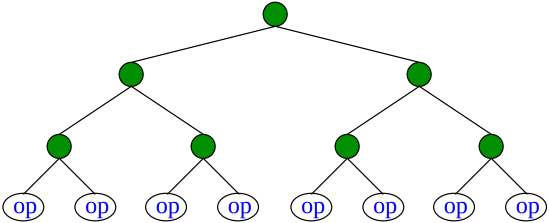
Generate a random sequence of operations. Choose uniformly between:

- update(random index, random Δ)
- sum(random index)

op op op op op op op op

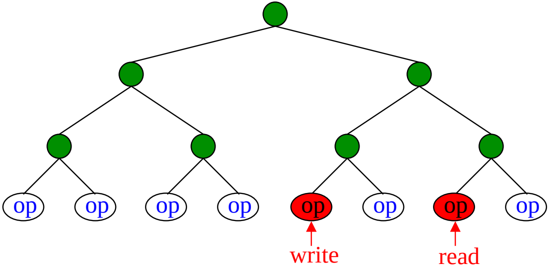
Lower Bound, Version 1

Build a balanced tree with operations in the leaves (considered in chronological order – “time tree”)



A cell probe is characterized by:

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



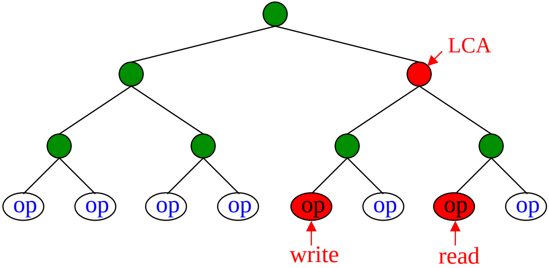
Lower Bound, Version 1

Cell probe counted as "information transfer" through LCA

Prove lower bounds on information transfer through each node

Then sum up

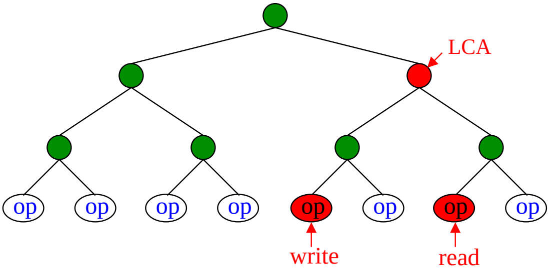
- not double counting any cell probe
- summing works for average case lower bounds



Lower Bound, Version 1

Cell probe counted as “information transfer” through LCA
Prove lower bounds on information transfer through each node
Then sum up

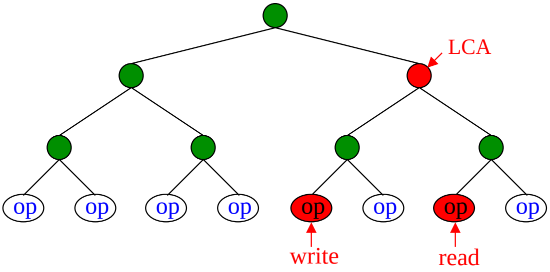
- not double counting any cell probe
- summing works for average case lower bounds



Lower Bound, Version 1

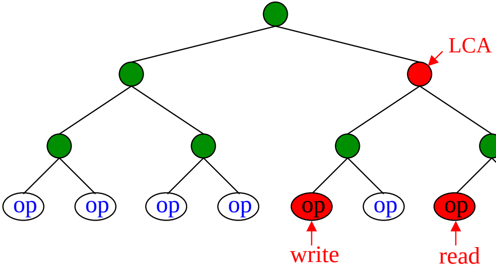
Cell probe counted as "information transfer" through LCA
Prove lower bounds on information transfer through each node
Then sum up

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



Cell probe counted as “information transfer” through LCA
Prove lower bounds on information transfer through each node
Then sum up

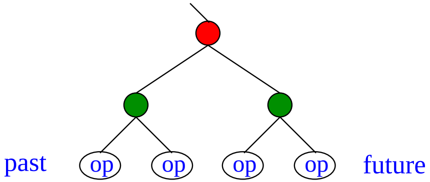
- not double counting any cell probe
- summing works for average case lower bounds



How to prove a lower bound on the information transfer?

Consider scenario:

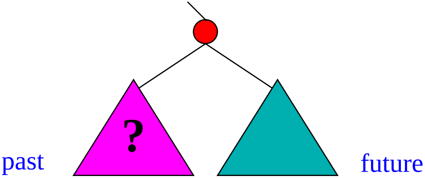
- know operations from the past and right subtree
- don't know updates from left subtree
- given the addresses and contents for cells written in left subtree, read in right subtree



Lower bound for information transfer through one node

How to prove a lower bound on the information transfer?
Consider scenario:

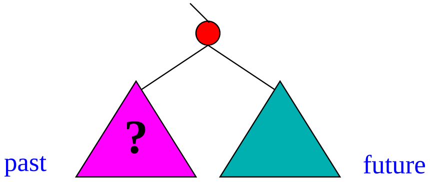
- know operations from the past and right subtree
- don't know updates from left subtree
- given the addresses and contents for cells written in left subtree, read in right subtree



Lower bound for information transfer through one node

How to prove a lower bound on the information transfer?
Consider scenario:

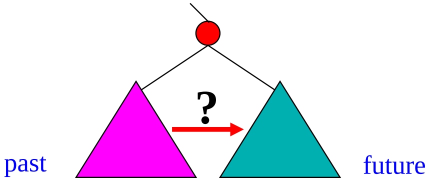
- know operations from the past and right subtree
- don't know updates from left subtree
- given the addresses and contents for cells written in left subtree, read in right subtree



Lower bound for information transfer through one node

How to prove a lower bound on the information transfer?
Consider scenario:

- know operations from the past and right subtree
- don't know updates from left subtree
- given the addresses and contents for
  cells written in left subtree, read in right subtree



Lower bound for information transfer through one node

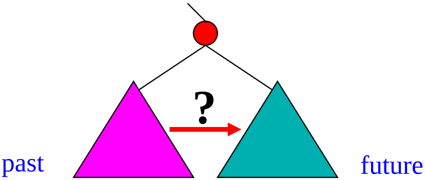
Claim: can simulate data structure for time in right subtree

To simulate read to a cell written at time t_u:

t_u in "past" $\Rightarrow$ have complete information about past

t_u in left subtree $\Rightarrow$ adress and contests in information transfer list

t_u in right subtree $\Rightarrow$ already simulated the write



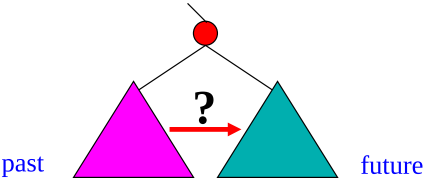
Claim: can simulate data structure for time in right subtree

To simulate read to a cell written at time t_u:

t_u in "past" $\Rightarrow$ have complete information about past

t_u in left subtree $\Rightarrow$ adress and contests in information transfer list

t_u in right subtree $\Rightarrow$ already simulated the write



Lower bound for information transfer through one node

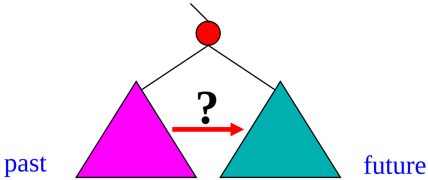
Claim: can simulate data structure for time in right subtree

To simulate read to a cell written at time t_u:

t_u in "past" $\Rightarrow$ have complete information about past

t_u in left subtree $\Rightarrow$ adress and contests in information transfer list

t_u in right subtree $\Rightarrow$ already simulated the write



Lower bound for information transfer through one node

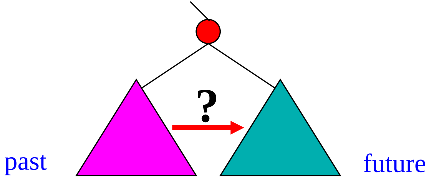
Claim: can simulate data structure for time in right subtree

To simulate read to a cell written at time t_u:

t_u in "past" $\Rightarrow$ have complete information about past

t_u in left subtree $\Rightarrow$ adress and contests in information transfer list

t_u in right subtree $\Rightarrow$ already simulated the write

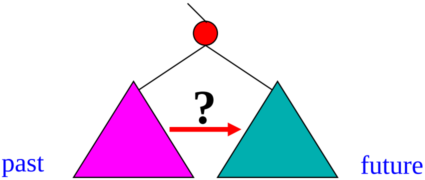


Lower bound for information transfer through one node

Claim: can simulate data structure for time in right subtree

To simulate read to a cell written at time t_u:

- t_u in "past" $\Rightarrow$ have complete information about past
- t_u in left subtree $\Rightarrow$ adress and contests in information transfer list
- t_u in right subtree $\Rightarrow$ already simulated the write



Can simulate data structure for time interval in right subtree

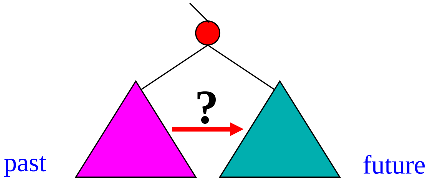
$\Rightarrow$ can recover answer to queries from right subtree

Expected linear interleave between update indices in left subtree and query indices in right subtree

$\Rightarrow$ query answers encode a linear amount of information about left subtree

$\Rightarrow$ information transfer is linear

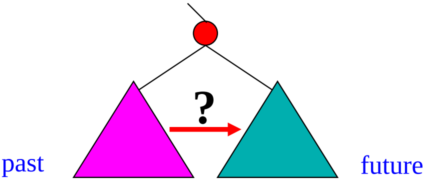
$\Rightarrow$ summing over all nodes: $\Omega(\lg n)$ per leaf (operation)



Lower bound for information transfer through one node

Can simulate data structure for time interval in right subtree
$\Rightarrow$ can recover answer to queries from right subtree

Expected linear interleave between update indices in left subtree and query indices in right subtree
$\Rightarrow$ query answers encode a linear amount of information about left subtree
$\Rightarrow$ information transfer is linear
$\Rightarrow$ summing over all nodes: $\Omega(\lg n)$ per leaf (operation)



Lower bound for information transfer through one node

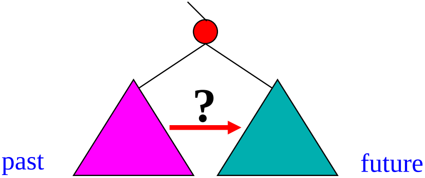
Can simulate data structure for time interval in right subtree
$\Rightarrow$ can recover answer to queries from right subtree

Expected linear interleave between update indices in left subtree and query indices in right subtree

$\Rightarrow$ query answers encode a linear amount of information about left subtree

$\Rightarrow$ information transfer is linear

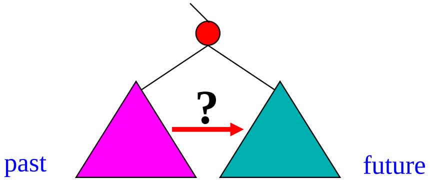
$\Rightarrow$ summing over all nodes: $\Omega(\lg n)$ per leaf (operation)



Lower bound for information transfer through one node

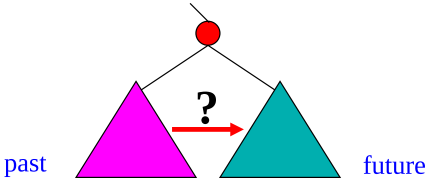
Can simulate data structure for time interval in right subtree
$\Rightarrow$ can recover answer to queries from right subtree

Expected linear interleave between update indices in left subtree and query indices in right subtree
$\Rightarrow$ query answers encode a linear amount of information about left subtree
$\Rightarrow$ information transfer is linear
$\Rightarrow$ summing over all nodes: $\Omega(\lg n)$ per leaf (operation)



Can simulate data structure for time interval in right subtree
$\Rightarrow$ can recover answer to queries from right subtree

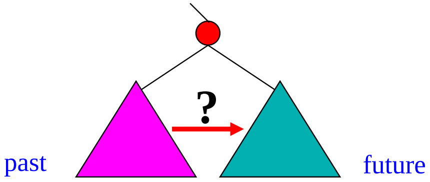
Expected linear interleave between update indices in left subtree and query indices in right subtree
$\Rightarrow$ query answers encode a linear amount of information about left subtree
$\Rightarrow$ information transfer is linear
$\Rightarrow$ summing over all nodes: $\Omega(\lg n)$ per leaf (operation)



Lower bound for information transfer through one node

Can simulate data structure for time interval in right subtree
$\Rightarrow$ can recover answer to queries from right subtree

Expected linear interleave between update indices in left subtree and query indices in right subtree
$\Rightarrow$ query answers encode a linear amount of information about left subtree
$\Rightarrow$ information transfer is linear
$\Rightarrow$ summing over all nodes: $\Omega(\lg n)$ per leaf (operation)



New Problem: Dynamic Connectivity

Most elementary dynamic graph problem.
Maintain a dynamic graph on n vertices under:

$\text{insert}(u, v)$	insert an edge (u, v) into the graph.
$\text{delete}(u, v)$	delete the edge (u, v) from the graph.
$\text{connected}(u, v)$	u, v in the same connected component?

- partial sums was not dynamic language membership (queries had nonbinary answers)
- dynamic connectivity can be made expressed as such

New Problem: Dynamic Connectivity

Most elementary dynamic graph problem.
Maintain a dynamic graph on n vertices under:

$\text{insert}(u, v)$	insert an edge (u, v) into the graph.
$\text{delete}(u, v)$	delete the edge (u, v) from the graph.
$\text{connected}(u, v)$	u, v in the same connected component?

- partial sums was not dynamic language membership (queries had nonbinary answers)
- dynamic connectivity can be made expressed as such

Results for Dynamic Connectivity

Thorup $O(\lg n(\lg\lg n)^3)$ updates, $O(\frac{\lg n}{\lg\lg\lg n})$ queries

Holm et al $O(\lg^2 n)$ updates, $O(\frac{\lg n}{\lg\lg n})$ queries

Sleator, Tarjan $O(\lg n)$ for trees

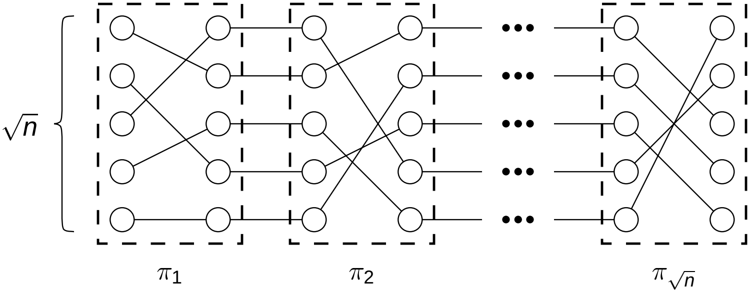
Eppstein et al $O(\lg n)$ for plane graphs

several $\Omega(\lg n / \lg\lg n)$

new $\Omega(\lg n)$

- holds for paths (thus, also for trees, plane graphs)
- tradeoff matched for trees (for $t_u > t_q$)
- Thorup and Holm et al are on tradeoff curve

Lower bound for dynamic connectivity – setup



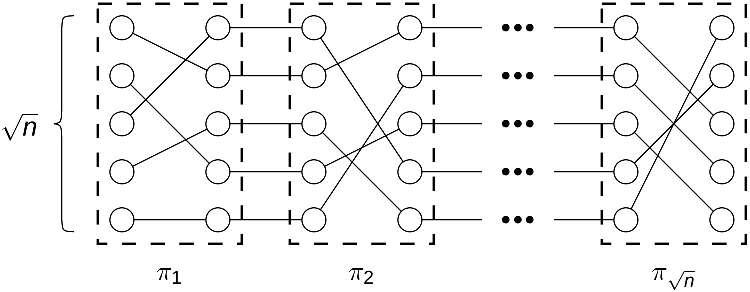
graph $\approx$ array of $\sqrt{n}$ elements in permutation group $S_{\sqrt{n}}$

update $\approx$ change a position of the array
delete $\sqrt{n}$ edges, insert $\sqrt{n}$ edges

query $\approx$ find a partial sum

Hmmm. . . Really?

Lower bound for dynamic connectivity – setup



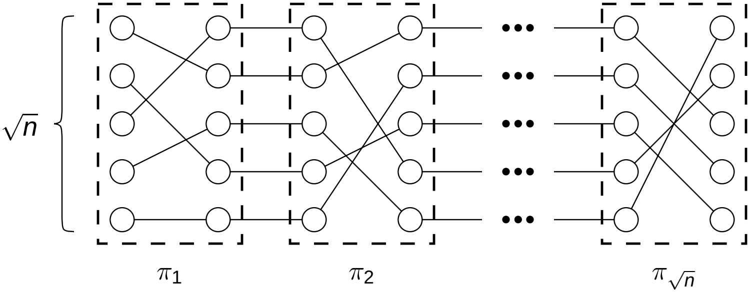
graph $\approx$ array of $\sqrt{n}$ elements in permutation group $S_{\sqrt{n}}$

update $\approx$ change a position of the array
delete $\sqrt{n}$ edges, insert $\sqrt{n}$ edges

query $\approx$ find a partial sum

Hmmm. . . Really?

Lower bound for dynamic connectivity – setup



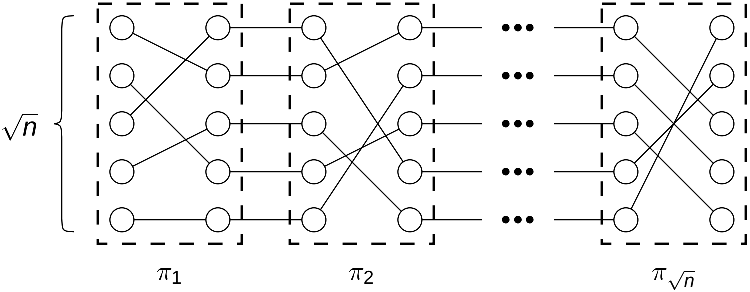
graph $\approx$ array of $\sqrt{n}$ elements in permutation group $S_{\sqrt{n}}$

update $\approx$ change a position of the array
delete $\sqrt{n}$ edges, insert $\sqrt{n}$ edges

query $\approx$ find a partial sum

Hmmm. . . Really?

Lower bound for dynamic connectivity – setup



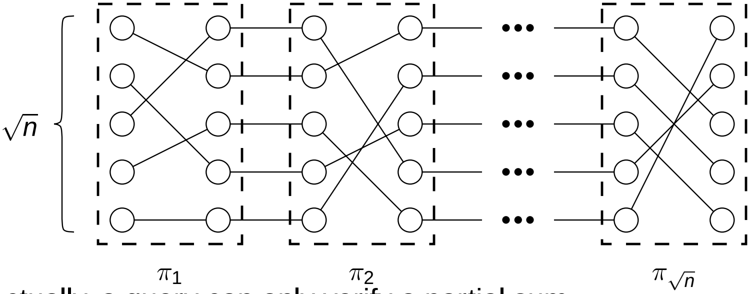
graph $\approx$ array of $\sqrt{n}$ elements in permutation group $S_{\sqrt{n}}$

update $\approx$ change a position of the array
delete $\sqrt{n}$ edges, insert $\sqrt{n}$ edges

query $\approx$ find a partial sum

Hmmm. . . Really?

Lower bound for dynamic connectivity – setup



Actually, a query can only verify a partial sum through $\sqrt{n}$ connectivity queries

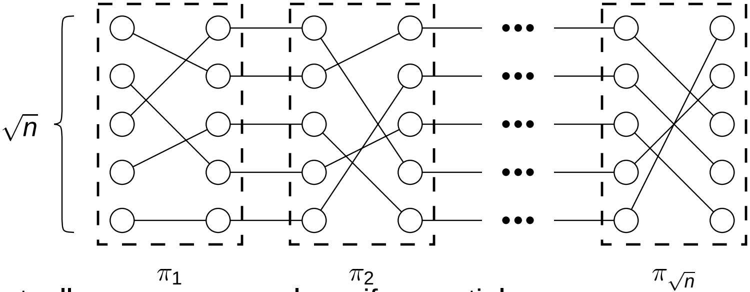
The Partial-Sums Problem with Verify

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

$\text{update}(k, x)$ modify $A[k] \leftarrow x$.

$\text{verify}(k, x)$ test whether $\sum_{i=1}^{k} A[i] = x$.

Lower bound for dynamic connectivity – setup



Actually, a query can only verify a partial sum through $\sqrt{n}$ connectivity queries

The Partial-Sums Problem with Verify

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

update(k, x) modify $A[k] \leftarrow x$.

verify(k, x) test whether $\sum_{i=1}^{k} A[i] = x$.

Coping with boolean queries

Problem: Entropy of query answers is very low (one bit)

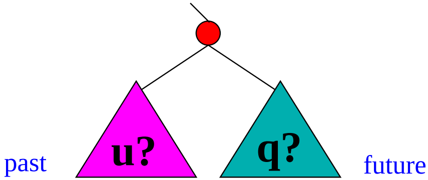
Idea:

- construct hard sequence, where all queries return true
- there is a unique x for which $\mathrm{verify}(k, x)$ returns true
- information is in the parameter x, not the answer
- information is given to the algorithm for verification (not produced by the algorithm)

Lower bounds for information transfer, version 2

- know everything that happened in the past
- don't know updates from left subtree
- don't know parameter for queries from right subtree

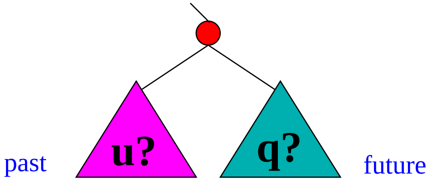
Strive to recover parameters for queries
knowing that answers are always true



Lower bounds for information transfer, version 2

- know everything that happened in the past
- don't know updates from left subtree
- don't know parameter for queries from right subtree

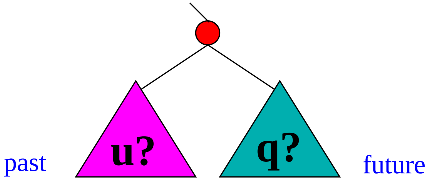
Strive to recover parameters for queries
knowing that answers are always true



Lower bounds for information transfer, version 2

- know everything that happened in the past
- don't know updates from left subtree
- don't know parameter for queries from right subtree

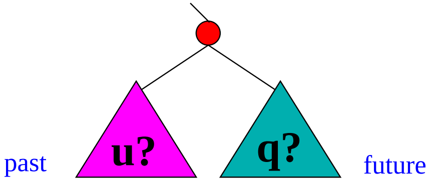
Strive to recover parameters for queries
knowing that answers are always true



Lower bounds for information transfer, version 2

- know everything that happened in the past
- don't know updates from left subtree
- don't know parameter for queries from right subtree

Strive to recover parameters for queries
knowing that answers are always true

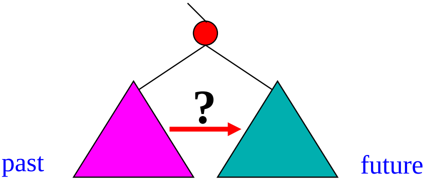


Encoding: cells (address and contents) written in left subtree that are read in right subtree by the correct queries

To decode:

- simulate all possible queries for right subtree
- find the parameter setting which returns true

Doesn't quite work!



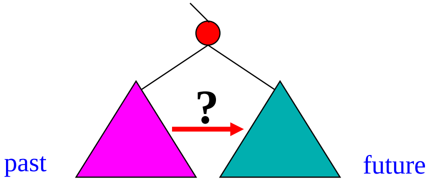
Lower bounds for information transfer, version 2

Encoding: cells (address and contents) written in left subtree that are read in right subtree by the correct queries

To decode:

- simulate all possible queries for right subtree
- find the parameter setting which returns true

Doesn't quite work!

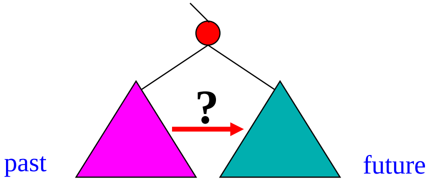


Encoding: cells (address and contents) written in left subtree that are read in right subtree by the correct queries

To decode:

- simulate all possible queries for right subtree
- find the parameter setting which returns true

Doesn't quite work!



What is needed for correct decoding?

Trouble with previous scheme:

- say correct query is $\mathrm{verify}(k, x)$
- when we simulate $\mathrm{verify}(k, x')$ it reads cell A
- A is written in left subtree, but not read by $\mathrm{verify}(k, x)$
- hence A is not in our list of probed cells
- while simulating $\mathrm{verify}(k, x')$ we think A has an old value
- with incorrect simulation, $\mathrm{verify}(k, x')$ might return true!

Alternative view: covert information channel
The fact that some cell was not modified is information!

What is needed for correct decoding?

Trouble with previous scheme:

- say correct query is $\text{verify}(k, x)$
- when we simulate $\text{verify}(k, x')$ it reads cell A
- A is written in left subtree, but not read by $\text{verify}(k, x)$
- hence A is not in our list of probed cells
- while simulating $\text{verify}(k, x')$ we think A has an old value
- with incorrect simulation, $\text{verify}(k, x')$ might return true!

Alternative view: covert information channel
The fact that some cell was not modified **is** information!

What is needed for correct decoding?

Trouble with previous scheme:

- say correct query is $\mathrm{verify}(k, x)$
- when we simulate $\mathrm{verify}(k, x')$ it reads cell A
- A is written in left subtree, but not read by $\mathrm{verify}(k, x)$
- hence A is not in our list of probed cells
- while simulating $\mathrm{verify}(k, x')$ we think A has an old value
- with incorrect simulation, $\mathrm{verify}(k, x')$ might return true!

Alternative view: covert information channel
The fact that some cell was not modified **is** information!

What is needed for correct decoding?

Trouble with previous scheme:

- say correct query is $\mathrm{verify}(k, x)$
- when we simulate $\mathrm{verify}(k, x')$ it reads cell A
- A is written in left subtree, but not read by $\mathrm{verify}(k, x)$
- hence A is not in our list of probed cells
- while simulating $\mathrm{verify}(k, x')$ we think A has an old value
- with incorrect simulation, $\mathrm{verify}(k, x')$ might return true!

Alternative view: covert information channel

The fact that some cell was not modified **is** information!

What is needed for correct decoding?

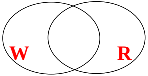
Trouble with previous scheme:

- say correct query is $\mathrm{verify}(k, x)$
- when we simulate $\mathrm{verify}(k, x')$ it reads cell A
- A is written in left subtree, but not read by $\mathrm{verify}(k, x)$
- hence A is not in our list of probed cells
- while simulating $\mathrm{verify}(k, x')$ we think A has an old value
- with incorrect simulation, $\mathrm{verify}(k, x')$ might return true!

Alternative view: covert information channel

The fact that some cell was not modified **is** information!

The final encoding



$W =$ cells written in left subtree

$R =$ cells read in right subtree by the correct queries

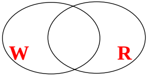
Encoding contains:

- complete information for $W \cap R$
- separator for $W \setminus R$ and $R \setminus W$

This suffices for correct simulation:

- cell accessed from $W \cap R$ – have complete information
- cell accessed from R's side of separator
  – know it was not modified in left subtree
- cell accessed from W's side of separator
  – kill simulation thread; this cannot be the correct query

The final encoding



$W =$ cells written in left subtree

$R =$ cells read in right subtree by the correct queries

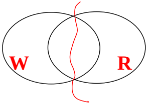
Encoding contains:

- complete information for $W \cap R$
- separator for $W \setminus R$ and $R \setminus W$

This suffices for correct simulation:

- cell accessed from $W \cap R$ – have complete information
- cell accessed from R's side of separator – know it was not modified in left subtree
- cell accessed from W's side of separator – kill simulation thread; this cannot be the correct query

The final encoding



W = cells written in left subtree

R = cells read in right subtree by the correct queries

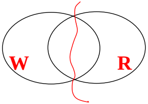
Encoding contains:

- complete information for $W \cap R$
- separator for $W \setminus R$ and $R \setminus W$

This suffices for correct simulation:

- cell accessed from $W \cap R$ – have complete information
- cell accessed from R's side of separator – know it was not modified in left subtree
- cell accessed from W's side of separator – kill simulation thread; this cannot be the correct query

The final encoding



$W =$ cells written in left subtree
$R =$ cells read in right subtree by the correct queries

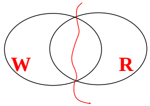
Encoding contains:

- complete information for $W \cap R$
- separator for $W \setminus R$ and $R \setminus W$

This suffices for correct simulation:

- cell accessed from $W \cap R$ – have complete information
- cell accessed from R's side of separator – know it was not modified in left subtree
- cell accessed from W's side of separator – kill simulation thread; this cannot be the correct query

The final encoding



$W =$ cells written in left subtree

$R =$ cells read in right subtree by the correct queries

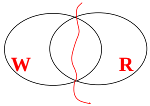
Encoding contains:

- complete information for $W \cap R$
- separator for $W \setminus R$ and $R \setminus W$

This suffices for correct simulation:

- cell accessed from $W \cap R$ – have complete information
- cell accessed from R's side of separator – know it was not modified in left subtree
- cell accessed from W's side of separator – kill simulation thread; this cannot be the correct query

The final encoding



$W =$ cells written in left subtree
$R =$ cells read in right subtree by the correct queries

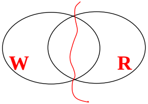
Encoding contains:

- complete information for $W \cap R$
- separator for $W \setminus R$ and $R \setminus W$

This suffices for correct simulation:

- cell accessed from $W \cap R$ – have complete information
- cell accessed from R's side of separator – know it was not modified in left subtree
- cell accessed from W's side of separator – kill simulation thread; this cannot be the correct query

The final encoding



W = cells written in left subtree
R = cells read in right subtree by the correct queries

Encoding contains:

- complete information for $W \cap R$
- separator for $W \setminus R$ and $R \setminus W$

This suffices for correct simulation:

- cell accessed from $W \cap R$ – have complete information
- cell accessed from R's side of separator
  – know it was not modified in left subtree
- cell accessed from W's side of separator
  – kill simulation thread; this cannot be the correct query

Other Stuff in the Papers

- handling higher word size b: nontrivial idea, somewhat similar to the round elimination lemma
- details: tradeoffs, randomized/nondeterministic lower bounds etc.
- reductions to other dynamic graph problems
- tight upper bound for partial sums

Future Research

Questions related to dynamic connectivity:

- find $O(\lg n)$ upper bound
- optimal bound for decremental connectivity, grid graphs?
- upper bounds in external memory

General questions:

- can we go beyond $\Omega(\lg n)$?

  long record of log barriers (**P** vs **L**, circuit depth)

  some progress: $\Omega\left(\left(\frac{\lg n}{\lg\lg n}\right)^2\right)$ in bit-probe model
- understand "reverse tradeoffs": $t_q > t_u$ (nontrivial!)

Future Research

Questions related to dynamic connectivity:

- find $O(\lg n)$ upper bound
- optimal bound for decremental connectivity, grid graphs?
- upper bounds in external memory

General questions:

- can we go beyond $\Omega(\lg n)$?

  long record of log barriers (**P** vs **L**, circuit depth)

  some progress: $\Omega\left(\left(\frac{\lg n}{\lg\lg n}\right)^2\right)$ in bit-probe model

- understand "reverse tradeoffs": $t_q > t_u$ (nontrivial!)

Thank you!