The Geometry Of Data Structures

John Iacono

Université Libre de Bruxelles
1-D Data Structure + Time = 2D
1-D Data structure + Time = 2D
Classic example: Line Sweep

Persistent search Trees [Sanacki +Turjan 86]
Classic example: Line Sweep

Persistent search Trees [Sanack & Turjan 86]
Classic example: Line Sweep

Persistent search Trees [Scanack +Turjan 86]
Line Sweep: Use 1-D "standard" Data Structures to solve geometric problems

Here: Use 2D geometry to solve "standard" data structure problems
Why?
Why?

- Can simplify
- Can see things that would be hard to see otherwise
- Can use standard geometric tools and intuition
OUTLINE

- Binary search Trees

= Persistent Cache - Oblivious

≡ Forbidden Submatricies and friends
Part — Binary Search Trees
Part — Binary Search Trees

Many collaborators

P.J. Bose
M.L. Fredman
E.D. Demaine
S. Langerman
M. Patrascu
D. Harmon
1878
Phone Book

1878
How To Find a Name in a Phone Book
How To Find a Name in a Phone Book

- Method 1: Read it like a book
How To Find a Name in a Phone Book

- Method 1: Read it like a book

- Method 2: Look in the middle, throw away half the book
How To Find a Name in a Phone Book

- Method 1: Read it like a book
  Phone Book Doubles in size: Twice as much time

- Method 2: Look in the middle,
  Throw away half the book
  Phone Book Doubles in size: One more lookup
How To Find a Name in a Phone Book

- Method 1: Read it like a book
  Phone Book Doubles in size: Twice as much time
  1,000,000 names → 10,000,000 lookups

- Method 2: Look in the middle,
  Throw away half the book
  Phone Book Doubles in size: One more lookup
  1,000,000 names → 20 lookups
How To Find a Name in a Phone Book

- Method 1: Read it like a book
  Phone Book Doubles in size: Twice as much time
  1,000,000 names → 1,000,000 lookups

- Method 2: Look in the middle,
  Throw away half the book
  Phone Book Doubles in size: One more lookup
  1,000,000 names → 20 lookups

Better
How about a Flow Chart?

Search in: ABCDEFG
How about a Flow Chart?

Search in: A B C D E F G

D
How about a Flow Chart?

Search in: A B C D E F G

D
How about a Flow Chart?

Search in: ABCD EFG

D

B

A C E G
Which is better?
Which is better?
Which is better?

But what if we only search for A?
Changing Your Mind on How to Search
Binary Search Tree Model
Binary Search Tree Model

Executes searches
Eg. search for 8
Binary Search Tree Model

Executes Searches
EG search for 8
Single pointer, starts each search at root
Binary Search Tree Model

Executes Searches
EG: search for 8
- Single pointer, starts each search at root
- Can move pointer left, right, up at each step.
Binary Search Tree Model

Executes searches

Eg. search for 8

Single pointer, starts each search at root
can move pointer Left, Right, Up at Unit Cost
Binary Search Tree Model

Executes Searches
EG search for 8
Single pointer, starts each search at root
Can move pointer
Left, Right, Up at Unit Cost
Binary Search Tree Model

- Executes Searches
  - E.g. search for 8
- Single pointer, starts each search at root
  - Can move pointer
  - Left, Right, Up at unit cost
  - Can rotate with parent at unit cost
Binary Search Tree Model

Executes Searches

EG search for 8

Single pointer, starts
Each search at root
Can move pointer
Left, Right, Up at Unit Cost
Can rotate with parent
at Unit Cost
Binary Search Tree Model

Executes Searches
Example search for 8
Single pointer, starts each search at root
can move pointer Left, Right, Up at Unit Cost
Can rotate with parent at Unit Cost

Must go to the searched item at Unit Cost
Binary Search Tree Model

Executes Searches
EG search for 8
Single pointer, starts each search at root
Can move pointer
Left, Right, Up at unit cost
Can rotate with parent

Must go to the searched item at unit cost
Binary Search Tree Model

Executes Searches

E.g. search for 8

Single pointer, starts each search at root
Can move pointer
Left, Right, Up at unit cost
Can rotate with parent

Must go to the searched item at unit cost
Binary Search Tree Model

- Executes searches
  - E.g., search for 8

- Single pointer, starts each search at root
  - Can move pointer
    - Left, Right, Up at unit cost
  - Can rotate with parent

- Must go to the searched item at unit cost
How would you arrange your stuff?
How would you arrange your stuff?

↑ stuff you access frequently

↓ stuff you never access
Idea: When you find something
Move it to the top
Idea: When you find something move it to the top.
Idea: When you find something, move it to the top.
Idea: When you find something
Move it to the top

Move me up!
Idea: When you find something
Move it to the top

Move me up!
Idea: When you find something, move it to the top.

Move me up!

I'm at the top.
- Moving what you are looking for to the top by moving up two at a time works pretty well.

- Name of this idea:

  Spley trees

  Sleator + Torijan 84
What if you knew the future?
What if you knew the future?

What if you knew the future?

Searches: A, B, D, B, D, C, C, C, C, L, L
What if you knew the future?

What if you knew the future?

What if you knew the future?

Searches: A, B, D, B, D, C, C, C, L, C
What if you knew the future?

Searches: A, B, D, B, D, C, C, C, C, C

= 12 searches + 3 changes

15 costs
Dynamic Optimality Conjecture

Cost to run splay trees on a sequence is at most something like double the smallest cost possible for that sequence even knowing the future.
Notions of Optimality

\( N = \# \text{ of items in tree} \)

\( X = x_1, x_2, x_3, \ldots, x_m \)  \( \text{Searches} \)

\( R_A(X) = \text{Time to execute } X \text{ using Alg } A \)
Notions of Optimality

\[ N = \text{\# of items in tree} \]

\[ X = x_1, x_2, x_3, \ldots, x_m \quad \text{Searches} \]

\[ R_A(X) = \text{Time to execute } X \text{ using Alg A} \]

Amortized Worst-Case

\[ WC(N) = \min_{A} \lim_{m \to \infty} \frac{\max_{x_1,1 \leq i \leq m} R_A(X)}{m} \]
Notions of Optimality

N = # of items in tree

X = x₁, x₂, x₃ ... xₘ  Searches

$R_A(X) = \text{Time to execute } X \text{ using Alg } A$

Amortized worst-case

$\overline{WC}(N) = \min_{A} \lim_{m \to \infty} \max_{\sum_{i=1}^{m} x_i = m} \frac{R_A(X)}{m} = O(m \log n)$

For BST: $\overline{WC}(N) = \lceil \log_2(N+1) \rceil$
Notions of Optimality

\( N = \# \text{ of items in tree} \)

\( X = x_1, x_2, x_3 \ldots x_m \) \quad \text{Searches}

\( R_A(X) = \text{Time to execute } X \text{ using Alg } A \)

Amortized \quad \text{Worst-Case}

\( WC(N) = \min_A \lim_{m \to \infty} \frac{\max_{x_1, x_2 \ldots x_m} R_A(X)}{m} \)

Instance-Based \quad \text{Optimality}

\( \text{OPT}(X) = \min_A R_A(X) \)
Online vs Offline

\[ X = x_1, x_2, x_3 \ldots x_m \] sequence of operations

Alg A is Online: Executes \( x_i \) based on \( x_1, x_2, \ldots, x_i \) only.

Alg A is Offline: Executes \( x_i \) based on all of \( X \).
Dynamic Optimality

$$\text{OPT}(x) = \min_A R_A(x)$$

Algorithm $A$ is **Dynamically Optimal** if

$$\forall x : R_A(x) = O(\text{OPT}(x))$$
Dynamic Optimality

$$\text{OPT}(X) = \min_{A} R_A(X)$$

Different $A$ for each $X$

Algorithm $A$ is **Dynamically Optimal** if

$$\forall X \ R_A(X) = O(\text{OPT}(X))$$

One $A$ for all $X$
Dynamic Optimality

$$\text{OPT}(X) = \min_{A} R_{A}(X)$$

Different $A$ for each $X$

Algorithm $A$ is **Dynamically Optimal** if

One $A$ for all $X$

$$\forall X \, R_{A}(X) = O(\text{OPT}(X))$$

Interesting even if $A$ is offline

... more interesting if $A$ is online....
If this were true:

Knowing the future is useless.
If at first you don't understand...
If at first you don't understand...

Draw a Different Picture
Search for I
Search for I
Search for L
Search for
Cost to search in tree = Number of X's in this picture
Stare AND Think
Can you find

Stare AND Think
Definition:

No \[ \rightarrow \text{"Arbitrarily Satisfied Set"} \]
Definition:

\[ \text{NO} \quad \rightarrow \quad \text{"Arbitrarily Satisfied set"} \]

\[ \text{ASS} \quad \quad \text{Not ASS} \]
Definition:

$\text{NO } \rightarrow \text{"Arbitrarily Satisfied set"}$

ASS

Not ASS

Theorem: Plotting any way to search for stuff $\rightarrow$ ASS Points
Definition: 

$NO \rightarrow "Arbitrary Satisfied Set"$

What about the reverse way? 
Given an ASS point set like
Can you find a way to find stuff so that this is the plot?

Theorem: Plotting any way to search for stuff

$\rightarrow$ ASS Points
Definition:
NO \rightarrow "Arbitrarily Satisfied Set"

What about the reverse way?
Given an ASS point set like

\[ \text{can you find a way to find stuff so that this is the plot? YES!} \]

Theorem: Plotting any way to search for stuff
\rightarrow ASS Points
So we can forget about

And work with
The best way to find stuff looking for C, A, F, D, B, E
The best way to find stuff
looking for C, A, F, D, B, E
The best way to find stuff looking for C, A, F, D, B, E

What is the minimum number of X's to add to make this ASS?
The best way to find stuff
Looking for C, A, F, D, B, E

What is the minimum number of X's to add to make this ASS?

8 I think
What is the most obvious way to add points to make this 33?
What is the most obvious way to add points to make this ASS?
What is the most obvious way to add points to make this ass?
What is the most obvious way to add points to make this ASS?
What is the most obvious way to add points to make this ass?
What is the most obvious way to add points to make this ass?
What is the most obvious way to add points to make this a 3?
What is the most obvious way to add points to make this chart?
What is the most obvious way to add points to make this ass?
What is the most obvious way to add points to make this ass?
What is the most obvious way to add points to make this ASS?

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Is this the minimum # of x's?
Is this the minimum # of X's?

No, but it appears to be at most double the minimum.
Independent Rectangles

Good

Bad
10

Independent Rectangles

Good

Bad
Independent Rectangles = Lower Bound

# of ind Rects \leq OPT(X)
in geometric view of X
Independent Rectangles = Lower Bound

\[
\text{MAX} \quad \#	ext{ of ind Rects} \leq \text{OPT}(X)
\]

in geometric view of \( X \).
Right only
Right only
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>X</strong></td>
<td></td>
<td><strong>X</strong></td>
</tr>
<tr>
<td><strong>X</strong></td>
<td><strong>X</strong></td>
<td><strong>X</strong></td>
</tr>
<tr>
<td><strong>X</strong></td>
<td><strong>X</strong></td>
<td><strong>X</strong></td>
</tr>
<tr>
<td><strong>X</strong></td>
<td><strong>X</strong></td>
<td><strong>X</strong></td>
</tr>
<tr>
<td><strong>X</strong></td>
<td><strong>X</strong></td>
<td><strong>X</strong></td>
</tr>
</tbody>
</table>

Don't code this.

Right only.
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Right only
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

**Draw Rectangles**

- Red rectangle:
  - Top-left corner: (1, 1)
  - Bottom-right corner: (4, 4)

- Blue rectangle:
  - Top-left corner: (2, 2)
  - Bottom-right corner: (3, 3)

- Blue rectangle:
  - Top-left corner: (3, 3)
  - Bottom-right corner: (4, 4)
Draw Rectangles

They are Independent!
Draw Rectangles

They are Independent!

Thus \( \# \) is a lower bound to make this ASS!
Left only
Right only

Number of D D is less than the minimum X needed to make it ASS
is less than

Minimum X to make it Ass

is less than
Questions

Find minimal ASS superset: Par NC
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**Questions**

- Find minimal ASS superset: Par NPC
- Approximate minimal ASS superset, $O(\log n \cdot \log n)$-approx known but nothing better
Questions

Find minimal ASS superset: Par NEC

Approximate minimal ASS superset, $O(\log^{\alpha} n)$ - approx known but nothing better

Problem is "secondary effects"
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Questions**

- Find minimal ASS superset: Par NC
- Approximate minimal ASS superset, \( O(\log|\mathcal{G}|) \)-approx known but nothing better
- Problem is "secondary effects"
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### Questions

- Find minimal ASS superset: Par NC
- Approximate minimal ASS superset, $O(\log n)$ - approx known but nothing better
- Problem is "secondary effects"
Seems Similar To Minimum Manhattan Network
Seems Similar To Minimum Manhattan Network
Seems Similar To Minimum Manhattan Network

- Measure of interest is # of added junctions
Seems Similar To Minimum Manhattan Network

- Measure of interest is # of added junctions
- But network must treat junctions as points
PART =

Cache Oblivious

+ Persistence
PART 2

Cache Oblivious

+ 

Persistence

with P. Davoodi, O. Ozkan, J. Fineman
Disk Access Model

Memory

Blocks of size $B$

Disk

(size $M$)

(size $\infty$)

at unit cost
Two Examples

Search: B Tree

\[ \log_B N \]
Two Examples

Search: B Tree

Mini: For loop
Two Examples

Search: B Tree

Min: For loop

\[ \log_B N \]

\[ \left\lfloor \frac{N}{B} \right\rfloor \]
Computation Happens Here

Data Fits Here
How do we deal with this?

Data Fits Here
1. Make an algorithm that uses $m, m_2, \ldots, m_7, B_1, B_2, \ldots, B_7$

Data Fits Here

How do we deal with this?
How do we deal with this?

1. Make an algorithm that uses $M_1, M_2, \ldots, M_7, B_1, B_2, \ldots, B_7$.

   - **Yuck**

2. Use a 2-level (DAM) algorithm that does not know $M, B$.

   - **Cache Oblivious**
How do we deal with this?

1. Make alg that uses $M_1, M_2 \ldots M_7; B_1, B_2 \ldots B_7$
   
   \textit{Yuck}

2. Use 2-level (DAM) alg that does not know $M, B$
   
   \textit{Cache Oblivious}

E.g. Scan works well
How Do We Make a O tree with no O?
How Do We Make a D tree with no B?

Search goes through

\[ \frac{\log N}{\log B} \] red trees

0 items

\[ \log B \]

\[ \log N \]
How Do We Make a B tree with no B?

How are these blocks stored?

Search goes through

$\frac{\log N}{\log B}$ red trees
How Do We Make a B tree with no B?

Search goes through

\[ \frac{\log N}{\log B} \text{ red trees} \]

How are these blocks stored?
How Do We Make a D Tree with no B?

Search goes through

\[ \frac{\log N}{\log B} \text{ red trees} \]

How are these blocks stored?
How do you put a tree into memory?
How do you put a tree into memory?
How do you put a tree into memory?
How do you put a tree into memory?

RECURSE
How do you put a tree into memory?

Memory

RECURSE

2 Blocks $\sqrt{N}$
4 Blocks $4\sqrt{N}$
...
$\log_2 N$ Blocks $B$
ULTIMATE LOCALITY
Lots of results in
Cache – Oblivious Model
Persistence
Persistence

See the past
Persistence
See the past

Operation
  Query  Read Only
  Update  Read/Write
Types of Persistence

Partial  Full  Confluent  Retroactive
Types of Persistence

Partial  Full  Confluent  Retroactive

- Can Update most recent version
- Query at any point in the past
- Time is linear
Types of Persistence

Partial

[ST86]

Full

[DSSST89]

Confluent

[PRO3] [OC12]

Retroactive

[DL07]

Update

Update

Update

Update

Update

Update

Update

Query

Update

Update

Update

Update

Update

Update
Persistence - Results

For pointer-based structures of constant indegree:

- Partial, full persistence is free
- Confluent persistence is complicated
- Retroactive is impossible
Persistence — Results

For pointer-based structures of constant indegree

- Partial, full persistence is free
- Confluent persistence is complicated
- Retroactive is impossible

But we care about the cache-oblivious model

C-O = locality    Pointer = anti-locality
So, again, what is the C-O model?
So, again, what is the C-O model?

From point of view of an alg:

- Memory is an Array
- Read
- Write
So, again, what is the C-O model?

From point of view of an alg:

- Memory is an Array
- Read
- Write

So we need a persistent array that maintains locality
Geometric View

Store in obvious way: No Locality
Copy memory at every time: Horrible but has locality

A[8] = 4
A[15] = 0
A[1] = 8
i = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Query in past

16

4

5

0
Store as a 3-way tree
Each k-block is stored in $3^{\lceil \log_3 k \rceil} = k^{\lceil \log_2 3 \rceil}$ space
Suppose I wanted to look at a block of size $\Theta$ in memory at some time.
Suppose I wanted to look at 1 block of size $\Theta$ in memory at some time.
Suppose I wanted to look at a block of size $\Theta^{1/\alpha_{2.3}}$ in memory at some time.

A block of size $B^{1/\alpha_{5.2.3}} \approx 0.63$

In the past can be found in 2 ranges of size $B$ in memory.
Suppose I wanted to look at 1 block of size $\Theta^{1/\log 2^3}$ in memory at some time.

$T(B) =$ Runtime of Alg A in C-O model with block size $B$

THM: Persistent queries take time $O(T(\Theta^{1/\log 2^3}))$

A block of size $B^{1/10923} \approx 0.63$

In the past can be found in 2 ranges of size $B$ in memory.
Suppose I wanted to look at 1 block of size $\frac{Y_{10523}}{3}$ in memory at some time.

$T(B) =$ Runtime of Alg A in C-O model with Block size B

THM: Persistent queries take time $O(T(B^{(1-\epsilon)/10523})\log B)$

In the past can be found in 2 ranges of size B in memory.
Suppose I wanted to look at 1 block of size $B^{\sqrt{10^{0.63}}}$ in memory at some time.

$T(B) =$ runtime of Alg A in C-O model with block size $B$.

**THM:** Persistent queries take time $O(T(B^{(1-\varepsilon)/10^{0.63}}) \log B S)$

A block of size $B^{\sqrt{10^{0.63}}} \approx 0.63$ in the past can be found in 2 ranges of size $B$ in memory.

$O(\log B N) \rightarrow O(\log B N)$ (worse than VCR OR)

$O(N/B) \rightarrow O(N \log B N / B^{0.63})$.
Notes

- I've ignored $M$
- Log $S$ space blowup
- Full persistence probably possible
Thoughts

The C-R model is a torturous way to get algs with great locality.

Better way?

Geometry?
Part III: Forbidden Matrices
Idea: Use results on extremal functions of 0-1 matrices with forbidden submatrices
Idea: Use results on extremal functions of 0-1 matrices with forbidden submatrices

\[
\begin{array}{|c|c|c|}
\hline
1 & 0 & 1 \\
\hline
0 & 1 & 1 \\
\hline
\end{array}
\]

\[\text{has } O(n \log n) \text{ } 1\text{s}\]

[\text{G. Tardos 05}]
Method

Data Structure
Method

Data Structure

Geometric View

Find Forbidden Submatrices
Method

Data Structure

Geometric View

Find Forbidden Submatricies
Method

Data Structure

Geometric View

Find Forbidden Submatricies

Compute/look up extremal Function
Method

Data Structure

Geometric View

Find Forbidden Submatricies

Runtime Bound!

Compute/look up extremal Function
Example: Union-Find w Path Compression.

\[ \{GF \text{64}, HU \text{73, T75} \} \]

- Who is my root
- Union
Example: Union-Find w Path Compression.

- Who is my root
- Combine
Example: Union-Find w Path Compression.

- Who is my root
- Combine

Union
Example: Union-Find w Path Compression.

- Who is my root
- Combine
Example: Union-Find w Path Compression.

- Who is my root
- Combine
Example: Union-Find w Path Compression.

- Who is my root
- Combine

↑ who is my root
Example: Union-Find with Path Compression.

- Who is my root
- Combine

↑ who is my root

COMPRESSION!
Plot Geometric View

Data

Observe

Forbidden

Thus $O(n \log n)$

I if touched during a compression
Technique seems powerfull
Technique seems powerful.

Yet most results are simplifications.
Another Example

Restricted → Incremental

A problem having to do with Geometric View

Binary Search Trees

Forbidden Matrix

complicated Bound!

[Pettie 10]
Voronoi Diagrams

- Incremental
- Add hull points in clockwise order
Flarb Oil bag

Add Root

new

Flarb!

Cost is # of children who have had their parent change
Geometric View

Key Value

\( x = \text{parent changed} \)

Pei-Tie's Observation

\[ \text{No as a submatrix} \]

\[ \#x's = O(N^{\log N}) \]
Forbidden Pattern Bound on algorithm in the search to make a satisfied set
Pattern Avoidance in BSTs
Preorder Conjecture
Preorder Conjecture
Preorder Conjecture

Diagram with nodes and grid.
Preorder Conjecture
Preorder Conjecture
Preorder Conjecture
Preorder Conjecture
Preorder Conjecture
Preorder Conjecture
Preorder Conjecture
Preorder Conjecture

Can you find?
Preorder Conjecture

Can you find?
Dynamic Opt \( \rightarrow \) Preorder takes \( O(N) \) time
Pattern Avoidance in BST

No submatrix

→ Greedy’s Runtime is $\leq n^2 \alpha(n) o(1)$

Chulermsook Goswami Ko2m Melhorn Saranurak 2015
Pattern Avoidance in BST

Generalize!

No sub matrix

\[ \text{any } k \times k \text{ permutation} \]

\[ \rightarrow \text{Greedy's runtime is } O(k) \]

\[ n^2 \alpha(n) \]
Recursive Decompositions
Recursive Decompositions

3 - decomposable
Recursive Decompositions

3 - decomposable
Recursive Decompositions

recursively
3 - decomposable
Recursive Decompositions

Recursively decomposable

3 - decomposable

Recursively k-decomposable

→ Greedy* adds:

\[ n \theta(nk^2) \]

[\text{Chatterjee, Goswami, Kozma, Melhorn, Saranurak} 2015]
Recursive Decompositions

Recursively K-decomposable

Greedy* adds:

$\Omega(k^2)$

$\Omega(n \log K)$

Some 2017 using [Lauers Langerman]"
More Applications?
Possible Future Directions

- Geometry of other fundamental DS's, E.g. Heaps
- Modeling other models geometrically?
- Other forbidden matrix applications e.g., compact encoding of geometric objects.
The End
ULB Algorithms Group Has Multiple Postdocs!

Jean Cardinal
Elena Khramtcova
Till Miltzan
Stefan Langerman
John Iacono
Audien Ooms
Gwen Javé
Sam Fiorini