Multi-finger
Binary Search Trees

Parinya Chalermsook    Mayank Goswami
László Kozma    Kurt Mehlhorn    Thatchaphol Saranurak
Binary Search Tree (BST)

$n$ keys

search cost = depth + 1

Search sequence:
$x_1, x_2, x_3, \ldots$

Tree can be adapted using rotations, to prepare for next search.

cost = pointer moves + rotations

- offline cost: **OPT**
- online algorithms: e.g. **Splay tree**
  
  [Sleator, Tarjan, 1983]
Finger Search

Search starts from previous location.

Cost = amount of movement by finger.

Optimal finger search cost: $OPT_1$

Tree is static, chosen optimally.

search(2)
search(5)
search(7)
...

Search starts from previous location.
Multiple fingers

k fingers, stationed at nodes
search starts from any of the fingers

To serve request, must move some finger there
Tree is static, chosen optimally

cost = amount of movement by all fingers

optimal k-finger search cost: \( \text{OPT}_k \)
Multiple fingers

\[ OPT_1 \geq OPT_2 \geq \cdots \geq OPT_k \geq \cdots \geq OPT_n \]

matched by online BST with rotations

[Cole et al., 2000]
[Iacono-Langerman, 2016]

Our result

matched by online BST with rotations (with small overhead)

\[ OPT \leq O(OPT_1) \]

\[ OPT \leq O(OPT_k) \cdot \log k \]
Our results

1. BSTs (with rotations) can simulate the $k$-finger optimum with small overhead:

   $$\text{OPT} \leq O(\text{OPT}_k) \cdot \log k$$

   log(k) factor is optimal.

2. There is an online BST with cost:

   $$O(\text{OPT}_k) \cdot \log^7 k$$
The proof has three ingredients:

1. Simulate k-finger strategy by BST with single root-pointer (and rotations)

2. Find online k-finger strategy

3. Learn optimal underlying tree
1. Simulate k-finger strategy by BST
with optimal $O(\log k)$ overhead,
refining previous approach with overhead $O(k)$

[Demaine, Iacono, Langerman, Özkan, 2013]

**Idea:** store nodes with fingers as a balanced subtree

binary search tree  

finger tree
finger tree

O(k) fingers and pseudofingers, can store as e.g. AVL-tree

tendons: almost sorted, can update in O(1) amortized time
**BST simulation of finger tree**

- fingers and pseudofingers stored here e.g. AVL-tree
- tendons and subtrees stored here
- tendons stored as BST deque

**Operations**

- **access finger**  -->  search AVL tree ... $O(\log k)$
- **move finger**   -->  update connected tendons
  - insert/delete in deque ...  $O(1)$
  - update node role as (pseudo)finger;
  - may trigger AVL re-balancing ...  $O(\log k)$
2. Find online k-finger strategy

Goal: serve requests, minimize total movement

k-server problem

Our task is the special case when the metric is a BST

Relevant result: $O(\log^6 k)$ competitive online algorithm [Lee; Bubeck et al. 2018]
3. Learn best underlying tree

Search sequence:
\[x_1, x_2, x_3, \ldots\]

- **Idea:** Process in epochs of length \(O(n \log n)\)
- Maintain distribution over all \(\sim 4^n\) trees

When epoch starts:
- pick a tree from distribution, rotate to it
  (rotation cost amortized over epoch)

When epoch ends:
- evaluate all trees, update distribution using multiplicative-weights-update (MWU)

\[
\text{Loss of tree } T = \text{cost during epoch if we used } T
\]
MWU guarantee

overall cost < \((1 + \varepsilon)T^* + \frac{\ln N \cdot T^{\text{max}}}{\varepsilon}\)

- Max possible cost within epoch \(\sim n^2\)
- Nr. of possible trees \(\sim 4^n\)
- Total cost with best tree in hindsight
- Poly(n) additive term
Summary of main result

1. **Simulate** $k$-finger strategy by BST with single root-pointer (and rotations)
   - $O(\log k)$ factor loss
   - Refines technique of [Demaine et al.]

2. Find **online** $k$-finger strategy
   - $O(\log^6 k)$ factor loss
   - First connection between $k$-server and BST?

3. **Learn** optimal underlying tree
   - $O(1+\epsilon)$ factor, additive term
   - MWU technique, randomized
Applications

1. Better understand limitations of BST model

[Sleator, Tarjan, 1983]:
Splay tree cost is $O(OPT)$

Dynamic Optimality Conjecture

From our results:
then it must be $O(OPT_k \log k)$
(even $k=1$ case is hard to prove)

barrier to Dynamic Optimality
Applications

2. Understand which search sequences are easy

Example:

k-monotone sequence
(shuffle of k runs)

k-finger strategy:
each finger serves one run
(O(n) traversal of tree)
total cost: \( O(nk) \)

=> BST cost: \( O(n \log k) \)
Applications

2. Understand which search sequences are easy

**Example:**
weak unified bound
(search is close to a recent search)

amortized cost for search($x_k$):

$$\min_t \{ \log \left| x_k - x_{k-t} \right| \} \cdot f(t)$$

- rank-diff. from recent search
- penalty factor for looking back too far

we give a k-finger strategy (not so easy)
OPEN QUESTIONS

1. De-randomize our online BST
(randomness comes from picking tree in MWU strategy)
   Possible randomized/deterministic separation?

2. Towards dynamic optimality
   Show that Splay tree matches \( O(\text{OPT}_k) \ f(k) \)
   Show that some online BST matches \( O(\text{OPT}_k) \ \log k \)
   (another \( \log(k) \) factor unavoidable if online k-server is used)