Self-Adjusting Binary Search Trees: Recent Results

Talk based on papers in WADS 2015, ESA 2015, FOCS 2015, and unpublished work.

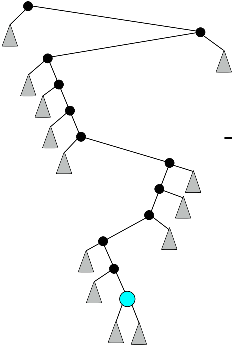
Parinya Chalermsook[1]
Mayank Goswami[2]
László Kozma[3]
Kurt Mehlhorn
Thatchaphol Saranurak[4]

max planck institut
informatik

January 15, 2017

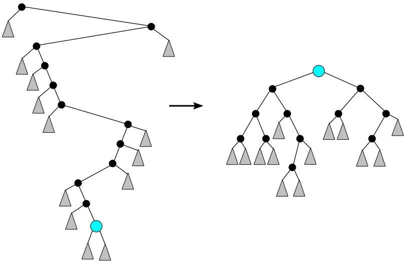
[1] Aalto University, Helsinki
[2] CUNY, New York
[3] Tel Aviv University, Tel Aviv
[4] KTH, Stockholm

Binary Search Trees (BSTs)



- A search for x in a binary search tree walks down a path. If x is equal to the key stored in the current node, we have found x. If x is smaller than the key stored in the node, we go left. If x is larger than the key stored in the node, we go right.
- Different flavors of BSTs:
  - static BSTs
  - balanced BSTs: AVL-trees, 2-4-trees, red-black trees, ...
  - self-adjusting BSTs: Splay trees.

The Self-Adjusting BST-Model



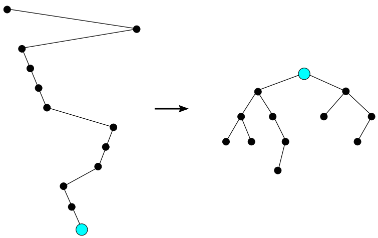
- After an access, replace the search path by an arbitrary tree (the after-tree) on the same set of nodes rooted at the accessed element.
- Reattach the dangling subtrees (uniquely defined).
- **Cost = length of search path.**

Question: Which re-arrangements lead to an efficient online algorithm?

OPT = cost of the offline optimum.

OPT knows the entire access sequence in advance and can act accordingly. The online algorithm has to rebuild without knowing future accesses.

The Self-Adjusting BST-Model



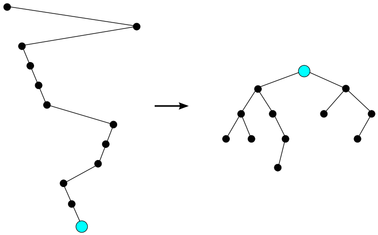
- After an access, replace the search path by an arbitrary tree (the after-tree) on the same set of nodes rooted at the accessed element.
- Reattach the dangling subtrees (uniquely defined).
- **Cost = length of search path.**

Question: Which re-arrangements lead to an efficient online algorithm?

OPT = cost of the offline optimum.

OPT knows the entire access sequence in advance and can act accordingly. The online algorithm has to rebuild without knowing future accesses.

The Self-Adjusting BST-Model



- After an access, replace the search path by an arbitrary tree (the after-tree) on the same set of nodes rooted at the accessed element.
- Reattach the dangling subtrees (uniquely defined).
- **Cost = length of search path.**

Question: Which re-arrangements lead to an efficient online algorithm?

OPT = cost of the offline optimum.

OPT knows the entire access sequence in advance and can act accordingly. The online algorithm has to rebuild without knowing future accesses.

The Dynamic Optimality Conjecture (Sleator/Tarjan '85)

Splay trees are $O(1)$-competitive, i.e., for every access sequence X, the cost of serving X by splay trees is at most a constant factor larger than serving X optimally.

A path towards proving or disproving the conjecture:

- Understand better which variants of splay trees might also work.
- Show special cases of the dynamic optimality conjecture.
- Exhibit easy sequences, i.e., sequences which OPT serves in time $o(n \log n)$.

ESA-paper addresses the first item.

Self-Adjusting BSTs: What Makes them Tick? (ESA 2015)

Splay trees have many nice properties, e.g.,

- Logarithmic access cost — Static optimality
- Working set property — Static finger property
- Sequential access — Dynamic finger property

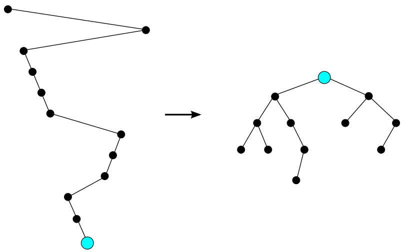
We give sufficient (and necessary) conditions for the first four properties.

Previous work: Sleator, Tarjan, Subramanian, Georgakopoulos, McClurkin prove first four properties for splay-trees and variants thereof.

All of these results are corollaries of the main theorem in the ESA paper. Also prove new results about depth-halving.

Main Result

Characteristic quantities of the search path and the after-tree.



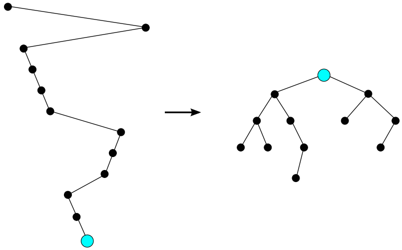
- length of the search path: $|P|$ 12
- number of side changes: z 4
- number of leaves: ℓ 5
- max left-depth of left subtree (max right-depth of right subtree): d 3

Theorem: If accessed element goes to root, $d = O(1)$, and $\ell = \Omega(|P| - z)$, then the BST has the first four properties.

We also have a partial converse (more later).

Main Result

Characteristic quantities of the search path and the after-tree.



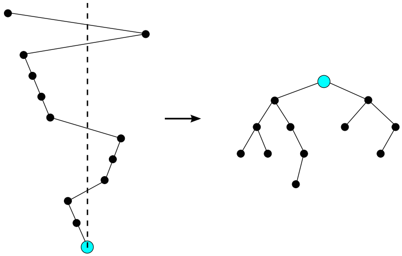
- length of the search path: $|P|$ 12
- number of side changes: z 4
- number of leaves: ℓ 5
- max left-depth of left subtree (max right-depth of right subtree): d 3

Theorem: If accessed element goes to root, $d = O(1)$, and $\ell = \Omega(|P| - z)$, then the BST has the first four properties.

We also have a partial converse (more later).

Main Result

Characteristic quantities of the search path and the after-tree.



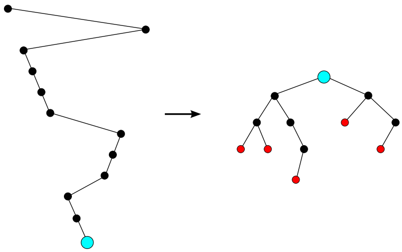
- length of the search path: $|P|$ 12
- number of side changes: z 4
- number of leaves: ℓ 5
- max left-depth of left subtree (max right-depth of right subtree): d 3

Theorem: If accessed element goes to root, $d = O(1)$, and $\ell = \Omega(|P| - z)$, then the BST has the first four properties.

We also have a partial converse (more later).

Main Result

Characteristic quantities of the search path and the after-tree.



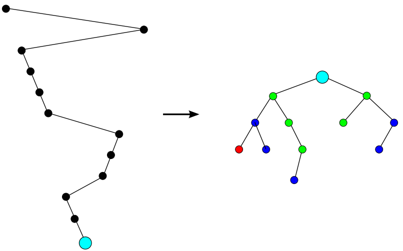
- length of the search path: $|P|$ 12
- number of side changes: z 4
- number of leaves: ℓ 5
- max left-depth of left subtree (max right-depth of right subtree): d 3

Theorem: If accessed element goes to root, $d = O(1)$, and $\ell = \Omega(|P| - z)$, then the BST has the first four properties.

We also have a partial converse (more later).

Main Result

Characteristic quantities of the search path and the after-tree.



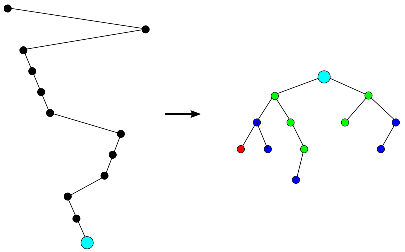
- length of the search path: $|P|$ 12
- number of side changes: z 4
- number of leaves: ℓ 5
- max left-depth of left subtree (max right-depth of right subtree): d 3

Theorem: If accessed element goes to root, $d = O(1)$, and $\ell = \Omega(|P| - z)$, then the BST has the first four properties.

We also have a partial converse (more later).

Main Result

Characteristic quantities of the search path and the after-tree.



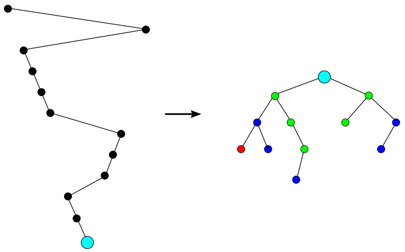
- length of the search path: $|P|$ 12
- number of side changes: z 4
- number of leaves: ℓ 5
- max left-depth of left subtree (max right-depth of right subtree): d 3

Theorem: If accessed element goes to root, $d = O(1)$, and $\ell = \Omega(|P| - z)$, then the BST has the first four properties.

We also have a partial converse (more later).

Main Result

Characteristic quantities of the search path and the after-tree.

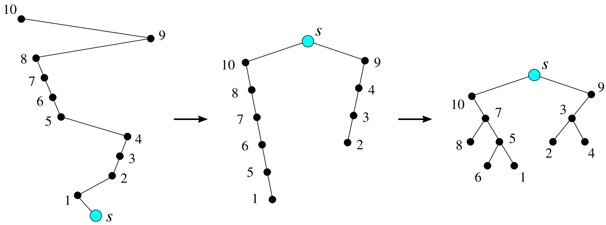


- length of the search path: $|P|$ 12
- number of side changes: z 4
- number of leaves: ℓ 5
- max left-depth of left subtree (max right-depth of right subtree): d 3

Theorem: If accessed element goes to root, $d = O(1)$, and $\ell = \Omega(|P| - z)$, then the BST has the first four properties.

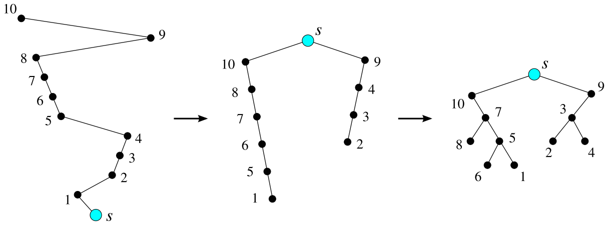
We also have a partial converse (more later).

Application I: Splay Trees



- Split the search path at *s* and swap adjacent odd-even pairs.
- **This is a global view on splay trees; seems to be new.**

Application I: Splay Trees

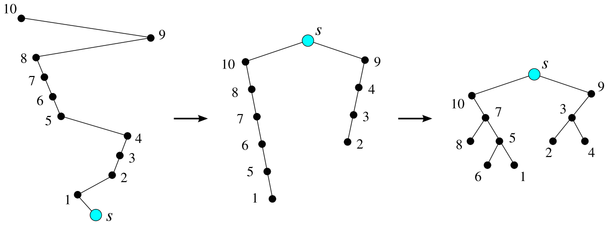


- accessed element becomes root
- max right-(left) depth is $d = 2$
- $z + \ell \geq |P|/2 - 1$

  Proof: There are $|P|/2 - 1$ odd-even pairs. Each side change can move the elements of one pair to different sides. Each odd-even pair on the same side creates a leaf. Thus

$$\text{\# of leaves} \geq |P|/2 - 1 - \text{\# of side changes}$$

Application I: Splay Trees

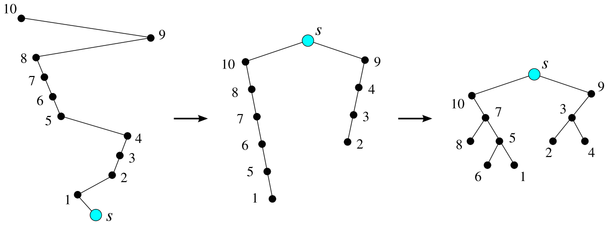


- accessed element becomes root
- max right-(left) depth is $d = 2$
- $z + \ell \geq |P|/2 - 1$

  Proof: There are $|P|/2 - 1$ odd-even pairs. Each side change can move the elements of one pair to different sides. Each odd-even pair on the same side creates a leaf. Thus

$$\# \text{ of leaves} \geq |P|/2 - 1 - \# \text{ of side changes}$$

Application I: Splay Trees

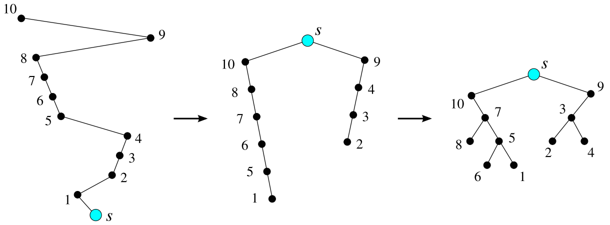


- accessed element becomes root
- max right-(left) depth is $d = 2$
- $z + \ell \geq |P|/2 - 1$

  Proof: There are $|P|/2 - 1$ odd-even pairs. Each side change can move the elements of one pair to different sides. Each odd-even pair on the same side creates a leaf. Thus

$$\text{\# of leaves} \geq |P|/2 - 1 - \text{\# of side changes}$$

Application I: Splay Trees



- accessed element becomes root
- max right-(left) depth is $d = 2$
- $z + \ell \geq |P|/2 - 1$

  Proof: There are $|P|/2 - 1$ odd-even pairs. Each side change can move the elements of one pair to different sides. Each odd-even pair on the same side creates a leaf. Thus

  $$\text{\# of leaves} \geq |P|/2 - 1 - \text{\# of side changes}$$

Depth Halving

In splay every node on the search path roughly halves its depth.

Sleator: is this property sufficient?

We don't know, but strict depth-halving is sufficient: the accessed element becomes the root and every node x on the search path loses at least $(1/2 + \epsilon)d(x) - O(1)$ ancestors and gains at most $O(1)$ new descendants.

Depth Halving

In splay every node on the search path roughly halves its depth.

Sleator: is this property sufficient?

We don't know, but strict depth-halving is sufficient: the accessed element becomes the root and every node x on the search path loses at least $(1/2+\epsilon)d(x) - O(1)$ ancestors and gains at most $O(1)$ new descendants.

Partial Converses

If the after-tree may have non-constant left-depth or right-depth, then the good properties (logarithmic access, static optimality, ...) cannot be shown with the sum-of-logs potential function.

If the number of leaves of the after-tree is allowed to be $o(|P| - \text{number of side changes})$, then the traversal conjecture does not hold.

Partial Converses

If the after-tree may have non-constant left-depth or right-depth, then the good properties (logarithmic access, static optimality, ...) cannot be shown with the sum-of-logs potential function.

If the number of leaves of the after-tree is allowed to be $o(|P| -$ number of side changes), then the traversal conjecture does not hold.

The Dynamic Optimality Conjecture (Sleator/Tarjan ’85)

Splay trees are $O(1)$-competitive, i.e., for every access sequence X, the cost of serving X by splay trees is at most a constant factor larger than serving X optimally.

A path towards proving or disproving the conjecture:

- Understand better which variants of splay trees might also work.
- Show special cases of the dynamic optimality conjecture.
- Exhibit additional easy sequences, i.e., sequences which OPT serves in time $o(n \log n)$.

FOCS-paper addresses items 2 and 3.

Traversal conjecture: Let X be the preorder traversal of a tree T. Process X starting with a tree T'. OPT = $O(n)$.

Only shown for $T' = T$ or $X = 1, 2, \ldots, n$.

Pattern Avoiding Accesses (FOCS 2015)

An access sequence X avoids a pattern P if there is no subsequence of X that is order-isomorphic to P.

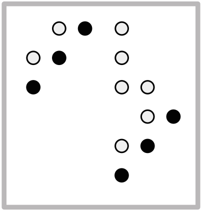
- $X = 1, 2, \ldots, n$ avoids $2, 1$.
- Preorder traversal of a tree avoids $2, 3, 1$.
- Special cases of the optimality conjecture.
  - GREEDY serves any sequence that avoids a permutation pattern of size k with cost $O(2^{\alpha(n)^{O(k^2)}} \cdot n)$.
  - GREEDY with chosen initial tree serves any such sequence with cost $O(2^{O(k^2)} \cdot n)$.
  - Traversal conjecture: $k = 3$.
- New easy sequences.
  - OPT serves any k-decomposable sequence with cost $O(n \log k)$.

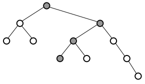
Pattern Avoiding Accesses (FOCS 2015)

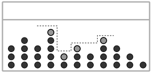
An access sequence X avoids a pattern P if there is no subsequence of X that is order-isomorphic to P.

- $X = 1, 2, \ldots, n$ avoids $2, 1$.
- Preorder traversal of a tree avoids $2, 3, 1$.
- Special cases of the optimality conjecture.
  - GREEDY serves any sequence that avoids a permutation pattern of size k with cost $O(2^{\alpha(n)^{O(k^2)}} \cdot n)$.
  - GREEDY with chosen initial tree serves any such sequence with cost $O(2^{O(k^2)} \cdot n)$.
  - Traversal conjecture: $k = 3$.
- New easy sequences.
  - OPT serves any k-decomposable sequence with cost $O(n \log k)$.

Satisfied Point Sets



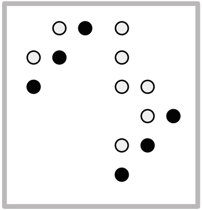




- M = a $\{0,1\}$-matrix (a point set).
- Ignore the colors for the moment.
- $\Box_{pq}$ = closed rectangle with corners p and q.
- M is satisfied if for any two points $p, q \in M$ with distinct x and y coordinates there is another point from M in the rectangle.
- Access sequence $X \to$ matrix X.

  Point $(x, t) \in X$ iff the element x is accessed at time t .
- A tree T gives rise to a matrix T.

Geometric BSTs (Demaine, Harmon, Iacono, Kane, Patrascu (SODA '09))



Cost = 8

Geometric BST $\mathcal{A}$ on input $\left[\begin{smallmatrix} X \\ T \end{smallmatrix}\right]$ outputs a satisfied matrix $\left[\begin{smallmatrix} \mathcal{A}_T(X) \\ T \end{smallmatrix}\right]$, where $\mathcal{A}_T(X) \supseteq X$.

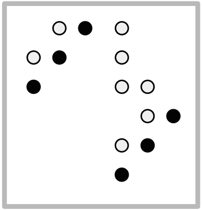
Chosen initial tree: On input X outputs $\mathcal{A}(X)$.

Cost = number of points (ones) in $\mathcal{A}_T(X)$.

Offline versus online

Theorem(DHIKP): (online) geometric-BSTs = (online) BSTs.

Geometric BSTs (Demaine, Harmon, Iacono, Kane, Patrascu (SODA '09))



Cost = 8

Geometric BST $\mathcal{A}$ on input $\left[\frac{X}{T}\right]$ outputs a satisfied matrix $\left[\frac{\mathcal{A}_T(X)}{T}\right]$, where $\mathcal{A}_T(X) \supseteq X$.

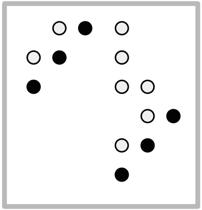
Chosen initial tree: On input X outputs $\mathcal{A}(X)$.

Cost = number of points (ones) in $\mathcal{A}_T(X)$.

Offline versus online

Theorem(DHIKP): (online) geometric-BSTs = (online) BSTs.

Geometric BSTs (Demaine, Harmon, Iacono, Kane, Patrascu (SODA '09))



Cost = 8

Geometric BST $\mathcal{A}$ on input $\left[\begin{smallmatrix} X \\ T \end{smallmatrix}\right]$ outputs a satisfied matrix $\left[\begin{smallmatrix} \mathcal{A}_T(X) \\ T \end{smallmatrix}\right]$, where $\mathcal{A}_T(X) \supseteq X$.

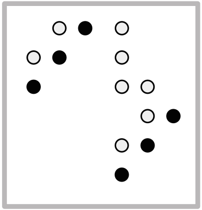
Chosen initial tree: On input X outputs $\mathcal{A}(X)$.

Cost = number of points (ones) in $\mathcal{A}_T(X)$.

Offline versus online

Theorem(DHIKP): (online) geometric-BSTs = (online) BSTs.

Geometric BSTs (Demaine, Harmon, Iacono, Kane, Patrascu (SODA '09))



Cost = 8

Geometric BST $\mathcal{A}$ on input $\left[\begin{smallmatrix} X \\ T \end{smallmatrix}\right]$ outputs a satisfied matrix $\left[\begin{smallmatrix} \mathcal{A}_T(X) \\ T \end{smallmatrix}\right]$, where $\mathcal{A}_T(X) \supseteq X$.

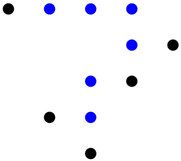
Chosen initial tree: On input X outputs $\mathcal{A}(X)$.

Cost = number of points (ones) in $\mathcal{A}_T(X)$.

Offline versus online

Theorem(DHIKP): (online) geometric-BSTs = (online) BSTs.

Greedy (Lucas, Munro, DHIKP)



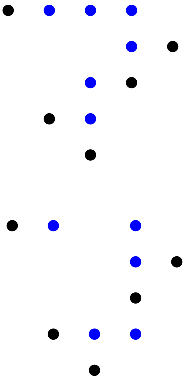
Accessed points in black.

Points added by Greedy in blue.

- After access to x at time t add exactly the (y, t) that are needed for satisfaction.
- Greedy is not optimal.
- Conjecture (Lucas, Munro, DHIKP): Greedy is $O(1)$-competitive.
- Theorem: Greedy almost satisfies traversal conjecture (cost $n \cdot 2^{\alpha(n)^{O(1)}}$).

  Greedy with chosen initial tree satisfies traversal conjecture.

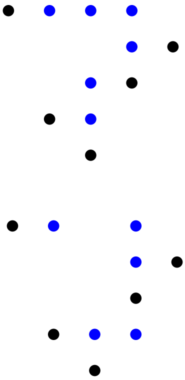
Greedy (Lucas, Munro, DHIKP)



- After access to x at time t add exactly the (y, t) that are needed for satisfaction.
- Greedy is not optimal.
- Conjecture (Lucas, Munro, DHIKP): Greedy is $O(1)$-competitive.
- Theorem: Greedy almost satisfies traversal conjecture (cost $n \cdot 2^{\alpha(n)^{O(1)}}$).

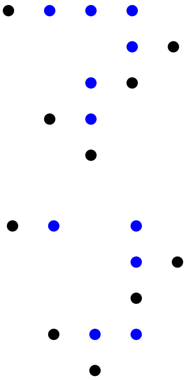
  Greedy with chosen initial tree satisfies traversal conjecture.

Greedy (Lucas, Munro, DHIKP)



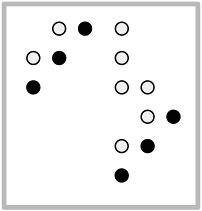
- After access to x at time t add exactly the (y, t) that are needed for satisfaction.
- Greedy is not optimal.
- Conjecture (Lucas, Munro, DHIKP): Greedy is $O(1)$-competitive.
- Theorem: Greedy almost satisfies traversal conjecture (cost $n \cdot 2^{\alpha(n)^{O(1)}}$). Greedy with chosen initial tree satisfies traversal conjecture.

Greedy (Lucas, Munro, DHIKP)

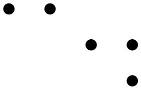


- After access to x at time t add exactly the (y, t) that are needed for satisfaction.
- Greedy is not optimal.
- Conjecture (Lucas, Munro, DHIKP): Greedy is $O(1)$-competitive.
- Theorem: Greedy almost satisfies traversal conjecture (cost $n \cdot 2^{\alpha(n)^{O(1)}}$).
  Greedy with chosen initial tree satisfies traversal conjecture.

Forbidden Matrix Theory



contains



Let M and P be $n \times n$ and $k \times k$ matrices s.t. M avoids P.

- (Marcus, Tardos, Fox): If P is a permutation matrix, the number of ones in M is at most $n2^{O(k)}$.
- (Klasar, Keszegh): If P is light (only one 1 per column), the number of ones in M is at most $n2^{\alpha(n)^{O(k^2)}}$.

S. Pettie pioneered the use of forbidden matrix theory for the study of data structures.

Greedy and Forbidden Matrices I

Theorem

If the access sequence X avoids a pattern P, then for any initial tree T, $\textsc{Greedy}(X)$ avoids $P \otimes \mathtt{Cap}$, where $\mathtt{Cap} = \begin{pmatrix} & \bullet & \\ \bullet & & \bullet \end{pmatrix}$.

For $P = \begin{pmatrix} \bullet & & \\ & & \bullet \\ & \bullet & \end{pmatrix}$, $P \otimes \mathtt{Cap} = \begin{pmatrix} & \bullet & & & & & & & \\ \bullet & & \bullet & & & & & & \\ & & & & & & & \bullet & \\ & & & & & & \bullet & & \bullet \\ & & & & \bullet & & & & \\ & & & \bullet & & \bullet & & & \end{pmatrix}$.

Proof: Assume that at time t columns a and b are touched. If all accesses after time t are to columns $\leq a$ or $\geq b$, then columns $a+1$ to $b-1$ will not be touched after time t.

Thus, if $\textsc{Greedy}(X)$ contains a point in $[a+1, \ldots, b-1] \times [t+1, \ldots]$, X must contain a point in this set.

In particular, if $\textsc{Greedy}(X)$ contains $P \otimes$ *capture* then X contains P.

Greedy and Forbidden Matrices I

Theorem

If the access sequence X avoids a pattern P, then for any initial tree T, $\textsc{Greedy}(X)$ avoids $P \otimes \mathtt{Cap}$, where $\mathtt{Cap} = \begin{pmatrix} & \bullet & \\ \bullet & & \bullet \end{pmatrix}$.

For $P = \begin{pmatrix} \bullet & & \\ & & \bullet \\ & \bullet & \end{pmatrix}$, $P \otimes \mathtt{Cap} = \begin{pmatrix} & \bullet & & & & & & & \\ \bullet & & \bullet & & & & & & \\ & & & & & & & \bullet & \\ & & & & & & \bullet & & \bullet \\ & & & & \bullet & & & & \\ & & & \bullet & & \bullet & & & \end{pmatrix}$.

For preorder sequence X, $\textsc{Greedy}(X)$ avoids a light 6×9 matrix.

Theorem

$\textsc{Greedy}$ *with arbitrary initial tree serves preorder sequences with cost $n \cdot 2^{\alpha(n)^{O(1)}}$.*

Greedy and Forbidden Matrices II

Theorem

If access sequence X avoids a pattern P, then $\textsc{Greedy}(X)$ with chosen initial tree avoids $P \otimes P'$, where P' is a particular permutation matrix (of the same size as P).

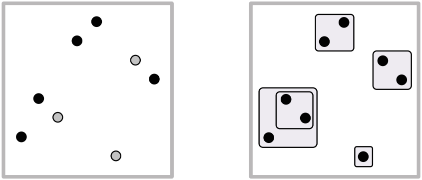
This proof is more involved.

For $P = \begin{pmatrix} \bullet & & \\ & & \bullet \\ & \bullet & \end{pmatrix}$, $P \otimes P'$ is 9×9.

Theorem

Greedy with chosen initial tree satisfies traversal conjecture.

Decomposable Permutations



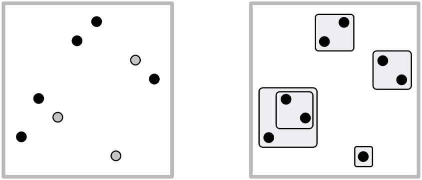
A 4-decomposable permutation.

- k-decomposable = avoid all non-decomposable permutations of size $k + 1$ or more.
- OPT serves k-decomposable permutations with cost $O(n \log k)$.

A new challenge!!!

Proof Technique: We introduce an offline variant of GREEDY and analyse its behavior on k-decomposable permutations.

Decomposable Permutations



A 4-decomposable permutation.

- k-decomposable = avoid all non-decomposable permutations of size $k+1$ or more.
- OPT serves k-decomposable permutations with cost $O(n \log k)$. A new challenge!!!

Theorem

- GREEDY *serves k-decomposable sequences with cost $O(n2^{\alpha(n)^{O(k^2)}})$.*
- GREEDY *with chosen initial tree matches performance of OPT.*

Iacono-Langermann (SODA 2016): Dynamic Finger Property

GREEDY has dynamic finger property, i.e., total search cost is $O(\sum_i \log |x_i - x_{i-1}|)$.

Cole has previously proven dynamic finger property for splay trees (80 page paper, complex proof).

Proof for GREEDY is 10 pages and easy to check.

Summary

- A wide class of BSTs with logarithmic access cost, static optimality, working set and static finger property.
- GREEDY does well on inputs that avoid patterns. In particular,
  - Traversal conjecture almost holds for GREEDY.
  - Traversal conjecture holds for GREEDY with chosen initial tree.
- New challenges for self-adjusting BSTs: OPT serves any sequence that can be decomposed into k monotone sequences with cost $O(n \log k)$.
- Next steps:
  - Show that GREEDY does well on k-monotone sequences
  - Traversal conjecture for arbitrary initial tree.

Summary

- A wide class of BSTs with logarithmic access cost, static optimality, working set and static finger property.
- GREEDY does well on inputs that avoid patterns. In particular,
  - Traversal conjecture almost holds for GREEDY.
  - Traversal conjecture holds for GREEDY with chosen initial tree.
- New challenges for self-adjusting BSTs: OPT serves any sequence that can be decomposed into k monotone sequences with cost $O(n \log k)$.
- Next steps:
  - Show that GREEDY does well on k-monotone sequences
  - Traversal conjecture for arbitrary initial tree.

Summary

- A wide class of BSTs with logarithmic access cost, static optimality, working set and static finger property.
- GREEDY does well on inputs that avoid patterns. In particular,
  - Traversal conjecture almost holds for GREEDY.
  - Traversal conjecture holds for GREEDY with chosen initial tree.
- New challenges for self-adjusting BSTs: OPT serves any sequence that can be decomposed into k monotone sequences with cost $O(n \log k)$.
- Next steps:
  - Show that GREEDY does well on k-monotone sequences
  - Traversal conjecture for arbitrary initial tree.