2015 IEEE 56th Annual Symposium on Foundations of Computer Science

Pattern-avoiding access in binary search trees

Parinya Chalermsook*, Mayank Goswami*, László Kozma†, Kurt Mehlhorn*, Thatchaphol Saranurak‡§

*Max-Planck Institute for Informatics, Saarbrücken, Germany
{parinya, gmayank, mehlhorn}@mpi-inf.mpg.de
†Saarland University, Saarbrücken, Germany
kozma@cs.uni-saarland.de
‡KTH Royal Institute of Technology, Stockholm, Sweden
thasar@kth.se

Abstract

The *dynamic optimality conjecture* is perhaps the most fundamental open question about binary search trees (BST). It postulates the existence of an asymptotically optimal online BST, i.e. one that is *constant factor competitive* with any BST on any input access sequence. The two main candidates for dynamic optimality in the literature are *splay trees* [Sleator and Tarjan, 1985], and GREEDY [Lucas, 1988; Munro, 2000; Demaine et al. 2009]. Despite BSTs being among the simplest data structures in computer science, and despite extensive effort over the past three decades, the conjecture remains elusive. Dynamic optimality is trivial for *almost all* sequences: the optimum access cost of most length-n sequences is $\Theta(n \log n)$, achievable by any balanced BST. Thus, the obvious missing step towards the conjecture is an understanding of the "easy" access sequences, and indeed the most fruitful research direction so far has been the study of specific sequences, whose "easiness" is captured by a parameter of interest. For instance, splay provably achieves the bound of $O(nd)$ when d roughly measures the distances between consecutive accesses (dynamic finger), the average entropy (static optimality), or the delays between multiple accesses of an element (working set). The difficulty of proving dynamic optimality is witnessed by other highly restricted special cases that remain unresolved; one prominent example is the *traversal conjecture* [Sleator and Tarjan, 1985], which states that *preorder sequences* (whose optimum is linear) are linear-time accessed by splay trees; no online BST is known to satisfy this conjecture.

In this paper, we prove two different relaxations of the traversal conjecture for GREEDY: (i) GREEDY is almost linear for preorder traversal, (ii) if a linear-time preprocessing[1] is allowed, GREEDY is in fact linear. These statements are corollaries of our more general results that express the complexity of access sequences in terms of a *pattern avoidance* parameter k. Pattern avoidance is a well-established concept in combinatorics, and the classes of input sequences thus defined are rich, e.g. the $k = 3$ case includes preorder sequences. For any sequence X with parameter k, our most general result shows that GREEDY achieves the cost $n2^{\alpha(n)^{O(k)}}$ where α is the inverse Ackermann function. Furthermore, a broad subclass of parameter-k sequences has a natural combinatorial interpretation as k-*decomposable sequences*. For this class of inputs, we obtain an $n2^{O(k^2)}$ bound for GREEDY when preprocessing is allowed. For $k = 3$, these results imply (i) and (ii). To our knowledge, these are the first upper bounds for GREEDY that are not known to hold for any other online BST. To obtain these results we identify an *input-revealing* property of GREEDY. Informally, this means that the execution log partially reveals the structure of the access sequence. This property facilitates the use of rich technical tools from *forbidden submatrix theory*.

Further studying the intrinsic complexity of k-decomposable sequences, we make several observations. First, in order to obtain an offline optimal BST, it is enough to bound GREEDY on non-decomposable access sequences. Furthermore, we show that the optimal cost for k-decomposable sequences is $\Theta(n \log k)$, which is well below the proven performance of all known BST algorithms. Hence, sequences in this class can be seen as a "candidate counterexample" to dynamic optimality.

I. INTRODUCTION

The binary search tree (BST) model is one of the most fundamental and thoroughly studied data access models in computer science. In this model, given a sequence $X \in [n]^m$ of keys, we are interested in $\mathsf{OPT}(X)$, the optimum cost of accessing X by a binary search tree. When the tree does not change between accesses, OPT is well understood: both efficient exact algorithms (Knuth [22]) and a linear time approximation (Mehlhorn [25])

§Work partly done while at Saarland University.

[1]The purpose of preprocessing is to bring the data structure into a state of our choice. This state is independent of the actual input. This kind of preprocessing is implicit in the Demaine et al. definition of GREEDY.

0272-5428/15 $31.00 © 2015 IEEE
DOI 10.1109/FOCS.2015.32

have long been known. By contrast, in the dynamic BST model (where restructuring of the tree is allowed between accesses) our understanding of OPT is very limited. No polynomial time exact or constant-approximation algorithms are known for computing OPT in the dynamic BST model.

Theoretical interest in the dynamic BST model is partly due to the fact that it holds the possibility of a meaningful "instance optimality", i.e. of a BST algorithm that is constant factor competitive (in the amortized sense) with any other BST algorithm, even with those that are allowed to see into the future. The existence of such a "dynamically optimal" algorithm has been postulated in 1985 by Sleator and Tarjan [32]; they conjectured their splay tree to have this property. Later, Lucas [23] and independently Munro [26] proposed a greedy offline algorithm as another candidate for dynamic optimality. In 2009 Demaine, Harmon, Iacono, Kane, and Pătraşcu (DHIKP) [11] presented a geometric view of the BST model, in which the Lucas-Munro offline algorithm naturally turns into an online one (simply called GREEDY). Both splay and GREEDY are considered plausible candidates for achieving optimality. However, despite decades of research, neither of the two algorithms are known to be $o(\log n)$-competitive (the best known approximation ratio for any BST is $O(\log \log n)$ [12], but the algorithms achieving this bound are less natural). Thus, the dynamic optimality conjecture remains wide open.

Why is dynamic optimality difficult to prove (or disprove)? For a vast majority of access sequences, the optimal access cost by any dynamic BST (online or offline) is $\Theta(m \log n)$ and any static balanced tree can achieve this bound.[2] Hence, dynamic optimality is almost trivial on most sequences. However, when an input sequence has complexity $o(m \log n)$, a candidate BST must "learn" and "exploit" the specific structure of the sequence, in order to match the optimum. This observation motivates the popular research direction that aims at understanding "easy" sequences, i.e. those with $\mathsf{OPT} = O(md)$ when d is a parameter that depends on the structure of the sequence; typically $d = o(\log n)$. Parameter d can be seen as a measure of "easiness", and the objective is to prove that a candidate algorithm achieves the absolute bound of $O(md)$; note that this does not immediately imply that the algorithm matches the optimum.

The splay tree is known to perform very well on many "easy" sequences: It achieves the access time of $O(md)$ when d is a parameter that roughly captures the average distances between consecutive accesses [9], [10], the average entropy, or the recentness of the previous accesses [32]; these properties are called *dynamic finger*, *static optimality* and *working set* bounds respectively (the latter two bounds are also satisfied by GREEDY [15]). The notorious difficulty of dynamic optimality is perhaps witnessed by the fact that many other highly restricted special cases have resisted attempts and remain unresolved. These special cases stand as a roadblock to proving (or disproving) the optimality of any algorithm: proving that $\mathcal{A}$ is optimal requires proving that it performs well on the easy inputs; refuting the optimality of $\mathcal{A}$ requires a special "easy" subclass on which $\mathcal{A}$ is provably inefficient. In any case, a better understanding of easy sequences is needed, but currently lacking, as observed by several authors [11], [28].

One famous example of such roadblocks is the *traversal conjecture*, proposed by Sleator and Tarjan in 1985. This conjecture states that starting with an arbitrary tree T (called *initial tree*) and accessing its keys using the splay algorithm in the order given by the preorder sequence of another tree T' takes linear time. Since the optimum of such preorder sequences is linear, any dynamically optimal algorithm must match this bound. After three decades, only very special cases of this conjecture are known to hold, namely when T' is a monotone path (the so-called *sequential access* [34]) or when $T' = T$ [8].[3] Prior to this work, no online BST algorithm was known to be linear on such access sequences.

Motivated by well-established concepts in combinatorics [20], [21], [35], in this paper we initiate the study of the complexity of access sequences in terms of pattern avoidance parameters. Our main contributions are two-fold (the precise statement of the results is in § I-A, and the results are summarized in Table I).

(i) We study access sequences parametrized by pattern avoidance and analyze the access time of GREEDY in terms of this parameter. As a by-product, we almost settle the traversal conjecture for GREEDY in two orthogonal directions: (a) GREEDY is "almost linear" for the traversal conjecture, and (b) if a linear-cost preprocessing is allowed, GREEDY is in fact linear. This is perhaps the first evidence in support of GREEDY as a candidate for dynamic optimality (all previously known properties of GREEDY were subsumed by splay). These results are derived via an *input-revealing property* of GREEDY, in the sense that the execution log of GREEDY partially reveals the structure of the input sequence.

[2] As customary, we only consider successful accesses, we ignore inserts and deletes, and we assume that $m \geq n$.

[3] This result implies an offline BST with $O(n)$ cost for accessing a preorder sequence, by first rotating T to T'.

	Structure	splay bound	GREEDY bound	Remark
Static optimality	low entropy	$O(\sum_{i=1}^{n} f_i(1+\log \frac{m}{f_i}))$ [32]	same as splay [6], [15]	corollaries of Access Lemma [32]
Working set	temporal locality	$O(m + n\log n + \sum_{t=1}^{m} \log(\tau_t + 1))$ [32]	same as splay [6], [15]	
Dynamic finger	key-space locality	$O(m + \sum_{t=2}^{m} \log \lvert x_t - x_{t-1}\rvert)$ [9], [10]	?	
Deque	n updates at min/max elements	$O(n\alpha^*(n))$ [28]	$O(n2^{\alpha(n)})$ [5]	$O(n)$ for multi-splay [36] from empty tree
Sequential	avoid $(2,1)$	$O(n)$ [29], [34]	$O(n)$ [5], [15]	
Traversal	avoid $(2,3,1)$	?	$n2^{\alpha(n)^{O(1)}}$ [* Cor I.2]	$\mathsf{OPT} = O(n)$
Pattern-avoiding (new)	avoid $(2,3,1)$ with preprocessing	?	$O(n)$ [* Cor I.5]	$\mathsf{OPT} = O(n)$
	avoid $(k,\ldots,1)$ (k-increasing)	?	$n2^{O(k^2)}$ [* Thm I.6]	
	avoid all simple perm. of size $> k$ (k-decomposable) with preprocessing	?	$n2^{O(k^2)}$ [* Thm I.4]	$\mathsf{OPT} = O(n\log k)$ [* Cor I.10]
	avoid an arbitrary perm. of size k	?	$n2^{\alpha(n)^{O(k)}}$ [* Thm I.1]	Best known lower bound is at most $n2^{O(k)}$ [* Thm I.7]

Table I: Easy sequences for BST and best known upper bounds. Results referenced as $[*]$ are new. The symbol "?" means that no nontrivial bound is known. In the first rows, f_i is the number of times that i is accessed, x_t is the element accessed at time t, and τ_t is the number of distinct accesses since the last access of x_t). In the "Sequential", "Traversal", and "Pattern-avoiding" rows it is assumed that the access sequence is a permutation, i.e. $m = n$.

(ii) We study a decomposability property of sequences and show that a wide subclass of the pattern-avoiding class satisfies this property. This allows us to (a) identify the core difficulty of designing an offline algorithm for dynamic optimality, (b) obtain a tight bound for the optimum of this input class, and (c) derive a simple proof that Cole's showcase sequence [10] is linear.

A. Our results

We first define the pattern avoidance parameters. Let $[n] = \{1,\ldots,n\}$ be the set of keys. We represent a length-m sequence by an m-tuple $X = (x_1,\ldots,x_m) \in [n]^m$. Throughout the paper, we assume that X is a permutation, i.e. $m = n$ and $x_i \neq x_j$ for all $i \neq j$. This is justified by Theorem II.1, but we remark that many of our results can be extended to non-permutation access sequences.

Let S_k denote the set of all permutations in $[k]^k$. Given $X \in S_n$, we say that X *contains* a permutation pattern $\pi \in S_k$, if there are indices $1 \le i_1 < i_2 < \cdots i_k \le n$, such that $(x_{i_1},\ldots,x_{i_k})$ is order-isomorphic to $\pi = (\pi_1,\ldots,\pi_k)$. Otherwise we say that X *avoids* π, or that X is π*-free*. Two sequences of the same length are *order-isomorphic* if they have the same pairwise comparisons, e.g. $(5,8,2)$ is order-isomorphic to $(2,3,1)$ (Figure 1). As examples, we mention that $(2,1)$-free and $(2,3,1)$-free sequences are exactly the sequential, respectively, preorder access sequences.

For each permutation X, a *pattern avoidance characteristic* of X, denoted by Σ_X, is the set of all patterns not contained in X, i.e. $\Sigma_X = \{\pi \in S_\ell :\ \ell \ge 1 \text{ and } X \text{ is } \pi\text{-free}\}$. The *pattern avoidance parameter* of X, denoted by $k(X)$, is the minimum $k \in \mathbb{N}$ for which $S_k \cap \Sigma_X \neq \emptyset$.

For each $k \in \mathbb{N}$, the pattern avoidance class $\mathcal{C}_k$ contains all sequences X with pattern avoidance parameter k. This characterization is powerful and universal: $\mathcal{C}_i \subseteq \mathcal{C}_{i+1}$ for all i, and $\mathcal{C}_{n+1}$ contains all sequences.

Bounds for GREEDY *in terms of* k*:* We study the access cost of GREEDY in terms of the pattern avoidance parameter. Our most general upper bound of $n2^{\alpha(n)^{O(k)}}$ holds for any access sequence in $\mathcal{C}_k$ (Theorem I.1). Later, we show a stronger bound of $n2^{O(k^2)}$ for two broad subclasses of $\mathcal{C}_k$ that have intuitive geometric interpretation: k-decomposable sequences (Theorem I.4), which generalize preorder sequences and k-increasing sequences (Theorem I.6), which generalize sequential access.

Theorem I.1. *Let $X \in \mathcal{C}_k$ be an access sequence of length n. The cost of accessing X using* GREEDY *is at most $n2^{\alpha(n)^{O(k)}}$.*

The following corollary simply follows from the fact that all preorder sequences X belong to $\mathcal{C}_3$ (plugging $k = 3$ into the theorem).

Corollary I.2 ("Almost" traversal conjecture for GREEDY)**.** *The cost of accessing a preorder sequence of length n using* GREEDY*, with arbitrary initial tree is at most $n2^{\alpha(n)^{O(1)}}$.*

Next, we show sharper bounds for GREEDY when some structural property of Σ_X is assumed. First, we prove a bound when Σ_X contains all "non-decomposable" permutations of length at least $k+1$. We show that this property leads to a decomposition that has intuitive geometric meaning, which we describe below.

Given a permutation $\sigma = (\sigma_1, \ldots, \sigma_\ell) \in S_\ell$ we call a set $[a, b]$ a *block* of σ, if $\{\sigma_a, \sigma_{a+1}, \ldots, \sigma_b\} = \{c, c+1, \ldots, d\}$ for some integers $c, d \in [\ell]$. In words, a block corresponds to a contiguous interval that is mapped to a contiguous interval. We say that a permutation $\sigma \in S_\ell$ is decomposable if there is a block of σ of size strictly betweeen 1 and ℓ; otherwise, we say that it is *non-decomposable* (a.k.a. *simple*[4]).

We say that an input sequence X is decomposable into k blocks if there exist disjoint $[a_1, b_1], \ldots, [a_k, b_k]$ such that each $[a_i, b_i]$ is a block for X and $(\bigcup_i [a_i, b_i]) \cap \mathbb{N} = [n]$. The *recursive decomposition complexity* of a sequence X is at most d if one can recursively decompose X into subblocks until singletons are obtained, and each step creates at most d blocks (equivalently we say that X is d-recursively decomposable, or simply d-*decomposable*). See Figure 1. It is easy to see that preorder sequences are 2-decomposable.[5]

The following lemma connects pattern avoidance and recursive decomposition complexity of a sequence.

Lemma I.3. *Let $X \in S_n$. Then Σ_X contains all simple permutations of length at least $k+1$ if and only if X is k-decomposable.*

This lemma implies in particular that if X is k-decomposable, then $X \in \mathcal{C}_{k+1}$ and Theorem I.1 can be applied. The following theorem gives a sharper bound, when a preprocessing (independent of X) is performed. This yields another relaxed version of the traversal conjecture.

Theorem I.4. *Let X be a k-decomposable access sequence. The cost of accessing X using* GREEDY*, with a linear-cost preprocessing, is at most $n2^{O(k^2)}$.*

Corollary I.5 (Traversal conjecture for GREEDY with preprocessing)**.** *The cost of accessing a preorder sequence using* GREEDY*, with preprocessing, is linear.*

Our preprocessing step is done in the geometric view (explained in § II) in order to "hide" the initial tree T from GREEDY. In the corresponding tree-view, our algorithm preprocesses the initial tree T, turning it into another tree T', independent of X, and then starts accessing the keys in X using the tree-view variant of GREEDY (as defined in [11]). We mention that DHIKP [11] define GREEDY without initial tree, and thus their algorithm (implicitly) performs this preprocessing.

As another natural special case, we look at $\Sigma_X \supseteq \{(k, \ldots, 1)\}$, for some value k. In words, X has no decreasing subsequence of length k (alternatively, X can be partitioned into at most $k-1$ increasing subsequences Observe that the $k = 2$ case corresponds to sequential access. We call this the k-*increasing* property of X. Similarly, if $\Sigma_X \supseteq \{(1, \ldots, k)\}$, we say that X is k-*decreasing*. We obtain the following generalization of the sequential access theorem (this result holds for arbitrary initial tree).

Theorem I.6. *Let X be a k-increasing or k-decreasing sequence. The cost of accessing X using* GREEDY *is at most $n2^{O(k^2)}$.*

All the aforementioned results are obtained via the *input-revealing* property of GREEDY (the ideas are sketched in § I-B).

Bound for signed GREEDY *in terms of k:* We further show the applicability of the input-revealing technique. Signed GREEDY [11] (denoted SGREEDY) is an algorithm similar to GREEDY. It does not always produce a feasible BST solution, but its cost serves as a lower bound for the optimum. Let us denote by $\mathcal{A}(X)$ and $\mathsf{OPT}(X)$ the cost of an algorithm $\mathcal{A}$ on X, respectively the optimum cost of X. Then we have $\mathcal{A}(X) \geq \mathsf{OPT}(X)$, and $\mathsf{OPT}(X) = \Omega(\text{SGREEDY}(X))$.

Conjecture 1 (Wilber [37], using [11])**.** $\mathsf{OPT}(X) = \Theta(\text{SGREEDY}(X))$ *for all input sequences X.*

[4]See [3] for a survey of this well-studied concept. The decomposition described here appears in the literature as *substitution-decomposition*.

[5]The entire set of 2-decomposable permutations is known as the set of *separable permutations* [2].

We show that our techniques can be used to upper bound the cost of SGREEDY.

Theorem I.7. *The cost of* SGREEDY *on an access sequence $X \in \mathcal{C}_k$ is at most $n2^{O(k)}$.*

Corollary I.8. *Let $\mathcal{A}$ be an algorithm. If Conjecture 1 is true, then one of the following must hold:*
(i) The cost of accessing X using $\mathcal{A}$ is at most $n2^{O(k)}$, for all $X \in \mathcal{C}_k$, for all k. In particular, the traversal conjecture holds for $\mathcal{A}$.
(ii) $\mathcal{A}$ is not dynamically optimal.

Decomposability: Next, we prove a general theorem on the complexity of sequences with respect to the recursive decomposition. Let $X \in S_n$ be an arbitrary access sequence. We represent the process of decomposing X into blocks until singletons are obtained by a tree $\mathcal{T}$, where each node v in $\mathcal{T}$ is associated with a sequence X_v (see Figure 1).

Theorem I.9 (Decomposition theorem, informal)**.** $\mathsf{OPT}(X) \leq \sum_{v \in \mathcal{T}} \textsc{Greedy}(X_v) + O(n)$.

Corollary I.10. *Let X be a k-decomposable sequence of length n (with k possibly depending on n). Then X has optimum cost $\mathsf{OPT}(X) \leq O(n \log k)$.*

This result is tight for all values of k, since for all k, there is a k-decomposable sequence whose complexity is $\Omega(n \log k)$ [7]. This result gives a broad range of input sequences whose optimum is well below the performance of all known BST algorithms. The class can thus be useful as "candidate counterexample" for dynamic optimality. (Since we know OPT, we only need to show that no online algorithm can match it.) We remark that GREEDY asymptotically matches this bound when k is constant (Theorem I.4).

The following observation follows from the fact that the bound in Theorem I.9 is achieved by our proposed offline algorithm, and the fact that $\mathsf{OPT}(X) \geq \sum_{v \in \mathcal{T}} \mathsf{OPT}(X_v)$ (see [7] for details).

Corollary I.11. *If* GREEDY *is constant competitive on non-decomposable sequences, then there is an offline $O(1)$-approximate algorithm on any sequence X.*

This corollary shows that, in some sense, non-decomposable sequences serve as the "core difficulty" of proving dynamic optimality, and understanding the behaviour of GREEDY on them might be sufficient.

Finally, Cole et al. [10] developed sophisticated technical tools for proving the dynamic finger theorem for splay trees. They introduced a "showcase" sequence to illustrate the power of their tools, and showed that splay has linear cost on such a sequence. Theorem I.9 immediately gives an upper bound on the optimum access cost of this sequence.

Corollary I.12 (informal)**.** *Let X be Cole's showcase sequence. Then $\mathsf{OPT}(X) = O(n)$.*

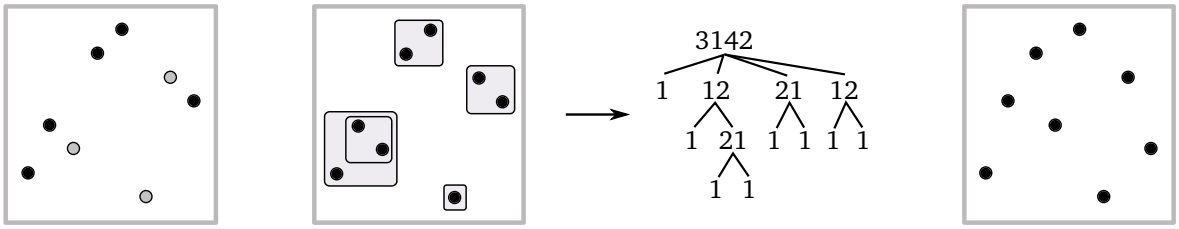


Figure 1: From left to right: (i) plot of permutation $\sigma = (6, 1, 3, 2, 8, 7, 4, 5)$ containing $(2, 1, 3)$ (highlighted) and avoiding $(4, 3, 2, 1)$, (ii) permutation σ, and (iii) its block decomposition tree; permutation $(3, 1, 4, 2)$ at the root is obtained by contracting the four blocks into points, (iv) simple permutation $(6, 1, 8, 4, 2, 7, 3, 5)$.

B. Overview of techniques

We sketch the main technical ideas of our proofs. First, we define roughly the execution log matrix of GREEDY. The log of executing GREEDY on $X \in [n]^m$ is an n-by-m matrix G_X such that $G_X(a, t) = 1$ if and only if element a is touched by GREEDY at time t. The cost of this execution is exactly the number of ones in G_X, denoted by $w(G_X)$ (we make this claim precise in § II).

A submatrix of G_X is obtained by removing some rows and columns of G_X. We say that G_X *contains* a matrix B if there is a submatrix B' of G_X such that $B(i,j) \leq B'(i,j)$ for all i,j (in words, any 'one' entry in B must also be a one in B').

We say that an algorithm $\mathcal{A}$ is *input-revealing* if there is a constant-size matrix B such that any submatrix of an execution matrix of $\mathcal{A}$ containing B must contain an access point. When such a matrix B exists for an algorithm, we say that B is a *capture gadget*. The existence for GREEDY of a capture gadget B allows us to observe that G_X contains the pattern[6] $Q \otimes B$ only if the input sequence contains the pattern defined by Q. So, if we execute GREEDY on an input sequence $X \in \mathcal{C}_k$ that is Q-free for $Q \in S_k$, then G_X cannot contain $Q \otimes B$. Now, we can use results from forbidden submatrix theory for bounding $w(G_X)$. In short, forbidden submatrix theory is a collection of theorems of the following type:

Theorem I.13. *Let M and P be n-by-n and k-by-k matrices respectively, such that $k \leq n$ (mostly k should be thought of as constant). If M does not contain P, then $w(M) \leq n f_P(n,k)$.*

The function f_P depends on the matrix P that is forbidden in M. For instance, if P contains at most one 1-entry per row and column, then $f_P(n,k)$ is only a function of k. If P contains at most one 1-entry per column but possibly multiple 1s per row, then $f_P(n,k) = 2^{\alpha(n)^{O(k)}}$. The tightness of the bounds we obtain depends on the structure of the capture gadget B. This is the reason we have a weaker bound for any $X \in \mathcal{C}_k$ and stronger bounds when X is more restricted.

We also develop a different set of tools to study the intrinsic complexity of decomposable sequences. Here, in order to compute bounds on OPT, we decompose the execution matrix M_X of an algorithm in a way that mirrors the recursive decomposition of the input sequence X. Unfortunately, the behaviour of GREEDY is too capricious to afford such a clean decomposition. We therefore modify GREEDY to reduce the interference between blocks in the input sequence. The modified algorithm might perform more work locally than GREEDY, but globally it is more "robust". It is however, an offline algorithm; whether there exists an online algorithm competitive with our robust GREEDY, is an interesting open question.

Related work: For an overview of results related to dynamic optimality, we refer to the survey of Iacono [19]. Besides Tango trees, other $O(\log\log n)$-competitive BST algorithms have been proposed, similarly using Wilber's first bound (multi-splay [36] and chain-splay [17]). Using [13], all the known bounds can be simultaneously achieved by combining several BST algorithms. Our study of the execution log of BST algorithms makes use of the geometric view of BST introduced by DHIKP [11].

Pattern avoidance in sequences has a large literature in combinatorics [20], [35], as well as in computer science [2], [4], [21], [31], [33]. The theory of forbidden submatrices can be traced back to a 1951 problem of Zarankiewicz, and its systematic study has been initiated by Füredi [16], and Bienstock and Győri [1]. Our use of this theory is inspired by work of Pettie [28], [29], who applied forbidden pattern techniques to prove bounds about data structures (in particular he proves that splay trees achieve $O(n\alpha^*(n))$ cost on deque-sequences, and he reproves the sequential access theorem for splay trees). There are marked differences in our use of the techniques compared with the work of Pettie. In particular, we look for patterns directly in the execution log of a BST algorithm, without any additional encoding step. Furthermore, instead of using fixed forbidden patterns, we make use of patterns that depend on the input.

Organization of the paper: In § II we give a short description of the geometric view of the BST model (following DHIKP [11]), and we introduce concepts used in the subsequent sections. In § III we study the input-revealing property of GREEDY, and prove our bounds for pattern-avoiding sequences. In § IV we introduce a robust GREEDY, state the decomposition theorem and its applications. Finally, § V is reserved for open questions. In this extended abstract some of the proofs are omitted. We refer the reader to the preprint version of the paper [7] for full details and for further auxiliary material.

II. PRELIMINARIES

The main object of study in this paper are $\{0,1\}$-matrices (corresponding to the execution matrix of an algorithm). For geometric reasons, we write matrix entries bottom-to-top, i.e. the first row is the bottom-most. Thus, $M(i,j)$ is the value in the i-th column (left-to-right) and the j-th row (bottom-to-top) of M. We simultaneously view M

[6]The symbol $\otimes$ denotes the tensor (Kronecker) product, to be defined later.

as a set of integral points in the plane. For a point $p = (x, y)$, we say that $p \in M$, if $M(x, y) = 1$. We denote by $p.x$ and $p.y$ the x- and y-coordinates of point p respectively.

Given a point set (matrix) M, let p and q be points not necessarily in M. Let $\square_{pq}$ be the closed rectangle whose two corners are p and q. Rectangle $\square_{pq}$ is *empty* if $M \cap \square_{pq} \setminus \{p, q\} = \emptyset$. Rectangle $\square_{pq}$ is *satisfied* if p, q have the same x or y coordinate, or there is another point $r \in M \cap \square_{pq} \setminus \{p, q\}$. A point set M is *(arboreally) satisfied* if $\square_{pq}$ is satisfied for any two points $p, q \in M$.

Geometric setting for online BST: We describe the geometric model for analyzing online BST algorithms (or simply online BST). This model is essentially the same as [11], except for the fact that our model allows cost analysis of BST algorithms that start accessing an input sequence from arbitrary initial tree.

For any access sequence $X \in [n]^m$, we also refer to X as an n-by-m matrix, called *access matrix*, where there is a point $(x, t) \in X$ iff an element x is accessed at time t. We call (x, t) the access point (or input point) at time t. We also refer to the x-th column of X as *element* x, and to the t-th row of X as *time* t.

For any tree T of depth d, let $d(x)$ be the depth of element x in T. We also refer to T as an n-by-d matrix, called *initial tree matrix*, where in each column x, there is a stack of points $(x, 1), \ldots, (x, d - d(x) + 1) \in T$. See Figure 2 for an illustration. Observe that matrix T is satisfied.

An *online geometric* BST $\mathcal{A}$ accessing sequence X starting with initial tree T works as follows. Given a matrix $\left[\begin{smallmatrix} X \\ T \end{smallmatrix}\right]$ as input, $\mathcal{A}$ outputs a satisfied matrix $\left[\begin{smallmatrix} \mathcal{A}_T(X) \\ T \end{smallmatrix}\right]$, where $\mathcal{A}_T(X) \supseteq X$ is an n-by-m matrix called the *touch matrix* of $\mathcal{A}$ on X with initial tree T. More precisely, $\mathcal{A}$ proceeds in an online fashion: At time $t = 1, \ldots, n$, the algorithm $\mathcal{A}$ has access to T as well as to the first t rows of matrix X, and it must decide on the t-th row of the output $\mathcal{A}_T(X)$ which cannot be changed afterwards. Algorithm $\mathcal{A}$ is *feasible* if $\left[\begin{smallmatrix} \mathcal{A}_T(X) \\ T \end{smallmatrix}\right]$ is satisfied.

The matrix $\mathcal{A}_T(X)$ can be thought of as the execution log of $\mathcal{A}$, and points in $\mathcal{A}_T(X)$ are called *touch points*; the *cost* of $\mathcal{A}$ is simply $w(\mathcal{A}_T(X))$, i.e. the number of points (ones) in $\mathcal{A}_T(X)$.

An online algorithm with *preprocessing* is an online algorithm that is allowed to perform differently at time $t = 0$. The linear-cost preprocessing used in this paper, as well as in DHIKP [11], is to put all elements into a *split tree* (see [11] for the details of this data structure). The consequences of this preprocessing are that (i) in the geometric view the initial tree matrix can be removed, i.e. an algorithm $\mathcal{A}$ only gets X as input and outputs $\mathcal{A}(X)$, and (ii) in the tree-view, an input initial tree T is turned into another tree T' independent of the input X, and then keys in X are accessed using the tree-view variant of GREEDY.

DHIKP [11] showed[7] that online *geometric* BSTs are essentially equivalent to online BSTs (in tree-view). In particular, given any online geometric BST $\mathcal{A}$, there is an online BST $\mathcal{A}'$ where the cost of running $\mathcal{A}'$ on any sequence X with any initial tree T is a constant factor away from $w(\mathcal{A}_T(X))$. Therefore, in this paper, we focus on online geometric BSTs and refer to them from now on simply as online BSTs.

Unless explicitly stated otherwise, we assume that the access sequence X is a permutation, i.e. each element is accessed only once, and thus X and $\mathcal{A}_T(X)$ are n-by-n matrices. This is justified by the following theorem [7].

Theorem II.1. *Assume that there is a c-competitive geometric BST $\mathcal{A}^p$ for all permutations. Then there is a $4c$-competitive geometric BST $\mathcal{A}$ for all access sequences.*[8]

GREEDY *and signed* GREEDY*:* Next, we describe the algorithms GREEDY and signed GREEDY [11] (denoted as SGREEDY).

Consider the execution of an online BST $\mathcal{A}$ on sequence X with initial tree T just before time t. Let O_{t-1} denote the output at time $t - 1$ (i.e. the initial tree T and the first $t - 1$ rows of $\mathcal{A}_T(X)$). Let $\tau(b, t - 1) = \max\{i \mid (b, i) \in O_{t-1}\}$ for any $b \in [n]$. In words, $\tau(b, t - 1)$ is the last time *strictly before* t when b is touched, otherwise it is a non-positive number defined by the initial tree. Let $(a, t) \in X$ be the access point at time t. We define $\mathsf{stair}_t(a)$ as the set of elements $b \in [n]$ for which $\square_{(a,t),(b,\tau(b,t-1))}$ is not satisfied. GREEDY touches element $x \in [n]$ if and only if $x \in \mathsf{stair}_t(a)$. See Figure 2 for an illustration of stairs.

We define $\mathsf{stairLeft}_t(a) = \{b \in \mathsf{stair}_t(a) \mid b < a\}$ and $\mathsf{stairRight}_t(a) = \{b \in \mathsf{stair}_t(a) \mid b > a\}$. The algorithms GREEDYLEFT and GREEDYRIGHT are analogous to GREEDY, but they only touch $\mathsf{stairLeft}_t(a)$, respectively $\mathsf{stairRight}_t(a)$ for every time t; SGREEDY simply refers to the union of the GREEDYLEFT and GREEDYRIGHT

[7] with some minor additional argument about initial trees.

[8] The statement holds unconditionally for offline algorithms, and under a mild assumption if we require $\mathcal{A}^p$ and $\mathcal{A}$ to be online.

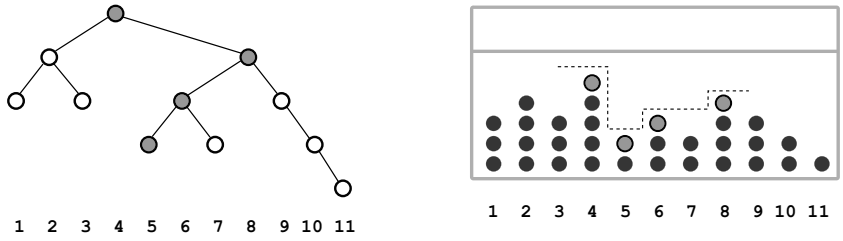


Figure 2: Tree-view and geometric view of initial tree. Search path and stair (at time 1) of element 5.

outputs, which is not necessarily satisfied. It is known [11], [18] that SGREEDY constant-approximates the *independent rectangle bound* which subsumes the two previously known lower bounds on OPT, called Wilber's first and second bounds [37].

Theorem II.2 ([11], [18], [37])**.** *For any BST $\mathcal{A}$ (online or offline) that runs on sequence X with arbitrary initial tree T, $w(\mathcal{A}_T(X)) = \Omega(\text{SGREEDY}(X))$.*

A nice property of GREEDY and SGREEDY is that an element x can become "hidden" in some range in the sense that if we do not access in x's hidden range, then x will not be touched. Next, we formally define this concept, and list some cases when elements become hidden during the execution of GREEDY and SGREEDY. The concept is illustrated in Figure 3.

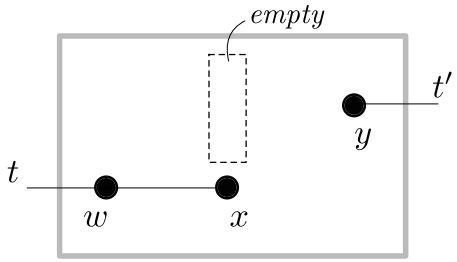


Figure 3: After time t, element x is hidden in $(w, n]$ for GREEDY, and in $(w, x]$ for GREEDYRIGHT. After time t', element x is hidden in (w, y) for GREEDY; it is easy to verify that any access outside of $[w, y]$ will never touch x.

Definition II.3 (Hidden)**.** *For any algorithm $\mathcal{A}$, an element x is* hidden in *range $[w, y]$ after t if, given that there is no access point $p \in [w, y] \times (t, t']$ for any $t' > t$, then x will not be touched by $\mathcal{A}$ in time $(t, t']$.*

Lemma II.4. *Let x be some element.*

(i) If there is an element $w < x$ where $\tau(w, t) \geq \tau(x, t)$, then x is hidden in $(w, n]$ and $(w, x]$ after t for GREEDY *and* GREEDYRIGHT *respectively.*

(ii) If there is an element $y > x$ where $\tau(y, t) \geq \tau(x, t)$, then x is hidden in $[1, y)$ and $[x, y)$ after t for GREEDY *and* GREEDYLEFT *respectively.*

(iii) If there are elements w, y where $w < x < y$, and $\tau(w, t), \tau(y, t) \geq \tau(x, t)$, then x is hidden in (w, y) after t for GREEDY.

Forbidden submatrix theory: Let M be an n-by-m matrix. Given $I \subseteq [n]$ and $J \subseteq [m]$, a *submatrix* $M|_{I,J}$ is obtained by removing columns in $[n] \setminus I$ and rows in $[m] \setminus J$ from M.

Definition II.5 (Forbidden submatrix)**.** *Given a matrix M and another matrix P of size $c \times d$, we say that M* contains *P if there is a $c \times d$ submatrix M' of M such that $M'(i, j) = 1$ whenever $P(i, j) = 1$. We say that M* avoids *(forbids) P if M does not contain P.*

We often call an avoided (forbidden) matrix a "pattern". For any permutation $\pi = (\pi_1, \ldots, \pi_k) \in S_k$, we also refer to π as a k-by-k matrix where $(\pi_i, i) \in \pi$ for all $1 \leq i \leq k$. Permutation matrices have a single one-entry in

every row and in every column. Relaxing this condition slightly, we call a matrix that has a single one in every column a *light* matrix. We make use of the following results.

Lemma II.6. *Let M and P be n-by-n and k-by-k matrices such that M avoids P.*

(i) (Marcus, Tardos [24], Fox [14]) If P is a permutation, the number of ones in M is at most $n2^{O(k)}$.

(ii) (Nivasch [27], Pettie [30]) If P is light, the number of ones in M is at most $n2^{\alpha(n)^{O(k)}}$.

We define the *tensor product* between matrices as follows: $M \otimes G$ is the matrix obtained by replacing each one-entry of M by a copy of G, and replacing each zero-entry of M by an all-zero matrix equal in size with G. Suppose that M contains P. A *bounding box* B of P is a minimal submatrix $M|_{I,J}$ such that P is contained in B, and I and J are sets of consecutive columns, resp. rows. We denote $\min(I)$, $\max(I)$, $\min(J)$, $\max(J)$, as $B.xmin$, $B.xmax$, $B.ymin$, $B.ymax$, respectively.

III. Greedy is input-revealing

In this section, we prove Theorems I.1, I.4, I.6 and I.7. We refer to § I-A for definitions of pattern-avoiding, k-decomposable and k-increasing permutations.

To bound the cost of algorithm $\mathcal{A}$, we show that $\mathcal{A}_T(X)$ (or $\mathcal{A}(X)$ if preprocessing is allowed) avoids some pattern. In the following, $\texttt{Inc}_k$, $\texttt{Dec}_k$, and $\texttt{Alt}_k$ are permutation matrices of size k, and $\texttt{Cap}$ is a light matrix. These matrices will be defined later.

Lemma III.1. *Let X be an access sequence.*

(i) If X avoids a permutation P of size k, then for any initial tree T, $\textsc{GreedyRight}_T(X)$, $\textsc{GreedyLeft}_T(X)$ and $\textsc{Greedy}_T(X)$ avoid $P \otimes (1,2)$, $P \otimes (2,1)$ and $P \otimes \texttt{Cap}$ respectively.

(ii) If X is k-increasing (k-decreasing), then for any initial tree T, $\textsc{Greedy}_T(X)$ avoids $(k,\dots,1) \otimes \texttt{Dec}_{k+1}$ ($(1,\dots,k) \otimes \texttt{Inc}_{k+1}$).

(iii) If X is k-decomposable, then $\textsc{Greedy}(X)$ avoids $P \otimes \texttt{Alt}_{k+4}$ for all simple permutations P of size at least $k+1$.

The application of Lemma II.6 is sufficient to complete all the proofs, together with the observation that, for any permutation matrix P of size $k \times k$, if G is a permutation (resp. light) matrix of size $\ell \times \ell$, then $P \otimes G$ is a permutation (resp. light) matrix of size $k\ell \times k\ell$.

The rest of this section is devoted to the proof of Lemma III.1. We need the definition of the *input-revealing gadget* which is a pattern whose existence in the touch matrix (where access points are indistinguishable from touch points) gives some information about the locations of access points.

Definition III.2 (Input-revealing gadgets)**.** *Let X be an access matrix, $\mathcal{A}$ be an algorithm and $\mathcal{A}(X)$ be the touch matrix of $\mathcal{A}$ running on X.*

- **Capture Gadget.** *A pattern P is a* capture gadget *for algorithm $\mathcal{A}$ if, for any bounding box B of P in $\mathcal{A}(X)$, there is an access point $p \in B$.*
- **Monotone Gadget.** *A pattern P is a k-*increasing (k-decreasing) gadget *for algorithm $\mathcal{A}$ if, for any bounding box B of P in $\mathcal{A}(X)$, either (i) there is an access point $p \in B$ or, (ii) there are k access points $p_1,\dots,p_k$ such that $B.ymin \le p_1.y < \cdots < p_k.y \le B.ymax$ and $p_1.x < \cdots < p_k.x < B.xmin$ ($p_1.x > \cdots > p_k.x > B.xmax$). In words, $p_1,\dots,p_k$ form the pattern $(1,\dots,k)$ (resp. $(k,\dots,1)$).*
- **Alternating Gadget.** *A pattern P is a k-*alternating gadget *for algorithm $\mathcal{A}$ if, for any bounding box B of P in $\mathcal{A}(X)$, either (i) there is an access point $p \in B$ or, (ii) there are k access points $p_1,\dots,p_k$ such that $B.ymin \le p_1.y < \cdots < p_k.y \le B.ymax$ and $B.xmax < p_i.x$ for all odd i and $p_i.x < B.xmin$ for all even i.*

Notice that if we had a permutation capture gadget G for algorithm $\mathcal{A}$, we would be done: If X avoids pattern Q, then $\mathcal{A}_T(X)$ avoids $Q \otimes G$ and forbidden submatrix theory applies. With arbitrary initial tree T, it is impossible to construct a permutation capture gadget for Greedy even for 2-decomposable sequences [7]. Instead, we define a *light* capture gadget for Greedy for any sequence X and any initial tree T. Additionally, we construct permutation monotone and alternating gadgets. After a preprocessing step, we can use the alternating gadget to construct a permutation capture gadget for k-decomposable sequences.

Lemma III.3. *(i) (12) is a capture gadget for* GreedyRight *and (21) is a capture gadget for* GreedyLeft.

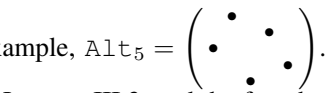
(ii) $\texttt{Cap}$ *is a capture gadget for* GREEDY *where* $\texttt{Cap} = (\bullet\ \bullet\ \bullet)$.

(iii) $\texttt{Inc}_{k+1}$ ($\texttt{Dec}_{k+1}$) *is a* k*-increasing (*k*-decreasing) gadget for* GREEDY *where* $\texttt{Inc}_k = (k, \dots, 1)$ *and* $\texttt{Dec}_k = (1, 2, \dots, k)$.

(iv) $\texttt{Dec}_{k+1}$ ($\texttt{Inc}_{k+1}$) *is a capture gadget for* GREEDY *that runs on* k*-increasing* (k*-decreasing*) *sequence.*

(v) $\texttt{Alt}_{k+1}$ *is a* k*-alternating gadget for* GREEDY *where* $\texttt{Alt}_k = (\lfloor (k+1)/2 \rfloor, k, 1, k-1, 2, \dots)$.

(vi) $\texttt{Alt}_{k+4}$ *is a capture gadget for* GREEDY *that runs on* k*-decomposable sequence without initial tree.*

For example, $\texttt{Alt}_5 =$  . We emphasize that only result (vi) requires a preprocessing.

Given Lemma III.3, and the fact that k-increasing, k-decreasing, and k-decomposable sequences avoid $(k, \dots, 1)$, $(1, \dots, k)$, resp. all simple permutations of size at least $k+1$, Lemma III.1 follows immediately.

Proof: We only show the proofs for (i) and (ii) in order to convey the flavor of the arguments. The other proofs are more involved and we omit them in this version of the paper.

(i) We only prove for GREEDYRIGHT. Let a, b be touch points in $\text{GREEDYRIGHT}_T(X)$ such that $a.y < b.y$ and form $(1, 2)$ with a bounding box B. As $a.y = \tau(a.x, a.y) \geq \tau(b.x, a.y)$, by Lemma II.4 (i), $b.x$ is hidden in $(a.x, b.x]$ after $a.y$. Suppose that there is no access point in $B = \square_{ab}$, then b cannot be touched, which is a contradiction.

(ii) Let a, b, c be touch points in $\text{GREEDY}_T(X)$ such that $a.x < b.x < c.x$ and a, b, c form $\texttt{Cap}$ with bounding box B. As $\tau(a.x, a.y), \tau(c.x, a.y) \geq \tau(b.x, a.y)$, by Lemma II.4 (iii), $b.x$ is hidden in $(a.x, c.x)$ after $a.y$. Suppose that there is no access point in B, then b cannot be touched, which is a contradiction. ■

IV. DECOMPOSITION THEOREM AND APPLICATIONS

In this section, we take a deeper look at decomposable permutations and investigate whether GREEDY is "decomposable", in the sense that one can bound the cost of GREEDY on input sequences by summing over local costs on blocks. Proving such a property for GREEDY seems to be prohibitively difficult [7]. Therefore, we propose a more "robust" but offline variant of GREEDY which turns out to be decomposable. We briefly describe the new algorithm, and state the main theorems and three applications.

Recursive decomposition: We refer the reader to § I-A for the definitions of blocks and recursive decomposition. Let S be a k-decomposable permutation access sequence, and let $\mathcal{T}$ be its (not necessarily unique) decomposition tree (of arity at most k). Let P be a block in the decomposition tree $\mathcal{T}$ of S. Consider the subblocks $P_1, P_2, \dots, P_k$ of P, and let $\tilde{P}$ denote the unique permutation of size $[k]$ that is order-isomorphic to a sequence of arbitrary representative elements from the blocks $P_1, P_2, \dots, P_k$. We denote $P = \tilde{P}[P_1, P_2, \dots, P_k]$ a *deflation* of P, and we call $\tilde{P}$ the *skeleton* of P. Visually, $\tilde{P}$ is obtained from P by contracting each block P_i to a single point. We associate with each block P (i.e. each node in $\mathcal{T}$) both the corresponding rectangular region (in fact, a square), and the corresponding skeleton $\tilde{P}$ (Figure 1). We denote the set of leaves and non-leaves of the tree $\mathcal{T}$ by $L(\mathcal{T})$ and $N(\mathcal{T})$ respectively.

Consider the k subblocks $P_1, \dots, P_k$ of a block P in this decomposition, numbered by increasing y-coordinate. We partition the rectangular region of P into k^2 rectangular regions, whose boundaries are aligned with those of $P_1, \dots, P_k$. We index these regions as $R(i, j)$ with i going left to right and j going bottom to top. In particular, the region $R(i, j)$ is aligned (same y-coordinates) with P_j.

A robust version of GREEDY*:* The main difficulty in analyzing the cost of GREEDY on decomposable access sequences is the "interference" between different blocks of the decomposition. Our RGREEDY algorithm is an extension of GREEDY. Locally it touches at least those points that GREEDY would touch; additionally it may touch other points, depending on the recursive decomposition of the access sequence (this a priori knowledge of the decomposition tree is what makes RGREEDY an offline algorithm). The purpose of touching extra points is to "seal off" the blocks, limiting their effect to the outside.

We now define the concept of *topwing*. The *topwing* of a rectangular region R at a certain time in the execution of RGREEDY contains a subset of the points in R (at most one in each column). A point is in the topwing if it forms an empty rectangle with the top left or the top right corner of R (this definition includes the topmost touch point of the leftmost and rightmost columns of R, if such points exist in R). When accessing an element at time t, RGREEDY touches the projections to the timeline t of the topwings of all regions $R(i, j)$ aligned with a block P_j

ending at time t (this includes the region of P_j itself). Observe that multiple nested blocks might end at the same time, in this case we process them in increasing order of their size. See Figure 4 for an illustration of RGREEDY.

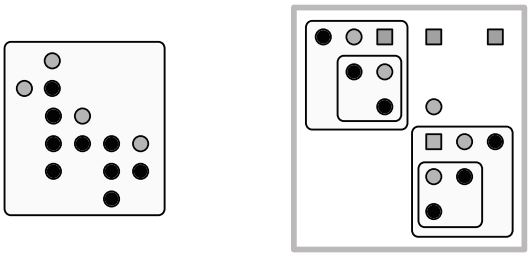


Figure 4: Illustration of RGREEDY. Left: region with points and *topwing* highlighted. Right: a sample run of RGREEDY on a 2-decomposable input. Access points are dark circles, points touched by GREEDY are gray circles, points touched by the RGREEDY augmentation are gray squares. The access point in the topmost line completes the black square and also the enclosing gray square. The topwing of the black square consists of the black circle at position (1,6) and the gray circle at (3,5). Therefore the augmentation of the black square adds the gray square at (3,6). The topwing of $R(2,2)$ consists of the gray circle at (4,4) and hence the gray square at (4,6) is added. The topwing of the gray square consists of points (1,6) and (6,3); the point (4,4) does not belong to the topwing because (4,6) has already been added. Therefore, the gray square at (6,6) is added.

Theorem IV.1 (Decomposition theorem)**.** *For any decomposition tree $\mathcal{T}$ of a permutation P, the total cost of* RGREEDY *is bounded as*

$$\text{RGREEDY}(P) \le 4 \cdot \sum_{\tilde{P} \in N(\mathcal{T})} \text{GREEDY}(\tilde{P}) + \sum_{P \in L(\mathcal{T})} \text{GREEDY}(P) + 3n.$$

Application 1: Cole et al.'s "showcase" sequences: This sequence can be defined as a permutation $P = \tilde{P}[P_1, \ldots, P_{n/\log n}]$ where each P_j is the permutation $(1, \ldots, \log n)$, i.e. a sequential access. It is known that GREEDY is linear on sequential access, so $\text{GREEDY}(P_j) = O(\log n)$. Now, by applying Theorem IV.1, we can say that the cost of RGREEDY on P is $\text{RGREEDY}(P) \le 4 \cdot \text{GREEDY}(\tilde{P}) + \sum_j \text{GREEDY}(P_j) + O(n) \le O(\frac{n}{\log n} \cdot \log(\frac{n}{\log n})) + \frac{n}{\log n} O(\log n) + O(n) = O(n)$. This implies the linear optima of this class of sequences.

Application 2: Simple permutations as the "core" difficulty of BST inputs: We prove that, if GREEDY is c-competitive on simple permutations, then RGREEDY is $O(c)$-approximate for all permutations.

To prove this, we consider a block decomposition tree $\mathcal{T}$ in which the permutation corresponding to every node is simple. Using Theorem IV.1 and the hypothesis that GREEDY is c-competitive for simple permutations:

$$\text{RGREEDY}(P) \le 4c \cdot \sum_{\tilde{P} \in N(\mathcal{T})} \mathsf{OPT}(\tilde{P}) + c \cdot \sum_{P \in L(\mathcal{T})} \mathsf{OPT}(P) + 3n.$$

Now we decompose OPT into subproblems the same way as done for RGREEDY. It can be shown that $\mathsf{OPT}(P) \ge \sum_{P \in \mathcal{T}} \mathsf{OPT}(P) - 2n$. Combined with Theorem IV.1 this yields $\text{RGREEDY}(P) \le 4c \cdot \mathsf{OPT}(P) + (8c+3)n$. Using Theorem II.1 we can extend this statement to arbitrary access sequences.

Application 3: Recursively decomposable permutations: Let X be a k-decomposable permutation. Then there is a decomposition tree $\mathcal{T}$ of X in which each node is a permutation of size at most k. Each block $P \in L(\mathcal{T}) \cup N(\mathcal{T})$ has $\text{GREEDY}(P) = O(|P| \log |P|) = O(|P| \log k)$. By standard charging arguments, the sum $\sum_{P \in N(\mathcal{T}) \cup L(\mathcal{T})} |P| = O(n)$, so the total cost of RGREEDY is $O(n \log k)$. For details of the proof we refer the reader to the preprint version of the paper [7].

We remark that the logarithmic dependence on k is optimal, and that GREEDY asymptotically matches this bound when k is constant (Theorem I.4).

V. DISCUSSION AND OPEN QUESTIONS

Besides the long-standing open question of resolving dynamic optimality, our work raises several new ones in the context of pattern avoidance. We list some that are most interesting to us.

Can the bound of Theorem I.1 be improved? While our input-revealing techniques are unlikely to yield a linear bound, a slight improvement could come from strengthening Lemma II.6(ii) for the special kind of light matrices that arise in our proof.

An intriguing question is whether there is a natural characterization of all sequences with linear optimum. How big is the overlap between "easy inputs" and inputs with pattern avoidance properties? We have shown that avoidance of a small pattern makes sequences easy. The converse does not hold: There is a permutation $X \in S_n$ with $k(X) = \sqrt{n}$ but for which $\textsc{Greedy}(X) = O(n)$ [7]. Note that our pattern-avoiding properties are incomparable with the earlier parametrizations (e.g. dynamic finger). Is there a parameter that subsumes both pattern-avoidance and dynamic finger?

A question directly related to our work is to close the gap between $\mathsf{OPT} = O(n \log k)$ and $n2^{O(k^2)}$ by GREEDY on k-decomposable sequences (when $k = \omega(1)$). Matching the optimum (if at all possible) likely requires novel techniques: splay is not even known to be linear on preorder sequences with preprocessing, and with forbidden-submatrix-arguments it seems impossible to obtain bounds beyond[9] $O(nk)$.

We proved that if GREEDY is optimal on simple permutations, then RGREEDY is optimal on all access sequences. Can some property of simple permutations be algorithmically exploited? Can RGREEDY be made online? Can our application for Cole's showcase sequence be extended in order to prove the dynamic finger property for GREEDY?

Finally, making further progress on the traversal conjecture will likely require novel ideas. We propose a simpler question that captures the barrier for three famous conjectures: *traversal*, *deque*, and *split*. Given any initial tree T, access a preorder sequence defined by a path P (a BST where each non-leaf node has a single child). Prove a linear bound for your favorite online BST!

[9] An n-by-n matrix avoiding *all* permutations of size at least k can contain $\Omega(nk)$ ones.

REFERENCES

[1] Daniel Bienstock and Ervin Györi. An extremal problem on sparse 0-1 matrices. *SIAM J. Discrete Math.*, 4(1):17–27, 1991.

[2] Prosenjit Bose, Jonathan F Buss, and Anna Lubiw. Pattern matching for permutations. *Information Processing Letters*, 65(5):277–283, 1998.

[3] Robert Brignall. A survey of simple permutations. *CoRR*, abs/0801.0963, 2008.

[4] Marie-Louise Bruner and Martin Lackner. The computational landscape of permutation patterns. *CoRR*, abs/1301.0340, 2013.

[5] P. Chalermsook, M. Goswami, L. Kozma, K. Mehlhorn, and T. Saranurak. Greedy is an almost optimal deque. *WADS*, 2015.

[6] P. Chalermsook, M. Goswami, L. Kozma, K. Mehlhorn, and T. Saranurak. Self-adjusting binary search trees: What makes them tick? *ESA*, 2015.

[7] Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Pattern-avoiding access in binary search trees. *CoRR*, abs/1507.06953, 2015.

[8] R. Chaudhuri and H. Höft. Splaying a search tree in preorder takes linear time. *SIGACT News*, 24(2):88–93, April 1993.

[9] R. Cole. On the dynamic finger conjecture for splay trees. part ii: The proof. *SIAM Journal on Computing*, 30(1):44–85, 2000.

[10] Richard Cole, Bud Mishra, Jeanette Schmidt, and Alan Siegel. On the dynamic finger conjecture for splay trees. part i: Splay sorting log n-block sequences. *SIAM J. Comput.*, 30(1):1–43, April 2000.

[11] Erik D. Demaine, Dion Harmon, John Iacono, Daniel M. Kane, and Mihai Patrascu. The geometry of binary search trees. In *SODA 2009*, pages 496–505, 2009.

[12] Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Patrascu. Dynamic optimality - almost. *SIAM J. Comput.*, 37(1):240–251, 2007.

[13] Erik D. Demaine, John Iacono, Stefan Langerman, and Özgür Özkan. Combining binary search trees. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 388–399, 2013.

[14] Jacob Fox. Stanley-Wilf limits are typically exponential. *CoRR*, pages –1–1, 2013.

[15] Kyle Fox. Upper bounds for maximally greedy binary search trees. In *WADS 2011*, pages 411–422, 2011.

[16] Zoltán Füredi. The maximum number of unit distances in a convex n-gon. *J. Comb. Theory, Ser. A*, 55(2):316–320, 1990.

[17] George F. Georgakopoulos. Chain-splay trees, or, how to achieve and prove loglogn-competitiveness by splaying. *Inf. Process. Lett.*, 106(1):37–43, 2008.

[18] Dion Harmon. *New Bounds on Optimal Binary Search Trees*. PhD thesis, Massachusetts Institute of Technology, 2006.

[19] John Iacono. In pursuit of the dynamic optimality conjecture. In *Space-Efficient Data Structures, Streams, and Algorithms*, volume 8066 of *Lecture Notes in Computer Science*, pages 236–250. Springer Berlin Heidelberg, 2013.

[20] S. Kitaev. *Patterns in Permutations and Words*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2011.

[21] Donald E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms*. Addison-Wesley, 1968.

[22] Donald E. Knuth. Optimum binary search trees. *Acta Informatica*, 1(1):14–25, 1971.

[23] Joan M. Lucas. Canonical forms for competitive binary search tree algorithms. *Tech. Rep. DCS-TR-250, Rutgers University*, 1988.

[24] Adam Marcus and Gábor Tardos. Excluded permutation matrices and the Stanley-Wilf conjecture. *Journal of Combinatorial Theory, Series A*, 107(1):153 – 160, 2004.

[25] Kurt Mehlhorn. Nearly optimal binary search trees. *Acta Informatica*, 5(4):287–295, 1975.

[26] J.Ian Munro. On the competitiveness of linear search. In Mike S. Paterson, editor, *Algorithms - ESA 2000*, volume 1879 of *Lecture Notes in Computer Science*, pages 338–345. Springer Berlin Heidelberg, 2000.

[27] Gabriel Nivasch. Improved bounds and new techniques for davenport–schinzel sequences and their generalizations. *J. ACM*, 57(3), 2010.

[28] Seth Pettie. Splay trees, Davenport-Schinzel sequences, and the deque conjecture. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, pages 1115–1124, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[29] Seth Pettie. Applications of forbidden 0-1 matrices to search tree and path compression-based data structures. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1457–1467. SIAM, 2010.

[30] Seth Pettie. Sharp bounds on formation-free sequences. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 592–604, 2015.

[31] Vaughan R. Pratt. Computing permutations with double-ended queues, parallel stacks and parallel queues. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, pages 268–277, New York, NY, USA, 1973. ACM.

[32] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, July 1985.

[33] Robert Endre Tarjan. Sorting using networks of queues and stacks. *J. ACM*, 19(2):341–346, April 1972.

[34] Robert Endre Tarjan. Sequential access in splay trees takes linear time. *Combinatorica*, 5(4):367–378, 1985.

[35] Vincent Vatter. Permutation classes. *CoRR*, abs/1409.5159, 2014.

[36] Chengwen C. Wang, Jonathan C. Derryberry, and Daniel D. Sleator. O(log log n)-competitive dynamic binary search trees. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 374–383, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics.

[37] R. Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM Journal on Computing*, 18(1):56–67, 1989.