The Cost of Offline Binary Search Tree Algorithms and the Complexity of the Request Sequence

Jussi Kujala[*], **Tapio Elomaa**

Institute of Software Systems
Tampere University of Technology
P. O. Box 553, FI-33101 Tampere, Finland
{jussi.kujala,tapio.elomaa}@tut.fi

Abstract

In evaluating the performance of online algorithms for search trees, one wants to compare them to the best offline algorithm available. In this paper we lower bound the cost of an optimal offline binary search tree using the Kolmogorov complexity of the request sequence. We obtain several applications for this result. First, any offline binary search tree algorithm can be at most a constant factor away from the entropy of the process producing the request sequence. Second, for a fraction $1 - 1/2^m$ of request sequences of length m on n items the cost of any offline algorithm is $\Omega(m(\log n - 1))$. Third, the expected cost of splay trees is within a constant factor of the expected cost of an optimal offline binary search tree algorithm in a subset of Markov chains.

Keywords: Offline binary search tree algorithms, Kolmogorov complexity, Splay trees

[*]Corresponding author, Phone: +358 3 3115 3731, Fax: +358 3 3115 2913

1 Introduction

Binary search trees (BSTs) are one of the most fundamental data structures that allow efficient access and update operations to the stored items. They store keys from some totally ordered set and maintain them in symmetric order by reorganizing the tree, when required, using *rotations*. Through rotations one can also maintain the tree in balance in order to reduce the search cost of an item in the BST. In their landmark paper Sleator and Tarjan [14] showed that *splaying* — a particular way of updating a BST online by rotations — also leads to many good properties without the need to maintain explicit balancing information. Splay trees, though, cannot guarantee the efficiency of an operation in the worst case, but achieve excellent amortized efficiency. An alternative data structure with many of the good dynamical properties of splay trees and with guaranteed $O(\log n)$ worst-case access time has been proposed by Iacono [9].

The aim of an online BST algorithm is to serve a sequence of requests efficiently, without knowing it in advance, by using a BST to store the data and rotations to restructure the tree at will. The cost of the algorithm is a combination of its search and rotation cost: Search cost is the number of edges traversed to reach the requested item and each rotation costs one unit of time. The efficiency of an online algorithm is measured competitively: A BST algorithm is said to achieve *static optimality* if its cost for any request sequence never exceeds that of the best static tree by more than a constant factor. Moreover, the algorithm has *dynamic optimality* if its cost is never more than a constant times that of any offline BST algorithm, which knows the request sequence in advance and is allowed to alter the tree.

Contrasting an online BST algorithm with an optimal offline one requires knowing the performance of the latter in detail. However, the cost of optimal offline algorithms is not currently known. Our aim in this paper is to improve this situation by providing more information on their behavior. Obviously, the best update strategy depends on the request sequence in question. Therefore, our analysis also takes its complexity into account.

More exactly, we consider the following problem. Given a sequence s of access requests, possibly generated by a random process, what lower bounds can we give for the minimum cost $\mathbf{c}(s)$ required by an optimal offline BST algorithm? Our focus is on finding a lower bound for $\mathbf{c}(s)$ in terms of the

complexity of the request sequence, which we measure by its *Kolmogorov complexity* $C(s)$. We also study bounding $\mathbf{c}(s)$ below by the entropy of an assumed request generating process. Some random processes may generate any request sequence possible, albeit with very small probability. Therefore, we are not able to bound the cost for all sequences generated by a random process. Instead, we bound the *expected* cost of the offline algorithm with regards to entropy of the request generating process. Throughout this paper we assume that the number of items stored in a BST is n.

Wilber [16] gave two methods for computing a lower bound for $\mathbf{c}(s)$. He showed that the cost of an optimal offline algorithm for the bit reversal permutation is $\Theta(n \log n)$, where n is both the number of items in the tree and the length of the request sequence. Wilber also gave a bound for the expected cost of sequences generated by independent and identically distributed (i.i.d.) stochastic processes. In particular, he showed that for a sequence

$$s = s_1, s_2, \ldots, s_m$$

with m accesses, each generated i.i.d. with probability p_i for the ith item, it holds that

$$\mathbb{E}\left(\mathbf{c}(s)\right) \geq \frac{m}{3}\left(1 + \sum_{i=1}^{n} -p_i \log p_i\right) = \Omega\left(mH\left(X_{i.i.d.}\right)\right), \tag{1}$$

where $H(X_{i.i.d.}) = \sum_{i=1}^{n} -p_i \log p_i$ is the entropy of a process producing one random access.[1] However, Wilber's methods are not easy to use in relating $\mathbf{c}(s)$ to the complexity of a sequence or the entropy of a process in which accesses may depend on each other.

Sleator and Tarjan [14] showed that splaying leads, among many other good properties, to static optimality. They also conjectured that splay trees would have dynamic optimality. This *Dynamic Optimality Conjecture* is still an open problem, which has inspired a lot of work on splay trees. Note that the result of Wilber [16] together with Static Optimality Theorem [14] implies that under expected cost splay trees are constant competitive with an optimal offline BST algorithm in the class of i.i.d. processes. This, of course, is a necessary condition for dynamic optimality of splay trees to hold.

Blum, Chawla, and Kalai [2] were able to show that there exists an inefficient online BST algorithm, which has so-called *dynamic search-optimality*: its

[1] All logarithms in this paper are taken to have base 2.

search cost without the cost of rotations for any sequence is a constant factor away from the total cost of any offline algorithm. This is another necessary prerequisite for Dynamic Optimality Conjecture to hold.

Demaine et al. [7] present an online BST algorithm that is $O(\log \log n)$-competitive with respect to the best offline algorithm. Their result is the first one that is better than the trivial $O(\log n)$-competitiveness bound achieved by balanced search trees. The algorithm relies on Wilber's first bound.

Munro [13] has studied self-organizing lists and observed that the cost of the best offline list algorithm is within a constant factor of the empirical entropy of the request distribution. He also considered whether an offline BST algorithm could beat the entropy bound by more than a constant factor, which would lead to a counterexample to Dynamic Optimality Conjecture. In this paper we refute this possibility under the expected cost of an offline BST algorithm.

1.1 Our Results

In this paper we prove that $\mathbf{c}(s) = \Omega(C(s))$, where $C(s)$ is the Kolmogorov complexity of the sequence s. Using this result we obtain a lower bound for the expectation of the cost in terms of the entropy of the assumed request generating stochastic process:

$$\mathbb{E}\left(\mathbf{c}(s)\right) = \Omega(H(X))\,,$$

where $H(X) = \sum_s -p(s) \lg p(s)$ is the entropy of a process generating a request sequence s with probability $p(s)$. This generalizes Wilber's result to sequences in which the probability distribution on a request may depend of the other requests on the sequence. In addition, we show that if a stationary ergodic process repeatedly generates new requests then asymptotically the average cost is lower bounded by the average increase in the entropy. That is, if m is the number of requests that the process has so far generated and $H_m(X)$ is the entropy over these sequences with m requests, then $\mathbf{c}(s)/m$ is asymptotically $\Omega(H_m(X)/m)$. Stationarity and ergodicity essentially limit the class of probability distributions to the ones that behave well. We proceed to prove that for a fraction $1 - 1/2^m$ of possible request sequences of length m the following bound holds: $\mathbf{c}(s) = \Omega(m \log n - m)$. The significance of this result stems from the fact that $m \log n$ is the worst case cost of a balanced

search tree serving a sequence s. We then modify the fraction to $1-1/2^{c\,m\log n}$ to have $\mathbf{c}(s) = \Omega(c\,m\log n)$ for any c, $0 \le c \le 1$.

As examples of implications of these results we show the following:

- The expected competitive ratio of any balanced search tree with respect to an optimal offline algorithm is $O(H_{\max}(m,n)/H_m(X))$. $H_{\max}(m,n) = m\log n$ is the maximum entropy achievable by a stochastic process that generates a sequence of length m on n items. $H_m(X)$ is the actual entropy of the stochastic process that generates the request sequence with m requests.
- The expected cost of splay trees is a constant times that of an optimal offline algorithm in a subset of stationary Markov chains with conditional probabilities satisfying $p(i \mid j) \le 1/\sqrt{1+|i-j|}$.

A further technical contribution is that r rotations, which rotate edges that form a subtree including the root, can be described with $5r$ bits, thus, tightening slightly the previously known best result $6r$ [2].

Together these results deepen the understanding of the limits of an optimal offline BST algorithm. They are particularly helpful if the request sequence is modeled stochastically — which is often the case — as they show that no offline algorithm can beat the entropy of the stochastic process. In order to prove the competitiveness of an online BST algorithm with an optimal offline BST algorithm, it suffices to prove competitiveness of the online algorithm w.r.t. entropy. This however may be difficult and sometimes even impossible, if the cost of the offline algorithm is greater than a constant times the entropy. Finally, our results provide some insight to proving or disproving the dynamic optimality of splay trees, as it is shown that splay trees perform well in a certain class of request generating processes. Unfortunately, nothing can be said about the worst case for splay trees.

2 Kolmogorov Complexity Preliminaries

We introduce Kolmogorov complexity and its relationship with stochastic processes to the extent needed in this article. For a comprehensive introduction to Kolmogorov complexity and the results presented here see [11].

A short introduction can be found, e.g., in the book of Cover and Thomas [5]. For a comparison of Kolmogorov complexity and information theoretic entropy see [8].

The Kolmogorov complexity of a binary string x is defined as the length of the shortest program for a universal Turing machine that produces x. Thus, Kolmogorov complexity depends on the universal Turing machine used. Let $C(x)$ denote the Kolmogorov complexity of x and $l(x)$ be the length of x. Counting all possible programs of length $l(x) - c$ shows that for a fraction $1 - 1/2^c$ of strings of length $l(x)$ the following must hold:

$$C(x) \geq l(x) - c. \tag{2}$$

If we assume that x is generated by a stochastic process then an interesting connection between $C(x)$ and properties of the stochastic process has been proved [11]. Prefix Kolmogorov complexity $K(x)$ is actually required to formulate the connection. $K(x)$ is defined as the length of the shortest prefix-free program for producing x. Informally this means that the programs form a prefix code. This is not a crucial difference, since one can always catenate the length of the program in self-delimiting form in front of the program, hence, achieving $C(x) \leq K(x) \leq C(x) + 2\lceil \log l(x) \rceil + 2$. Define a stochastic process by probability distribution on binary strings, i.e., each x has a probability $p(x)$ of being generated. The following formula relates the Kolmogorov complexity and the entropy $H(X)$ of the stochastic process producing a binary string x:

$$\mathbb{E}\left(K(x)\right) = \sum_x p(x)K(x) \geq \sum_x -p(x)\log p(x) = H(X). \tag{3}$$

The inequality follows from the fact that the set of prefix-free programs form a prefix code and, thus, the lengths of such programs must obey Shannon's classical entropy bound.

Inequality 3 can be applied to the case where we generate m i.i.d. requests to the set $\{1, \ldots, n\}$ of items with probability p_i for item i. Associate to each generated sequence with m requests a different binary code word x:

$$\mathbb{E}\left(K(x)\right) \geq H(X) = mH(X_{i.i.d.}) = m\sum_{i=1}^{n} -p_i \log p_i,$$

where $H(X_{i.i.d.})$, again, is the entropy of the process producing a single i.i.d. request.

A lower bound holding always is, of course, preferable to one that holds under expected value, as probability might not be concentrated around the expected value. It is possible to remove the expected value and obtain an asymptotic lower bound by limiting the set of probability distributions to the ones that are ergodic and stationary. These limitations allow us to work around the expected value, but still let the sequence generating processes to produce items that depend on each other, i.e., like *working sets* [1] do. Intuitively (strong) *stationarity* means that without knowing the other items in the sequence, each item is distributed identically. Informally, an *ergodic process* is one in which knowledge of the present state does not help to estimate where the process is going to be in distant future. For exact definitions see, e.g., [5].

Let us recall the asymptotic equipartition property for ergodic processes (see [5, p. 61, pp. 474–479]).

Theorem 1 (The Shannon-McMillan-Breiman theorem). *Assume that a finite-valued string $x = x_1 \cdots x_m$ is being generated by a stationary ergodic process and define the entropy of the process after m generated values as:*

$$H_m(X) = \sum_{l(x)=m} p_m(x) \lg \frac{1}{p_m(x)},$$

where $p_m(x)$ is the probability of having generated x. With probability one:

$$\lim_{m\to\infty} -\frac{\log p_m(x)}{m} = \lim_{m\to\infty} \frac{H_m(X)}{m},$$

if the limits exist.

We also need the following lemma [5] due to Andrew Barron.

Lemma 2. *Let $l_c(x)$ be codeword lengths associated with any code and $p(x)$ be the probability of obtaining x. Then*

$$\mathbb{P}\left(l_c(x) \leq -\log p(x) - v\right) \leq \frac{1}{2^v}.$$

Corollary 3. *For a finite-valued string x produced by a stationary ergodic process with entropy $H_m(X)$ after m generated values the following holds with probability one*

$$\lim_{m\to\infty} \frac{K(x)}{m} \geq \lim_{m\to\infty} \frac{H_m(X)}{m},$$

if the limits exist.

Proof. Lemma 2 asserts that $K(x)$ is not much less than $-\log p_m(x)$ with high probability as $K(x)$ is a codeword for x. More precisely, let $l_c(x) = K(x)$ and set $v = o(m)$, it follows that with probability $1 - 2^{-o(m)}$ it holds that

$$K(x) \geq -\log p_m(x) - o(m)$$

and otherwise $K(x) \geq O(1)$. The Shannon-McMillan-Breiman theorem shows that $-\log p_m(x)/m$ converges to $H_m(X)/m$, which proves the result, because the limits are assumed to exist. □

3 Offline Binary Search Tree Algorithm

Assume that there is an initial BST containing items $1, \ldots, n$ and a sequence s consisting of m access requests to the n items to be served. An offline BST algorithm serves the request sequence using a BST and knows the whole request sequence beforehand. Thus, the algorithm may rotate items in the tree at will while serving the sequence in order to optimize its total cost.

In this paper we consider offline algorithms that rotate the searched item to the root. It has been often noted (e.g., [16, 2]) that we lose at most a factor of two by assuming this. If the searched item is at depth d then accessing it costs d. On the other hand, we can bring the item to the root with $d-1$ rotations, then access it at cost 1, and do the reverse sequence of rotations to bring the item back to its original place. Hence, the total cost in this case is $2d-1$. The minimum cost for such an offline algorithm serving a sequence s is denoted as $\mathbf{c}(s)$.

Strictly speaking $\mathbf{c}(s)$ depends on the initial tree, but the number of rotations needed to change any n-node, $n \geq 11$, BST into another is at most $2n-6$ and for all sufficiently large n this bound is tight [6, 15]. Thus, the choice of

the original tree has negligible cost and there is an implicit additional cost of $O(n)$ in the notation $\mathbf{c}(s)$.

In order to describe a request sequence s we can do the following. Because the accessed item is always rotated to the root, it suffices to describe how the BST changes in between requests. Blum, Chawla, and Kalai [2] showed that the trees in between rotations in an optimal offline algorithm can be described in $6\mathbf{c}(s)$ bits. We provide an alternative way to code rotations between trees using only $5\mathbf{c}(s)$ bits. Because our subsequent bounds depend on the number of bits, it is important to code rotations as concisely as possible.

Lemma 4. *A request sequence s can be coded in $5\mathbf{c}(s)$ bits.*

Proof. Fix a request sequence $s = s_1, s_2, \ldots, s_m$ and an offline algorithm that serves s using BSTs $T_0, T_1, \ldots, T_m$, where T_0 is the initial BST and T_t is the BST after the request s_t.

Lucas [12] thinks of a rotation as changing one edge either from left to right or right to left and connecting it to different nodes. She argues that the rotated edges during a single request s_t form a connected subtree of T_{t-1} that includes the root and the node accessed next. Thus, it suffices to describe this subtree before and after rotations to describe the transformation from T_{t-1} to T_t and obtain the request s_t. We proceed by showing that there is a prefix-free code such that each transformation from T_{t-1} to T_t is assigned a word with length less than $5k(t)$ bits, where $k(t)$ is the number of edges in the subtree of T_{t-1} that changes.

The number of possible BSTs with u nodes is given by Catalan number $C_u = \binom{2u}{u}/(u+1)$ [10]. We describe the subtree that changes in T_{t-1} and how it changes by indexing into an ordered pair of BSTs. We use $k(t)$ bits for coding the number $k(t)$, simply by having $k(t) - 1$ ones followed by a zero. After the description of the number $k(t)$ we put the description of the pair of the subtrees taking $\lceil \log C^2_{k(t)+1} \rceil$ bits. This is an index to a pair of BSTs, each with $k(t)$ edges. As there are $C_{k(t)+1}$ such BSTs, $\lceil \log C^2_{k(t)+1} \rceil$ bits are enough to index the pair. Thus the total description length is $k(t) + \lceil 2 \log C_{k(t)+1} \rceil$ bits.

For values of $k(t)$ smaller than 8 we can check one by one that the number of bits is less than $5k(t)$. For larger values of $k(t)$ we bound the logarithm of

C_u by integrals:

$$\begin{aligned}
\log C_u &= \log\left(\binom{2u}{u} \Big/ (u+1)\right) \\
&= \sum_{i=u+1}^{2u} \log i - \sum_{i=1}^{u} \log i - \log(u+1) \\
&\leq \int_{u+1}^{2u+1} \log i \,\mathrm{d}i - \int_{1}^{u} \log i \,\mathrm{d}i - \log(u+1) \\
&\leq 2(u-1) - 1/\ln 2 + 3 + (2u+1)\log(u+1/2) - (2u+2)\log(u),
\end{aligned}$$

which is less than $2(u-1)$ for all $u \geq 9$. This can be verified by noting that the derivative of the last two terms is strictly negative for $u \geq 1$. Thus, for all $k(t)$, it holds that $k(t) + \lceil 2 \log C_{k(t)+1} \rceil \leq 5k(t)$.

Now for each t the transformation from T_{t-1} to T_t can be coded in $5k(t)$ bits. Because these codewords are prefix-free, we can catenate them to have a description for s with $5\sum_{t=1}^{m} k(t)$ bits. As the number of edges rotated is less than the number of rotations by an optimal offline algorithm and the cost of the sequence is more than that, it follows that the request sequence can be coded in $5\mathbf{c}(s)$ bits. □

4 Analysis of an Offline Binary Search Tree Algorithm

In this section, we lower bound $\mathbf{c}(s)$ in terms of the Kolmogorov complexity of the request sequence s and in terms of the entropy of the assumed request generating process. A request sequence s can be thought of as a binary string s_{str} that codes m requests with $\lceil \log n \rceil$ bits each, thus totaling a length of $m\lceil \log n \rceil$ in bits.

Theorem 5. *For all s such that $m \geq 2\log n + O(1)$, it holds that $\mathbf{c}(s) \geq C(s_{str})/6$.*

Proof. We prove the claim by describing a program that prints the request sequence and compare the length of this program to the Kolmogorov complexity of the sequence. The idea of the program is to input n, produce some

fixed BST with items $1, \ldots, n$, and use rotations to modify this tree. It was assumed that the accessed item will always be rotated to the root, so trees in between rotations suffice to describe the request sequence. This program for a universal Turing machine has the number n in a self-delimiting form taking $2 \log n + 2$ bits, description of rotations as given in previous section takes at most $5\mathbf{c}(s)$ bits, and a constant length portion that actually does the printing takes $O(1)$ bits. Thus, the total length is $5\mathbf{c}(s)+2 \log n+O(1) \leq 6\mathbf{c}(s)$ assuming that the length of the request sequence is comparable to $2 \log n + O(1)$. Note that $\mathbf{c}(s) \geq m$.

By the definition of the Kolmogorov complexity $6\mathbf{c}(s) \geq C(s_{str})$, since $C(s_{str})$ is the length of the shortest program for printing s_{str}. □

A consequence of Equation 2 and Theorem 5 is that for a fraction $1 - 1/2^c$ of possible request sequences it holds that

$$6\mathbf{c}(s) \geq C(s_{str}) \geq m \log n - c.$$

Note that although s_{str} has a length of $m\lceil \log n \rceil$ in bits, some bits are wasted if n is not a power of two, thus there is no ceiling function on the right hand side of the inequality. For example, when $c \geq 7$ the equation holds for over 99% of possible request sequences. As further results we have the following.

Corollary 6. *If $c = m$, then*

$$\mathbf{c}(s) \geq \frac{m(\log n - 1)}{6}$$

for a fraction $1 - 1/2^m$ of request sequences of length m.

Corollary 7. *By choosing $c = c' m \log n$, for some $0 < c' < 1$, we get*

$$\mathbf{c}(s) \geq \frac{(1 - c')m \log n}{6}$$

for a fraction $1 - 1/2^{c' m \log n}$ of request sequences of length m.

Theorem 8. *Assume that the request sequence s has been generated by a stochastic process with entropy $H(X) = \sum_s p(s) \log 1/p(s)$, where $p(s)$ is the probability of generating the sequence s. Then*

$$\mathbb{E}\left(\mathbf{c}(s)\right) = \Omega(H(X)).$$

Proof. If s is long enough, then $\mathbf{c}(s) \geq 2\log(m \log n) + 2 \log n + O(1)$. Thus, $6\mathbf{c}(s) \geq C(s_{str}) + 2\log(m \log n) \geq K(s_{str})$. By linearity of expectation

$$\mathbb{E}\left(6\mathbf{c}(s)\right) \geq \mathbb{E}\left(K(s_{str})\right) \geq H(X),$$

where the last inequality follows by Equation 3. Thus, the claim follows. □

Theorem 9. *Assume that the request sequence s is generated by a stationary ergodic process with entropy $H_m(X)$ after m requests. Then with probability one*

$$\lim_{m\to\infty} \frac{\mathbf{c}(s)}{m} \geq \lim_{m\to\infty} \frac{H_m(X)}{m}.$$

Proof. The proof is similar to that of Theorem 8, except that this time Corollary 3 is used. □

Note that we get Wilber's [16] result for i.i.d. processes (Equation 1) as a special case of Theorem 8. However, our coefficient is 1/6, while Wilber was able to attain 1/3.

Altogether, the bounds for costs presented here are by no means tight. This follows from the fact that Kolmogorov complexity is uncomputable [11]. If our bounds were tight, they would provide a way to calculate Kolmogorov complexity. To show in a practical way that Kolmogorov complexity cannot be achieved, consider a very simple access pattern of length m where two items alternate, e.g., 01010101010101 . . . 01. The Kolmogorov complexity of the sequence grows in $\log m$, but an optimal offline algorithm must pay cost that is linear in m.

5 Competitiveness of Balanced Binary Search Trees

Balanced BSTs have a worst-case cost of $m \log n$ for any sequence. Using Corollary 6 we can state the worst-case competitiveness of balanced BSTs when compared to an optimal offline BST algorithm. Let us first assume that a sequence s_{complex} is of high complexity, say among the fraction $1 - 1/2^m$ of the most complex sequences of length m.

Theorem 10. *Balanced search trees are constant competitive for all $s_{complex}$.*

Proof. Let the competitiveness ratio of balanced BSTs w.r.t. the optimal offline algorithm be D. Let the cost of a balanced BST for a sequence s be $\mathbf{c}_{\mathrm{BAL}}(s)$. Then, from Corollary 6 and by assuming that $n \geq 4$ we get:

$$D\,\mathbf{c}(s_{\mathrm{complex}}) \geq \frac{D\,m(\log n - 1)}{6} \geq m \log n \geq \mathbf{c}_{\mathrm{BAL}}(s_{\mathrm{complex}})\,.$$

The first and the last terms in this sequence of inequalities sandwich the middle terms by definition of competitive ratio, implying that $D \leq 12$. □

Wilber's [16] results imply that the cost of the offline algorithm is similar to the cost of a balanced tree for random accesses. Theorem 10 gives a lower bound for the relative size of a set of sequences having a high offline cost.

We can relate the entropy of a sequence generating process and the competitiveness of balanced search trees using the following corollary.

Corollary 11. *Let a request sequence of length m be generated by a stochastic process with entropy $H_m(X)$. Let $H_{\max}(m,n) = m \log n$ be the maximum entropy achievable by the process. Now, the expected competitive ratio of a balanced search tree with respect to an optimal offline algorithm is*

$$O\left(\frac{H_{\max}(m,n)}{H_m(X)}\right).$$

Proof. Theorem 8 states that $6\mathbb{E}\left(\mathbf{c}(s)\right) \geq H_m(X)$. The claim follows by similar argumentation as in Theorem 10. □

This shows how good balanced search trees are when compared to the optimal offline algorithm in terms of the simplicity of the process producing the request sequence.

6 On the Competitiveness of Splay Trees

Because splay trees are constant competitive with balanced BSTs [14], the results of the previous section hold also for splay trees. However, because of the distribution sensitive properties, splay trees can sometimes do better. For example, balanced search trees perform poorly on simple request sequences,

but we would expect splay trees to perform better, at least on some simple request sequences.

In order to prove the competitiveness of splay trees for i.i.d. processes, by Theorem 8, it is enough to prove that the expected cost of splay trees is upper bounded by the entropy of the request generating process.

Lemma 12. *The expected cost of splay trees for an i.i.d. request sequence s of length m is $O(mH(X_{i.i.d.}))$, where $H(X_{i.i.d.})$ is the entropy of the process producing one i.i.d. request.*

Proof. The Static Optimality Theorem of Sleator and Tarjan [14] shows that the cost of splay trees is $O(m \sum_i p_i \log p_i)$ (assuming that each item is accessed at least once), where p_i is the frequency of an item i. Empirical frequencies converge to probabilities of the i.i.d. process under expected value. Hence, the result follows. □

Corollary 13. *Under expected cost splay trees are constant competitive with an optimal offline BST algorithm for i.i.d. processes.*

One of the open problems left by Sleator and Tarjan [14] was the *Dynamic Finger Conjecture*. This result was later proved by Cole [4, 3].

Theorem 14 (Dynamic Finger Theorem). *The total cost of a splay tree serving a request sequence $s = s_1, \ldots, s_m$ is*

$$O\left(\sum_{i=1}^{m} \log(|s_i - s_{i-1}| + 1)\right).$$

Assume now that the sequence has been generated by a stationary Markov chain having conditional probabilities $p(i \mid j) \leq 1/\sqrt{1 + |i - j|}$. Let us calculate the expected cost of serving a sequence $s_1, \ldots, s_m$, which is generated from a Markov chain that has run long enough to be approximately in the stationary distribution, i.e., the request s_1 is generated from the stationary distribution. The entropy $H_m(X)$ for m generated requests is defined as

$$\sum_{s} -p(s_1, \ldots, s_m) \log p(s_1, \ldots, s_m).$$

Lemma 15. $\mathbb{E}\left(\mathbf{c}_{\text{SPLAY}}(s)\right) = O(H_m(X))$ *for a request sequence* $s = s_1, \ldots, s_m$ *generated by a stationary Markov chain having conditional probabilities*

$$p(i \mid j) \leq 1/\sqrt{1 + |i - j|}$$

that has converged to its stationary distribution.

Proof. By Dynamic Finger Theorem

$$\begin{aligned}
\mathbb{E}\left(\mathbf{c}_{\text{SPLAY}}(s_1, \ldots, s_m)\right) &= O\left(\mathbb{E}\left(\sum_{t=1}^{m} \log(|s_t - s_{t-1}| + 1)\right)\right) \\
&= O\left(\sum_{t=1}^{m} \mathbb{E}\left(-\log p\left(s_t \mid s_{t-1}\right)\right)\right) \\
&= O\left(\sum_{t=1}^{m} \mathbb{E}\left(-\log p\left(s_t \mid s_{t-1}, \ldots, s_1\right)\right)\right) \\
&= O\left(\sum_{t=1}^{m} H\left(X_t \mid X_{t-1}, \ldots, X_1\right)\right) \\
&= O\left(H_m\left(X\right)\right).
\end{aligned}$$

The second equality follows from our assumption on $p(i \mid j)$, the third equality comes from the definition of our Markov chain, the fourth is a definition of the conditional entropy, and the final equality follows from the chain rule of entropies. □

By Theorem 8 we get a corollary:

Corollary 16. *Under expected cost splay trees are constant competitive with an optimal offline BST algorithm for Markov processes that satisfy* $p(i \mid j) \leq 1/\sqrt{1 + |i - j|}$.

7 Discussion and Conclusions

In this paper we have shown that the cost of an optimal BST algorithm is lower bounded by the complexity of the request sequence as measured by the Kolmogorov complexity. It is theoretically satisfying to know that

sequences that are complex seem to cost more to serve than simple sequences. Furthermore, this non-tight bound is able to provide us a lower bound in terms of the entropy of the request generating process, thus generalizing the result of Wilber [16].

This entropy bound is of limited use for two reasons. First, the cost of the best offline algorithm is often actually worse than the entropy of the process and, thus, the cost of an online algorithm cannot be upper bounded by the entropy. Second, it can be extremely difficult to analyze the competitiveness of an online algorithm in terms of the entropy of the request generating process. Nevertheless, as demonstrated above, there are useful results that can be obtained by these techniques.

One motivation for this work was to show that it is possible to construct an online algorithm which is dynamically competitive by predicting next access, in the spirit of [2]. We found that such an algorithm is possible if the complexity of the sequence is very low, i.e., does not grow as a function of the length of sequence, but the general case is left as an open problem.

Acknowledgments

This work has been in part supported by the Academy of Finland.

References

[1] S. Albers, L.M. Favrholdt, and O. Giel. On paging with locality of reference. In *Proceedings of the Thirty-Fourth annual ACM Symposium on Theory of Computing*, pages 258–267. ACM Press, 2002.

[2] A. Blum, S. Chawla, and A. Kalai. Static optimality and dynamic search-optimality in lists and trees. *Algorithmica*, 36(3):249–260, 2003.

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

[4] R. Cole, B. Mishra, J. Schmidt, and A. Siegel. On the dynamic finger conjecture for splay trees. part I: Splay sorting $\log n$-block sequences. *SIAM Journal on Computing*, 30(1):1–43, 2000.

[5] T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley & Sons, 1991.

[6] K. Culik II and D. Wood. A note on some tree similarity measures. *Information Processing Letters*, 15(1):39–42, 1982.

[7] E.D. Demaine, D. Harmon, J. Iacono, and M. Pătraşcu. Dynamic optimality — almost. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 484–490. IEEE Computer Society Press, 2004.

[8] P. Grünwald and P. Vitányi. Kolmogorov complexity and information theory, with an interpretation in terms of questions and answers. *Journal of Logic, Language, and Information*, 12(4):497–529, 2003.

[9] J. Iacono. Alternatives to splay trees with $O(\log n)$ worst-case access times. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 516–522. ACM/SIAM, 2001.

[10] D.E. Knuth. *The Art of Computer Programming, 2nd Ed.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1978.

[11] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Verlag, second edition, 1997.

[12] J.M. Lucas. Canonical forms for competitive binary search tree algorithms. Technical Report DCS-TR-250, Computer Science Department, Rutgers University, 1988.

[13] J.I. Munro. Competitiveness of linear search. In Mike Paterson, editor, *ESA 2000, Proceedings of the 8th Annual European Symposium*, volume 1879 of *Lecture Notes in Computer Science*, pages 338–345. Springer, 2000.

[14] D.D. Sleator and R.E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, 1985.

[15] D.D. Sleator, R.E. Tarjan, and W.P. Thurston. Rotation distance, triangulations, and hyperbolic geometry. *Journal of the American Mathematical Society*, 1(3):647–681, 1988.

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